UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ESTRUTURAS E CONSTRUÇÃO CIVIL

MARCELO EDUARDO FERNANDES DE OLIVEIRA

Desenvolvimento de algoritmo e programa de análise de estruturas em concreto armado levando em consideração as etapas construtivas e a variação das características do concreto com o tempo

São Carlos
2019

MARCELO EDUARDO FERNANDES DE OLIVEIRA

Desenvolvimento de algoritmo e programa de análise de estruturas em concreto armado levando em consideração as etapas construtivas e a variação das características do concreto com o tempo

Dissertação apresentada ao Programa de Pós-Graduação em Estruturas e Construção Civil da Universidade Federal de São Carlos como parte dos requisitos para obtenção do título de Mestre em Estruturas e Construção Civil.

Área de Concentração: Sistemas Construtivos

Orientação: Prof. Dr. Roberto Chust Carvalho

São Carlos
2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Estruturas e Construção Civil

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Marcelo Eduardo Fernandes de Oliveira, realizada em 30/04/2019:

Prof. Dr. Roberto Chust Carvalho
UFSCar

Prof. Dr. Marcelo de Araujo Ferreira
UFSCar

p/

Profa. Dra. Maria Cristina Vidigal de Lima
UFU

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) Maria Cristina Vidigal de Lima e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

Prof. Dr. Roberto Chust Carvalho

Dedico este trabalho a meus pais, Luiz e Sueli, que sem seus incentivos eu não seria capaz de tamanho esforço; aos meus irmãos, Adriana e Rafael, que servem de inspiração para mim.

AGRADECIMENTO

Mencionar todas as pessoas que possibilitaram este trabalho é uma tarefa impossível, pois são tantas as pessoas responsáveis por quem sou. Mesmo assim deixo meus mais sinceros agradecimentos:

Ao Prof. Dr. Roberto Chust Carvalho, pela orientação e genialidade, que com conhecimento e amizade, me ajudou a elaborar este trabalho;

Ao Prof. Dr. Marcelo de Araújo Ferreira, pelas incontáveis ajudas durante a minha pesquisa, e pelas sugestões para melhoria deste trabalho;

À Prof. Dra. Maria Cristina Vidigal de Lima, cujo conhecimento é de valor inestimável, e pelas sugestões feitas;

Ao corpo docente da Universidade Federal de São Carlos, que pela insistência e dedicação, passa seu valioso conhecimento para frente;

À minha amada mãe, Prof. Dr. Sueli Machado Pereira de Oliveira, cujo apoio pela busca do conhecimento me tornou quem eu sou. Obrigado;

Ao meu querido pai, Luiz Augusto de Oliveira, quem me ensinou a ser curioso e nunca conformista;

À minha querida avó, Evany Machado Pereira, menina no coração e que sempre esteve ao meu lado;

À minha fantástica irmã, Me. Adriana Fernandes Machado de Oliveira, pela sua ajuda na revisão deste trabalho, pela sua amizade, companheirismo, conselho, brigas e inspiração.

Ao meu incansável irmão, Rafael Fernandes de Oliveira, pela inspiração, amizade, sede de conhecimento e dedicação, e por provar que tudo é possível;

Ao meu cunhado, Alex Benito Alves, pela amizade, que durante o processo de desenvolvimento deste mestrado me ajudou em incontáveis situações;

À minha companheira, Josiane Luzia Dias, pela amizade, compreensão, carinho, conselhos, por me fazer querer todo dia ser uma pessoa melhor.

Aos meus queridos amigos Giovanni Francisco Molina-Aguirre, Leonardo Mendonça Figueiredo, Amanda Pioli Ribeiro e Gabriel Eller Gusmão, pelas amizades sinceras.

Obrigado a todos, citados e não citados, que de algum modo contribuiu para a minha elaboração deste trabalho.

RESUMO

A análise estrutural de edifícios de concreto armado moldado no local é feita, usualmente, considerando a estrutura em sua situação final, com o carregamento aplicado em uma determinada data e o concreto com seus valores característicos. Para examinar uma análise estrutural mais próxima da realidade, torna-se necessário considerar o sistema evolutivo da construção da edificação, as datas em que as diversas ações começam a atuar, os valores reais da resistência, do módulo de elasticidade e das características geométricas da estrutura até o momento. Este trabalho procura fazer um levantamento minucioso para a criação de um procedimento que considere todos os aspectos citados, incluindo as etapas provisórias de escoramento, sem usar técnicas simplificadas de estruturas equivalentes ou similares encontradas atualmente na literatura. Determinados programas de análise estrutural podem considerar algumas das variáveis enumeradas, mas devem ser configurados através de um processo trabalhoso e repetitivo, e acabam tornando a análise lenta, em geral incompleta e sujeita a erros pela dificuldade de entrada de dados. Com técnicas modernas de programação e uso de banco de dados, elabora-se um algoritmo que faz a análise da estrutura da edificação considerando as variáveis apresentadas, e desenvolve-se um programa de pórticos tridimensionais específico para o mesmo. Assim, o comportamento e o deslocamento da estrutura durante sua construção e utilização podem ser visualizados, possibilitando melhor escolha de dimensões de vigas, pilares e lajes.

Palavras-chave: Análise estrutural temporal. Sistema evolutivo. Algoritmo. Programa de cálculo estrutural. Pórticos tridimensionais. Grelha.

ABSTRACT

The structural analysis of reinforced concrete buildings fabricated on site is usually done considering the structure in its final state, with the load applied on a given date and the concrete with its characteristic values. To examine a structural analysis closer to reality, it is necessary to consider the evolutionary stages of the building construction, the dates in which the various actions begin to act, the actual values of the resistance, the modulus of elasticity and the current structure geometry at the time. This work aims to elaborate a detailed procedure that takes all the mentioned aspects in consideration, including the provisional stages of shoring, without using simplified techniques currently found in literature. Certain structural analysis programs may consider some of the variables listed, but they must be configured through a laborious and repetitive process, and end up making the analysis slow, often incomplete and error-prone, due to the difficulty of data entry. With modern database programming techniques, we have elaborated an algorithm that analyzes the structure of the building considering the presented variables, developing a three-dimensional frames program specially tailored to suit the algorithm presented in this work. Thus, the behavior and the displacement of the structure throughout its construction and use can be visualized, enabling a better choice for the cross sections of beams, columns and slabs.

Keywords: Temporal structural analysis. Evolutionary stages. Algorithm. Structural engineering software. Three-dimensional frames.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

$\{D\}$	Vetor deslocamento
$E_c(t)$	Modulo de elasticidade do concreto no tempo
$E_{ci}(t)$	Módulo de elasticidade do concreto no tempo (ABNT)
$f_c(t)$	Resistência do concreto no tempo
$f_{ck,j}$	Resistência característica do concreto no dia j
$[K]$	Matriz de rigidez da estrutura
$[K_{el}]$	Matriz de rigidez do elemento
$\{P\}$	Vetor reações

SUMÁRIO

1. INTRODUÇÃO

1.1. Generalidades

Os edifícios de múltiplos andares em concreto armado usualmente são construídos de modo sequencial (construção evolutiva). A partir do término da execução da fundação, concretam-se pilares que ligam um andar i (no início pode ser o térreo) ao imediatamente superior $i+1$. Depois, são concretadas as vigas e as lajes do andar $i+1$. Este procedimento é executado repetidamente ao longo da execução da estrutura. Como as estruturas de concreto armado têm a propriedade – guardados os cuidados de detalhamentos de armadura e de execução de concretagem adequados – de se obter, ao término da obra, uma estrutura como se fosse feita de uma só vez, ou seja, monolítica, torna-se razoável imaginar que só será preciso analisar a situação final – quando a estrutura estiver totalmente pronta e todos os carregamentos estiverem atuando. Assim, os escritórios de projetos e os programas de cálculo profissionais (que fazem análise e detalham armadura) realizam o dimensionamento e detalhamento da estrutura deste tipo de edificação já na sua condição final estrutural (com todos os elementos). As ações verticais permanentes são consideradas aplicadas simultaneamente, da mesma forma que as ações variáveis – embora todas sejam introduzidas de forma gradual no tempo. Há também a questão do processo construtivo, pois normalmente um piso pode receber, por exemplo, o escoramento de outros dois. Interfere em todo o procedimento construtivo a questão da resistência do concreto que varia ao longo do tempo. As especificações normativas e hipóteses de cálculo são feitas de forma que se deve considerar o valor pleno da resistência do concreto, dado aos 28 dias de idade (resistência característica do mesmo). Finalmente, soma-se a esta questão a variação do módulo de elasticidade do concreto.

Dessa maneira, resumidamente, o ideal seria:

a) Considerar a estrutura sendo construída de forma evolutiva (andar por andar), resolvendo, a cada tempo, a estrutura com o número de andares existentes e o carregamento atual que estará atuando até chegar ao final da construção;
b) Considerar, durante o processo evolutivo, a análise da atuação e retirada dos escoramentos, levando em conta os valores adequados de resistência e deformação do concreto – que, geralmente, têm idade inferior a 28 dias.

c) Considerar carregamentos incrementais, tanto para as ações permanentes como para as varáveis, que permitam avaliar de forma mais precisa o estado de deformação da estrutura.

Vários autores já se debruçaram sobre estes problemas, porém, em geral, avaliando apenas parte deles.

Prado (1999) declara que as ações presentes durante as etapas construtivas de uma edificação em concreto armado são significantemente influenciadas pelos processos construtivos, e podem até chegar a ultrapassar as capacidades resistentes definidas pelo projeto estrutural na data em que ocorrem. Fissurações prematuras em elementos estruturais podem resultar em maiores deformações ao longo do tempo, e uma deterioração mais rápida da estrutura. Além disso, as ações de construção normalmente solicitam o concreto antes que o mesmo tenha atingido as características de resistência e deformabilidade previstas – aos 28 dias, usadas normalmente no desenvolvimento do projeto estrutural.

Freitas (2004) afirma que o conhecimento das propriedades do concreto jovem, das ações ocorridas durante a construção e das deformações impostas por essas ações são relevantes para a verificação dos estados limites último e de serviço de uma estrutura.

De acordo com Putziski e Vargas (2011), reiterando a natureza visco-elástico-plástica do concreto, tem-se que o mesmo sofre deformações lentas significativas. Assim, evidencia-se a importância do estudo das deformações ocorridas ao longo do tempo em estruturas de concreto armado para se avaliar o impacto do carregamento precoce da estrutura nas deformações finais.

Ferreira (2017) expõe que o modelo de análise estrutural tradicionalmente usado (estrutura completa no primeiro dia, com os valores característicos de resistência do concreto, onde os carregamentos são aplicados ao mesmo tempo para todos os Estados de utilização) não leva em conta o processo construtivo. Portanto, pode apresentar grandes diferenças nos resultados dos esforços internos e deslocamento se comparado a um modelo que leve em conta as etapas construtivas.

Dessa forma, dois aspectos são importantes na análise de edificações de múltiplos andares – como já destacados anteriormente: considerar seu processo de construção conforme a estrutura vai evoluindo – o que chamamos aqui de procedimento “evolutivo”; e as características do concreto com relação à sua idade, quando da entrada de ação de caráter permanente. Neste último caso, costuma-se usar o procedimento de carregamento incremental. Assim, o modelo a ser usado – mais próximo da realidade da edificação – é do tipo evolutivo e com carregamento incremental.

Hoje em dia, para que os engenheiros projetistas possam levar em consideração todos esses efeitos e ações, é necessário a modelagem de dezenas – se não centenas – de estruturas e cargas diferentes. Logo se entende o porquê de não serem considerados esses efeitos e ações, principalmente em estruturas de pequeno e médio porte – ou até mesmo nas de grande porte, pela ausência de equipes que entendam a importância do estudo dessas ações e efeitos.

A grande discussão proposta neste trabalho é verificar se realmente o cálculo pode ser feito unicamente para a situação construtiva final da estrutura, com todas as ações presentes em um mesmo instante, e com o concreto já com sua resistência característica à compressão. Isto será possível comparando os esforços finais e eventualmente aqueles que ocorrem durante uma fase construtiva.

1.2. Objetivos

Este trabalho tem como objetivos: (a) desenvolver um algoritmo que leve em consideração a evolução da estrutura na execução e o efeito da idade do concreto na data de entrada das cargas na estrutura; (b) Automatizar a consideração dos efeitos das etapas construtivas e da mudança das características geométricas da estrutura; (c) desenvolver um programa capaz de incluir o algoritmo presente neste trabalho para demonstrar sua funcionalidade e, futuramente, servir como ferramenta para uso de engenheiros calculistas, estudantes e demais profissionais.

O algoritmo desenvolvido neste trabalho engloba métodos de análise estrutural já comprovadamente eficazes encontrados na literatura – com modificações no comportamento gradual da estrutura ao longo da construção, levando em consideração a evolução das características do concreto ao longo do tempo. Usamos, a princípio, o método de resolução de estruturas de barras prismáticas com análise matricial.

O método de automatização se resume ao melhor método de implementação do algoritmo, de modo que o resultado do cálculo estrutural fique pronto de maneira eficiente. Dessa forma, não é necessário que o engenheiro projetista faça centenas de pequenas modificações na estrutura, simulando a evolução da sua geometria, para obter o resultado da análise estrutural levando em conta os pontos citados no algoritmo – ou que modifique manualmente a geometria da estrutura e as características do concreto ao longo do tempo.

Por fim, o programa a ser desenvolvido deve ser capaz de demonstrar o algoritmo e a possibilidade de automatização – a fim de prover fácil visualização do comportamento

estrutural, com uma interface gráfica condizente com a proposta de simplificar o cálculo mais completo da estrutura de concreto armado.

1.3. Justificativas

Embora estudos anteriores – Prado (1999), Freitas (2004), Putziski e Vargas (2011), e Ferreira (2017) – já tenham comprovado a importância tanto da consideração das ações na estrutura durante as etapas construtivas em edificações de concreto armado, quanto da variação da resistência do concreto, nenhum destes se apresentou resultados da análise das deformações.

Como é comum no exterior (e o programa europeu CYPECAD® é exemplo), também no Brasil a maior parte dos edifícios em concreto armado é calculada sem levar em consideração tais ações e efeitos. Os principais programas de cálculo nacionais (TQS®, EBERICK®) utilizados hoje em dia pelos engenheiros projetistas falham em prover um método adequado e confiável para levar em consideração tais efeitos. Assim, fica a cargo dos projetistas fazer um imenso trabalho repetitivo de modificação estrutural e carregamentos para simular os efeitos citados – o que, na prática, acaba não acontecendo. Alguns programas de análise permitem fazer uma análise evolutiva, ainda que com restrições e dificuldades – como o SAP2000® (ver FERREIRA, 2017). Além disso, o módulo de pré-moldados do TQS® leva em conta uma certa sequência de execução da estrutura (explicitada no site e nos manuais da empresa), porém sem levar em conta a variação das propriedades do concreto.

Surge então a necessidade do desenvolvimento de um método de cálculo adequado para consideração de tais ações e efeitos, que possua as seguintes características: prático de ser adotado, adaptável para diferentes formas estruturais, confiável e rápido. Este método poderá ser empregado por meio da criação de um programa que consiga automatizar o processo necessário para levar em consideração todos estes efeitos – parcialmente ou de modo integral. É possível, a partir das informações obtidas pelo programa, realizar cálculos mais adequados, garantir uma execução segura e prever de forma mais precisa o estado de formação da estrutura. Ao realizar um cálculo mais próximo do que realmente acontece na prática, isso garanta uma maior longevidade estrutural, com chances reduzidas de aparecimento de esforços solicitantes não considerados previamente.

1.4. Métodos

Para o desenvolvimento do melhor método genérico para automatizar o cálculo de estruturas – levando em consideração as etapas construtivas e a variação das características do concreto –, serão utilizados fluxogramas computacionais com auxílio de sistemas de repetição de ações. Assim, o programa a ser desenvolvido poderá processar talvez centenas de simulações da estrutura ao longo do tempo, determinando, portanto, uma envoltória de esforços máximo e mínimo presentes em cada um dos elementos componentes da estrutura.

Essa pesquisa se dividiu nas seguintes etapas: revisão bibliográfica sobre o assunto; desenvolvimento do algoritmo e do programa; testes; e validação.

O conhecimento necessário para a elaboração do cálculo e do programa foi adquirido por meio de pesquisa bibliográfica em livros, revistas especializadas, artigos técnicos, congressos e publicações da área, nos quais buscamos métodos de resolução otimizados para análise computacional dos problemas. Foram adotados os métodos mais atuais – brasileiros e internacionais – que contribuíram para o melhor resultado dos cálculos computacionais, de modo a torná-los confiáveis e verificáveis.

Modernas linguagens de programação foram utilizadas, como C# para o desenvolvimento básico do programa, e XAML para o desenvolvimento da interface do mesmo. Diversos Ambiente Integral de Desenvolvimento são utilizados, como Visual Studio 2017 e Unity 2018.

O desenvolvimento do algoritmo envolveu pesquisas aprofundadas a respeito do funcionamento e da implementação de sistemas modernos de cálculo, compatíveis com a implementação das variáveis consideradas neste trabalho.

No desenvolvimento do programa, foi focado em como melhor elaborar uma interface voltada para a engenharia, com facilidade e rapidez de modificação e de adaptação do programa para alterações no sistema de cálculo e/ou nas normas vigentes.

Os testes e a validação do algoritmo e do programa foram realizados com base em exemplos já existentes, confirmando-se seus cálculos e validando-se o método empregado.

1.5. Resumo do trabalho

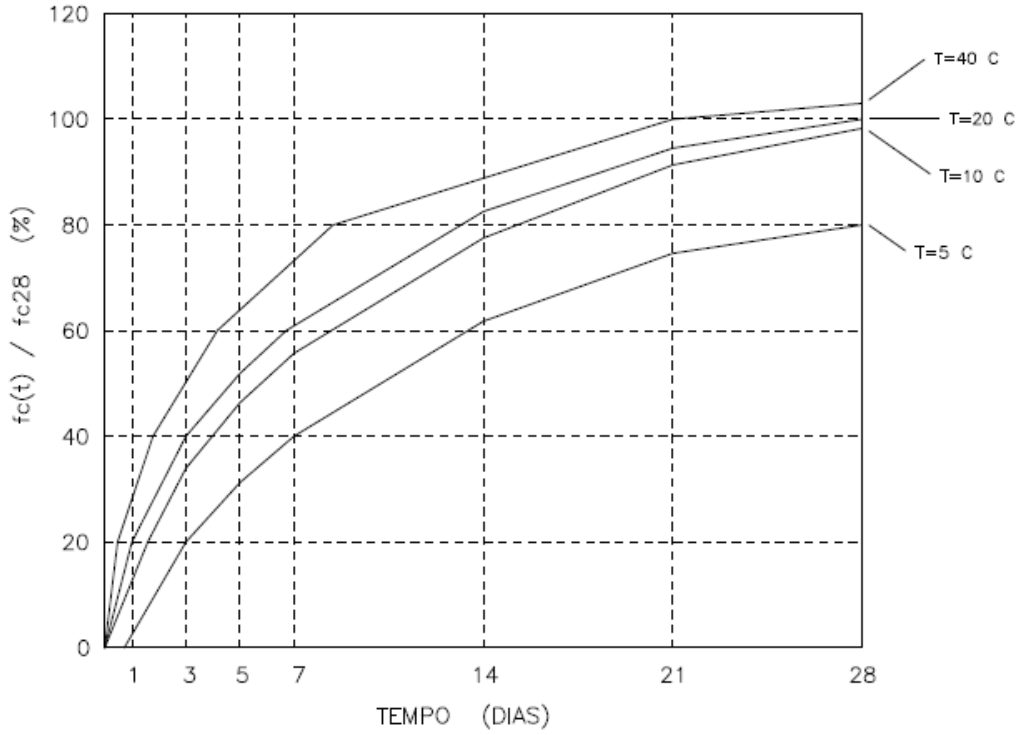
No capítulo inicial deste trabalho, apresentamos a introdução ao tema, os objetivos e a metodologia empregada. O segundo capítulo discute a variação, ao longo do tempo, do módulo de elasticidade e resistência do concreto. Em um terceiro momento, apresentamos de forma

resumida o procedimento de análise matricial de estruturas. Tratamos, no quarto capítulo, do desenvolvimento do processo construtivo de uma edificação de múltiplos andares, identificando as variáveis a considerar no algoritmo. O quinto capítulo versa sobre o desenvolvimento do algoritmo, enquanto que o sexto capítulo demonstra o desenvolvimento do programa. Por fim, os capítulos 7 e 8 abrangem a discussão dos resultados e considerações finais.

2. MÓDULO DE ELASTICIDADE E RESISTÊNCIA DO CONCRETO

Já nos anos 1950, sabia-se da grande variação no valor do módulo de elasticidade do concreto, principalmente em idade jovem, como podemos perceber em estudo feito por Price (1951). Nota-se a pequena variação na resistência do concreto quando a cura se dá entre 10 a 40 graus Celsius, com a resistência variando drasticamente durante o período de 28 dias.

FIGURA 2.1 - Desenvolvimento da resistência à compressão do concreto para diferentes temperaturas



Fonte: PRICE (1951)

Diversos métodos de cálculo do valor da resistência do concreto ao longo do tempo são sugeridos pela literatura:

(a) Gardner e Zhao (1993):

$$f_c(t) = \left(\frac{t^{0,75}}{2,8 + 0,77 \cdot t^{0,75}} \right) \cdot f_{c_{28}} \tag{2.1}$$

$$E_c(t) = 3500 + 4300 \cdot \sqrt[2]{f_c(t)} \tag{2.2}$$

(b) ACI 318 (1989):

$$f_c(t) = \left(\frac{t}{4 + 0,85 \cdot t} \right) \cdot f_{c_{28}} \tag{2.3}$$

$$E_c(t) = 4730 \cdot \sqrt[2]{f_c(t)} \tag{2.4}$$

(c) ABNT NBR 6118/2014:

Para concretos com data inferior a 28 dias:

$$f_{ck,j} = \beta_1 \cdot f_{ck} \tag{2.5}$$

onde

$$\beta_1 = e^{\left\{s\cdot\left[1-\left(\frac{28}{t}\right)^{\frac{1}{2}}\right]\right\}} \tag{2.6}$$

sendo

$s = 0,38$ para concreto de cimento CPIII e IV;

$s = 0,25$ para concreto de cimento CPI e II;

$s = 0,20$ para concreto de cimento CPV-ARI.

Para os concretos com f_{ck} de $20MPa$ a $45MPa$:

$$E_{ci} = \alpha_E \cdot 5600 \cdot \sqrt[2]{f_{ck}} \tag{2.7}$$

Para os concretos com f_{ck} de $50MPa$ a $90MPa$:

$$E_{ci} = 21,5 \cdot 10^3 \cdot \alpha_E \cdot \left(\frac{f_{ck}}{10} + 1,25\right)^{\frac{1}{3}} \tag{2.8}$$

Para concretos de classe C20 a C45:

$$E_{ci}(t) = \left[\frac{f_{ckj}}{f_{ck}}\right]^{0,5} \cdot E_{ci} \tag{2.9}$$

Para concretos de classe C50 a C90:

$$E_{ci}(t) = \left[\frac{f_{ckj}}{f_{ck}}\right]^{0,3} \cdot E_{ci} \tag{2.10}$$

sendo

$\alpha_E = 1,2$ para basalto e diabásio;

$\alpha_E = 1,0$ para granito e gnaisse;

$\alpha_E = 0,9$ para calcário;

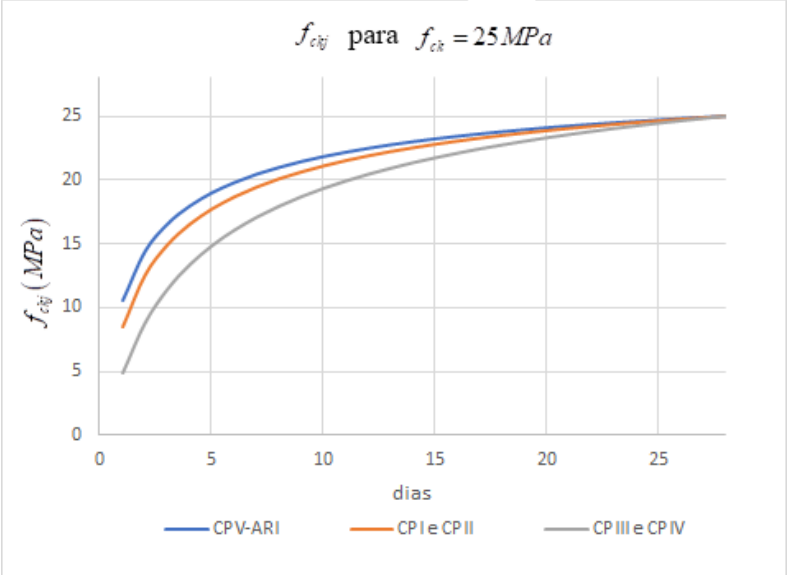
$\alpha_E = 0,7$ para arenito.

A escolha da equação mais apropriada para o cálculo das propriedades do concreto no programa desenvolvido foi pela norma ABNT, considerando os coeficientes $\alpha_E = 1,0$ e $s = 0,25$.

A importância destes valores está não só no cálculo da deformação da estrutura, como na distribuição dos esforços, principalmente quando se analisa as etapas de escoramento. Vale lembrar que as escoras (metálicas e de madeira), em geral, têm valor de módulo de elasticidade constante – e podem, assim, serem consideradas tanto no algoritmo quanto no programa desenvolvido.

Segundo Ferreira (2017), existem vários fatores que influenciam na variação das propriedades do concreto. Entre eles, como mostra a FIGURA 2.2, a idade do concreto jovem é o que mais tem influência. O ganho de resistência com a idade está relacionado ao grau de hidratação dos compostos do cimento. Cimentos como o CP V ARI apresentam maior resistência em idades menores, se comparados aos cimentos com adições minerais (CP III AF e CP IV).

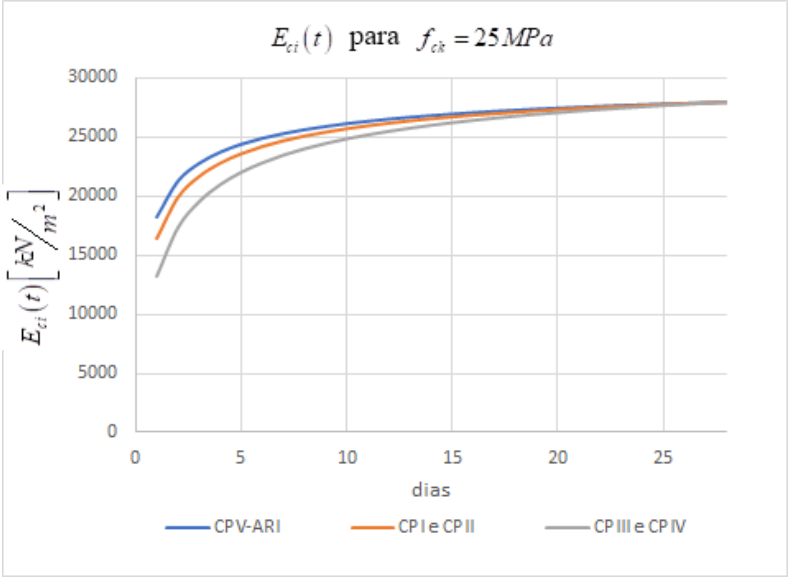
FIGURA 2.2 - Evolução da resistência do concreto à compressão com $f_{ck} = 25MPa$



Fonte: Próprio autor (2019)

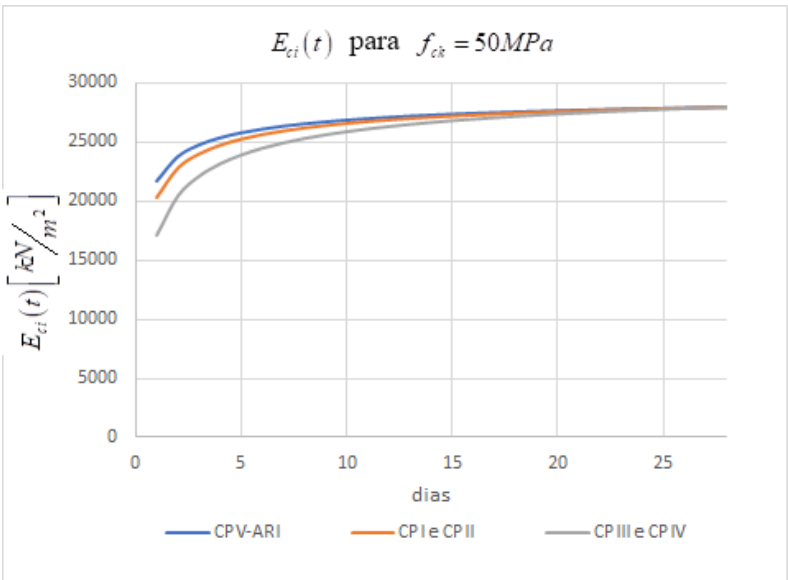
A evolução do módulo de elasticidade do concreto, para as diferentes situações que a ABNT NBR 6118/2014 cobre, pode ser visualizada na FIGURA 2.3, para concretos com resistência característica aos 28 dias de $25MPa$, e na FIGURA 2.4, para concretos com resistência característica aos 28 dias de $50MPa$. Consideramos estes exemplos pontuais para ilustrar o comportamento semelhante da variação do módulo de elasticidade para as equações (2.9) e (2.10), respectivamente.

FIGURA 2.3 - Evolução do módulo de elasticidade do concreto em kN/m^2 para concretos com as características ($f_{ck} = 25MPa$, agregado granito) dos 0 aos 28 dias.



Fonte: Próprio autor (2019)

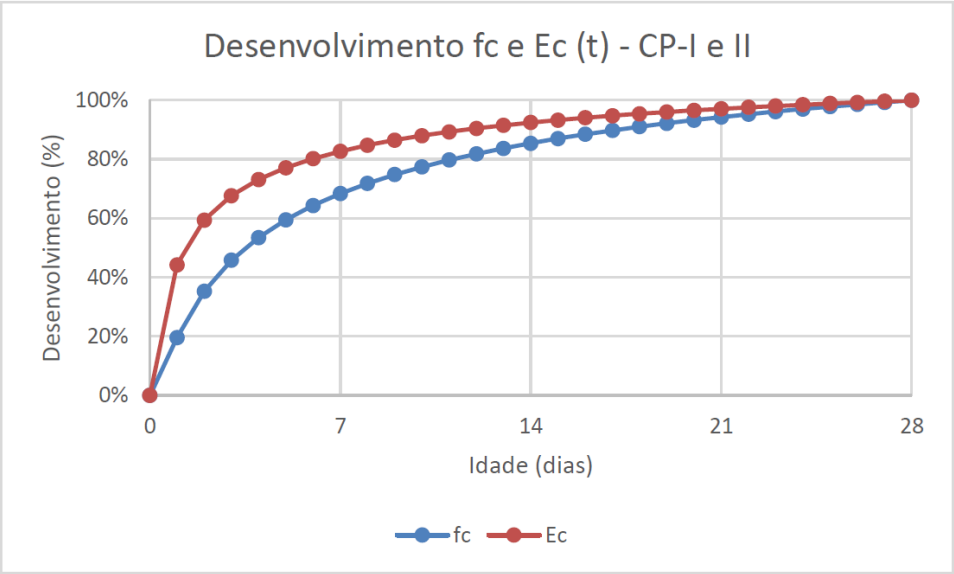
FIGURA 2.4 - Evolução do módulo de elasticidade do concreto (E) em kN/m^2 para concretos com as características ($f_{ck} = 50MPa$, agregado granito) dos 0 aos 28 dias.



Fonte: Próprio autor (2019)

Segundo Ferreira (2017), tanto o módulo de elasticidade quanto a resistência aumentam com o tempo – porém em proporções diferentes, sendo que a resistência característica do concreto a j dias aumenta proporcionalmente mais devagar que o seu módulo de elasticidade, conforme a FIGURA 2.5

FIGURA 2.5 - Evolução da resistência característica do concreto e do módulo de elasticidade com o tempo, de 0 a 28 dias.

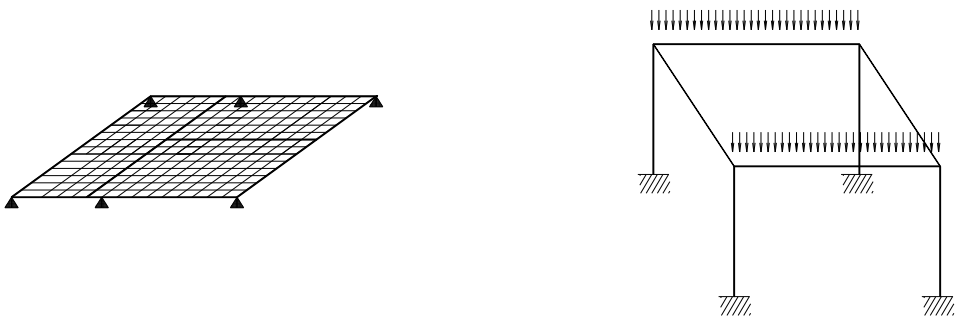


Fonte: FERREIRA (2017)

3. ANÁLISE MATRICIAL DE ESTRUTURAS

Considera-se que fazer a análise de uma estrutura é – conhecida a geometria, vínculos e ações atuantes – determinar os esforços solicitantes, reações de vínculos e deslocamentos dos diversos pontos. O uso da análise matricial é o método ideal para se usar e programar as atividades necessárias para alcançar este objetivo. Além disso, o uso do método dos deslocamentos torna o procedimento geral (e que é feito de forma única) condição importante para se usar em programação. Foi utilizado neste trabalho estruturas constituídas por barras prismáticas, como a mostrada na FIGURA 3.1.

FIGURA 3.1 - Estruturas típicas de barras prismáticas



grelha | pórtico tridimensional

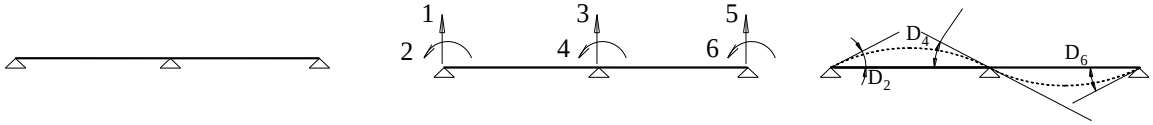
Fonte: CARVALHO (2018)

As propriedades de rigidez do material são computadas com as ações presentes, resultando nos deslocamentos nodais da estrutura, forças internas e reações de apoio. Neste estudo, partimos do princípio de que o leitor esteja familiarizado com os conceitos de nós, barras, sistema local e global de coordenadas e, finalmente, coeficientes de rigidez.

3.1. Princípio básico do processo dos deslocamentos com análise matricial

Conforme dito anteriormente, parte da análise de uma estrutura é obter os esforços solicitantes e os deslocamentos dos diversos pontos da mesma a partir da geometria, das características geométricas e elásticas, e das ações atuantes. Por exemplo, para resolver uma estrutura igual à indicada na FIGURA 3.2, do tipo viga, pode-se usar o sistema de coordenadas globais indicado. As incógnitas, quando se opta pelo processo dos deslocamentos, passam a ser os deslocamentos nodais D_1, D_2, D_3, D_4, D_5 e D_6. Assim, para a estrutura em questão, as incógnitas passam a ser D_2, D_4 e D_6, pois os demais são nulos.

FIGURA 3.2 - Estrutura do tipo viga, com coordenadas locais e deslocamentos desconhecidos



Fonte: CARVALHO (2018)

De qualquer maneira, dada uma estrutura de viga, e conhecidos seus coeficientes de rigidez (matriz de rigidez da estrutura) e os valores das ações nodais externa, pode-se determinar os deslocamentos nodais D_1, D_2, $\cdots$, D_n e os esforços solicitantes (esforços internos). Inicia-se o raciocínio imaginando a estrutura resolvida e, portanto, todos os deslocamentos D conhecidos. Assim, se ocorre D_1 (um deslocamento na direção 1), surge nesta direção 1 um esforço igual a $K_{11} \cdot D_1$. Mas o estado de deformação da estrutura pode considerar a existência de D_2, que deve provocar na direção 1 um esforço igual a $K_{12} \cdot D_2$ e assim sucessivamente. Considerando a superposição de efeitos e existindo todos os deslocamentos na direção 1, o esforço final é dado por:

$$\left(K_{11} \cdot D_1\right)+\left(K_{12} \cdot D_2\right)+\left(K_{13} \cdot D_3\right)+\left(\cdots\right)+\left(K_{1n} \cdot D_n\right)=P_1$$

Para a direção 2, a expressão seria análoga:

$$\left(K_{21} \cdot D_1\right)+\left(K_{22} \cdot D_2\right)+\left(K_{23} \cdot D_3\right)+\left(\cdots\right)+\left(K_{1n} \cdot D_n\right)=P_2$$

Finalmente, resumimos de forma geral:

$$\left(K_{i1} \cdot D_1\right)+\left(\cdots\right)+\left(K_{ij} \cdot D_j\right)+\left(\cdots\right)+\left(K_{in} \cdot D_n\right)=P_i$$

onde

K_{ij} são os coeficientes de rigidez relativo nas direções i e j;

D_j são os deslocamentos nodais na direção j;

P_i é a ação nodal (por equilíbrio) na direção i.

Colocando sob forma matricial, ficamos com

$$\begin{bmatrix} K_{11} \cdots K_{1n} \\ \vdots \ \ddots \ \vdots \\ K_{n1} \cdots K_{nn} \end{bmatrix} \cdot \begin{bmatrix} D_1 \\ \vdots \\ D_n \end{bmatrix} = \begin{bmatrix} P_1 \\ \vdots \\ P_n \end{bmatrix}$$

ou, de maneira mais condensada,

$$\left[K\right] \cdot \left\{D\right\} = \left\{P\right\} \tag{3.1}$$

que pode ser lido como: “A matriz de ações atuantes nodais (externas) é igual ao produto da matriz de rigidez da estrutura pelos deslocamentos nodais”.

Resolver a estrutura (ou fazer sua análise) significa conhecer o estado de deslocamento dos pontos nodais. Assim, se resolve o conjunto de equações lineares representado por (3.1) fazendo:

$$\{D\} = [K]^{-1} \cdot \{P\} \tag{3.2}$$

Com $[K]^{-1}$ a matriz inversa de $[K]$ (o produto de ambas resulta na matriz identidade). Ao transformar os deslocamentos nodais globais correspondentes aos locais correspondentes, por exemplo, da primeira barra – e chamando-os aqui de u_1, u_2, u_3 e u_4 –, resulta-se nos esforços solicitantes nas extremidades das barras, a partir de:

$$\{p\} = [K_{el}] \cdot \{u\} \tag{3.3}$$

com

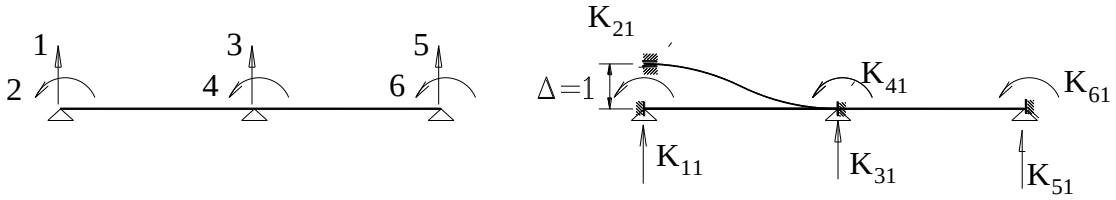
$\{p\}$ vetor dos esforços solicitantes nos nós do elemento;

$[K_{el}]$ matriz de rigidez do elemento;

$\{u\}$ vetor deslocamentos nodais do elemento (nas coordenadas locais).

O conceito de rigidez, a princípio, é aplicado a uma estrutura do tipo viga como a da FIGURA 3.3. Com o sistema de coordenadas globais indicado, pode-se obter os coeficientes de rigidez devido a um deslocamento unitário em 1 e indicados, desde que se façam os demais deslocamentos nulos. Assim, ao se aplicar uma deformação $\Delta_1 = 1$ na direção 1 da viga da FIGURA 3.3, mantendo-se os demais deslocamentos nulos, surgem os esforços K_{11}, K_{12}, K_{13}, K_{14}, K_{15} e K_{16}. Estes coeficientes $(K_{11},\cdots,K_{16})$ podem ser arranjados como uma coluna de uma matriz (coluna 1), como mostrado na FIGURA 3.4.

FIGURA 3.3 - Estrutura de viga contínua como sistema de coordenadas globais e o deslocamento unitário em 1 (com os demais nulos) e os esforços resultantes (coeficientes de rigidez)



Fonte: CARVALHO (2018)

Os demais coeficientes de rigidez possíveis da estrutura de viga podem ser determinados de forma similar, impondo-se deslocamentos unitários em cada direção e obtendo-se a matriz de RIGIDEZ DA ESTRUTURA indicada na FIGURA 3.4.

FIGURA 3.4 - Coeficientes de rigidez da estrutura de viga contínua (**FIGURA 3.3**) armazenados em forma matricial – Matriz de rigidez da estrutura

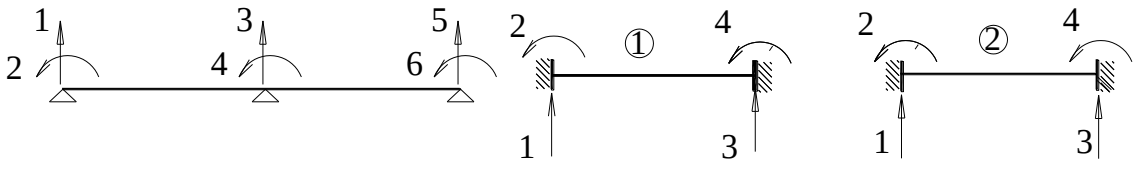
$$K_{el} = \begin{bmatrix} K_{11} & K_{21} & K_{31} & K_{41} & K_{51} & K_{61} \\ K_{12} & K_{22} & K_{32} & K_{42} & K_{52} & K_{62} \\ K_{13} & K_{23} & K_{33} & K_{43} & K_{53} & K_{63} \\ K_{14} & K_{24} & K_{34} & K_{44} & K_{54} & K_{64} \\ K_{15} & K_{25} & K_{35} & K_{45} & K_{55} & K_{65} \\ K_{16} & K_{26} & K_{36} & K_{46} & K_{56} & K_{66} \end{bmatrix}$$

Fonte: CARVALHO (2018)

3.2. Resolução de uma estrutura de barras prismáticas a partir dos elementos básicos

Há uma maneira mais simples de gerar a matriz de rigidez da estrutura da viga contínua da FIGURA 3.3. Basta considerar a estrutura composta de duas barras, calcular a matriz de rigidez de cada barra e somar os coeficientes de cada uma adequadamente para formar a matriz de rigidez da estrutura a partir da matriz de rigidez de cada elemento. Assim, consideremos que a viga da FIGURA 3.5 pode ser decomposta em duas barras. Mostramos ainda, na FIGURA 3.5, as coordenadas globais e as locais de uma barra (e da outra).

FIGURA 3.5 - Estrutura de viga contínua com um sistema de coordenadas globais; as barras que compõem a estrutura (já com os deslocamentos impedidos); e o sistema de coordenadas locais

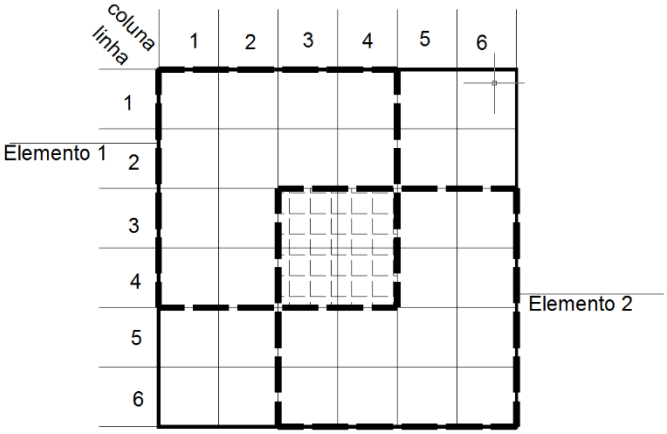


Fonte: CARVALHO (2018)

A matriz de rigidez da estrutura pode ser montada através da matriz do elemento 1 e da matriz do elemento 2 – como esquematizamos na FIGURA 3.6, indicando, através de quadrados, as posições dos elementos na matriz. A matriz de rigidez de um elemento é aquela obtida a partir das coordenadas locais.

O retângulo à esquerda do esquema da matriz indica a região em que os coeficientes do elemento 1 estarão influenciando na matriz da estrutura. Similarmente, o retângulo da direita representa a região onde os coeficientes do elemento 2 estarão influenciando na matriz de rigidez da estrutura.

FIGURA 3.6 - Esquema para mostrar a montagem de matriz de rigidez da estrutura a partir da matriz de rigidez de cada elemento



Fonte: CARVALHO (2018)

Assim, para a matriz de rigidez da estrutura, tem-se:

$$K_{33} = K_{33,1} + K_{11,2}$$

onde:

K_{33} é o coeficiente de rigidez da estrutura na direção 3 (coordenada global) devido a um deslocamento na direção 3;

$K_{33,1}$ é o coeficiente de rigidez do elemento 1 na direção 3 (coordenada local) devido a um deslocamento na direção 3;

$K_{11,2}$ é o coeficiente de rigidez do elemento 2 na direção 1 (coordenada local) devido a um deslocamento na direção 1.

É importante perceber que, apenas para esta situação, foi relativamente simples montar a matriz de rigidez da estrutura a partir da matriz de rigidez do elemento. Veremos, posteriormente, que é preciso usar uma matriz de transformação ou de rotação para efetuar a montagem da matriz da estrutura em função da matriz do elemento.

Considerando que a viga da FIGURA 3.5 tenha vão de valor L, inércia à flexão I, a matriz de rigidez fica com o aspecto apresentado na FIGURA 3.7.

FIGURA 3.7 - Matriz de rigidez da estrutura da **FIGURA 3.5**

$$K = \begin{bmatrix} \frac{12 \cdot E \cdot I}{L^3} & \frac{6 \cdot E \cdot I}{L^2} & \frac{-12 \cdot E \cdot I}{L^3} & \frac{6 \cdot E \cdot I}{L^2} & 0 & 0 \\ \frac{6 \cdot E \cdot I}{L^2} & \frac{4 \cdot E \cdot I}{L} & \frac{-6 \cdot E \cdot I}{L^2} & \frac{2 \cdot E \cdot I}{L} & 0 & 0 \\ \frac{12 \cdot E \cdot I}{L^3} & \frac{-6 \cdot E \cdot I}{L^2} & \frac{24 \cdot E \cdot I}{L^3} & 0 & \frac{-12 \cdot E \cdot I}{L^3} & \frac{6 \cdot E \cdot I}{L^2} \\ \frac{6 \cdot E \cdot I}{L^2} & \frac{2 \cdot E \cdot I}{L} & 0 & \frac{8 \cdot E \cdot I}{L} & \frac{-6 \cdot E \cdot I}{L^2} & \frac{2 \cdot E \cdot I}{L} \\ 0 & 0 & \frac{-12 \cdot E \cdot I}{L^3} & \frac{-6 \cdot E \cdot I}{L^2} & \frac{12 \cdot E \cdot I}{L^3} & \frac{-6 \cdot E \cdot I}{L^2} \\ 0 & 0 & \frac{6 \cdot E \cdot I}{L^2} & \frac{2 \cdot E \cdot I}{L} & \frac{-6 \cdot E \cdot I}{L^2} & \frac{4 \cdot E \cdot I}{L} \end{bmatrix}$$

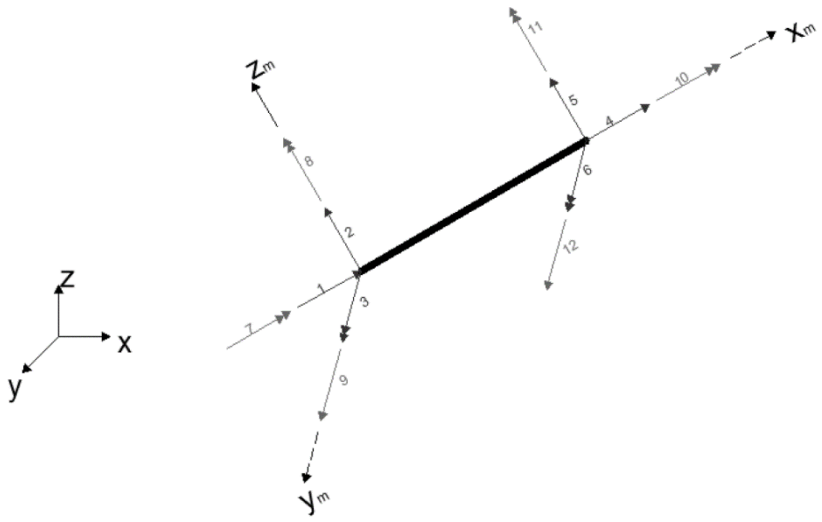
Fonte: CARVALHO (2018)

Esta maneira de montar a matriz de rigidez da estrutura a partir do elemento básico, que é sempre o mesmo, faz com que o procedimento seja geral.

3.3. Resolução de uma estrutura tridimensional de barras prismáticas

Para análises mais complexas – como pórticos tridimensionais, combinações de grelhas e pórticos, e estruturas em geral –, os elementos básicos de barra contêm, em cada nó, seis graus de liberdade, conforme FIGURA 3.8. Consequentemente, a matriz de rigidez de cada elemento aumenta proporcionalmente ao graus de liberdade, como demonstrado na FIGURA 3.9.

FIGURA 3.8 - Graus de liberdade de um elemento de barra tridimensional



Fonte: FERREIRA (2017)

FIGURA 3.9 - Matriz de rigidez de um elemento de pórtico tridimensional

$$[K_{el}]_{12\times12} = \begin{bmatrix}
\frac{E\cdot A}{L} & 0 & 0 & 0 & 0 & 0 & \frac{-E\cdot A}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{12\cdot E\cdot I_z}{L^3} & 0 & 0 & 0 & \frac{6\cdot E\cdot I_z}{L^2} & 0 & \frac{-12\cdot E\cdot I_z}{L^3} & 0 & 0 & 0 & \frac{6\cdot E\cdot I_z}{L^2} \\
0 & 0 & \frac{12\cdot E\cdot I_y}{L^3} & 0 & \frac{-6\cdot E\cdot I_y}{L^2} & 0 & 0 & 0 & \frac{-12\cdot E\cdot I_y}{L^3} & 0 & \frac{-6\cdot E\cdot I_y}{L^2} & 0 \\
0 & 0 & 0 & \frac{G\cdot J}{L} & 0 & 0 & 0 & 0 & 0 & \frac{-G\cdot J}{L} & 0 & 0 \\
0 & 0 & \frac{-6\cdot E\cdot I_y}{L^2} & 0 & \frac{4\cdot E\cdot I_y}{L} & 0 & 0 & 0 & \frac{6\cdot E\cdot I_y}{L^2} & 0 & \frac{2\cdot E\cdot I_y}{L} & 0 \\
0 & \frac{6\cdot E\cdot I_z}{L^2} & 0 & 0 & 0 & \frac{4\cdot E\cdot I_z}{L} & 0 & \frac{-6\cdot E\cdot I_z}{L^2} & 0 & 0 & 0 & \frac{2\cdot E\cdot I_z}{L} \\
\frac{-E\cdot A}{L} & 0 & 0 & 0 & 0 & 0 & \frac{E\cdot A}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{-12\cdot E\cdot I_z}{L^3} & 0 & 0 & 0 & \frac{-6\cdot E\cdot I_z}{L^2} & 0 & \frac{12\cdot E\cdot I_z}{L^3} & 0 & 0 & 0 & \frac{-6\cdot E\cdot I_z}{L^2} \\
0 & 0 & \frac{-12\cdot E\cdot I_y}{L^3} & 0 & \frac{6\cdot E\cdot I_y}{L^2} & 0 & 0 & 0 & \frac{12\cdot E\cdot I_y}{L^3} & 0 & \frac{6\cdot E\cdot I_y}{L^2} & 0 \\
0 & 0 & 0 & \frac{-G\cdot J}{L} & 0 & 0 & 0 & 0 & 0 & \frac{G\cdot J}{L} & 0 & 0 \\
0 & 0 & \frac{-6\cdot E\cdot I_y}{L^2} & 0 & \frac{2\cdot E\cdot I_y}{L} & 0 & 0 & 0 & \frac{6\cdot E\cdot I_y}{L^2} & 0 & \frac{4\cdot E\cdot I_y}{L} & 0 \\
0 & \frac{6\cdot E\cdot I_z}{L^2} & 0 & 0 & 0 & \frac{2\cdot E\cdot I_z}{L} & 0 & \frac{-6\cdot E\cdot I_z}{L^2} & 0 & 0 & 0 & \frac{4\cdot E\cdot I_z}{L}
\end{bmatrix}$$

Fonte: Próprio autor (2019)

A utilização de modelagem tridimensional possibilita a determinação dos esforços normais, cortantes, fletores e torsores de todos os elementos da estrutura, bem como a determinação dos deslocamentos e rotações em cada um dos seis graus de liberdade de cada um dos nós que a compõem. Dessa forma, pode ser empregada para avaliar carregamentos verticais e horizontais.

Ferreira (2017) explica que a evolução da análise matricial de estruturas em softwares comerciais tornou acessível a simulação de modelos mais complexos. Os dois maiores softwares de análise computacional, o EBERICK® e o TQS®, divergem nos resultados encontrados entre os mesmos modelos – principalmente nos esforços axiais dos pilares, devido a mudanças na distribuição das cargas nos pavimentos. O primeiro considera os pavimentos isolados e adiciona as cargas dos mesmos aos pilares como se estes fossem apoios indeslocáveis, e o segundo usa modelagem dos pavimentos por grelha, apresentando deslocamentos diferenciais entre os pilares.

4. DESENVOLVIMENTO DO PROCESSO CONSTRUTIVO DE UMA EDIFICAÇÃO DE MÚLTIPLOS ANDARES – VARIÁVEIS A CONSIDERAR NO ALGORITMO

Este capítulo apresenta, de forma resumida, a descrição de como se efetua a execução de uma obra. Isto definirá quais variáveis serão elegidas e consideradas as principais no algoritmo de resolução da estrutura.

4.1. Elaboração

Como premissa inicial, a execução se dará com as atividades de: concretagem de pilares, escoramento, execução de vigas e lajes, retirada de escoramento, introdução de sobrecargas (paredes e revestimento), introdução de ações variáveis (carga acidental).

Para ilustrar o descrito anteriormente, o QUADRO 4.1 indica a repetição das etapas construtivas estruturais de um período de uma obra.

QUADRO 4.1 – Atividades de execução da estrutura em 4 andares

dia	andar	elementos	atividade
0	0-1	pilar	concretagem
2	0-1	escoramento	montagem
7	1	vigas e lajes	concretagem
9	1-2	pilar	concretagem
11	1-2	escoramento	montagem
14	2	vigas e lajes	concretagem
16	2-3	pilar	concretagem
18	2-3	escoramento	concretagem
21	3	vigas e lajes	concretagem
22	0-1		retirada de escoramento
23	3-4	pilar	concretagem
25	3-4	escoramento	montagem
28	4	vigas e lajes	concretagem
29	0-1		início de paredes

Fonte: Próprio autor (2019).

O QUADRO 4.1 é ilustrativo, podendo variar de acordo com outros fatores e necessidades. Destaque para a introdução das paredes, na última linha, já está ligada às ações que a estrutura sofrerá – e não propriamente à execução da estrutura, porém interferirá nos esforços da estrutura ainda não completada.

Em relação ao procedimento de escoramento, atualmente é possível identificar dois modos gerais de processos construtivos das estruturas de concreto: o processo convencional e o processo construtivo racionalizado.

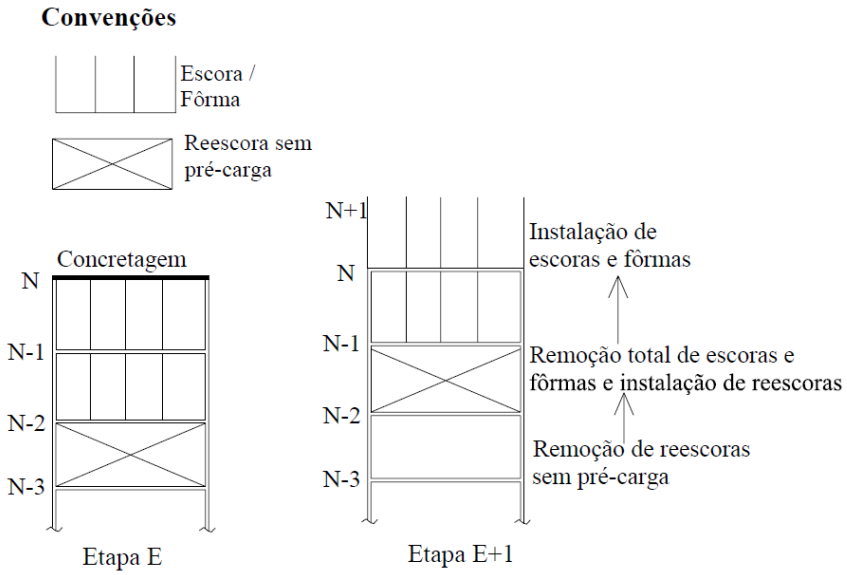
Freitas (2004) explica que o processo construtivo convencional se caracteriza pela remoção total de escoras e formas para a realização do reescoramento, ou seja, as reescoras são instaladas sem pré-carga. Este processo é utilizado principalmente em cimbramentos que fazem uso de escoras de madeira, embora escoras metálicas também possam ser utilizadas de maneira semelhante.

Normalmente, cinco etapas constituem a operação de construção. São elas:

1. Remoção de reescoras do nível mais baixo;
2. Remoção total de escoras e formas do nível mais baixo;
3. Instalação das reescoras no nível do pavimento onde as escoras e formas foram removidas;
4. Instalação das escoras e formas para a concretagem do próximo pavimento;
5. Concretagem.

Na FIGURA 4.1, está graficamente demonstrado o descrito anteriormente. Ressalto que, nas situações que Freitas coloca, há, no máximo, dois andares escorados sobre um terceiro.

FIGURA 4.1 - Processo construtivo convencional com sistema temporário de apoio (2+1)

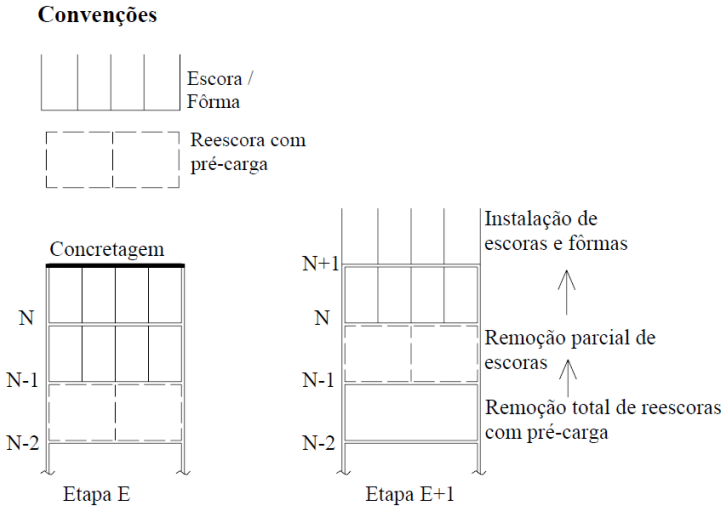


Fonte: FREITAS (2004)

Já o processo construtivo racionalizado se diferencia pela possibilidade de retirada parcial das escoras e retirada das formas. É dividido em quatro etapas, e se encontra ilustrado na FIGURA 4.2:

1. Remoção do nível mais baixo de reescoras;
2. Remoção parcial do nível mais baixo de escoras e formas (escoras não retiradas serão classificadas como reescoras);
3. Instalação das escoras e formas para a concretagem do próximo pavimento;
4. Concretagem.

FIGURA 4.2 - Processo construtivo racionalizado com sistema temporário de apoio (2+1)



Fonte: FREITAS (2004)

Independentemente do método de cimbramentos adotado, cada um dos elementos que o compõem deverá ter uma identidade atribuída, em que se devem:

1. Definir as propriedades físicas e geométricas para cada elemento;
2. Definir a data de início em que o elemento começará a fazer parte da estrutura;
3. Definir a data de término em que o elemento deixará de fazer parte da estrutura.

4.2. Elementos Estruturais (barras)

O modelo estrutural empregado é o de barras prismáticas. Visto o que já foi discutido aqui, pode-se afirmar que cada elemento definitivo – chamado de barra – (não as escoras) deverá representar cada pilar, viga, e, por meio de vigas em duas direções, uma grelha equivalente, representando as lajes do pavimento.

Conclui-se, assim, que cada barra terá uma identidade distinta, contendo:

1. Coordenadas dos nós iniciais e finais;
2. Ações nodais no elemento;
3. Ações distribuídas no elemento;
4. Restrições nodais;
5. Data de início em que o elemento começa a fazer parte da estrutura.

4.3. Linha do tempo e características

Como já exposto anteriormente, é primordial que a variável tempo seja considerada no modelo de análise. O que deve ficar claro em cada data é:

- Qual o sistema da estrutura naquele momento – por exemplo, são quatro pavimentos, sendo três escorados e um sem escora;
- Qual a resistência e o módulo de elasticidade do concreto de cada trecho de estrutura já executada;
- Quais as ações atuantes e em que data começaram a atuar no sistema estrutural (histórico da introdução das ações).

Desta forma, todas as datas devem ser incluídas no algoritmo. A análise (esforços e deslocamentos) se dará por acúmulo nos cálculos das diversas etapas. O número de etapas (e o mínimo de datas a considerar) condiz com o número das operações de execução da estrutura, mais aquelas que podem ocorrem simultaneamente à execução ou não, que correspondem à introdução das ações.

5. DESENVOLVIMENTO DO ALGORITMO

O complexo funcionamento de um algoritmo de análise estrutural, considerando as etapas construtivas e a variação das características do concreto, pode ser dividido em itens – e seu funcionamento é representado mais adiante, pela FIGURA 5.1.

Entrada de dados

A etapa de entrada de dados é a parte mais importante do desenvolvimento do algoritmo. Através da interpretação automatizada desses dados, o algoritmo faz, de forma eficiente e automática, calcular o comportamento estrutural ao longo do tempo, levando em consideração as variações nas características do concreto e as variações geométricas na estrutura.

Define-se, assim, que a entrada de dados deve conter:

1. Entrada de fórmulas adequadas para o cálculo da variação das características do concreto ao longo do tempo, conforme capítulo 2;
2. Modelagem geométrica completa da estrutura;
3. Definição das características do concreto utilizado com seus valores característicos em 28 dias;
4. Definição das datas de início de concretagem dos elementos da estrutura, e os valores atualizados das propriedades dos concretos em cada etapa a ser realizada;
5. Modelagem dos sistemas de cimbramentos para o suporte da estrutura em sua idade jovem;
6. Definição das datas de retirada dos cimbramentos;
7. Definição das ações permanentes, acidentais e de construção na estrutura;
8. Definição da data de atuação de cada uma das ações;
9. Definição de quantos dias são necessários para a simulação da estrutura.

Etapa 1

O algoritmo busca automaticamente, nos arquivos originários da entrada de dados, quais elementos entram na modelagem feita para o dia i. Sendo o dia i o primeiro dia da estrutura, não é necessário adaptar a geometria da estrutura, montada para o dia i. Cada modelagem leva em conta dados da modelagem anterior, como:

1. Geometria;
2. Cimbramento e carregamentos.

Caso o dia i tenha a retirada de algum cimbramento, este não entra mais na modelagem da estrutura e seus dados são arquivados.

Etapa 2

São aplicadas as cargas na estrutura relativas ao dia i da construção, levando em conta os carregamentos permanentes, acidentais e de construção aplicados até esse dia, e atualizando as características do concreto para a data em questão.

Etapa 3

Realiza-se, por meio do algoritmo, o cálculo estrutural relativo ao dia i da construção. Este cálculo é feito de acordo com os capítulos 2 e 3.

Ocorrido o equilíbrio da estrutura, os resultados dos cálculos para o dia i são salvos em um banco de dados, para posterior análise completa da mesma.

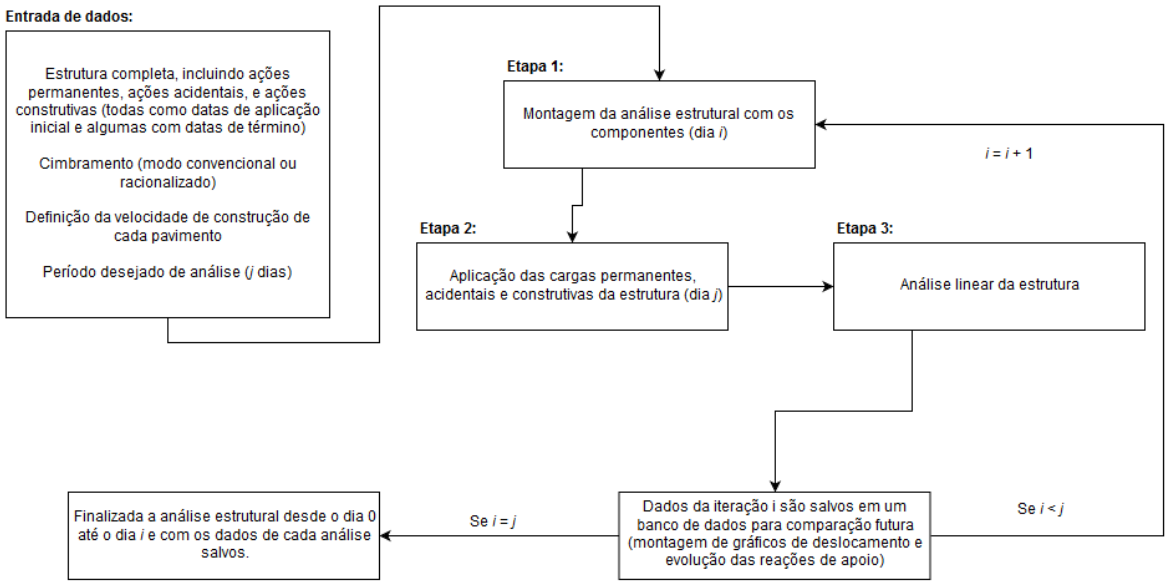
Após o término do cálculo do dia i, se ainda houver mais dias à frente, as etapas 1, 2 e 3 são repetidas, levando em consideração agora os valores atualizados para:

1. Módulo de elasticidade do concreto no dia $i+1$;
2. Resistência do concreto no dia $i+1$;

Finalização

Completado o cálculo estrutural, e com todos os resultados salvos apropriadamente em um banco de dados, é possível, através do programa desenvolvido com o algoritmo em questão, acompanhar e analisar a evolução dos deslocamentos e forças internas dentro da estrutura ao longo de sua construção – e, posteriormente, ao longo de seu uso. Tais análises permitem melhor definir as datas de carregamento e concretagem estrutural para que seja possível obter uma estrutura e obras mais seguras.

FIGURA 5.1 - Fluxograma de funcionamento do algoritmo



Fonte: Próprio autor (2019)

6. DESENVOLVIMENTO DO PROGRAMA

O desenvolvimento do programa se deu, basicamente, em três etapas distintas. A primeira fase constituiu-se em implementar o cálculo matricial compatível com estruturas de concreto armado moldado *in loco*. A segunda etapa consistiu na implementação do algoritmo, considerando o resultado da primeira etapa. Na terceira e última etapa, foi implementada a visualização gráfica tridimensional da estrutura e dos resultados do cálculo estrutural, realizado de acordo com as especificações do algoritmo.

6.1. Etapa 1

Primeiramente, para implementar uma análise matricial de estruturas tridimensionais, é necessário definir o que compõe um nó e um elemento. Assim, é dada a estrutura de um nó, conforme FIGURA 6.1, e o que compõe um elemento, conforme FIGURA 6.2.

FIGURA 6.1 - Classe de nós

```
1. [System.Serializable]
2.     public class Nodes
3.     {
4.         public int number;
5.
6.         public Vector3 position;
7.         public Restrictions restriction;
8.
9.         public Nodes()
10.          {
11.              position = new Vector3();
12.              restriction = new Restrictions();
13.          }
14.
15.          public Nodes(int number, string x, string y, string z,
   bool? x1, bool? y1, bool? z1, bool? x2, bool? y2, bool? z2)
16.          {
17.              this.number = number;
18.
19.              position = new Vector3(x, y, z);
20.              restriction = new Restrictions(x1.GetValueOrDefault(),
   y1.GetValueOrDefault(), z1.GetValueOrDefault(),
   x2.GetValueOrDefault(), y2.GetValueOrDefault(),
   z2.GetValueOrDefault());
21.          }
22.      }
```

FIGURA 6.2 - Classe de elementos

```
1. [System.Serializable]
2.     public class Member
3.     {
4.         public int number;
5.
6.         public int n0;
7.         public int n1;
8.
9.         public string material;
10.          public string section;
11.
12.          public int startDay;
13.          public int endDay;
14.
15.          public Member(int number, int n0, int n1, string material,
   string section, string startDay, string endDay)
16.          {
17.              this.number = number;
18.
19.              this.n0 = n0;
20.              this.n1 = n1;
21.              this.material = material;
22.              this.section = section;
23.
24.              this.startDay = int.Parse(startDay);
25.              this.endDay = int.Parse(endDay);
26.          }
27.      }
```

Fonte: Próprio autor (2019)

Necessariamente, tanto a classe de nós quanto a de elementos requerem subclasses, dadas pela FIGURA 6.3.

FIGURA 6.3 - Subclasses necessárias para montagem dos nós e elementos.

```
1. [System.Serializable]
2.     public class Vector3
3.     {
4.         public double x;
5.         public double y;
6.         public double z;
7.
8.         public Vector3()
9.         {
10.              x = 0;
11.              y = 0;
12.              z = 0;
13.          }
14.
15.          public Vector3(string x, string y, string z)
16.          {
17.              this.x = double.Parse(x);
18.              this.y = double.Parse(y);
19.              this.z = double.Parse(z);
20.          }
```

```
21.
22.         public Vector3(double x, double y, double z)
23.         {
24.             this.x = x;
25.             this.y = y;
26.             this.z = z;
27.         }
28.     }
29.
30.     [System.Serializable]
31.     public class Restrictions
32.     {
33.         public bool x1;
34.         public bool y1;
35.         public bool z1;
36.         public bool x2;
37.         public bool y2;
38.         public bool z2;
39.
40.         public Restrictions()
41.         {
42.             x1 = false;
43.             y1 = false;
44.             z1 = false;
45.             x2 = false;
46.             y2 = false;
47.             z2 = false;
48.         }
49.
50.         public Restrictions(bool x1, bool y1, bool z1, bool x2,
   bool y2, bool z2)
51.         {
52.             this.x1 = x1;
53.             this.y1 = y1;
54.             this.z1 = z1;
55.             this.x2 = x2;
56.             this.y2 = y2;
57.             this.z2 = z2;
58.         }
59.     }
60.
61.     [System.Serializable]
62.     public class Material
63.     {
64.         public string name;
65.
66.         public double E;
67.         public double G;
68.         public double v;
69.
70.         public bool timeDependent;
71.
72.         public double fck;
73.
74.         public Material()
75.         {
76.             name = "";
77.
78.             E = 0;
79.             G = 0;
80.             v = 0;
```

```
81.
82.             timeDependent = false;
83.
84.             fck = 0;
85.         }
86.
87.         public Material(string name, string E, string G, bool time,
    string fck = "0", string v = "0")
88.         {
89.             this.name = name;
90.
91.             this.E = double.Parse(E);
92.             this.G = double.Parse(G);
93.             this.v = double.Parse(v);
94.
95.             this.timeDependent = time;
96.
97.             this.fck = double.Parse(fck);
98.         }
99.     }
100.
101.    [System.Serializable]
102.    public class Property
103.    {
104.        public string name;
105.
106.        public double A;
107.        public double Ix;
108.        public double Iy;
109.        public double Iz;
110.
111.        public string type;
112.
113.        public double d;
114.
115.        public double h;
116.        public double b;
117.
118.        public Property(string name, string A, string Ix, string
    Iy, string Iz, string type, string d = "0", string h = "0", string b
    = "0")
119.        {
120.            if (d == "") d = "0";
121.            if (b == "") b = "0";
122.            if (h == "") h = "0";
123.
124.            if (type == "0") type = "None";
125.            if (type == "1") type = "Circular";
126.            if (type == "2") type = "Retangular";
127.
128.            this.name = name;
129.
130.            this.A = double.Parse(A);
131.            this.Ix = double.Parse(Ix);
132.            this.Iy = double.Parse(Iy);
133.            this.Iz = double.Parse(Iz);
134.
135.            this.type = type;
136.
137.            this.d = double.Parse(d);
138.            this.h = double.Parse(h);
```

```
139.             this.b = double.Parse(b);
140.         }
141.     }
```

Fonte: Próprio autor (2019)

Também são necessárias classes para as forças nodais e para as forças distribuídas nos elementos, conforme FIGURA 6.4.

FIGURA 6.4 - Classes de forças nodais e forças distribuídas na estrutura

```
1. [System.Serializable]
2.     public class NodalForces
3.     {
4.         public string name;
5.
6.         public Vector3 disp;
7.         public Vector3 rott;
8.
9.         public List<int> nodes;
10.
11.          public int startDay;
12.          public int endDay;
13.
14.          public NodalForces()
15.          {
16.              name = "";
17.
18.              disp = new Vector3();
19.              rott = new Vector3();
20.
21.              nodes = new List<int>();
22.
23.              startDay = 0;
24.              endDay = 0;
25.          }
26.
27.          public NodalForces(string name, double dx, double dy,
    double dz, double rx, double ry, double rz, int startDay, int endDay,
    List<int> list)
28.          {
29.              this.name = name;
30.
31.              disp = new Vector3(dx, dy, dz);
32.              rott = new Vector3(rz, ry, rz);
33.
34.              nodes = list;
35.
36.              this.startDay = startDay;
37.              this.endDay = endDay;
38.          }
39.      }
40.
41.      [System.Serializable]
42.      public class MemberForces
43.      {
44.          public string name;
45.
46.          public Vector3 disp;
```

```
47.
48.          public List<int> members;
49.
50.          public int startDay;
51.          public int endDay;
52.
53.          public MemberForces()
54.          {
55.              name = "";
56.
57.              disp = new Vector3();
58.
59.              members = new List<int>();
60.
61.              startDay = 0;
62.              endDay = 0;
63.          }
64.
65.          public MemberForces(string name, double dx, double dy,
   double dz, int startDay, int endDay, List<int> list)
66.          {
67.              this.name = name;
68.
69.              disp = new Vector3(dx, dy, dz);
70.
71.              members = list;
72.
73.              this.startDay = startDay;
74.              this.endDay = endDay;
75.          }
76.      }
```

Fonte: Próprio autor (2019)

Para dar início ao processo de análise matricial, é necessário montar a matriz de rigidez de um elemento, dada pela função MountKMatrix, conforme FIGURA 6.5.

FIGURA 6.5 - Função MountKMatrix

```
1. public void MountKMatrix()
2.          {
3.              double L_2 = Math.Pow(L, 2);
4.              double L_3 = Math.Pow(L, 3);
5.
6.              double E_Iz = material.E * properties.Iz;
7.              double E_Iy = material.E * properties.Iy;
8.              double G_Ix = material.G * properties.Ix;
9.              double E_A = material.E * properties.A;
10.
11.               // Line 0.
12.               K.SetElement(0, 0, (E_A) / L);
13.               K.SetElement(0, 6, -(E_A) / L);
14.
15.               // Line 1.
16.               K.SetElement(1, 1, (12d * E_Iz) / L_3);
17.               K.SetElement(1, 5, (6d * E_Iz) / L_2);
18.               K.SetElement(1, 7, -(12d * E_Iz) / L_3);
19.               K.SetElement(1, 11, (6d * E_Iz) / L_2);
20.
```

```
21.             // Line 2.
22.             K.SetElement(2, 2, (12d * E_Iy) / L_3);
23.             K.SetElement(2, 4, -(6d * E_Iy) / L_2);
24.             K.SetElement(2, 8, -(12d * E_Iy) / L_3);
25.             K.SetElement(2, 10, -(6d * E_Iy) / L_2);
26.
27.             // Line 3.
28.             K.SetElement(3, 3, (G_Ix) / L);
29.             K.SetElement(3, 9, -(G_Ix) / L);
30.
31.             // Line 4.
32.             K.SetElement(4, 2, -(6d * E_Iy) / L_2);
33.             K.SetElement(4, 4, (4d * E_Iy) / L);
34.             K.SetElement(4, 8, (6d * E_Iy) / L_2);
35.             K.SetElement(4, 10, (2d * E_Iy) / L);
36.
37.             // Line 5.
38.             K.SetElement(5, 1, (6d * E_Iz) / L_2);
39.             K.SetElement(5, 5, (4d * E_Iz) / L);
40.             K.SetElement(5, 7, -(6d * E_Iz) / L_2);
41.             K.SetElement(5, 11, (2d * E_Iz) / L);
42.
43.             // Line 6.
44.             K.SetElement(6, 0, -(E_A) / L);
45.             K.SetElement(6, 6, (E_A) / L);
46.
47.             // Line 7.
48.             K.SetElement(7, 1, -(12d * E_Iz) / L_3);
49.             K.SetElement(7, 5, -(6d * E_Iz) / L_2);
50.             K.SetElement(7, 7, (12d * E_Iz) / L_3);
51.             K.SetElement(7, 11, -(6d * E_Iz) / L_2);
52.
53.             // Line 8.
54.             K.SetElement(8, 2, -(12d * E_Iy) / L_3);
55.             K.SetElement(8, 4, (6d * E_Iy) / L_2);
56.             K.SetElement(8, 8, (12d * E_Iy) / L_3);
57.             K.SetElement(8, 10, (6d * E_Iy) / L_2);
58.
59.             // Line 9.
60.             K.SetElement(9, 3, -(G_Ix) / L);
61.             K.SetElement(9, 9, (G_Ix) / L);
62.
63.             // Line 10.
64.             K.SetElement(10, 2, -(6d * E_Iy) / L_2);
65.             K.SetElement(10, 4, (2d * E_Iy) / L);
66.             K.SetElement(10, 8, (6d * E_Iy) / L_2);
67.             K.SetElement(10, 10, (4d * E_Iy) / L);
68.
69.             // Line 11.
70.             K.SetElement(11, 1, (6d * E_Iz) / L_2);
71.             K.SetElement(11, 5, (2d * E_Iz) / L);
72.             K.SetElement(11, 7, -(6d * E_Iz) / L_2);
73.             K.SetElement(11, 11, (4d * E_Iz) / L);
74.         }
```

Fonte: Próprio autor (2019)

A montagem da matriz de rotação de cada elemento se dá pela função descrita na FIGURA 6.6, com atenção especial para casos em que a barra se encontra na vertical.

FIGURA 6.6 - Função MountRMatrix, de montagem da matriz de rotação de um elemento

```
1. public void MountRMatrix()
2.         {
3.             GeneralMatrix R0 = new GeneralMatrix(3, 3);
4.
5.             double CX = (N1.X - N0.X) / L;
6.             double CY = (N1.Y - N0.Y) / L;
7.             double CZ = (N1.Z - N0.Z) / L;
8.
9.             double alpha = rotation;
10.
11.              double cosAlpha = Math.Cos(alpha);
12.              double sinAlpha = Math.Sin(alpha);
13.
14.              if (CX == 0 && CZ == 0)
15.              {
16.                  R0.SetElement(0, 0, 0);
17.                  R0.SetElement(0, 1, CY);
18.                  R0.SetElement(0, 2, 0);
19.
20.                  R0.SetElement(1, 0, -CY * cosAlpha);
21.                  R0.SetElement(1, 1, 0);
22.                  R0.SetElement(1, 2, sinAlpha);
23.
24.                  R0.SetElement(2, 0, CY * sinAlpha);
25.                  R0.SetElement(2, 1, 0);
26.                  R0.SetElement(2, 2, cosAlpha);
27.              }
28.              else
29.              {
30.                  R0.SetElement(0, 0, CX);
31.                  R0.SetElement(0, 1, CY);
32.                  R0.SetElement(0, 2, CZ);
33.
34.                  R0.SetElement(1, 0, ((-CX * CY * cosAlpha) - (CZ *
   sinAlpha)) / Math.Sqrt(Math.Pow(CX, 2) + Math.Pow(CZ, 2)));
35.                  R0.SetElement(1, 1, Math.Sqrt(Math.Pow(CX, 2) +
   Math.Pow(CZ, 2)) * cosAlpha);
36.                  R0.SetElement(1, 2, ((-CY * CZ * cosAlpha) + (CX *
   sinAlpha)) / Math.Sqrt(Math.Pow(CX, 2) + Math.Pow(CZ, 2)));
37.
38.                  R0.SetElement(2, 0, ((CX * CY * sinAlpha) - (CZ *
   cosAlpha)) / Math.Sqrt(Math.Pow(CX, 2) + Math.Pow(CZ, 2)));
39.                  R0.SetElement(2, 1, -Math.Sqrt(Math.Pow(CX, 2) +
   Math.Pow(CZ, 2)) * sinAlpha);
40.                  R0.SetElement(2, 2, ((CY * CZ * sinAlpha) + (CX *
   cosAlpha)) / Math.Sqrt(Math.Pow(CX, 2) + Math.Pow(CZ, 2)));
41.              }
42.
43.              for (int i = 0; i <= 2; i++)
44.              {
45.                  for (int j = 0; j <= 2; j++)
46.                  {
47.                      for (int k = 0; k <= 9; k += 3)
48.                      {
49.                          R.SetElement(i + k, j + k, R0.GetElement(i,
   j));
50.                      }
51.                  }
```

```
52.                 }
53.             }
```

Fonte: Próprio autor (2019)

A montagem da matriz de rigidez da estrutura, K_g, é dada conforme FIGURA 6.7, com especial atenção para o correto posicionamento de cada um dos elementos de barra dentro da matriz de rigidez global.

FIGURA 6.7 - Montagem da matriz de rigidez global da estrutura

```
1. public GeneralMatrix Mount_KG_Matrix(List<Node> nodes, List<Member>
   members)
2.         {
3.             GeneralMatrix KG = new GeneralMatrix(6 * nodes.Count(), 6
   * nodes.Count());
4.             //GeneralMatrix KGCheck = new GeneralMatrix(6 *
   nodes.Count(), 6 * nodes.Count());
5.
6.             double tempKG = 0;
7.
8.             for (int i = 0; i < members.Count(); i++)
9.             {
10.                 for (int j = 0; j < 6; j++)
11.                 {
12.                     for (int k = 0; k < 6; k++)
13.                     {
14.                         tempKG = KG.GetElement((int)members[i].NI *
   6 + j, (int)members[i].NI * 6 + k);
15.                         KG.SetElement((int)members[i].NI * 6 + j,
   (int)members[i].NI * 6 + k, tempKG + members[i].RT_K_R.GetElement(j,
   k));
16.                     }
17.                 }
18.
19.                 for (int j = 0; j < 6; j++)
20.                 {
21.                     for (int k = 6; k < 12; k++)
22.                     {
23.                         tempKG = KG.GetElement((int)members[i].NI *
   6 + j, (int)members[i].NF * 6 + k - 6);
24.                         KG.SetElement((int)members[i].NI * 6 + j,
   (int)members[i].NF * 6 + k - 6, tempKG +
   members[i].RT_K_R.GetElement(j, k));
25.                     }
26.                 }
27.
28.                 for (int j = 6; j < 12; j++)
29.                 {
30.                     for (int k = 0; k < 6; k++)
31.                     {
32.                         tempKG = KG.GetElement((int)members[i].NF *
   6 + j - 6, (int)members[i].NI * 6 + k);
33.                         KG.SetElement((int)members[i].NF * 6 + j -
   6, (int)members[i].NI * 6 + k, tempKG +
   members[i].RT_K_R.GetElement(j, k));
34.                     }
35.                 }
```

```
36.
37.                     for (int j = 6; j < 12; j++)
38.                     {
39.                         for (int k = 6; k < 12; k++)
40.                         {
41.                             tempKG = KG.GetElement((int)members[i].NF *
   6 + j - 6, (int)members[i].NF * 6 + k - 6);
42.                             KG.SetElement((int)members[i].NF * 6 + j -
   6, (int)members[i].NF * 6 + k - 6, tempKG +
   members[i].RT_K_R.GetElement(j, k));
43.                         }
44.                     }
45.                 }
46.
47.                 return KG;
48.             }
```

Fonte: Próprio autor (2019)

A função de aplicação das condições de contorno na estrutura ocorre conforme FIGURA 6.8.

FIGURA 6.8 - Função de aplicação das condições de contorno na matriz global

```
1. public GeneralMatrix Mount_KGC_Matrix(List<Node> nodes, GeneralMatrix
   KG)
2.         {
3.             GeneralMatrix KGC = new GeneralMatrix(KG.RowDimension,
   KG.ColumnDimension);
4.
5.             KGC = KG.Copy();
6.
7.             for (int i = 0; i < nodes.Count; i++)
8.             {
9.                 if (nodes[i].restrictionF.X)
10.                 {
11.                     for (int j = 0; j < KG.RowDimension; j++)
12.                     {
13.                         KGC.SetElement(6 * i, j, 0);
14.                         KGC.SetElement(j, 6 * i, 0);
15.                     }
16.                     KGC.SetElement(6 * i, 6 * i, 1);
17.                 }
18.
19.                 if (nodes[i].restrictionF.Y)
20.                 {
21.                     for (int j = 0; j < KG.RowDimension; j++)
22.                     {
23.                         KGC.SetElement(6 * i + 1, j, 0);
24.                         KGC.SetElement(j, 6 * i + 1, 0);
25.                     }
26.                     KGC.SetElement(6 * i + 1, 6 * i + 1, 1);
27.                 }
28.
29.                 if (nodes[i].restrictionF.Z)
30.                 {
31.                     for (int j = 0; j < KG.RowDimension; j++)
32.                     {
33.                         KGC.SetElement(6 * i + 2, j, 0);
```

```
34.                         KGC.SetElement(j, 6 * i + 2, 0);
35.                     }
36.                     KGC.SetElement(6 * i + 2, 6 * i + 2, 1);
37.                 }
38.
39.                 if (nodes[i].restrictionM.X)
40.                 {
41.                     for (int j = 0; j < KG.RowDimension; j++)
42.                     {
43.                         KGC.SetElement(6 * i + 3, j, 0);
44.                         KGC.SetElement(j, 6 * i + 3, 0);
45.                     }
46.                     KGC.SetElement(6 * i + 3, 6 * i + 3, 1);
47.                 }
48.
49.                 if (nodes[i].restrictionM.Y)
50.                 {
51.                     for (int j = 0; j < KG.RowDimension; j++)
52.                     {
53.                         KGC.SetElement(6 * i + 4, j, 0);
54.                         KGC.SetElement(j, 6 * i + 4, 0);
55.                     }
56.                     KGC.SetElement(6 * i + 4, 6 * i + 4, 1);
57.                 }
58.
59.                 if (nodes[i].restrictionM.Z)
60.                 {
61.                     for (int j = 0; j < KG.RowDimension; j++)
62.                     {
63.                         KGC.SetElement(6 * i + 5, j, 0);
64.                         KGC.SetElement(j, 6 * i + 5, 0);
65.                     }
66.                     KGC.SetElement(6 * i + 5, 6 * i + 5, 1);
67.                 }
68.             }
69.
70.             return KGC;
71.         }
```

Fonte: Próprio autor (2019)

A função de reorganização das matrizes de rigidez global da estrutura, necessária para a correta simulação estrutural, é demonstrada na FIGURA 6.9.

FIGURA 6.9 - Função de reorganização da matriz de rigidez global da estrutura, considerando as condições de contorno.

```
1. public GeneralMatrix Reord_KG(List<Node> nodes, GeneralMatrix KG)
2.         {
3.             GeneralMatrix KGR = new GeneralMatrix(KG.RowDimension,
   KG.ColumnDimension);
4.
5.             int nodeCount = 0;
6.
7.             for (int i = 0; i < KG.RowDimension; i++)
8.             {
9.                 for (int j = 0; j < KG.ColumnDimension; j++)
10.                 {
```

```
11.                     KGR.SetElement(nodes[nodeCount].reord.a1, j,
    KG.GetElement(i, j));
12.                 }
13.                 i++;
14.                 for (int j = 0; j < KG.ColumnDimension; j++)
15.                 {
16.                     KGR.SetElement(nodes[nodeCount].reord.a2, j,
    KG.GetElement(i, j));
17.                 }
18.                 i++;
19.                 for (int j = 0; j < KG.ColumnDimension; j++)
20.                 {
21.                     KGR.SetElement(nodes[nodeCount].reord.a3, j,
    KG.GetElement(i, j));
22.                 }
23.                 i++;
24.                 for (int j = 0; j < KG.ColumnDimension; j++)
25.                 {
26.                     KGR.SetElement(nodes[nodeCount].reord.a4, j,
    KG.GetElement(i, j));
27.                 }
28.                 i++;
29.                 for (int j = 0; j < KG.ColumnDimension; j++)
30.                 {
31.                     KGR.SetElement(nodes[nodeCount].reord.a5, j,
    KG.GetElement(i, j));
32.                 }
33.                 i++;
34.                 for (int j = 0; j < KG.ColumnDimension; j++)
35.                 {
36.                     KGR.SetElement(nodes[nodeCount].reord.a6, j,
    KG.GetElement(i, j));
37.                 }
38.                 nodeCount++;
39.             }
40.
41.             GeneralMatrix KGRC = Reord_KGR_Column(nodes, KGR);
42.
43.             return KGRC;
44.         }
45.
46.         public GeneralMatrix Reord_KGR_Column(List<Node> nodes,
    GeneralMatrix KGR)
47.         {
48.             GeneralMatrix KGRC = new
    GeneralMatrix(KGR.RowDimension, KGR.ColumnDimension);
49.
50.             int nodeCount = 0;
51.
52.             for (int i = 0; i < KGR.RowDimension; i++)
53.             {
54.                 for (int j = 0; j < KGR.ColumnDimension; j++)
55.                 {
56.                     KGRC.SetElement(j, nodes[nodeCount].reord.a1,
    KGR.GetElement(j, i));
57.                 }
58.                 i++;
59.                 for (int j = 0; j < KGR.ColumnDimension; j++)
60.                 {
61.                     KGRC.SetElement(j, nodes[nodeCount].reord.a2,
    KGR.GetElement(j, i));
```

```
62.                     }
63.                     i++;
64.                     for (int j = 0; j < KGR.ColumnDimension; j++)
65.                     {
66.                         KGRC.SetElement(j, nodes[nodeCount].reord.a3,
   KGR.GetElement(j, i));
67.                     }
68.                     i++;
69.                     for (int j = 0; j < KGR.ColumnDimension; j++)
70.                     {
71.                         KGRC.SetElement(j, nodes[nodeCount].reord.a4,
   KGR.GetElement(j, i));
72.                     }
73.                     i++;
74.                     for (int j = 0; j < KGR.ColumnDimension; j++)
75.                     {
76.                         KGRC.SetElement(j, nodes[nodeCount].reord.a5,
   KGR.GetElement(j, i));
77.                     }
78.                     i++;
79.                     for (int j = 0; j < KGR.ColumnDimension; j++)
80.                     {
81.                         KGRC.SetElement(j, nodes[nodeCount].reord.a6,
   KGR.GetElement(j, i));
82.                     }
83.                     nodeCount++;
84.                 }
85.
86.                 return KGRC;
87.             }
```

Fonte: Próprio autor (2019)

6.2. Etapa 2

A montagem de cada simulação, para cada dia *i* da estrutura, é dada pela função descrita na FIGURA 6.10. Para cada análise, é montada uma geometria diferente da estrutura, considerando as características do concreto de cada elemento até o momento, atualizando-se o módulo de elasticidade e a resistência do concreto à compressão. Ressalta-se que, se o nó não existe ainda na estrutura (o elemento conectado a este nó não começou a ser construído), ele não é adicionado à simulação daquele dia, mantendo assim a eficiência do algoritmo de resolução da estrutura e evitando problemas de simulação computacional.

FIGURA 6.10 - Função de montagem da análise para cada dia de simulação

```
1. private void MountModels()
2.         {
3.             try
4.             {
5.                 startDay = int.Parse(simulationDayStart.Text);
6.                 endDay = int.Parse(simulationDayEnd.Text);
7.
```

```
8.                  models.Clear();
9.
10.                   for (int i = startDay; i <= endDay; i++)
11.                   {
12.                       var model = new BriefFiniteElementNet.Model();
13.
14.                       var acceptedNodes = new List<int>();
15.
16.                       foreach (Member member in members)
17.                       {
18.                           if (i >= member.startDay)
19.                           {
20.                               if (member.endDay >= i)
21.                               {
22.                                   if
   (!acceptedNodes.Contains(member.n0)) acceptedNodes.Add(member.n0);
23.                                   if
   (!acceptedNodes.Contains(member.n1)) acceptedNodes.Add(member.n1);
24.                               }
25.                           }
26.                       }
27.
28.                       foreach (int n in acceptedNodes)
29.                       {
30.                           var node = new Node(nodes[n].position.x,
   nodes[n].position.y, nodes[n].position.z);
31.
32.                           if (nodes[n].restriction.x1)
   node.Constraints &= Constraints.FixedDX;
33.                           if (nodes[n].restriction.y1)
   node.Constraints &= Constraints.FixedDY;
34.                           if (nodes[n].restriction.z1)
   node.Constraints &= Constraints.FixedDZ;
35.                           if (nodes[n].restriction.x2)
   node.Constraints &= Constraints.FixedRX;
36.                           if (nodes[n].restriction.y2)
   node.Constraints &= Constraints.FixedRY;
37.                           if (nodes[n].restriction.z2)
   node.Constraints &= Constraints.FixedRZ;
38.
39.                           node.Label = n.ToString();
40.
41.                           model.Nodes.Add(node);
42.                       }
43.
44.                       foreach (Member member in members)
45.                       {
46.                           if (i >= member.startDay)
47.                           {
48.                               if (member.endDay >= i)
49.                               {
50.                                   var mem = new
   FrameElement2Node(model.Nodes.Where(item => item.Label ==
   member.n0.ToString()).First(), model.Nodes.Where(item => item.Label
   == member.n1.ToString()).First());
51.
52.                                   mem.Label = "M" +
   member.number.ToString();
53.
54.                                   if (materials.Where(item =>
   item.name == member.material).First().timeDependent)
```

```
55.                                     {
56.                                         mem.E =
   GetElasticityModulusABNT(materials.Where(item => item.name ==
   member.material).First().fck / 1000, i - member.startDay);
57.                                         mem.G = CalculateG(mem.E,
   materials.Where(item => item.name == member.material).First().v);
58.                                     }
59.                                     else
60.                                     {
61.                                         mem.E = materials.Where(item =>
   item.name == member.material).First().E;
62.                                         mem.G = materials.Where(item =>
   item.name == member.material).First().G;
63.                                     }
64.
65.                                     mem.A = properties.Where(item =>
   item.name == member.section).First().A;
66.                                     mem.Iy = properties.Where(item =>
   item.name == member.section).First().Ix;
67.                                     mem.Iz = properties.Where(item =>
   item.name == member.section).First().Iy;
68.                                     mem.J = properties.Where(item =>
   item.name == member.section).First().Iz;
69.
70.                                     mem.UseOverridedProperties = true;
71.
72.                                     model.Elements.Add(mem);
73.                                 }
74.                             }
75.                         }
76.
77.                         foreach (NodalForces force in nodalForces)
78.                         {
79.                             if (i >= force.startDay)
80.                             {
81.                                 if (force.endDay >= i)
82.                                 {
83.                                     foreach (int node in force.nodes)
84.                                     {
85.                                         if (model.Nodes.Where(item =>
   item.Label == node.ToString()).ToList().Count > 0)
86.                                         {
87.                                             var f = new
   Force(force.disp.x, force.disp.y, force.disp.z, force.rott.x,
   force.rott.y, force.rott.z);
88.                                             model.Nodes.Where(item =>
   item.Label == node.ToString()).First().Loads.Add(new NodalLoad(f));
89.                                         }
90.                                     }
91.                                 }
92.                             }
93.                         }
94.
95.                         foreach (MemberForces force in memberForces)
96.                         {
97.                             if (i >= force.startDay)
98.                             {
99.                                 if (force.endDay >= i)
100.                                {
101.                                    foreach (int member in
    force.members)
```

```
102.                                     {
103.                                         if (model.Elements.Where(item
    => item.Label == "M" + member.ToString()).ToList().Count > 0)
104.                                         {
105.                                             if (force.disp.x != 0)
106.                                             {
107.                                                 var load = new
    UniformLoad1D(force.disp.x, LoadDirection.X,
    CoordinationSystem.Global);
108.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
109.                                             }
110.                                             if (force.disp.y != 0)
111.                                             {
112.                                                 var load = new
    UniformLoad1D(force.disp.y, LoadDirection.Y,
    CoordinationSystem.Global);
113.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
114.                                             }
115.                                             if (force.disp.z != 0)
116.                                             {
117.                                                 var load = new
    UniformLoad1D(force.disp.z, LoadDirection.Z,
    CoordinationSystem.Global);
118.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
119.                                             }
120.                                         }
121.                                     }
122.
123.                                 }
124.                             }
125.                         }
126.
127.                         models.Add(model);
128.                     }
129.                 }
130.                 catch
131.                 {
132.
133.                 }
134.             }
```

Fonte: Próprio autor (2019)

Em seguida, as simulações são rodadas em paralelo, utilizando toda a capacidade computacional do processador, através da função dada pela FIGURA 6.11.

FIGURA 6.11 - Função paralelizada para chamar a análise de cada dia da estrutura

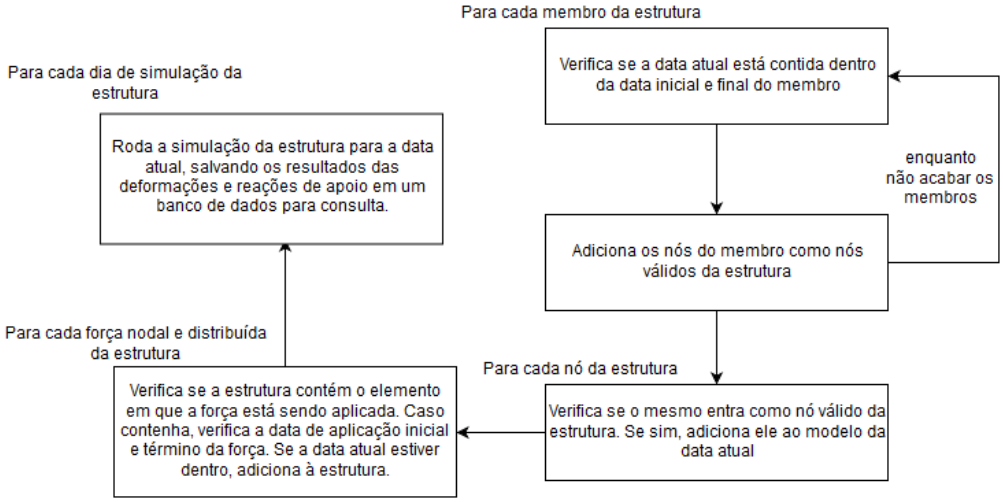
```
1. private void RunSimulations()
2.         {
3.             Parallel.ForEach(models, model =>
4.             {
```

```
5.                  if (model.Nodes.Count > 0 && model.Elements.Count >
    0)
6.                  {
7.                      model.Solve();
8.                  }
9.              });
10.         }
```

Fonte: Próprio autor (2019)

A FIGURA 6.12 ilustra o fluxograma do funcionamento da montagem da estrutura para cada dia de análise. O programa toma cuidado para não adicionar à análise nós que não pertençam ainda a nenhum membro atual da estrutura. As simulações contam com cada nó e elemento tendo um identificador único, para posterior análise dos resultados de todas as simulações daquele nó/elemento.

FIGURA 6.12 - Fluxograma do funcionamento da montagem da estrutura



Fonte: Próprio autor (2019)

6.3. Etapa 3

Demonstramos aqui o comportamento do programa em relação à montagem da estrutura. Utilizando uma disposição simples de vigas e pilares, o programa simula uma estrutura com um montagem estrutural similar à disposta no QUADRO 4.1, composta por 5 pilares e 6 vigas em cada andar, conforme FIGURA 6.15:

Passo 1:

De acordo com a geometria e os dados fornecidos, o programa começa a posicionar automaticamente os pilares a serem concretados. São rodadas as simulações da estrutura envolvendo apenas a situação atual da estrutura.

Passo 2:

Automaticamente, o programa passa a colocar o cimbramentos, bem como as vigas e lajes. São rodadas as simulações da estrutura nos dias em que a estrutura mantém esta configuração estrutural.

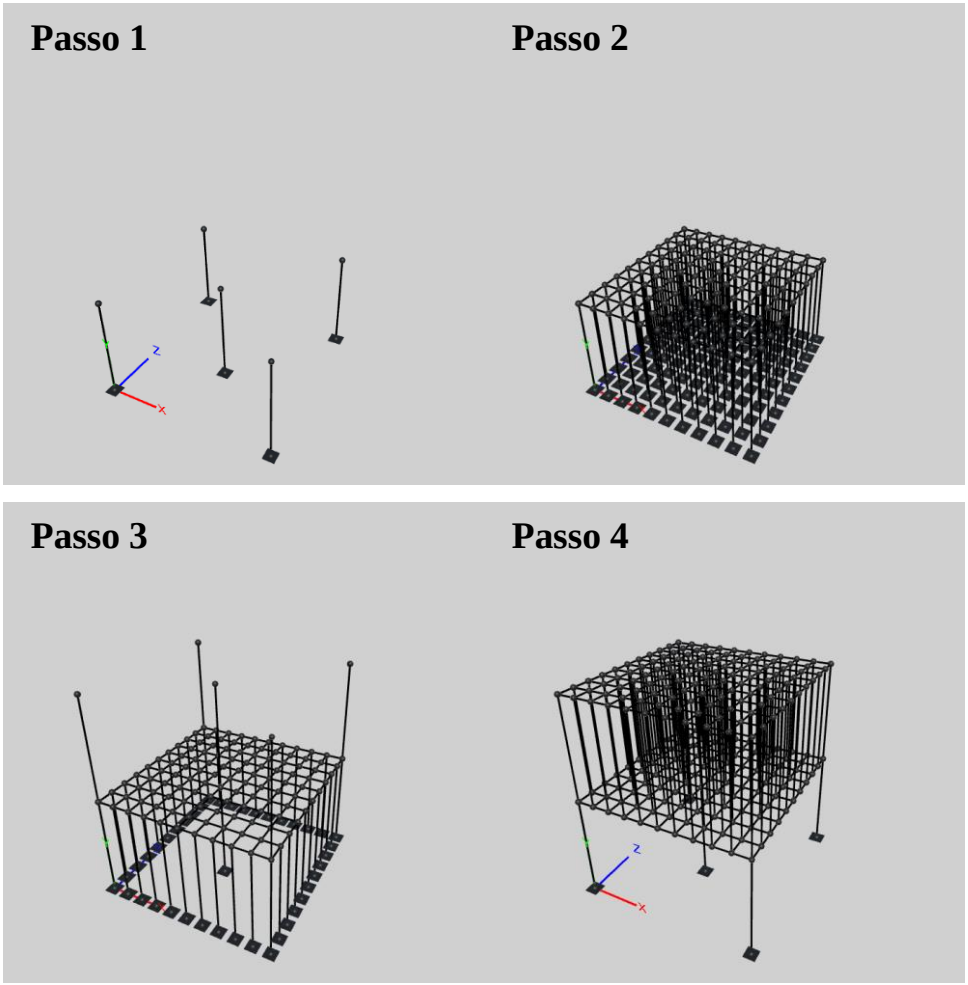
Passo 3:

De acordo com o calendário de execução dado pelo engenheiro projetista, o programa automaticamente começa a colocar novas seções de pilares. É retirado o cimbramento de suporte da laje, mas mantêm-se o da viga. Novamente são rodadas simulações da estrutura na situação atual.

Passo 4:

Repete-se o passo 2 para o segundo andar. É retirado completamente o cimbramento do primeiro andar.

FIGURA 6.13 - Simulação da montagem da estrutura, passos 1 a 4

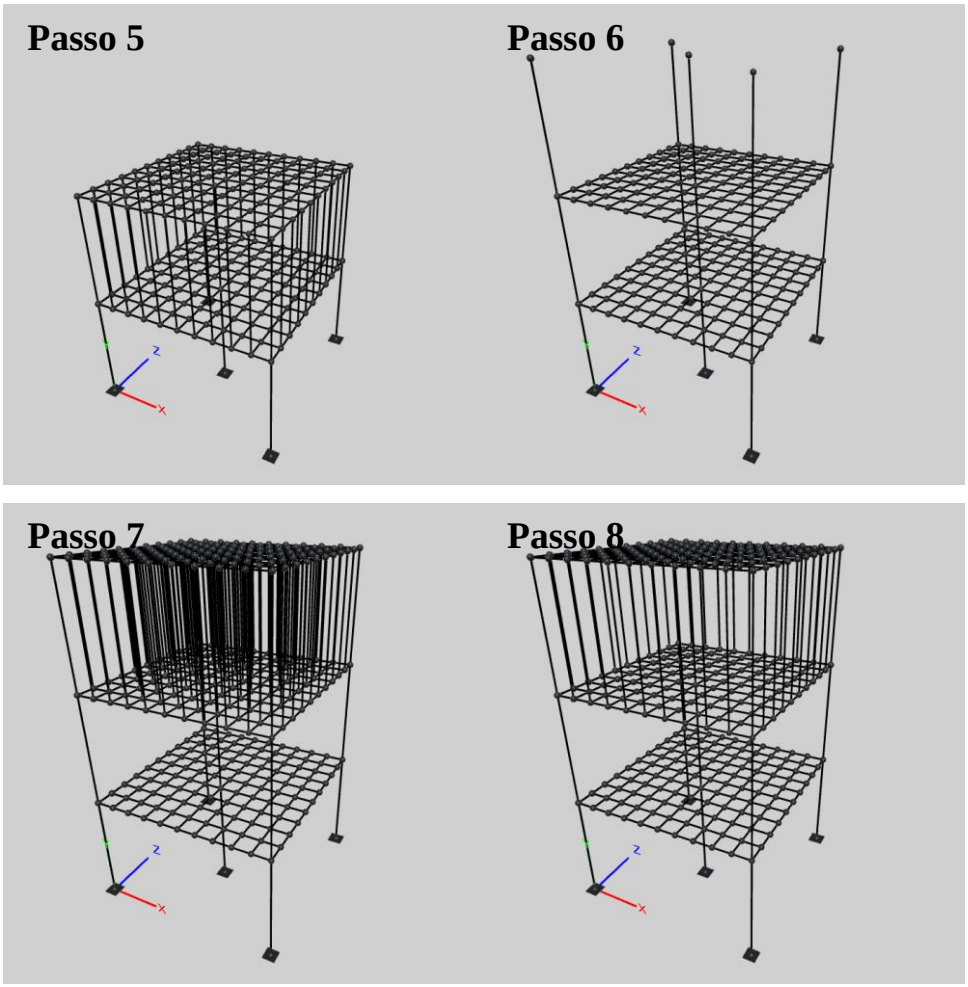


Fonte: Próprio autor (2019)

Passos 5 a 8:

Análogos aos passos 1 a 4.

FIGURA 6.14 – Simulação da montagem da estrutura, passos 5 a 8

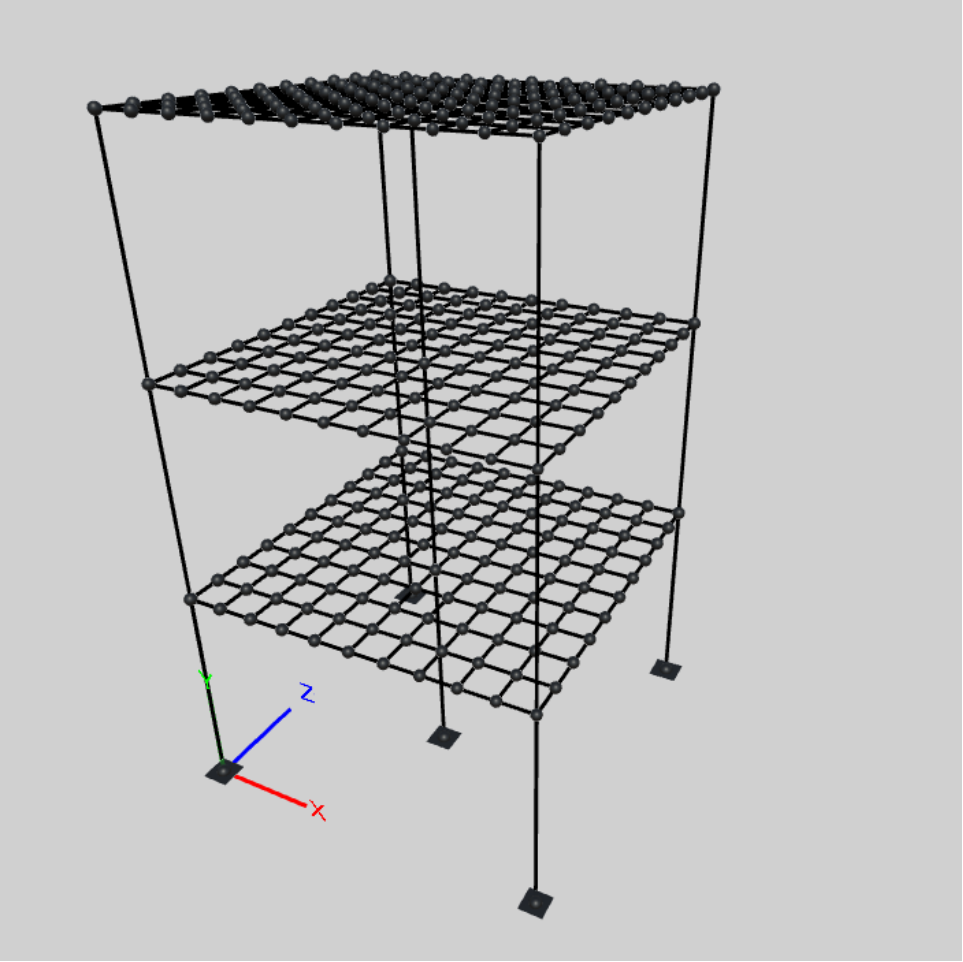


Fonte: Próprio autor (2019)

Passo 9:

Simulação final da estrutura, em sua forma final. Caso esta análise fosse convencional, apenas essa geometria e carregamento configurariam toda a análise da estrutura.

FIGURA 6.15 – Simulação da montagem da estrutura, passo 9



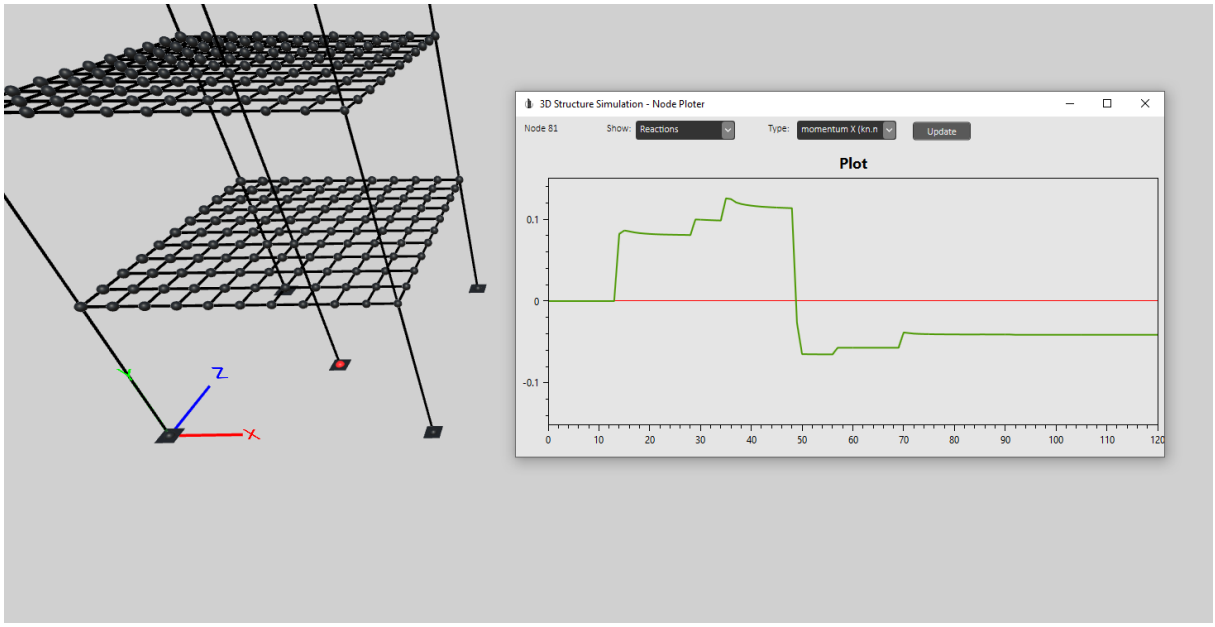
Fonte: Próprio autor (2019)

De posse de todos os dados da simulação, é possível, através do programa, verificar a evolução das reações de apoio nos nós da estrutura, bem como os deslocamentos.

Para a estrutura montada anteriormente, foram rodadas as simulações dentro do programa do dia 0 ao dia 120. Os resultados e análises de forças de reação e deslocamento são apresentados nas FIGURA 6.16 à FIGURA 6.22.

Pode-se verificar na FIGURA 6.16 que o momento na direção X na base do pilar central sofre uma inversão de sentido durante o processo construtivo, indo de 0.1 kNm para -0.1 kNm em menos de um dia, e estabilizando ao longo do processo construtivo.

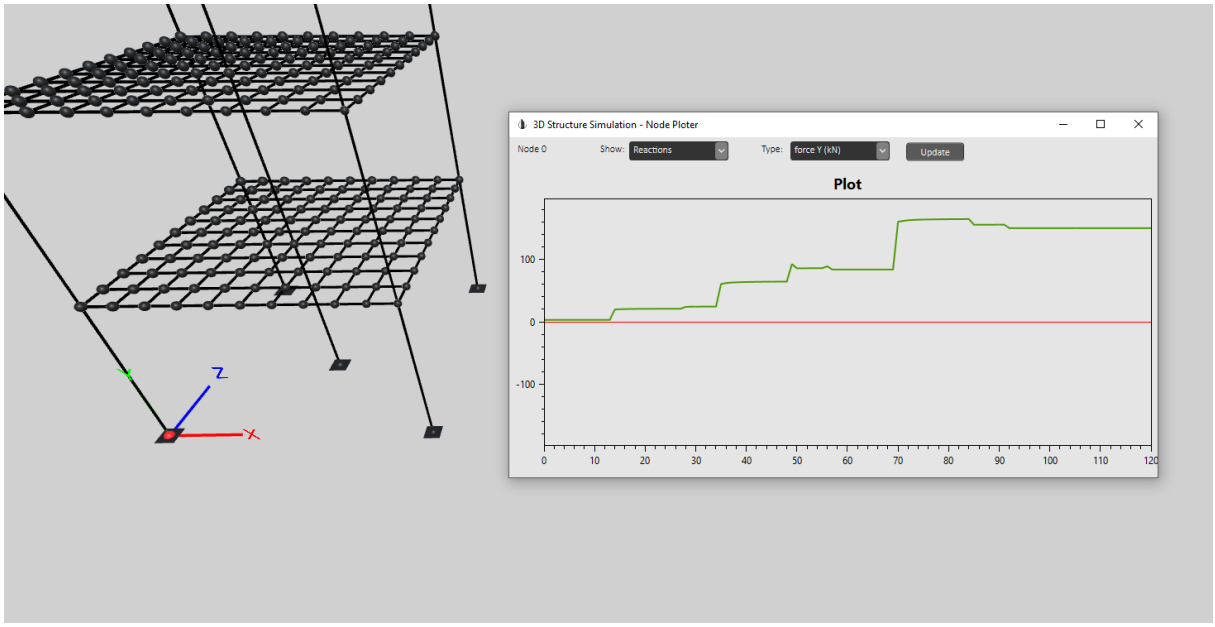
FIGURA 6.16 – Resultados da análise estrutural: Evolução do momento em X em kNm do nó da base do pilar central de 0 à 120 dias



Fonte: Próprio autor (2019)

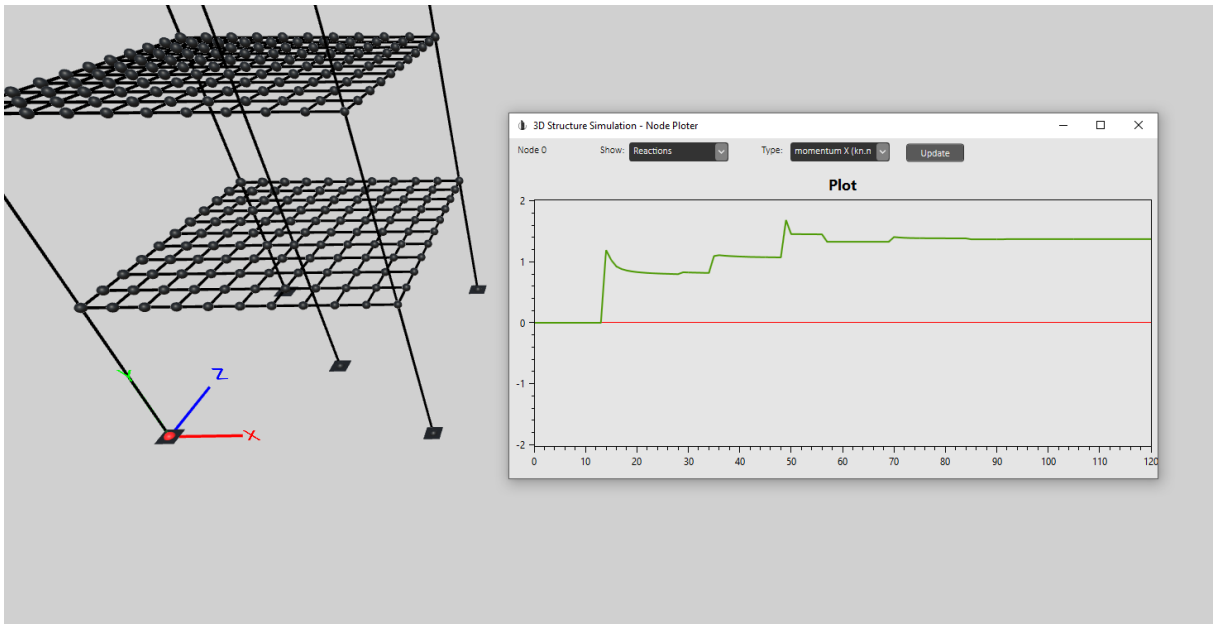
Verifica-se, na FIGURA 6.17, que o maior valor da força em Y na base do pilar de canto não ocorre ao término da construção da estrutura, e sim durante o seu processo construtivo. O mesmo ocorre com os valores do momento, conforme FIGURA 6.18.

FIGURA 6.17 - Resultados da análise estrutural: Evolução da força em Y em kN no nó da base de um pilar de canto de 0 à 120 dias



Fonte: Próprio autor (2019)

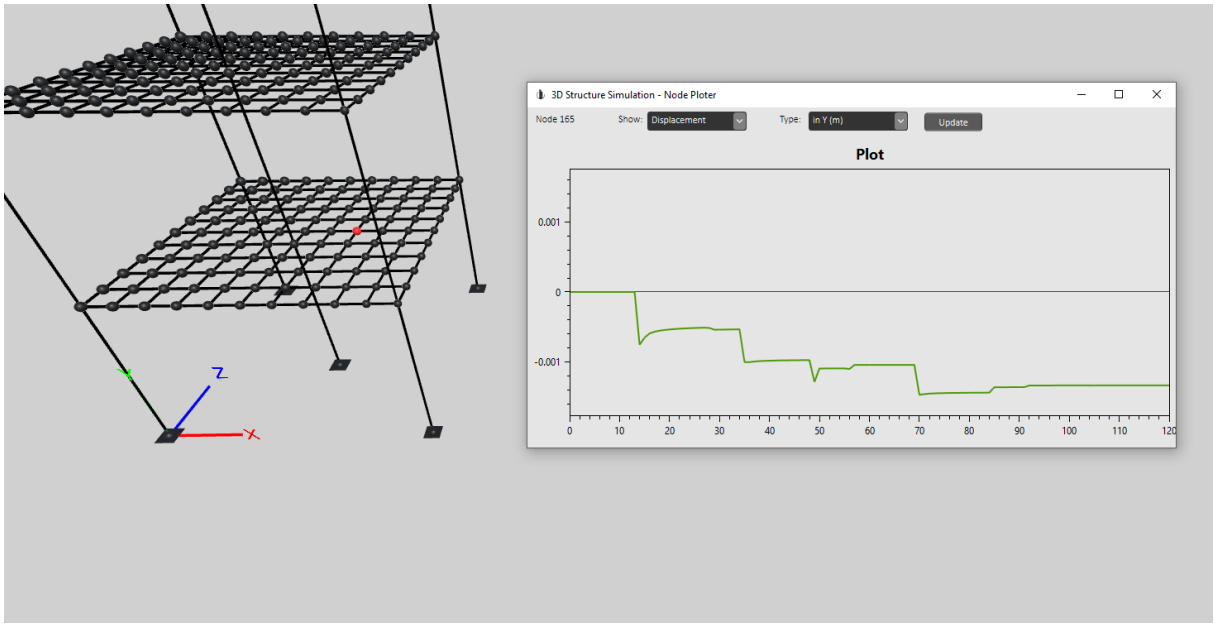
FIGURA 6.18 - Resultados da análise estrutural: Evolução do momento em X em kNm no nó da base de um pilar de canto de 0 à 120 dias



Fonte: Próprio autor (2019)

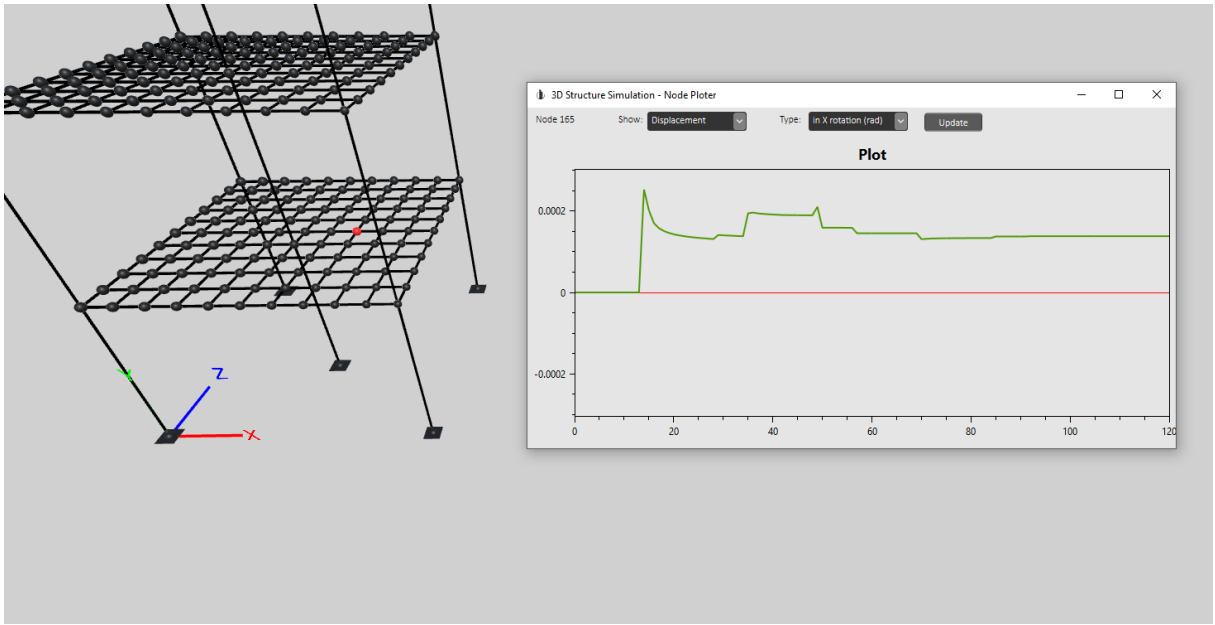
A variação dos deslocamentos dos nós pertencentes à laje pode ser visto nas FIGURA 6.19, FIGURA 6.20 e FIGURA 6.21, onde percebe-se a evolução gradual do deslocamento do nó, e pode-se visualizar o efeito do aumento do módulo de elasticidade do concreto ao longo do tempo mais proeminentemente.

FIGURA 6.19 - Resultados da análise estrutural: Evolução do deslocamento em Y em m em um nó da laje do piso 1 de 0 à 120 dias



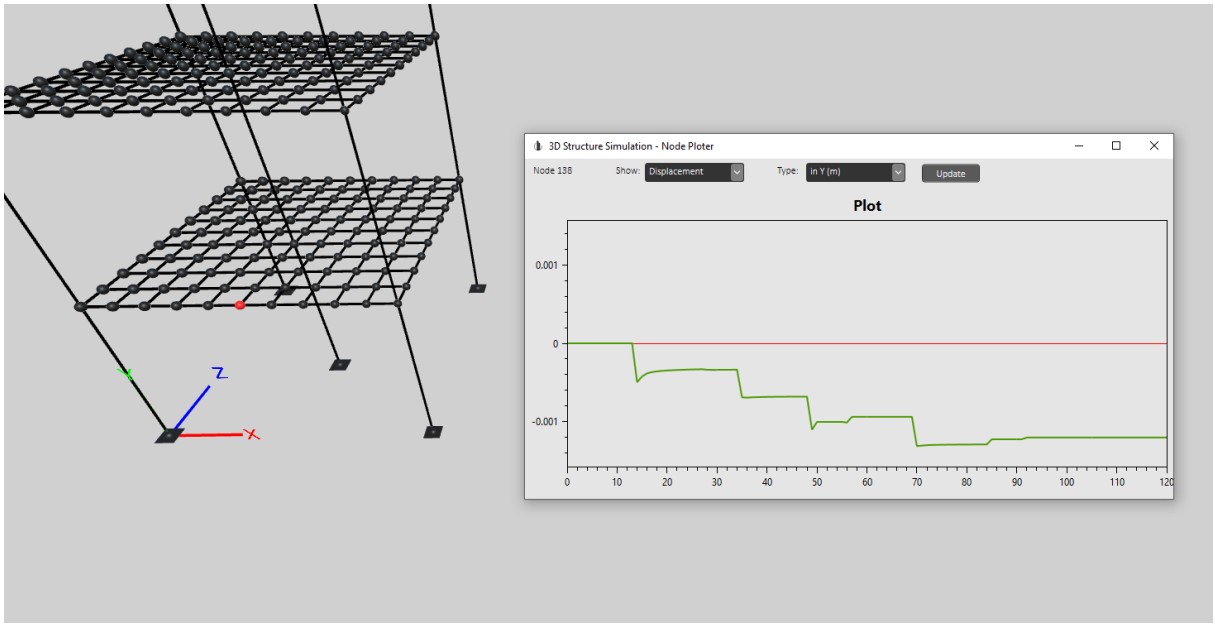
Fonte: Próprio autor (2019)

FIGURA 6.20 - Resultados da análise estrutural: Evolução da rotação em X em rad em um nó da laje do piso 1 de 0 à 120 dias



Fonte: Próprio autor (2019)

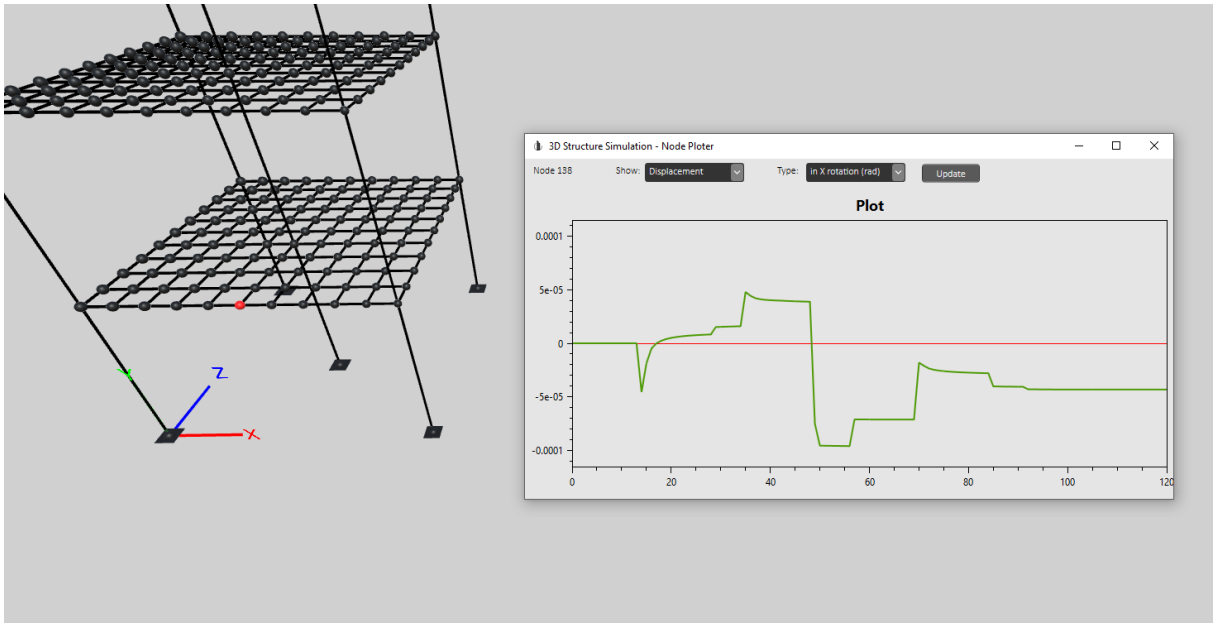
FIGURA 6.21 - Resultados da análise estrutural: Evolução do deslocamento em Y em m em um nó de uma viga no piso 1 de 0 à 120 dias



Fonte: Próprio autor (2019)

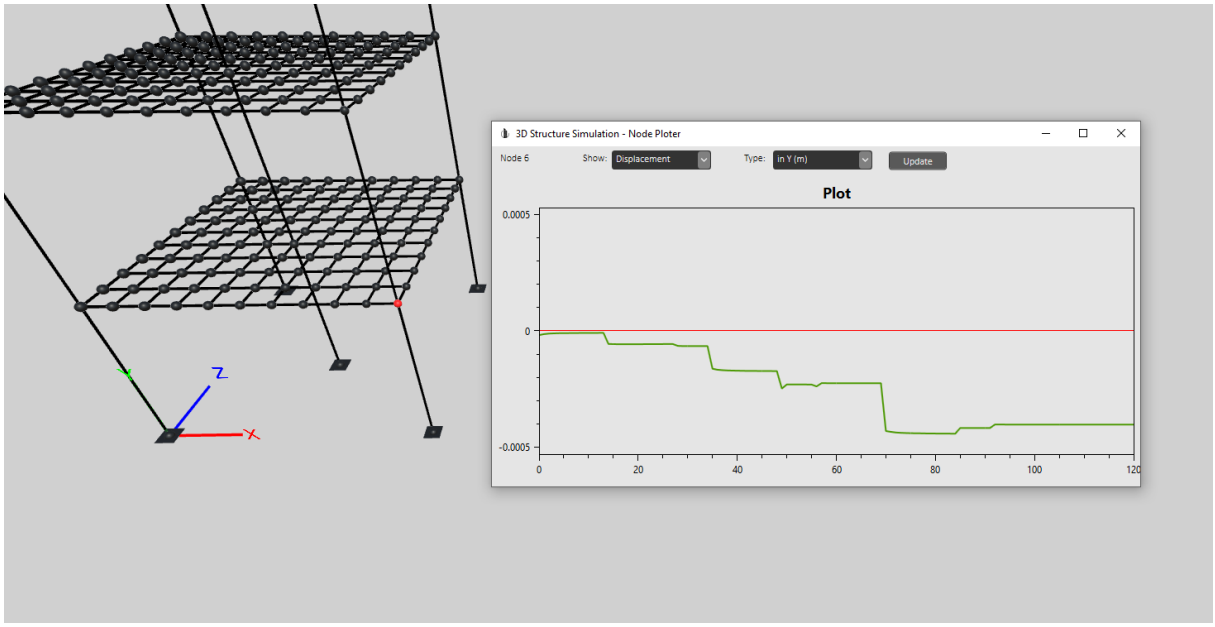
A FIGURA 6.22 mostra um dos nós da laje do primeiro andar, onde o mesmo sofre inversão no sentido da rotação resultante dos deslocamentos.

FIGURA 6.22 - Resultados da análise estrutural: Evolução da rotação em X em rad em um nó de uma viga do piso 1 de 0 à 120 dias



Fonte: Próprio autor (2019)

FIGURA 6.23 - Resultados da análise estrutural: Evolução do deslocamento em Y em m em um nó de um pilar no piso 1 de 0 à 120 dias



Fonte: Próprio autor (2019)

6.4. Validação 1

Dados da validação de barra:

Material:

$$E = 21000000 \, {kN}/{m^2}$$

$$v = 0.25$$

Seção:

$$A = 0.04 m^2$$

$$I_x = 0.0001333 m^4$$

$$I_y = 0.0001333 m^4$$

$$J = 0.0002666 m^4$$

Altura:

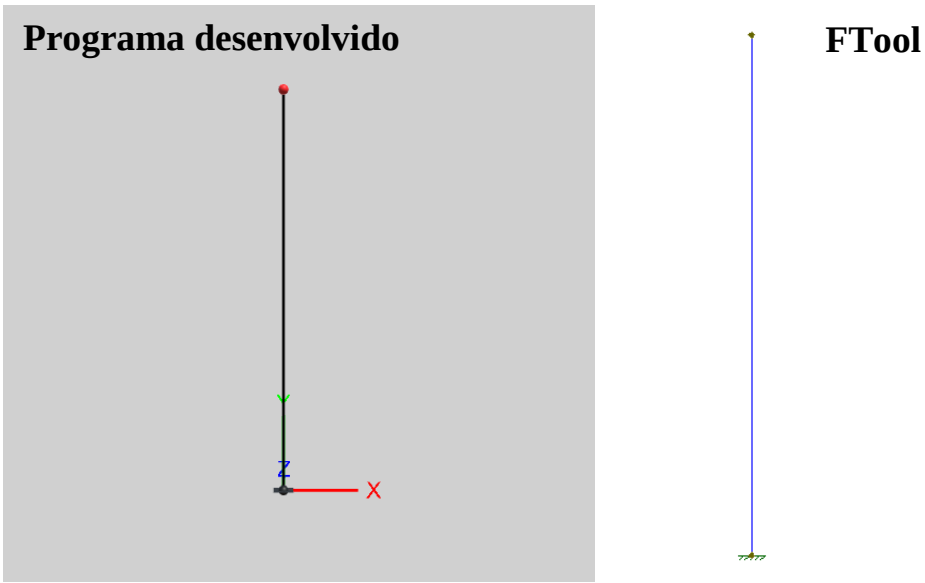
$$altura = 10 m$$

Forças:

$$F_x = 1,0 kN$$

$$M_z = 1,0 kN.m$$

FIGURA 6.24 – Modelagem do exemplo



Fonte: Próprio autor (2019)

Resultados:

Conforme pode ser visto no QUADRO 6.1, os resultados da simulação estrutural feita com o programa desenvolvido neste trabalho, comparado com os resultados obtidos no programa de cálculo estrutural FTOOL são muito próximos, com diferenças dadas devido à diferentes arredondamentos.

QUADRO 6.1 – Comparação de resultados

Resultados	FTOOL	3D Structure Simulation
Nó do topo		
Deslocamento X (m)	0,1012	0.101190476190476
Deslocamento Y (m)	0	0
Rotação em Z (rad)	-0,01429	-0.0142857142857143
Nó da base		
Reação Fx (kN)	-1	-1
Reação Fy (kN)	0	0
Reação Mz (kN.m)	9	9

Fonte: Próprio autor (2019)

6.5. Validação 2

Uma segunda validação também foi feita com um pórtico, com as propriedades a seguir, e os resultados dados no QUADRO 6.2.

Dados da validação:

Material:

$$E = 21000000\,{kN}/{m^2}$$

$$v = 0.25$$

Seção:

$$A = 0.04m^2$$

$$I_x = 0.0001333m^4$$

$$I_y = 0.0001333m^4$$

$$J = 0.0002666m^4$$

Altura:

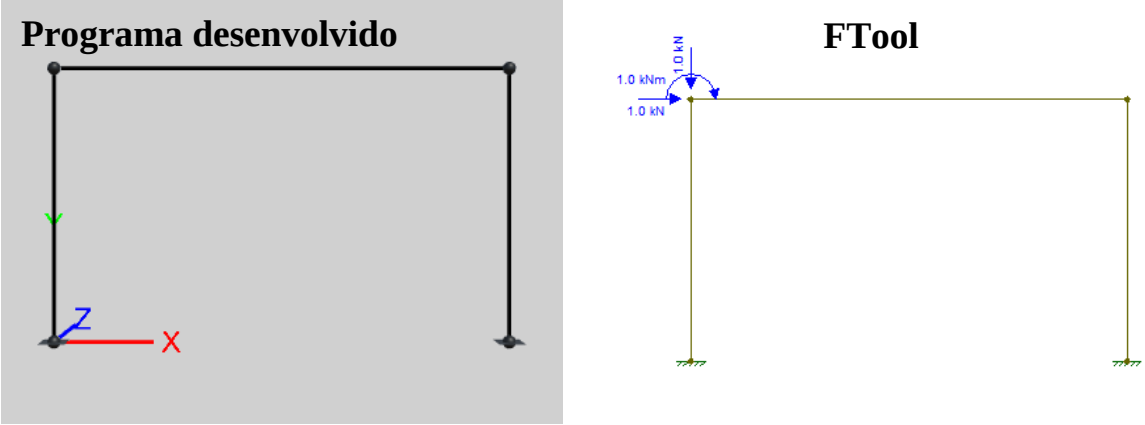
$$altura = 3m$$

$$comprimento = 5m$$

Forças:

$$F_x = 1,0kN$$
$$F_y = -1,0kN$$
$$M_z = -1,0kN.m$$

FIGURA 6.25 - Modelagem do exemplo 2



Fonte: Próprio autor (2019)

Resultados:

Conforme pode ser visto no QUADRO 6.2, os resultados novamente são similares.

QUADRO 6.2 – Comparação de resultados

Resultados	FTOOL	3D Structure Simulation
Nó do topo (onde está aplicada a força)		
Deslocamento X (m)	0,0008404	0,000840384324190976
Deslocamento Y (m)	-2,174E-06	-2,17375108029445E-06
Rotação em Z (rad)	-0,0003948	-0,000394840810881166
Nó da base (embaixo de onde está sendo aplicada a força)		
Reação Fx (kN)	-0,3	-0,308775423126147
Reação Fy (kN)	0,6	0,608650302482446
Reação Mz (kN.m)	0,8	0,831681224844976

Fonte: Próprio autor (2019)

Conforme pode-se verificar, os resultados obtidos com o programa FTOOL e com o programa desenvolvido neste trabalho são muito próximos, confirmando assim a funcionalidade do módulo de cálculo desenvolvido neste trabalho, e permitindo utilizá-lo para as análises mais complexas que são permitidas com o programa.

7. DISCUSSÃO DOS RESULTADOS

São necessárias demasiadas simulações para implementar manualmente o algoritmo neste trabalho exposto. Portanto, para realizar uma simulação adequada, foi preciso o desenvolvimento do programa de computador com a capacidade de considerar de forma automatizada a evolução temporal da estrutura. A discretização da estrutura – composta por: fórmulas adequadas para o cálculo da resistência do concreto a cada dia, módulo de elasticidade, modelo tridimensional da estrutura disposto de acordo com a evolução construtiva, disposição do cimbramento e retirada dele pelas etapas da construção – torna este processo inviável de ser realizado tempestivamente em outros programas de análise estrutural, exceto se desenvolvido especificamente para isso.

O programa aqui apresentado implementa o algoritmo que propusemos. Considerando o processo construtivo nas análises, verificamos que há uma variação na distribuição das ações nos pilares da estrutura. A simulação do processo construtivo altera a distribuição de cargas nos apoios, principalmente por conta da variação geométrica que ocorre na estrutura durante sua construção, mas também devido a variações de cargas dadas pela adição e retirada dos cimbramentos.

Conforme Ferreira (2017), nas análises que consideram o processo evolutivo, é importante considerar os efeitos do sistema de escoramento. Caso, neste tipo de análise, não seja modelado o cimbramento, é possível simular o mesmo pelo método de Grundy e Kabaila, conforme demonstrado por Ferreira (2017). A simplificação de não considerar o sistema de escoramento resulta em uma supervalorização do efeito da construção gradual na análise, apresentando diferenças maiores do que as existentes entre a análise com a estrutura pronta e a análise utilizando o método de Grundy e Kabaila.

8. CONSIDERAÇÕES FINAIS E CONCLUSÕES

Conforme já demostrado por Freitas (2004), em situações usuais, o cálculo previsto em norma garante a segurança estrutural durante e após as etapas construtivas. Porém, com o advento de estruturas de diferentes tipologias e de novas técnicas na execução das estruturas, faz-se necessária uma análise mais cuidadosa das ações introduzidas pelas etapas construtivas, combinada às verificações de segurança durante sua construção e utilização.

A possibilidade de se visualizar e analisar o comportamento e deslocamento da estrutura ao longo de sua construção e utilização, através do algoritmo explorado neste trabalho, abre caminho para otimizações estruturais, tais como: melhor escolha de seções transversais nas vigas e pilares; menores deformações resultantes da fissuração do concreto; melhor distribuição do sistema de cimbramento; entre outras. Além disso, como afirma Vieira (2008), a redução no modulo de elasticidade do concreto, quando do carregamento precoce, pode chegar a até 35% do mesmo. Como consequência, a utilização de valores mais condizentes com a realidade resultará em estruturas mais confiáveis e obras mais seguras, bem como na possível obtenção de tempos reduzidos de execução da obra.

A implementação do algoritmo que propusemos aqui envolveu a criação de um programa de cálculo estrutural que, desde sua concepção, levou em consideração variáveis temporais e as características do concreto. Neste programa, acoplamos uma interface gráfica que facilita a visualização dos cálculos realizados pelo algoritmo, aproximando sua aplicação do usuário final.

Para o dimensionamento de estruturas usuais de concreto armado, costuma-se utilizar a construção finalizada no modelo estrutural, com carregamentos definidos de acordo com sua utilização. Porém, quando analisamos a variação dos esforços atuantes na estrutura durante as etapas construtivas, encontramos valores para as reações de apoio maiores durante uma das etapas, quando comparadas à análise com a estrutura pronta. Conforme exige a ABNT NBR 6118:2014, é necessário garantir a segurança da estrutura em todas as suas etapas. Dessa maneira, considerando que, durante o processo construtivo, ocorrem – em certos locais da estrutura – reações maiores do que normalmente ocorreriam com a mesma pronta, é necessário realizar uma análise incremental, levando em conta o processo construtivo.

Com a utilização do programa desenvolvido neste trabalho, parte do esforço de modelagem de uma estrutura complexa em todas as etapas de construção não existe mais. As propriedades dos materiais são calculadas de forma automatizada ao longo do progresso da

estrutura (algo que o programa SAP2000® já realiza), mas, pela primeira vez, temos um programa de análise estrutural modificando massivamente a geometria da estrutura ao longo do tempo, permitindo assim uma análise mais completa e mais realista da evolução da estrutura, obtendo resultados de reações e deslocamentos diários durante a construção da mesma.

9. REFERÊNCIAS BIBLIOGRÁFICAS E BIBLIOGRAFIA

ACI COMMITTEE 318 (1989). **Building Code Requirements for reinforced concrete**, Detroit.

ASSAN, A. E. **Métodos dos elementos finitos: primeiros passos.** Editora da Unicamp. Campinas. 1999.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (2003). **NBR 6118 - Projeto e Execução de Obras de Concreto Armado**. ABNT, Rio de Janeiro. 2003.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (2014). **NBR 6118 - Projeto e Execução de Obras de Concreto Armado**. ABNT, Rio de Janeiro. 2014.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (2004). **NBR 14931 – Execução de estruturas de concreto - Procedimento**. ABNT, Rio de Janeiro. 2014.

AZEVEDO, A. F. M. **Análise não Linear de Estruturas Planas de Betão Armado**. 1985. 186 f. Dissertação (Mestrado em Engenharia Civil), Faculdade de Engenharia Universidade do Porto, Portugal, 1985

BATHE, K. J. **Finite Element Procedures.** Prentice-Hall, Englewood Cliffs, New Jersey. 1996.

BRANSON, D. E., **Procedures for computing deflections**. ACI Journal: New York. 1968.

CARVALHO, R. C. **Análise não-linear de pavimentos de concreto através da analogia de grelha.** EESC, USP. São Carlos, 1994.

CARVALHO, R. C, FIGUEIREDO FILHO, J. R. **Cálculo e detalhamento de estruturas de concreto armado.** 3 ed. São Carlos: EDUFSCAR. 2005.

CEB-FIP Model Code for Concrete structures (1990). **Evaluation of the time dependent behaviour of concrete**. Bulletin d'information No199, Lausanne,201 p.

COELHO, A. O. L. **Influência das etapas de construção e carregamento na análise de pórticos planos em edifícios altos.** Dissertação (Mestrado em Engenharia de Estruturas). UFMG. 2003.

COTTA, I. F. S., **Desenvolvimento de programa livre para análise de pórticos tridimensionais considerando-se a não linearidade geométrica, fissuração do concreto e ligações semi-rígidas**. UFSCar, São Carlos. 2007.

FERREIRA, W. E. R; **Influência das etapas de construção na análise de estruturas de concreto armado**, UFSC, Santa Catarina. 2017.

FONTES, F. F. **Análise estrutural de elementos lineares segundo a NBR 6118:2003.** EESC, USP. 2004.

FRETAS, A. A., **Situações críticas no projeto de edifícios de concreto armado submetidos a ações de construção**. USP, São Carlos. 2004.

GERE, J. M.; WEAVER JR, W. **Análise de estruturas reticuladas.** Rio de Janeiro, Guanabara. 1965.

GARDNER, N.J.; ZHAO, J.W. (1993). **Creep and shrinkage revisited. ACI Materials Journal**, v.90, No 3, May-June, pp. 236-246.

GUSYATNIKOV, P; REZNICHENKO, S. **Vector Álgebra.** Moscow: Mir Publishers Moscow. 1988.

KATAOKA, M. N. **Estudo da continuidade em ligações laje-viga-pilar em estruturas pré-moldadas de concreto.** UFSCar, São Carlos. 2007.

KAEFER, L. F. **Desenvolvimento de uma ferramenta gráfica para análise de pórticos de concreto armado.** EPSP, USP. São Paulo. 2000.

PRADO, J. F. M. A; **Estruturas de edifícios em concreto armado submetidas a ações de construção**. UFSCar, São Carlos. 1999.

PRADO, J. F. M. A.; CORRÊA, M. R. S. **Estrutura de edifícios em concreto armado submetidas a ações de construção**. Cadernos de Engenharia de Estruturas. São Carlos. 2002.

SALVADOR, P. F. **Influência dos ciclos de execução nas deformações das estruturas de concreto armado de edifícios de andares múltiplos**. UFRGS, Rio Grande do Sul. 2007.

SALVADOR, P. F. **Investigação teórica e experimental da transferência de cargas entre pavimentos de concreto escorados**. UFRGS, Rio Grande do Sul. 2013.

SANTOS, H. C. **Análise de estruturas de concreto sob o efeito do tempo: Uma abordagem consistente com consideração da viscoelasticidade, da plasticidade, da fissuração, da protensão e de etapas construtivas**. Tese (Doutorado em Engenharia Civil). USP, São Paulo. 2006.

VIEIRA, G. L. **Influência da microfissuração causada por carregamentos precoces nas propriedades mecânicas de concretos produzidos com diferentes tipos de cimento**. UFRGS, Rio Grande do Sul. 2008.

APÊNDICE A – CÓDIGO DA INTERFACE E SIMULAÇÃO ESTRUTURAL

Código de gerenciamento da interface:

```
1. using System;
2. using System.Collections.Generic;
3. using System.Diagnostics;
4. using System.Linq;
5. using System.Runtime.InteropServices;
6. using System.Text;
7. using System.Threading.Tasks;
8. using System.Windows;
9. using System.Windows.Controls;
10.  using System.Windows.Data;
11.  using System.Windows.Documents;
12.  using System.Windows.Input;
13.  using System.Windows.Media;
14.  using System.Windows.Media.Imaging;
15.  using System.Windows.Navigation;
16.  using System.Windows.Shapes;
17.  using System.Windows.Forms;
18.  using System.Threading;
19.  using System.Net;
20.  using System.Net.Sockets;
21.  using System.Windows.Interop;
22.  using System.Windows.Threading;
23.  using BriefFiniteElementNet;
24.  using System.Runtime.Serialization.Formatters.Binary;
25.  using System.IO;
26.
27.  namespace ProjectTango
28.  {
29.      [System.Serializable]
30.      public class Vector3
31.      {
32.          public double x;
33.          public double y;
34.          public double z;
35.
36.          public Vector3()
37.          {
38.              x = 0;
39.              y = 0;
40.              z = 0;
41.          }
42.
43.          public Vector3(string x, string y, string z)
44.          {
45.              this.x = double.Parse(x);
46.              this.y = double.Parse(y);
47.              this.z = double.Parse(z);
48.          }
49.
50.          public Vector3(double x, double y, double z)
51.          {
52.              this.x = x;
```

```
53.             this.y = y;
54.             this.z = z;
55.         }
56.     }
57.
58.     [System.Serializable]
59.     public class Restrictions
60.     {
61.         public bool x1;
62.         public bool y1;
63.         public bool z1;
64.         public bool x2;
65.         public bool y2;
66.         public bool z2;
67.
68.         public Restrictions()
69.         {
70.             x1 = false;
71.             y1 = false;
72.             z1 = false;
73.             x2 = false;
74.             y2 = false;
75.             z2 = false;
76.         }
77.
78.         public Restrictions(bool x1, bool y1, bool z1, bool x2,
   bool y2, bool z2)
79.         {
80.             this.x1 = x1;
81.             this.y1 = y1;
82.             this.z1 = z1;
83.             this.x2 = x2;
84.             this.y2 = y2;
85.             this.z2 = z2;
86.         }
87.     }
88.
89.     [System.Serializable]
90.     public class Material
91.     {
92.         public string name;
93.
94.         public double E;
95.         public double G;
96.         public double v;
97.
98.         public bool timeDependent;
99.
100.        public double fck;
101.
102.        public Material()
103.        {
104.            name = "";
105.
106.            E = 0;
107.            G = 0;
108.            v = 0;
109.
110.            timeDependent = false;
111.
112.            fck = 0;
```

```
113.         }
114.
115.         public Material(string name, string E, string G, bool time,
   string fck = "0", string v = "0")
116.         {
117.             this.name = name;
118.
119.             this.E = double.Parse(E);
120.             this.G = double.Parse(G);
121.             this.v = double.Parse(v);
122.
123.             this.timeDependent = time;
124.
125.             this.fck = double.Parse(fck);
126.         }
127.     }
128.
129.     [System.Serializable]
130.     public class Property
131.     {
132.         public string name;
133.
134.         public double A;
135.         public double Ix;
136.         public double Iy;
137.         public double Iz;
138.
139.         public string type;
140.
141.         public double d;
142.
143.         public double h;
144.         public double b;
145.
146.         public Property(string name, string A, string Ix, string
   Iy, string Iz, string type, string d = "0", string h = "0", string b
   = "0")
147.         {
148.             if (d == "") d = "0";
149.             if (b == "") b = "0";
150.             if (h == "") h = "0";
151.
152.             if (type == "0") type = "None";
153.             if (type == "1") type = "Circular";
154.             if (type == "2") type = "Retangular";
155.
156.             this.name = name;
157.
158.             this.A = double.Parse(A);
159.             this.Ix = double.Parse(Ix);
160.             this.Iy = double.Parse(Iy);
161.             this.Iz = double.Parse(Iz);
162.
163.             this.type = type;
164.
165.             this.d = double.Parse(d);
166.             this.h = double.Parse(h);
167.             this.b = double.Parse(b);
168.         }
169.     }
170.
```

```
171.     [System.Serializable]
172.     public class Member
173.     {
174.         public int number;
175.
176.         public int n0;
177.         public int n1;
178.
179.         public string material;
180.         public string section;
181.
182.         public int startDay;
183.         public int endDay;
184.
185.         public Member(int number, int n0, int n1, string material,
    string section, string startDay, string endDay)
186.         {
187.             this.number = number;
188.
189.             this.n0 = n0;
190.             this.n1 = n1;
191.             this.material = material;
192.             this.section = section;
193.
194.             this.startDay = int.Parse(startDay);
195.             this.endDay = int.Parse(endDay);
196.         }
197.     }
198.
199.     [System.Serializable]
200.     public class Nodes
201.     {
202.         public int number;
203.
204.         public Vector3 position;
205.         public Restrictions restriction;
206.
207.         public Nodes()
208.         {
209.             position = new Vector3();
210.             restriction = new Restrictions();
211.         }
212.
213.         public Nodes(int number, string x, string y, string z,
    bool? x1, bool? y1, bool? z1, bool? x2, bool? y2, bool? z2)
214.         {
215.             this.number = number;
216.
217.             position = new Vector3(x, y, z);
218.             restriction = new Restrictions(x1.GetValueOrDefault(),
    y1.GetValueOrDefault(), z1.GetValueOrDefault(),
    x2.GetValueOrDefault(), y2.GetValueOrDefault(),
    z2.GetValueOrDefault());
219.         }
220.     }
221.
222.     [System.Serializable]
223.     public class NodalForces
224.     {
225.         public string name;
226.
```

```
227.         public Vector3 disp;
228.         public Vector3 rott;
229.
230.         public List<int> nodes;
231.
232.         public int startDay;
233.         public int endDay;
234.
235.         public NodalForces()
236.         {
237.             name = "";
238.
239.             disp = new Vector3();
240.             rott = new Vector3();
241.
242.             nodes = new List<int>();
243.
244.             startDay = 0;
245.             endDay = 0;
246.         }
247.
248.         public NodalForces(string name, double dx, double dy,
    double dz, double rx, double ry, double rz, int startDay, int endDay,
    List<int> list)
249.         {
250.             this.name = name;
251.
252.             disp = new Vector3(dx, dy, dz);
253.             rott = new Vector3(rz, ry, rz);
254.
255.             nodes = list;
256.
257.             this.startDay = startDay;
258.             this.endDay = endDay;
259.         }
260.     }
261.
262.     [System.Serializable]
263.     public class MemberForces
264.     {
265.         public string name;
266.
267.         public Vector3 disp;
268.
269.         public List<int> members;
270.
271.         public int startDay;
272.         public int endDay;
273.
274.         public MemberForces()
275.         {
276.             name = "";
277.
278.             disp = new Vector3();
279.
280.             members = new List<int>();
281.
282.             startDay = 0;
283.             endDay = 0;
284.         }
285.
```

```
286.         public MemberForces(string name, double dx, double dy,
   double dz, int startDay, int endDay, List<int> list)
287.         {
288.             this.name = name;
289.
290.             disp = new Vector3(dx, dy, dz);
291.
292.             members = list;
293.
294.             this.startDay = startDay;
295.             this.endDay = endDay;
296.         }
297.     }
298.
299.     /// <summary>
300.     /// Interaction logic for MainWindow.xaml
301.     /// </summary>
302.     public partial class MainWindow : Window
303.     {
304.         public static MainWindow instance;
305.
306.         // Create Form Panel to use as host for Unity 3D Window
307.         private System.Windows.Forms.Panel _panel;
308.         // Create a Process to call Unity 3D Window
309.         private Process _process;
310.         private IntPtr _unityHWND = IntPtr.Zero;
311.
312.         [DllImport("user32.dll")]
313.         private static extern int SetWindowLong(IntPtr hWnd, int
   nIndex, int dwNewLong);
314.
315.         [DllImport("user32.dll", SetLastError = true)]
316.         private static extern int GetWindowLong(IntPtr hWnd, int
   nIndex);
317.
318.         [DllImport("user32")]
319.         private static extern IntPtr SetParent(IntPtr hWnd, IntPtr
   hWndParent);
320.
321.         [DllImport("user32")]
322.         private static extern bool SetWindowPos(IntPtr hWnd, IntPtr
   hWndInsertAfter, int X, int Y, int cx, int cy, int uFlags);
323.
324.         [DllImport("User32.dll")]
325.         static extern bool MoveWindow(IntPtr handle, int x, int y,
   int width, int height, bool redraw);
326.
327.         internal delegate int WindowEnumProc(IntPtr hwnd, IntPtr
   lparam);
328.         [DllImport("user32.dll")]
329.         internal static extern bool EnumChildWindows(IntPtr hwnd,
   WindowEnumProc func, IntPtr lParam);
330.
331.         [DllImport("user32.dll")]
332.         static extern int SendMessage(IntPtr hWnd, int msg, IntPtr
   wParam, IntPtr lParam);
333.
334.         private const int WM_ACTIVATE = 0x0006;
335.         private readonly IntPtr WA_ACTIVE = new IntPtr(1);
336.         private readonly IntPtr WA_INACTIVE = new IntPtr(0);
337.         private bool finishedLoading = false;
```

```
338.
339.         Socket s;
340.         TcpListener myList;
341.         public static string threadCommand = "";
342.         public int tempMemberN0;
343.         public int tempMemberN1;
344.
345.         //Analysis variables.
346.         public List<Nodes> nodes = new List<Nodes>();
347.         public List<Member> members = new List<Member>();
348.
349.         public List<Material> materials = new List<Material>();
350.         public List<Property> properties = new List<Property>();
351.
352.         public List<NodalForces> nodalForces = new
    List<NodalForces>();
353.         public List<MemberForces> memberForces = new
    List<MemberForces>();
354.
355.         public List<BriefFiniteElementNet.Model> models = new
    List<BriefFiniteElementNet.Model>();
356.
357.         // Simulation Variables
358.         public int startDay = 0;
359.         public int endDay = 720;
360.         public bool simulationSuccess = false;
361.         public string lastSimulatedSelected = "";
362.
363.         public MainWindow()
364.         {
365.             // Unhandled exceptions for our Application Domain
366.             AppDomain.CurrentDomain.UnhandledException += new
    System.UnhandledExceptionEventHandler(AppDomain_UnhandledException);
367.
368.             // Unhandled exceptions for the executing UI thread
369.             System.Windows.Forms.Application.ThreadException += new
    System.Threading.ThreadExceptionEventHandler(Application_ThreadExcept
    ion);
370.
371.             instance = this;
372.             //Example3();
373.             //Example2();
374.             //StructuralAnalysisMathNet analysis = new
    StructuralAnalysisMathNet();
375.             InitializeComponent();
376.             InitializeUnity3D();
377.             InitializeSocket();
378.             finishedLoading = true;
379.         }
380.
381.         //private static void Example3()
382.         //{
383.         //    var Analysis = new SpatialFrameAnalysis.Analysis();
384.
385.         //    var material = new MaterialProprieties(21000000000d,
    6000000000d);
386.         //    var proprieties = new GeometricalProprieties(0.04d,
    0.00013333333d, 0.00013333333d, 0.00026666666d);
387.
388.         //    var n0 = new Node(0, 0, 0, 0, Restriction.Fixed);
389.         //    var n1 = new Node(1, 0, 6, 0, Restriction.Released);
```

```
390.
391.         //    Analysis.Nodes.Add(n0);
392.         //    Analysis.Nodes.Add(n1);
393.
394.         //    var m0 = new Member(n0, n1, proprieties, material);
395.
396.         //    Analysis.Members.Add(m0);
397.
398.         //    Analysis.SolveLinear();
399.
400.         //    Console.Write(Analysis.data);
401.         //}
402.
403.         public static bool LineSegmentIntersection(double Ax,
   double Ay, double Bx, double By, double Cx, double Cy, double Dx,
   double Dy, ref double X, ref double Y)
404.         {
405.             double distAB, theCos, theSin, newX, ABpos;
406.
407.             //  Fail if either line segment is zero-length.
408.
409.             if (Ax == Bx && Ay == By || Cx == Dx && Cy == Dy)
   return false;
410.
411.             //  Fail if the segments share an end-point.
412.
413.             if (Ax == Cx && Ay == Cy || Bx == Cx && By == Cy || Ax
   == Dx && Ay == Dy || Bx == Dx && By == Dy)
414.             {
415.                 return false;
416.             }
417.
418.
419.             //  (1) Translate the system so that point A is on the
   origin.
420.
421.             Bx -= Ax; By -= Ay;
422.
423.             Cx -= Ax; Cy -= Ay;
424.
425.             Dx -= Ax; Dy -= Ay;
426.
427.
428.             //  Discover the length of segment A-B.
429.
430.             distAB = Math.Sqrt(Bx * Bx + By * By);
431.
432.
433.             //  (2) Rotate the system so that point B is on the
   positive X axis.
434.
435.             theCos = Bx / distAB;
436.
437.             theSin = By / distAB;
438.
439.             newX = Cx * theCos + Cy * theSin;
440.
441.             Cy = Cy * theCos - Cx * theSin; Cx = newX;
442.
443.             newX = Dx * theCos + Dy * theSin;
444.
```

```
445.             Dy = Dy * theCos - Dx * theSin; Dx = newX;
446.
447.             //  Fail if segment C-D doesn't cross line A-B.
448.
449.             if (Cy < 0 && Dy < 0 || Cy >= 0 && Dy >= 0) return
   false;
450.
451.             //  (3) Discover the position of the intersection point
   along line A-B.
452.
453.             ABpos = Dx + (Cx - Dx) * Dy / (Dy - Cy);
454.
455.             //  Fail if segment C-D crosses line A-B outside of
   segment A-B.
456.
457.             if (ABpos < 0 || ABpos > distAB) return false;
458.
459.             //  (4) Apply the discovered position to line A-B in
   the original coordinate system.
460.
461.             X = (Ax + ABpos * theCos);
462.
463.             Y = (Ay + ABpos * theSin);
464.
465.
466.             //  Success.
467.
468.             return true;
469.         }
470.
471.         public double GetElasticityModulusABNT(double fck, double
   t, double alphae = 1f, double s = 0.25f)
472.         {
473.             if (t < 1) t = 0.5f;
474.
475.             double beta1 = Math.Pow(Math.E, (s * (1 - Math.Sqrt(28
   / t))));
476.
477.             double fckj = beta1 * fck;
478.
479.             if (t > 28) fckj = fck;
480.
481.             double Eci = 0;
482.             double Ecit = 0;
483.
484.             if (fck >= 20 && fck <= 45)
485.             {
486.                 Eci = alphae * 5600 * Math.Sqrt(fck);
487.                 Ecit = Math.Pow((fckj / fck), 0.5) * Eci;
488.             }
489.             if (fck >= 50 && fck <= 90)
490.             {
491.                 Eci = 21.5 * Math.Pow(10, 3) * alphae *
   Math.Pow((fck / 10) + 1.25, 1 / 3);
492.                 Ecit = Math.Pow((fckj / fck), 0.3) * Eci;
493.             }
494.
495.             // Conversion from MPa to kN/m2;
496.             Ecit = Ecit * 1000;
497.
498.             return Ecit;
```

```
499.         }
500.
501.         public double CalculateG(double E, double v)
502.         {
503.             double G = E / (2 * (1 + v));
504.
505.             return G;
506.         }
507.
508.         private void MountModels()
509.         {
510.             try
511.             {
512.                 startDay = int.Parse(simulationDayStart.Text);
513.                 endDay = int.Parse(simulationDayEnd.Text);
514.
515.                 models.Clear();
516.
517.                 for (int i = startDay; i <= endDay; i++)
518.                 {
519.                     var model = new BriefFiniteElementNet.Model();
520.
521.                     var acceptedNodes = new List<int>();
522.
523.                     foreach (Member member in members)
524.                     {
525.                         if (i >= member.startDay)
526.                         {
527.                             if (member.endDay >= i)
528.                             {
529.                                 if
    (!acceptedNodes.Contains(member.n0)) acceptedNodes.Add(member.n0);
530.                                 if
    (!acceptedNodes.Contains(member.n1)) acceptedNodes.Add(member.n1);
531.                             }
532.                         }
533.                     }
534.
535.                     foreach (int n in acceptedNodes)
536.                     {
537.                         var node = new Node(nodes[n].position.x,
    nodes[n].position.y, nodes[n].position.z);
538.
539.                         if (nodes[n].restriction.x1)
    node.Constraints &= Constraints.FixedDX;
540.                         if (nodes[n].restriction.y1)
    node.Constraints &= Constraints.FixedDY;
541.                         if (nodes[n].restriction.z1)
    node.Constraints &= Constraints.FixedDZ;
542.                         if (nodes[n].restriction.x2)
    node.Constraints &= Constraints.FixedRX;
543.                         if (nodes[n].restriction.y2)
    node.Constraints &= Constraints.FixedRY;
544.                         if (nodes[n].restriction.z2)
    node.Constraints &= Constraints.FixedRZ;
545.
546.                         node.Label = n.ToString();
547.
548.                         model.Nodes.Add(node);
549.                     }
550.
```

```
551.                     foreach (Member member in members)
552.                     {
553.                         if (i >= member.startDay)
554.                         {
555.                             if (member.endDay >= i)
556.                             {
557.                                 var mem = new
    FrameElement2Node(model.Nodes.Where(item => item.Label ==
    member.n0.ToString()).First(), model.Nodes.Where(item => item.Label
    == member.n1.ToString()).First());
558.
559.                                 mem.Label = "M" +
    member.number.ToString();
560.
561.                                 if (materials.Where(item =>
    item.name == member.material).First().timeDependent)
562.                                 {
563.                                     mem.E =
    GetElasticityModulusABNT(materials.Where(item => item.name ==
    member.material).First().fck / 1000, i - member.startDay);
564.                                     mem.G = CalculateG(mem.E,
    materials.Where(item => item.name == member.material).First().v);
565.                                 }
566.                                 else
567.                                 {
568.                                     mem.E = materials.Where(item =>
    item.name == member.material).First().E;
569.                                     mem.G = materials.Where(item =>
    item.name == member.material).First().G;
570.                                 }
571.
572.                                 mem.A = properties.Where(item =>
    item.name == member.section).First().A;
573.                                 mem.Iy = properties.Where(item =>
    item.name == member.section).First().Ix;
574.                                 mem.Iz = properties.Where(item =>
    item.name == member.section).First().Iy;
575.                                 mem.J = properties.Where(item =>
    item.name == member.section).First().Iz;
576.
577.                                 mem.UseOverridedProperties = true;
578.
579.                                 model.Elements.Add(mem);
580.                             }
581.                         }
582.                     }
583.
584.                     foreach (NodalForces force in nodalForces)
585.                     {
586.                         if (i >= force.startDay)
587.                         {
588.                             if (force.endDay >= i)
589.                             {
590.                                 foreach (int node in force.nodes)
591.                                 {
592.                                     if (model.Nodes.Where(item =>
    item.Label == node.ToString()).ToList().Count > 0)
593.                                     {
594.                                         var f = new
    Force(force.disp.x, force.disp.y, force.disp.z, force.rott.x,
    force.rott.y, force.rott.z);
```

```
595.                                                 model.Nodes.Where(item =>
    item.Label == node.ToString()).First().Loads.Add(new NodalLoad(f));
596.                                             }
597.                                         }
598.                                     }
599.                                 }
600.                             }
601.
602.                             foreach (MemberForces force in memberForces)
603.                             {
604.                                 if (i >= force.startDay)
605.                                 {
606.                                     if (force.endDay >= i)
607.                                     {
608.                                         foreach (int member in
    force.members)
609.                                         {
610.                                             if (model.Elements.Where(item
    => item.Label == "M" + member.ToString()).ToList().Count > 0)
611.                                             {
612.                                                 if (force.disp.x != 0)
613.                                                 {
614.                                                     var load = new
    UniformLoad1D(force.disp.x, LoadDirection.X,
    CoordinationSystem.Global);
615.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
616.                                                 }
617.                                                 if (force.disp.y != 0)
618.                                                 {
619.                                                     var load = new
    UniformLoad1D(force.disp.y, LoadDirection.Y,
    CoordinationSystem.Global);
620.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
621.                                                 }
622.                                                 if (force.disp.z != 0)
623.                                                 {
624.                                                     var load = new
    UniformLoad1D(force.disp.z, LoadDirection.Z,
    CoordinationSystem.Global);
625.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
626.                                                 }
627.                                             }
628.                                         }
629.
630.                                     }
631.                                 }
632.                             }
633.
634.                             models.Add(model);
635.                         }
636.                     }
637.                     catch
638.                     {
639.
640.                     }
```

```
641.         }
642.
643.         private void RunSimulations()
644.         {
645.             Parallel.ForEach(models, model =>
646.             {
647.                 if (model.Nodes.Count > 0 && model.Elements.Count >
    0)
648.                 {
649.                     model.Solve();
650.                 }
651.             });
652.         }
653.
654.         //private static void Example2()
655.         //{
656.         //    Console.WriteLine("Example 1: Simple 3D Frame with
    distributed loads");
657.
658.         //    var model = new BriefFiniteElementNet.Model();
659.
660.         //    var n1 = new Node(-10, 0, 0);
661.         //    var n2 = new Node(-10, 0, 6);
662.         //    var n3 = new Node(0, 0, 8);
663.         //    var n4 = new Node(10, 0, 6);
664.         //    var n5 = new Node(10, 0, 0);
665.         //    var n6 = new Node(20, 0, 0);
666.
667.         //    model.Nodes.Add(n1, n2, n3, n4, n5);
668.
669.         //    var secAA =
    SectionGenerator.GetRectangularSection(0.2, 0.2);
670.
671.         //    var b =
    PolygonYz.GetSectionGeometricalProperties(secAA);
672.
673.         //    var e1 = new FrameElement2Node(n1, n2);
674.         //    e1.Label = "e1";
675.         //    var e2 = new FrameElement2Node(n2, n3);
676.         //    e2.Label = "e2";
677.         //    var e3 = new FrameElement2Node(n3, n4);
678.         //    e3.Label = "e3";
679.         //    var e4 = new FrameElement2Node(n4, n5);
680.         //    e4.Label = "e4";
681.
682.         //    e1.Geometry = new PolygonYz(secAA);
683.
684.         //    e1.Geometry = e4.Geometry = e2.Geometry = e3.Geometry
    = new PolygonYz(secAA);
685.
686.         //    e1.E = e2.E = e3.E = e4.E = 210e9;
687.         //    e1.G = e2.G = e3.G = e4.G = 210e9 / (2 * (1 +
    0.3));//G = E / (2*(1+no))
688.
689.         //    e1.UseOverridedProperties =
690.         //        e2.UseOverridedProperties =
    e3.UseOverridedProperties =
691.         //
    e4.UseOverridedProperties = false;
692.
693.         //    model.Elements.Add(e1, e2, e3, e4);
```

```
694.
695.         //    n1.Constraints =
696.         //        n2.Constraints =
697.         //            n3.Constraints =
698.         //                n4.Constraints =
699.         //                    n5.Constraints =
700.         //                        Constraint.FixedDY &
   Constraint.FixedRX &
701.         //                        Constraint.FixedRZ;//DY fixed and
   RX fixed and RZ fixed
702.
703.         //    n1.Constraints = n1.Constraints &
   Constraint.MovementFixed;
704.         //    n5.Constraints = n5.Constraints &
   Constraint.MovementFixed;
705.
706.
707.         //    var ll = new UniformLoad1D(-10000, LoadDirection.Z,
   CoordinationSystem.Global);
708.         //    var lr = new UniformLoad1D(-10000, LoadDirection.Z,
   CoordinationSystem.Local);
709.
710.         //    e2.Loads.Add(ll);
711.         //    e3.Loads.Add(lr);
712.
713.         //    //var wnd = WpfTraceListener.CreateModelTrace(model);
714.         //    new ModelWarningChecker().CheckModel(model);
715.         //    //wnd.ShowDialog();
716.
717.         //    model.Solve();
718.
719.         //    var a = n3.GetNodalDisplacement();
720.
721.         //    var bcc = 0;
722.         //}
723.
724.         //private static void Example1()
725.         //{
726.         //    Console.WriteLine("Example 1: Simple 3D truss with
   four members");
727.
728.         //    // Initiating Model, Nodes and Members
729.         //    var model = new Model();
730.
731.         //    var n1 = new Node(1, 1, 0);
732.         //    n1.Label = "n1";//Set a unique label for node
733.         //    var n2 = new Node(-1, 1, 0) { Label = "n2" };//using
   object initializer for assigning Label
734.         //    var n3 = new Node(1, -1, 0) { Label = "n3" };
735.         //    var n4 = new Node(-1, -1, 0) { Label = "n4" };
736.         //    var n5 = new Node(0, 0, 1) { Label = "n5" };
737.
738.         //    var e1 = new TrussElement2Node(n1, n5) { Label = "e1"
   };
739.         //    var e2 = new TrussElement2Node(n2, n5) { Label = "e2"
   };
740.         //    var e3 = new TrussElement2Node(n3, n5) { Label = "e3"
   };
741.         //    var e4 = new TrussElement2Node(n4, n5) { Label = "e4"
   };
742.
```

```
743.         //    //Note: labels for all members should be unique,
744.         //    //else you will receive InvalidLabelException when
   adding it to model
745.
746.         //    e1.A = e2.A = e3.A = e4.A = 9e-4;
747.         //    e1.E = e2.E = e3.E = e4.E = 210e9;
748.
749.
750.         //    model.Nodes.Add(n1, n2, n3, n4, n5);
751.         //    model.Elements.Add(e1, e2, e3, e4);
752.
753.         //    //Applying restraints
754.
755.         //    n1.Constraints = n2.Constraints = n3.Constraints =
   n4.Constraints = Constraint.Fixed;
756.         //    n5.Constraints = Constraint.FixedRX &
   Constraint.FixedRY & Constraint.FixedRZ;/*
   Constraint.RotationFixed;*/
757.
758.         //    //Applying load
759.         //    var force = new Force(0, 1000, 0, 0, 0, 0);
760.         //    n5.Loads.Add(new NodalLoad(force));//adds a load with
   LoadCase of DefaultLoadCase to node loads
761.
762.         //    //Adds a NodalLoad with Default LoadCase
763.
764.         //    model.Solve();
765.
766.         //    var r1 = n1.GetSupportReaction();
767.         //    var r2 = n2.GetSupportReaction();
768.         //    var r3 = n3.GetSupportReaction();
769.         //    var r4 = n4.GetSupportReaction();
770.
771.         //    var dis = n5.GetNodalDisplacement();
772.
773.         //    var a = e1.GetInternalForceAt(1);
774.
775.         //    var rt = r1 + r2 + r3 + r4;//shows the Fz=1000 and
   Fx=Fy=Mx=My=Mz=0.0
776.
777.         //    Console.WriteLine("Total reactions SUM :" +
   rt.ToString());
778.         //}
779.
780.         private void InitializeSocket()
781.         {
782.             ActivateServer();
783.             CheckMesages();
784.         }
785.
786.         private void ActivateServer()
787.         {
788.             try
789.             {
790.                 IPAddress ipAd = IPAddress.Parse("127.0.0.1");
791.                 // use local m/c IP address, and
792.                 // use the same in the client
793.
794.                 /* Initializes the Listener */
795.                 myList = new TcpListener(ipAd, 8001);
796.
```

```
797.                 /* Start Listeneting at the specified port */
798.                 myList.Start();
799.
800.                 Console.WriteLine("The server is running at port
   8001...");
801.                 Console.WriteLine("The local End point is: " +
   myList.LocalEndpoint);
802.                 Console.WriteLine("Waiting for a connection...");
803.
804.                 s = myList.AcceptSocket();
805.
806.                 Console.WriteLine("Connection accepted from " +
   s.RemoteEndPoint);
807.             }
808.             catch (Exception e)
809.             {
810.                 Console.WriteLine("Error: " + e.StackTrace);
811.             }
812.         }
813.
814.         public void CheckMesages()
815.         {
816.             if (threadCommand != "")
817.             {
818.                 string[] commands = threadCommand.Split('&');
819.
820.                 threadCommand = "";
821.
822.
823.             }
824.
825.             new Thread(() =>
826.             {
827.                 Thread.CurrentThread.IsBackground = true;
828.
829.                 try
830.                 {
831.                     bool threadShouldContinue = true;
832.
833.                     while (threadShouldContinue)
834.                     {
835.
836.
837.                         byte[] b = new byte[131072];
838.
839.                         int k = s.Receive(b);
840.
841.                         string recived = "";
842.
843.                         for (int i = 0; i < k; i++)
844.                         {
845.                             recived += Convert.ToChar(b[i]);
846.                         }
847.
848.                         string[] commands = recived.Split('&');
849.
850.                         if (commands[0] == "No Nodes")
851.                         {
852.                             try
853.                             {
```

```
854.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
855.                                     DispatcherPriority.Normal,
856.                                     (Action)(() =>
857.                                     {
858.                                         xCoordNodeEdit.Text = "0
   node selected";
859.                                         yCoordNodeEdit.Text = "0
   node selected";
860.                                         zCoordNodeEdit.Text = "0
   node selected";
861.
862.                                         xCoordNodeEdit.IsEnabled =
   false;
863.                                         yCoordNodeEdit.IsEnabled =
   false;
864.                                         zCoordNodeEdit.IsEnabled =
   false;
865.                                     }
866.                                     ));
867.                                 }
868.                                 catch
869.                                 {
870.
871.                                 }
872.                             }
873.
874.                             if (commands[0] == "Edit Node")
875.                             {
876.                                 try
877.                                 {
878.                                     // Send message to UI thread
879.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
880.                                     DispatcherPriority.Normal,
881.                                     (Action)(() =>
882.                                         {
883.                                             threadCommand =
   recived;
884.
885.
   xCoordNodeEdit.IsEnabled = true;
886.
   yCoordNodeEdit.IsEnabled = true;
887.
   zCoordNodeEdit.IsEnabled = true;
888.
889.                                             xCoordNodeEdit.Text =
   nodes[int.Parse(commands[1])].position.x.ToString();
890.                                             yCoordNodeEdit.Text =
   nodes[int.Parse(commands[1])].position.y.ToString();
891.                                             zCoordNodeEdit.Text =
   nodes[int.Parse(commands[1])].position.z.ToString();
892.
893.                                             xDispEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.x1;
894.                                             yDispEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.y1;
895.                                             zDispEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.z1;
```

```
896.                                                xRotEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.x2;
897.                                                yRotEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.y2;
898.                                                zRotEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.z2;
899.                                            }
900.                                        ));
901.                                }
902.                                catch
903.                                {
904.
905.                                }
906.                            }
907.
908.                            if (commands[0] == "Edit Nodes")
909.                            {
910.                                try
911.                                {
912.                                    // Send message to UI thread
913.
    System.Windows.Application.Current.Dispatcher.BeginInvoke(
914.                                        DispatcherPriority.Normal,
915.                                        (Action)(() =>
916.                                        {
917.                                            threadCommand = recived;
918.
919.                                            xCoordNodeEdit.Text =
    "Multiple nodes";
920.                                            yCoordNodeEdit.Text =
    "Multiple nodes";
921.                                            zCoordNodeEdit.Text =
    "Multiple nodes";
922.
923.                                            xCoordNodeEdit.IsEnabled =
    false;
924.                                            yCoordNodeEdit.IsEnabled =
    false;
925.                                            zCoordNodeEdit.IsEnabled =
    false;
926.                                        }
927.                                        ));
928.                                }
929.                                catch
930.                                {
931.
932.                                }
933.                            }
934.
935.                            if (commands[0] == "Create Member - No
    Node")
936.                            {
937.                                try
938.                                {
939.
    System.Windows.Application.Current.Dispatcher.BeginInvoke(
940.                                        DispatcherPriority.Normal,
941.                                        (Action)(() =>
942.                                        {
943.
    memberFirstNodeLabel.Content = "- First Node";
```

```
944.
   memberFirstNodeLabel.Foreground = Brushes.Red;
945.
946.                                            tempMemberN0 = -1;
947.                                            tempMemberN1 = -1;
948.                                        }
949.                                        ));
950.                                }
951.                                catch
952.                                {
953.
954.                                }
955.                            }
956.
957.                            if (commands[0] == "Create Member - N0")
958.                            {
959.                                try
960.                                {
961.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
962.                                        DispatcherPriority.Normal,
963.                                        (Action)(() =>
964.                                        {
965.
   memberFirstNodeLabel.Content = "- First Node: OK";
966.
   memberFirstNodeLabel.Foreground = Brushes.Black;
967.
968.
   memberSecondNodeLabel.Content = "- Second Node";
969.
   memberSecondNodeLabel.Foreground = Brushes.Red;
970.
971.                                            tempMemberN0 =
   int.Parse(commands[1]);
972.                                            tempMemberN1 = -1;
973.                                        }
974.                                        ));
975.                                }
976.                                catch
977.                                {
978.
979.                                }
980.                            }
981.
982.                            if (commands[0] == "Create Member - N1")
983.                            {
984.                                try
985.                                {
986.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
987.                                        DispatcherPriority.Normal,
988.                                        (Action)(() =>
989.                                        {
990.
   memberSecondNodeLabel.Content = "- Second Node: OK";
991.
   memberSecondNodeLabel.Foreground = Brushes.Black;
992.
993.                                            tempMemberN1 =
   int.Parse(commands[1]);
```

```
994.                                        }
995.                                        ));
996.                                }
997.                                catch
998.                                {
999.
1000.                                       }
1001.                               }
1002.
1003.                               if (commands[0] == "Try Member
    Creation")
1004.                               {
1005.                                   try
1006.                                   {
1007.
    System.Windows.Application.Current.Dispatcher.BeginInvoke(
1008.
    DispatcherPriority.Normal,
1009.                                                   (Action)(() =>
1010.                                                   {
1011.                                                       if (tempMemberN0 !=
    -1 && tempMemberN1 != -1 && materialMemberComboBox.SelectedItem !=
    null && propertyMemberComboBox.SelectedItem != null)
1012.                                                       {
1013.                                                           members.Add(new
    Member(members.Count, tempMemberN0, tempMemberN1,
    materials[materialMemberComboBox.SelectedIndex].name,
    properties[propertyMemberComboBox.SelectedIndex].name,
    dayMemberStartText.Text, dayMemberEndText.Text));
1014.
1015.                                                          bool changed =
    false;
1016.
1017.                                                          do
1018.                                                          {
1019.                                                              changed =
    false;
1020.
1021.                                                              int
    memberCount = members.Count;
1022.
1023.                                                              for (int i =
    0; i < memberCount; i++)
1024.                                                              {
1025.                                                                  for (int
    j = i; j < memberCount; j++)
1026.                                                                  {
1027.                                                                      if
    ((nodes[members[i].n0].position.y == nodes[members[j].n0].position.y)
    &&
1028.
    (nodes[members[i].n1].position.y == nodes[members[j].n1].position.y))
1029.                                                                      {
1030.
    double X = 0;
1031.
    double Z = 0;
1032.
1033.
    if (LineSegmentIntersection(
```

```
1034.
   nodes[members[i].n0].position.x, nodes[members[i].n0].position.z,
1035.
   nodes[members[i].n1].position.x, nodes[members[i].n1].position.z,
1036.
   nodes[members[j].n0].position.x, nodes[members[j].n0].position.z,
1037.
   nodes[members[j].n1].position.x, nodes[members[j].n1].position.z,
1038.
   ref X, ref Z
1039.
   ))
1040.
   {
1041.
   var tempVector3 = new Vector3(X, nodes[members[i].n0].position.y, Z);
1042.
1043.
   if (nodes[members[i].n0].position == tempVector3)
1044.
   {
1045.
   if ((nodes[members[j].n0].position == tempVector3) ||
   nodes[members[j].n1].position == tempVector3)
1046.
   {
1047.
1048.
   }
1049.
   else
1050.
   {
1051.
   int nodeExisting = members[i].n0;
1052.
1053.
   int tempNode = members[j].n1;
1054.
   members[j].n1 = nodeExisting;
1055.
1056.
   members.Add(new Member(members.Count, nodeExisting, tempNode,
   members[j].material, members[j].section,
1057.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1058.
1059.
   UpdateMemberForces(members[j].number, members.Count - 1);
1060.
   changed = true;
1061.
   }
1062.
   }
1063.
1064.
   else if (nodes[members[i].n1].position == tempVector3)
1065.
   {
```

```
1066.
   if ((nodes[members[j].n0].position == tempVector3) ||
   nodes[members[j].n1].position == tempVector3)
1067.
   {
1068.
1069.
   }
1070.
   else
1071.
   {
1072.
   int nodeExisting = members[i].n1;
1073.
1074.
   int tempNode = members[j].n1;
1075.
   members[j].n1 = nodeExisting;
1076.
1077.
   members.Add(new Member(members.Count, nodeExisting, tempNode,
   members[j].material, members[j].section,
1078.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1079.
1080.
   UpdateMemberForces(members[j].number, members.Count - 1);
1081.
   changed = true;
1082.
   }
1083.
   }
1084.
   else if (nodes[members[j].n0].position == tempVector3)
1085.
   {
1086.
   if ((nodes[members[i].n0].position == tempVector3) ||
   nodes[members[i].n1].position == tempVector3)
1087.
   {
1088.
1089.
   }
1090.
   else
1091.
   {
1092.
   int nodeExisting = members[j].n0;
1093.
1094.
   int tempNode = members[i].n1;
1095.
   members[i].n1 = nodeExisting;
1096.
1097.
   members.Add(new Member(members.Count, nodeExisting, tempNode,
   members[i].material, members[i].section,
```

```
1098.
   members[i].startDay.ToString(), members[i].endDay.ToString()));
1099.
1100.
   UpdateMemberForces(members[i].number, members.Count - 1);
1101.
   changed = true;
1102.
   }
1103.
   }
1104.
1105.
   else if (nodes[members[j].n1].position == tempVector3)
1106.
   {
1107.
   if ((nodes[members[i].n0].position == tempVector3) ||
   nodes[members[i].n1].position == tempVector3)
1108.
   {
1109.
1110.
   }
1111.
   else
1112.
   {
1113.
   int nodeExisting = members[j].n1;
1114.
1115.
   int tempNode = members[i].n1;
1116.
   members[i].n1 = nodeExisting;
1117.
1118.
   members.Add(new Member(members.Count, nodeExisting, tempNode,
   members[i].material, members[i].section,
1119.
   members[i].startDay.ToString(), members[i].endDay.ToString()));
1120.
1121.
   UpdateMemberForces(members[i].number, members.Count - 1);
1122.
   changed = true;
1123.
   }
1124.
   }
1125.
1126.
   else
1127.
   {
1128.
   nodes.Add(new Nodes(nodes.Count, tempVector3.x.ToString(),
   tempVector3.y.ToString(), tempVector3.z.ToString(), false, false,
   false, false, false, false));
1129.
```

```
1130.
   int newNode = nodes.Count - 1;
1131.
1132.
   // FIRST MEMBER:
1133.
   int tempNode = members[i].n1;
1134.
   members[i].n1 = newNode;
1135.
1136.
   members.Add(new Member(members.Count, newNode, tempNode,
   members[i].material, members[i].section,
1137.
   members[i].startDay.ToString(), members[i].endDay.ToString()));
1138.
1139.
   // SECOND MEMBER:
1140.
   tempNode = members[j].n1;
1141.
   members[j].n1 = newNode;
1142.
1143.
   members.Add(new Member(members.Count, newNode, tempNode,
   members[j].material, members[j].section,
1144.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1145.
1146.
   UpdateMemberForces(members[i].number, members.Count - 2);
1147.
   UpdateMemberForces(members[j].number, members.Count - 1);
1148.
1149.
   changed = true;
1150.
   }
1151.
   }
1152.                                                                    }
1153.                                                                    if
   (changed) break;
1154.                                                                 }
1155.                                                                 if
   (changed) break;
1156.                                                              }
1157.
1158.                                                          } while (changed
   == true);
1159.                                                    }
1160.
1161.
   memberFirstNodeLabel.Content = "- First Node";
1162.
   memberFirstNodeLabel.Foreground = Brushes.Red;
1163.
1164.
   memberSecondNodeLabel.Content = "- Second Node";
1165.
   memberSecondNodeLabel.Foreground = Brushes.Red;
```

```
1166.
1167.                                                   tempMemberN0 = -1;
1168.                                                   tempMemberN1 = -1;
1169.
1170.                                                   MassiveUpdate3DUI();
1171.                                               }
1172.                                               ));
1173.                                   }
1174.                                   catch
1175.                                   {
1176.
1177.                                   }
1178.                               }
1179.
1180.                               if (commands[0] == "Edit Member")
1181.                               {
1182.                                   try
1183.                                   {
1184.                                       // Send message to UI thread
1185.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1186.
   DispatcherPriority.Normal,
1187.                                               (Action)(() =>
1188.                                               {
1189.                                                   threadCommand =
   recived;
1190.
1191.
   materialMemberEditComboBox.SelectedItem =
   members[int.Parse(commands[1])].material;
1192.
   propertyMemberEditComboBox.SelectedItem =
   members[int.Parse(commands[1])].section;
1193.
1194.
   dayMemberStartEditText.Text =
   members[int.Parse(commands[1])].startDay.ToString();
1195.
   dayMemberEndEditText.Text =
   members[int.Parse(commands[1])].endDay.ToString();
1196.                                               }
1197.                                               ));
1198.                                   }
1199.                                   catch
1200.                                   {
1201.
1202.                                   }
1203.                               }
1204.
1205.                               if (commands[0] == "Edit Members")
1206.                               {
1207.                                   try
1208.                                   {
1209.                                       // Send message to UI thread
1210.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1211.
   DispatcherPriority.Normal,
1212.                                               (Action)(() =>
1213.                                               {
```

```
1214.                                                   threadCommand =
   recived;
1215.                                               }
1216.                                               ));
1217.                                       }
1218.                                       catch
1219.                                       {
1220.
1221.                                       }
1222.                                   }
1223.
1224.                                   if (commands[0] == "No Nodal Force"
   || commands[0] == "Nodal Force")
1225.                                   {
1226.                                       try
1227.                                       {
1228.                                           // Send message to UI thread
1229.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1230.
   DispatcherPriority.Normal,
1231.                                               (Action)(() =>
1232.                                               {
1233.                                                   threadCommand =
   recived;
1234.                                               }
1235.                                               ));
1236.                                       }
1237.                                       catch
1238.                                       {
1239.
1240.                                       }
1241.                                   }
1242.
1243.                                   if (commands[0] == "No Member Force"
   || commands[0] == "Member Force")
1244.                                   {
1245.                                       try
1246.                                       {
1247.                                           // Send message to UI thread
1248.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1249.
   DispatcherPriority.Normal,
1250.                                               (Action)(() =>
1251.                                               {
1252.                                                   threadCommand =
   recived;
1253.                                               }
1254.                                               ));
1255.                                       }
1256.                                       catch
1257.                                       {
1258.
1259.                                       }
1260.                                   }
1261.
1262.                                   if (commands[0] == "Simulation
   Object")
1263.                                   {
1264.                                       try
```

```
1265.                                       {
1266.                                           // Send message to UI thread
1267.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1268.
   DispatcherPriority.Normal,
1269.                                                   (Action)(() =>
1270.                                                   {
1271.                                                       if (commands[1] ==
   "Node")
1272.                                                       {
1273.
   UpdateSelected(commands[2]);
1274.                                                       }
1275.
1276.                                                       if (commands[1] ==
   "Member")
1277.                                                       {
1278.
   UpdateSelected("M" + commands[2]);
1279.                                                       }
1280.                                                   }
1281.                                                   ));
1282.                                       }
1283.                                       catch
1284.                                       {
1285.
1286.                                       }
1287.                                   }
1288.
1289.                                   if (commands[0] == "Selected
   Split:")
1290.                                   {
1291.                                       try
1292.                                       {
1293.                                           // Send message to UI thread
1294.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1295.
   DispatcherPriority.Normal,
1296.                                                   (Action)(() =>
1297.                                                   {
1298.                                                       threadCommand =
   recived;
1299.                                                   }
1300.                                                   ));
1301.                                       }
1302.                                       catch
1303.                                       {
1304.
1305.                                       }
1306.                                   }
1307.
1308.
1309.                                   Console.WriteLine("Message: " +
   recived);
1310.                                   //ASCIIEncoding asen = new
   ASCIIEncoding();
1311.                                   //s.Send(asen.GetBytes("Message
   received"));
1312.                               }
```

```
1313.                         }
1314.                         catch (Exception e)
1315.                         {
1316.                             Console.WriteLine("Error... " +
   e.StackTrace);
1317.                         }
1318.
1319.                     }).Start();
1320.                 }
1321.
1322.                 private void UpdateMemberForces(int memb, int toAdd)
1323.                 {
1324.                     for (int i = 0; i < memberForces.Count; i++)
1325.                     {
1326.                         if (memberForces[i].members.Contains(memb))
   memberForces[i].members.Add(toAdd);
1327.                     }
1328.                 }
1329.
1330.                 private void UpdateSelected(string s)
1331.                 {
1332.                     if (s == "") return;
1333.
1334.                     int day = int.Parse(SimShowDay.Text) - startDay;
1335.
1336.                     if (s.ElementAt(0) == 'M')
1337.                     {
1338.                         string memberSelected = s;
1339.
1340.                         SimPlot.IsEnabled = false;
1341.
1342.                         lastSimulatedSelected = memberSelected;
1343.
1344.                         try
1345.                         {
1346.                             var elm =
   models[day].Elements.Where(item => item.Label ==
   memberSelected).First() as FrameElement2Node;
1347.
1348.                             SimulationElementResults.Text =
   string.Format("Member selected: {0}", memberSelected) +
   System.Environment.NewLine;
1349.                             SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1350.                             SimulationElementResults.Text += "Forces
   at beginning:" + System.Environment.NewLine;
1351.                             SimulationElementResults.Text +=
   "Internal Force (kN):" + System.Environment.NewLine;
1352.                             SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};",
   elm.GetInternalForceAt(0).Fx, elm.GetInternalForceAt(0).Fy,
   elm.GetInternalForceAt(0).Fz) + System.Environment.NewLine;
1353.                             SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1354.                             SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};",
   elm.GetInternalForceAt(0).Mx, elm.GetInternalForceAt(0).My,
   elm.GetInternalForceAt(0).Mz) + System.Environment.NewLine;
1355.                             SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
```

```
1356.                               SimulationElementResults.Text += "Forces
   at end:" + System.Environment.NewLine;
1357.                               SimulationElementResults.Text +=
   "Internal Force (kN):" + System.Environment.NewLine;
1358.                               SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};",
   elm.GetInternalForceAt(1).Fx, elm.GetInternalForceAt(1).Fy,
   elm.GetInternalForceAt(1).Fz) + System.Environment.NewLine;
1359.                               SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1360.                               SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};",
   elm.GetInternalForceAt(1).Mx, elm.GetInternalForceAt(1).My,
   elm.GetInternalForceAt(1).Mz) + System.Environment.NewLine;
1361.                           }
1362.                           catch
1363.                           {
1364.                               SimulationElementResults.Text =
   string.Format("Member selected: {0}", memberSelected) +
   System.Environment.NewLine;
1365.                               SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1366.                               SimulationElementResults.Text += "Member
   does not yet exist!";
1367.                           }
1368.
1369.
1370.                       }
1371.                       else
1372.                       {
1373.                           string nodeSelected = s;
1374.
1375.                           lastSimulatedSelected = nodeSelected;
1376.
1377.                           SimPlot.IsEnabled = true;
1378.
1379.                           try
1380.                           {
1381.                               var disp = models[day].Nodes.Where(item
   => item.Label == nodeSelected).First().GetNodalDisplacement();
1382.                               var reac = models[day].Nodes.Where(item
   => item.Label == nodeSelected).First().GetSupportReaction();
1383.
1384.                               SimulationElementResults.Text =
   string.Format("Node selected: {0}", nodeSelected) +
   System.Environment.NewLine;
1385.                               SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1386.                               SimulationElementResults.Text +=
   "Displacement (m):" + System.Environment.NewLine;
1387.                               SimulationElementResults.Text +=
   string.Format("X: {0}; Y: {1}; Z: {2};", disp.DX, disp.DY, disp.DZ) +
   System.Environment.NewLine;
1388.                               SimulationElementResults.Text +=
   "Rotation (rad):" + System.Environment.NewLine;
1389.                               SimulationElementResults.Text +=
   string.Format("X: {0}; Y: {1}; Z: {2};", disp.RX, disp.RY, disp.RZ) +
   System.Environment.NewLine;
1390.                               SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
```

```
1391.                           SimulationElementResults.Text +=
   "Reactions (kN):" + System.Environment.NewLine;
1392.                           SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};", reac.Fx, reac.Fy,
   reac.Fz) + System.Environment.NewLine;
1393.                           SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1394.                           SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};", reac.Mx, reac.My,
   reac.Mz) + System.Environment.NewLine;
1395.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1396.                       }
1397.                       catch
1398.                       {
1399.                           SimulationElementResults.Text =
   string.Format("Node selected: {0}", nodeSelected) +
   System.Environment.NewLine;
1400.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1401.                           SimulationElementResults.Text += "Node
   does not yet exist!";
1402.                       }
1403.
1404.
1405.                   }
1406.               }
1407.
1408.               private void InitializeUnity3D()
1409.               {
1410.                   try
1411.                   {
1412.                       _panel = new System.Windows.Forms.Panel();
1413.                       wfhUnity3DCanvas.Child = _panel;
1414.
1415.                       //ProcessStartInfo psi = new
   ProcessStartInfo("notepad.exe");
1416.                       //_process = Process.Start(psi);
1417.                       //_process.StartInfo.UseShellExecute = true;
1418.                       //_process.StartInfo.CreateNoWindow = true;
1419.                       //_process.WaitForInputIdle();
1420.
1421.                       _process = new Process();
1422.                       string a =
   AppDomain.CurrentDomain.BaseDirectory;
1423.                       _process.StartInfo.FileName = a +
   "3DInterface\\3DInterface.exe";
1424.                       _process.StartInfo.Arguments = "-parentHWND
   " + _panel.Handle.ToInt32() + " " + Environment.CommandLine;
1425.                       _process.StartInfo.UseShellExecute = true;
1426.                       _process.StartInfo.CreateNoWindow = true;
1427.                       _process.Start();
1428.                       _process.WaitForInputIdle();
1429.
1430.                       SetParent(_process.MainWindowHandle,
   _panel.Handle);
1431.
1432.                       EnumChildWindows(_panel.Handle, WindowEnum,
   IntPtr.Zero);
1433.
1434.                       // remove control box
```

```
1435.                       //int style =
   GetWindowLong(_process.MainWindowHandle, GWL_STYLE);
1436.                       //style = style & ~WS_CAPTION &
   ~WS_THICKFRAME;
1437.                       //SetWindowLong(_process.MainWindowHandle,
   GWL_STYLE, style);
1438.
1439.                       // resize embedded application & refresh
1440.                       wfhUnityMockUpCanvas.SizeChanged +=
   WPFCanvas_SizeChanged;
1441.                   }
1442.                   catch (Exception e)
1443.                   {
1444.                   }
1445.               }
1446.               /// <summary>
1447.               /// Main thread exception handler
1448.               /// </summary>
1449.               /// <param name="sender">sender</param>
1450.               /// <param name="e">event</param>
1451.               public void Application_ThreadException(object
   sender, System.Threading.ThreadExceptionEventArgs e)
1452.               {
1453.                   if (_process != null)
1454.                   {
1455.                       _process.Refresh();
1456.                       _process.Close();
1457.                   }
1458.
1459.                   // Clean up sockets.
1460.                   s.Close();
1461.                   myList.Stop();
1462.               }
1463.
1464.               /// <summary>
1465.               /// Application domain exception handler
1466.               /// </summary>
1467.               /// <param name="sender">sender</param>
1468.               /// <param name="e">event</param>
1469.               public void AppDomain_UnhandledException(object
   sender, System.UnhandledExceptionEventArgs e)
1470.               {
1471.                   if (_process != null)
1472.                   {
1473.                       _process.Refresh();
1474.                       _process.Close();
1475.                   }
1476.
1477.                   // Clean up sockets.
1478.                   s.Close();
1479.                   myList.Stop();
1480.               }
1481.
1482.
1483.               protected override void
   OnClosing(System.ComponentModel.CancelEventArgs e)
1484.               {
1485.                   base.OnClosing(e);
1486.                   if (_process != null)
1487.                   {
1488.                       _process.Refresh();
```

```
1489.                       _process.Close();
1490.                   }
1491.
1492.                   // Clean up sockets.
1493.                   s.Close();
1494.                   myList.Stop();
1495.               }
1496.
1497.               private void ResizeEmbeddedApp()
1498.               {
1499.                   if (_process == null)
1500.                       return;
1501.
1502.                   MoveWindow(_unityHWND, 0, 0,
   (int)wfhUnityMockUpCanvas.ActualWidth,
   (int)wfhUnityMockUpCanvas.ActualHeight, true);
1503.               }
1504.
1505.               private void WPFCanvas_SizeChanged(object sender,
   SizeChangedEventArgs e)
1506.               {
1507.                   ResizeEmbeddedApp();
1508.               }
1509.
1510.               private int WindowEnum(IntPtr hwnd, IntPtr lparam)
1511.               {
1512.                   _unityHWND = hwnd;
1513.                   ActivateUnityWindow();
1514.                   return 0;
1515.               }
1516.
1517.               private void ActivateUnityWindow()
1518.               {
1519.                   SendMessage(_unityHWND, WM_ACTIVATE, WA_ACTIVE,
   IntPtr.Zero);
1520.               }
1521.
1522.               private void DeactivateUnityWindow()
1523.               {
1524.                   SendMessage(_unityHWND, WM_ACTIVATE,
   WA_INACTIVE, IntPtr.Zero);
1525.               }
1526.
1527.               private void Hand_Click(object sender,
   RoutedEventArgs e)
1528.               {
1529.                   //double[] time = new double[100];
1530.
1531.                   //Parallel.For(0, time.Length,
1532.                   //       index =>
1533.                   //       {
1534.                   //           StructuralAnalysisMathNet analysis
   = new StructuralAnalysisMathNet();
1535.                   //           time[index] = analysis.time;
1536.                   //       });
1537.
1538.                   //double result = 0;
1539.
1540.                   //for (int i = 0; i < time.Length; i++)
1541.                   //{
1542.                   //    result += time[i];
```

```
1543.                   //}
1544.
1545.                   //result = result / time.Length;
1546.
1547.                   //abc.Text = result.ToString();
1548.               }
1549.
1550.               private void MainWindow_Loaded(object sender,
   RoutedEventArgs e)
1551.               {
1552.                   ResizeEmbeddedApp();
1553.                   IntPtr hwnd = new
   WindowInteropHelper(this).Handle;
1554.                   HwndSource src = HwndSource.FromHwnd(hwnd);
1555.                   //Receive window message handler implementation
   (based on System.Windows.Interop.HwndSourceHook commissioned)
1556.                   src.AddHook(new HwndSourceHook(WndProc));
1557.               }
1558.
1559.               IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam,
   IntPtr lParam, ref bool handled)
1560.               {
1561.                   switch (msg)
1562.                   {
1563.                       case 528:
1564.                           // Mouse click on Unity Window
1565.                           ActivateUnityWindow();
1566.                           break;
1567.                   }
1568.                   return IntPtr.Zero;
1569.               }
1570.
1571.               private void SendData(byte[] data)
1572.               {
1573.                   SocketAsyncEventArgs socketAsyncData = new
   SocketAsyncEventArgs();
1574.                   socketAsyncData.SetBuffer(data, 0, data.Length);
1575.                   s.SendAsync(socketAsyncData);
1576.
1577.               }
1578.
1579.               public void CreateMaterial(object sender,
   RoutedEventArgs e)
1580.               {
1581.                   if (matName.Text != "" && matE.Text != "" &&
   matG.Text != "")
1582.                   {
1583.                       try
1584.                       {
1585.                           if (materials.Where(item => item.name ==
   matName.Text).ToList().Count == 0)
1586.                           {
1587.                               if (fckText.Text == "") fckText.Text
   = "0";
1588.
1589.                               materials.Add(new
   Material(matName.Text, matE.Text, matG.Text,
   isTimeDependent.IsChecked.GetValueOrDefault(), fckText.Text,
   matV.Text));
1590.
1591.                               matName.Text = "";
```

```
1592.                                   matE.Text = "";
1593.                                   matG.Text = "";
1594.                                   matV.Text = "";
1595.
1596.                                   isTimeDependent.IsChecked = false;
1597.
1598.                                   fckText.Text = "";
1599.
1600.                                   labelWrongMatP.Content = "Material
   Saved!";
1601.                                   labelWrongMatP.Visibility =
   Visibility.Visible;
1602.                               }
1603.                               else
1604.                               {
1605.                                   labelWrongMatP.Content = "Wrong
   Properties!";
1606.                                   labelWrongMatP.Visibility =
   Visibility.Visible;
1607.                               }
1608.                           }
1609.                           catch
1610.                           {
1611.                               labelWrongMatP.Content = "Wrong
   Properties!";
1612.                               labelWrongMatP.Visibility =
   Visibility.Visible;
1613.                           }
1614.                       }
1615.                       else
1616.                       {
1617.                           labelWrongMatP.Content = "Wrong
   Properties!";
1618.                           labelWrongMatP.Visibility =
   Visibility.Visible;
1619.                       }
1620.                   }
1621.
1622.                   public void UpdateG(object sender, RoutedEventArgs
   e)
1623.                   {
1624.                       if (matE.Text != "" && matV.Text != "")
1625.                       {
1626.                           try
1627.                           {
1628.                               matG.Text = (double.Parse(matE.Text) /
   (2 * (1 + double.Parse(matV.Text)))).ToString();
1629.                           }
1630.                           catch
1631.                           {
1632.                               labelWrongMatP.Content = "Wrong
   Properties!";
1633.                               labelWrongMatP.Visibility =
   Visibility.Visible;
1634.                           }
1635.                       }
1636.                       else
1637.                       {
1638.                           labelWrongMatP.Content = "Wrong
   Properties!";
```

```
1639.                         labelWrongMatP.Visibility =
   Visibility.Visible;
1640.                     }
1641.                 }
1642.
1643.                 public void UpdateGEdit(object sender,
   RoutedEventArgs e)
1644.                 {
1645.                     if (matEEdit.Text != "" && matVEdit.Text != "")
1646.                     {
1647.                         try
1648.                         {
1649.                             matGEdit.Text =
   (double.Parse(matEEdit.Text) / (2 * (1 +
   double.Parse(matVEdit.Text)))).ToString();
1650.                         }
1651.                         catch
1652.                         {
1653.                             labelWrongMatPEdit.Content = "Wrong
   Properties!";
1654.                             labelWrongMatPEdit.Visibility =
   Visibility.Visible;
1655.                         }
1656.                     }
1657.                     else
1658.                     {
1659.                         labelWrongMatPEdit.Content = "Wrong
   Properties!";
1660.                         labelWrongMatPEdit.Visibility =
   Visibility.Visible;
1661.                     }
1662.                 }
1663.
1664.                 public void CreateNodalForce(object sender,
   RoutedEventArgs e)
1665.                 {
1666.                     if (nodalForceName.Text == "")
1667.                     {
1668.                         NodalForceError();
1669.                         return;
1670.                     }
1671.
1672.                     if (nodalForces.Where(item => item.name ==
   nodalForceName.Text).ToList().Count > 0)
1673.                     {
1674.                         NodalForceError();
1675.                         return;
1676.                     }
1677.
1678.                     if (threadCommand == "No Nodal Force")
1679.                     {
1680.                         NodalForceError();
1681.                         return;
1682.                     }
1683.
1684.                     try
1685.                     {
1686.                         if (nodalForceDisX.Text == "")
   nodalForceDisX.Text = "0";
1687.                         if (nodalForceDisY.Text == "")
   nodalForceDisY.Text = "0";
```

```
1688.                         if (nodalForceDisZ.Text == "")
   nodalForceDisZ.Text = "0";
1689.                         if (nodalForceRotX.Text == "")
   nodalForceRotX.Text = "0";
1690.                         if (nodalForceRotY.Text == "")
   nodalForceRotY.Text = "0";
1691.                         if (nodalForceRotZ.Text == "")
   nodalForceRotZ.Text = "0";
1692.
1693.                         double dx =
   double.Parse(nodalForceDisX.Text);
1694.                         double dy =
   double.Parse(nodalForceDisY.Text);
1695.                         double dz =
   double.Parse(nodalForceDisZ.Text);
1696.
1697.                         double rx =
   double.Parse(nodalForceRotX.Text);
1698.                         double ry =
   double.Parse(nodalForceRotY.Text);
1699.                         double rz =
   double.Parse(nodalForceRotZ.Text);
1700.
1701.                         int startDay =
   int.Parse(nodalForceStartDay.Text);
1702.                         int endDay =
   int.Parse(nodalForceEndDay.Text);
1703.
1704.                         List<string> listString =
   threadCommand.Split('&').ToList<string>();
1705.
1706.                         List<int> list = new List<int>();
1707.
1708.                         for (int i = 1; i < listString.Count; i++)
1709.                         {
1710.                             list.Add(int.Parse(listString[i]));
1711.                         }
1712.
1713.                         if (dx == 0 && dy == 0 && dz == 0 && rx == 0
   && ry == 0 && rz == 0)
1714.                         {
1715.                             NodalForceError();
1716.                         }
1717.
1718.                         nodalForces.Add(new
   NodalForces(nodalForceName.Text, dx, dy, dz, rx, ry, rz, startDay,
   endDay, list));
1719.
1720.                         nodalForceName.Text = "";
1721.                         nodalForceDisX.Text = "";
1722.                         nodalForceDisY.Text = "";
1723.                         nodalForceDisZ.Text = "";
1724.                         nodalForceRotX.Text = "";
1725.                         nodalForceRotY.Text = "";
1726.                         nodalForceRotZ.Text = "";
1727.                         nodalForceStartDay.Text = "";
1728.                         nodalForceEndDay.Text = "";
1729.
1730.                         nodalForceError.Visibility =
   Visibility.Collapsed;
1731.
```

```
1732.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation"));
1733.                     }
1734.                     catch
1735.                     {
1736.                         NodalForceError();
1737.                     }
1738.                 }
1739.
1740.                 public void AddNodalForceHeaderSelected(object
   sender, RoutedEventArgs e)
1741.                 {
1742.                     SendStringToInterface(string.Format("Mode&Nodal
   Force Creation"));
1743.                     ForcesMainHeader.IsSelected = true;
1744.                     AddNodalForceHeader.IsSelected = true;
1745.
1746.                     nodalForceName.Text = "";
1747.                     nodalForceDisX.Text = "";
1748.                     nodalForceDisY.Text = "";
1749.                     nodalForceDisZ.Text = "";
1750.                     nodalForceRotX.Text = "";
1751.                     nodalForceRotY.Text = "";
1752.                     nodalForceRotZ.Text = "";
1753.                     nodalForceStartDay.Text = "";
1754.                     nodalForceEndDay.Text = "";
1755.
1756.                     nodalForceError.Visibility =
   Visibility.Collapsed;
1757.                 }
1758.
1759.                 public void EditNodalForcesHeaderSelected(object
   sender, RoutedEventArgs e)
1760.                 {
1761.                     nodalForceEditComboBox.Items.Clear();
1762.
1763.                     for (int i = 0; i < nodalForces.Count; i++)
1764.                     {
1765.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1766.                     }
1767.
1768.                     nodalForceNameEdit.Text = "";
1769.                     nodalForceDisXEdit.Text = "";
1770.                     nodalForceDisYEdit.Text = "";
1771.                     nodalForceDisZEdit.Text = "";
1772.                     nodalForceRotXEdit.Text = "";
1773.                     nodalForceRotYEdit.Text = "";
1774.                     nodalForceRotZEdit.Text = "";
1775.                     nodalForceStartDayEdit.Text = "";
1776.                     nodalForceEndDayEdit.Text = "";
1777.
1778.                     nodalForceNameEdit.IsEnabled = false;
1779.                     nodalForceDisXEdit.IsEnabled = false;
1780.                     nodalForceDisYEdit.IsEnabled = false;
1781.                     nodalForceDisZEdit.IsEnabled = false;
1782.                     nodalForceRotXEdit.IsEnabled = false;
1783.                     nodalForceRotYEdit.IsEnabled = false;
1784.                     nodalForceRotZEdit.IsEnabled = false;
1785.                     nodalForceStartDayEdit.IsEnabled = false;
1786.                     nodalForceEndDayEdit.IsEnabled = false;
```

```
1787.
1788.                     nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1789.
1790.                     EditNodalForceHeader.IsSelected = true;
1791.                 }
1792.
1793.                 public void
   EditNodalForcesComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
1794.                 {
1795.                     if (!finishedLoading) return;
1796.
1797.                     if (nodalForceEditComboBox.SelectedItem != null)
1798.                     {
1799.                         nodalForceNameEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().name;
1800.                         nodalForceDisXEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.x.ToStri
   ng();
1801.                         nodalForceDisYEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.y.ToStri
   ng();
1802.                         nodalForceDisZEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.z.ToStri
   ng();
1803.                         nodalForceRotXEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.x.ToStri
   ng();
1804.                         nodalForceRotYEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.y.ToStri
   ng();
1805.                         nodalForceRotZEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.z.ToStri
   ng();
1806.                         nodalForceStartDayEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().startDay.ToSt
   ring();
1807.                         nodalForceEndDayEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().endDay.ToStri
   ng();
1808.
1809.                         nodalForceNameEdit.IsEnabled = true;
1810.                         nodalForceDisXEdit.IsEnabled = true;
1811.                         nodalForceDisYEdit.IsEnabled = true;
1812.                         nodalForceDisZEdit.IsEnabled = true;
1813.                         nodalForceRotXEdit.IsEnabled = true;
1814.                         nodalForceRotYEdit.IsEnabled = true;
1815.                         nodalForceRotZEdit.IsEnabled = true;
1816.                         nodalForceStartDayEdit.IsEnabled = true;
1817.                         nodalForceEndDayEdit.IsEnabled = true;
1818.                         nodalForcesEditSave.IsEnabled = true;
```

```
1819.                         nodalForcesEditDelete.IsEnabled = true;
1820.
1821.                         nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1822.
1823.                         string listString = "";
1824.
1825.                         for (int i = 0; i < nodalForces.Where(item
   => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes.Count;
   i++)
1826.                         {
1827.                             if (listString == "")
1828.                             {
1829.                                 listString += nodalForces.Where(item
   => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes[i].ToSt
   ring();
1830.                             }
1831.                             else
1832.                             {
1833.                                 listString += "&" +
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes[i].ToSt
   ring();
1834.                             }
1835.                         }
1836.
1837.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation&{0}",
   listString));
1838.                     }
1839.                     else
1840.                     {
1841.                         nodalForceNameEdit.Text = "";
1842.                         nodalForceDisXEdit.Text = "";
1843.                         nodalForceDisYEdit.Text = "";
1844.                         nodalForceDisZEdit.Text = "";
1845.                         nodalForceRotXEdit.Text = "";
1846.                         nodalForceRotYEdit.Text = "";
1847.                         nodalForceRotZEdit.Text = "";
1848.                         nodalForceStartDayEdit.Text = "";
1849.                         nodalForceEndDayEdit.Text = "";
1850.
1851.                         nodalForceNameEdit.IsEnabled = false;
1852.                         nodalForceDisXEdit.IsEnabled = false;
1853.                         nodalForceDisYEdit.IsEnabled = false;
1854.                         nodalForceDisZEdit.IsEnabled = false;
1855.                         nodalForceRotXEdit.IsEnabled = false;
1856.                         nodalForceRotYEdit.IsEnabled = false;
1857.                         nodalForceRotZEdit.IsEnabled = false;
1858.                         nodalForceStartDayEdit.IsEnabled = false;
1859.                         nodalForceEndDayEdit.IsEnabled = false;
1860.                         nodalForcesEditSave.IsEnabled = false;
1861.                         nodalForcesEditDelete.IsEnabled = false;
1862.
1863.                         nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1864.                     }
1865.                 }
1866.
```

```
1867.               public void EditNodalForce(object sender,
    RoutedEventArgs e)
1868.               {
1869.                   if (nodalForceNameEdit.Text == "")
1870.                   {
1871.                       NodalForceErrorEdit();
1872.                       return;
1873.                   }
1874.
1875.                   if
    (nodalForceEditComboBox.SelectedItem.ToString() !=
    nodalForceNameEdit.Text && nodalForces.Where(item => item.name ==
    nodalForceNameEdit.Text).ToList().Count != 0)
1876.                   {
1877.                       NodalForceErrorEdit();
1878.                       return;
1879.                   }
1880.
1881.                   if (threadCommand == "No Nodal Force")
1882.                   {
1883.                       NodalForceErrorEdit();
1884.                       return;
1885.                   }
1886.
1887.                   try
1888.                   {
1889.                       if (nodalForceDisXEdit.Text == "")
    nodalForceDisXEdit.Text = "0";
1890.                       if (nodalForceDisYEdit.Text == "")
    nodalForceDisYEdit.Text = "0";
1891.                       if (nodalForceDisZEdit.Text == "")
    nodalForceDisZEdit.Text = "0";
1892.                       if (nodalForceRotXEdit.Text == "")
    nodalForceRotXEdit.Text = "0";
1893.                       if (nodalForceRotYEdit.Text == "")
    nodalForceRotYEdit.Text = "0";
1894.                       if (nodalForceRotZEdit.Text == "")
    nodalForceRotZEdit.Text = "0";
1895.
1896.                       double dx =
    double.Parse(nodalForceDisXEdit.Text);
1897.                       double dy =
    double.Parse(nodalForceDisYEdit.Text);
1898.                       double dz =
    double.Parse(nodalForceDisZEdit.Text);
1899.
1900.                       double rx =
    double.Parse(nodalForceRotXEdit.Text);
1901.                       double ry =
    double.Parse(nodalForceRotYEdit.Text);
1902.                       double rz =
    double.Parse(nodalForceRotZEdit.Text);
1903.
1904.                       int startDay =
    int.Parse(nodalForceStartDayEdit.Text);
1905.                       int endDay =
    int.Parse(nodalForceEndDayEdit.Text);
1906.
1907.                       List<string> listString =
    threadCommand.Split('&').ToList<string>();
1908.
```

```
1909.                       List<int> list = new List<int>();
1910.
1911.                       for (int i = 1; i < listString.Count; i++)
1912.                       {
1913.                           list.Add(int.Parse(listString[i]));
1914.                       }
1915.
1916.                       if (dx == 0 && dy == 0 && dz == 0 && rx == 0
   && ry == 0 && rz == 0)
1917.                       {
1918.                           NodalForceErrorEdit();
1919.                       }
1920.
1921.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.x = dx;
1922.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.y = dy;
1923.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.z = dz;
1924.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.x = rx;
1925.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.y = ry;
1926.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.z = rz;
1927.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().startDay =
   startDay;
1928.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().endDay =
   endDay;
1929.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes.Clear()
   ;
1930.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes = list;
1931.
1932.                       nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().name =
   nodalForceNameEdit.Text;
1933.
1934.                       nodalForceNameEdit.Text = "";
1935.                       nodalForceDisXEdit.Text = "";
1936.                       nodalForceDisYEdit.Text = "";
1937.                       nodalForceDisZEdit.Text = "";
1938.                       nodalForceRotXEdit.Text = "";
1939.                       nodalForceRotYEdit.Text = "";
1940.                       nodalForceRotZEdit.Text = "";
1941.                       nodalForceStartDayEdit.Text = "";
1942.                       nodalForceEndDayEdit.Text = "";
1943.
1944.                       nodalForceNameEdit.IsEnabled = false;
1945.                       nodalForceDisXEdit.IsEnabled = false;
1946.                       nodalForceDisYEdit.IsEnabled = false;
1947.                       nodalForceDisZEdit.IsEnabled = false;
1948.                       nodalForceRotXEdit.IsEnabled = false;
1949.                       nodalForceRotYEdit.IsEnabled = false;
1950.                       nodalForceRotZEdit.IsEnabled = false;
1951.                       nodalForceStartDayEdit.IsEnabled = false;
1952.                       nodalForceEndDayEdit.IsEnabled = false;
1953.                       nodalForcesEditSave.IsEnabled = false;
```

```
1954.                       nodalForcesEditDelete.IsEnabled = false;
1955.
1956.                       nodalForceEditComboBox.Items.Clear();
1957.
1958.                       for (int i = 0; i < nodalForces.Count; i++)
1959.                       {
1960.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1961.                       }
1962.
1963.                       nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1964.
1965.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation"));
1966.                   }
1967.                   catch
1968.                   {
1969.                       NodalForceErrorEdit();
1970.                   }
1971.               }
1972.
1973.               public void DeleteNodalForce(object sender,
   RoutedEventArgs e)
1974.               {
1975.                   nodalForces.Remove(nodalForces.Where(item =>
   item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First());
1976.
1977.                   nodalForceEditComboBox.Items.Clear();
1978.
1979.                   for (int i = 0; i < nodalForces.Count; i++)
1980.                   {
1981.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1982.                   }
1983.
1984.                   nodalForceNameEdit.Text = "";
1985.                   nodalForceDisXEdit.Text = "";
1986.                   nodalForceDisYEdit.Text = "";
1987.                   nodalForceDisZEdit.Text = "";
1988.                   nodalForceRotXEdit.Text = "";
1989.                   nodalForceRotYEdit.Text = "";
1990.                   nodalForceRotZEdit.Text = "";
1991.                   nodalForceStartDayEdit.Text = "";
1992.                   nodalForceEndDayEdit.Text = "";
1993.
1994.                   nodalForceNameEdit.IsEnabled = false;
1995.                   nodalForceDisXEdit.IsEnabled = false;
1996.                   nodalForceDisYEdit.IsEnabled = false;
1997.                   nodalForceDisZEdit.IsEnabled = false;
1998.                   nodalForceRotXEdit.IsEnabled = false;
1999.                   nodalForceRotYEdit.IsEnabled = false;
2000.                   nodalForceRotZEdit.IsEnabled = false;
2001.                   nodalForceStartDayEdit.IsEnabled = false;
2002.                   nodalForceEndDayEdit.IsEnabled = false;
2003.
2004.                   nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
2005.
2006.                   EditNodalForceHeader.IsSelected = true;
```

```
2007.
2008.                     SendStringToInterface(string.Format("Mode&Nodal
   Force Creation"));
2009.                 }
2010.
2011.                 public void NodalForceError()
2012.                 {
2013.                     nodalForceError.Visibility = Visibility.Visible;
2014.                 }
2015.
2016.                 public void NodalForceErrorEdit()
2017.                 {
2018.                     nodalForceErrorEdit.Visibility =
   Visibility.Visible;
2019.                 }
2020.
2021.
2022.
2023.                 public void CreateMemberForce(object sender,
   RoutedEventArgs e)
2024.                 {
2025.                     if (memberForceName.Text == "")
2026.                     {
2027.                         MemberForceError();
2028.                         return;
2029.                     }
2030.
2031.                     if (memberForces.Where(item => item.name ==
   memberForceName.Text).ToList().Count > 0)
2032.                     {
2033.                         MemberForceError();
2034.                         return;
2035.                     }
2036.
2037.                     if (threadCommand == "No Member Force")
2038.                     {
2039.                         MemberForceError();
2040.                         return;
2041.                     }
2042.
2043.                     try
2044.                     {
2045.                         if (memberForceDisX.Text == "")
   memberForceDisX.Text = "0";
2046.                         if (memberForceDisY.Text == "")
   memberForceDisY.Text = "0";
2047.                         if (memberForceDisZ.Text == "")
   memberForceDisZ.Text = "0";
2048.
2049.                         double dx =
   double.Parse(memberForceDisX.Text);
2050.                         double dy =
   double.Parse(memberForceDisY.Text);
2051.                         double dz =
   double.Parse(memberForceDisZ.Text);
2052.
2053.                         int startDay =
   int.Parse(memberForceStartDay.Text);
2054.                         int endDay =
   int.Parse(memberForceEndDay.Text);
2055.
```

```
2056.                       List<string> listString =
   threadCommand.Split('&').ToList<string>();
2057.
2058.                       List<int> list = new List<int>();
2059.
2060.                       for (int i = 1; i < listString.Count; i++)
2061.                       {
2062.                           list.Add(int.Parse(listString[i]));
2063.                       }
2064.
2065.                       if (dx == 0 && dy == 0 && dz == 0)
2066.                       {
2067.                           MemberForceError();
2068.                       }
2069.
2070.                       memberForces.Add(new
   MemberForces(memberForceName.Text, dx, dy, dz, startDay, endDay,
   list));
2071.
2072.                       memberForceName.Text = "";
2073.                       memberForceDisX.Text = "";
2074.                       memberForceDisY.Text = "";
2075.                       memberForceDisZ.Text = "";
2076.                       memberForceStartDay.Text = "";
2077.                       memberForceEndDay.Text = "";
2078.
2079.                       memberForceError.Visibility =
   Visibility.Collapsed;
2080.
2081.
   SendStringToInterface(string.Format("Mode&Member Force Creation"));
2082.                   }
2083.                   catch
2084.                   {
2085.                       MemberForceError();
2086.                   }
2087.               }
2088.
2089.               public void AddMemberForceHeaderSelected(object
   sender, RoutedEventArgs e)
2090.               {
2091.                   SendStringToInterface(string.Format("Mode&Member
   Force Creation"));
2092.                   ForcesMainHeader.IsSelected = true;
2093.                   AddMemberForceHeader.IsSelected = true;
2094.
2095.                   memberForceName.Text = "";
2096.                   memberForceDisX.Text = "";
2097.                   memberForceDisY.Text = "";
2098.                   memberForceDisZ.Text = "";
2099.                   memberForceStartDay.Text = "";
2100.                   memberForceEndDay.Text = "";
2101.
2102.                   memberForceError.Visibility =
   Visibility.Collapsed;
2103.               }
2104.
2105.               public void EditMemberForcesHeaderSelected(object
   sender, RoutedEventArgs e)
2106.               {
2107.                   memberForceEditComboBox.Items.Clear();
```

```
2108.
2109.                   for (int i = 0; i < memberForces.Count; i++)
2110.                   {
2111.
   memberForceEditComboBox.Items.Add(memberForces[i].name);
2112.                   }
2113.
2114.                   memberForceNameEdit.Text = "";
2115.                   memberForceDisXEdit.Text = "";
2116.                   memberForceDisYEdit.Text = "";
2117.                   memberForceDisZEdit.Text = "";
2118.                   memberForceStartDayEdit.Text = "";
2119.                   memberForceEndDayEdit.Text = "";
2120.
2121.                   memberForceNameEdit.IsEnabled = false;
2122.                   memberForceDisXEdit.IsEnabled = false;
2123.                   memberForceDisYEdit.IsEnabled = false;
2124.                   memberForceDisZEdit.IsEnabled = false;
2125.                   memberForceStartDayEdit.IsEnabled = false;
2126.                   memberForceEndDayEdit.IsEnabled = false;
2127.
2128.                   memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2129.
2130.                   EditMemberForceHeader.IsSelected = true;
2131.               }
2132.
2133.               public void
   EditMemberForcesComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
2134.               {
2135.                   if (!finishedLoading) return;
2136.
2137.                   if (memberForceEditComboBox.SelectedItem !=
   null)
2138.                   {
2139.                       memberForceNameEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().name;
2140.                       memberForceDisXEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.x.ToStr
   ing();
2141.                       memberForceDisYEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.y.ToStr
   ing();
2142.                       memberForceDisZEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.z.ToStr
   ing();
2143.                       memberForceStartDayEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().startDay.ToS
   tring();
2144.                       memberForceEndDayEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().endDay.ToStr
   ing();
2145.
2146.                       memberForceNameEdit.IsEnabled = true;
```

```
2147.                         memberForceDisXEdit.IsEnabled = true;
2148.                         memberForceDisYEdit.IsEnabled = true;
2149.                         memberForceDisZEdit.IsEnabled = true;
2150.                         memberForceStartDayEdit.IsEnabled = true;
2151.                         memberForceEndDayEdit.IsEnabled = true;
2152.                         memberForcesEditSave.IsEnabled = true;
2153.                         memberForcesEditDelete.IsEnabled = true;
2154.
2155.                         memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2156.
2157.                         string listString = "";
2158.
2159.                         for (int i = 0; i < memberForces.Where(item
   => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members.Coun
   t; i++)
2160.                         {
2161.                             if (listString == "")
2162.                             {
2163.                                 listString +=
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members[i].T
   oString();
2164.                             }
2165.                             else
2166.                             {
2167.                                 listString += "&" +
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members[i].T
   oString();
2168.                             }
2169.                         }
2170.
2171.
   SendStringToInterface(string.Format("Mode&Member Force Creation&{0}",
   listString));
2172.                     }
2173.                     else
2174.                     {
2175.                         memberForceNameEdit.Text = "";
2176.                         memberForceDisXEdit.Text = "";
2177.                         memberForceDisYEdit.Text = "";
2178.                         memberForceDisZEdit.Text = "";
2179.                         memberForceStartDayEdit.Text = "";
2180.                         memberForceEndDayEdit.Text = "";
2181.
2182.                         memberForceNameEdit.IsEnabled = false;
2183.                         memberForceDisXEdit.IsEnabled = false;
2184.                         memberForceDisYEdit.IsEnabled = false;
2185.                         memberForceDisZEdit.IsEnabled = false;
2186.                         memberForceStartDayEdit.IsEnabled = false;
2187.                         memberForceEndDayEdit.IsEnabled = false;
2188.                         memberForcesEditSave.IsEnabled = false;
2189.                         memberForcesEditDelete.IsEnabled = false;
2190.
2191.                         memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2192.                     }
2193.                 }
2194.
```

```
2195.               public void EditMemberForce(object sender,
    RoutedEventArgs e)
2196.               {
2197.                   if (memberForceNameEdit.Text == "")
2198.                   {
2199.                       MemberForceErrorEdit();
2200.                       return;
2201.                   }
2202.
2203.                   if
    (memberForceEditComboBox.SelectedItem.ToString() !=
    memberForceNameEdit.Text && memberForces.Where(item => item.name ==
    memberForceNameEdit.Text).ToList().Count != 0)
2204.                   {
2205.                       MemberForceErrorEdit();
2206.                       return;
2207.                   }
2208.
2209.                   if (threadCommand == "No Member Force")
2210.                   {
2211.                       MemberForceErrorEdit();
2212.                       return;
2213.                   }
2214.
2215.                   try
2216.                   {
2217.                       if (memberForceDisXEdit.Text == "")
    memberForceDisXEdit.Text = "0";
2218.                       if (memberForceDisYEdit.Text == "")
    memberForceDisYEdit.Text = "0";
2219.                       if (memberForceDisZEdit.Text == "")
    memberForceDisZEdit.Text = "0";
2220.
2221.                       double dx =
    double.Parse(memberForceDisXEdit.Text);
2222.                       double dy =
    double.Parse(memberForceDisYEdit.Text);
2223.                       double dz =
    double.Parse(memberForceDisZEdit.Text);
2224.
2225.                       int startDay =
    int.Parse(memberForceStartDayEdit.Text);
2226.                       int endDay =
    int.Parse(memberForceEndDayEdit.Text);
2227.
2228.                       List<string> listString =
    threadCommand.Split('&').ToList<string>();
2229.
2230.                       List<int> list = new List<int>();
2231.
2232.                       for (int i = 1; i < listString.Count; i++)
2233.                       {
2234.                           if (listString[i] != "")
2235.                           {
2236.                               list.Add(int.Parse(listString[i]));
2237.                           }
2238.                       }
2239.
2240.                       if (dx == 0 && dy == 0 && dz == 0)
2241.                       {
2242.                           MemberForceErrorEdit();
```

```
2243.                         }
2244.
2245.                         memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.x = dx;
2246.                         memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.y = dy;
2247.                         memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.z = dz;
2248.                         memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().startDay =
   startDay;
2249.                         memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().endDay =
   endDay;
2250.                         memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members.Clea
   r();
2251.                         memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members =
   list;
2252.
2253.                         memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().name =
   memberForceNameEdit.Text;
2254.
2255.                         memberForceNameEdit.Text = "";
2256.                         memberForceDisXEdit.Text = "";
2257.                         memberForceDisYEdit.Text = "";
2258.                         memberForceDisZEdit.Text = "";
2259.                         memberForceStartDayEdit.Text = "";
2260.                         memberForceEndDayEdit.Text = "";
2261.
2262.                         memberForceNameEdit.IsEnabled = false;
2263.                         memberForceDisXEdit.IsEnabled = false;
2264.                         memberForceDisYEdit.IsEnabled = false;
2265.                         memberForceDisZEdit.IsEnabled = false;
2266.                         memberForceStartDayEdit.IsEnabled = false;
2267.                         memberForceEndDayEdit.IsEnabled = false;
2268.                         memberForcesEditSave.IsEnabled = false;
2269.                         memberForcesEditDelete.IsEnabled = false;
2270.
2271.                         memberForceEditComboBox.Items.Clear();
2272.
2273.                         for (int i = 0; i < memberForces.Count; i++)
2274.                         {
2275.
   memberForceEditComboBox.Items.Add(memberForces[i].name);
2276.                         }
2277.
2278.                         memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2279.
2280.
   SendStringToInterface(string.Format("Mode&Member Force Creation"));
2281.                     }
2282.                     catch
2283.                     {
2284.                         MemberForceErrorEdit();
2285.                     }
2286.                 }
2287.
```

```
2288.               public void DeleteMemberForce(object sender,
    RoutedEventArgs e)
2289.               {
2290.                   memberForces.Remove(memberForces.Where(item =>
    item.name ==
    memberForceEditComboBox.SelectedItem.ToString()).First());
2291.
2292.                   memberForceEditComboBox.Items.Clear();
2293.
2294.                   for (int i = 0; i < memberForces.Count; i++)
2295.                   {
2296.
    memberForceEditComboBox.Items.Add(memberForces[i].name);
2297.                   }
2298.
2299.                   memberForceNameEdit.Text = "";
2300.                   memberForceDisXEdit.Text = "";
2301.                   memberForceDisYEdit.Text = "";
2302.                   memberForceDisZEdit.Text = "";
2303.                   memberForceStartDayEdit.Text = "";
2304.                   memberForceEndDayEdit.Text = "";
2305.
2306.                   memberForceNameEdit.IsEnabled = false;
2307.                   memberForceDisXEdit.IsEnabled = false;
2308.                   memberForceDisYEdit.IsEnabled = false;
2309.                   memberForceDisZEdit.IsEnabled = false;
2310.                   memberForceStartDayEdit.IsEnabled = false;
2311.                   memberForceEndDayEdit.IsEnabled = false;
2312.
2313.                   memberForceErrorEdit.Visibility =
    Visibility.Collapsed;
2314.
2315.                   EditMemberForceHeader.IsSelected = true;
2316.
2317.                   SendStringToInterface(string.Format("Mode&Member
    Force Creation"));
2318.               }
2319.
2320.               public void MemberForceError()
2321.               {
2322.                   nodalForceError.Visibility = Visibility.Visible;
2323.               }
2324.
2325.               public void MemberForceErrorEdit()
2326.               {
2327.                   nodalForceErrorEdit.Visibility =
    Visibility.Visible;
2328.               }
2329.
2330.               public void CreateNode(object sender,
    RoutedEventArgs e)
2331.               {
2332.                   try
2333.                   {
2334.                       nodes.Add(new Nodes(nodes.Count,
    xCoordNode.Text, yCoordNode.Text, zCoordNode.Text, xDisp.IsChecked,
    yDisp.IsChecked, zDisp.IsChecked, xRot.IsChecked, yRot.IsChecked,
    zRot.IsChecked));
2335.
2336.                       SendStringToInterface(string.Format("Create
    Node&{0}&{1}&{2}&{3}&{4}", nodes.Count - 1, xCoordNode.Text,
```

```
   yCoordNode.Text, zCoordNode.Text, xDisp.IsChecked.GetValueOrDefault()
   || yDisp.IsChecked.GetValueOrDefault() ||
   zDisp.IsChecked.GetValueOrDefault() ||
   xRot.IsChecked.GetValueOrDefault() ||
   yRot.IsChecked.GetValueOrDefault() ||
   zRot.IsChecked.GetValueOrDefault()));
2337.                   }
2338.                   catch
2339.                   {
2340.
2341.                   }
2342.               }
2343.
2344.               public void EditNode(object sender, RoutedEventArgs
   e)
2345.               {
2346.                   if (xCoordNodeEdit.Text == "0 node selected" ||
   xCoordNodeEdit.Text == "Multiple nodes")
2347.                   {
2348.                       if (xCoordNodeEdit.Text == "Multiple nodes")
2349.                       {
2350.                           List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2351.
2352.                           for (int i = 1; i < tempList.Count; i++)
2353.                           {
2354.
   nodes[int.Parse(tempList[i])].restriction = new
   Restrictions(xDispEdit.IsChecked.GetValueOrDefault(),
   yDispEdit.IsChecked.GetValueOrDefault(),
   zDispEdit.IsChecked.GetValueOrDefault(),
   xRotEdit.IsChecked.GetValueOrDefault(),
   yRotEdit.IsChecked.GetValueOrDefault(),
   zRotEdit.IsChecked.GetValueOrDefault());
2355.                           }
2356.                       }
2357.
2358.                       //SendStringToInterface(string.Format("Edit
   Node Support&{0}", xDispEdit.IsChecked.GetValueOrDefault() ||
   yDispEdit.IsChecked.GetValueOrDefault() ||
   zDispEdit.IsChecked.GetValueOrDefault() ||
   xRotEdit.IsChecked.GetValueOrDefault() ||
   yRotEdit.IsChecked.GetValueOrDefault() ||
   zRotEdit.IsChecked.GetValueOrDefault()));
2359.                   }
2360.                   else
2361.                   {
2362.                       List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2363.
2364.                       nodes[int.Parse(tempList[1])].position = new
   Vector3(xCoordNodeEdit.Text, yCoordNodeEdit.Text,
   zCoordNodeEdit.Text);
2365.                       nodes[int.Parse(tempList[1])].restriction =
   new Restrictions(xDispEdit.IsChecked.GetValueOrDefault(),
   yDispEdit.IsChecked.GetValueOrDefault(),
   zDispEdit.IsChecked.GetValueOrDefault(),
   xRotEdit.IsChecked.GetValueOrDefault(),
   yRotEdit.IsChecked.GetValueOrDefault(),
   zRotEdit.IsChecked.GetValueOrDefault());
2366.
```

```
2367.                         //SendStringToInterface(string.Format("Edit
   Node&{0}&{1}&{2}&{3}", xCoordNodeEdit.Text, yCoordNodeEdit.Text,
   zCoordNodeEdit.Text, xDispEdit.IsChecked.GetValueOrDefault() ||
   yDispEdit.IsChecked.GetValueOrDefault() ||
   zDispEdit.IsChecked.GetValueOrDefault() ||
   xRotEdit.IsChecked.GetValueOrDefault() ||
   yRotEdit.IsChecked.GetValueOrDefault() ||
   zRotEdit.IsChecked.GetValueOrDefault()));
2368.                     }
2369.
2370.                     MassiveUpdate3DUI();
2371.                 }
2372.
2373.                 public void DeleteNode(object sender,
   RoutedEventArgs e)
2374.                 {
2375.                     try
2376.                     {
2377.                         List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2378.
2379.                         int count = 0;
2380.
2381.                         for (int i = 1; i < tempList.Count; i++)
2382.                         {
2383.                             for (int j = 0; j < members.Count; j++)
2384.                             {
2385.                                 bool remove = false;
2386.
2387.                                 if (members[j].n0 ==
   int.Parse(tempList[i]))
2388.                                 {
2389.                                     remove = true;
2390.                                 }
2391.                                 if (members[j].n1 ==
   int.Parse(tempList[i]))
2392.                                 {
2393.                                     remove = true;
2394.                                 }
2395.
2396.                                 if (remove)
2397.                                 {
2398.                                     members.RemoveAt(j);
2399.
2400.                                     for (int k = 0; k <
   memberForces.Count; k++)
2401.                                     {
2402.                                         if
   (memberForces[k].members.Contains(j))
2403.                                         {
2404.
   memberForces[k].members.Remove(j);
2405.
2406.                                             for (int l = 0; l <
   memberForces[k].members.Count; l++)
2407.                                             {
2408.                                                 if
   (memberForces[k].members[l] > j)
2409.                                                 {
2410.
   memberForces[k].members[l]--;
```

```
2411.                                                   }
2412.                                               }
2413.                                           }
2414.                                           else
2415.                                           {
2416.                                               for (int l = 0; l <
   memberForces[k].members.Count; l++)
2417.                                               {
2418.                                                   if
   (memberForces[k].members[l] > j)
2419.                                                   {
2420.
   memberForces[k].members[l]--;
2421.                                                   }
2422.                                               }
2423.                                           }
2424.
2425.
2426.                                       }
2427.
2428.                                       j--;
2429.                                   }
2430.                               }
2431.
2432.                               for (int j = 0; j < nodalForces.Count;
   j++)
2433.                               {
2434.                                   if
   (nodalForces[j].nodes.Contains(int.Parse(tempList[i])))
   nodalForces[j].nodes.Remove(int.Parse(tempList[i]));
2435.                               }
2436.
2437.                               for (int j = 0; j < memberForces.Count;
   j++)
2438.                               {
2439.                                   if (memberForces[j].members.Count ==
   0) memberForces.RemoveAt(j);
2440.                               }
2441.
2442.                               nodes.RemoveAt(int.Parse(tempList[i]) -
   count);
2443.                               count++;
2444.                           }
2445.
2446.                           for (int i = 0; i < nodes.Count; i++)
2447.                           {
2448.                               for (int j = 0; j < members.Count; j++)
2449.                               {
2450.                                   if (members[j].n0 ==
   nodes[i].number) members[j].n0 = i;
2451.                                   if (members[j].n1 ==
   nodes[i].number) members[j].n1 = i;
2452.                               }
2453.
2454.                               nodes[i].number = i;
2455.                           }
2456.
2457.                           for (int i = 0; i < members.Count; i++)
2458.                           {
2459.                               members[i].number = i;
2460.                           }
```

```
2461.
2462.                         for (int j = 0; j < nodalForces.Count; j++)
2463.                         {
2464.                             if (nodalForces[j].nodes.Count == 0)
2465.                             {
2466.                                 nodalForces.RemoveAt(j);
2467.                                 j--;
2468.                             }
2469.                         }
2470.
2471.                         xCoordNodeEdit.Text = "0 node selected";
2472.                         yCoordNodeEdit.Text = "0 node selected";
2473.                         zCoordNodeEdit.Text = "0 node selected";
2474.
2475.                         xCoordNodeEdit.IsEnabled = false;
2476.                         yCoordNodeEdit.IsEnabled = false;
2477.                         zCoordNodeEdit.IsEnabled = false;
2478.
2479.
   //SendStringToInterface(string.Format("Delete Node"));
2480.
2481.                         MassiveUpdate3DUI();
2482.                     }
2483.                     catch
2484.                     {
2485.
2486.                     }
2487.                 }
2488.
2489.                 public void MainHeaderSelectedMouse(object sender,
   MouseButtonEventArgs e)
2490.                 {
2491.                     MainHeader();
2492.                 }
2493.
2494.                 public void MainHeaderSelected(object sender,
   RoutedEventArgs e)
2495.                 {
2496.                     MainHeader();
2497.                 }
2498.
2499.                 public void MainHeader()
2500.                 {
2501.
   SendStringToInterface(string.Format("Mode&General"));
2502.
2503.                     if (GeometryMainHeader.IsSelected)
2504.                     {
2505.                         GeometryGeneralHeader.IsSelected = true;
2506.                     }
2507.
2508.                     if (PropertiesMainHeader.IsSelected)
2509.                     {
2510.                         PropertiesGeneralHeader.IsSelected = true;
2511.                     }
2512.
2513.                     if (ForcesMainHeader.IsSelected)
2514.                     {
2515.                         ForcesGeneralHeader.IsSelected = true;
2516.                     }
2517.
```

```
2518.                   if (SimulationMainHeader.IsSelected)
2519.                   {
2520.                       SimulationGeneral();
2521.                   }
2522.               }
2523.
2524.               public void SimulationGeneralHeaderSelected(object
   sender, RoutedEventArgs e)
2525.               {
2526.                   SimulationGeneral();
2527.               }
2528.
2529.               public void SimulationClear(object sender,
   RoutedEventArgs e)
2530.               {
2531.                   simulationSuccess = false;
2532.
2533.                   models.Clear();
2534.
2535.                   SimulationGeneral();
2536.               }
2537.
2538.               public void SimulationGeneral()
2539.               {
2540.                   SimulationGeneralHeader.IsSelected = true;
2541.
2542.                   simulationDayStart.Text = startDay.ToString();
2543.                   simulationDayEnd.Text = endDay.ToString();
2544.
2545.                   if (!simulationSuccess)
2546.                   {
2547.                       SimSep1.Visibility = Visibility.Collapsed;
2548.                       SimLabelDay.Visibility =
   Visibility.Collapsed;
2549.                       SimShowDay.Visibility =
   Visibility.Collapsed;
2550.                       SimUpdate.Visibility = Visibility.Collapsed;
2551.                       SimSep2.Visibility = Visibility.Collapsed;
2552.                       SimSlider.Visibility = Visibility.Collapsed;
2553.                       SimLabelObject.Visibility =
   Visibility.Collapsed;
2554.                       SimPlot.Visibility = Visibility.Collapsed;
2555.                       SimulationElementResults.Visibility =
   Visibility.Collapsed;
2556.                   }
2557.               }
2558.
2559.               public void AddNodeHeaderSelected(object sender,
   RoutedEventArgs e)
2560.               {
2561.                   SendStringToInterface(string.Format("Mode&Node
   Creation"));
2562.                   GeometryMainHeader.IsSelected = true;
2563.                   AddNodeHeader.IsSelected = true;
2564.               }
2565.
2566.               public void EditNodeHeaderSelected(object sender,
   RoutedEventArgs e)
2567.               {
2568.                   SendStringToInterface(string.Format("Mode&Node
   Editor"));
```

```
2569.                   GeometryMainHeader.IsSelected = true;
2570.                   EditNodeHeader.IsSelected = true;
2571.
2572.                   xCoordNodeEdit.Text = "0 node selected";
2573.                   yCoordNodeEdit.Text = "0 node selected";
2574.                   zCoordNodeEdit.Text = "0 node selected";
2575.
2576.                   xCoordNodeEdit.IsEnabled = false;
2577.                   yCoordNodeEdit.IsEnabled = false;
2578.                   zCoordNodeEdit.IsEnabled = false;
2579.
2580.                   xDispEdit.IsChecked = false;
2581.                   yDispEdit.IsChecked = false;
2582.                   zDispEdit.IsChecked = false;
2583.                   xRotEdit.IsChecked = false;
2584.                   yRotEdit.IsChecked = false;
2585.                   zRotEdit.IsChecked = false;
2586.               }
2587.
2588.               public void OptionsHeaderSelected(object sender,
   RoutedEventArgs e)
2589.               {
2590.
   SendStringToInterface(string.Format("Mode&Options"));
2591.                   OptionsHeader.IsSelected = true;
2592.               }
2593.
2594.               public void AddMaterialHeaderSelected(object sender,
   RoutedEventArgs e)
2595.               {
2596.
   SendStringToInterface(string.Format("Mode&Material Creation"));
2597.                   PropertiesMainHeader.IsSelected = true;
2598.                   AddMaterialHeader.IsSelected = true;
2599.
2600.                   fckText.IsEnabled = false;
2601.                   labelWrongMatP.Visibility =
   Visibility.Collapsed;
2602.               }
2603.
2604.               public void AddMemberHeaderSelected(object sender,
   RoutedEventArgs e)
2605.               {
2606.                   SendStringToInterface(string.Format("Mode&Member
   Creation"));
2607.                   GeometryMainHeader.IsSelected = true;
2608.                   AddMemberHeader.IsSelected = true;
2609.
2610.                   materialMemberComboBox.Items.Clear();
2611.                   propertyMemberComboBox.Items.Clear();
2612.
2613.                   for (int i = 0; i < materials.Count; i++)
2614.                   {
2615.
   materialMemberComboBox.Items.Add(materials[i].name);
2616.                   }
2617.
2618.                   for (int i = 0; i < properties.Count; i++)
2619.                   {
2620.
   propertyMemberComboBox.Items.Add(properties[i].name);
```

```
2621.                     }
2622.
2623.                     memberMatSecLabel.Content = "- Material /
   Selection";
2624.                     memberMatSecLabel.Foreground = Brushes.Red;
2625.
2626.                     memberFirstNodeLabel.Content = "- First Node";
2627.                     memberFirstNodeLabel.Foreground = Brushes.Red;
2628.
2629.                     memberSecondNodeLabel.Content = "- Second Node";
2630.                     memberSecondNodeLabel.Foreground = Brushes.Red;
2631.                 }
2632.
2633.                 public void AddMemberMaterialComboBoxChanged(object
   sender, SelectionChangedEventArgs e)
2634.                 {
2635.                     if (materialMemberComboBox.SelectedItem != null)
2636.                     {
2637.                         if (propertyMemberComboBox.SelectedItem !=
   null)
2638.                         {
2639.                             memberMatSecLabel.Content = "-
   Material/Selection OK";
2640.                             memberMatSecLabel.Foreground =
   Brushes.Black;
2641.                         }
2642.                     }
2643.                 }
2644.
2645.                 public void AddMemberPropertyComboBoxChanged(object
   sender, SelectionChangedEventArgs e)
2646.                 {
2647.                     if (propertyMemberComboBox.SelectedItem != null)
2648.                     {
2649.                         if (materialMemberComboBox.SelectedItem !=
   null)
2650.                         {
2651.                             memberMatSecLabel.Content = "-
   Material/Selection OK";
2652.                             memberMatSecLabel.Foreground =
   Brushes.Black;
2653.                         }
2654.                     }
2655.                 }
2656.
2657.                 public void EditMemberHeaderSelected(object sender,
   RoutedEventArgs e)
2658.                 {
2659.                     SendStringToInterface(string.Format("Mode&Member
   Editor"));
2660.                     GeometryMainHeader.IsSelected = true;
2661.                     EditMemberHeader.IsSelected = true;
2662.
2663.                     materialMemberEditComboBox.Items.Clear();
2664.                     propertyMemberEditComboBox.Items.Clear();
2665.
2666.                     for (int i = 0; i < materials.Count; i++)
2667.                     {
2668.
   materialMemberEditComboBox.Items.Add(materials[i].name);
2669.                     }
```

```
2670.
2671.                   for (int i = 0; i < properties.Count; i++)
2672.                   {
2673.
   propertyMemberEditComboBox.Items.Add(properties[i].name);
2674.                   }
2675.
2676.                   //SendStringToInterface(string.Format("Show
   Days&All"));
2677.               }
2678.
2679.               public void SplitMemberHeaderSelected(object sender,
   RoutedEventArgs e)
2680.               {
2681.                   SendStringToInterface(string.Format("Mode&Split
   Member"));
2682.                   GeometryMainHeader.IsSelected = true;
2683.                   SplitMemberHeader.IsSelected = true;
2684.               }
2685.
2686.               public void EditMember(object sender,
   RoutedEventArgs e)
2687.               {
2688.                   try
2689.                   {
2690.                       List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2691.
2692.                       if (tempList.Count > 1)
2693.                       {
2694.                           for (int i = 1; i < tempList.Count; i++)
2695.                           {
2696.
   members[int.Parse(tempList[i])].material =
   materialMemberEditComboBox.SelectedItem.ToString();
2697.
   members[int.Parse(tempList[i])].section =
   propertyMemberEditComboBox.SelectedItem.ToString();
2698.
   members[int.Parse(tempList[i])].startDay =
   int.Parse(dayMemberStartEditText.Text);
2699.
   members[int.Parse(tempList[i])].endDay =
   int.Parse(dayMemberEndEditText.Text);
2700.                           }
2701.                       }
2702.
2703.                       MassiveUpdate3DUI();
2704.                   }
2705.                   catch
2706.                   {
2707.
2708.                   }
2709.               }
2710.
2711.               public void DeleteMember(object sender,
   RoutedEventArgs e)
2712.               {
2713.                   try
2714.                   {
```

```
2715.                         List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2716.
2717.                         if (tempList.Count > 1)
2718.                         {
2719.                             int count = 0;
2720.                             for (int i = 1; i < tempList.Count; i++)
2721.                             {
2722.
   members.RemoveAt(int.Parse(tempList[i]) - count);
2723.
2724.                                 for (int j = 0; j <
   memberForces.Count; j++)
2725.                                 {
2726.                                     if
   (memberForces[j].members.Contains(int.Parse(tempList[i]) - count))
2727.                                     {
2728.
   memberForces[j].members.Remove(int.Parse(tempList[i]) - count);
2729.
2730.                                         for (int k = 0; k <
   memberForces[j].members.Count; k++)
2731.                                         {
2732.                                             if
   (memberForces[j].members[k] > int.Parse(tempList[i]) - count)
2733.                                             {
2734.
   memberForces[j].members[k]--;
2735.                                             }
2736.                                         }
2737.                                     }
2738.                                     else
2739.                                     {
2740.                                         for (int k = 0; k <
   memberForces[j].members.Count; k++)
2741.                                         {
2742.                                             if
   (memberForces[j].members[k] > int.Parse(tempList[i]) - count)
2743.                                             {
2744.
   memberForces[j].members[k]--;
2745.                                             }
2746.                                         }
2747.                                     }
2748.                                 }
2749.
2750.                                 count++;
2751.                             }
2752.                         }
2753.
2754.                         for (int j = 0; j < memberForces.Count; j++)
2755.                         {
2756.                             if (memberForces[j].members.Count == 0)
   memberForces.RemoveAt(j);
2757.                         }
2758.
2759.                         for (int i = 0; i < members.Count; i++)
2760.                         {
2761.                             members[i].number = i;
2762.                         }
2763.
```

```
2764.                         MassiveUpdate3DUI();
2765.                     }
2766.                     catch
2767.                     {
2768.
2769.                     }
2770.                 }
2771.
2772.                 private void SplitMember(object sender,
   RoutedEventArgs e)
2773.                 {
2774.                     try
2775.                     {
2776.                         List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2777.
2778.                         if (tempList[0] == "Selected Split:" &&
   tempList.Count > 1)
2779.                         {
2780.                             int splitCount =
   int.Parse(splitPieces.Text);
2781.
2782.                             if (splitCount < 1)
2783.                             {
2784.                                 splitCount = 1;
2785.                                 splitPieces.Text = "1";
2786.                             }
2787.
2788.                             int originalMember =
   int.Parse(tempList[1]);
2789.
2790.                             double diffX =
   nodes[members[originalMember].n1].position.x -
   nodes[members[originalMember].n0].position.x;
2791.                             double diffY =
   nodes[members[originalMember].n1].position.y -
   nodes[members[originalMember].n0].position.y;
2792.                             double diffZ =
   nodes[members[originalMember].n1].position.z -
   nodes[members[originalMember].n0].position.z;
2793.
2794.                             diffX = diffX / splitCount;
2795.                             diffY = diffY / splitCount;
2796.                             diffZ = diffZ / splitCount;
2797.
2798.                             int originalEndNode =
   members[originalMember].n1;
2799.                             int lastNode = 0;
2800.
2801.                             for (int i = 1; i <= splitCount; i++)
2802.                             {
2803.                                 if (i == 1)
2804.                                 {
2805.                                     nodes.Add(new Nodes(nodes.Count,
2806.
   (nodes[members[originalMember].n0].position.x + i *
   diffX).ToString(),
2807.
   (nodes[members[originalMember].n0].position.y + i *
   diffY).ToString(),
```

```
2808.
   (nodes[members[originalMember].n0].position.z + i *
   diffZ).ToString(),
2809.                                          false, false, false, false,
   false, false));
2810.
2811.                                      lastNode = nodes.Count - 1;
2812.
2813.                                      members[originalMember].n1 =
   lastNode;
2814.                                  }
2815.                                  else if (i == splitCount)
2816.                                  {
2817.                                      members.Add(new
   Member(members.Count, lastNode, originalEndNode,
   members[originalMember].material, members[originalMember].section,
2818.
   members[originalMember].startDay.ToString(),
   members[originalMember].endDay.ToString()));
2819.
2820.
   UpdateMemberForces(originalMember, members.Count - 1);
2821.                                  }
2822.                                  else
2823.                                  {
2824.                                      nodes.Add(new Nodes(nodes.Count,
2825.
   (nodes[members[originalMember].n0].position.x + i *
   diffX).ToString(),
2826.
   (nodes[members[originalMember].n0].position.y + i *
   diffY).ToString(),
2827.
   (nodes[members[originalMember].n0].position.z + i *
   diffZ).ToString(),
2828.                                          false, false, false, false,
   false, false));
2829.
2830.                                      members.Add(new
   Member(members.Count, lastNode, nodes.Count - 1,
   members[originalMember].material, members[originalMember].section,
2831.
   members[originalMember].startDay.ToString(),
   members[originalMember].endDay.ToString()));
2832.
2833.                                      lastNode = nodes.Count - 1;
2834.
2835.
   UpdateMemberForces(originalMember, members.Count - 1);
2836.                                  }
2837.                              }
2838.                          }
2839.
2840.                          MassiveUpdate3DUI();
2841.                      }
2842.                      catch
2843.                      {
2844.
2845.                      }
2846.                  }
2847.
```

```
2848.                public void NewFile(object sender, RoutedEventArgs
   e)
2849.                {
2850.                    try
2851.                    {
2852.                        members.Clear();
2853.                        nodes.Clear();
2854.                        materials.Clear();
2855.                        properties.Clear();
2856.                        nodalForces.Clear();
2857.                        memberForces.Clear();
2858.
2859.                        PropertiesMainHeader.IsSelected = true;
2860.
2861.                        simulationSuccess = false;
2862.                        lastSimulatedSelected = "";
2863.
2864.
2865.                        MassiveUpdate3DUI();
2866.                    }
2867.                    catch
2868.                    {
2869.
2870.                    }
2871.                }
2872.
2873.                public void Exit(object sender, RoutedEventArgs e)
2874.                {
2875.                    System.Windows.Application.Current.Shutdown();
2876.                }
2877.
2878.
2879.                public void IsTimeDependentChecked(object sender,
   RoutedEventArgs e)
2880.                {
2881.                    fckText.IsEnabled = true;
2882.                }
2883.
2884.                public void IsTimeDependentUnchecked(object sender,
   RoutedEventArgs e)
2885.                {
2886.                    fckText.IsEnabled = false;
2887.                    fckText.Text = "";
2888.                }
2889.
2890.                public void IsTimeDependentEditChecked(object
   sender, RoutedEventArgs e)
2891.                {
2892.                    fckTextEdit.IsEnabled = true;
2893.                }
2894.
2895.                public void IsTimeDependentEditUnchecked(object
   sender, RoutedEventArgs e)
2896.                {
2897.                    fckTextEdit.IsEnabled = false;
2898.                    fckTextEdit.Text = "";
2899.                }
2900.
2901.                public void EntireStructureChecked(object sender,
   RoutedEventArgs e)
2902.                {
```

```
2903.                   if (!finishedLoading) return;
2904.                   structureDays.IsEnabled = false;
2905.               }
2906.
2907.               public void EntireStructureUnchecked(object sender,
   RoutedEventArgs e)
2908.               {
2909.                   if (!finishedLoading) return;
2910.                   structureDays.IsEnabled = true;
2911.               }
2912.
2913.               public void UpdateStructureTime(object sender,
   RoutedEventArgs e)
2914.               {
2915.                   if (structureDays.IsEnabled)
2916.                   {
2917.                       try
2918.                       {
2919.                           int day = int.Parse(structureDays.Text);
2920.
   SendStringToInterface(string.Format("Show Days&{0}", day));
2921.                       }
2922.                       catch
2923.                       {
2924.
2925.                       }
2926.                   }
2927.                   else
2928.                   {
2929.                       SendStringToInterface(string.Format("Show
   Days&All"));
2930.                   }
2931.               }
2932.
2933.               public void EditMaterialHeaderSelected(object
   sender, RoutedEventArgs e)
2934.               {
2935.
   SendStringToInterface(string.Format("Mode&Material Editor"));
2936.                   EditMaterialHeader.IsSelected = true;
2937.                   PropertiesMainHeader.IsSelected = true;
2938.                   labelWrongMatPEdit.Visibility =
   Visibility.Collapsed;
2939.
2940.                   matComboBox.Items.Clear();
2941.
2942.                   for (int i = 0; i < materials.Count; i++)
2943.                   {
2944.                       matComboBox.Items.Add(materials[i].name);
2945.                   }
2946.               }
2947.
2948.               public void
   EditMaterialComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
2949.               {
2950.                   if (matComboBox.SelectedItem != null)
2951.                   {
2952.                       matNameEdit.Text = materials.Where(item =>
   item.name == matComboBox.SelectedItem.ToString()).First().name;
```

```
2953.                         matEEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().E.ToString();
2954.                         matGEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().G.ToString();
2955.                         matVEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().v.ToString();
2956.                         fckTextEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().fck.ToString();
2957.
2958.                         isTimeDependentEdit.IsChecked =
   materials.Where(item => item.name ==
   matComboBox.SelectedItem.ToString()).First().timeDependent;
2959.
2960.
2961.                         matNameEdit.IsEnabled = true;
2962.                         matEEdit.IsEnabled = true;
2963.                         matGEdit.IsEnabled = true;
2964.                         matVEdit.IsEnabled = true;
2965.                         isTimeDependentEdit.IsEnabled = true;
2966.                         fckTextEdit.IsEnabled =
   isTimeDependentEdit.IsChecked.GetValueOrDefault();
2967.
2968.                         saveEditMaterial.IsEnabled = true;
2969.                         deleteEditMaterial.IsEnabled = true;
2970.                         calcGEdit.IsEnabled = true;
2971.                     }
2972.                     else
2973.                     {
2974.                         matNameEdit.Text = "";
2975.                         matEEdit.Text = "";
2976.                         matGEdit.Text = "";
2977.                         matVEdit.Text = "";
2978.                         fckTextEdit.Text = "";
2979.
2980.                         matNameEdit.IsEnabled = false;
2981.                         matEEdit.IsEnabled = false;
2982.                         matGEdit.IsEnabled = false;
2983.                         matVEdit.IsEnabled = false;
2984.                         isTimeDependentEdit.IsChecked = false;
2985.                         isTimeDependentEdit.IsEnabled = false;
2986.
2987.                         saveEditMaterial.IsEnabled = false;
2988.                         deleteEditMaterial.IsEnabled = false;
2989.                         calcGEdit.IsEnabled = false;
2990.                     }
2991.                 }
2992.
2993.                 public void SaveEditMaterial(object sender,
   RoutedEventArgs e)
2994.                 {
2995.                     try
2996.                     {
2997.                         string matName =
   matComboBox.SelectedItem.ToString();
2998.
```

```
2999.                         if (matName != matNameEdit.Text &&
   materials.Where(item => item.name == matNameEdit.Text).ToList().Count
   != 0)
3000.                         {
3001.                             labelWrongMatPEdit.Content = "Wrong
   Properties!";
3002.                             labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3003.                         }
3004.                         else
3005.                         {
3006.                             materials.Where(item => item.name ==
   matName).First().E = double.Parse(matEEdit.Text);
3007.                             materials.Where(item => item.name ==
   matName).First().G = double.Parse(matGEdit.Text);
3008.                             materials.Where(item => item.name ==
   matName).First().v = double.Parse(matVEdit.Text);
3009.                             materials.Where(item => item.name ==
   matName).First().timeDependent =
   isTimeDependentEdit.IsChecked.GetValueOrDefault();
3010.                             if (fckTextEdit.Text == "")
   fckTextEdit.Text = "0";
3011.                             materials.Where(item => item.name ==
   matName).First().fck = double.Parse(fckTextEdit.Text);
3012.                             materials.Where(item => item.name ==
   matName).First().name = matNameEdit.Text;
3013.
3014.                             for (int i = 0; i < members.Count; i++)
3015.                             {
3016.                                 if (members[i].material ==
   matComboBox.SelectedItem.ToString()) members[i].material =
   matNameEdit.Text;
3017.                             }
3018.
3019.                             matComboBox.Items.Clear();
3020.
3021.                             for (int i = 0; i < materials.Count;
   i++)
3022.                             {
3023.
   matComboBox.Items.Add(materials[i].name);
3024.                             }
3025.
3026.                             labelWrongMatPEdit.Content = "Material
   Saved!";
3027.                             labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3028.                         }
3029.                     }
3030.                     catch
3031.                     {
3032.
3033.                     }
3034.                 }
3035.
3036.                 public void DeleteEditMaterial(object sender,
   RoutedEventArgs e)
3037.                 {
3038.                     try
3039.                     {
```

```
3040.                         materials.Remove(materials.Where(item =>
   item.name == matComboBox.SelectedItem.ToString()).First());
3041.
3042.                         matComboBox.Items.Clear();
3043.
3044.                         for (int i = 0; i < materials.Count; i++)
3045.                         {
3046.
   matComboBox.Items.Add(materials[i].name);
3047.                         }
3048.
3049.                         labelWrongMatPEdit.Content = "Material
   Removed!";
3050.                         labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3051.                     }
3052.                     catch
3053.                     {
3054.
3055.                     }
3056.                 }
3057.
3058.                 public void AddPropertyHeaderSelected(object sender,
   RoutedEventArgs e)
3059.                 {
3060.
   SendStringToInterface(string.Format("Mode&Property Creation"));
3061.                     AddPropertiesHeader.IsSelected = true;
3062.                     PropertiesMainHeader.IsSelected = true;
3063.                     labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3064.                     labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3065.                     labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3066.                 }
3067.
3068.                 public void
   EditPropertyComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
3069.                 {
3070.                     if (!finishedLoading) return;
3071.
3072.                     if (propSecComboBox.SelectedIndex == 0)
3073.                     {
3074.                         propNoneSep.Visibility = Visibility.Visible;
3075.                         propNoneSave.Visibility =
   Visibility.Visible;
3076.                         labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3077.
3078.                         propDiamLabel.Visibility =
   Visibility.Collapsed;
3079.                         propDiam.Visibility = Visibility.Collapsed;
3080.                         propDiamCalc.Visibility =
   Visibility.Collapsed;
3081.                         propDiamSep.Visibility =
   Visibility.Collapsed;
3082.                         propDiamSave.Visibility =
   Visibility.Collapsed;
```

```
3083.                       labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3084.
3085.                       propHLabel.Visibility =
   Visibility.Collapsed;
3086.                       propH.Visibility = Visibility.Collapsed;
3087.                       propBLabel.Visibility =
   Visibility.Collapsed;
3088.                       propB.Visibility = Visibility.Collapsed;
3089.                       propRetCalc.Visibility =
   Visibility.Collapsed;
3090.                       propRetSep.Visibility =
   Visibility.Collapsed;
3091.                       propRetSave.Visibility =
   Visibility.Collapsed;
3092.                       labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3093.                   }
3094.
3095.                   if (propSecComboBox.SelectedIndex == 1)
3096.                   {
3097.                       propNoneSep.Visibility =
   Visibility.Collapsed;
3098.                       propNoneSave.Visibility =
   Visibility.Collapsed;
3099.                       labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3100.
3101.                       propDiamLabel.Visibility =
   Visibility.Visible;
3102.                       propDiam.Visibility = Visibility.Visible;
3103.                       propDiamCalc.Visibility =
   Visibility.Visible;
3104.                       propDiamSep.Visibility = Visibility.Visible;
3105.                       propDiamSave.Visibility =
   Visibility.Visible;
3106.                       labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3107.
3108.                       propHLabel.Visibility =
   Visibility.Collapsed;
3109.                       propH.Visibility = Visibility.Collapsed;
3110.                       propBLabel.Visibility =
   Visibility.Collapsed;
3111.                       propB.Visibility = Visibility.Collapsed;
3112.                       propRetCalc.Visibility =
   Visibility.Collapsed;
3113.                       propRetSep.Visibility =
   Visibility.Collapsed;
3114.                       propRetSave.Visibility =
   Visibility.Collapsed;
3115.                       labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3116.                   }
3117.
3118.                   if (propSecComboBox.SelectedIndex == 2)
3119.                   {
3120.                       propNoneSep.Visibility =
   Visibility.Collapsed;
3121.                       propNoneSave.Visibility =
   Visibility.Collapsed;
```

```
3122.                         labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3123.
3124.                         propDiamLabel.Visibility =
   Visibility.Collapsed;
3125.                         propDiam.Visibility = Visibility.Collapsed;
3126.                         propDiamCalc.Visibility =
   Visibility.Collapsed;
3127.                         propDiamSep.Visibility =
   Visibility.Collapsed;
3128.                         propDiamSave.Visibility =
   Visibility.Collapsed;
3129.                         labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3130.
3131.                         propHLabel.Visibility = Visibility.Visible;
3132.                         propH.Visibility = Visibility.Visible;
3133.                         propBLabel.Visibility = Visibility.Visible;
3134.                         propB.Visibility = Visibility.Visible;
3135.                         propRetCalc.Visibility = Visibility.Visible;
3136.                         propRetSep.Visibility = Visibility.Visible;
3137.                         propRetSave.Visibility = Visibility.Visible;
3138.                         labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3139.                     }
3140.                 }
3141.
3142.                 public void CalculateCircularProperty(object sender,
   RoutedEventArgs e)
3143.                 {
3144.                     try
3145.                     {
3146.                         double A = Math.PI *
   Math.Pow(double.Parse(propDiam.Text) / 2, 2);
3147.                         double Ix = (Math.PI / 4) *
   Math.Pow(double.Parse(propDiam.Text) / 2, 4);
3148.                         double Iy = Ix;
3149.                         double Iz = (Math.PI / 2) *
   Math.Pow(double.Parse(propDiam.Text) / 2, 4);
3150.
3151.                         propA.Text = A.ToString();
3152.                         propIx.Text = Ix.ToString();
3153.                         propIy.Text = Iy.ToString();
3154.                         propIz.Text = Iz.ToString();
3155.                     }
3156.                     catch
3157.                     {
3158.
3159.                     }
3160.                 }
3161.
3162.                 public void CalculateRetangularProperty(object
   sender, RoutedEventArgs e)
3163.                 {
3164.                     try
3165.                     {
3166.                         double A = double.Parse(propH.Text) *
   double.Parse(propB.Text);
3167.                         double Ix = double.Parse(propB.Text) *
   Math.Pow(double.Parse(propH.Text), 3) / 12;
```

```
3168.                         double Iy = double.Parse(propH.Text) *
   Math.Pow(double.Parse(propB.Text), 3) / 12;
3169.                         double Iz = Ix + Iy;
3170.
3171.                         propA.Text = A.ToString();
3172.                         propIx.Text = Ix.ToString();
3173.                         propIy.Text = Iy.ToString();
3174.                         propIz.Text = Iz.ToString();
3175.                     }
3176.                     catch
3177.                     {
3178.
3179.                     }
3180.                 }
3181.
3182.                 public void CreateProperty(object sender,
   RoutedEventArgs e)
3183.                 {
3184.                     if (propName.Text != "" && propA.Text != "" &&
   propIx.Text != "" && propIy.Text != "" && propIz.Text != "")
3185.                     {
3186.                         try
3187.                         {
3188.                             if (properties.Where(item => item.name
   == propName.Text).ToList().Count == 0)
3189.                             {
3190.                                 properties.Add(new
   Property(propName.Text, propA.Text, propIx.Text, propIy.Text,
   propIz.Text, propSecComboBox.SelectedIndex.ToString(), propDiam.Text,
   propH.Text, propB.Text));
3191.
3192.                                 propName.Text = "";
3193.                                 propA.Text = "";
3194.                                 propIx.Text = "";
3195.                                 propIy.Text = "";
3196.                                 propIz.Text = "";
3197.                                 propSecComboBox.SelectedIndex = 0;
3198.                                 propDiam.Text = "";
3199.                                 propH.Text = "";
3200.                                 propB.Text = "";
3201.
3202.                                 labelWrongPropNone.Content =
   "Property Saved!";
3203.                                 labelWrongPropNone.Visibility =
   Visibility.Visible;
3204.
3205.                             }
3206.                             else
3207.                             {
3208.                                 if (propSecComboBox.SelectedIndex ==
   0)
3209.                                 {
3210.                                     labelWrongPropNone.Content =
   "Wrong Properties!";
3211.                                     labelWrongPropNone.Visibility =
   Visibility.Visible;
3212.                                 }
3213.                                 if (propSecComboBox.SelectedIndex ==
   1)
3214.                                 {
```

```
3215.                                   labelWrongPropDiam.Content =
   "Wrong Properties!";
3216.                                   labelWrongPropDiam.Visibility =
   Visibility.Visible;
3217.                               }
3218.                               if (propSecComboBox.SelectedIndex ==
   2)
3219.                               {
3220.                                   labelWrongPropRet.Content =
   "Wrong Properties!";
3221.                                   labelWrongPropRet.Visibility =
   Visibility.Visible;
3222.                               }
3223.                           }
3224.                       }
3225.                       catch
3226.                       {
3227.                           if (propSecComboBox.SelectedIndex == 0)
3228.                           {
3229.                               labelWrongPropNone.Content = "Wrong
   Properties!";
3230.                               labelWrongPropNone.Visibility =
   Visibility.Visible;
3231.                           }
3232.                           if (propSecComboBox.SelectedIndex == 1)
3233.                           {
3234.                               labelWrongPropDiam.Content = "Wrong
   Properties!";
3235.                               labelWrongPropDiam.Visibility =
   Visibility.Visible;
3236.                           }
3237.                           if (propSecComboBox.SelectedIndex == 2)
3238.                           {
3239.                               labelWrongPropRet.Content = "Wrong
   Properties!";
3240.                               labelWrongPropRet.Visibility =
   Visibility.Visible;
3241.                           }
3242.                       }
3243.                   }
3244.                   else
3245.                   {
3246.                       if (propSecComboBox.SelectedIndex == 0)
3247.                       {
3248.                           labelWrongPropNone.Content = "Wrong
   Properties!";
3249.                           labelWrongPropNone.Visibility =
   Visibility.Visible;
3250.                       }
3251.                       if (propSecComboBox.SelectedIndex == 1)
3252.                       {
3253.                           labelWrongPropDiam.Content = "Wrong
   Properties!";
3254.                           labelWrongPropDiam.Visibility =
   Visibility.Visible;
3255.                       }
3256.                       if (propSecComboBox.SelectedIndex == 2)
3257.                       {
3258.                           labelWrongPropRet.Content = "Wrong
   Properties!";
```

```
3259.                           labelWrongPropRet.Visibility =
   Visibility.Visible;
3260.                       }
3261.                   }
3262.               }
3263.
3264.               public void EditPropertyHeaderSelected(object
   sender, RoutedEventArgs e)
3265.               {
3266.
   SendStringToInterface(string.Format("Mode&Property Editor"));
3267.                   EditPropertiesHeader.IsSelected = true;
3268.                   PropertiesMainHeader.IsSelected = true;
3269.                   labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3270.                   labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3271.                   labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3272.
3273.                   editComboBox.Items.Clear();
3274.
3275.                   for (int i = 0; i < properties.Count; i++)
3276.                   {
3277.                       editComboBox.Items.Add(properties[i].name);
3278.                   }
3279.
3280.                   propNameEdit.Text = "";
3281.                   propAEdit.Text = "";
3282.                   propIxEdit.Text = "";
3283.                   propIyEdit.Text = "";
3284.                   propIzEdit.Text = "";
3285.
3286.                   propNameEdit.IsEnabled = false;
3287.
3288.                   propAEdit.IsEnabled = false;
3289.                   propIxEdit.IsEnabled = false;
3290.                   propIyEdit.IsEnabled = false;
3291.                   propIzEdit.IsEnabled = false;
3292.
3293.                   editComboBox.IsEnabled = true;
3294.
3295.                   propSecComboBoxEdit.IsEnabled = false;
3296.                   propSecComboBoxEdit.SelectedIndex = 0;
3297.
3298.                   propNoneSaveEdit.IsEnabled = false;
3299.                   propNoneDeleteEdit.IsEnabled = false;
3300.
3301.                   propNoneSaveEdit.Visibility =
   Visibility.Visible;
3302.                   propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3303.                   propNoneSepEdit.Visibility = Visibility.Visible;
3304.                   labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3305.
3306.                   propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3307.                   propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
```

```
3308.                   propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3309.                   propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3310.                   propDiamEdit.Visibility = Visibility.Collapsed;
3311.                   labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3312.
3313.                   propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3314.                   propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3315.                   propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3316.                   propRetSep.Visibility = Visibility.Collapsed;
3317.                   propHEdit.Visibility = Visibility.Collapsed;
3318.                   propBEdit.Visibility = Visibility.Collapsed;
3319.                   labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3320.               }
3321.
3322.               public void
   EditPropertyListComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
3323.               {
3324.                   if (editComboBox.SelectedItem != null)
3325.                   {
3326.                       propNameEdit.Text = properties.Where(item =>
   item.name == editComboBox.SelectedItem.ToString()).First().name;
3327.                       propAEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().A.ToString();
3328.                       propIxEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Ix.ToString();
3329.                       propIyEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Iy.ToString();
3330.                       propIzEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Iz.ToString();
3331.
3332.                       if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "None")
3333.                       {
3334.                           propSecComboBoxEdit.IsEnabled = true;
3335.                           propSecComboBoxEdit.SelectedIndex = 0;
3336.
3337.                           propNoneSaveEdit.IsEnabled = true;
3338.                           propNoneDeleteEdit.IsEnabled = true;
3339.
3340.                           propNoneSaveEdit.Visibility =
   Visibility.Visible;
3341.                           propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3342.                           propNoneSepEdit.Visibility =
   Visibility.Visible;
3343.                           labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3344.
```

```
3345.                           propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3346.                           propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3347.                           propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3348.                           propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3349.                           propDiamEdit.Visibility =
   Visibility.Collapsed;
3350.                           propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3351.                           labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3352.
3353.                           propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3354.                           propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3355.                           propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3356.                           propRetSepEdit.Visibility =
   Visibility.Collapsed;
3357.                           propHEdit.Visibility =
   Visibility.Collapsed;
3358.                           propBEdit.Visibility =
   Visibility.Collapsed;
3359.                           propHLabelEdit.Visibility =
   Visibility.Collapsed;
3360.                           propBLabelEdit.Visibility =
   Visibility.Collapsed;
3361.                           labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3362.
3363.                       }
3364.                       if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "Circular")
3365.                       {
3366.                           propSecComboBoxEdit.IsEnabled = true;
3367.                           propSecComboBoxEdit.SelectedIndex = 1;
3368.
3369.                           propNoneSaveEdit.IsEnabled = false;
3370.                           propNoneDeleteEdit.IsEnabled = false;
3371.
3372.                           propDiamCalcEdit.IsEnabled = true;
3373.                           propDiamSaveEdit.IsEnabled = true;
3374.                           propDiamDeleteEdit.IsEnabled = true;
3375.                           propDiamEdit.IsEnabled = true;
3376.                           propDiamSepEdit.IsEnabled = true;
3377.
3378.                           propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3379.                           propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3380.                           propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3381.                           labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3382.
3383.                           propDiamCalcEdit.Visibility =
   Visibility.Visible;
```

```
3384.                           propDiamSaveEdit.Visibility =
   Visibility.Visible;
3385.                           propDiamDeleteEdit.Visibility =
   Visibility.Visible;
3386.                           propDiamSepEdit.Visibility =
   Visibility.Visible;
3387.                           propDiamEdit.Visibility =
   Visibility.Visible;
3388.                           propDiamLabelEdit.Visibility =
   Visibility.Visible;
3389.                           labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3390.
3391.                           propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3392.                           propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3393.                           propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3394.                           propRetSepEdit.Visibility =
   Visibility.Collapsed;
3395.                           propHEdit.Visibility =
   Visibility.Collapsed;
3396.                           propBEdit.Visibility =
   Visibility.Collapsed;
3397.                           propHLabelEdit.Visibility =
   Visibility.Collapsed;
3398.                           propBLabelEdit.Visibility =
   Visibility.Collapsed;
3399.                           labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3400.                       }
3401.                       if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "Retangular")
3402.                       {
3403.                           propSecComboBoxEdit.IsEnabled = true;
3404.                           propSecComboBoxEdit.SelectedIndex = 2;
3405.
3406.                           propNoneSaveEdit.IsEnabled = false;
3407.                           propNoneDeleteEdit.IsEnabled = false;
3408.
3409.                           propRetCalcEdit.IsEnabled = true;
3410.                           propRetSaveEdit.IsEnabled = true;
3411.                           propRetDeleteEdit.IsEnabled = true;
3412.                           propHEdit.IsEnabled = true;
3413.                           propBEdit.IsEnabled = true;
3414.                           propRetSepEdit.IsEnabled = true;
3415.
3416.                           propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3417.                           propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3418.                           propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3419.                           labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3420.
3421.                           propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3422.                           propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
```

```
3423.                           propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3424.                           propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3425.                           propDiamEdit.Visibility =
   Visibility.Collapsed;
3426.                           propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3427.                           labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3428.
3429.                           propRetCalcEdit.Visibility =
   Visibility.Visible;
3430.                           propRetSaveEdit.Visibility =
   Visibility.Visible;
3431.                           propRetDeleteEdit.Visibility =
   Visibility.Visible;
3432.                           propRetSepEdit.Visibility =
   Visibility.Visible;
3433.                           propHEdit.Visibility =
   Visibility.Visible;
3434.                           propBEdit.Visibility =
   Visibility.Visible;
3435.                           propHLabelEdit.Visibility =
   Visibility.Visible;
3436.                           propBLabelEdit.Visibility =
   Visibility.Visible;
3437.                           labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3438.                       }
3439.
3440.                       propDiamEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().d.ToString();
3441.                       propHEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().h.ToString();
3442.                       propBEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().b.ToString();
3443.
3444.                       propNameEdit.IsEnabled = true;
3445.
3446.                       propAEdit.IsEnabled = true;
3447.                       propIxEdit.IsEnabled = true;
3448.                       propIyEdit.IsEnabled = true;
3449.                       propIzEdit.IsEnabled = true;
3450.
3451.                       editComboBox.IsEnabled = true;
3452.                   }
3453.                   else
3454.                   {
3455.
3456.                   }
3457.               }
3458.
3459.               public void
   PropSecComboBoxEditSelectionChanged(object sender,
   SelectionChangedEventArgs e)
3460.               {
3461.                   if (!finishedLoading) return;
```

```
3462.
3463.                     if (propSecComboBoxEdit.SelectedIndex == 0)
3464.                     {
3465.                         propNoneSaveEdit.IsEnabled = true;
3466.                         propNoneDeleteEdit.IsEnabled = true;
3467.
3468.                         propNoneSaveEdit.Visibility =
   Visibility.Visible;
3469.                         propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3470.                         propNoneSepEdit.Visibility =
   Visibility.Visible;
3471.
3472.                         propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3473.                         propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3474.                         propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3475.                         propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3476.                         propDiamEdit.Visibility =
   Visibility.Collapsed;
3477.                         propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3478.
3479.                         propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3480.                         propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3481.                         propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3482.                         propRetSepEdit.Visibility =
   Visibility.Collapsed;
3483.                         propHEdit.Visibility = Visibility.Collapsed;
3484.                         propBEdit.Visibility = Visibility.Collapsed;
3485.                         propHLabelEdit.Visibility =
   Visibility.Collapsed;
3486.                         propBLabelEdit.Visibility =
   Visibility.Collapsed;
3487.                     }
3488.
3489.                     if (propSecComboBoxEdit.SelectedIndex == 1)
3490.                     {
3491.                         propNoneSaveEdit.IsEnabled = false;
3492.                         propNoneDeleteEdit.IsEnabled = false;
3493.
3494.                         propDiamCalcEdit.IsEnabled = true;
3495.                         propDiamSaveEdit.IsEnabled = true;
3496.                         propDiamDeleteEdit.IsEnabled = true;
3497.                         propDiamEdit.IsEnabled = true;
3498.                         propDiamSepEdit.IsEnabled = true;
3499.
3500.                         propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3501.                         propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3502.                         propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3503.
```

```
3504.                       propDiamCalcEdit.Visibility =
   Visibility.Visible;
3505.                       propDiamSaveEdit.Visibility =
   Visibility.Visible;
3506.                       propDiamDeleteEdit.Visibility =
   Visibility.Visible;
3507.                       propDiamSepEdit.Visibility =
   Visibility.Visible;
3508.                       propDiamEdit.Visibility =
   Visibility.Visible;
3509.                       propDiamLabelEdit.Visibility =
   Visibility.Visible;
3510.
3511.                       propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3512.                       propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3513.                       propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3514.                       propRetSepEdit.Visibility =
   Visibility.Collapsed;
3515.                       propHEdit.Visibility = Visibility.Collapsed;
3516.                       propBEdit.Visibility = Visibility.Collapsed;
3517.                       propHLabelEdit.Visibility =
   Visibility.Collapsed;
3518.                       propBLabelEdit.Visibility =
   Visibility.Collapsed;
3519.                   }
3520.
3521.                   if (propSecComboBoxEdit.SelectedIndex == 2)
3522.                   {
3523.                       propNoneSaveEdit.IsEnabled = false;
3524.                       propNoneDeleteEdit.IsEnabled = false;
3525.
3526.                       propRetCalcEdit.IsEnabled = true;
3527.                       propRetSaveEdit.IsEnabled = true;
3528.                       propRetDeleteEdit.IsEnabled = true;
3529.                       propHEdit.IsEnabled = true;
3530.                       propBEdit.IsEnabled = true;
3531.                       propRetSepEdit.IsEnabled = true;
3532.
3533.                       propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3534.                       propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3535.                       propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3536.
3537.                       propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3538.                       propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3539.                       propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3540.                       propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3541.                       propDiamEdit.Visibility =
   Visibility.Collapsed;
3542.                       propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3543.
```

```
3544.                         propRetCalcEdit.Visibility =
   Visibility.Visible;
3545.                         propRetSaveEdit.Visibility =
   Visibility.Visible;
3546.                         propRetDeleteEdit.Visibility =
   Visibility.Visible;
3547.                         propRetSepEdit.Visibility =
   Visibility.Visible;
3548.                         propHEdit.Visibility = Visibility.Visible;
3549.                         propBEdit.Visibility = Visibility.Visible;
3550.                         propHLabelEdit.Visibility =
   Visibility.Visible;
3551.                         propBLabelEdit.Visibility =
   Visibility.Visible;
3552.                     }
3553.                 }
3554.
3555.                 public void DeleteEditProperty(object sender,
   RoutedEventArgs e)
3556.                 {
3557.                     try
3558.                     {
3559.                         properties.Remove(properties.Where(item =>
   item.name == editComboBox.SelectedItem.ToString()).First());
3560.
3561.                         editComboBox.Items.Clear();
3562.
3563.                         for (int i = 0; i < properties.Count; i++)
3564.                         {
3565.
   editComboBox.Items.Add(properties[i].name);
3566.                         }
3567.
3568.                         propNameEdit.Text = "";
3569.                         propAEdit.Text = "";
3570.                         propIxEdit.Text = "";
3571.                         propIyEdit.Text = "";
3572.                         propIzEdit.Text = "";
3573.
3574.                         propNameEdit.IsEnabled = false;
3575.
3576.                         propAEdit.IsEnabled = false;
3577.                         propIxEdit.IsEnabled = false;
3578.                         propIyEdit.IsEnabled = false;
3579.                         propIzEdit.IsEnabled = false;
3580.
3581.                         editComboBox.IsEnabled = true;
3582.
3583.                         propSecComboBoxEdit.IsEnabled = false;
3584.                         propSecComboBoxEdit.SelectedIndex = 0;
3585.
3586.                         propNoneSaveEdit.IsEnabled = false;
3587.                         propNoneDeleteEdit.IsEnabled = false;
3588.
3589.                         propNoneSaveEdit.Visibility =
   Visibility.Visible;
3590.                         propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3591.                         propNoneSepEdit.Visibility =
   Visibility.Visible;
3592.
```

```
3593.                         propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3594.                         propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3595.                         propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3596.                         propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3597.                         propDiamEdit.Visibility =
   Visibility.Collapsed;
3598.
3599.                         propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3600.                         propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3601.                         propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3602.                         propRetSep.Visibility =
   Visibility.Collapsed;
3603.                         propHEdit.Visibility = Visibility.Collapsed;
3604.                         propBEdit.Visibility = Visibility.Collapsed;
3605.                     }
3606.                     catch
3607.                     {
3608.
3609.                     }
3610.                 }
3611.
3612.                 public void CalculateCircularEditProperty(object
   sender, RoutedEventArgs e)
3613.                 {
3614.                     try
3615.                     {
3616.                         double A = Math.PI *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 2);
3617.                         double Ix = (Math.PI / 4) *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 4);
3618.                         double Iy = Ix;
3619.                         double Iz = (Math.PI / 2) *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 4);
3620.
3621.                         propAEdit.Text = A.ToString();
3622.                         propIxEdit.Text = Ix.ToString();
3623.                         propIyEdit.Text = Iy.ToString();
3624.                         propIzEdit.Text = Iz.ToString();
3625.                     }
3626.                     catch
3627.                     {
3628.                         if (propSecComboBoxEdit.SelectedIndex == 0)
3629.                         {
3630.                             labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3631.                             labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3632.                         }
3633.                         if (propSecComboBoxEdit.SelectedIndex == 1)
3634.                         {
3635.                             labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3636.                             labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
```

```
3637.                         }
3638.                         if (propSecComboBoxEdit.SelectedIndex == 2)
3639.                         {
3640.                             labelWrongPropRetEdit.Content =
   "Property Wrong!";
3641.                             labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3642.                         }
3643.                     }
3644.                 }
3645.
3646.                 public void CalculateRetangularEditProperty(object
   sender, RoutedEventArgs e)
3647.                 {
3648.                     try
3649.                     {
3650.                         double A = double.Parse(propHEdit.Text) *
   double.Parse(propBEdit.Text);
3651.                         double Ix = double.Parse(propBEdit.Text) *
   Math.Pow(double.Parse(propHEdit.Text), 3) / 12;
3652.                         double Iy = double.Parse(propHEdit.Text) *
   Math.Pow(double.Parse(propBEdit.Text), 3) / 12;
3653.                         double Iz = Ix + Iy;
3654.
3655.                         propAEdit.Text = A.ToString();
3656.                         propIxEdit.Text = Ix.ToString();
3657.                         propIyEdit.Text = Iy.ToString();
3658.                         propIzEdit.Text = Iz.ToString();
3659.                     }
3660.                     catch
3661.                     {
3662.                         if (propSecComboBoxEdit.SelectedIndex == 0)
3663.                         {
3664.                             labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3665.                             labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3666.                         }
3667.                         if (propSecComboBoxEdit.SelectedIndex == 1)
3668.                         {
3669.                             labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3670.                             labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3671.                         }
3672.                         if (propSecComboBoxEdit.SelectedIndex == 2)
3673.                         {
3674.                             labelWrongPropRetEdit.Content =
   "Property Wrong!";
3675.                             labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3676.                         }
3677.                     }
3678.                 }
3679.
3680.                 public void SaveEditProperty(object sender,
   RoutedEventArgs e)
3681.                 {
3682.                     if (propNameEdit.Text != "" && propAEdit.Text !=
   "" && propIxEdit.Text != "" && propIyEdit.Text != "" &&
   propIzEdit.Text != "")
```

```
3683.                     {
3684.                         try
3685.                         {
3686.
3687.                             string propName =
   editComboBox.SelectedItem.ToString();
3688.
3689.                             if (propName != propNameEdit.Text &&
   properties.Where(item => item.name ==
   propNameEdit.Text).ToList().Count != 0)
3690.                             {
3691.                                 //labelWrongMatPEdit.Content =
   "Wrong Properties!";
3692.                                 //labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3693.                             }
3694.                             else
3695.                             {
3696.                                 properties.Where(item => item.name
   == propName).First().A = double.Parse(propAEdit.Text);
3697.                                 properties.Where(item => item.name
   == propName).First().Ix = double.Parse(propIxEdit.Text);
3698.                                 properties.Where(item => item.name
   == propName).First().Iy = double.Parse(propIyEdit.Text);
3699.                                 properties.Where(item => item.name
   == propName).First().Iz = double.Parse(propIzEdit.Text);
3700.                                 properties.Where(item => item.name
   == propName).First().d = double.Parse(propDiamEdit.Text);
3701.                                 properties.Where(item => item.name
   == propName).First().h = double.Parse(propHEdit.Text);
3702.                                 properties.Where(item => item.name
   == propName).First().b = double.Parse(propBEdit.Text);
3703.                                 if
   (propSecComboBoxEdit.SelectedIndex == 0) properties.Where(item =>
   item.name == propName).First().type = "None";
3704.                                 if
   (propSecComboBoxEdit.SelectedIndex == 1) properties.Where(item =>
   item.name == propName).First().type = "Circular";
3705.                                 if
   (propSecComboBoxEdit.SelectedIndex == 2) properties.Where(item =>
   item.name == propName).First().type = "Retangular";
3706.                                 properties.Where(item => item.name
   == propName).First().name = propNameEdit.Text;
3707.
3708.                                 for (int i = 0; i < members.Count;
   i++)
3709.                                 {
3710.                                     if (members[i].section ==
   editComboBox.SelectedItem.ToString()) members[i].section =
   propNameEdit.Text;
3711.                                 }
3712.
3713.                                 editComboBox.Items.Clear();
3714.
3715.                                 for (int i = 0; i <
   properties.Count; i++)
3716.                                 {
3717.
   editComboBox.Items.Add(properties[i].name);
3718.                                 }
3719.
```

```
3720.                                   propNameEdit.Text = "";
3721.                                   propAEdit.Text = "";
3722.                                   propIxEdit.Text = "";
3723.                                   propIyEdit.Text = "";
3724.                                   propIzEdit.Text = "";
3725.
3726.                                   propNameEdit.IsEnabled = false;
3727.
3728.                                   propAEdit.IsEnabled = false;
3729.                                   propIxEdit.IsEnabled = false;
3730.                                   propIyEdit.IsEnabled = false;
3731.                                   propIzEdit.IsEnabled = false;
3732.
3733.                                   editComboBox.IsEnabled = true;
3734.
3735.                                   propSecComboBoxEdit.IsEnabled =
   false;
3736.                                   propSecComboBoxEdit.SelectedIndex =
   0;
3737.
3738.                                   propNoneSaveEdit.IsEnabled = false;
3739.                                   propNoneDeleteEdit.IsEnabled =
   false;
3740.
3741.                                   propNoneSaveEdit.Visibility =
   Visibility.Visible;
3742.                                   propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3743.                                   propNoneSepEdit.Visibility =
   Visibility.Visible;
3744.
3745.                                   propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3746.                                   propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3747.                                   propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3748.                                   propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3749.                                   propDiamEdit.Visibility =
   Visibility.Collapsed;
3750.
3751.                                   propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3752.                                   propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3753.                                   propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3754.                                   propRetSep.Visibility =
   Visibility.Collapsed;
3755.                                   propHEdit.Visibility =
   Visibility.Collapsed;
3756.                                   propBEdit.Visibility =
   Visibility.Collapsed;
3757.
3758.                                   labelWrongPropNoneEdit.Content =
   "Property Saved!";
3759.                                   labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3760.                               }
3761.                           }
```

```
3762.                         catch
3763.                         {
3764.                             if (propSecComboBoxEdit.SelectedIndex ==
   0)
3765.                             {
3766.                                 labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3767.                                 labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3768.                             }
3769.                             if (propSecComboBoxEdit.SelectedIndex ==
   1)
3770.                             {
3771.                                 labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3772.                                 labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3773.                             }
3774.                             if (propSecComboBoxEdit.SelectedIndex ==
   2)
3775.                             {
3776.                                 labelWrongPropRetEdit.Content =
   "Property Wrong!";
3777.                                 labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3778.                             }
3779.                         }
3780.                     }
3781.                     else
3782.                     {
3783.                         if (propSecComboBoxEdit.SelectedIndex == 0)
3784.                         {
3785.                             labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3786.                             labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3787.                         }
3788.                         if (propSecComboBoxEdit.SelectedIndex == 1)
3789.                         {
3790.                             labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3791.                             labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3792.                         }
3793.                         if (propSecComboBoxEdit.SelectedIndex == 2)
3794.                         {
3795.                             labelWrongPropRetEdit.Content =
   "Property Wrong!";
3796.                             labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3797.                         }
3798.                     }
3799.                 }
3800.
3801.                 public void Enable3DOrbit(object sender,
   RoutedEventArgs e)
3802.                 {
3803.                     SendStringToInterface(string.Format("Orbit
   Status&True"));
3804.                 }
3805.
```

```
3806.                public void Disable3DOrbit(object sender,
   RoutedEventArgs e)
3807.                {
3808.                    SendStringToInterface(string.Format("Orbit
   Status&False"));
3809.                }
3810.
3811.                public void EnableNodes(object sender,
   RoutedEventArgs e)
3812.                {
3813.                    SendStringToInterface(string.Format("Node
   Status&True"));
3814.                }
3815.
3816.                public void DisableNodes(object sender,
   RoutedEventArgs e)
3817.                {
3818.                    SendStringToInterface(string.Format("Node
   Status&False"));
3819.                }
3820.
3821.                public void EnableMembers(object sender,
   RoutedEventArgs e)
3822.                {
3823.                    SendStringToInterface(string.Format("Member
   Status&True"));
3824.                }
3825.
3826.                public void DisableMembers(object sender,
   RoutedEventArgs e)
3827.                {
3828.                    SendStringToInterface(string.Format("Member
   Status&False"));
3829.                }
3830.
3831.                public void ResetView(object sender, RoutedEventArgs
   e)
3832.                {
3833.                    SendStringToInterface(string.Format("Reset
   View"));
3834.                }
3835.
3836.                public void RunSimulationClick(object sender,
   RoutedEventArgs e)
3837.                {
3838.
   SendStringToInterface(string.Format("Mode&Mounting Simulation"));
3839.                    MountModels();
3840.
   SendStringToInterface(string.Format("Mode&Running Simulation"));
3841.                    RunSimulations();
3842.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   endDay));
3843.
3844.                    simulationSuccess = true;
3845.                    EnableSimulationSuccessfullUI();
3846.                }
3847.
3848.                public void UpdateUISimulationClick(object sender,
   RoutedEventArgs e)
```

```
3849.                {
3850.                    double diff = endDay - startDay;
3851.
3852.                    double value = double.Parse(SimShowDay.Text) -
   startDay;
3853.
3854.                    double percent = value / diff;
3855.
3856.                    percent = percent * 20d;
3857.
3858.                    SimSlider.Value = percent;
3859.
3860.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   SimShowDay.Text));
3861.                }
3862.
3863.                public void EnableSimulationSuccessfullUI()
3864.                {
3865.                    SimSep1.Visibility = Visibility.Visible;
3866.                    SimLabelDay.Visibility = Visibility.Visible;
3867.                    SimShowDay.Visibility = Visibility.Visible;
3868.                    SimUpdate.Visibility = Visibility.Visible;
3869.                    SimSep2.Visibility = Visibility.Visible;
3870.                    SimSlider.Visibility = Visibility.Visible;
3871.                    SimLabelObject.Visibility = Visibility.Visible;
3872.                    SimPlot.Visibility = Visibility.Visible;
3873.                    SimulationElementResults.Visibility =
   Visibility.Visible;
3874.
3875.                    SimShowDay.Text = endDay.ToString();
3876.                    SimulationElementResults.Text = "No object
   selected";
3877.                    SimPlot.IsEnabled = false;
3878.
3879.                    double diff = endDay - startDay;
3880.
3881.                    double value = double.Parse(SimShowDay.Text) -
   startDay;
3882.
3883.                    double percent = value / diff;
3884.
3885.                    percent = percent * 20d;
3886.
3887.                    SimSlider.Value = percent;
3888.                }
3889.
3890.                private void SimSliderValue(object sender,
   RoutedPropertyChangedEventArgs<double> e)
3891.                {
3892.                    if (!finishedLoading) return;
3893.
3894.                    int diff = endDay - startDay;
3895.
3896.                    double value = SimSlider.Value / 20d;
3897.
3898.                    SimShowDay.Text = Convert.ToInt32(startDay +
   diff * value).ToString();
3899.
```

```
3900.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   SimShowDay.Text));
3901.                }
3902.
3903.                public void MountPlotForNode(object sender,
   RoutedEventArgs e)
3904.                {
3905.                    Plot plot = new Plot();
3906.
3907.                    plot.Show();
3908.                }
3909.
3910.                public void About(object sender, RoutedEventArgs e)
3911.                {
3912.                    About about = new About();
3913.
3914.                    about.Show();
3915.                }
3916.
3917.                public void SendStringToInterface(string text)
3918.                {
3919.                    if (!finishedLoading) return;
3920.
3921.                    text = "%%&" + text + "&%%";
3922.
3923.                    ASCIIEncoding ASEn = new ASCIIEncoding();
3924.
3925.                    byte[] byteArray = ASEn.GetBytes(text);
3926.
3927.                    SendData(byteArray);
3928.                }
3929.
3930.                public void Save(object sender, RoutedEventArgs e)
3931.                {
3932.                    SaveFileDialog saveFileDialog1 = new
   SaveFileDialog();
3933.                    saveFileDialog1.Filter = "Structure
   File|*.struct";
3934.                    saveFileDialog1.Title = "Save a Structure File";
3935.                    saveFileDialog1.ShowDialog();
3936.
3937.                    if (saveFileDialog1.FileName != "")
3938.                    {
3939.                        BinaryFormatter bf = new BinaryFormatter();
3940.                        FileStream file =
   File.Open(saveFileDialog1.FileName, FileMode.OpenOrCreate);
3941.
3942.                        SaveFile save = new SaveFile();
3943.                        save.nodes = nodes;
3944.                        save.members = members;
3945.                        save.materials = materials;
3946.                        save.properties = properties;
3947.                        save.nodalForces = nodalForces;
3948.                        save.memberForces = memberForces;
3949.                        save.startDay = startDay;
3950.                        save.endDay = endDay;
3951.
3952.                        bf.Serialize(file, save);
3953.                        file.Close();
3954.                    }
```

```
3955.                 }
3956.
3957.                 public void Load(object sender, RoutedEventArgs e)
3958.                 {
3959.                     OpenFileDialog openFileDialog1 = new
   OpenFileDialog();
3960.                     openFileDialog1.Filter = "Structure
   File|*.struct";
3961.                     openFileDialog1.Title = "Open a Structure File";
3962.                     openFileDialog1.ShowDialog();
3963.
3964.                     if (openFileDialog1.FileName != "")
3965.                     {
3966.                         BinaryFormatter bf = new BinaryFormatter();
3967.                         FileStream file =
   File.Open(openFileDialog1.FileName, FileMode.Open);
3968.
3969.                         SaveFile save =
   (SaveFile)bf.Deserialize(file);
3970.
3971.                         nodes = save.nodes;
3972.                         members = save.members;
3973.                         materials = save.materials;
3974.                         properties = save.properties;
3975.                         nodalForces = save.nodalForces;
3976.                         memberForces = save.memberForces;
3977.                         startDay = save.startDay;
3978.                         endDay = save.endDay;
3979.
3980.                         MassiveUpdate3DUI();
3981.
3982.                         file.Close();
3983.                     }
3984.                 }
3985.
3986.                 public void MassiveUpdate3DUI()
3987.                 {
3988.                     string command = "Open Massive Structure
   Changes";
3989.
3990.                     for (int i = 0; i < nodes.Count; i++)
3991.                     {
3992.                         command +=
   string.Format("&N&{0}&{1:0.00}&{2:0.00}&{3:0.00}&{4}",
   nodes[i].number, nodes[i].position.x, nodes[i].position.y,
   nodes[i].position.z, nodes[i].restriction.x1 ||
   nodes[i].restriction.y1 || nodes[i].restriction.z1 ||
   nodes[i].restriction.x2 || nodes[i].restriction.y2 ||
   nodes[i].restriction.z2);
3993.                     }
3994.
3995.                     for (int i = 0; i < members.Count; i++)
3996.                     {
3997.                         command +=
   string.Format("&M&{0}&{1}&{2}&{3}", members[i].n0, members[i].n1,
   members[i].startDay, members[i].endDay);
3998.                     }
3999.
4000.                     SendStringToInterface(command);
4001.                 }
4002.             }
```

```
4003.
4004.           [System.Serializable]
4005.           public class SaveFile
4006.           {
4007.               public List<Nodes> nodes = new List<Nodes>();
4008.               public List<Member> members = new List<Member>();
4009.
4010.               public List<Material> materials = new
   List<Material>();
4011.               public List<Property> properties = new
   List<Property>();
4012.
4013.               public List<NodalForces> nodalForces = new
   List<NodalForces>();
4014.               public List<MemberForces> memberForces = new
   List<MemberForces>();
4015.
4016.               public int startDay;
4017.               public int endDay;
4018.           }
4019.       }
```

```
1.  using System;
2. using System.Collections.Generic;
3. using System.Diagnostics;
4. using System.Linq;
5. using System.Runtime.InteropServices;
6. using System.Text;
7. using System.Threading.Tasks;
8. using System.Windows;
9. using System.Windows.Controls;
10.  using System.Windows.Data;
11.  using System.Windows.Documents;
12.  using System.Windows.Input;
13.  using System.Windows.Media;
14.  using System.Windows.Media.Imaging;
15.  using System.Windows.Navigation;
16.  using System.Windows.Shapes;
17.  using System.Windows.Forms;
18.  using System.Threading;
19.  using System.Net;
20.  using System.Net.Sockets;
21.  using System.Windows.Interop;
22.  using System.Windows.Threading;
23.  using BriefFiniteElementNet;
24.  using System.Runtime.Serialization.Formatters.Binary;
25.  using System.IO;
26.
27.  namespace ProjectTango
28.  {
29.      [System.Serializable]
30.      public class Vector3
31.      {
32.          public double x;
33.          public double y;
34.          public double z;
35.
36.          public Vector3()
37.          {
38.              x = 0;
39.              y = 0;
```

```
40.             z = 0;
41.         }
42.
43.         public Vector3(string x, string y, string z)
44.         {
45.             this.x = double.Parse(x);
46.             this.y = double.Parse(y);
47.             this.z = double.Parse(z);
48.         }
49.
50.         public Vector3(double x, double y, double z)
51.         {
52.             this.x = x;
53.             this.y = y;
54.             this.z = z;
55.         }
56.     }
57.
58.     [System.Serializable]
59.     public class Restrictions
60.     {
61.         public bool x1;
62.         public bool y1;
63.         public bool z1;
64.         public bool x2;
65.         public bool y2;
66.         public bool z2;
67.
68.         public Restrictions()
69.         {
70.             x1 = false;
71.             y1 = false;
72.             z1 = false;
73.             x2 = false;
74.             y2 = false;
75.             z2 = false;
76.         }
77.
78.         public Restrictions(bool x1, bool y1, bool z1, bool x2,
   bool y2, bool z2)
79.         {
80.             this.x1 = x1;
81.             this.y1 = y1;
82.             this.z1 = z1;
83.             this.x2 = x2;
84.             this.y2 = y2;
85.             this.z2 = z2;
86.         }
87.     }
88.
89.     [System.Serializable]
90.     public class Material
91.     {
92.         public string name;
93.
94.         public double E;
95.         public double G;
96.         public double v;
97.
98.         public bool timeDependent;
99.
```

```
100.         public double fck;
101.
102.         public Material()
103.         {
104.             name = "";
105.
106.             E = 0;
107.             G = 0;
108.             v = 0;
109.
110.             timeDependent = false;
111.
112.             fck = 0;
113.         }
114.
115.         public Material(string name, string E, string G, bool time,
    string fck = "0", string v = "0")
116.         {
117.             this.name = name;
118.
119.             this.E = double.Parse(E);
120.             this.G = double.Parse(G);
121.             this.v = double.Parse(v);
122.
123.             this.timeDependent = time;
124.
125.             this.fck = double.Parse(fck);
126.         }
127.     }
128.
129.     [System.Serializable]
130.     public class Property
131.     {
132.         public string name;
133.
134.         public double A;
135.         public double Ix;
136.         public double Iy;
137.         public double Iz;
138.
139.         public string type;
140.
141.         public double d;
142.
143.         public double h;
144.         public double b;
145.
146.         public Property(string name, string A, string Ix, string
    Iy, string Iz, string type, string d = "0", string h = "0", string b
    = "0")
147.         {
148.             if (d == "") d = "0";
149.             if (b == "") b = "0";
150.             if (h == "") h = "0";
151.
152.             if (type == "0") type = "None";
153.             if (type == "1") type = "Circular";
154.             if (type == "2") type = "Retangular";
155.
156.             this.name = name;
157.
```

```
158.             this.A = double.Parse(A);
159.             this.Ix = double.Parse(Ix);
160.             this.Iy = double.Parse(Iy);
161.             this.Iz = double.Parse(Iz);
162.
163.             this.type = type;
164.
165.             this.d = double.Parse(d);
166.             this.h = double.Parse(h);
167.             this.b = double.Parse(b);
168.         }
169.     }
170.
171.     [System.Serializable]
172.     public class Member
173.     {
174.         public int number;
175.
176.         public int n0;
177.         public int n1;
178.
179.         public string material;
180.         public string section;
181.
182.         public int startDay;
183.         public int endDay;
184.
185.         public Member(int number, int n0, int n1, string material,
    string section, string startDay, string endDay)
186.         {
187.             this.number = number;
188.
189.             this.n0 = n0;
190.             this.n1 = n1;
191.             this.material = material;
192.             this.section = section;
193.
194.             this.startDay = int.Parse(startDay);
195.             this.endDay = int.Parse(endDay);
196.         }
197.     }
198.
199.     [System.Serializable]
200.     public class Nodes
201.     {
202.         public int number;
203.
204.         public Vector3 position;
205.         public Restrictions restriction;
206.
207.         public Nodes()
208.         {
209.             position = new Vector3();
210.             restriction = new Restrictions();
211.         }
212.
213.         public Nodes(int number, string x, string y, string z,
    bool? x1, bool? y1, bool? z1, bool? x2, bool? y2, bool? z2)
214.         {
215.             this.number = number;
216.
```

```
217.             position = new Vector3(x, y, z);
218.             restriction = new Restrictions(x1.GetValueOrDefault(),
    y1.GetValueOrDefault(), z1.GetValueOrDefault(),
    x2.GetValueOrDefault(), y2.GetValueOrDefault(),
    z2.GetValueOrDefault());
219.         }
220.     }
221.
222.     [System.Serializable]
223.     public class NodalForces
224.     {
225.         public string name;
226.
227.         public Vector3 disp;
228.         public Vector3 rott;
229.
230.         public List<int> nodes;
231.
232.         public int startDay;
233.         public int endDay;
234.
235.         public NodalForces()
236.         {
237.             name = "";
238.
239.             disp = new Vector3();
240.             rott = new Vector3();
241.
242.             nodes = new List<int>();
243.
244.             startDay = 0;
245.             endDay = 0;
246.         }
247.
248.         public NodalForces(string name, double dx, double dy,
    double dz, double rx, double ry, double rz, int startDay, int endDay,
    List<int> list)
249.         {
250.             this.name = name;
251.
252.             disp = new Vector3(dx, dy, dz);
253.             rott = new Vector3(rz, ry, rz);
254.
255.             nodes = list;
256.
257.             this.startDay = startDay;
258.             this.endDay = endDay;
259.         }
260.     }
261.
262.     [System.Serializable]
263.     public class MemberForces
264.     {
265.         public string name;
266.
267.         public Vector3 disp;
268.
269.         public List<int> members;
270.
271.         public int startDay;
272.         public int endDay;
```

```
273.
274.         public MemberForces()
275.         {
276.             name = "";
277.
278.             disp = new Vector3();
279.
280.             members = new List<int>();
281.
282.             startDay = 0;
283.             endDay = 0;
284.         }
285.
286.         public MemberForces(string name, double dx, double dy,
    double dz, int startDay, int endDay, List<int> list)
287.         {
288.             this.name = name;
289.
290.             disp = new Vector3(dx, dy, dz);
291.
292.             members = list;
293.
294.             this.startDay = startDay;
295.             this.endDay = endDay;
296.         }
297.     }
298.
299.     /// <summary>
300.     /// Interaction logic for MainWindow.xaml
301.     /// </summary>
302.     public partial class MainWindow : Window
303.     {
304.         public static MainWindow instance;
305.
306.         // Create Form Panel to use as host for Unity 3D Window
307.         private System.Windows.Forms.Panel _panel;
308.         // Create a Process to call Unity 3D Window
309.         private Process _process;
310.         private IntPtr _unityHWND = IntPtr.Zero;
311.
312.         [DllImport("user32.dll")]
313.         private static extern int SetWindowLong(IntPtr hWnd, int
    nIndex, int dwNewLong);
314.
315.         [DllImport("user32.dll", SetLastError = true)]
316.         private static extern int GetWindowLong(IntPtr hWnd, int
    nIndex);
317.
318.         [DllImport("user32")]
319.         private static extern IntPtr SetParent(IntPtr hWnd, IntPtr
    hWndParent);
320.
321.         [DllImport("user32")]
322.         private static extern bool SetWindowPos(IntPtr hWnd, IntPtr
    hWndInsertAfter, int X, int Y, int cx, int cy, int uFlags);
323.
324.         [DllImport("User32.dll")]
325.         static extern bool MoveWindow(IntPtr handle, int x, int y,
    int width, int height, bool redraw);
326.
```

```
327.         internal delegate int WindowEnumProc(IntPtr hwnd, IntPtr
   lparam);
328.         [DllImport("user32.dll")]
329.         internal static extern bool EnumChildWindows(IntPtr hwnd,
   WindowEnumProc func, IntPtr lParam);
330.
331.         [DllImport("user32.dll")]
332.         static extern int SendMessage(IntPtr hWnd, int msg, IntPtr
   wParam, IntPtr lParam);
333.
334.         private const int WM_ACTIVATE = 0x0006;
335.         private readonly IntPtr WA_ACTIVE = new IntPtr(1);
336.         private readonly IntPtr WA_INACTIVE = new IntPtr(0);
337.         private bool finishedLoading = false;
338.
339.         Socket s;
340.         TcpListener myList;
341.         public static string threadCommand = "";
342.         public int tempMemberN0;
343.         public int tempMemberN1;
344.
345.         //Analysis variables.
346.         public List<Nodes> nodes = new List<Nodes>();
347.         public List<Member> members = new List<Member>();
348.
349.         public List<Material> materials = new List<Material>();
350.         public List<Property> properties = new List<Property>();
351.
352.         public List<NodalForces> nodalForces = new
   List<NodalForces>();
353.         public List<MemberForces> memberForces = new
   List<MemberForces>();
354.
355.         public List<BriefFiniteElementNet.Model> models = new
   List<BriefFiniteElementNet.Model>();
356.
357.         // Simulation Variables
358.         public int startDay = 0;
359.         public int endDay = 720;
360.         public bool simulationSuccess = false;
361.         public string lastSimulatedSelected = "";
362.
363.         public MainWindow()
364.         {
365.             // Unhandled exceptions for our Application Domain
366.             AppDomain.CurrentDomain.UnhandledException += new
   System.UnhandledExceptionEventHandler(AppDomain_UnhandledException);
367.
368.             // Unhandled exceptions for the executing UI thread
369.             System.Windows.Forms.Application.ThreadException += new
   System.Threading.ThreadExceptionEventHandler(Application_ThreadExcept
   ion);
370.
371.             instance = this;
372.             //Example3();
373.             //Example2();
374.             //StructuralAnalysisMathNet analysis = new
   StructuralAnalysisMathNet();
375.             InitializeComponent();
376.             InitializeUnity3D();
377.             InitializeSocket();
```

```
378.             finishedLoading = true;
379.         }
380.
381.         //private static void Example3()
382.         //{
383.         //    var Analysis = new SpatialFrameAnalysis.Analysis();
384.
385.         //    var material = new MaterialProprieties(21000000000d,
    6000000000d);
386.         //    var proprieties = new GeometricalProprieties(0.04d,
    0.00013333333d, 0.00013333333d, 0.00026666666d);
387.
388.         //    var n0 = new Node(0, 0, 0, 0, Restriction.Fixed);
389.         //    var n1 = new Node(1, 0, 6, 0, Restriction.Released);
390.
391.         //    Analysis.Nodes.Add(n0);
392.         //    Analysis.Nodes.Add(n1);
393.
394.         //    var m0 = new Member(n0, n1, proprieties, material);
395.
396.         //    Analysis.Members.Add(m0);
397.
398.         //    Analysis.SolveLinear();
399.
400.         //    Console.Write(Analysis.data);
401.         //}
402.
403.         public static bool LineSegmentIntersection(double Ax,
    double Ay, double Bx, double By, double Cx, double Cy, double Dx,
    double Dy, ref double X, ref double Y)
404.         {
405.             double distAB, theCos, theSin, newX, ABpos;
406.
407.             //  Fail if either line segment is zero-length.
408.
409.             if (Ax == Bx && Ay == By || Cx == Dx && Cy == Dy)
    return false;
410.
411.             //  Fail if the segments share an end-point.
412.
413.             if (Ax == Cx && Ay == Cy || Bx == Cx && By == Cy || Ax
    == Dx && Ay == Dy || Bx == Dx && By == Dy)
414.             {
415.                 return false;
416.             }
417.
418.
419.             //  (1) Translate the system so that point A is on the
    origin.
420.
421.             Bx -= Ax; By -= Ay;
422.
423.             Cx -= Ax; Cy -= Ay;
424.
425.             Dx -= Ax; Dy -= Ay;
426.
427.
428.             //  Discover the length of segment A-B.
429.
430.             distAB = Math.Sqrt(Bx * Bx + By * By);
431.
```

```
432.
433.             //  (2) Rotate the system so that point B is on the
   positive X axis.
434.
435.             theCos = Bx / distAB;
436.
437.             theSin = By / distAB;
438.
439.             newX = Cx * theCos + Cy * theSin;
440.
441.             Cy = Cy * theCos - Cx * theSin; Cx = newX;
442.
443.             newX = Dx * theCos + Dy * theSin;
444.
445.             Dy = Dy * theCos - Dx * theSin; Dx = newX;
446.
447.             //  Fail if segment C-D doesn't cross line A-B.
448.
449.             if (Cy < 0 && Dy < 0 || Cy >= 0 && Dy >= 0) return
   false;
450.
451.             //  (3) Discover the position of the intersection point
   along line A-B.
452.
453.             ABpos = Dx + (Cx - Dx) * Dy / (Dy - Cy);
454.
455.             //  Fail if segment C-D crosses line A-B outside of
   segment A-B.
456.
457.             if (ABpos < 0 || ABpos > distAB) return false;
458.
459.             //  (4) Apply the discovered position to line A-B in
   the original coordinate system.
460.
461.             X = (Ax + ABpos * theCos);
462.
463.             Y = (Ay + ABpos * theSin);
464.
465.
466.             //  Success.
467.
468.             return true;
469.         }
470.
471.         public double GetElasticityModulusABNT(double fck, double
   t, double alphae = 1f, double s = 0.25f)
472.         {
473.             if (t < 1) t = 0.5f;
474.
475.             double beta1 = Math.Pow(Math.E, (s * (1 - Math.Sqrt(28
   / t))));
476.
477.             double fckj = beta1 * fck;
478.
479.             if (t > 28) fckj = fck;
480.
481.             double Eci = 0;
482.             double Ecit = 0;
483.
484.             if (fck >= 20 && fck <= 45)
485.             {
```

```
486.                 Eci = alphae * 5600 * Math.Sqrt(fck);
487.                 Ecit = Math.Pow((fckj / fck), 0.5) * Eci;
488.             }
489.             if (fck >= 50 && fck <= 90)
490.             {
491.                 Eci = 21.5 * Math.Pow(10, 3) * alphae *
    Math.Pow((fck / 10) + 1.25, 1 / 3);
492.                 Ecit = Math.Pow((fckj / fck), 0.3) * Eci;
493.             }
494.
495.             // Conversion from MPa to kN/m2;
496.             Ecit = Ecit * 1000;
497.
498.             return Ecit;
499.         }
500.
501.         public double CalculateG(double E, double v)
502.         {
503.             double G = E / (2 * (1 + v));
504.
505.             return G;
506.         }
507.
508.         private void MountModels()
509.         {
510.             try
511.             {
512.                 startDay = int.Parse(simulationDayStart.Text);
513.                 endDay = int.Parse(simulationDayEnd.Text);
514.
515.                 models.Clear();
516.
517.                 for (int i = startDay; i <= endDay; i++)
518.                 {
519.                     var model = new BriefFiniteElementNet.Model();
520.
521.                     var acceptedNodes = new List<int>();
522.
523.                     foreach (Member member in members)
524.                     {
525.                         if (i >= member.startDay)
526.                         {
527.                             if (member.endDay >= i)
528.                             {
529.                                 if
    (!acceptedNodes.Contains(member.n0)) acceptedNodes.Add(member.n0);
530.                                 if
    (!acceptedNodes.Contains(member.n1)) acceptedNodes.Add(member.n1);
531.                             }
532.                         }
533.                     }
534.
535.                     foreach (int n in acceptedNodes)
536.                     {
537.                         var node = new Node(nodes[n].position.x,
    nodes[n].position.y, nodes[n].position.z);
538.
539.                         if (nodes[n].restriction.x1)
    node.Constraints &= Constraints.FixedDX;
540.                         if (nodes[n].restriction.y1)
    node.Constraints &= Constraints.FixedDY;
```

```
541.                         if (nodes[n].restriction.z1)
    node.Constraints &= Constraints.FixedDZ;
542.                         if (nodes[n].restriction.x2)
    node.Constraints &= Constraints.FixedRX;
543.                         if (nodes[n].restriction.y2)
    node.Constraints &= Constraints.FixedRY;
544.                         if (nodes[n].restriction.z2)
    node.Constraints &= Constraints.FixedRZ;
545.
546.                         node.Label = n.ToString();
547.
548.                         model.Nodes.Add(node);
549.                     }
550.
551.                     foreach (Member member in members)
552.                     {
553.                         if (i >= member.startDay)
554.                         {
555.                             if (member.endDay >= i)
556.                             {
557.                                 var mem = new
    FrameElement2Node(model.Nodes.Where(item => item.Label ==
    member.n0.ToString()).First(), model.Nodes.Where(item => item.Label
    == member.n1.ToString()).First());
558.
559.                                 mem.Label = "M" +
    member.number.ToString();
560.
561.                                 if (materials.Where(item =>
    item.name == member.material).First().timeDependent)
562.                                 {
563.                                     mem.E =
    GetElasticityModulusABNT(materials.Where(item => item.name ==
    member.material).First().fck / 1000, i - member.startDay);
564.                                     mem.G = CalculateG(mem.E,
    materials.Where(item => item.name == member.material).First().v);
565.                                 }
566.                                 else
567.                                 {
568.                                     mem.E = materials.Where(item =>
    item.name == member.material).First().E;
569.                                     mem.G = materials.Where(item =>
    item.name == member.material).First().G;
570.                                 }
571.
572.                                 mem.A = properties.Where(item =>
    item.name == member.section).First().A;
573.                                 mem.Iy = properties.Where(item =>
    item.name == member.section).First().Ix;
574.                                 mem.Iz = properties.Where(item =>
    item.name == member.section).First().Iy;
575.                                 mem.J = properties.Where(item =>
    item.name == member.section).First().Iz;
576.
577.                                 mem.UseOverridedProperties = true;
578.
579.                                 model.Elements.Add(mem);
580.                             }
581.                         }
582.                     }
583.
```

```
584.                     foreach (NodalForces force in nodalForces)
585.                     {
586.                         if (i >= force.startDay)
587.                         {
588.                             if (force.endDay >= i)
589.                             {
590.                                 foreach (int node in force.nodes)
591.                                 {
592.                                     if (model.Nodes.Where(item =>
    item.Label == node.ToString()).ToList().Count > 0)
593.                                     {
594.                                         var f = new
    Force(force.disp.x, force.disp.y, force.disp.z, force.rott.x,
    force.rott.y, force.rott.z);
595.                                         model.Nodes.Where(item =>
    item.Label == node.ToString()).First().Loads.Add(new NodalLoad(f));
596.                                     }
597.                                 }
598.                             }
599.                         }
600.                     }
601.
602.                     foreach (MemberForces force in memberForces)
603.                     {
604.                         if (i >= force.startDay)
605.                         {
606.                             if (force.endDay >= i)
607.                             {
608.                                 foreach (int member in
    force.members)
609.                                 {
610.                                     if (model.Elements.Where(item
    => item.Label == "M" + member.ToString()).ToList().Count > 0)
611.                                     {
612.                                         if (force.disp.x != 0)
613.                                         {
614.                                             var load = new
    UniformLoad1D(force.disp.x, LoadDirection.X,
    CoordinationSystem.Global);
615.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
616.                                         }
617.                                         if (force.disp.y != 0)
618.                                         {
619.                                             var load = new
    UniformLoad1D(force.disp.y, LoadDirection.Y,
    CoordinationSystem.Global);
620.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
621.                                         }
622.                                         if (force.disp.z != 0)
623.                                         {
624.                                             var load = new
    UniformLoad1D(force.disp.z, LoadDirection.Z,
    CoordinationSystem.Global);
625.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
626.                                         }
```

```
627.                                         }
628.                                     }
629.
630.                                 }
631.                             }
632.                         }
633.
634.                         models.Add(model);
635.                     }
636.                 }
637.                 catch
638.                 {
639.
640.                 }
641.             }
642.
643.             private void RunSimulations()
644.             {
645.                 Parallel.ForEach(models, model =>
646.                 {
647.                     if (model.Nodes.Count > 0 && model.Elements.Count >
    0)
648.                     {
649.                         model.Solve();
650.                     }
651.                 });
652.             }
653.
654.             //private static void Example2()
655.             //{
656.             //    Console.WriteLine("Example 1: Simple 3D Frame with
    distributed loads");
657.
658.             //    var model = new BriefFiniteElementNet.Model();
659.
660.             //    var n1 = new Node(-10, 0, 0);
661.             //    var n2 = new Node(-10, 0, 6);
662.             //    var n3 = new Node(0, 0, 8);
663.             //    var n4 = new Node(10, 0, 6);
664.             //    var n5 = new Node(10, 0, 0);
665.             //    var n6 = new Node(20, 0, 0);
666.
667.             //    model.Nodes.Add(n1, n2, n3, n4, n5);
668.
669.             //    var secAA =
    SectionGenerator.GetRectangularSection(0.2, 0.2);
670.
671.             //    var b =
    PolygonYz.GetSectionGeometricalProperties(secAA);
672.
673.             //    var e1 = new FrameElement2Node(n1, n2);
674.             //    e1.Label = "e1";
675.             //    var e2 = new FrameElement2Node(n2, n3);
676.             //    e2.Label = "e2";
677.             //    var e3 = new FrameElement2Node(n3, n4);
678.             //    e3.Label = "e3";
679.             //    var e4 = new FrameElement2Node(n4, n5);
680.             //    e4.Label = "e4";
681.
682.             //    e1.Geometry = new PolygonYz(secAA);
683.
```

```
684.         //    e1.Geometry = e4.Geometry = e2.Geometry = e3.Geometry
   = new PolygonYz(secAA);
685.
686.         //    e1.E = e2.E = e3.E = e4.E = 210e9;
687.         //    e1.G = e2.G = e3.G = e4.G = 210e9 / (2 * (1 +
   0.3));//G = E / (2*(1+no))
688.
689.         //    e1.UseOverridedProperties =
690.         //        e2.UseOverridedProperties =
   e3.UseOverridedProperties =
691.         //
   e4.UseOverridedProperties = false;
692.
693.         //    model.Elements.Add(e1, e2, e3, e4);
694.
695.         //    n1.Constraints =
696.         //        n2.Constraints =
697.         //            n3.Constraints =
698.         //                n4.Constraints =
699.         //                    n5.Constraints =
700.         //                        Constraint.FixedDY &
   Constraint.FixedRX &
701.         //                        Constraint.FixedRZ;//DY fixed and
   RX fixed and RZ fixed
702.
703.         //    n1.Constraints = n1.Constraints &
   Constraint.MovementFixed;
704.         //    n5.Constraints = n5.Constraints &
   Constraint.MovementFixed;
705.
706.
707.         //    var ll = new UniformLoad1D(-10000, LoadDirection.Z,
   CoordinationSystem.Global);
708.         //    var lr = new UniformLoad1D(-10000, LoadDirection.Z,
   CoordinationSystem.Local);
709.
710.         //    e2.Loads.Add(ll);
711.         //    e3.Loads.Add(lr);
712.
713.         //    //var wnd = WpfTraceListener.CreateModelTrace(model);
714.         //    new ModelWarningChecker().CheckModel(model);
715.         //    //wnd.ShowDialog();
716.
717.         //    model.Solve();
718.
719.         //    var a = n3.GetNodalDisplacement();
720.
721.         //    var bcc = 0;
722.         //}
723.
724.         //private static void Example1()
725.         //{
726.         //    Console.WriteLine("Example 1: Simple 3D truss with
   four members");
727.
728.         //    // Initiating Model, Nodes and Members
729.         //    var model = new Model();
730.
731.         //    var n1 = new Node(1, 1, 0);
732.         //    n1.Label = "n1";//Set a unique label for node
```

```
733.         //    var n2 = new Node(-1, 1, 0) { Label = "n2" };//using
   object initializer for assigning Label
734.         //    var n3 = new Node(1, -1, 0) { Label = "n3" };
735.         //    var n4 = new Node(-1, -1, 0) { Label = "n4" };
736.         //    var n5 = new Node(0, 0, 1) { Label = "n5" };
737.
738.         //    var e1 = new TrussElement2Node(n1, n5) { Label = "e1"
   };
739.         //    var e2 = new TrussElement2Node(n2, n5) { Label = "e2"
   };
740.         //    var e3 = new TrussElement2Node(n3, n5) { Label = "e3"
   };
741.         //    var e4 = new TrussElement2Node(n4, n5) { Label = "e4"
   };
742.
743.         //    //Note: labels for all members should be unique,
744.         //    //else you will receive InvalidLabelException when
   adding it to model
745.
746.         //    e1.A = e2.A = e3.A = e4.A = 9e-4;
747.         //    e1.E = e2.E = e3.E = e4.E = 210e9;
748.
749.
750.         //    model.Nodes.Add(n1, n2, n3, n4, n5);
751.         //    model.Elements.Add(e1, e2, e3, e4);
752.
753.         //    //Applying restraints
754.
755.         //    n1.Constraints = n2.Constraints = n3.Constraints =
   n4.Constraints = Constraint.Fixed;
756.         //    n5.Constraints = Constraint.FixedRX &
   Constraint.FixedRY & Constraint.FixedRZ;/*
   Constraint.RotationFixed;*/
757.
758.         //    //Applying load
759.         //    var force = new Force(0, 1000, 0, 0, 0, 0);
760.         //    n5.Loads.Add(new NodalLoad(force));//adds a load with
   LoadCase of DefaultLoadCase to node loads
761.
762.         //    //Adds a NodalLoad with Default LoadCase
763.
764.         //    model.Solve();
765.
766.         //    var r1 = n1.GetSupportReaction();
767.         //    var r2 = n2.GetSupportReaction();
768.         //    var r3 = n3.GetSupportReaction();
769.         //    var r4 = n4.GetSupportReaction();
770.
771.         //    var dis = n5.GetNodalDisplacement();
772.
773.         //    var a = e1.GetInternalForceAt(1);
774.
775.         //    var rt = r1 + r2 + r3 + r4;//shows the Fz=1000 and
   Fx=Fy=Mx=My=Mz=0.0
776.
777.         //    Console.WriteLine("Total reactions SUM :" +
   rt.ToString());
778.         //}
779.
780.         private void InitializeSocket()
781.         {
```

```
782.             ActivateServer();
783.             CheckMesages();
784.         }
785.
786.         private void ActivateServer()
787.         {
788.             try
789.             {
790.                 IPAddress ipAd = IPAddress.Parse("127.0.0.1");
791.                 // use local m/c IP address, and
792.                 // use the same in the client
793.
794.                 /* Initializes the Listener */
795.                 myList = new TcpListener(ipAd, 8001);
796.
797.                 /* Start Listeneting at the specified port */
798.                 myList.Start();
799.
800.                 Console.WriteLine("The server is running at port
    8001...");
801.                 Console.WriteLine("The local End point is: " +
    myList.LocalEndpoint);
802.                 Console.WriteLine("Waiting for a connection...");
803.
804.                 s = myList.AcceptSocket();
805.
806.                 Console.WriteLine("Connection accepted from " +
    s.RemoteEndPoint);
807.             }
808.             catch (Exception e)
809.             {
810.                 Console.WriteLine("Error: " + e.StackTrace);
811.             }
812.         }
813.
814.         public void CheckMesages()
815.         {
816.             if (threadCommand != "")
817.             {
818.                 string[] commands = threadCommand.Split('&');
819.
820.                 threadCommand = "";
821.
822.
823.             }
824.
825.             new Thread(() =>
826.             {
827.                 Thread.CurrentThread.IsBackground = true;
828.
829.                 try
830.                 {
831.                     bool threadShouldContinue = true;
832.
833.                     while (threadShouldContinue)
834.                     {
835.
836.
837.                         byte[] b = new byte[131072];
838.
839.                         int k = s.Receive(b);
```

```
840.
841.                           string recived = "";
842.
843.                           for (int i = 0; i < k; i++)
844.                           {
845.                               recived += Convert.ToChar(b[i]);
846.                           }
847.
848.                           string[] commands = recived.Split('&');
849.
850.                           if (commands[0] == "No Nodes")
851.                           {
852.                               try
853.                               {
854.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
855.                                       DispatcherPriority.Normal,
856.                                       (Action)(() =>
857.                                       {
858.                                           xCoordNodeEdit.Text = "0
   node selected";
859.                                           yCoordNodeEdit.Text = "0
   node selected";
860.                                           zCoordNodeEdit.Text = "0
   node selected";
861.
862.                                           xCoordNodeEdit.IsEnabled =
   false;
863.                                           yCoordNodeEdit.IsEnabled =
   false;
864.                                           zCoordNodeEdit.IsEnabled =
   false;
865.                                       }
866.                                       ));
867.                               }
868.                               catch
869.                               {
870.
871.                               }
872.                           }
873.
874.                           if (commands[0] == "Edit Node")
875.                           {
876.                               try
877.                               {
878.                                   // Send message to UI thread
879.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
880.                                       DispatcherPriority.Normal,
881.                                       (Action)(() =>
882.                                           {
883.                                               threadCommand =
   recived;
884.
885.
   xCoordNodeEdit.IsEnabled = true;
886.
   yCoordNodeEdit.IsEnabled = true;
887.
   zCoordNodeEdit.IsEnabled = true;
888.
```

```
889.                                                xCoordNodeEdit.Text =
    nodes[int.Parse(commands[1])].position.x.ToString();
890.                                                yCoordNodeEdit.Text =
    nodes[int.Parse(commands[1])].position.y.ToString();
891.                                                zCoordNodeEdit.Text =
    nodes[int.Parse(commands[1])].position.z.ToString();
892.
893.                                                xDispEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.x1;
894.                                                yDispEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.y1;
895.                                                zDispEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.z1;
896.                                                xRotEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.x2;
897.                                                yRotEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.y2;
898.                                                zRotEdit.IsChecked =
    nodes[int.Parse(commands[1])].restriction.z2;
899.                                            }
900.                                        ));
901.                                    }
902.                                    catch
903.                                    {
904.
905.                                    }
906.                                }
907.
908.                                if (commands[0] == "Edit Nodes")
909.                                {
910.                                    try
911.                                    {
912.                                        // Send message to UI thread
913.
    System.Windows.Application.Current.Dispatcher.BeginInvoke(
914.                                            DispatcherPriority.Normal,
915.                                            (Action)(() =>
916.                                            {
917.                                                threadCommand = recived;
918.
919.                                                xCoordNodeEdit.Text =
    "Multiple nodes";
920.                                                yCoordNodeEdit.Text =
    "Multiple nodes";
921.                                                zCoordNodeEdit.Text =
    "Multiple nodes";
922.
923.                                                xCoordNodeEdit.IsEnabled =
    false;
924.                                                yCoordNodeEdit.IsEnabled =
    false;
925.                                                zCoordNodeEdit.IsEnabled =
    false;
926.                                            }
927.                                            ));
928.                                    }
929.                                    catch
930.                                    {
931.
932.                                    }
933.                                }
```

```
934.
935.                           if (commands[0] == "Create Member - No
   Node")
936.                           {
937.                               try
938.                               {
939.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
940.                                       DispatcherPriority.Normal,
941.                                       (Action)(() =>
942.                                       {
943.
   memberFirstNodeLabel.Content = "- First Node";
944.
   memberFirstNodeLabel.Foreground = Brushes.Red;
945.
946.                                           tempMemberN0 = -1;
947.                                           tempMemberN1 = -1;
948.                                       }
949.                                       ));
950.                               }
951.                               catch
952.                               {
953.
954.                               }
955.                           }
956.
957.                           if (commands[0] == "Create Member - N0")
958.                           {
959.                               try
960.                               {
961.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
962.                                       DispatcherPriority.Normal,
963.                                       (Action)(() =>
964.                                       {
965.
   memberFirstNodeLabel.Content = "- First Node: OK";
966.
   memberFirstNodeLabel.Foreground = Brushes.Black;
967.
968.
   memberSecondNodeLabel.Content = "- Second Node";
969.
   memberSecondNodeLabel.Foreground = Brushes.Red;
970.
971.                                           tempMemberN0 =
   int.Parse(commands[1]);
972.                                           tempMemberN1 = -1;
973.                                       }
974.                                       ));
975.                               }
976.                               catch
977.                               {
978.
979.                               }
980.                           }
981.
982.                           if (commands[0] == "Create Member - N1")
983.                           {
984.                               try
```

```
985.                                {
986.                                    System.Windows.Application.Current.Dispatcher.BeginInvoke(
987.                                        DispatcherPriority.Normal,
988.                                        (Action)(() =>
989.                                        {
990.                                            memberSecondNodeLabel.Content = "- Second Node: OK";
991.                                            memberSecondNodeLabel.Foreground = Brushes.Black;
992.                                            
993.                                            tempMemberN1 = int.Parse(commands[1]);
994.                                        }
995.                                        ));
996.                                }
997.                                catch
998.                                {
999.                                    
1000.                                   }
1001.                               }
1002.                               
1003.                               if (commands[0] == "Try Member Creation")
1004.                               {
1005.                                   try
1006.                                   {
1007.                                       System.Windows.Application.Current.Dispatcher.BeginInvoke(
1008.                                       DispatcherPriority.Normal,
1009.                                           (Action)(() =>
1010.                                           {
1011.                                               if (tempMemberN0 != -1 && tempMemberN1 != -1 && materialMemberComboBox.SelectedItem != null && propertyMemberComboBox.SelectedItem != null)
1012.                                               {
1013.                                                   members.Add(new Member(members.Count, tempMemberN0, tempMemberN1, materials[materialMemberComboBox.SelectedIndex].name, properties[propertyMemberComboBox.SelectedIndex].name, dayMemberStartText.Text, dayMemberEndText.Text));
1014.                                                   
1015.                                                   bool changed = false;
1016.                                                   
1017.                                                   do
1018.                                                   {
1019.                                                       changed = false;
1020.                                                       
1021.                                                       int memberCount = members.Count;
1022.                                                       
1023.                                                       for (int i = 0; i < memberCount; i++)
1024.                                                       {
1025.                                                           for (int j = i; j < memberCount; j++)
1026.                                                           {
```

```
1027.                                                                   if
   ((nodes[members[i].n0].position.y == nodes[members[j].n0].position.y)
   &&
1028.
   (nodes[members[i].n1].position.y == nodes[members[j].n1].position.y))
1029.                                                                   {
1030.
   double X = 0;
1031.
   double Z = 0;
1032.
1033.
   if (LineSegmentIntersection(
1034.
   nodes[members[i].n0].position.x, nodes[members[i].n0].position.z,
1035.
   nodes[members[i].n1].position.x, nodes[members[i].n1].position.z,
1036.
   nodes[members[j].n0].position.x, nodes[members[j].n0].position.z,
1037.
   nodes[members[j].n1].position.x, nodes[members[j].n1].position.z,
1038.
   ref X, ref Z
1039.
   ))
1040.
   {
1041.
   var tempVector3 = new Vector3(X, nodes[members[i].n0].position.y, Z);
1042.
1043.
   if (nodes[members[i].n0].position == tempVector3)
1044.
   {
1045.
   if ((nodes[members[j].n0].position == tempVector3) ||
   nodes[members[j].n1].position == tempVector3)
1046.
   {
1047.
1048.
   }
1049.
   else
1050.
   {
1051.
   int nodeExisting = members[i].n0;
1052.
1053.
   int tempNode = members[j].n1;
1054.
   members[j].n1 = nodeExisting;
1055.
1056.
   members.Add(new Member(members.Count, nodeExisting, tempNode,
   members[j].material, members[j].section,
1057.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1058.
```

```
1059.
   UpdateMemberForces(members[j].number, members.Count - 1);
1060.
   changed = true;
1061.
   }
1062.
   }
1063.
1064.
   else if (nodes[members[i].n1].position == tempVector3)
1065.
   {
1066.
   if ((nodes[members[j].n0].position == tempVector3) ||
   nodes[members[j].n1].position == tempVector3)
1067.
   {
1068.
1069.
   }
1070.
   else
1071.
   {
1072.
   int nodeExisting = members[i].n1;
1073.
1074.
   int tempNode = members[j].n1;
1075.
   members[j].n1 = nodeExisting;
1076.
1077.
   members.Add(new Member(members.Count, nodeExisting, tempNode,
   members[j].material, members[j].section,
1078.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1079.
1080.
   UpdateMemberForces(members[j].number, members.Count - 1);
1081.
   changed = true;
1082.
   }
1083.
   }
1084.
   else if (nodes[members[j].n0].position == tempVector3)
1085.
   {
1086.
   if ((nodes[members[i].n0].position == tempVector3) ||
   nodes[members[i].n1].position == tempVector3)
1087.
   {
1088.
1089.
   }
1090.
   else
```

```
1091.
    {
1092.
    int nodeExisting = members[j].n0;
1093.
1094.
    int tempNode = members[i].n1;
1095.
    members[i].n1 = nodeExisting;
1096.
1097.
    members.Add(new Member(members.Count, nodeExisting, tempNode,
    members[i].material, members[i].section,
1098.
    members[i].startDay.ToString(), members[i].endDay.ToString()));
1099.
1100.
    UpdateMemberForces(members[i].number, members.Count - 1);
1101.
    changed = true;
1102.
    }
1103.
    }
1104.
1105.
    else if (nodes[members[j].n1].position == tempVector3)
1106.
    {
1107.
    if ((nodes[members[i].n0].position == tempVector3) ||
    nodes[members[i].n1].position == tempVector3)
1108.
    {
1109.
1110.
    }
1111.
    else
1112.
    {
1113.
    int nodeExisting = members[j].n1;
1114.
1115.
    int tempNode = members[i].n1;
1116.
    members[i].n1 = nodeExisting;
1117.
1118.
    members.Add(new Member(members.Count, nodeExisting, tempNode,
    members[i].material, members[i].section,
1119.
    members[i].startDay.ToString(), members[i].endDay.ToString()));
1120.
1121.
    UpdateMemberForces(members[i].number, members.Count - 1);
1122.
    changed = true;
1123.
    }
```

```
1124.
   }
1125.
1126.
   else
1127.
   {
1128.
   nodes.Add(new Nodes(nodes.Count, tempVector3.x.ToString(),
   tempVector3.y.ToString(), tempVector3.z.ToString(), false, false,
   false, false, false, false));
1129.
1130.
   int newNode = nodes.Count - 1;
1131.
1132.
   // FIRST MEMBER:
1133.
   int tempNode = members[i].n1;
1134.
   members[i].n1 = newNode;
1135.
1136.
   members.Add(new Member(members.Count, newNode, tempNode,
   members[i].material, members[i].section,
1137.
   members[i].startDay.ToString(), members[i].endDay.ToString()));
1138.
1139.
   // SECOND MEMBER:
1140.
   tempNode = members[j].n1;
1141.
   members[j].n1 = newNode;
1142.
1143.
   members.Add(new Member(members.Count, newNode, tempNode,
   members[j].material, members[j].section,
1144.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1145.
1146.
   UpdateMemberForces(members[i].number, members.Count - 2);
1147.
   UpdateMemberForces(members[j].number, members.Count - 1);
1148.
1149.
   changed = true;
1150.
   }
1151.
   }
1152.                                                                    }
1153.                                                                    if
   (changed) break;
1154.                                                                 }
1155.                                                                 if
   (changed) break;
1156.                                                              }
1157.
```

```
1158.                                                   } while (changed
   == true);
1159.                                               }
1160.
1161.
   memberFirstNodeLabel.Content = "- First Node";
1162.
   memberFirstNodeLabel.Foreground = Brushes.Red;
1163.
1164.
   memberSecondNodeLabel.Content = "- Second Node";
1165.
   memberSecondNodeLabel.Foreground = Brushes.Red;
1166.
1167.                                               tempMemberN0 = -1;
1168.                                               tempMemberN1 = -1;
1169.
1170.                                               MassiveUpdate3DUI();
1171.                                           }
1172.                                           ));
1173.                                       }
1174.                                       catch
1175.                                       {
1176.
1177.                                       }
1178.                                   }
1179.
1180.                                   if (commands[0] == "Edit Member")
1181.                                   {
1182.                                       try
1183.                                       {
1184.                                           // Send message to UI thread
1185.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1186.
   DispatcherPriority.Normal,
1187.                                           (Action)(() =>
1188.                                           {
1189.                                               threadCommand =
   recived;
1190.
1191.
   materialMemberEditComboBox.SelectedItem =
   members[int.Parse(commands[1])].material;
1192.
   propertyMemberEditComboBox.SelectedItem =
   members[int.Parse(commands[1])].section;
1193.
1194.
   dayMemberStartEditText.Text =
   members[int.Parse(commands[1])].startDay.ToString();
1195.
   dayMemberEndEditText.Text =
   members[int.Parse(commands[1])].endDay.ToString();
1196.                                           }
1197.                                           ));
1198.                                       }
1199.                                       catch
1200.                                       {
1201.
1202.                                       }
```

```
1203.                                   }
1204.
1205.                                   if (commands[0] == "Edit Members")
1206.                                   {
1207.                                       try
1208.                                       {
1209.                                           // Send message to UI thread
1210.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1211.
   DispatcherPriority.Normal,
1212.                                                   (Action)(() =>
1213.                                                   {
1214.                                                       threadCommand =
   recived;
1215.                                                   }
1216.                                                   ));
1217.                                       }
1218.                                       catch
1219.                                       {
1220.
1221.                                       }
1222.                                   }
1223.
1224.                                   if (commands[0] == "No Nodal Force"
   || commands[0] == "Nodal Force")
1225.                                   {
1226.                                       try
1227.                                       {
1228.                                           // Send message to UI thread
1229.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1230.
   DispatcherPriority.Normal,
1231.                                                   (Action)(() =>
1232.                                                   {
1233.                                                       threadCommand =
   recived;
1234.                                                   }
1235.                                                   ));
1236.                                       }
1237.                                       catch
1238.                                       {
1239.
1240.                                       }
1241.                                   }
1242.
1243.                                   if (commands[0] == "No Member Force"
   || commands[0] == "Member Force")
1244.                                   {
1245.                                       try
1246.                                       {
1247.                                           // Send message to UI thread
1248.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1249.
   DispatcherPriority.Normal,
1250.                                                   (Action)(() =>
1251.                                                   {
1252.                                                       threadCommand =
   recived;
```

```
1253.                                                   }
1254.                                                   ));
1255.                                           }
1256.                                           catch
1257.                                           {
1258.
1259.                                           }
1260.                                   }
1261.
1262.                                   if (commands[0] == "Simulation
     Object")
1263.                                   {
1264.                                       try
1265.                                       {
1266.                                           // Send message to UI thread
1267.
     System.Windows.Application.Current.Dispatcher.BeginInvoke(
1268.
     DispatcherPriority.Normal,
1269.                                                   (Action)(() =>
1270.                                                   {
1271.                                                       if (commands[1] ==
     "Node")
1272.                                                       {
1273.
     UpdateSelected(commands[2]);
1274.                                                       }
1275.
1276.                                                       if (commands[1] ==
     "Member")
1277.                                                       {
1278.
     UpdateSelected("M" + commands[2]);
1279.                                                       }
1280.                                                   }
1281.                                                   ));
1282.                                           }
1283.                                           catch
1284.                                           {
1285.
1286.                                           }
1287.                                   }
1288.
1289.                                   if (commands[0] == "Selected
     Split:")
1290.                                   {
1291.                                       try
1292.                                       {
1293.                                           // Send message to UI thread
1294.
     System.Windows.Application.Current.Dispatcher.BeginInvoke(
1295.
     DispatcherPriority.Normal,
1296.                                                   (Action)(() =>
1297.                                                   {
1298.                                                       threadCommand =
     recived;
1299.                                                   }
1300.                                                   ));
1301.                                           }
1302.                                           catch
```

```
1303.                                       {
1304.
1305.                                       }
1306.                                   }
1307.
1308.
1309.                                   Console.WriteLine("Message: " +
   recived);
1310.                                   //ASCIIEncoding asen = new
   ASCIIEncoding();
1311.                                   //s.Send(asen.GetBytes("Message
   received"));
1312.                               }
1313.                           }
1314.                           catch (Exception e)
1315.                           {
1316.                               Console.WriteLine("Error... " +
   e.StackTrace);
1317.                           }
1318.
1319.                       }).Start();
1320.                   }
1321.
1322.                   private void UpdateMemberForces(int memb, int toAdd)
1323.                   {
1324.                       for (int i = 0; i < memberForces.Count; i++)
1325.                       {
1326.                           if (memberForces[i].members.Contains(memb))
   memberForces[i].members.Add(toAdd);
1327.                       }
1328.                   }
1329.
1330.                   private void UpdateSelected(string s)
1331.                   {
1332.                       if (s == "") return;
1333.
1334.                       int day = int.Parse(SimShowDay.Text) - startDay;
1335.
1336.                       if (s.ElementAt(0) == 'M')
1337.                       {
1338.                           string memberSelected = s;
1339.
1340.                           SimPlot.IsEnabled = false;
1341.
1342.                           lastSimulatedSelected = memberSelected;
1343.
1344.                           try
1345.                           {
1346.                               var elm =
   models[day].Elements.Where(item => item.Label ==
   memberSelected).First() as FrameElement2Node;
1347.
1348.                               SimulationElementResults.Text =
   string.Format("Member selected: {0}", memberSelected) +
   System.Environment.NewLine;
1349.                               SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1350.                               SimulationElementResults.Text += "Forces
   at beginning:" + System.Environment.NewLine;
1351.                               SimulationElementResults.Text +=
   "Internal Force (kN):" + System.Environment.NewLine;
```

```
1352.                               SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};",
   elm.GetInternalForceAt(0).Fx, elm.GetInternalForceAt(0).Fy,
   elm.GetInternalForceAt(0).Fz) + System.Environment.NewLine;
1353.                               SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1354.                               SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};",
   elm.GetInternalForceAt(0).Mx, elm.GetInternalForceAt(0).My,
   elm.GetInternalForceAt(0).Mz) + System.Environment.NewLine;
1355.                               SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1356.                               SimulationElementResults.Text += "Forces
   at end:" + System.Environment.NewLine;
1357.                               SimulationElementResults.Text +=
   "Internal Force (kN):" + System.Environment.NewLine;
1358.                               SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};",
   elm.GetInternalForceAt(1).Fx, elm.GetInternalForceAt(1).Fy,
   elm.GetInternalForceAt(1).Fz) + System.Environment.NewLine;
1359.                               SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1360.                               SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};",
   elm.GetInternalForceAt(1).Mx, elm.GetInternalForceAt(1).My,
   elm.GetInternalForceAt(1).Mz) + System.Environment.NewLine;
1361.                           }
1362.                           catch
1363.                           {
1364.                               SimulationElementResults.Text =
   string.Format("Member selected: {0}", memberSelected) +
   System.Environment.NewLine;
1365.                               SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1366.                               SimulationElementResults.Text += "Member
   does not yet exist!";
1367.                           }
1368.
1369.
1370.                   }
1371.                   else
1372.                   {
1373.                       string nodeSelected = s;
1374.
1375.                       lastSimulatedSelected = nodeSelected;
1376.
1377.                       SimPlot.IsEnabled = true;
1378.
1379.                       try
1380.                       {
1381.                           var disp = models[day].Nodes.Where(item
   => item.Label == nodeSelected).First().GetNodalDisplacement();
1382.                           var reac = models[day].Nodes.Where(item
   => item.Label == nodeSelected).First().GetSupportReaction();
1383.
1384.                           SimulationElementResults.Text =
   string.Format("Node selected: {0}", nodeSelected) +
   System.Environment.NewLine;
1385.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
```

```
1386.                           SimulationElementResults.Text +=
   "Displacement (m):" + System.Environment.NewLine;
1387.                           SimulationElementResults.Text +=
   string.Format("X: {0}; Y: {1}; Z: {2};", disp.DX, disp.DY, disp.DZ) +
   System.Environment.NewLine;
1388.                           SimulationElementResults.Text +=
   "Rotation (rad):" + System.Environment.NewLine;
1389.                           SimulationElementResults.Text +=
   string.Format("X: {0}; Y: {1}; Z: {2};", disp.RX, disp.RY, disp.RZ) +
   System.Environment.NewLine;
1390.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1391.                           SimulationElementResults.Text +=
   "Reactions (kN):" + System.Environment.NewLine;
1392.                           SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};", reac.Fx, reac.Fy,
   reac.Fz) + System.Environment.NewLine;
1393.                           SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1394.                           SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};", reac.Mx, reac.My,
   reac.Mz) + System.Environment.NewLine;
1395.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1396.                       }
1397.                       catch
1398.                       {
1399.                           SimulationElementResults.Text =
   string.Format("Node selected: {0}", nodeSelected) +
   System.Environment.NewLine;
1400.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1401.                           SimulationElementResults.Text += "Node
   does not yet exist!";
1402.                       }
1403.
1404.
1405.                   }
1406.               }
1407.
1408.               private void InitializeUnity3D()
1409.               {
1410.                   try
1411.                   {
1412.                       _panel = new System.Windows.Forms.Panel();
1413.                       wfhUnity3DCanvas.Child = _panel;
1414.
1415.                       //ProcessStartInfo psi = new
   ProcessStartInfo("notepad.exe");
1416.                       //_process = Process.Start(psi);
1417.                       //_process.StartInfo.UseShellExecute = true;
1418.                       //_process.StartInfo.CreateNoWindow = true;
1419.                       //_process.WaitForInputIdle();
1420.
1421.                       _process = new Process();
1422.                       string a =
   AppDomain.CurrentDomain.BaseDirectory;
1423.                       _process.StartInfo.FileName = a +
   "3DInterface\\3DInterface.exe";
1424.                       _process.StartInfo.Arguments = "-parentHWND
   " + _panel.Handle.ToInt32() + " " + Environment.CommandLine;
```

```
1425.                         _process.StartInfo.UseShellExecute = true;
1426.                         _process.StartInfo.CreateNoWindow = true;
1427.                         _process.Start();
1428.                         _process.WaitForInputIdle();
1429.
1430.                         SetParent(_process.MainWindowHandle,
   _panel.Handle);
1431.
1432.                         EnumChildWindows(_panel.Handle, WindowEnum,
   IntPtr.Zero);
1433.
1434.                         // remove control box
1435.                         //int style =
   GetWindowLong(_process.MainWindowHandle, GWL_STYLE);
1436.                         //style = style & ~WS_CAPTION &
   ~WS_THICKFRAME;
1437.                         //SetWindowLong(_process.MainWindowHandle,
   GWL_STYLE, style);
1438.
1439.                         // resize embedded application & refresh
1440.                         wfhUnityMockUpCanvas.SizeChanged +=
   WPFCanvas_SizeChanged;
1441.                     }
1442.                     catch (Exception e)
1443.                     {
1444.                     }
1445.                 }
1446.                 /// <summary>
1447.                 /// Main thread exception handler
1448.                 /// </summary>
1449.                 /// <param name="sender">sender</param>
1450.                 /// <param name="e">event</param>
1451.                 public void Application_ThreadException(object
   sender, System.Threading.ThreadExceptionEventArgs e)
1452.                 {
1453.                     if (_process != null)
1454.                     {
1455.                         _process.Refresh();
1456.                         _process.Close();
1457.                     }
1458.
1459.                     // Clean up sockets.
1460.                     s.Close();
1461.                     myList.Stop();
1462.                 }
1463.
1464.                 /// <summary>
1465.                 /// Application domain exception handler
1466.                 /// </summary>
1467.                 /// <param name="sender">sender</param>
1468.                 /// <param name="e">event</param>
1469.                 public void AppDomain_UnhandledException(object
   sender, System.UnhandledExceptionEventArgs e)
1470.                 {
1471.                     if (_process != null)
1472.                     {
1473.                         _process.Refresh();
1474.                         _process.Close();
1475.                     }
1476.
1477.                     // Clean up sockets.
```

```
1478.                   s.Close();
1479.                   myList.Stop();
1480.               }
1481.
1482.
1483.               protected override void
   OnClosing(System.ComponentModel.CancelEventArgs e)
1484.               {
1485.                   base.OnClosing(e);
1486.                   if (_process != null)
1487.                   {
1488.                       _process.Refresh();
1489.                       _process.Close();
1490.                   }
1491.
1492.                   // Clean up sockets.
1493.                   s.Close();
1494.                   myList.Stop();
1495.               }
1496.
1497.               private void ResizeEmbeddedApp()
1498.               {
1499.                   if (_process == null)
1500.                       return;
1501.
1502.                   MoveWindow(_unityHWND, 0, 0,
   (int)wfhUnityMockUpCanvas.ActualWidth,
   (int)wfhUnityMockUpCanvas.ActualHeight, true);
1503.               }
1504.
1505.               private void WPFCanvas_SizeChanged(object sender,
   SizeChangedEventArgs e)
1506.               {
1507.                   ResizeEmbeddedApp();
1508.               }
1509.
1510.               private int WindowEnum(IntPtr hwnd, IntPtr lparam)
1511.               {
1512.                   _unityHWND = hwnd;
1513.                   ActivateUnityWindow();
1514.                   return 0;
1515.               }
1516.
1517.               private void ActivateUnityWindow()
1518.               {
1519.                   SendMessage(_unityHWND, WM_ACTIVATE, WA_ACTIVE,
   IntPtr.Zero);
1520.               }
1521.
1522.               private void DeactivateUnityWindow()
1523.               {
1524.                   SendMessage(_unityHWND, WM_ACTIVATE,
   WA_INACTIVE, IntPtr.Zero);
1525.               }
1526.
1527.               private void Hand_Click(object sender,
   RoutedEventArgs e)
1528.               {
1529.                   //double[] time = new double[100];
1530.
1531.                   //Parallel.For(0, time.Length,
```

```
1532.                   //        index =>
1533.                   //        {
1534.                   //            StructuralAnalysisMathNet analysis
   = new StructuralAnalysisMathNet();
1535.                   //            time[index] = analysis.time;
1536.                   //        });
1537.
1538.                   //double result = 0;
1539.
1540.                   //for (int i = 0; i < time.Length; i++)
1541.                   //{
1542.                   //    result += time[i];
1543.                   //}
1544.
1545.                   //result = result / time.Length;
1546.
1547.                   //abc.Text = result.ToString();
1548.               }
1549.
1550.               private void MainWindow_Loaded(object sender,
   RoutedEventArgs e)
1551.               {
1552.                   ResizeEmbeddedApp();
1553.                   IntPtr hwnd = new
   WindowInteropHelper(this).Handle;
1554.                   HwndSource src = HwndSource.FromHwnd(hwnd);
1555.                   //Receive window message handler implementation
   (based on System.Windows.Interop.HwndSourceHook commissioned)
1556.                   src.AddHook(new HwndSourceHook(WndProc));
1557.               }
1558.
1559.               IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam,
   IntPtr lParam, ref bool handled)
1560.               {
1561.                   switch (msg)
1562.                   {
1563.                       case 528:
1564.                           // Mouse click on Unity Window
1565.                           ActivateUnityWindow();
1566.                           break;
1567.                   }
1568.                   return IntPtr.Zero;
1569.               }
1570.
1571.               private void SendData(byte[] data)
1572.               {
1573.                   SocketAsyncEventArgs socketAsyncData = new
   SocketAsyncEventArgs();
1574.                   socketAsyncData.SetBuffer(data, 0, data.Length);
1575.                   s.SendAsync(socketAsyncData);
1576.
1577.               }
1578.
1579.               public void CreateMaterial(object sender,
   RoutedEventArgs e)
1580.               {
1581.                   if (matName.Text != "" && matE.Text != "" &&
   matG.Text != "")
1582.                   {
1583.                       try
1584.                       {
```

```
1585.                             if (materials.Where(item => item.name ==
   matName.Text).ToList().Count == 0)
1586.                             {
1587.                                 if (fckText.Text == "") fckText.Text
   = "0";
1588.
1589.                                 materials.Add(new
   Material(matName.Text, matE.Text, matG.Text,
   isTimeDependent.IsChecked.GetValueOrDefault(), fckText.Text,
   matV.Text));
1590.
1591.                                 matName.Text = "";
1592.                                 matE.Text = "";
1593.                                 matG.Text = "";
1594.                                 matV.Text = "";
1595.
1596.                                 isTimeDependent.IsChecked = false;
1597.
1598.                                 fckText.Text = "";
1599.
1600.                                 labelWrongMatP.Content = "Material
   Saved!";
1601.                                 labelWrongMatP.Visibility =
   Visibility.Visible;
1602.                             }
1603.                             else
1604.                             {
1605.                                 labelWrongMatP.Content = "Wrong
   Properties!";
1606.                                 labelWrongMatP.Visibility =
   Visibility.Visible;
1607.                             }
1608.                         }
1609.                         catch
1610.                         {
1611.                             labelWrongMatP.Content = "Wrong
   Properties!";
1612.                             labelWrongMatP.Visibility =
   Visibility.Visible;
1613.                         }
1614.                     }
1615.                     else
1616.                     {
1617.                         labelWrongMatP.Content = "Wrong
   Properties!";
1618.                         labelWrongMatP.Visibility =
   Visibility.Visible;
1619.                     }
1620.                 }
1621.
1622.                 public void UpdateG(object sender, RoutedEventArgs
   e)
1623.                 {
1624.                     if (matE.Text != "" && matV.Text != "")
1625.                     {
1626.                         try
1627.                         {
1628.                             matG.Text = (double.Parse(matE.Text) /
   (2 * (1 + double.Parse(matV.Text)))).ToString();
1629.                         }
1630.                         catch
```

```
1631.                           {
1632.                               labelWrongMatP.Content = "Wrong
   Properties!";
1633.                               labelWrongMatP.Visibility =
   Visibility.Visible;
1634.                           }
1635.                       }
1636.                       else
1637.                       {
1638.                           labelWrongMatP.Content = "Wrong
   Properties!";
1639.                           labelWrongMatP.Visibility =
   Visibility.Visible;
1640.                       }
1641.                   }
1642.
1643.                   public void UpdateGEdit(object sender,
   RoutedEventArgs e)
1644.                   {
1645.                       if (matEEdit.Text != "" && matVEdit.Text != "")
1646.                       {
1647.                           try
1648.                           {
1649.                               matGEdit.Text =
   (double.Parse(matEEdit.Text) / (2 * (1 +
   double.Parse(matVEdit.Text)))).ToString();
1650.                           }
1651.                           catch
1652.                           {
1653.                               labelWrongMatPEdit.Content = "Wrong
   Properties!";
1654.                               labelWrongMatPEdit.Visibility =
   Visibility.Visible;
1655.                           }
1656.                       }
1657.                       else
1658.                       {
1659.                           labelWrongMatPEdit.Content = "Wrong
   Properties!";
1660.                           labelWrongMatPEdit.Visibility =
   Visibility.Visible;
1661.                       }
1662.                   }
1663.
1664.                   public void CreateNodalForce(object sender,
   RoutedEventArgs e)
1665.                   {
1666.                       if (nodalForceName.Text == "")
1667.                       {
1668.                           NodalForceError();
1669.                           return;
1670.                       }
1671.
1672.                       if (nodalForces.Where(item => item.name ==
   nodalForceName.Text).ToList().Count > 0)
1673.                       {
1674.                           NodalForceError();
1675.                           return;
1676.                       }
1677.
1678.                       if (threadCommand == "No Nodal Force")
```

```
1679.                     {
1680.                         NodalForceError();
1681.                         return;
1682.                     }
1683.
1684.                     try
1685.                     {
1686.                         if (nodalForceDisX.Text == "")
   nodalForceDisX.Text = "0";
1687.                         if (nodalForceDisY.Text == "")
   nodalForceDisY.Text = "0";
1688.                         if (nodalForceDisZ.Text == "")
   nodalForceDisZ.Text = "0";
1689.                         if (nodalForceRotX.Text == "")
   nodalForceRotX.Text = "0";
1690.                         if (nodalForceRotY.Text == "")
   nodalForceRotY.Text = "0";
1691.                         if (nodalForceRotZ.Text == "")
   nodalForceRotZ.Text = "0";
1692.
1693.                         double dx =
   double.Parse(nodalForceDisX.Text);
1694.                         double dy =
   double.Parse(nodalForceDisY.Text);
1695.                         double dz =
   double.Parse(nodalForceDisZ.Text);
1696.
1697.                         double rx =
   double.Parse(nodalForceRotX.Text);
1698.                         double ry =
   double.Parse(nodalForceRotY.Text);
1699.                         double rz =
   double.Parse(nodalForceRotZ.Text);
1700.
1701.                         int startDay =
   int.Parse(nodalForceStartDay.Text);
1702.                         int endDay =
   int.Parse(nodalForceEndDay.Text);
1703.
1704.                         List<string> listString =
   threadCommand.Split('&').ToList<string>();
1705.
1706.                         List<int> list = new List<int>();
1707.
1708.                         for (int i = 1; i < listString.Count; i++)
1709.                         {
1710.                             list.Add(int.Parse(listString[i]));
1711.                         }
1712.
1713.                         if (dx == 0 && dy == 0 && dz == 0 && rx == 0
   && ry == 0 && rz == 0)
1714.                         {
1715.                             NodalForceError();
1716.                         }
1717.
1718.                         nodalForces.Add(new
   NodalForces(nodalForceName.Text, dx, dy, dz, rx, ry, rz, startDay,
   endDay, list));
1719.
1720.                         nodalForceName.Text = "";
1721.                         nodalForceDisX.Text = "";
```

```
1722.                       nodalForceDisY.Text = "";
1723.                       nodalForceDisZ.Text = "";
1724.                       nodalForceRotX.Text = "";
1725.                       nodalForceRotY.Text = "";
1726.                       nodalForceRotZ.Text = "";
1727.                       nodalForceStartDay.Text = "";
1728.                       nodalForceEndDay.Text = "";
1729.
1730.                       nodalForceError.Visibility =
   Visibility.Collapsed;
1731.
1732.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation"));
1733.                   }
1734.                   catch
1735.                   {
1736.                       NodalForceError();
1737.                   }
1738.               }
1739.
1740.               public void AddNodalForceHeaderSelected(object
   sender, RoutedEventArgs e)
1741.               {
1742.                   SendStringToInterface(string.Format("Mode&Nodal
   Force Creation"));
1743.                   ForcesMainHeader.IsSelected = true;
1744.                   AddNodalForceHeader.IsSelected = true;
1745.
1746.                   nodalForceName.Text = "";
1747.                   nodalForceDisX.Text = "";
1748.                   nodalForceDisY.Text = "";
1749.                   nodalForceDisZ.Text = "";
1750.                   nodalForceRotX.Text = "";
1751.                   nodalForceRotY.Text = "";
1752.                   nodalForceRotZ.Text = "";
1753.                   nodalForceStartDay.Text = "";
1754.                   nodalForceEndDay.Text = "";
1755.
1756.                   nodalForceError.Visibility =
   Visibility.Collapsed;
1757.               }
1758.
1759.               public void EditNodalForcesHeaderSelected(object
   sender, RoutedEventArgs e)
1760.               {
1761.                   nodalForceEditComboBox.Items.Clear();
1762.
1763.                   for (int i = 0; i < nodalForces.Count; i++)
1764.                   {
1765.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1766.                   }
1767.
1768.                   nodalForceNameEdit.Text = "";
1769.                   nodalForceDisXEdit.Text = "";
1770.                   nodalForceDisYEdit.Text = "";
1771.                   nodalForceDisZEdit.Text = "";
1772.                   nodalForceRotXEdit.Text = "";
1773.                   nodalForceRotYEdit.Text = "";
1774.                   nodalForceRotZEdit.Text = "";
1775.                   nodalForceStartDayEdit.Text = "";
```

```
1776.                     nodalForceEndDayEdit.Text = "";
1777.
1778.                     nodalForceNameEdit.IsEnabled = false;
1779.                     nodalForceDisXEdit.IsEnabled = false;
1780.                     nodalForceDisYEdit.IsEnabled = false;
1781.                     nodalForceDisZEdit.IsEnabled = false;
1782.                     nodalForceRotXEdit.IsEnabled = false;
1783.                     nodalForceRotYEdit.IsEnabled = false;
1784.                     nodalForceRotZEdit.IsEnabled = false;
1785.                     nodalForceStartDayEdit.IsEnabled = false;
1786.                     nodalForceEndDayEdit.IsEnabled = false;
1787.
1788.                     nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1789.
1790.                     EditNodalForceHeader.IsSelected = true;
1791.                 }
1792.
1793.                 public void
   EditNodalForcesComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
1794.                 {
1795.                     if (!finishedLoading) return;
1796.
1797.                     if (nodalForceEditComboBox.SelectedItem != null)
1798.                     {
1799.                         nodalForceNameEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().name;
1800.                         nodalForceDisXEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.x.ToStri
   ng();
1801.                         nodalForceDisYEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.y.ToStri
   ng();
1802.                         nodalForceDisZEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.z.ToStri
   ng();
1803.                         nodalForceRotXEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.x.ToStri
   ng();
1804.                         nodalForceRotYEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.y.ToStri
   ng();
1805.                         nodalForceRotZEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.z.ToStri
   ng();
1806.                         nodalForceStartDayEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().startDay.ToSt
   ring();
1807.                         nodalForceEndDayEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().endDay.ToStri
   ng();
```

```
1808.
1809.                         nodalForceNameEdit.IsEnabled = true;
1810.                         nodalForceDisXEdit.IsEnabled = true;
1811.                         nodalForceDisYEdit.IsEnabled = true;
1812.                         nodalForceDisZEdit.IsEnabled = true;
1813.                         nodalForceRotXEdit.IsEnabled = true;
1814.                         nodalForceRotYEdit.IsEnabled = true;
1815.                         nodalForceRotZEdit.IsEnabled = true;
1816.                         nodalForceStartDayEdit.IsEnabled = true;
1817.                         nodalForceEndDayEdit.IsEnabled = true;
1818.                         nodalForcesEditSave.IsEnabled = true;
1819.                         nodalForcesEditDelete.IsEnabled = true;
1820.
1821.                         nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1822.
1823.                         string listString = "";
1824.
1825.                         for (int i = 0; i < nodalForces.Where(item
   => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes.Count;
   i++)
1826.                         {
1827.                             if (listString == "")
1828.                             {
1829.                                 listString += nodalForces.Where(item
   => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes[i].ToSt
   ring();
1830.                             }
1831.                             else
1832.                             {
1833.                                 listString += "&" +
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes[i].ToSt
   ring();
1834.                             }
1835.                         }
1836.
1837.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation&{0}",
   listString));
1838.                     }
1839.                     else
1840.                     {
1841.                         nodalForceNameEdit.Text = "";
1842.                         nodalForceDisXEdit.Text = "";
1843.                         nodalForceDisYEdit.Text = "";
1844.                         nodalForceDisZEdit.Text = "";
1845.                         nodalForceRotXEdit.Text = "";
1846.                         nodalForceRotYEdit.Text = "";
1847.                         nodalForceRotZEdit.Text = "";
1848.                         nodalForceStartDayEdit.Text = "";
1849.                         nodalForceEndDayEdit.Text = "";
1850.
1851.                         nodalForceNameEdit.IsEnabled = false;
1852.                         nodalForceDisXEdit.IsEnabled = false;
1853.                         nodalForceDisYEdit.IsEnabled = false;
1854.                         nodalForceDisZEdit.IsEnabled = false;
1855.                         nodalForceRotXEdit.IsEnabled = false;
1856.                         nodalForceRotYEdit.IsEnabled = false;
```

```
1857.                       nodalForceRotZEdit.IsEnabled = false;
1858.                       nodalForceStartDayEdit.IsEnabled = false;
1859.                       nodalForceEndDayEdit.IsEnabled = false;
1860.                       nodalForcesEditSave.IsEnabled = false;
1861.                       nodalForcesEditDelete.IsEnabled = false;
1862.
1863.                       nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1864.                   }
1865.               }
1866.
1867.               public void EditNodalForce(object sender,
   RoutedEventArgs e)
1868.               {
1869.                   if (nodalForceNameEdit.Text == "")
1870.                   {
1871.                       NodalForceErrorEdit();
1872.                       return;
1873.                   }
1874.
1875.                   if
   (nodalForceEditComboBox.SelectedItem.ToString() !=
   nodalForceNameEdit.Text && nodalForces.Where(item => item.name ==
   nodalForceNameEdit.Text).ToList().Count != 0)
1876.                   {
1877.                       NodalForceErrorEdit();
1878.                       return;
1879.                   }
1880.
1881.                   if (threadCommand == "No Nodal Force")
1882.                   {
1883.                       NodalForceErrorEdit();
1884.                       return;
1885.                   }
1886.
1887.                   try
1888.                   {
1889.                       if (nodalForceDisXEdit.Text == "")
   nodalForceDisXEdit.Text = "0";
1890.                       if (nodalForceDisYEdit.Text == "")
   nodalForceDisYEdit.Text = "0";
1891.                       if (nodalForceDisZEdit.Text == "")
   nodalForceDisZEdit.Text = "0";
1892.                       if (nodalForceRotXEdit.Text == "")
   nodalForceRotXEdit.Text = "0";
1893.                       if (nodalForceRotYEdit.Text == "")
   nodalForceRotYEdit.Text = "0";
1894.                       if (nodalForceRotZEdit.Text == "")
   nodalForceRotZEdit.Text = "0";
1895.
1896.                       double dx =
   double.Parse(nodalForceDisXEdit.Text);
1897.                       double dy =
   double.Parse(nodalForceDisYEdit.Text);
1898.                       double dz =
   double.Parse(nodalForceDisZEdit.Text);
1899.
1900.                       double rx =
   double.Parse(nodalForceRotXEdit.Text);
1901.                       double ry =
   double.Parse(nodalForceRotYEdit.Text);
```

```
1902.                         double rz =
   double.Parse(nodalForceRotZEdit.Text);
1903.
1904.                         int startDay =
   int.Parse(nodalForceStartDayEdit.Text);
1905.                         int endDay =
   int.Parse(nodalForceEndDayEdit.Text);
1906.
1907.                         List<string> listString =
   threadCommand.Split('&').ToList<string>();
1908.
1909.                         List<int> list = new List<int>();
1910.
1911.                         for (int i = 1; i < listString.Count; i++)
1912.                         {
1913.                             list.Add(int.Parse(listString[i]));
1914.                         }
1915.
1916.                         if (dx == 0 && dy == 0 && dz == 0 && rx == 0
   && ry == 0 && rz == 0)
1917.                         {
1918.                             NodalForceErrorEdit();
1919.                         }
1920.
1921.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.x = dx;
1922.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.y = dy;
1923.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.z = dz;
1924.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.x = rx;
1925.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.y = ry;
1926.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.z = rz;
1927.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().startDay =
   startDay;
1928.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().endDay =
   endDay;
1929.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes.Clear()
   ;
1930.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes = list;
1931.
1932.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().name =
   nodalForceNameEdit.Text;
1933.
1934.                         nodalForceNameEdit.Text = "";
1935.                         nodalForceDisXEdit.Text = "";
1936.                         nodalForceDisYEdit.Text = "";
1937.                         nodalForceDisZEdit.Text = "";
1938.                         nodalForceRotXEdit.Text = "";
1939.                         nodalForceRotYEdit.Text = "";
1940.                         nodalForceRotZEdit.Text = "";
1941.                         nodalForceStartDayEdit.Text = "";
1942.                         nodalForceEndDayEdit.Text = "";
```

```
1943.
1944.                       nodalForceNameEdit.IsEnabled = false;
1945.                       nodalForceDisXEdit.IsEnabled = false;
1946.                       nodalForceDisYEdit.IsEnabled = false;
1947.                       nodalForceDisZEdit.IsEnabled = false;
1948.                       nodalForceRotXEdit.IsEnabled = false;
1949.                       nodalForceRotYEdit.IsEnabled = false;
1950.                       nodalForceRotZEdit.IsEnabled = false;
1951.                       nodalForceStartDayEdit.IsEnabled = false;
1952.                       nodalForceEndDayEdit.IsEnabled = false;
1953.                       nodalForcesEditSave.IsEnabled = false;
1954.                       nodalForcesEditDelete.IsEnabled = false;
1955.
1956.                       nodalForceEditComboBox.Items.Clear();
1957.
1958.                       for (int i = 0; i < nodalForces.Count; i++)
1959.                       {
1960.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1961.                       }
1962.
1963.                       nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1964.
1965.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation"));
1966.                   }
1967.                   catch
1968.                   {
1969.                       NodalForceErrorEdit();
1970.                   }
1971.               }
1972.
1973.               public void DeleteNodalForce(object sender,
   RoutedEventArgs e)
1974.               {
1975.                   nodalForces.Remove(nodalForces.Where(item =>
   item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First());
1976.
1977.                   nodalForceEditComboBox.Items.Clear();
1978.
1979.                   for (int i = 0; i < nodalForces.Count; i++)
1980.                   {
1981.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1982.                   }
1983.
1984.                   nodalForceNameEdit.Text = "";
1985.                   nodalForceDisXEdit.Text = "";
1986.                   nodalForceDisYEdit.Text = "";
1987.                   nodalForceDisZEdit.Text = "";
1988.                   nodalForceRotXEdit.Text = "";
1989.                   nodalForceRotYEdit.Text = "";
1990.                   nodalForceRotZEdit.Text = "";
1991.                   nodalForceStartDayEdit.Text = "";
1992.                   nodalForceEndDayEdit.Text = "";
1993.
1994.                   nodalForceNameEdit.IsEnabled = false;
1995.                   nodalForceDisXEdit.IsEnabled = false;
1996.                   nodalForceDisYEdit.IsEnabled = false;
```

```
1997.                   nodalForceDisZEdit.IsEnabled = false;
1998.                   nodalForceRotXEdit.IsEnabled = false;
1999.                   nodalForceRotYEdit.IsEnabled = false;
2000.                   nodalForceRotZEdit.IsEnabled = false;
2001.                   nodalForceStartDayEdit.IsEnabled = false;
2002.                   nodalForceEndDayEdit.IsEnabled = false;
2003.
2004.                   nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
2005.
2006.                   EditNodalForceHeader.IsSelected = true;
2007.
2008.                   SendStringToInterface(string.Format("Mode&Nodal
   Force Creation"));
2009.               }
2010.
2011.               public void NodalForceError()
2012.               {
2013.                   nodalForceError.Visibility = Visibility.Visible;
2014.               }
2015.
2016.               public void NodalForceErrorEdit()
2017.               {
2018.                   nodalForceErrorEdit.Visibility =
   Visibility.Visible;
2019.               }
2020.
2021.
2022.
2023.               public void CreateMemberForce(object sender,
   RoutedEventArgs e)
2024.               {
2025.                   if (memberForceName.Text == "")
2026.                   {
2027.                       MemberForceError();
2028.                       return;
2029.                   }
2030.
2031.                   if (memberForces.Where(item => item.name ==
   memberForceName.Text).ToList().Count > 0)
2032.                   {
2033.                       MemberForceError();
2034.                       return;
2035.                   }
2036.
2037.                   if (threadCommand == "No Member Force")
2038.                   {
2039.                       MemberForceError();
2040.                       return;
2041.                   }
2042.
2043.                   try
2044.                   {
2045.                       if (memberForceDisX.Text == "")
   memberForceDisX.Text = "0";
2046.                       if (memberForceDisY.Text == "")
   memberForceDisY.Text = "0";
2047.                       if (memberForceDisZ.Text == "")
   memberForceDisZ.Text = "0";
2048.
```

```
2049.                         double dx =
   double.Parse(memberForceDisX.Text);
2050.                         double dy =
   double.Parse(memberForceDisY.Text);
2051.                         double dz =
   double.Parse(memberForceDisZ.Text);
2052.
2053.                         int startDay =
   int.Parse(memberForceStartDay.Text);
2054.                         int endDay =
   int.Parse(memberForceEndDay.Text);
2055.
2056.                         List<string> listString =
   threadCommand.Split('&').ToList<string>();
2057.
2058.                         List<int> list = new List<int>();
2059.
2060.                         for (int i = 1; i < listString.Count; i++)
2061.                         {
2062.                             list.Add(int.Parse(listString[i]));
2063.                         }
2064.
2065.                         if (dx == 0 && dy == 0 && dz == 0)
2066.                         {
2067.                             MemberForceError();
2068.                         }
2069.
2070.                         memberForces.Add(new
   MemberForces(memberForceName.Text, dx, dy, dz, startDay, endDay,
   list));
2071.
2072.                         memberForceName.Text = "";
2073.                         memberForceDisX.Text = "";
2074.                         memberForceDisY.Text = "";
2075.                         memberForceDisZ.Text = "";
2076.                         memberForceStartDay.Text = "";
2077.                         memberForceEndDay.Text = "";
2078.
2079.                         memberForceError.Visibility =
   Visibility.Collapsed;
2080.
2081.
   SendStringToInterface(string.Format("Mode&Member Force Creation"));
2082.                     }
2083.                     catch
2084.                     {
2085.                         MemberForceError();
2086.                     }
2087.                 }
2088.
2089.                 public void AddMemberForceHeaderSelected(object
   sender, RoutedEventArgs e)
2090.                 {
2091.                     SendStringToInterface(string.Format("Mode&Member
   Force Creation"));
2092.                     ForcesMainHeader.IsSelected = true;
2093.                     AddMemberForceHeader.IsSelected = true;
2094.
2095.                     memberForceName.Text = "";
2096.                     memberForceDisX.Text = "";
2097.                     memberForceDisY.Text = "";
```

```
2098.                   memberForceDisZ.Text = "";
2099.                   memberForceStartDay.Text = "";
2100.                   memberForceEndDay.Text = "";
2101.
2102.                   memberForceError.Visibility =
   Visibility.Collapsed;
2103.               }
2104.
2105.               public void EditMemberForcesHeaderSelected(object
   sender, RoutedEventArgs e)
2106.               {
2107.                   memberForceEditComboBox.Items.Clear();
2108.
2109.                   for (int i = 0; i < memberForces.Count; i++)
2110.                   {
2111.
   memberForceEditComboBox.Items.Add(memberForces[i].name);
2112.                   }
2113.
2114.                   memberForceNameEdit.Text = "";
2115.                   memberForceDisXEdit.Text = "";
2116.                   memberForceDisYEdit.Text = "";
2117.                   memberForceDisZEdit.Text = "";
2118.                   memberForceStartDayEdit.Text = "";
2119.                   memberForceEndDayEdit.Text = "";
2120.
2121.                   memberForceNameEdit.IsEnabled = false;
2122.                   memberForceDisXEdit.IsEnabled = false;
2123.                   memberForceDisYEdit.IsEnabled = false;
2124.                   memberForceDisZEdit.IsEnabled = false;
2125.                   memberForceStartDayEdit.IsEnabled = false;
2126.                   memberForceEndDayEdit.IsEnabled = false;
2127.
2128.                   memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2129.
2130.                   EditMemberForceHeader.IsSelected = true;
2131.               }
2132.
2133.               public void
   EditMemberForcesComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
2134.               {
2135.                   if (!finishedLoading) return;
2136.
2137.                   if (memberForceEditComboBox.SelectedItem !=
   null)
2138.                   {
2139.                       memberForceNameEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().name;
2140.                       memberForceDisXEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.x.ToStr
   ing();
2141.                       memberForceDisYEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.y.ToStr
   ing();
2142.                       memberForceDisZEdit.Text =
   memberForces.Where(item => item.name ==
```

```
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.z.ToStr
   ing();
2143.                       memberForceStartDayEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().startDay.ToS
   tring();
2144.                       memberForceEndDayEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().endDay.ToStr
   ing();
2145.
2146.                       memberForceNameEdit.IsEnabled = true;
2147.                       memberForceDisXEdit.IsEnabled = true;
2148.                       memberForceDisYEdit.IsEnabled = true;
2149.                       memberForceDisZEdit.IsEnabled = true;
2150.                       memberForceStartDayEdit.IsEnabled = true;
2151.                       memberForceEndDayEdit.IsEnabled = true;
2152.                       memberForcesEditSave.IsEnabled = true;
2153.                       memberForcesEditDelete.IsEnabled = true;
2154.
2155.                       memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2156.
2157.                       string listString = "";
2158.
2159.                       for (int i = 0; i < memberForces.Where(item
   => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members.Coun
   t; i++)
2160.                       {
2161.                           if (listString == "")
2162.                           {
2163.                               listString +=
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members[i].T
   oString();
2164.                           }
2165.                           else
2166.                           {
2167.                               listString += "&" +
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members[i].T
   oString();
2168.                           }
2169.                       }
2170.
2171.
   SendStringToInterface(string.Format("Mode&Member Force Creation&{0}",
   listString));
2172.                   }
2173.                   else
2174.                   {
2175.                       memberForceNameEdit.Text = "";
2176.                       memberForceDisXEdit.Text = "";
2177.                       memberForceDisYEdit.Text = "";
2178.                       memberForceDisZEdit.Text = "";
2179.                       memberForceStartDayEdit.Text = "";
2180.                       memberForceEndDayEdit.Text = "";
2181.
2182.                       memberForceNameEdit.IsEnabled = false;
2183.                       memberForceDisXEdit.IsEnabled = false;
```

```
2184.                         memberForceDisYEdit.IsEnabled = false;
2185.                         memberForceDisZEdit.IsEnabled = false;
2186.                         memberForceStartDayEdit.IsEnabled = false;
2187.                         memberForceEndDayEdit.IsEnabled = false;
2188.                         memberForcesEditSave.IsEnabled = false;
2189.                         memberForcesEditDelete.IsEnabled = false;
2190.
2191.                         memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2192.                     }
2193.                 }
2194.
2195.                 public void EditMemberForce(object sender,
   RoutedEventArgs e)
2196.                 {
2197.                     if (memberForceNameEdit.Text == "")
2198.                     {
2199.                         MemberForceErrorEdit();
2200.                         return;
2201.                     }
2202.
2203.                     if
   (memberForceEditComboBox.SelectedItem.ToString() !=
   memberForceNameEdit.Text && memberForces.Where(item => item.name ==
   memberForceNameEdit.Text).ToList().Count != 0)
2204.                     {
2205.                         MemberForceErrorEdit();
2206.                         return;
2207.                     }
2208.
2209.                     if (threadCommand == "No Member Force")
2210.                     {
2211.                         MemberForceErrorEdit();
2212.                         return;
2213.                     }
2214.
2215.                     try
2216.                     {
2217.                         if (memberForceDisXEdit.Text == "")
   memberForceDisXEdit.Text = "0";
2218.                         if (memberForceDisYEdit.Text == "")
   memberForceDisYEdit.Text = "0";
2219.                         if (memberForceDisZEdit.Text == "")
   memberForceDisZEdit.Text = "0";
2220.
2221.                         double dx =
   double.Parse(memberForceDisXEdit.Text);
2222.                         double dy =
   double.Parse(memberForceDisYEdit.Text);
2223.                         double dz =
   double.Parse(memberForceDisZEdit.Text);
2224.
2225.                         int startDay =
   int.Parse(memberForceStartDayEdit.Text);
2226.                         int endDay =
   int.Parse(memberForceEndDayEdit.Text);
2227.
2228.                         List<string> listString =
   threadCommand.Split('&').ToList<string>();
2229.
2230.                         List<int> list = new List<int>();
```

```
2231.
2232.                       for (int i = 1; i < listString.Count; i++)
2233.                       {
2234.                           if (listString[i] != "")
2235.                           {
2236.                               list.Add(int.Parse(listString[i]));
2237.                           }
2238.                       }
2239.
2240.                       if (dx == 0 && dy == 0 && dz == 0)
2241.                       {
2242.                           MemberForceErrorEdit();
2243.                       }
2244.
2245.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.x = dx;
2246.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.y = dy;
2247.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.z = dz;
2248.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().startDay =
   startDay;
2249.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().endDay =
   endDay;
2250.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members.Clea
   r();
2251.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members =
   list;
2252.
2253.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().name =
   memberForceNameEdit.Text;
2254.
2255.                       memberForceNameEdit.Text = "";
2256.                       memberForceDisXEdit.Text = "";
2257.                       memberForceDisYEdit.Text = "";
2258.                       memberForceDisZEdit.Text = "";
2259.                       memberForceStartDayEdit.Text = "";
2260.                       memberForceEndDayEdit.Text = "";
2261.
2262.                       memberForceNameEdit.IsEnabled = false;
2263.                       memberForceDisXEdit.IsEnabled = false;
2264.                       memberForceDisYEdit.IsEnabled = false;
2265.                       memberForceDisZEdit.IsEnabled = false;
2266.                       memberForceStartDayEdit.IsEnabled = false;
2267.                       memberForceEndDayEdit.IsEnabled = false;
2268.                       memberForcesEditSave.IsEnabled = false;
2269.                       memberForcesEditDelete.IsEnabled = false;
2270.
2271.                       memberForceEditComboBox.Items.Clear();
2272.
2273.                       for (int i = 0; i < memberForces.Count; i++)
2274.                       {
2275.
   memberForceEditComboBox.Items.Add(memberForces[i].name);
2276.                       }
2277.
```

```
2278.                       memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2279.
2280.
   SendStringToInterface(string.Format("Mode&Member Force Creation"));
2281.                   }
2282.                   catch
2283.                   {
2284.                       MemberForceErrorEdit();
2285.                   }
2286.               }
2287.
2288.               public void DeleteMemberForce(object sender,
   RoutedEventArgs e)
2289.               {
2290.                   memberForces.Remove(memberForces.Where(item =>
   item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First());
2291.
2292.                   memberForceEditComboBox.Items.Clear();
2293.
2294.                   for (int i = 0; i < memberForces.Count; i++)
2295.                   {
2296.
   memberForceEditComboBox.Items.Add(memberForces[i].name);
2297.                   }
2298.
2299.                   memberForceNameEdit.Text = "";
2300.                   memberForceDisXEdit.Text = "";
2301.                   memberForceDisYEdit.Text = "";
2302.                   memberForceDisZEdit.Text = "";
2303.                   memberForceStartDayEdit.Text = "";
2304.                   memberForceEndDayEdit.Text = "";
2305.
2306.                   memberForceNameEdit.IsEnabled = false;
2307.                   memberForceDisXEdit.IsEnabled = false;
2308.                   memberForceDisYEdit.IsEnabled = false;
2309.                   memberForceDisZEdit.IsEnabled = false;
2310.                   memberForceStartDayEdit.IsEnabled = false;
2311.                   memberForceEndDayEdit.IsEnabled = false;
2312.
2313.                   memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2314.
2315.                   EditMemberForceHeader.IsSelected = true;
2316.
2317.                   SendStringToInterface(string.Format("Mode&Member
   Force Creation"));
2318.               }
2319.
2320.               public void MemberForceError()
2321.               {
2322.                   nodalForceError.Visibility = Visibility.Visible;
2323.               }
2324.
2325.               public void MemberForceErrorEdit()
2326.               {
2327.                   nodalForceErrorEdit.Visibility =
   Visibility.Visible;
2328.               }
2329.
```

```
2330.               public void CreateNode(object sender,
    RoutedEventArgs e)
2331.               {
2332.                   try
2333.                   {
2334.                       nodes.Add(new Nodes(nodes.Count,
    xCoordNode.Text, yCoordNode.Text, zCoordNode.Text, xDisp.IsChecked,
    yDisp.IsChecked, zDisp.IsChecked, xRot.IsChecked, yRot.IsChecked,
    zRot.IsChecked));
2335.
2336.                       SendStringToInterface(string.Format("Create
    Node&{0}&{1}&{2}&{3}&{4}", nodes.Count - 1, xCoordNode.Text,
    yCoordNode.Text, zCoordNode.Text, xDisp.IsChecked.GetValueOrDefault()
    || yDisp.IsChecked.GetValueOrDefault() ||
    zDisp.IsChecked.GetValueOrDefault() ||
    xRot.IsChecked.GetValueOrDefault() ||
    yRot.IsChecked.GetValueOrDefault() ||
    zRot.IsChecked.GetValueOrDefault()));
2337.                   }
2338.                   catch
2339.                   {
2340.
2341.                   }
2342.               }
2343.
2344.               public void EditNode(object sender, RoutedEventArgs
    e)
2345.               {
2346.                   if (xCoordNodeEdit.Text == "0 node selected" ||
    xCoordNodeEdit.Text == "Multiple nodes")
2347.                   {
2348.                       if (xCoordNodeEdit.Text == "Multiple nodes")
2349.                       {
2350.                           List<string> tempList =
    threadCommand.Split('&').ToList<string>();
2351.
2352.                           for (int i = 1; i < tempList.Count; i++)
2353.                           {
2354.
    nodes[int.Parse(tempList[i])].restriction = new
    Restrictions(xDispEdit.IsChecked.GetValueOrDefault(),
    yDispEdit.IsChecked.GetValueOrDefault(),
    zDispEdit.IsChecked.GetValueOrDefault(),
    xRotEdit.IsChecked.GetValueOrDefault(),
    yRotEdit.IsChecked.GetValueOrDefault(),
    zRotEdit.IsChecked.GetValueOrDefault());
2355.                           }
2356.                       }
2357.
2358.                       //SendStringToInterface(string.Format("Edit
    Node Support&{0}", xDispEdit.IsChecked.GetValueOrDefault() ||
    yDispEdit.IsChecked.GetValueOrDefault() ||
    zDispEdit.IsChecked.GetValueOrDefault() ||
    xRotEdit.IsChecked.GetValueOrDefault() ||
    yRotEdit.IsChecked.GetValueOrDefault() ||
    zRotEdit.IsChecked.GetValueOrDefault()));
2359.                   }
2360.                   else
2361.                   {
2362.                       List<string> tempList =
    threadCommand.Split('&').ToList<string>();
```

```
2363.
2364.                       nodes[int.Parse(tempList[1])].position = new
   Vector3(xCoordNodeEdit.Text, yCoordNodeEdit.Text,
   zCoordNodeEdit.Text);
2365.                       nodes[int.Parse(tempList[1])].restriction =
   new Restrictions(xDispEdit.IsChecked.GetValueOrDefault(),
   yDispEdit.IsChecked.GetValueOrDefault(),
   zDispEdit.IsChecked.GetValueOrDefault(),
   xRotEdit.IsChecked.GetValueOrDefault(),
   yRotEdit.IsChecked.GetValueOrDefault(),
   zRotEdit.IsChecked.GetValueOrDefault());
2366.
2367.                       //SendStringToInterface(string.Format("Edit
   Node&{0}&{1}&{2}&{3}", xCoordNodeEdit.Text, yCoordNodeEdit.Text,
   zCoordNodeEdit.Text, xDispEdit.IsChecked.GetValueOrDefault() ||
   yDispEdit.IsChecked.GetValueOrDefault() ||
   zDispEdit.IsChecked.GetValueOrDefault() ||
   xRotEdit.IsChecked.GetValueOrDefault() ||
   yRotEdit.IsChecked.GetValueOrDefault() ||
   zRotEdit.IsChecked.GetValueOrDefault()));
2368.                   }
2369.
2370.                   MassiveUpdate3DUI();
2371.               }
2372.
2373.               public void DeleteNode(object sender,
   RoutedEventArgs e)
2374.               {
2375.                   try
2376.                   {
2377.                       List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2378.
2379.                       int count = 0;
2380.
2381.                       for (int i = 1; i < tempList.Count; i++)
2382.                       {
2383.                           for (int j = 0; j < members.Count; j++)
2384.                           {
2385.                               bool remove = false;
2386.
2387.                               if (members[j].n0 ==
   int.Parse(tempList[i]))
2388.                               {
2389.                                   remove = true;
2390.                               }
2391.                               if (members[j].n1 ==
   int.Parse(tempList[i]))
2392.                               {
2393.                                   remove = true;
2394.                               }
2395.
2396.                               if (remove)
2397.                               {
2398.                                   members.RemoveAt(j);
2399.
2400.                                   for (int k = 0; k <
   memberForces.Count; k++)
2401.                                   {
2402.                                       if
   (memberForces[k].members.Contains(j))
```

```
2403.                                           {
2404.
   memberForces[k].members.Remove(j);
2405.
2406.                                               for (int l = 0; l <
   memberForces[k].members.Count; l++)
2407.                                               {
2408.                                                   if
   (memberForces[k].members[l] > j)
2409.                                                   {
2410.
   memberForces[k].members[l]--;
2411.                                                   }
2412.                                               }
2413.                                           }
2414.                                           else
2415.                                           {
2416.                                               for (int l = 0; l <
   memberForces[k].members.Count; l++)
2417.                                               {
2418.                                                   if
   (memberForces[k].members[l] > j)
2419.                                                   {
2420.
   memberForces[k].members[l]--;
2421.                                                   }
2422.                                               }
2423.                                           }
2424.
2425.
2426.                                       }
2427.
2428.                                       j--;
2429.                                   }
2430.                               }
2431.
2432.                               for (int j = 0; j < nodalForces.Count;
   j++)
2433.                               {
2434.                                   if
   (nodalForces[j].nodes.Contains(int.Parse(tempList[i])))
   nodalForces[j].nodes.Remove(int.Parse(tempList[i]));
2435.                               }
2436.
2437.                               for (int j = 0; j < memberForces.Count;
   j++)
2438.                               {
2439.                                   if (memberForces[j].members.Count ==
   0) memberForces.RemoveAt(j);
2440.                               }
2441.
2442.                               nodes.RemoveAt(int.Parse(tempList[i]) -
   count);
2443.                               count++;
2444.                           }
2445.
2446.                           for (int i = 0; i < nodes.Count; i++)
2447.                           {
2448.                               for (int j = 0; j < members.Count; j++)
2449.                               {
```

```
2450.                                 if (members[j].n0 ==
   nodes[i].number) members[j].n0 = i;
2451.                                 if (members[j].n1 ==
   nodes[i].number) members[j].n1 = i;
2452.                             }
2453.
2454.                             nodes[i].number = i;
2455.                         }
2456.
2457.                         for (int i = 0; i < members.Count; i++)
2458.                         {
2459.                             members[i].number = i;
2460.                         }
2461.
2462.                         for (int j = 0; j < nodalForces.Count; j++)
2463.                         {
2464.                             if (nodalForces[j].nodes.Count == 0)
2465.                             {
2466.                                 nodalForces.RemoveAt(j);
2467.                                 j--;
2468.                             }
2469.                         }
2470.
2471.                         xCoordNodeEdit.Text = "0 node selected";
2472.                         yCoordNodeEdit.Text = "0 node selected";
2473.                         zCoordNodeEdit.Text = "0 node selected";
2474.
2475.                         xCoordNodeEdit.IsEnabled = false;
2476.                         yCoordNodeEdit.IsEnabled = false;
2477.                         zCoordNodeEdit.IsEnabled = false;
2478.
2479.
   //SendStringToInterface(string.Format("Delete Node"));
2480.
2481.                         MassiveUpdate3DUI();
2482.                     }
2483.                     catch
2484.                     {
2485.
2486.                     }
2487.                 }
2488.
2489.                 public void MainHeaderSelectedMouse(object sender,
   MouseButtonEventArgs e)
2490.                 {
2491.                     MainHeader();
2492.                 }
2493.
2494.                 public void MainHeaderSelected(object sender,
   RoutedEventArgs e)
2495.                 {
2496.                     MainHeader();
2497.                 }
2498.
2499.                 public void MainHeader()
2500.                 {
2501.
   SendStringToInterface(string.Format("Mode&General"));
2502.
2503.                     if (GeometryMainHeader.IsSelected)
2504.                     {
```

```
2505.                       GeometryGeneralHeader.IsSelected = true;
2506.                   }
2507.
2508.                   if (PropertiesMainHeader.IsSelected)
2509.                   {
2510.                       PropertiesGeneralHeader.IsSelected = true;
2511.                   }
2512.
2513.                   if (ForcesMainHeader.IsSelected)
2514.                   {
2515.                       ForcesGeneralHeader.IsSelected = true;
2516.                   }
2517.
2518.                   if (SimulationMainHeader.IsSelected)
2519.                   {
2520.                       SimulationGeneral();
2521.                   }
2522.               }
2523.
2524.               public void SimulationGeneralHeaderSelected(object
   sender, RoutedEventArgs e)
2525.               {
2526.                   SimulationGeneral();
2527.               }
2528.
2529.               public void SimulationClear(object sender,
   RoutedEventArgs e)
2530.               {
2531.                   simulationSuccess = false;
2532.
2533.                   models.Clear();
2534.
2535.                   SimulationGeneral();
2536.               }
2537.
2538.               public void SimulationGeneral()
2539.               {
2540.                   SimulationGeneralHeader.IsSelected = true;
2541.
2542.                   simulationDayStart.Text = startDay.ToString();
2543.                   simulationDayEnd.Text = endDay.ToString();
2544.
2545.                   if (!simulationSuccess)
2546.                   {
2547.                       SimSep1.Visibility = Visibility.Collapsed;
2548.                       SimLabelDay.Visibility =
   Visibility.Collapsed;
2549.                       SimShowDay.Visibility =
   Visibility.Collapsed;
2550.                       SimUpdate.Visibility = Visibility.Collapsed;
2551.                       SimSep2.Visibility = Visibility.Collapsed;
2552.                       SimSlider.Visibility = Visibility.Collapsed;
2553.                       SimLabelObject.Visibility =
   Visibility.Collapsed;
2554.                       SimPlot.Visibility = Visibility.Collapsed;
2555.                       SimulationElementResults.Visibility =
   Visibility.Collapsed;
2556.                   }
2557.               }
2558.
```

```
2559.                public void AddNodeHeaderSelected(object sender,
   RoutedEventArgs e)
2560.                {
2561.                    SendStringToInterface(string.Format("Mode&Node
   Creation"));
2562.                    GeometryMainHeader.IsSelected = true;
2563.                    AddNodeHeader.IsSelected = true;
2564.                }
2565.
2566.                public void EditNodeHeaderSelected(object sender,
   RoutedEventArgs e)
2567.                {
2568.                    SendStringToInterface(string.Format("Mode&Node
   Editor"));
2569.                    GeometryMainHeader.IsSelected = true;
2570.                    EditNodeHeader.IsSelected = true;
2571.
2572.                    xCoordNodeEdit.Text = "0 node selected";
2573.                    yCoordNodeEdit.Text = "0 node selected";
2574.                    zCoordNodeEdit.Text = "0 node selected";
2575.
2576.                    xCoordNodeEdit.IsEnabled = false;
2577.                    yCoordNodeEdit.IsEnabled = false;
2578.                    zCoordNodeEdit.IsEnabled = false;
2579.
2580.                    xDispEdit.IsChecked = false;
2581.                    yDispEdit.IsChecked = false;
2582.                    zDispEdit.IsChecked = false;
2583.                    xRotEdit.IsChecked = false;
2584.                    yRotEdit.IsChecked = false;
2585.                    zRotEdit.IsChecked = false;
2586.                }
2587.
2588.                public void OptionsHeaderSelected(object sender,
   RoutedEventArgs e)
2589.                {
2590.
   SendStringToInterface(string.Format("Mode&Options"));
2591.                    OptionsHeader.IsSelected = true;
2592.                }
2593.
2594.                public void AddMaterialHeaderSelected(object sender,
   RoutedEventArgs e)
2595.                {
2596.
   SendStringToInterface(string.Format("Mode&Material Creation"));
2597.                    PropertiesMainHeader.IsSelected = true;
2598.                    AddMaterialHeader.IsSelected = true;
2599.
2600.                    fckText.IsEnabled = false;
2601.                    labelWrongMatP.Visibility =
   Visibility.Collapsed;
2602.                }
2603.
2604.                public void AddMemberHeaderSelected(object sender,
   RoutedEventArgs e)
2605.                {
2606.                    SendStringToInterface(string.Format("Mode&Member
   Creation"));
2607.                    GeometryMainHeader.IsSelected = true;
2608.                    AddMemberHeader.IsSelected = true;
```

```
2609.
2610.                   materialMemberComboBox.Items.Clear();
2611.                   propertyMemberComboBox.Items.Clear();
2612.
2613.                   for (int i = 0; i < materials.Count; i++)
2614.                   {
2615.
   materialMemberComboBox.Items.Add(materials[i].name);
2616.                   }
2617.
2618.                   for (int i = 0; i < properties.Count; i++)
2619.                   {
2620.
   propertyMemberComboBox.Items.Add(properties[i].name);
2621.                   }
2622.
2623.                   memberMatSecLabel.Content = "- Material /
   Selection";
2624.                   memberMatSecLabel.Foreground = Brushes.Red;
2625.
2626.                   memberFirstNodeLabel.Content = "- First Node";
2627.                   memberFirstNodeLabel.Foreground = Brushes.Red;
2628.
2629.                   memberSecondNodeLabel.Content = "- Second Node";
2630.                   memberSecondNodeLabel.Foreground = Brushes.Red;
2631.               }
2632.
2633.               public void AddMemberMaterialComboBoxChanged(object
   sender, SelectionChangedEventArgs e)
2634.               {
2635.                   if (materialMemberComboBox.SelectedItem != null)
2636.                   {
2637.                       if (propertyMemberComboBox.SelectedItem !=
   null)
2638.                       {
2639.                           memberMatSecLabel.Content = "-
   Material/Selection OK";
2640.                           memberMatSecLabel.Foreground =
   Brushes.Black;
2641.                       }
2642.                   }
2643.               }
2644.
2645.               public void AddMemberPropertyComboBoxChanged(object
   sender, SelectionChangedEventArgs e)
2646.               {
2647.                   if (propertyMemberComboBox.SelectedItem != null)
2648.                   {
2649.                       if (materialMemberComboBox.SelectedItem !=
   null)
2650.                       {
2651.                           memberMatSecLabel.Content = "-
   Material/Selection OK";
2652.                           memberMatSecLabel.Foreground =
   Brushes.Black;
2653.                       }
2654.                   }
2655.               }
2656.
2657.               public void EditMemberHeaderSelected(object sender,
   RoutedEventArgs e)
```

```
2658.                {
2659.                    SendStringToInterface(string.Format("Mode&Member
   Editor"));
2660.                    GeometryMainHeader.IsSelected = true;
2661.                    EditMemberHeader.IsSelected = true;
2662.
2663.                    materialMemberEditComboBox.Items.Clear();
2664.                    propertyMemberEditComboBox.Items.Clear();
2665.
2666.                    for (int i = 0; i < materials.Count; i++)
2667.                    {
2668.
   materialMemberEditComboBox.Items.Add(materials[i].name);
2669.                    }
2670.
2671.                    for (int i = 0; i < properties.Count; i++)
2672.                    {
2673.
   propertyMemberEditComboBox.Items.Add(properties[i].name);
2674.                    }
2675.
2676.                    //SendStringToInterface(string.Format("Show
   Days&All"));
2677.                }
2678.
2679.                public void SplitMemberHeaderSelected(object sender,
   RoutedEventArgs e)
2680.                {
2681.                    SendStringToInterface(string.Format("Mode&Split
   Member"));
2682.                    GeometryMainHeader.IsSelected = true;
2683.                    SplitMemberHeader.IsSelected = true;
2684.                }
2685.
2686.                public void EditMember(object sender,
   RoutedEventArgs e)
2687.                {
2688.                    try
2689.                    {
2690.                        List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2691.
2692.                        if (tempList.Count > 1)
2693.                        {
2694.                            for (int i = 1; i < tempList.Count; i++)
2695.                            {
2696.
   members[int.Parse(tempList[i])].material =
   materialMemberEditComboBox.SelectedItem.ToString();
2697.
   members[int.Parse(tempList[i])].section =
   propertyMemberEditComboBox.SelectedItem.ToString();
2698.
   members[int.Parse(tempList[i])].startDay =
   int.Parse(dayMemberStartEditText.Text);
2699.
   members[int.Parse(tempList[i])].endDay =
   int.Parse(dayMemberEndEditText.Text);
2700.                            }
2701.                        }
2702.
```

```
2703.                         MassiveUpdate3DUI();
2704.                     }
2705.                     catch
2706.                     {
2707.
2708.                     }
2709.                 }
2710.
2711.                 public void DeleteMember(object sender,
   RoutedEventArgs e)
2712.                 {
2713.                     try
2714.                     {
2715.                         List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2716.
2717.                         if (tempList.Count > 1)
2718.                         {
2719.                             int count = 0;
2720.                             for (int i = 1; i < tempList.Count; i++)
2721.                             {
2722.
   members.RemoveAt(int.Parse(tempList[i]) - count);
2723.
2724.                                 for (int j = 0; j <
   memberForces.Count; j++)
2725.                                 {
2726.                                     if
   (memberForces[j].members.Contains(int.Parse(tempList[i]) - count))
2727.                                     {
2728.
   memberForces[j].members.Remove(int.Parse(tempList[i]) - count);
2729.
2730.                                         for (int k = 0; k <
   memberForces[j].members.Count; k++)
2731.                                         {
2732.                                             if
   (memberForces[j].members[k] > int.Parse(tempList[i]) - count)
2733.                                             {
2734.
   memberForces[j].members[k]--;
2735.                                             }
2736.                                         }
2737.                                     }
2738.                                     else
2739.                                     {
2740.                                         for (int k = 0; k <
   memberForces[j].members.Count; k++)
2741.                                         {
2742.                                             if
   (memberForces[j].members[k] > int.Parse(tempList[i]) - count)
2743.                                             {
2744.
   memberForces[j].members[k]--;
2745.                                             }
2746.                                         }
2747.                                     }
2748.                                 }
2749.
2750.                                 count++;
2751.                             }
```

```
2752.                         }
2753.
2754.                         for (int j = 0; j < memberForces.Count; j++)
2755.                         {
2756.                             if (memberForces[j].members.Count == 0)
   memberForces.RemoveAt(j);
2757.                         }
2758.
2759.                         for (int i = 0; i < members.Count; i++)
2760.                         {
2761.                             members[i].number = i;
2762.                         }
2763.
2764.                         MassiveUpdate3DUI();
2765.                     }
2766.                     catch
2767.                     {
2768.
2769.                     }
2770.                 }
2771.
2772.                 private void SplitMember(object sender,
   RoutedEventArgs e)
2773.                 {
2774.                     try
2775.                     {
2776.                         List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2777.
2778.                         if (tempList[0] == "Selected Split:" &&
   tempList.Count > 1)
2779.                         {
2780.                             int splitCount =
   int.Parse(splitPieces.Text);
2781.
2782.                             if (splitCount < 1)
2783.                             {
2784.                                 splitCount = 1;
2785.                                 splitPieces.Text = "1";
2786.                             }
2787.
2788.                             int originalMember =
   int.Parse(tempList[1]);
2789.
2790.                             double diffX =
   nodes[members[originalMember].n1].position.x -
   nodes[members[originalMember].n0].position.x;
2791.                             double diffY =
   nodes[members[originalMember].n1].position.y -
   nodes[members[originalMember].n0].position.y;
2792.                             double diffZ =
   nodes[members[originalMember].n1].position.z -
   nodes[members[originalMember].n0].position.z;
2793.
2794.                             diffX = diffX / splitCount;
2795.                             diffY = diffY / splitCount;
2796.                             diffZ = diffZ / splitCount;
2797.
2798.                             int originalEndNode =
   members[originalMember].n1;
2799.                             int lastNode = 0;
```

```
2800.
2801.                           for (int i = 1; i <= splitCount; i++)
2802.                           {
2803.                               if (i == 1)
2804.                               {
2805.                                   nodes.Add(new Nodes(nodes.Count,
2806.
   (nodes[members[originalMember].n0].position.x + i *
   diffX).ToString(),
2807.
   (nodes[members[originalMember].n0].position.y + i *
   diffY).ToString(),
2808.
   (nodes[members[originalMember].n0].position.z + i *
   diffZ).ToString(),
2809.                                       false, false, false, false,
   false, false));
2810.
2811.                                   lastNode = nodes.Count - 1;
2812.
2813.                                   members[originalMember].n1 =
   lastNode;
2814.                               }
2815.                               else if (i == splitCount)
2816.                               {
2817.                                   members.Add(new
   Member(members.Count, lastNode, originalEndNode,
   members[originalMember].material, members[originalMember].section,
2818.
   members[originalMember].startDay.ToString(),
   members[originalMember].endDay.ToString()));
2819.
2820.
   UpdateMemberForces(originalMember, members.Count - 1);
2821.                               }
2822.                               else
2823.                               {
2824.                                   nodes.Add(new Nodes(nodes.Count,
2825.
   (nodes[members[originalMember].n0].position.x + i *
   diffX).ToString(),
2826.
   (nodes[members[originalMember].n0].position.y + i *
   diffY).ToString(),
2827.
   (nodes[members[originalMember].n0].position.z + i *
   diffZ).ToString(),
2828.                                       false, false, false, false,
   false, false));
2829.
2830.                                   members.Add(new
   Member(members.Count, lastNode, nodes.Count - 1,
   members[originalMember].material, members[originalMember].section,
2831.
   members[originalMember].startDay.ToString(),
   members[originalMember].endDay.ToString()));
2832.
2833.                                   lastNode = nodes.Count - 1;
2834.
2835.
   UpdateMemberForces(originalMember, members.Count - 1);
```

```
2836.                                 }
2837.                             }
2838.                         }
2839.
2840.                         MassiveUpdate3DUI();
2841.                     }
2842.                     catch
2843.                     {
2844.
2845.                     }
2846.                 }
2847.
2848.                 public void NewFile(object sender, RoutedEventArgs
   e)
2849.                 {
2850.                     try
2851.                     {
2852.                         members.Clear();
2853.                         nodes.Clear();
2854.                         materials.Clear();
2855.                         properties.Clear();
2856.                         nodalForces.Clear();
2857.                         memberForces.Clear();
2858.
2859.                         PropertiesMainHeader.IsSelected = true;
2860.
2861.                         simulationSuccess = false;
2862.                         lastSimulatedSelected = "";
2863.
2864.
2865.                         MassiveUpdate3DUI();
2866.                     }
2867.                     catch
2868.                     {
2869.
2870.                     }
2871.                 }
2872.
2873.                 public void Exit(object sender, RoutedEventArgs e)
2874.                 {
2875.                     System.Windows.Application.Current.Shutdown();
2876.                 }
2877.
2878.
2879.                 public void IsTimeDependentChecked(object sender,
   RoutedEventArgs e)
2880.                 {
2881.                     fckText.IsEnabled = true;
2882.                 }
2883.
2884.                 public void IsTimeDependentUnchecked(object sender,
   RoutedEventArgs e)
2885.                 {
2886.                     fckText.IsEnabled = false;
2887.                     fckText.Text = "";
2888.                 }
2889.
2890.                 public void IsTimeDependentEditChecked(object
   sender, RoutedEventArgs e)
2891.                 {
2892.                     fckTextEdit.IsEnabled = true;
```

```
2893.                 }
2894.
2895.                 public void IsTimeDependentEditUnchecked(object
   sender, RoutedEventArgs e)
2896.                 {
2897.                     fckTextEdit.IsEnabled = false;
2898.                     fckTextEdit.Text = "";
2899.                 }
2900.
2901.                 public void EntireStructureChecked(object sender,
   RoutedEventArgs e)
2902.                 {
2903.                     if (!finishedLoading) return;
2904.                     structureDays.IsEnabled = false;
2905.                 }
2906.
2907.                 public void EntireStructureUnchecked(object sender,
   RoutedEventArgs e)
2908.                 {
2909.                     if (!finishedLoading) return;
2910.                     structureDays.IsEnabled = true;
2911.                 }
2912.
2913.                 public void UpdateStructureTime(object sender,
   RoutedEventArgs e)
2914.                 {
2915.                     if (structureDays.IsEnabled)
2916.                     {
2917.                         try
2918.                         {
2919.                             int day = int.Parse(structureDays.Text);
2920.
   SendStringToInterface(string.Format("Show Days&{0}", day));
2921.                         }
2922.                         catch
2923.                         {
2924.
2925.                         }
2926.                     }
2927.                     else
2928.                     {
2929.                         SendStringToInterface(string.Format("Show
   Days&All"));
2930.                     }
2931.                 }
2932.
2933.                 public void EditMaterialHeaderSelected(object
   sender, RoutedEventArgs e)
2934.                 {
2935.
   SendStringToInterface(string.Format("Mode&Material Editor"));
2936.                     EditMaterialHeader.IsSelected = true;
2937.                     PropertiesMainHeader.IsSelected = true;
2938.                     labelWrongMatPEdit.Visibility =
   Visibility.Collapsed;
2939.
2940.                     matComboBox.Items.Clear();
2941.
2942.                     for (int i = 0; i < materials.Count; i++)
2943.                     {
2944.                         matComboBox.Items.Add(materials[i].name);
```

```
2945.                     }
2946.                 }
2947.
2948.                 public void
   EditMaterialComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
2949.                 {
2950.                     if (matComboBox.SelectedItem != null)
2951.                     {
2952.                         matNameEdit.Text = materials.Where(item =>
   item.name == matComboBox.SelectedItem.ToString()).First().name;
2953.                         matEEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().E.ToString();
2954.                         matGEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().G.ToString();
2955.                         matVEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().v.ToString();
2956.                         fckTextEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().fck.ToString();
2957.
2958.                         isTimeDependentEdit.IsChecked =
   materials.Where(item => item.name ==
   matComboBox.SelectedItem.ToString()).First().timeDependent;
2959.
2960.
2961.                         matNameEdit.IsEnabled = true;
2962.                         matEEdit.IsEnabled = true;
2963.                         matGEdit.IsEnabled = true;
2964.                         matVEdit.IsEnabled = true;
2965.                         isTimeDependentEdit.IsEnabled = true;
2966.                         fckTextEdit.IsEnabled =
   isTimeDependentEdit.IsChecked.GetValueOrDefault();
2967.
2968.                         saveEditMaterial.IsEnabled = true;
2969.                         deleteEditMaterial.IsEnabled = true;
2970.                         calcGEdit.IsEnabled = true;
2971.                     }
2972.                     else
2973.                     {
2974.                         matNameEdit.Text = "";
2975.                         matEEdit.Text = "";
2976.                         matGEdit.Text = "";
2977.                         matVEdit.Text = "";
2978.                         fckTextEdit.Text = "";
2979.
2980.                         matNameEdit.IsEnabled = false;
2981.                         matEEdit.IsEnabled = false;
2982.                         matGEdit.IsEnabled = false;
2983.                         matVEdit.IsEnabled = false;
2984.                         isTimeDependentEdit.IsChecked = false;
2985.                         isTimeDependentEdit.IsEnabled = false;
2986.
2987.                         saveEditMaterial.IsEnabled = false;
2988.                         deleteEditMaterial.IsEnabled = false;
2989.                         calcGEdit.IsEnabled = false;
2990.                     }
2991.                 }
```

```
2992.
2993.                public void SaveEditMaterial(object sender,
   RoutedEventArgs e)
2994.                {
2995.                    try
2996.                    {
2997.                        string matName =
   matComboBox.SelectedItem.ToString();
2998.
2999.                        if (matName != matNameEdit.Text &&
   materials.Where(item => item.name == matNameEdit.Text).ToList().Count
   != 0)
3000.                        {
3001.                            labelWrongMatPEdit.Content = "Wrong
   Properties!";
3002.                            labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3003.                        }
3004.                        else
3005.                        {
3006.                            materials.Where(item => item.name ==
   matName).First().E = double.Parse(matEEdit.Text);
3007.                            materials.Where(item => item.name ==
   matName).First().G = double.Parse(matGEdit.Text);
3008.                            materials.Where(item => item.name ==
   matName).First().v = double.Parse(matVEdit.Text);
3009.                            materials.Where(item => item.name ==
   matName).First().timeDependent =
   isTimeDependentEdit.IsChecked.GetValueOrDefault();
3010.                            if (fckTextEdit.Text == "")
   fckTextEdit.Text = "0";
3011.                            materials.Where(item => item.name ==
   matName).First().fck = double.Parse(fckTextEdit.Text);
3012.                            materials.Where(item => item.name ==
   matName).First().name = matNameEdit.Text;
3013.
3014.                            for (int i = 0; i < members.Count; i++)
3015.                            {
3016.                                if (members[i].material ==
   matComboBox.SelectedItem.ToString()) members[i].material =
   matNameEdit.Text;
3017.                            }
3018.
3019.                            matComboBox.Items.Clear();
3020.
3021.                            for (int i = 0; i < materials.Count;
   i++)
3022.                            {
3023.
   matComboBox.Items.Add(materials[i].name);
3024.                            }
3025.
3026.                            labelWrongMatPEdit.Content = "Material
   Saved!";
3027.                            labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3028.                        }
3029.                    }
3030.                    catch
3031.                    {
3032.
```

```
3033.                     }
3034.                 }
3035.
3036.                 public void DeleteEditMaterial(object sender,
   RoutedEventArgs e)
3037.                 {
3038.                     try
3039.                     {
3040.                         materials.Remove(materials.Where(item =>
   item.name == matComboBox.SelectedItem.ToString()).First());
3041.
3042.                         matComboBox.Items.Clear();
3043.
3044.                         for (int i = 0; i < materials.Count; i++)
3045.                         {
3046.
   matComboBox.Items.Add(materials[i].name);
3047.                         }
3048.
3049.                         labelWrongMatPEdit.Content = "Material
   Removed!";
3050.                         labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3051.                     }
3052.                     catch
3053.                     {
3054.
3055.                     }
3056.                 }
3057.
3058.                 public void AddPropertyHeaderSelected(object sender,
   RoutedEventArgs e)
3059.                 {
3060.
   SendStringToInterface(string.Format("Mode&Property Creation"));
3061.                     AddPropertiesHeader.IsSelected = true;
3062.                     PropertiesMainHeader.IsSelected = true;
3063.                     labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3064.                     labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3065.                     labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3066.                 }
3067.
3068.                 public void
   EditPropertyComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
3069.                 {
3070.                     if (!finishedLoading) return;
3071.
3072.                     if (propSecComboBox.SelectedIndex == 0)
3073.                     {
3074.                         propNoneSep.Visibility = Visibility.Visible;
3075.                         propNoneSave.Visibility =
   Visibility.Visible;
3076.                         labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3077.
3078.                         propDiamLabel.Visibility =
   Visibility.Collapsed;
```

```
3079.                       propDiam.Visibility = Visibility.Collapsed;
3080.                       propDiamCalc.Visibility =
   Visibility.Collapsed;
3081.                       propDiamSep.Visibility =
   Visibility.Collapsed;
3082.                       propDiamSave.Visibility =
   Visibility.Collapsed;
3083.                       labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3084.
3085.                       propHLabel.Visibility =
   Visibility.Collapsed;
3086.                       propH.Visibility = Visibility.Collapsed;
3087.                       propBLabel.Visibility =
   Visibility.Collapsed;
3088.                       propB.Visibility = Visibility.Collapsed;
3089.                       propRetCalc.Visibility =
   Visibility.Collapsed;
3090.                       propRetSep.Visibility =
   Visibility.Collapsed;
3091.                       propRetSave.Visibility =
   Visibility.Collapsed;
3092.                       labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3093.                   }
3094.
3095.                   if (propSecComboBox.SelectedIndex == 1)
3096.                   {
3097.                       propNoneSep.Visibility =
   Visibility.Collapsed;
3098.                       propNoneSave.Visibility =
   Visibility.Collapsed;
3099.                       labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3100.
3101.                       propDiamLabel.Visibility =
   Visibility.Visible;
3102.                       propDiam.Visibility = Visibility.Visible;
3103.                       propDiamCalc.Visibility =
   Visibility.Visible;
3104.                       propDiamSep.Visibility = Visibility.Visible;
3105.                       propDiamSave.Visibility =
   Visibility.Visible;
3106.                       labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3107.
3108.                       propHLabel.Visibility =
   Visibility.Collapsed;
3109.                       propH.Visibility = Visibility.Collapsed;
3110.                       propBLabel.Visibility =
   Visibility.Collapsed;
3111.                       propB.Visibility = Visibility.Collapsed;
3112.                       propRetCalc.Visibility =
   Visibility.Collapsed;
3113.                       propRetSep.Visibility =
   Visibility.Collapsed;
3114.                       propRetSave.Visibility =
   Visibility.Collapsed;
3115.                       labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3116.                   }
```

```
3117.
3118.                   if (propSecComboBox.SelectedIndex == 2)
3119.                   {
3120.                       propNoneSep.Visibility =
   Visibility.Collapsed;
3121.                       propNoneSave.Visibility =
   Visibility.Collapsed;
3122.                       labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3123.
3124.                       propDiamLabel.Visibility =
   Visibility.Collapsed;
3125.                       propDiam.Visibility = Visibility.Collapsed;
3126.                       propDiamCalc.Visibility =
   Visibility.Collapsed;
3127.                       propDiamSep.Visibility =
   Visibility.Collapsed;
3128.                       propDiamSave.Visibility =
   Visibility.Collapsed;
3129.                       labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3130.
3131.                       propHLabel.Visibility = Visibility.Visible;
3132.                       propH.Visibility = Visibility.Visible;
3133.                       propBLabel.Visibility = Visibility.Visible;
3134.                       propB.Visibility = Visibility.Visible;
3135.                       propRetCalc.Visibility = Visibility.Visible;
3136.                       propRetSep.Visibility = Visibility.Visible;
3137.                       propRetSave.Visibility = Visibility.Visible;
3138.                       labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3139.                   }
3140.               }
3141.
3142.               public void CalculateCircularProperty(object sender,
   RoutedEventArgs e)
3143.               {
3144.                   try
3145.                   {
3146.                       double A = Math.PI *
   Math.Pow(double.Parse(propDiam.Text) / 2, 2);
3147.                       double Ix = (Math.PI / 4) *
   Math.Pow(double.Parse(propDiam.Text) / 2, 4);
3148.                       double Iy = Ix;
3149.                       double Iz = (Math.PI / 2) *
   Math.Pow(double.Parse(propDiam.Text) / 2, 4);
3150.
3151.                       propA.Text = A.ToString();
3152.                       propIx.Text = Ix.ToString();
3153.                       propIy.Text = Iy.ToString();
3154.                       propIz.Text = Iz.ToString();
3155.                   }
3156.                   catch
3157.                   {
3158.
3159.                   }
3160.               }
3161.
3162.               public void CalculateRetangularProperty(object
   sender, RoutedEventArgs e)
3163.               {
```

```
3164.                     try
3165.                     {
3166.                         double A = double.Parse(propH.Text) *
   double.Parse(propB.Text);
3167.                         double Ix = double.Parse(propB.Text) *
   Math.Pow(double.Parse(propH.Text), 3) / 12;
3168.                         double Iy = double.Parse(propH.Text) *
   Math.Pow(double.Parse(propB.Text), 3) / 12;
3169.                         double Iz = Ix + Iy;
3170.
3171.                         propA.Text = A.ToString();
3172.                         propIx.Text = Ix.ToString();
3173.                         propIy.Text = Iy.ToString();
3174.                         propIz.Text = Iz.ToString();
3175.                     }
3176.                     catch
3177.                     {
3178.
3179.                     }
3180.                 }
3181.
3182.                 public void CreateProperty(object sender,
   RoutedEventArgs e)
3183.                 {
3184.                     if (propName.Text != "" && propA.Text != "" &&
   propIx.Text != "" && propIy.Text != "" && propIz.Text != "")
3185.                     {
3186.                         try
3187.                         {
3188.                             if (properties.Where(item => item.name
   == propName.Text).ToList().Count == 0)
3189.                             {
3190.                                 properties.Add(new
   Property(propName.Text, propA.Text, propIx.Text, propIy.Text,
   propIz.Text, propSecComboBox.SelectedIndex.ToString(), propDiam.Text,
   propH.Text, propB.Text));
3191.
3192.                                 propName.Text = "";
3193.                                 propA.Text = "";
3194.                                 propIx.Text = "";
3195.                                 propIy.Text = "";
3196.                                 propIz.Text = "";
3197.                                 propSecComboBox.SelectedIndex = 0;
3198.                                 propDiam.Text = "";
3199.                                 propH.Text = "";
3200.                                 propB.Text = "";
3201.
3202.                                 labelWrongPropNone.Content =
   "Property Saved!";
3203.                                 labelWrongPropNone.Visibility =
   Visibility.Visible;
3204.
3205.                             }
3206.                             else
3207.                             {
3208.                                 if (propSecComboBox.SelectedIndex ==
   0)
3209.                                 {
3210.                                     labelWrongPropNone.Content =
   "Wrong Properties!";
```

```
3211.                                   labelWrongPropNone.Visibility =
   Visibility.Visible;
3212.                               }
3213.                               if (propSecComboBox.SelectedIndex ==
   1)
3214.                               {
3215.                                   labelWrongPropDiam.Content =
   "Wrong Properties!";
3216.                                   labelWrongPropDiam.Visibility =
   Visibility.Visible;
3217.                               }
3218.                               if (propSecComboBox.SelectedIndex ==
   2)
3219.                               {
3220.                                   labelWrongPropRet.Content =
   "Wrong Properties!";
3221.                                   labelWrongPropRet.Visibility =
   Visibility.Visible;
3222.                               }
3223.                           }
3224.                       }
3225.                       catch
3226.                       {
3227.                           if (propSecComboBox.SelectedIndex == 0)
3228.                           {
3229.                               labelWrongPropNone.Content = "Wrong
   Properties!";
3230.                               labelWrongPropNone.Visibility =
   Visibility.Visible;
3231.                           }
3232.                           if (propSecComboBox.SelectedIndex == 1)
3233.                           {
3234.                               labelWrongPropDiam.Content = "Wrong
   Properties!";
3235.                               labelWrongPropDiam.Visibility =
   Visibility.Visible;
3236.                           }
3237.                           if (propSecComboBox.SelectedIndex == 2)
3238.                           {
3239.                               labelWrongPropRet.Content = "Wrong
   Properties!";
3240.                               labelWrongPropRet.Visibility =
   Visibility.Visible;
3241.                           }
3242.                       }
3243.                   }
3244.                   else
3245.                   {
3246.                       if (propSecComboBox.SelectedIndex == 0)
3247.                       {
3248.                           labelWrongPropNone.Content = "Wrong
   Properties!";
3249.                           labelWrongPropNone.Visibility =
   Visibility.Visible;
3250.                       }
3251.                       if (propSecComboBox.SelectedIndex == 1)
3252.                       {
3253.                           labelWrongPropDiam.Content = "Wrong
   Properties!";
3254.                           labelWrongPropDiam.Visibility =
   Visibility.Visible;
```

```
3255.                         }
3256.                         if (propSecComboBox.SelectedIndex == 2)
3257.                         {
3258.                             labelWrongPropRet.Content = "Wrong
   Properties!";
3259.                             labelWrongPropRet.Visibility =
   Visibility.Visible;
3260.                         }
3261.                     }
3262.                 }
3263.
3264.                 public void EditPropertyHeaderSelected(object
   sender, RoutedEventArgs e)
3265.                 {
3266.
   SendStringToInterface(string.Format("Mode&Property Editor"));
3267.                     EditPropertiesHeader.IsSelected = true;
3268.                     PropertiesMainHeader.IsSelected = true;
3269.                     labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3270.                     labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3271.                     labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3272.
3273.                     editComboBox.Items.Clear();
3274.
3275.                     for (int i = 0; i < properties.Count; i++)
3276.                     {
3277.                         editComboBox.Items.Add(properties[i].name);
3278.                     }
3279.
3280.                     propNameEdit.Text = "";
3281.                     propAEdit.Text = "";
3282.                     propIxEdit.Text = "";
3283.                     propIyEdit.Text = "";
3284.                     propIzEdit.Text = "";
3285.
3286.                     propNameEdit.IsEnabled = false;
3287.
3288.                     propAEdit.IsEnabled = false;
3289.                     propIxEdit.IsEnabled = false;
3290.                     propIyEdit.IsEnabled = false;
3291.                     propIzEdit.IsEnabled = false;
3292.
3293.                     editComboBox.IsEnabled = true;
3294.
3295.                     propSecComboBoxEdit.IsEnabled = false;
3296.                     propSecComboBoxEdit.SelectedIndex = 0;
3297.
3298.                     propNoneSaveEdit.IsEnabled = false;
3299.                     propNoneDeleteEdit.IsEnabled = false;
3300.
3301.                     propNoneSaveEdit.Visibility =
   Visibility.Visible;
3302.                     propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3303.                     propNoneSepEdit.Visibility = Visibility.Visible;
3304.                     labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3305.
```

```
3306.                   propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3307.                   propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3308.                   propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3309.                   propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3310.                   propDiamEdit.Visibility = Visibility.Collapsed;
3311.                   labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3312.
3313.                   propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3314.                   propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3315.                   propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3316.                   propRetSep.Visibility = Visibility.Collapsed;
3317.                   propHEdit.Visibility = Visibility.Collapsed;
3318.                   propBEdit.Visibility = Visibility.Collapsed;
3319.                   labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3320.               }
3321.
3322.               public void
   EditPropertyListComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
3323.               {
3324.                   if (editComboBox.SelectedItem != null)
3325.                   {
3326.                       propNameEdit.Text = properties.Where(item =>
   item.name == editComboBox.SelectedItem.ToString()).First().name;
3327.                       propAEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().A.ToString();
3328.                       propIxEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Ix.ToString();
3329.                       propIyEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Iy.ToString();
3330.                       propIzEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Iz.ToString();
3331.
3332.                       if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "None")
3333.                       {
3334.                           propSecComboBoxEdit.IsEnabled = true;
3335.                           propSecComboBoxEdit.SelectedIndex = 0;
3336.
3337.                           propNoneSaveEdit.IsEnabled = true;
3338.                           propNoneDeleteEdit.IsEnabled = true;
3339.
3340.                           propNoneSaveEdit.Visibility =
   Visibility.Visible;
3341.                           propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3342.                           propNoneSepEdit.Visibility =
   Visibility.Visible;
```

```
3343.                           labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3344.
3345.                           propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3346.                           propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3347.                           propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3348.                           propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3349.                           propDiamEdit.Visibility =
   Visibility.Collapsed;
3350.                           propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3351.                           labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3352.
3353.                           propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3354.                           propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3355.                           propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3356.                           propRetSepEdit.Visibility =
   Visibility.Collapsed;
3357.                           propHEdit.Visibility =
   Visibility.Collapsed;
3358.                           propBEdit.Visibility =
   Visibility.Collapsed;
3359.                           propHLabelEdit.Visibility =
   Visibility.Collapsed;
3360.                           propBLabelEdit.Visibility =
   Visibility.Collapsed;
3361.                           labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3362.
3363.                       }
3364.                       if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "Circular")
3365.                       {
3366.                           propSecComboBoxEdit.IsEnabled = true;
3367.                           propSecComboBoxEdit.SelectedIndex = 1;
3368.
3369.                           propNoneSaveEdit.IsEnabled = false;
3370.                           propNoneDeleteEdit.IsEnabled = false;
3371.
3372.                           propDiamCalcEdit.IsEnabled = true;
3373.                           propDiamSaveEdit.IsEnabled = true;
3374.                           propDiamDeleteEdit.IsEnabled = true;
3375.                           propDiamEdit.IsEnabled = true;
3376.                           propDiamSepEdit.IsEnabled = true;
3377.
3378.                           propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3379.                           propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3380.                           propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3381.                           labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
```

```
3382.
3383.                           propDiamCalcEdit.Visibility =
   Visibility.Visible;
3384.                           propDiamSaveEdit.Visibility =
   Visibility.Visible;
3385.                           propDiamDeleteEdit.Visibility =
   Visibility.Visible;
3386.                           propDiamSepEdit.Visibility =
   Visibility.Visible;
3387.                           propDiamEdit.Visibility =
   Visibility.Visible;
3388.                           propDiamLabelEdit.Visibility =
   Visibility.Visible;
3389.                           labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3390.
3391.                           propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3392.                           propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3393.                           propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3394.                           propRetSepEdit.Visibility =
   Visibility.Collapsed;
3395.                           propHEdit.Visibility =
   Visibility.Collapsed;
3396.                           propBEdit.Visibility =
   Visibility.Collapsed;
3397.                           propHLabelEdit.Visibility =
   Visibility.Collapsed;
3398.                           propBLabelEdit.Visibility =
   Visibility.Collapsed;
3399.                           labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3400.                       }
3401.                       if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "Retangular")
3402.                       {
3403.                           propSecComboBoxEdit.IsEnabled = true;
3404.                           propSecComboBoxEdit.SelectedIndex = 2;
3405.
3406.                           propNoneSaveEdit.IsEnabled = false;
3407.                           propNoneDeleteEdit.IsEnabled = false;
3408.
3409.                           propRetCalcEdit.IsEnabled = true;
3410.                           propRetSaveEdit.IsEnabled = true;
3411.                           propRetDeleteEdit.IsEnabled = true;
3412.                           propHEdit.IsEnabled = true;
3413.                           propBEdit.IsEnabled = true;
3414.                           propRetSepEdit.IsEnabled = true;
3415.
3416.                           propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3417.                           propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3418.                           propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3419.                           labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3420.
```

```
3421.                           propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3422.                           propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3423.                           propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3424.                           propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3425.                           propDiamEdit.Visibility =
   Visibility.Collapsed;
3426.                           propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3427.                           labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3428.
3429.                           propRetCalcEdit.Visibility =
   Visibility.Visible;
3430.                           propRetSaveEdit.Visibility =
   Visibility.Visible;
3431.                           propRetDeleteEdit.Visibility =
   Visibility.Visible;
3432.                           propRetSepEdit.Visibility =
   Visibility.Visible;
3433.                           propHEdit.Visibility =
   Visibility.Visible;
3434.                           propBEdit.Visibility =
   Visibility.Visible;
3435.                           propHLabelEdit.Visibility =
   Visibility.Visible;
3436.                           propBLabelEdit.Visibility =
   Visibility.Visible;
3437.                           labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3438.                       }
3439.
3440.                       propDiamEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().d.ToString();
3441.                       propHEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().h.ToString();
3442.                       propBEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().b.ToString();
3443.
3444.                       propNameEdit.IsEnabled = true;
3445.
3446.                       propAEdit.IsEnabled = true;
3447.                       propIxEdit.IsEnabled = true;
3448.                       propIyEdit.IsEnabled = true;
3449.                       propIzEdit.IsEnabled = true;
3450.
3451.                       editComboBox.IsEnabled = true;
3452.                   }
3453.                   else
3454.                   {
3455.
3456.                   }
3457.               }
3458.
```

```
3459.               public void
    PropSecComboBoxEditSelectionChanged(object sender,
    SelectionChangedEventArgs e)
3460.               {
3461.                   if (!finishedLoading) return;
3462.
3463.                   if (propSecComboBoxEdit.SelectedIndex == 0)
3464.                   {
3465.                       propNoneSaveEdit.IsEnabled = true;
3466.                       propNoneDeleteEdit.IsEnabled = true;
3467.
3468.                       propNoneSaveEdit.Visibility =
    Visibility.Visible;
3469.                       propNoneDeleteEdit.Visibility =
    Visibility.Visible;
3470.                       propNoneSepEdit.Visibility =
    Visibility.Visible;
3471.
3472.                       propDiamCalcEdit.Visibility =
    Visibility.Collapsed;
3473.                       propDiamSaveEdit.Visibility =
    Visibility.Collapsed;
3474.                       propDiamDeleteEdit.Visibility =
    Visibility.Collapsed;
3475.                       propDiamSepEdit.Visibility =
    Visibility.Collapsed;
3476.                       propDiamEdit.Visibility =
    Visibility.Collapsed;
3477.                       propDiamLabelEdit.Visibility =
    Visibility.Collapsed;
3478.
3479.                       propRetCalcEdit.Visibility =
    Visibility.Collapsed;
3480.                       propRetSaveEdit.Visibility =
    Visibility.Collapsed;
3481.                       propRetDeleteEdit.Visibility =
    Visibility.Collapsed;
3482.                       propRetSepEdit.Visibility =
    Visibility.Collapsed;
3483.                       propHEdit.Visibility = Visibility.Collapsed;
3484.                       propBEdit.Visibility = Visibility.Collapsed;
3485.                       propHLabelEdit.Visibility =
    Visibility.Collapsed;
3486.                       propBLabelEdit.Visibility =
    Visibility.Collapsed;
3487.                   }
3488.
3489.                   if (propSecComboBoxEdit.SelectedIndex == 1)
3490.                   {
3491.                       propNoneSaveEdit.IsEnabled = false;
3492.                       propNoneDeleteEdit.IsEnabled = false;
3493.
3494.                       propDiamCalcEdit.IsEnabled = true;
3495.                       propDiamSaveEdit.IsEnabled = true;
3496.                       propDiamDeleteEdit.IsEnabled = true;
3497.                       propDiamEdit.IsEnabled = true;
3498.                       propDiamSepEdit.IsEnabled = true;
3499.
3500.                       propNoneSaveEdit.Visibility =
    Visibility.Collapsed;
```

```
3501.                       propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3502.                       propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3503.
3504.                       propDiamCalcEdit.Visibility =
   Visibility.Visible;
3505.                       propDiamSaveEdit.Visibility =
   Visibility.Visible;
3506.                       propDiamDeleteEdit.Visibility =
   Visibility.Visible;
3507.                       propDiamSepEdit.Visibility =
   Visibility.Visible;
3508.                       propDiamEdit.Visibility =
   Visibility.Visible;
3509.                       propDiamLabelEdit.Visibility =
   Visibility.Visible;
3510.
3511.                       propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3512.                       propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3513.                       propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3514.                       propRetSepEdit.Visibility =
   Visibility.Collapsed;
3515.                       propHEdit.Visibility = Visibility.Collapsed;
3516.                       propBEdit.Visibility = Visibility.Collapsed;
3517.                       propHLabelEdit.Visibility =
   Visibility.Collapsed;
3518.                       propBLabelEdit.Visibility =
   Visibility.Collapsed;
3519.                   }
3520.
3521.                   if (propSecComboBoxEdit.SelectedIndex == 2)
3522.                   {
3523.                       propNoneSaveEdit.IsEnabled = false;
3524.                       propNoneDeleteEdit.IsEnabled = false;
3525.
3526.                       propRetCalcEdit.IsEnabled = true;
3527.                       propRetSaveEdit.IsEnabled = true;
3528.                       propRetDeleteEdit.IsEnabled = true;
3529.                       propHEdit.IsEnabled = true;
3530.                       propBEdit.IsEnabled = true;
3531.                       propRetSepEdit.IsEnabled = true;
3532.
3533.                       propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3534.                       propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3535.                       propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3536.
3537.                       propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3538.                       propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3539.                       propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3540.                       propDiamSepEdit.Visibility =
   Visibility.Collapsed;
```

```
3541.                         propDiamEdit.Visibility =
   Visibility.Collapsed;
3542.                         propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3543.
3544.                         propRetCalcEdit.Visibility =
   Visibility.Visible;
3545.                         propRetSaveEdit.Visibility =
   Visibility.Visible;
3546.                         propRetDeleteEdit.Visibility =
   Visibility.Visible;
3547.                         propRetSepEdit.Visibility =
   Visibility.Visible;
3548.                         propHEdit.Visibility = Visibility.Visible;
3549.                         propBEdit.Visibility = Visibility.Visible;
3550.                         propHLabelEdit.Visibility =
   Visibility.Visible;
3551.                         propBLabelEdit.Visibility =
   Visibility.Visible;
3552.                     }
3553.                 }
3554.
3555.                 public void DeleteEditProperty(object sender,
   RoutedEventArgs e)
3556.                 {
3557.                     try
3558.                     {
3559.                         properties.Remove(properties.Where(item =>
   item.name == editComboBox.SelectedItem.ToString()).First());
3560.
3561.                         editComboBox.Items.Clear();
3562.
3563.                         for (int i = 0; i < properties.Count; i++)
3564.                         {
3565.
   editComboBox.Items.Add(properties[i].name);
3566.                         }
3567.
3568.                         propNameEdit.Text = "";
3569.                         propAEdit.Text = "";
3570.                         propIxEdit.Text = "";
3571.                         propIyEdit.Text = "";
3572.                         propIzEdit.Text = "";
3573.
3574.                         propNameEdit.IsEnabled = false;
3575.
3576.                         propAEdit.IsEnabled = false;
3577.                         propIxEdit.IsEnabled = false;
3578.                         propIyEdit.IsEnabled = false;
3579.                         propIzEdit.IsEnabled = false;
3580.
3581.                         editComboBox.IsEnabled = true;
3582.
3583.                         propSecComboBoxEdit.IsEnabled = false;
3584.                         propSecComboBoxEdit.SelectedIndex = 0;
3585.
3586.                         propNoneSaveEdit.IsEnabled = false;
3587.                         propNoneDeleteEdit.IsEnabled = false;
3588.
3589.                         propNoneSaveEdit.Visibility =
   Visibility.Visible;
```

```
3590.                         propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3591.                         propNoneSepEdit.Visibility =
   Visibility.Visible;
3592.
3593.                         propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3594.                         propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3595.                         propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3596.                         propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3597.                         propDiamEdit.Visibility =
   Visibility.Collapsed;
3598.
3599.                         propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3600.                         propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3601.                         propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3602.                         propRetSep.Visibility =
   Visibility.Collapsed;
3603.                         propHEdit.Visibility = Visibility.Collapsed;
3604.                         propBEdit.Visibility = Visibility.Collapsed;
3605.                     }
3606.                     catch
3607.                     {
3608.
3609.                     }
3610.                 }
3611.
3612.                 public void CalculateCircularEditProperty(object
   sender, RoutedEventArgs e)
3613.                 {
3614.                     try
3615.                     {
3616.                         double A = Math.PI *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 2);
3617.                         double Ix = (Math.PI / 4) *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 4);
3618.                         double Iy = Ix;
3619.                         double Iz = (Math.PI / 2) *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 4);
3620.
3621.                         propAEdit.Text = A.ToString();
3622.                         propIxEdit.Text = Ix.ToString();
3623.                         propIyEdit.Text = Iy.ToString();
3624.                         propIzEdit.Text = Iz.ToString();
3625.                     }
3626.                     catch
3627.                     {
3628.                         if (propSecComboBoxEdit.SelectedIndex == 0)
3629.                         {
3630.                             labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3631.                             labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3632.                         }
3633.                         if (propSecComboBoxEdit.SelectedIndex == 1)
```

```
3634.                         {
3635.                             labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3636.                             labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3637.                         }
3638.                         if (propSecComboBoxEdit.SelectedIndex == 2)
3639.                         {
3640.                             labelWrongPropRetEdit.Content =
   "Property Wrong!";
3641.                             labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3642.                         }
3643.                     }
3644.                 }
3645.
3646.                 public void CalculateRetangularEditProperty(object
   sender, RoutedEventArgs e)
3647.                 {
3648.                     try
3649.                     {
3650.                         double A = double.Parse(propHEdit.Text) *
   double.Parse(propBEdit.Text);
3651.                         double Ix = double.Parse(propBEdit.Text) *
   Math.Pow(double.Parse(propHEdit.Text), 3) / 12;
3652.                         double Iy = double.Parse(propHEdit.Text) *
   Math.Pow(double.Parse(propBEdit.Text), 3) / 12;
3653.                         double Iz = Ix + Iy;
3654.
3655.                         propAEdit.Text = A.ToString();
3656.                         propIxEdit.Text = Ix.ToString();
3657.                         propIyEdit.Text = Iy.ToString();
3658.                         propIzEdit.Text = Iz.ToString();
3659.                     }
3660.                     catch
3661.                     {
3662.                         if (propSecComboBoxEdit.SelectedIndex == 0)
3663.                         {
3664.                             labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3665.                             labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3666.                         }
3667.                         if (propSecComboBoxEdit.SelectedIndex == 1)
3668.                         {
3669.                             labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3670.                             labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3671.                         }
3672.                         if (propSecComboBoxEdit.SelectedIndex == 2)
3673.                         {
3674.                             labelWrongPropRetEdit.Content =
   "Property Wrong!";
3675.                             labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3676.                         }
3677.                     }
3678.                 }
3679.
```

```
3680.                public void SaveEditProperty(object sender,
   RoutedEventArgs e)
3681.                {
3682.                    if (propNameEdit.Text != "" && propAEdit.Text !=
   "" && propIxEdit.Text != "" && propIyEdit.Text != "" &&
   propIzEdit.Text != "")
3683.                    {
3684.                        try
3685.                        {
3686.
3687.                            string propName =
   editComboBox.SelectedItem.ToString();
3688.
3689.                            if (propName != propNameEdit.Text &&
   properties.Where(item => item.name ==
   propNameEdit.Text).ToList().Count != 0)
3690.                            {
3691.                                //labelWrongMatPEdit.Content =
   "Wrong Properties!";
3692.                                //labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3693.                            }
3694.                            else
3695.                            {
3696.                                properties.Where(item => item.name
   == propName).First().A = double.Parse(propAEdit.Text);
3697.                                properties.Where(item => item.name
   == propName).First().Ix = double.Parse(propIxEdit.Text);
3698.                                properties.Where(item => item.name
   == propName).First().Iy = double.Parse(propIyEdit.Text);
3699.                                properties.Where(item => item.name
   == propName).First().Iz = double.Parse(propIzEdit.Text);
3700.                                properties.Where(item => item.name
   == propName).First().d = double.Parse(propDiamEdit.Text);
3701.                                properties.Where(item => item.name
   == propName).First().h = double.Parse(propHEdit.Text);
3702.                                properties.Where(item => item.name
   == propName).First().b = double.Parse(propBEdit.Text);
3703.                                if
   (propSecComboBoxEdit.SelectedIndex == 0) properties.Where(item =>
   item.name == propName).First().type = "None";
3704.                                if
   (propSecComboBoxEdit.SelectedIndex == 1) properties.Where(item =>
   item.name == propName).First().type = "Circular";
3705.                                if
   (propSecComboBoxEdit.SelectedIndex == 2) properties.Where(item =>
   item.name == propName).First().type = "Retangular";
3706.                                properties.Where(item => item.name
   == propName).First().name = propNameEdit.Text;
3707.
3708.                                for (int i = 0; i < members.Count;
   i++)
3709.                                {
3710.                                    if (members[i].section ==
   editComboBox.SelectedItem.ToString()) members[i].section =
   propNameEdit.Text;
3711.                                }
3712.
3713.                                editComboBox.Items.Clear();
3714.
```

```
3715.                                   for (int i = 0; i <
   properties.Count; i++)
3716.                                   {
3717.
   editComboBox.Items.Add(properties[i].name);
3718.                                   }
3719.
3720.                                   propNameEdit.Text = "";
3721.                                   propAEdit.Text = "";
3722.                                   propIxEdit.Text = "";
3723.                                   propIyEdit.Text = "";
3724.                                   propIzEdit.Text = "";
3725.
3726.                                   propNameEdit.IsEnabled = false;
3727.
3728.                                   propAEdit.IsEnabled = false;
3729.                                   propIxEdit.IsEnabled = false;
3730.                                   propIyEdit.IsEnabled = false;
3731.                                   propIzEdit.IsEnabled = false;
3732.
3733.                                   editComboBox.IsEnabled = true;
3734.
3735.                                   propSecComboBoxEdit.IsEnabled =
   false;
3736.                                   propSecComboBoxEdit.SelectedIndex =
   0;
3737.
3738.                                   propNoneSaveEdit.IsEnabled = false;
3739.                                   propNoneDeleteEdit.IsEnabled =
   false;
3740.
3741.                                   propNoneSaveEdit.Visibility =
   Visibility.Visible;
3742.                                   propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3743.                                   propNoneSepEdit.Visibility =
   Visibility.Visible;
3744.
3745.                                   propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3746.                                   propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3747.                                   propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3748.                                   propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3749.                                   propDiamEdit.Visibility =
   Visibility.Collapsed;
3750.
3751.                                   propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3752.                                   propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3753.                                   propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3754.                                   propRetSep.Visibility =
   Visibility.Collapsed;
3755.                                   propHEdit.Visibility =
   Visibility.Collapsed;
3756.                                   propBEdit.Visibility =
   Visibility.Collapsed;
```

```
3757.
3758.                                   labelWrongPropNoneEdit.Content =
   "Property Saved!";
3759.                                   labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3760.                               }
3761.                           }
3762.                           catch
3763.                           {
3764.                               if (propSecComboBoxEdit.SelectedIndex ==
   0)
3765.                               {
3766.                                   labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3767.                                   labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3768.                               }
3769.                               if (propSecComboBoxEdit.SelectedIndex ==
   1)
3770.                               {
3771.                                   labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3772.                                   labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3773.                               }
3774.                               if (propSecComboBoxEdit.SelectedIndex ==
   2)
3775.                               {
3776.                                   labelWrongPropRetEdit.Content =
   "Property Wrong!";
3777.                                   labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3778.                               }
3779.                           }
3780.                       }
3781.                       else
3782.                       {
3783.                           if (propSecComboBoxEdit.SelectedIndex == 0)
3784.                           {
3785.                               labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3786.                               labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3787.                           }
3788.                           if (propSecComboBoxEdit.SelectedIndex == 1)
3789.                           {
3790.                               labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3791.                               labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3792.                           }
3793.                           if (propSecComboBoxEdit.SelectedIndex == 2)
3794.                           {
3795.                               labelWrongPropRetEdit.Content =
   "Property Wrong!";
3796.                               labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3797.                           }
3798.                       }
3799.                   }
3800.
```

```
3801.                public void Enable3DOrbit(object sender,
   RoutedEventArgs e)
3802.                {
3803.                    SendStringToInterface(string.Format("Orbit
   Status&True"));
3804.                }
3805.
3806.                public void Disable3DOrbit(object sender,
   RoutedEventArgs e)
3807.                {
3808.                    SendStringToInterface(string.Format("Orbit
   Status&False"));
3809.                }
3810.
3811.                public void EnableNodes(object sender,
   RoutedEventArgs e)
3812.                {
3813.                    SendStringToInterface(string.Format("Node
   Status&True"));
3814.                }
3815.
3816.                public void DisableNodes(object sender,
   RoutedEventArgs e)
3817.                {
3818.                    SendStringToInterface(string.Format("Node
   Status&False"));
3819.                }
3820.
3821.                public void EnableMembers(object sender,
   RoutedEventArgs e)
3822.                {
3823.                    SendStringToInterface(string.Format("Member
   Status&True"));
3824.                }
3825.
3826.                public void DisableMembers(object sender,
   RoutedEventArgs e)
3827.                {
3828.                    SendStringToInterface(string.Format("Member
   Status&False"));
3829.                }
3830.
3831.                public void ResetView(object sender, RoutedEventArgs
   e)
3832.                {
3833.                    SendStringToInterface(string.Format("Reset
   View"));
3834.                }
3835.
3836.                public void RunSimulationClick(object sender,
   RoutedEventArgs e)
3837.                {
3838.
   SendStringToInterface(string.Format("Mode&Mounting Simulation"));
3839.                    MountModels();
3840.
   SendStringToInterface(string.Format("Mode&Running Simulation"));
3841.                    RunSimulations();
3842.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   endDay));
```

```
3843.
3844.                     simulationSuccess = true;
3845.                     EnableSimulationSuccessfullUI();
3846.                 }
3847.
3848.                 public void UpdateUISimulationClick(object sender,
   RoutedEventArgs e)
3849.                 {
3850.                     double diff = endDay - startDay;
3851.
3852.                     double value = double.Parse(SimShowDay.Text) -
   startDay;
3853.
3854.                     double percent = value / diff;
3855.
3856.                     percent = percent * 20d;
3857.
3858.                     SimSlider.Value = percent;
3859.
3860.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   SimShowDay.Text));
3861.                 }
3862.
3863.                 public void EnableSimulationSuccessfullUI()
3864.                 {
3865.                     SimSep1.Visibility = Visibility.Visible;
3866.                     SimLabelDay.Visibility = Visibility.Visible;
3867.                     SimShowDay.Visibility = Visibility.Visible;
3868.                     SimUpdate.Visibility = Visibility.Visible;
3869.                     SimSep2.Visibility = Visibility.Visible;
3870.                     SimSlider.Visibility = Visibility.Visible;
3871.                     SimLabelObject.Visibility = Visibility.Visible;
3872.                     SimPlot.Visibility = Visibility.Visible;
3873.                     SimulationElementResults.Visibility =
   Visibility.Visible;
3874.
3875.                     SimShowDay.Text = endDay.ToString();
3876.                     SimulationElementResults.Text = "No object
   selected";
3877.                     SimPlot.IsEnabled = false;
3878.
3879.                     double diff = endDay - startDay;
3880.
3881.                     double value = double.Parse(SimShowDay.Text) -
   startDay;
3882.
3883.                     double percent = value / diff;
3884.
3885.                     percent = percent * 20d;
3886.
3887.                     SimSlider.Value = percent;
3888.                 }
3889.
3890.                 private void SimSliderValue(object sender,
   RoutedPropertyChangedEventArgs<double> e)
3891.                 {
3892.                     if (!finishedLoading) return;
3893.
3894.                     int diff = endDay - startDay;
3895.
```

```
3896.                   double value = SimSlider.Value / 20d;
3897.
3898.                   SimShowDay.Text = Convert.ToInt32(startDay +
   diff * value).ToString();
3899.
3900.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   SimShowDay.Text));
3901.               }
3902.
3903.               public void MountPlotForNode(object sender,
   RoutedEventArgs e)
3904.               {
3905.                   Plot plot = new Plot();
3906.
3907.                   plot.Show();
3908.               }
3909.
3910.               public void About(object sender, RoutedEventArgs e)
3911.               {
3912.                   About about = new About();
3913.
3914.                   about.Show();
3915.               }
3916.
3917.               public void SendStringToInterface(string text)
3918.               {
3919.                   if (!finishedLoading) return;
3920.
3921.                   text = "%%&" + text + "&%%";
3922.
3923.                   ASCIIEncoding ASEn = new ASCIIEncoding();
3924.
3925.                   byte[] byteArray = ASEn.GetBytes(text);
3926.
3927.                   SendData(byteArray);
3928.               }
3929.
3930.               public void Save(object sender, RoutedEventArgs e)
3931.               {
3932.                   SaveFileDialog saveFileDialog1 = new
   SaveFileDialog();
3933.                   saveFileDialog1.Filter = "Structure
   File|*.struct";
3934.                   saveFileDialog1.Title = "Save a Structure File";
3935.                   saveFileDialog1.ShowDialog();
3936.
3937.                   if (saveFileDialog1.FileName != "")
3938.                   {
3939.                       BinaryFormatter bf = new BinaryFormatter();
3940.                       FileStream file =
   File.Open(saveFileDialog1.FileName, FileMode.OpenOrCreate);
3941.
3942.                       SaveFile save = new SaveFile();
3943.                       save.nodes = nodes;
3944.                       save.members = members;
3945.                       save.materials = materials;
3946.                       save.properties = properties;
3947.                       save.nodalForces = nodalForces;
3948.                       save.memberForces = memberForces;
3949.                       save.startDay = startDay;
```

```
3950.                         save.endDay = endDay;
3951.
3952.                         bf.Serialize(file, save);
3953.                         file.Close();
3954.                     }
3955.                 }
3956.
3957.                 public void Load(object sender, RoutedEventArgs e)
3958.                 {
3959.                     OpenFileDialog openFileDialog1 = new
   OpenFileDialog();
3960.                     openFileDialog1.Filter = "Structure
   File|*.struct";
3961.                     openFileDialog1.Title = "Open a Structure File";
3962.                     openFileDialog1.ShowDialog();
3963.
3964.                     if (openFileDialog1.FileName != "")
3965.                     {
3966.                         BinaryFormatter bf = new BinaryFormatter();
3967.                         FileStream file =
   File.Open(openFileDialog1.FileName, FileMode.Open);
3968.
3969.                         SaveFile save =
   (SaveFile)bf.Deserialize(file);
3970.
3971.                         nodes = save.nodes;
3972.                         members = save.members;
3973.                         materials = save.materials;
3974.                         properties = save.properties;
3975.                         nodalForces = save.nodalForces;
3976.                         memberForces = save.memberForces;
3977.                         startDay = save.startDay;
3978.                         endDay = save.endDay;
3979.
3980.                         MassiveUpdate3DUI();
3981.
3982.                         file.Close();
3983.                     }
3984.                 }
3985.
3986.                 public void MassiveUpdate3DUI()
3987.                 {
3988.                     string command = "Open Massive Structure
   Changes";
3989.
3990.                     for (int i = 0; i < nodes.Count; i++)
3991.                     {
3992.                         command +=
   string.Format("&N&{0}&{1:0.00}&{2:0.00}&{3:0.00}&{4}",
   nodes[i].number, nodes[i].position.x, nodes[i].position.y,
   nodes[i].position.z, nodes[i].restriction.x1 ||
   nodes[i].restriction.y1 || nodes[i].restriction.z1 ||
   nodes[i].restriction.x2 || nodes[i].restriction.y2 ||
   nodes[i].restriction.z2);
3993.                     }
3994.
3995.                     for (int i = 0; i < members.Count; i++)
3996.                     {
3997.                         command +=
   string.Format("&M&{0}&{1}&{2}&{3}", members[i].n0, members[i].n1,
   members[i].startDay, members[i].endDay);
```

```
3998.                     }
3999.
4000.                     SendStringToInterface(command);
4001.                 }
4002.             }
4003.
4004.             [System.Serializable]
4005.             public class SaveFile
4006.             {
4007.                 public List<Nodes> nodes = new List<Nodes>();
4008.                 public List<Member> members = new List<Member>();
4009.
4010.                 public List<Material> materials = new
   List<Material>();
4011.                 public List<Property> properties = new
   List<Property>();
4012.
4013.                 public List<NodalForces> nodalForces = new
   List<NodalForces>();
4014.                 public List<MemberForces> memberForces = new
   List<MemberForces>();
4015.
4016.                 public int startDay;
4017.                 public int endDay;
4018.             }
4019.         }
```

```
1.  using System;
2. using System.Collections.Generic;
3. using System.Diagnostics;
4. using System.Linq;
5. using System.Runtime.InteropServices;
6. using System.Text;
7. using System.Threading.Tasks;
8. using System.Windows;
9. using System.Windows.Controls;
10.  using System.Windows.Data;
11.  using System.Windows.Documents;
12.  using System.Windows.Input;
13.  using System.Windows.Media;
14.  using System.Windows.Media.Imaging;
15.  using System.Windows.Navigation;
16.  using System.Windows.Shapes;
17.  using System.Windows.Forms;
18.  using System.Threading;
19.  using System.Net;
20.  using System.Net.Sockets;
21.  using System.Windows.Interop;
22.  using System.Windows.Threading;
23.  using BriefFiniteElementNet;
24.  using System.Runtime.Serialization.Formatters.Binary;
25.  using System.IO;
26.
27.  namespace ProjectTango
28.  {
29.      [System.Serializable]
30.      public class Vector3
31.      {
32.          public double x;
33.          public double y;
34.          public double z;
```

```
35.
36.         public Vector3()
37.         {
38.             x = 0;
39.             y = 0;
40.             z = 0;
41.         }
42.
43.         public Vector3(string x, string y, string z)
44.         {
45.             this.x = double.Parse(x);
46.             this.y = double.Parse(y);
47.             this.z = double.Parse(z);
48.         }
49.
50.         public Vector3(double x, double y, double z)
51.         {
52.             this.x = x;
53.             this.y = y;
54.             this.z = z;
55.         }
56.     }
57.
58.     [System.Serializable]
59.     public class Restrictions
60.     {
61.         public bool x1;
62.         public bool y1;
63.         public bool z1;
64.         public bool x2;
65.         public bool y2;
66.         public bool z2;
67.
68.         public Restrictions()
69.         {
70.             x1 = false;
71.             y1 = false;
72.             z1 = false;
73.             x2 = false;
74.             y2 = false;
75.             z2 = false;
76.         }
77.
78.         public Restrictions(bool x1, bool y1, bool z1, bool x2,
   bool y2, bool z2)
79.         {
80.             this.x1 = x1;
81.             this.y1 = y1;
82.             this.z1 = z1;
83.             this.x2 = x2;
84.             this.y2 = y2;
85.             this.z2 = z2;
86.         }
87.     }
88.
89.     [System.Serializable]
90.     public class Material
91.     {
92.         public string name;
93.
94.         public double E;
```

```
95.          public double G;
96.          public double v;
97.
98.          public bool timeDependent;
99.
100.         public double fck;
101.
102.         public Material()
103.         {
104.             name = "";
105.
106.             E = 0;
107.             G = 0;
108.             v = 0;
109.
110.             timeDependent = false;
111.
112.             fck = 0;
113.         }
114.
115.         public Material(string name, string E, string G, bool time,
    string fck = "0", string v = "0")
116.         {
117.             this.name = name;
118.
119.             this.E = double.Parse(E);
120.             this.G = double.Parse(G);
121.             this.v = double.Parse(v);
122.
123.             this.timeDependent = time;
124.
125.             this.fck = double.Parse(fck);
126.         }
127.     }
128.
129.     [System.Serializable]
130.     public class Property
131.     {
132.         public string name;
133.
134.         public double A;
135.         public double Ix;
136.         public double Iy;
137.         public double Iz;
138.
139.         public string type;
140.
141.         public double d;
142.
143.         public double h;
144.         public double b;
145.
146.         public Property(string name, string A, string Ix, string
    Iy, string Iz, string type, string d = "0", string h = "0", string b
    = "0")
147.         {
148.             if (d == "") d = "0";
149.             if (b == "") b = "0";
150.             if (h == "") h = "0";
151.
152.             if (type == "0") type = "None";
```

```
153.             if (type == "1") type = "Circular";
154.             if (type == "2") type = "Retangular";
155.
156.             this.name = name;
157.
158.             this.A = double.Parse(A);
159.             this.Ix = double.Parse(Ix);
160.             this.Iy = double.Parse(Iy);
161.             this.Iz = double.Parse(Iz);
162.
163.             this.type = type;
164.
165.             this.d = double.Parse(d);
166.             this.h = double.Parse(h);
167.             this.b = double.Parse(b);
168.         }
169.     }
170.
171.     [System.Serializable]
172.     public class Member
173.     {
174.         public int number;
175.
176.         public int n0;
177.         public int n1;
178.
179.         public string material;
180.         public string section;
181.
182.         public int startDay;
183.         public int endDay;
184.
185.         public Member(int number, int n0, int n1, string material,
    string section, string startDay, string endDay)
186.         {
187.             this.number = number;
188.
189.             this.n0 = n0;
190.             this.n1 = n1;
191.             this.material = material;
192.             this.section = section;
193.
194.             this.startDay = int.Parse(startDay);
195.             this.endDay = int.Parse(endDay);
196.         }
197.     }
198.
199.     [System.Serializable]
200.     public class Nodes
201.     {
202.         public int number;
203.
204.         public Vector3 position;
205.         public Restrictions restriction;
206.
207.         public Nodes()
208.         {
209.             position = new Vector3();
210.             restriction = new Restrictions();
211.         }
212.
```

```
213.         public Nodes(int number, string x, string y, string z,
   bool? x1, bool? y1, bool? z1, bool? x2, bool? y2, bool? z2)
214.         {
215.             this.number = number;
216.
217.             position = new Vector3(x, y, z);
218.             restriction = new Restrictions(x1.GetValueOrDefault(),
   y1.GetValueOrDefault(), z1.GetValueOrDefault(),
   x2.GetValueOrDefault(), y2.GetValueOrDefault(),
   z2.GetValueOrDefault());
219.         }
220.     }
221.
222.     [System.Serializable]
223.     public class NodalForces
224.     {
225.         public string name;
226.
227.         public Vector3 disp;
228.         public Vector3 rott;
229.
230.         public List<int> nodes;
231.
232.         public int startDay;
233.         public int endDay;
234.
235.         public NodalForces()
236.         {
237.             name = "";
238.
239.             disp = new Vector3();
240.             rott = new Vector3();
241.
242.             nodes = new List<int>();
243.
244.             startDay = 0;
245.             endDay = 0;
246.         }
247.
248.         public NodalForces(string name, double dx, double dy,
   double dz, double rx, double ry, double rz, int startDay, int endDay,
   List<int> list)
249.         {
250.             this.name = name;
251.
252.             disp = new Vector3(dx, dy, dz);
253.             rott = new Vector3(rz, ry, rz);
254.
255.             nodes = list;
256.
257.             this.startDay = startDay;
258.             this.endDay = endDay;
259.         }
260.     }
261.
262.     [System.Serializable]
263.     public class MemberForces
264.     {
265.         public string name;
266.
267.         public Vector3 disp;
```

```
268.
269.         public List<int> members;
270.
271.         public int startDay;
272.         public int endDay;
273.
274.         public MemberForces()
275.         {
276.             name = "";
277.
278.             disp = new Vector3();
279.
280.             members = new List<int>();
281.
282.             startDay = 0;
283.             endDay = 0;
284.         }
285.
286.         public MemberForces(string name, double dx, double dy,
    double dz, int startDay, int endDay, List<int> list)
287.         {
288.             this.name = name;
289.
290.             disp = new Vector3(dx, dy, dz);
291.
292.             members = list;
293.
294.             this.startDay = startDay;
295.             this.endDay = endDay;
296.         }
297.     }
298.
299.     /// <summary>
300.     /// Interaction logic for MainWindow.xaml
301.     /// </summary>
302.     public partial class MainWindow : Window
303.     {
304.         public static MainWindow instance;
305.
306.         // Create Form Panel to use as host for Unity 3D Window
307.         private System.Windows.Forms.Panel _panel;
308.         // Create a Process to call Unity 3D Window
309.         private Process _process;
310.         private IntPtr _unityHWND = IntPtr.Zero;
311.
312.         [DllImport("user32.dll")]
313.         private static extern int SetWindowLong(IntPtr hWnd, int
    nIndex, int dwNewLong);
314.
315.         [DllImport("user32.dll", SetLastError = true)]
316.         private static extern int GetWindowLong(IntPtr hWnd, int
    nIndex);
317.
318.         [DllImport("user32")]
319.         private static extern IntPtr SetParent(IntPtr hWnd, IntPtr
    hWndParent);
320.
321.         [DllImport("user32")]
322.         private static extern bool SetWindowPos(IntPtr hWnd, IntPtr
    hWndInsertAfter, int X, int Y, int cx, int cy, int uFlags);
323.
```

```
324.         [DllImport("User32.dll")]
325.         static extern bool MoveWindow(IntPtr handle, int x, int y,
   int width, int height, bool redraw);
326.
327.         internal delegate int WindowEnumProc(IntPtr hwnd, IntPtr
   lparam);
328.         [DllImport("user32.dll")]
329.         internal static extern bool EnumChildWindows(IntPtr hwnd,
   WindowEnumProc func, IntPtr lParam);
330.
331.         [DllImport("user32.dll")]
332.         static extern int SendMessage(IntPtr hWnd, int msg, IntPtr
   wParam, IntPtr lParam);
333.
334.         private const int WM_ACTIVATE = 0x0006;
335.         private readonly IntPtr WA_ACTIVE = new IntPtr(1);
336.         private readonly IntPtr WA_INACTIVE = new IntPtr(0);
337.         private bool finishedLoading = false;
338.
339.         Socket s;
340.         TcpListener myList;
341.         public static string threadCommand = "";
342.         public int tempMemberN0;
343.         public int tempMemberN1;
344.
345.         //Analysis variables.
346.         public List<Nodes> nodes = new List<Nodes>();
347.         public List<Member> members = new List<Member>();
348.
349.         public List<Material> materials = new List<Material>();
350.         public List<Property> properties = new List<Property>();
351.
352.         public List<NodalForces> nodalForces = new
   List<NodalForces>();
353.         public List<MemberForces> memberForces = new
   List<MemberForces>();
354.
355.         public List<BriefFiniteElementNet.Model> models = new
   List<BriefFiniteElementNet.Model>();
356.
357.         // Simulation Variables
358.         public int startDay = 0;
359.         public int endDay = 720;
360.         public bool simulationSuccess = false;
361.         public string lastSimulatedSelected = "";
362.
363.         public MainWindow()
364.         {
365.             // Unhandled exceptions for our Application Domain
366.             AppDomain.CurrentDomain.UnhandledException += new
   System.UnhandledExceptionEventHandler(AppDomain_UnhandledException);
367.
368.             // Unhandled exceptions for the executing UI thread
369.             System.Windows.Forms.Application.ThreadException += new
   System.Threading.ThreadExceptionEventHandler(Application_ThreadExcept
   ion);
370.
371.             instance = this;
372.             //Example3();
373.             //Example2();
```

```
374.             //StructuralAnalysisMathNet analysis = new
   StructuralAnalysisMathNet();
375.             InitializeComponent();
376.             InitializeUnity3D();
377.             InitializeSocket();
378.             finishedLoading = true;
379.         }
380.
381.         //private static void Example3()
382.         //{
383.         //    var Analysis = new SpatialFrameAnalysis.Analysis();
384.
385.         //    var material = new MaterialProprieties(21000000000d,
   6000000000d);
386.         //    var proprieties = new GeometricalProprieties(0.04d,
   0.00013333333d, 0.00013333333d, 0.00026666666d);
387.
388.         //    var n0 = new Node(0, 0, 0, 0, Restriction.Fixed);
389.         //    var n1 = new Node(1, 0, 6, 0, Restriction.Released);
390.
391.         //    Analysis.Nodes.Add(n0);
392.         //    Analysis.Nodes.Add(n1);
393.
394.         //    var m0 = new Member(n0, n1, proprieties, material);
395.
396.         //    Analysis.Members.Add(m0);
397.
398.         //    Analysis.SolveLinear();
399.
400.         //    Console.Write(Analysis.data);
401.         //}
402.
403.         public static bool LineSegmentIntersection(double Ax,
   double Ay, double Bx, double By, double Cx, double Cy, double Dx,
   double Dy, ref double X, ref double Y)
404.         {
405.             double distAB, theCos, theSin, newX, ABpos;
406.
407.             //  Fail if either line segment is zero-length.
408.
409.             if (Ax == Bx && Ay == By || Cx == Dx && Cy == Dy)
   return false;
410.
411.             //  Fail if the segments share an end-point.
412.
413.             if (Ax == Cx && Ay == Cy || Bx == Cx && By == Cy || Ax
   == Dx && Ay == Dy || Bx == Dx && By == Dy)
414.             {
415.                 return false;
416.             }
417.
418.
419.             //  (1) Translate the system so that point A is on the
   origin.
420.
421.             Bx -= Ax; By -= Ay;
422.
423.             Cx -= Ax; Cy -= Ay;
424.
425.             Dx -= Ax; Dy -= Ay;
426.
```

```
427.
428.             //  Discover the length of segment A-B.
429.
430.             distAB = Math.Sqrt(Bx * Bx + By * By);
431.
432.
433.             //  (2) Rotate the system so that point B is on the
    positive X axis.
434.
435.             theCos = Bx / distAB;
436.
437.             theSin = By / distAB;
438.
439.             newX = Cx * theCos + Cy * theSin;
440.
441.             Cy = Cy * theCos - Cx * theSin; Cx = newX;
442.
443.             newX = Dx * theCos + Dy * theSin;
444.
445.             Dy = Dy * theCos - Dx * theSin; Dx = newX;
446.
447.             //  Fail if segment C-D doesn't cross line A-B.
448.
449.             if (Cy < 0 && Dy < 0 || Cy >= 0 && Dy >= 0) return
    false;
450.
451.             //  (3) Discover the position of the intersection point
    along line A-B.
452.
453.             ABpos = Dx + (Cx - Dx) * Dy / (Dy - Cy);
454.
455.             //  Fail if segment C-D crosses line A-B outside of
    segment A-B.
456.
457.             if (ABpos < 0 || ABpos > distAB) return false;
458.
459.             //  (4) Apply the discovered position to line A-B in
    the original coordinate system.
460.
461.             X = (Ax + ABpos * theCos);
462.
463.             Y = (Ay + ABpos * theSin);
464.
465.
466.             //  Success.
467.
468.             return true;
469.         }
470.
471.         public double GetElasticityModulusABNT(double fck, double
    t, double alphae = 1f, double s = 0.25f)
472.         {
473.             if (t < 1) t = 0.5f;
474.
475.             double beta1 = Math.Pow(Math.E, (s * (1 - Math.Sqrt(28
    / t))));
476.
477.             double fckj = beta1 * fck;
478.
479.             if (t > 28) fckj = fck;
480.
```

```
481.             double Eci = 0;
482.             double Ecit = 0;
483.
484.             if (fck >= 20 && fck <= 45)
485.             {
486.                 Eci = alphae * 5600 * Math.Sqrt(fck);
487.                 Ecit = Math.Pow((fckj / fck), 0.5) * Eci;
488.             }
489.             if (fck >= 50 && fck <= 90)
490.             {
491.                 Eci = 21.5 * Math.Pow(10, 3) * alphae *
    Math.Pow((fck / 10) + 1.25, 1 / 3);
492.                 Ecit = Math.Pow((fckj / fck), 0.3) * Eci;
493.             }
494.
495.             // Conversion from MPa to kN/m2;
496.             Ecit = Ecit * 1000;
497.
498.             return Ecit;
499.         }
500.
501.         public double CalculateG(double E, double v)
502.         {
503.             double G = E / (2 * (1 + v));
504.
505.             return G;
506.         }
507.
508.         private void MountModels()
509.         {
510.             try
511.             {
512.                 startDay = int.Parse(simulationDayStart.Text);
513.                 endDay = int.Parse(simulationDayEnd.Text);
514.
515.                 models.Clear();
516.
517.                 for (int i = startDay; i <= endDay; i++)
518.                 {
519.                     var model = new BriefFiniteElementNet.Model();
520.
521.                     var acceptedNodes = new List<int>();
522.
523.                     foreach (Member member in members)
524.                     {
525.                         if (i >= member.startDay)
526.                         {
527.                             if (member.endDay >= i)
528.                             {
529.                                 if
    (!acceptedNodes.Contains(member.n0)) acceptedNodes.Add(member.n0);
530.                                 if
    (!acceptedNodes.Contains(member.n1)) acceptedNodes.Add(member.n1);
531.                             }
532.                         }
533.                     }
534.
535.                     foreach (int n in acceptedNodes)
536.                     {
537.                         var node = new Node(nodes[n].position.x,
    nodes[n].position.y, nodes[n].position.z);
```

```
538.
539.                         if (nodes[n].restriction.x1)
   node.Constraints &= Constraints.FixedDX;
540.                         if (nodes[n].restriction.y1)
   node.Constraints &= Constraints.FixedDY;
541.                         if (nodes[n].restriction.z1)
   node.Constraints &= Constraints.FixedDZ;
542.                         if (nodes[n].restriction.x2)
   node.Constraints &= Constraints.FixedRX;
543.                         if (nodes[n].restriction.y2)
   node.Constraints &= Constraints.FixedRY;
544.                         if (nodes[n].restriction.z2)
   node.Constraints &= Constraints.FixedRZ;
545.
546.                         node.Label = n.ToString();
547.
548.                         model.Nodes.Add(node);
549.                     }
550.
551.                     foreach (Member member in members)
552.                     {
553.                         if (i >= member.startDay)
554.                         {
555.                             if (member.endDay >= i)
556.                             {
557.                                 var mem = new
   FrameElement2Node(model.Nodes.Where(item => item.Label ==
   member.n0.ToString()).First(), model.Nodes.Where(item => item.Label
   == member.n1.ToString()).First());
558.
559.                                 mem.Label = "M" +
   member.number.ToString();
560.
561.                                 if (materials.Where(item =>
   item.name == member.material).First().timeDependent)
562.                                 {
563.                                     mem.E =
   GetElasticityModulusABNT(materials.Where(item => item.name ==
   member.material).First().fck / 1000, i - member.startDay);
564.                                     mem.G = CalculateG(mem.E,
   materials.Where(item => item.name == member.material).First().v);
565.                                 }
566.                                 else
567.                                 {
568.                                     mem.E = materials.Where(item =>
   item.name == member.material).First().E;
569.                                     mem.G = materials.Where(item =>
   item.name == member.material).First().G;
570.                                 }
571.
572.                                 mem.A = properties.Where(item =>
   item.name == member.section).First().A;
573.                                 mem.Iy = properties.Where(item =>
   item.name == member.section).First().Ix;
574.                                 mem.Iz = properties.Where(item =>
   item.name == member.section).First().Iy;
575.                                 mem.J = properties.Where(item =>
   item.name == member.section).First().Iz;
576.
577.                                 mem.UseOverridedProperties = true;
578.
```

```
579.                                     model.Elements.Add(mem);
580.                                 }
581.                             }
582.                         }
583.
584.                         foreach (NodalForces force in nodalForces)
585.                         {
586.                             if (i >= force.startDay)
587.                             {
588.                                 if (force.endDay >= i)
589.                                 {
590.                                     foreach (int node in force.nodes)
591.                                     {
592.                                         if (model.Nodes.Where(item =>
    item.Label == node.ToString()).ToList().Count > 0)
593.                                         {
594.                                             var f = new
    Force(force.disp.x, force.disp.y, force.disp.z, force.rott.x,
    force.rott.y, force.rott.z);
595.                                             model.Nodes.Where(item =>
    item.Label == node.ToString()).First().Loads.Add(new NodalLoad(f));
596.                                         }
597.                                     }
598.                                 }
599.                             }
600.                         }
601.
602.                         foreach (MemberForces force in memberForces)
603.                         {
604.                             if (i >= force.startDay)
605.                             {
606.                                 if (force.endDay >= i)
607.                                 {
608.                                     foreach (int member in
    force.members)
609.                                     {
610.                                         if (model.Elements.Where(item
    => item.Label == "M" + member.ToString()).ToList().Count > 0)
611.                                         {
612.                                             if (force.disp.x != 0)
613.                                             {
614.                                                 var load = new
    UniformLoad1D(force.disp.x, LoadDirection.X,
    CoordinationSystem.Global);
615.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
616.                                             }
617.                                             if (force.disp.y != 0)
618.                                             {
619.                                                 var load = new
    UniformLoad1D(force.disp.y, LoadDirection.Y,
    CoordinationSystem.Global);
620.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
621.                                             }
622.                                             if (force.disp.z != 0)
623.                                             {
```

```
624.                                                    var load = new
    UniformLoad1D(force.disp.z, LoadDirection.Z,
    CoordinationSystem.Global);
625.
    model.Elements.Where(item => item.Label == "M" +
    member.ToString()).First().Loads.Add(load);
626.                                                }
627.                                            }
628.                                        }
629.
630.                                    }
631.                                }
632.                            }
633.
634.                            models.Add(model);
635.                        }
636.                    }
637.                    catch
638.                    {
639.
640.                    }
641.                }
642.
643.                private void RunSimulations()
644.                {
645.                    Parallel.ForEach(models, model =>
646.                    {
647.                        if (model.Nodes.Count > 0 && model.Elements.Count >
    0)
648.                        {
649.                            model.Solve();
650.                        }
651.                    });
652.                }
653.
654.                //private static void Example2()
655.                //{
656.                //    Console.WriteLine("Example 1: Simple 3D Frame with
    distributed loads");
657.
658.                //    var model = new BriefFiniteElementNet.Model();
659.
660.                //    var n1 = new Node(-10, 0, 0);
661.                //    var n2 = new Node(-10, 0, 6);
662.                //    var n3 = new Node(0, 0, 8);
663.                //    var n4 = new Node(10, 0, 6);
664.                //    var n5 = new Node(10, 0, 0);
665.                //    var n6 = new Node(20, 0, 0);
666.
667.                //    model.Nodes.Add(n1, n2, n3, n4, n5);
668.
669.                //    var secAA =
    SectionGenerator.GetRectangularSection(0.2, 0.2);
670.
671.                //    var b =
    PolygonYz.GetSectionGeometricalProperties(secAA);
672.
673.                //    var e1 = new FrameElement2Node(n1, n2);
674.                //    e1.Label = "e1";
675.                //    var e2 = new FrameElement2Node(n2, n3);
676.                //    e2.Label = "e2";
```

```
677.         //    var e3 = new FrameElement2Node(n3, n4);
678.         //    e3.Label = "e3";
679.         //    var e4 = new FrameElement2Node(n4, n5);
680.         //    e4.Label = "e4";
681.
682.         //    e1.Geometry = new PolygonYz(secAA);
683.
684.         //    e1.Geometry = e4.Geometry = e2.Geometry = e3.Geometry
   = new PolygonYz(secAA);
685.
686.         //    e1.E = e2.E = e3.E = e4.E = 210e9;
687.         //    e1.G = e2.G = e3.G = e4.G = 210e9 / (2 * (1 +
   0.3));//G = E / (2*(1+no))
688.
689.         //    e1.UseOverridedProperties =
690.         //        e2.UseOverridedProperties =
   e3.UseOverridedProperties =
691.         //
   e4.UseOverridedProperties = false;
692.
693.         //    model.Elements.Add(e1, e2, e3, e4);
694.
695.         //    n1.Constraints =
696.         //        n2.Constraints =
697.         //            n3.Constraints =
698.         //                n4.Constraints =
699.         //                    n5.Constraints =
700.         //                        Constraint.FixedDY &
   Constraint.FixedRX &
701.         //                        Constraint.FixedRZ;//DY fixed and
   RX fixed and RZ fixed
702.
703.         //    n1.Constraints = n1.Constraints &
   Constraint.MovementFixed;
704.         //    n5.Constraints = n5.Constraints &
   Constraint.MovementFixed;
705.
706.
707.         //    var ll = new UniformLoad1D(-10000, LoadDirection.Z,
   CoordinationSystem.Global);
708.         //    var lr = new UniformLoad1D(-10000, LoadDirection.Z,
   CoordinationSystem.Local);
709.
710.         //    e2.Loads.Add(ll);
711.         //    e3.Loads.Add(lr);
712.
713.         //    //var wnd = WpfTraceListener.CreateModelTrace(model);
714.         //    new ModelWarningChecker().CheckModel(model);
715.         //    //wnd.ShowDialog();
716.
717.         //    model.Solve();
718.
719.         //    var a = n3.GetNodalDisplacement();
720.
721.         //    var bcc = 0;
722.         //}
723.
724.         //private static void Example1()
725.         //{
726.         //    Console.WriteLine("Example 1: Simple 3D truss with
   four members");
```

```
727.
728.         //    // Initiating Model, Nodes and Members
729.         //    var model = new Model();
730.
731.         //    var n1 = new Node(1, 1, 0);
732.         //    n1.Label = "n1";//Set a unique label for node
733.         //    var n2 = new Node(-1, 1, 0) { Label = "n2" };//using
   object initializer for assigning Label
734.         //    var n3 = new Node(1, -1, 0) { Label = "n3" };
735.         //    var n4 = new Node(-1, -1, 0) { Label = "n4" };
736.         //    var n5 = new Node(0, 0, 1) { Label = "n5" };
737.
738.         //    var e1 = new TrussElement2Node(n1, n5) { Label = "e1"
   };
739.         //    var e2 = new TrussElement2Node(n2, n5) { Label = "e2"
   };
740.         //    var e3 = new TrussElement2Node(n3, n5) { Label = "e3"
   };
741.         //    var e4 = new TrussElement2Node(n4, n5) { Label = "e4"
   };
742.
743.         //    //Note: labels for all members should be unique,
744.         //    //else you will receive InvalidLabelException when
   adding it to model
745.
746.         //    e1.A = e2.A = e3.A = e4.A = 9e-4;
747.         //    e1.E = e2.E = e3.E = e4.E = 210e9;
748.
749.
750.         //    model.Nodes.Add(n1, n2, n3, n4, n5);
751.         //    model.Elements.Add(e1, e2, e3, e4);
752.
753.         //    //Applying restraints
754.
755.         //    n1.Constraints = n2.Constraints = n3.Constraints =
   n4.Constraints = Constraint.Fixed;
756.         //    n5.Constraints = Constraint.FixedRX &
   Constraint.FixedRY & Constraint.FixedRZ;/*
   Constraint.RotationFixed;*/
757.
758.         //    //Applying load
759.         //    var force = new Force(0, 1000, 0, 0, 0, 0);
760.         //    n5.Loads.Add(new NodalLoad(force));//adds a load with
   LoadCase of DefaultLoadCase to node loads
761.
762.         //    //Adds a NodalLoad with Default LoadCase
763.
764.         //    model.Solve();
765.
766.         //    var r1 = n1.GetSupportReaction();
767.         //    var r2 = n2.GetSupportReaction();
768.         //    var r3 = n3.GetSupportReaction();
769.         //    var r4 = n4.GetSupportReaction();
770.
771.         //    var dis = n5.GetNodalDisplacement();
772.
773.         //    var a = e1.GetInternalForceAt(1);
774.
775.         //    var rt = r1 + r2 + r3 + r4;//shows the Fz=1000 and
   Fx=Fy=Mx=My=Mz=0.0
776.
```

```
777.         //    Console.WriteLine("Total reactions SUM :" +
   rt.ToString());
778.         //}
779.
780.         private void InitializeSocket()
781.         {
782.             ActivateServer();
783.             CheckMesages();
784.         }
785.
786.         private void ActivateServer()
787.         {
788.             try
789.             {
790.                 IPAddress ipAd = IPAddress.Parse("127.0.0.1");
791.                 // use local m/c IP address, and
792.                 // use the same in the client
793.
794.                 /* Initializes the Listener */
795.                 myList = new TcpListener(ipAd, 8001);
796.
797.                 /* Start Listeneting at the specified port */
798.                 myList.Start();
799.
800.                 Console.WriteLine("The server is running at port
   8001...");
801.                 Console.WriteLine("The local End point is: " +
   myList.LocalEndpoint);
802.                 Console.WriteLine("Waiting for a connection...");
803.
804.                 s = myList.AcceptSocket();
805.
806.                 Console.WriteLine("Connection accepted from " +
   s.RemoteEndPoint);
807.             }
808.             catch (Exception e)
809.             {
810.                 Console.WriteLine("Error: " + e.StackTrace);
811.             }
812.         }
813.
814.         public void CheckMesages()
815.         {
816.             if (threadCommand != "")
817.             {
818.                 string[] commands = threadCommand.Split('&');
819.
820.                 threadCommand = "";
821.
822.
823.             }
824.
825.             new Thread(() =>
826.             {
827.                 Thread.CurrentThread.IsBackground = true;
828.
829.                 try
830.                 {
831.                     bool threadShouldContinue = true;
832.
833.                     while (threadShouldContinue)
```

```
834.                     {
835.
836.
837.                         byte[] b = new byte[131072];
838.
839.                         int k = s.Receive(b);
840.
841.                         string recived = "";
842.
843.                         for (int i = 0; i < k; i++)
844.                         {
845.                             recived += Convert.ToChar(b[i]);
846.                         }
847.
848.                         string[] commands = recived.Split('&');
849.
850.                         if (commands[0] == "No Nodes")
851.                         {
852.                             try
853.                             {
854.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
855.                                     DispatcherPriority.Normal,
856.                                     (Action)(() =>
857.                                     {
858.                                         xCoordNodeEdit.Text = "0
   node selected";
859.                                         yCoordNodeEdit.Text = "0
   node selected";
860.                                         zCoordNodeEdit.Text = "0
   node selected";
861.
862.                                         xCoordNodeEdit.IsEnabled =
   false;
863.                                         yCoordNodeEdit.IsEnabled =
   false;
864.                                         zCoordNodeEdit.IsEnabled =
   false;
865.                                     }
866.                                     ));
867.                             }
868.                             catch
869.                             {
870.
871.                             }
872.                         }
873.
874.                         if (commands[0] == "Edit Node")
875.                         {
876.                             try
877.                             {
878.                                 // Send message to UI thread
879.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
880.                                     DispatcherPriority.Normal,
881.                                     (Action)(() =>
882.                                         {
883.                                             threadCommand =
   recived;
884.
```

```
885.
   xCoordNodeEdit.IsEnabled = true;
886.
   yCoordNodeEdit.IsEnabled = true;
887.
   zCoordNodeEdit.IsEnabled = true;
888.
889.                                                xCoordNodeEdit.Text =
   nodes[int.Parse(commands[1])].position.x.ToString();
890.                                                yCoordNodeEdit.Text =
   nodes[int.Parse(commands[1])].position.y.ToString();
891.                                                zCoordNodeEdit.Text =
   nodes[int.Parse(commands[1])].position.z.ToString();
892.
893.                                                xDispEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.x1;
894.                                                yDispEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.y1;
895.                                                zDispEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.z1;
896.                                                xRotEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.x2;
897.                                                yRotEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.y2;
898.                                                zRotEdit.IsChecked =
   nodes[int.Parse(commands[1])].restriction.z2;
899.                                            }
900.                                        ));
901.                                    }
902.                                    catch
903.                                    {
904.
905.                                    }
906.                                }
907.
908.                                if (commands[0] == "Edit Nodes")
909.                                {
910.                                    try
911.                                    {
912.                                        // Send message to UI thread
913.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
914.                                            DispatcherPriority.Normal,
915.                                            (Action)(() =>
916.                                            {
917.                                                threadCommand = recived;
918.
919.                                                xCoordNodeEdit.Text =
   "Multiple nodes";
920.                                                yCoordNodeEdit.Text =
   "Multiple nodes";
921.                                                zCoordNodeEdit.Text =
   "Multiple nodes";
922.
923.                                                xCoordNodeEdit.IsEnabled =
   false;
924.                                                yCoordNodeEdit.IsEnabled =
   false;
925.                                                zCoordNodeEdit.IsEnabled =
   false;
926.                                            }
```

```
927.                                        ));
928.                                }
929.                                catch
930.                                {
931.
932.                                }
933.                            }
934.
935.                            if (commands[0] == "Create Member - No
    Node")
936.                            {
937.                                try
938.                                {
939.
    System.Windows.Application.Current.Dispatcher.BeginInvoke(
940.                                        DispatcherPriority.Normal,
941.                                        (Action)(() =>
942.                                        {
943.
    memberFirstNodeLabel.Content = "- First Node";
944.
    memberFirstNodeLabel.Foreground = Brushes.Red;
945.
946.                                            tempMemberN0 = -1;
947.                                            tempMemberN1 = -1;
948.                                        }
949.                                        ));
950.                                }
951.                                catch
952.                                {
953.
954.                                }
955.                            }
956.
957.                            if (commands[0] == "Create Member - N0")
958.                            {
959.                                try
960.                                {
961.
    System.Windows.Application.Current.Dispatcher.BeginInvoke(
962.                                        DispatcherPriority.Normal,
963.                                        (Action)(() =>
964.                                        {
965.
    memberFirstNodeLabel.Content = "- First Node: OK";
966.
    memberFirstNodeLabel.Foreground = Brushes.Black;
967.
968.
    memberSecondNodeLabel.Content = "- Second Node";
969.
    memberSecondNodeLabel.Foreground = Brushes.Red;
970.
971.                                            tempMemberN0 =
    int.Parse(commands[1]);
972.                                            tempMemberN1 = -1;
973.                                        }
974.                                        ));
975.                                }
976.                                catch
977.                                {
```

```
978.
979.                                 }
980.                             }
981.
982.                             if (commands[0] == "Create Member - N1")
983.                             {
984.                                 try
985.                                 {
986.
    System.Windows.Application.Current.Dispatcher.BeginInvoke(
987.                                         DispatcherPriority.Normal,
988.                                         (Action)(() =>
989.                                         {
990.
    memberSecondNodeLabel.Content = "- Second Node: OK";
991.
    memberSecondNodeLabel.Foreground = Brushes.Black;
992.
993.                                             tempMemberN1 =
    int.Parse(commands[1]);
994.                                         }
995.                                         ));
996.                                 }
997.                                 catch
998.                                 {
999.
1000.                                        }
1001.                                    }
1002.
1003.                                    if (commands[0] == "Try Member
    Creation")
1004.                                    {
1005.                                        try
1006.                                        {
1007.
    System.Windows.Application.Current.Dispatcher.BeginInvoke(
1008.
    DispatcherPriority.Normal,
1009.                                                (Action)(() =>
1010.                                                {
1011.                                                    if (tempMemberN0 !=
    -1 && tempMemberN1 != -1 && materialMemberComboBox.SelectedItem !=
    null && propertyMemberComboBox.SelectedItem != null)
1012.                                                    {
1013.                                                        members.Add(new
    Member(members.Count, tempMemberN0, tempMemberN1,
    materials[materialMemberComboBox.SelectedIndex].name,
    properties[propertyMemberComboBox.SelectedIndex].name,
    dayMemberStartText.Text, dayMemberEndText.Text));
1014.
1015.                                                        bool changed =
    false;
1016.
1017.                                                        do
1018.                                                        {
1019.                                                            changed =
    false;
1020.
1021.                                                            int
    memberCount = members.Count;
1022.
```

```
1023.                                                       for (int i =
   0; i < memberCount; i++)
1024.                                                       {
1025.                                                           for (int
   j = i; j < memberCount; j++)
1026.                                                           {
1027.                                                               if
   ((nodes[members[i].n0].position.y == nodes[members[j].n0].position.y)
   &&
1028.
   (nodes[members[i].n1].position.y == nodes[members[j].n1].position.y))
1029.                                                                   {
1030.
   double X = 0;
1031.
   double Z = 0;
1032.
1033.
   if (LineSegmentIntersection(
1034.
   nodes[members[i].n0].position.x, nodes[members[i].n0].position.z,
1035.
   nodes[members[i].n1].position.x, nodes[members[i].n1].position.z,
1036.
   nodes[members[j].n0].position.x, nodes[members[j].n0].position.z,
1037.
   nodes[members[j].n1].position.x, nodes[members[j].n1].position.z,
1038.
   ref X, ref Z
1039.
   ))
1040.
   {
1041.
   var tempVector3 = new Vector3(X, nodes[members[i].n0].position.y, Z);
1042.
1043.
   if (nodes[members[i].n0].position == tempVector3)
1044.
   {
1045.
   if ((nodes[members[j].n0].position == tempVector3) ||
   nodes[members[j].n1].position == tempVector3)
1046.
   {
1047.
1048.
   }
1049.
   else
1050.
   {
1051.
   int nodeExisting = members[i].n0;
1052.
1053.
   int tempNode = members[j].n1;
1054.
   members[j].n1 = nodeExisting;
1055.
```

```
1056.
   members.Add(new Member(members.Count, nodeExisting, tempNode,
   members[j].material, members[j].section,
1057.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1058.
1059.
   UpdateMemberForces(members[j].number, members.Count - 1);
1060.
   changed = true;
1061.
   }
1062.
   }
1063.
1064.
   else if (nodes[members[i].n1].position == tempVector3)
1065.
   {
1066.
   if ((nodes[members[j].n0].position == tempVector3) ||
   nodes[members[j].n1].position == tempVector3)
1067.
   {
1068.
1069.
   }
1070.
   else
1071.
   {
1072.
   int nodeExisting = members[i].n1;
1073.
1074.
   int tempNode = members[j].n1;
1075.
   members[j].n1 = nodeExisting;
1076.
1077.
   members.Add(new Member(members.Count, nodeExisting, tempNode,
   members[j].material, members[j].section,
1078.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1079.
1080.
   UpdateMemberForces(members[j].number, members.Count - 1);
1081.
   changed = true;
1082.
   }
1083.
   }
1084.
   else if (nodes[members[j].n0].position == tempVector3)
1085.
   {
1086.
   if ((nodes[members[i].n0].position == tempVector3) ||
   nodes[members[i].n1].position == tempVector3)
```

```
1087.
    {
1088.
1089.
    }
1090.
    else
1091.
    {
1092.
    int nodeExisting = members[j].n0;
1093.
1094.
    int tempNode = members[i].n1;
1095.
    members[i].n1 = nodeExisting;
1096.
1097.
    members.Add(new Member(members.Count, nodeExisting, tempNode,
    members[i].material, members[i].section,
1098.
    members[i].startDay.ToString(), members[i].endDay.ToString()));
1099.
1100.
    UpdateMemberForces(members[i].number, members.Count - 1);
1101.
    changed = true;
1102.
    }
1103.
    }
1104.
1105.
    else if (nodes[members[j].n1].position == tempVector3)
1106.
    {
1107.
    if ((nodes[members[i].n0].position == tempVector3) ||
    nodes[members[i].n1].position == tempVector3)
1108.
    {
1109.
1110.
    }
1111.
    else
1112.
    {
1113.
    int nodeExisting = members[j].n1;
1114.
1115.
    int tempNode = members[i].n1;
1116.
    members[i].n1 = nodeExisting;
1117.
1118.
    members.Add(new Member(members.Count, nodeExisting, tempNode,
    members[i].material, members[i].section,
1119.
    members[i].startDay.ToString(), members[i].endDay.ToString()));
```

```
1120.
1121.
   UpdateMemberForces(members[i].number, members.Count - 1);
1122.
   changed = true;
1123.
   }
1124.
   }
1125.
1126.
   else
1127.
   {
1128.
   nodes.Add(new Nodes(nodes.Count, tempVector3.x.ToString(),
   tempVector3.y.ToString(), tempVector3.z.ToString(), false, false,
   false, false, false, false));
1129.
1130.
   int newNode = nodes.Count - 1;
1131.
1132.
   // FIRST MEMBER:
1133.
   int tempNode = members[i].n1;
1134.
   members[i].n1 = newNode;
1135.
1136.
   members.Add(new Member(members.Count, newNode, tempNode,
   members[i].material, members[i].section,
1137.
   members[i].startDay.ToString(), members[i].endDay.ToString()));
1138.
1139.
   // SECOND MEMBER:
1140.
   tempNode = members[j].n1;
1141.
   members[j].n1 = newNode;
1142.
1143.
   members.Add(new Member(members.Count, newNode, tempNode,
   members[j].material, members[j].section,
1144.
   members[j].startDay.ToString(), members[j].endDay.ToString()));
1145.
1146.
   UpdateMemberForces(members[i].number, members.Count - 2);
1147.
   UpdateMemberForces(members[j].number, members.Count - 1);
1148.
1149.
   changed = true;
1150.
   }
1151.
   }
1152.                                                                    }
```

```
1153.                                                                   if
   (changed) break;
1154.                                                               }
1155.                                                               if
   (changed) break;
1156.                                                           }
1157.
1158.                                                       } while (changed
   == true);
1159.                                                   }
1160.
1161.
   memberFirstNodeLabel.Content = "- First Node";
1162.
   memberFirstNodeLabel.Foreground = Brushes.Red;
1163.
1164.
   memberSecondNodeLabel.Content = "- Second Node";
1165.
   memberSecondNodeLabel.Foreground = Brushes.Red;
1166.
1167.                                                   tempMemberN0 = -1;
1168.                                                   tempMemberN1 = -1;
1169.
1170.                                                   MassiveUpdate3DUI();
1171.                                               }
1172.                                               ));
1173.                                       }
1174.                                       catch
1175.                                       {
1176.
1177.                                       }
1178.                                   }
1179.
1180.                                   if (commands[0] == "Edit Member")
1181.                                   {
1182.                                       try
1183.                                       {
1184.                                           // Send message to UI thread
1185.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1186.
   DispatcherPriority.Normal,
1187.                                               (Action)(() =>
1188.                                               {
1189.                                                   threadCommand =
   recived;
1190.
1191.
   materialMemberEditComboBox.SelectedItem =
   members[int.Parse(commands[1])].material;
1192.
   propertyMemberEditComboBox.SelectedItem =
   members[int.Parse(commands[1])].section;
1193.
1194.
   dayMemberStartEditText.Text =
   members[int.Parse(commands[1])].startDay.ToString();
1195.
   dayMemberEndEditText.Text =
   members[int.Parse(commands[1])].endDay.ToString();
```

```
1196.                                                   }
1197.                                                   ));
1198.                                       }
1199.                                       catch
1200.                                       {
1201.
1202.                                       }
1203.                                   }
1204.
1205.                                   if (commands[0] == "Edit Members")
1206.                                   {
1207.                                       try
1208.                                       {
1209.                                           // Send message to UI thread
1210.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1211.
   DispatcherPriority.Normal,
1212.                                                   (Action)(() =>
1213.                                                   {
1214.                                                       threadCommand =
   recived;
1215.                                                   }
1216.                                                   ));
1217.                                       }
1218.                                       catch
1219.                                       {
1220.
1221.                                       }
1222.                                   }
1223.
1224.                                   if (commands[0] == "No Nodal Force"
   || commands[0] == "Nodal Force")
1225.                                   {
1226.                                       try
1227.                                       {
1228.                                           // Send message to UI thread
1229.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1230.
   DispatcherPriority.Normal,
1231.                                                   (Action)(() =>
1232.                                                   {
1233.                                                       threadCommand =
   recived;
1234.                                                   }
1235.                                                   ));
1236.                                       }
1237.                                       catch
1238.                                       {
1239.
1240.                                       }
1241.                                   }
1242.
1243.                                   if (commands[0] == "No Member Force"
   || commands[0] == "Member Force")
1244.                                   {
1245.                                       try
1246.                                       {
1247.                                           // Send message to UI thread
```

```
1248.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1249.
   DispatcherPriority.Normal,
1250.                                                 (Action)(() =>
1251.                                                 {
1252.                                                     threadCommand =
   recived;
1253.                                                 }
1254.                                                 ));
1255.                                         }
1256.                                         catch
1257.                                         {
1258.
1259.                                         }
1260.                                     }
1261.
1262.                                     if (commands[0] == "Simulation
   Object")
1263.                                     {
1264.                                         try
1265.                                         {
1266.                                             // Send message to UI thread
1267.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1268.
   DispatcherPriority.Normal,
1269.                                                 (Action)(() =>
1270.                                                 {
1271.                                                     if (commands[1] ==
   "Node")
1272.                                                     {
1273.
   UpdateSelected(commands[2]);
1274.                                                     }
1275.
1276.                                                     if (commands[1] ==
   "Member")
1277.                                                     {
1278.
   UpdateSelected("M" + commands[2]);
1279.                                                     }
1280.                                                 }
1281.                                                 ));
1282.                                         }
1283.                                         catch
1284.                                         {
1285.
1286.                                         }
1287.                                     }
1288.
1289.                                     if (commands[0] == "Selected
   Split:")
1290.                                     {
1291.                                         try
1292.                                         {
1293.                                             // Send message to UI thread
1294.
   System.Windows.Application.Current.Dispatcher.BeginInvoke(
1295.
   DispatcherPriority.Normal,
```

```
1296.                                                   (Action)(() =>
1297.                                                   {
1298.                                                       threadCommand =
   recived;
1299.                                                   }
1300.                                                   ));
1301.                                               }
1302.                                               catch
1303.                                               {
1304.
1305.                                               }
1306.                                           }
1307.
1308.
1309.                                           Console.WriteLine("Message: " +
   recived);
1310.                                           //ASCIIEncoding asen = new
   ASCIIEncoding();
1311.                                           //s.Send(asen.GetBytes("Message
   received"));
1312.                                       }
1313.                                   }
1314.                                   catch (Exception e)
1315.                                   {
1316.                                       Console.WriteLine("Error... " +
   e.StackTrace);
1317.                                   }
1318.
1319.                               }).Start();
1320.                           }
1321.
1322.                           private void UpdateMemberForces(int memb, int toAdd)
1323.                           {
1324.                               for (int i = 0; i < memberForces.Count; i++)
1325.                               {
1326.                                   if (memberForces[i].members.Contains(memb))
   memberForces[i].members.Add(toAdd);
1327.                               }
1328.                           }
1329.
1330.                           private void UpdateSelected(string s)
1331.                           {
1332.                               if (s == "") return;
1333.
1334.                               int day = int.Parse(SimShowDay.Text) - startDay;
1335.
1336.                               if (s.ElementAt(0) == 'M')
1337.                               {
1338.                                   string memberSelected = s;
1339.
1340.                                   SimPlot.IsEnabled = false;
1341.
1342.                                   lastSimulatedSelected = memberSelected;
1343.
1344.                                   try
1345.                                   {
1346.                                       var elm =
   models[day].Elements.Where(item => item.Label ==
   memberSelected).First() as FrameElement2Node;
1347.
```

```
1348.                           SimulationElementResults.Text =
   string.Format("Member selected: {0}", memberSelected) +
   System.Environment.NewLine;
1349.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1350.                           SimulationElementResults.Text += "Forces
   at beginning:" + System.Environment.NewLine;
1351.                           SimulationElementResults.Text +=
   "Internal Force (kN):" + System.Environment.NewLine;
1352.                           SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};",
   elm.GetInternalForceAt(0).Fx, elm.GetInternalForceAt(0).Fy,
   elm.GetInternalForceAt(0).Fz) + System.Environment.NewLine;
1353.                           SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1354.                           SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};",
   elm.GetInternalForceAt(0).Mx, elm.GetInternalForceAt(0).My,
   elm.GetInternalForceAt(0).Mz) + System.Environment.NewLine;
1355.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1356.                           SimulationElementResults.Text += "Forces
   at end:" + System.Environment.NewLine;
1357.                           SimulationElementResults.Text +=
   "Internal Force (kN):" + System.Environment.NewLine;
1358.                           SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};",
   elm.GetInternalForceAt(1).Fx, elm.GetInternalForceAt(1).Fy,
   elm.GetInternalForceAt(1).Fz) + System.Environment.NewLine;
1359.                           SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1360.                           SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};",
   elm.GetInternalForceAt(1).Mx, elm.GetInternalForceAt(1).My,
   elm.GetInternalForceAt(1).Mz) + System.Environment.NewLine;
1361.                       }
1362.                       catch
1363.                       {
1364.                           SimulationElementResults.Text =
   string.Format("Member selected: {0}", memberSelected) +
   System.Environment.NewLine;
1365.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1366.                           SimulationElementResults.Text += "Member
   does not yet exist!";
1367.                       }
1368.
1369.
1370.                   }
1371.                   else
1372.                   {
1373.                       string nodeSelected = s;
1374.
1375.                       lastSimulatedSelected = nodeSelected;
1376.
1377.                       SimPlot.IsEnabled = true;
1378.
1379.                       try
1380.                       {
1381.                           var disp = models[day].Nodes.Where(item
   => item.Label == nodeSelected).First().GetNodalDisplacement();
```

```
1382.                           var reac = models[day].Nodes.Where(item
   => item.Label == nodeSelected).First().GetSupportReaction();
1383.
1384.                           SimulationElementResults.Text =
   string.Format("Node selected: {0}", nodeSelected) +
   System.Environment.NewLine;
1385.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1386.                           SimulationElementResults.Text +=
   "Displacement (m):" + System.Environment.NewLine;
1387.                           SimulationElementResults.Text +=
   string.Format("X: {0}; Y: {1}; Z: {2};", disp.DX, disp.DY, disp.DZ) +
   System.Environment.NewLine;
1388.                           SimulationElementResults.Text +=
   "Rotation (rad):" + System.Environment.NewLine;
1389.                           SimulationElementResults.Text +=
   string.Format("X: {0}; Y: {1}; Z: {2};", disp.RX, disp.RY, disp.RZ) +
   System.Environment.NewLine;
1390.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1391.                           SimulationElementResults.Text +=
   "Reactions (kN):" + System.Environment.NewLine;
1392.                           SimulationElementResults.Text +=
   string.Format("FX: {0}; FY: {1}; FZ: {2};", reac.Fx, reac.Fy,
   reac.Fz) + System.Environment.NewLine;
1393.                           SimulationElementResults.Text +=
   "Reactions (kN.m):" + System.Environment.NewLine;
1394.                           SimulationElementResults.Text +=
   string.Format("MX: {0}; MY: {1}; MZ: {2};", reac.Mx, reac.My,
   reac.Mz) + System.Environment.NewLine;
1395.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1396.                       }
1397.                       catch
1398.                       {
1399.                           SimulationElementResults.Text =
   string.Format("Node selected: {0}", nodeSelected) +
   System.Environment.NewLine;
1400.                           SimulationElementResults.Text += "------
   ---------------" + System.Environment.NewLine;
1401.                           SimulationElementResults.Text += "Node
   does not yet exist!";
1402.                       }
1403.
1404.
1405.                   }
1406.               }
1407.
1408.               private void InitializeUnity3D()
1409.               {
1410.                   try
1411.                   {
1412.                       _panel = new System.Windows.Forms.Panel();
1413.                       wfhUnity3DCanvas.Child = _panel;
1414.
1415.                       //ProcessStartInfo psi = new
   ProcessStartInfo("notepad.exe");
1416.                       //_process = Process.Start(psi);
1417.                       //_process.StartInfo.UseShellExecute = true;
1418.                       //_process.StartInfo.CreateNoWindow = true;
1419.                       //_process.WaitForInputIdle();
```

```
1420.
1421.                         _process = new Process();
1422.                         string a =
   AppDomain.CurrentDomain.BaseDirectory;
1423.                         _process.StartInfo.FileName = a +
   "3DInterface\\3DInterface.exe";
1424.                         _process.StartInfo.Arguments = "-parentHWND
   " + _panel.Handle.ToInt32() + " " + Environment.CommandLine;
1425.                         _process.StartInfo.UseShellExecute = true;
1426.                         _process.StartInfo.CreateNoWindow = true;
1427.                         _process.Start();
1428.                         _process.WaitForInputIdle();
1429.
1430.                         SetParent(_process.MainWindowHandle,
   _panel.Handle);
1431.
1432.                         EnumChildWindows(_panel.Handle, WindowEnum,
   IntPtr.Zero);
1433.
1434.                         // remove control box
1435.                         //int style =
   GetWindowLong(_process.MainWindowHandle, GWL_STYLE);
1436.                         //style = style & ~WS_CAPTION &
   ~WS_THICKFRAME;
1437.                         //SetWindowLong(_process.MainWindowHandle,
   GWL_STYLE, style);
1438.
1439.                         // resize embedded application & refresh
1440.                         wfhUnityMockUpCanvas.SizeChanged +=
   WPFCanvas_SizeChanged;
1441.                     }
1442.                     catch (Exception e)
1443.                     {
1444.                     }
1445.                 }
1446.                 /// <summary>
1447.                 /// Main thread exception handler
1448.                 /// </summary>
1449.                 /// <param name="sender">sender</param>
1450.                 /// <param name="e">event</param>
1451.                 public void Application_ThreadException(object
   sender, System.Threading.ThreadExceptionEventArgs e)
1452.                 {
1453.                     if (_process != null)
1454.                     {
1455.                         _process.Refresh();
1456.                         _process.Close();
1457.                     }
1458.
1459.                     // Clean up sockets.
1460.                     s.Close();
1461.                     myList.Stop();
1462.                 }
1463.
1464.                 /// <summary>
1465.                 /// Application domain exception handler
1466.                 /// </summary>
1467.                 /// <param name="sender">sender</param>
1468.                 /// <param name="e">event</param>
1469.                 public void AppDomain_UnhandledException(object
   sender, System.UnhandledExceptionEventArgs e)
```

```
1470.                 {
1471.                     if (_process != null)
1472.                     {
1473.                         _process.Refresh();
1474.                         _process.Close();
1475.                     }
1476.
1477.                     // Clean up sockets.
1478.                     s.Close();
1479.                     myList.Stop();
1480.                 }
1481.
1482.
1483.                 protected override void
   OnClosing(System.ComponentModel.CancelEventArgs e)
1484.                 {
1485.                     base.OnClosing(e);
1486.                     if (_process != null)
1487.                     {
1488.                         _process.Refresh();
1489.                         _process.Close();
1490.                     }
1491.
1492.                     // Clean up sockets.
1493.                     s.Close();
1494.                     myList.Stop();
1495.                 }
1496.
1497.                 private void ResizeEmbeddedApp()
1498.                 {
1499.                     if (_process == null)
1500.                         return;
1501.
1502.                     MoveWindow(_unityHWND, 0, 0,
   (int)wfhUnityMockUpCanvas.ActualWidth,
   (int)wfhUnityMockUpCanvas.ActualHeight, true);
1503.                 }
1504.
1505.                 private void WPFCanvas_SizeChanged(object sender,
   SizeChangedEventArgs e)
1506.                 {
1507.                     ResizeEmbeddedApp();
1508.                 }
1509.
1510.                 private int WindowEnum(IntPtr hwnd, IntPtr lparam)
1511.                 {
1512.                     _unityHWND = hwnd;
1513.                     ActivateUnityWindow();
1514.                     return 0;
1515.                 }
1516.
1517.                 private void ActivateUnityWindow()
1518.                 {
1519.                     SendMessage(_unityHWND, WM_ACTIVATE, WA_ACTIVE,
   IntPtr.Zero);
1520.                 }
1521.
1522.                 private void DeactivateUnityWindow()
1523.                 {
1524.                     SendMessage(_unityHWND, WM_ACTIVATE,
   WA_INACTIVE, IntPtr.Zero);
```

```
1525.               }
1526.
1527.               private void Hand_Click(object sender,
    RoutedEventArgs e)
1528.               {
1529.                   //double[] time = new double[100];
1530.
1531.                   //Parallel.For(0, time.Length,
1532.                   //       index =>
1533.                   //       {
1534.                   //           StructuralAnalysisMathNet analysis
    = new StructuralAnalysisMathNet();
1535.                   //           time[index] = analysis.time;
1536.                   //       });
1537.
1538.                   //double result = 0;
1539.
1540.                   //for (int i = 0; i < time.Length; i++)
1541.                   //{
1542.                   //    result += time[i];
1543.                   //}
1544.
1545.                   //result = result / time.Length;
1546.
1547.                   //abc.Text = result.ToString();
1548.               }
1549.
1550.               private void MainWindow_Loaded(object sender,
    RoutedEventArgs e)
1551.               {
1552.                   ResizeEmbeddedApp();
1553.                   IntPtr hwnd = new
    WindowInteropHelper(this).Handle;
1554.                   HwndSource src = HwndSource.FromHwnd(hwnd);
1555.                   //Receive window message handler implementation
    (based on System.Windows.Interop.HwndSourceHook commissioned)
1556.                   src.AddHook(new HwndSourceHook(WndProc));
1557.               }
1558.
1559.               IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam,
    IntPtr lParam, ref bool handled)
1560.               {
1561.                   switch (msg)
1562.                   {
1563.                       case 528:
1564.                           // Mouse click on Unity Window
1565.                           ActivateUnityWindow();
1566.                           break;
1567.                   }
1568.                   return IntPtr.Zero;
1569.               }
1570.
1571.               private void SendData(byte[] data)
1572.               {
1573.                   SocketAsyncEventArgs socketAsyncData = new
    SocketAsyncEventArgs();
1574.                   socketAsyncData.SetBuffer(data, 0, data.Length);
1575.                   s.SendAsync(socketAsyncData);
1576.
1577.               }
1578.
```

```
1579.                public void CreateMaterial(object sender,
   RoutedEventArgs e)
1580.                {
1581.                    if (matName.Text != "" && matE.Text != "" &&
   matG.Text != "")
1582.                    {
1583.                        try
1584.                        {
1585.                            if (materials.Where(item => item.name ==
   matName.Text).ToList().Count == 0)
1586.                            {
1587.                                if (fckText.Text == "") fckText.Text
   = "0";
1588.
1589.                                materials.Add(new
   Material(matName.Text, matE.Text, matG.Text,
   isTimeDependent.IsChecked.GetValueOrDefault(), fckText.Text,
   matV.Text));
1590.
1591.                                matName.Text = "";
1592.                                matE.Text = "";
1593.                                matG.Text = "";
1594.                                matV.Text = "";
1595.
1596.                                isTimeDependent.IsChecked = false;
1597.
1598.                                fckText.Text = "";
1599.
1600.                                labelWrongMatP.Content = "Material
   Saved!";
1601.                                labelWrongMatP.Visibility =
   Visibility.Visible;
1602.                            }
1603.                            else
1604.                            {
1605.                                labelWrongMatP.Content = "Wrong
   Properties!";
1606.                                labelWrongMatP.Visibility =
   Visibility.Visible;
1607.                            }
1608.                        }
1609.                        catch
1610.                        {
1611.                            labelWrongMatP.Content = "Wrong
   Properties!";
1612.                            labelWrongMatP.Visibility =
   Visibility.Visible;
1613.                        }
1614.                    }
1615.                    else
1616.                    {
1617.                        labelWrongMatP.Content = "Wrong
   Properties!";
1618.                        labelWrongMatP.Visibility =
   Visibility.Visible;
1619.                    }
1620.                }
1621.
1622.                public void UpdateG(object sender, RoutedEventArgs
   e)
1623.                {
```

```
1624.                     if (matE.Text != "" && matV.Text != "")
1625.                     {
1626.                         try
1627.                         {
1628.                             matG.Text = (double.Parse(matE.Text) /
   (2 * (1 + double.Parse(matV.Text)))).ToString();
1629.                         }
1630.                         catch
1631.                         {
1632.                             labelWrongMatP.Content = "Wrong
   Properties!";
1633.                             labelWrongMatP.Visibility =
   Visibility.Visible;
1634.                         }
1635.                     }
1636.                     else
1637.                     {
1638.                         labelWrongMatP.Content = "Wrong
   Properties!";
1639.                         labelWrongMatP.Visibility =
   Visibility.Visible;
1640.                     }
1641.                 }
1642.
1643.                 public void UpdateGEdit(object sender,
   RoutedEventArgs e)
1644.                 {
1645.                     if (matEEdit.Text != "" && matVEdit.Text != "")
1646.                     {
1647.                         try
1648.                         {
1649.                             matGEdit.Text =
   (double.Parse(matEEdit.Text) / (2 * (1 +
   double.Parse(matVEdit.Text)))).ToString();
1650.                         }
1651.                         catch
1652.                         {
1653.                             labelWrongMatPEdit.Content = "Wrong
   Properties!";
1654.                             labelWrongMatPEdit.Visibility =
   Visibility.Visible;
1655.                         }
1656.                     }
1657.                     else
1658.                     {
1659.                         labelWrongMatPEdit.Content = "Wrong
   Properties!";
1660.                         labelWrongMatPEdit.Visibility =
   Visibility.Visible;
1661.                     }
1662.                 }
1663.
1664.                 public void CreateNodalForce(object sender,
   RoutedEventArgs e)
1665.                 {
1666.                     if (nodalForceName.Text == "")
1667.                     {
1668.                         NodalForceError();
1669.                         return;
1670.                     }
1671.
```

```
1672.                     if (nodalForces.Where(item => item.name ==
   nodalForceName.Text).ToList().Count > 0)
1673.                     {
1674.                         NodalForceError();
1675.                         return;
1676.                     }
1677.
1678.                     if (threadCommand == "No Nodal Force")
1679.                     {
1680.                         NodalForceError();
1681.                         return;
1682.                     }
1683.
1684.                     try
1685.                     {
1686.                         if (nodalForceDisX.Text == "")
   nodalForceDisX.Text = "0";
1687.                         if (nodalForceDisY.Text == "")
   nodalForceDisY.Text = "0";
1688.                         if (nodalForceDisZ.Text == "")
   nodalForceDisZ.Text = "0";
1689.                         if (nodalForceRotX.Text == "")
   nodalForceRotX.Text = "0";
1690.                         if (nodalForceRotY.Text == "")
   nodalForceRotY.Text = "0";
1691.                         if (nodalForceRotZ.Text == "")
   nodalForceRotZ.Text = "0";
1692.
1693.                         double dx =
   double.Parse(nodalForceDisX.Text);
1694.                         double dy =
   double.Parse(nodalForceDisY.Text);
1695.                         double dz =
   double.Parse(nodalForceDisZ.Text);
1696.
1697.                         double rx =
   double.Parse(nodalForceRotX.Text);
1698.                         double ry =
   double.Parse(nodalForceRotY.Text);
1699.                         double rz =
   double.Parse(nodalForceRotZ.Text);
1700.
1701.                         int startDay =
   int.Parse(nodalForceStartDay.Text);
1702.                         int endDay =
   int.Parse(nodalForceEndDay.Text);
1703.
1704.                         List<string> listString =
   threadCommand.Split('&').ToList<string>();
1705.
1706.                         List<int> list = new List<int>();
1707.
1708.                         for (int i = 1; i < listString.Count; i++)
1709.                         {
1710.                             list.Add(int.Parse(listString[i]));
1711.                         }
1712.
1713.                         if (dx == 0 && dy == 0 && dz == 0 && rx == 0
   && ry == 0 && rz == 0)
1714.                         {
1715.                             NodalForceError();
```

```
1716.                       }
1717.
1718.                       nodalForces.Add(new
   NodalForces(nodalForceName.Text, dx, dy, dz, rx, ry, rz, startDay,
   endDay, list));
1719.
1720.                       nodalForceName.Text = "";
1721.                       nodalForceDisX.Text = "";
1722.                       nodalForceDisY.Text = "";
1723.                       nodalForceDisZ.Text = "";
1724.                       nodalForceRotX.Text = "";
1725.                       nodalForceRotY.Text = "";
1726.                       nodalForceRotZ.Text = "";
1727.                       nodalForceStartDay.Text = "";
1728.                       nodalForceEndDay.Text = "";
1729.
1730.                       nodalForceError.Visibility =
   Visibility.Collapsed;
1731.
1732.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation"));
1733.                   }
1734.                   catch
1735.                   {
1736.                       NodalForceError();
1737.                   }
1738.               }
1739.
1740.               public void AddNodalForceHeaderSelected(object
   sender, RoutedEventArgs e)
1741.               {
1742.                   SendStringToInterface(string.Format("Mode&Nodal
   Force Creation"));
1743.                   ForcesMainHeader.IsSelected = true;
1744.                   AddNodalForceHeader.IsSelected = true;
1745.
1746.                   nodalForceName.Text = "";
1747.                   nodalForceDisX.Text = "";
1748.                   nodalForceDisY.Text = "";
1749.                   nodalForceDisZ.Text = "";
1750.                   nodalForceRotX.Text = "";
1751.                   nodalForceRotY.Text = "";
1752.                   nodalForceRotZ.Text = "";
1753.                   nodalForceStartDay.Text = "";
1754.                   nodalForceEndDay.Text = "";
1755.
1756.                   nodalForceError.Visibility =
   Visibility.Collapsed;
1757.               }
1758.
1759.               public void EditNodalForcesHeaderSelected(object
   sender, RoutedEventArgs e)
1760.               {
1761.                   nodalForceEditComboBox.Items.Clear();
1762.
1763.                   for (int i = 0; i < nodalForces.Count; i++)
1764.                   {
1765.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1766.                   }
1767.
```

```
1768.                   nodalForceNameEdit.Text = "";
1769.                   nodalForceDisXEdit.Text = "";
1770.                   nodalForceDisYEdit.Text = "";
1771.                   nodalForceDisZEdit.Text = "";
1772.                   nodalForceRotXEdit.Text = "";
1773.                   nodalForceRotYEdit.Text = "";
1774.                   nodalForceRotZEdit.Text = "";
1775.                   nodalForceStartDayEdit.Text = "";
1776.                   nodalForceEndDayEdit.Text = "";
1777.
1778.                   nodalForceNameEdit.IsEnabled = false;
1779.                   nodalForceDisXEdit.IsEnabled = false;
1780.                   nodalForceDisYEdit.IsEnabled = false;
1781.                   nodalForceDisZEdit.IsEnabled = false;
1782.                   nodalForceRotXEdit.IsEnabled = false;
1783.                   nodalForceRotYEdit.IsEnabled = false;
1784.                   nodalForceRotZEdit.IsEnabled = false;
1785.                   nodalForceStartDayEdit.IsEnabled = false;
1786.                   nodalForceEndDayEdit.IsEnabled = false;
1787.
1788.                   nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1789.
1790.                   EditNodalForceHeader.IsSelected = true;
1791.               }
1792.
1793.               public void
   EditNodalForcesComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
1794.               {
1795.                   if (!finishedLoading) return;
1796.
1797.                   if (nodalForceEditComboBox.SelectedItem != null)
1798.                   {
1799.                       nodalForceNameEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().name;
1800.                       nodalForceDisXEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.x.ToStri
   ng();
1801.                       nodalForceDisYEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.y.ToStri
   ng();
1802.                       nodalForceDisZEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.z.ToStri
   ng();
1803.                       nodalForceRotXEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.x.ToStri
   ng();
1804.                       nodalForceRotYEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.y.ToStri
   ng();
1805.                       nodalForceRotZEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.z.ToStri
   ng();
```

```
1806.                         nodalForceStartDayEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().startDay.ToSt
   ring();
1807.                         nodalForceEndDayEdit.Text =
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().endDay.ToStri
   ng();
1808.
1809.                         nodalForceNameEdit.IsEnabled = true;
1810.                         nodalForceDisXEdit.IsEnabled = true;
1811.                         nodalForceDisYEdit.IsEnabled = true;
1812.                         nodalForceDisZEdit.IsEnabled = true;
1813.                         nodalForceRotXEdit.IsEnabled = true;
1814.                         nodalForceRotYEdit.IsEnabled = true;
1815.                         nodalForceRotZEdit.IsEnabled = true;
1816.                         nodalForceStartDayEdit.IsEnabled = true;
1817.                         nodalForceEndDayEdit.IsEnabled = true;
1818.                         nodalForcesEditSave.IsEnabled = true;
1819.                         nodalForcesEditDelete.IsEnabled = true;
1820.
1821.                         nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1822.
1823.                         string listString = "";
1824.
1825.                         for (int i = 0; i < nodalForces.Where(item
   => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes.Count;
   i++)
1826.                         {
1827.                             if (listString == "")
1828.                             {
1829.                                 listString += nodalForces.Where(item
   => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes[i].ToSt
   ring();
1830.                             }
1831.                             else
1832.                             {
1833.                                 listString += "&" +
   nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes[i].ToSt
   ring();
1834.                             }
1835.                         }
1836.
1837.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation&{0}",
   listString));
1838.                     }
1839.                     else
1840.                     {
1841.                         nodalForceNameEdit.Text = "";
1842.                         nodalForceDisXEdit.Text = "";
1843.                         nodalForceDisYEdit.Text = "";
1844.                         nodalForceDisZEdit.Text = "";
1845.                         nodalForceRotXEdit.Text = "";
1846.                         nodalForceRotYEdit.Text = "";
1847.                         nodalForceRotZEdit.Text = "";
1848.                         nodalForceStartDayEdit.Text = "";
```

```
1849.                         nodalForceEndDayEdit.Text = "";
1850.
1851.                         nodalForceNameEdit.IsEnabled = false;
1852.                         nodalForceDisXEdit.IsEnabled = false;
1853.                         nodalForceDisYEdit.IsEnabled = false;
1854.                         nodalForceDisZEdit.IsEnabled = false;
1855.                         nodalForceRotXEdit.IsEnabled = false;
1856.                         nodalForceRotYEdit.IsEnabled = false;
1857.                         nodalForceRotZEdit.IsEnabled = false;
1858.                         nodalForceStartDayEdit.IsEnabled = false;
1859.                         nodalForceEndDayEdit.IsEnabled = false;
1860.                         nodalForcesEditSave.IsEnabled = false;
1861.                         nodalForcesEditDelete.IsEnabled = false;
1862.
1863.                         nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1864.                     }
1865.                 }
1866.
1867.                 public void EditNodalForce(object sender,
   RoutedEventArgs e)
1868.                 {
1869.                     if (nodalForceNameEdit.Text == "")
1870.                     {
1871.                         NodalForceErrorEdit();
1872.                         return;
1873.                     }
1874.
1875.                     if
   (nodalForceEditComboBox.SelectedItem.ToString() !=
   nodalForceNameEdit.Text && nodalForces.Where(item => item.name ==
   nodalForceNameEdit.Text).ToList().Count != 0)
1876.                     {
1877.                         NodalForceErrorEdit();
1878.                         return;
1879.                     }
1880.
1881.                     if (threadCommand == "No Nodal Force")
1882.                     {
1883.                         NodalForceErrorEdit();
1884.                         return;
1885.                     }
1886.
1887.                     try
1888.                     {
1889.                         if (nodalForceDisXEdit.Text == "")
   nodalForceDisXEdit.Text = "0";
1890.                         if (nodalForceDisYEdit.Text == "")
   nodalForceDisYEdit.Text = "0";
1891.                         if (nodalForceDisZEdit.Text == "")
   nodalForceDisZEdit.Text = "0";
1892.                         if (nodalForceRotXEdit.Text == "")
   nodalForceRotXEdit.Text = "0";
1893.                         if (nodalForceRotYEdit.Text == "")
   nodalForceRotYEdit.Text = "0";
1894.                         if (nodalForceRotZEdit.Text == "")
   nodalForceRotZEdit.Text = "0";
1895.
1896.                         double dx =
   double.Parse(nodalForceDisXEdit.Text);
```

```
1897.                         double dy =
   double.Parse(nodalForceDisYEdit.Text);
1898.                         double dz =
   double.Parse(nodalForceDisZEdit.Text);
1899.
1900.                         double rx =
   double.Parse(nodalForceRotXEdit.Text);
1901.                         double ry =
   double.Parse(nodalForceRotYEdit.Text);
1902.                         double rz =
   double.Parse(nodalForceRotZEdit.Text);
1903.
1904.                         int startDay =
   int.Parse(nodalForceStartDayEdit.Text);
1905.                         int endDay =
   int.Parse(nodalForceEndDayEdit.Text);
1906.
1907.                         List<string> listString =
   threadCommand.Split('&').ToList<string>();
1908.
1909.                         List<int> list = new List<int>();
1910.
1911.                         for (int i = 1; i < listString.Count; i++)
1912.                         {
1913.                             list.Add(int.Parse(listString[i]));
1914.                         }
1915.
1916.                         if (dx == 0 && dy == 0 && dz == 0 && rx == 0
   && ry == 0 && rz == 0)
1917.                         {
1918.                             NodalForceErrorEdit();
1919.                         }
1920.
1921.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.x = dx;
1922.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.y = dy;
1923.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().disp.z = dz;
1924.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.x = rx;
1925.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.y = ry;
1926.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().rott.z = rz;
1927.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().startDay =
   startDay;
1928.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().endDay =
   endDay;
1929.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes.Clear()
   ;
1930.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().nodes = list;
1931.
1932.                         nodalForces.Where(item => item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First().name =
   nodalForceNameEdit.Text;
1933.
```

```
1934.                       nodalForceNameEdit.Text = "";
1935.                       nodalForceDisXEdit.Text = "";
1936.                       nodalForceDisYEdit.Text = "";
1937.                       nodalForceDisZEdit.Text = "";
1938.                       nodalForceRotXEdit.Text = "";
1939.                       nodalForceRotYEdit.Text = "";
1940.                       nodalForceRotZEdit.Text = "";
1941.                       nodalForceStartDayEdit.Text = "";
1942.                       nodalForceEndDayEdit.Text = "";
1943.
1944.                       nodalForceNameEdit.IsEnabled = false;
1945.                       nodalForceDisXEdit.IsEnabled = false;
1946.                       nodalForceDisYEdit.IsEnabled = false;
1947.                       nodalForceDisZEdit.IsEnabled = false;
1948.                       nodalForceRotXEdit.IsEnabled = false;
1949.                       nodalForceRotYEdit.IsEnabled = false;
1950.                       nodalForceRotZEdit.IsEnabled = false;
1951.                       nodalForceStartDayEdit.IsEnabled = false;
1952.                       nodalForceEndDayEdit.IsEnabled = false;
1953.                       nodalForcesEditSave.IsEnabled = false;
1954.                       nodalForcesEditDelete.IsEnabled = false;
1955.
1956.                       nodalForceEditComboBox.Items.Clear();
1957.
1958.                       for (int i = 0; i < nodalForces.Count; i++)
1959.                       {
1960.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1961.                       }
1962.
1963.                       nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
1964.
1965.
   SendStringToInterface(string.Format("Mode&Nodal Force Creation"));
1966.                   }
1967.                   catch
1968.                   {
1969.                       NodalForceErrorEdit();
1970.                   }
1971.               }
1972.
1973.               public void DeleteNodalForce(object sender,
   RoutedEventArgs e)
1974.               {
1975.                   nodalForces.Remove(nodalForces.Where(item =>
   item.name ==
   nodalForceEditComboBox.SelectedItem.ToString()).First());
1976.
1977.                   nodalForceEditComboBox.Items.Clear();
1978.
1979.                   for (int i = 0; i < nodalForces.Count; i++)
1980.                   {
1981.
   nodalForceEditComboBox.Items.Add(nodalForces[i].name);
1982.                   }
1983.
1984.                   nodalForceNameEdit.Text = "";
1985.                   nodalForceDisXEdit.Text = "";
1986.                   nodalForceDisYEdit.Text = "";
1987.                   nodalForceDisZEdit.Text = "";
```

```
1988.                   nodalForceRotXEdit.Text = "";
1989.                   nodalForceRotYEdit.Text = "";
1990.                   nodalForceRotZEdit.Text = "";
1991.                   nodalForceStartDayEdit.Text = "";
1992.                   nodalForceEndDayEdit.Text = "";
1993.
1994.                   nodalForceNameEdit.IsEnabled = false;
1995.                   nodalForceDisXEdit.IsEnabled = false;
1996.                   nodalForceDisYEdit.IsEnabled = false;
1997.                   nodalForceDisZEdit.IsEnabled = false;
1998.                   nodalForceRotXEdit.IsEnabled = false;
1999.                   nodalForceRotYEdit.IsEnabled = false;
2000.                   nodalForceRotZEdit.IsEnabled = false;
2001.                   nodalForceStartDayEdit.IsEnabled = false;
2002.                   nodalForceEndDayEdit.IsEnabled = false;
2003.
2004.                   nodalForceErrorEdit.Visibility =
   Visibility.Collapsed;
2005.
2006.                   EditNodalForceHeader.IsSelected = true;
2007.
2008.                   SendStringToInterface(string.Format("Mode&Nodal
   Force Creation"));
2009.               }
2010.
2011.               public void NodalForceError()
2012.               {
2013.                   nodalForceError.Visibility = Visibility.Visible;
2014.               }
2015.
2016.               public void NodalForceErrorEdit()
2017.               {
2018.                   nodalForceErrorEdit.Visibility =
   Visibility.Visible;
2019.               }
2020.
2021.
2022.
2023.               public void CreateMemberForce(object sender,
   RoutedEventArgs e)
2024.               {
2025.                   if (memberForceName.Text == "")
2026.                   {
2027.                       MemberForceError();
2028.                       return;
2029.                   }
2030.
2031.                   if (memberForces.Where(item => item.name ==
   memberForceName.Text).ToList().Count > 0)
2032.                   {
2033.                       MemberForceError();
2034.                       return;
2035.                   }
2036.
2037.                   if (threadCommand == "No Member Force")
2038.                   {
2039.                       MemberForceError();
2040.                       return;
2041.                   }
2042.
2043.                   try
```

```
2044.                   {
2045.                       if (memberForceDisX.Text == "")
   memberForceDisX.Text = "0";
2046.                       if (memberForceDisY.Text == "")
   memberForceDisY.Text = "0";
2047.                       if (memberForceDisZ.Text == "")
   memberForceDisZ.Text = "0";
2048.
2049.                       double dx =
   double.Parse(memberForceDisX.Text);
2050.                       double dy =
   double.Parse(memberForceDisY.Text);
2051.                       double dz =
   double.Parse(memberForceDisZ.Text);
2052.
2053.                       int startDay =
   int.Parse(memberForceStartDay.Text);
2054.                       int endDay =
   int.Parse(memberForceEndDay.Text);
2055.
2056.                       List<string> listString =
   threadCommand.Split('&').ToList<string>();
2057.
2058.                       List<int> list = new List<int>();
2059.
2060.                       for (int i = 1; i < listString.Count; i++)
2061.                       {
2062.                           list.Add(int.Parse(listString[i]));
2063.                       }
2064.
2065.                       if (dx == 0 && dy == 0 && dz == 0)
2066.                       {
2067.                           MemberForceError();
2068.                       }
2069.
2070.                       memberForces.Add(new
   MemberForces(memberForceName.Text, dx, dy, dz, startDay, endDay,
   list));
2071.
2072.                       memberForceName.Text = "";
2073.                       memberForceDisX.Text = "";
2074.                       memberForceDisY.Text = "";
2075.                       memberForceDisZ.Text = "";
2076.                       memberForceStartDay.Text = "";
2077.                       memberForceEndDay.Text = "";
2078.
2079.                       memberForceError.Visibility =
   Visibility.Collapsed;
2080.
2081.
   SendStringToInterface(string.Format("Mode&Member Force Creation"));
2082.                   }
2083.                   catch
2084.                   {
2085.                       MemberForceError();
2086.                   }
2087.               }
2088.
2089.               public void AddMemberForceHeaderSelected(object
   sender, RoutedEventArgs e)
2090.               {
```

```
2091.                   SendStringToInterface(string.Format("Mode&Member
   Force Creation"));
2092.                   ForcesMainHeader.IsSelected = true;
2093.                   AddMemberForceHeader.IsSelected = true;
2094.
2095.                   memberForceName.Text = "";
2096.                   memberForceDisX.Text = "";
2097.                   memberForceDisY.Text = "";
2098.                   memberForceDisZ.Text = "";
2099.                   memberForceStartDay.Text = "";
2100.                   memberForceEndDay.Text = "";
2101.
2102.                   memberForceError.Visibility =
   Visibility.Collapsed;
2103.               }
2104.
2105.               public void EditMemberForcesHeaderSelected(object
   sender, RoutedEventArgs e)
2106.               {
2107.                   memberForceEditComboBox.Items.Clear();
2108.
2109.                   for (int i = 0; i < memberForces.Count; i++)
2110.                   {
2111.
   memberForceEditComboBox.Items.Add(memberForces[i].name);
2112.                   }
2113.
2114.                   memberForceNameEdit.Text = "";
2115.                   memberForceDisXEdit.Text = "";
2116.                   memberForceDisYEdit.Text = "";
2117.                   memberForceDisZEdit.Text = "";
2118.                   memberForceStartDayEdit.Text = "";
2119.                   memberForceEndDayEdit.Text = "";
2120.
2121.                   memberForceNameEdit.IsEnabled = false;
2122.                   memberForceDisXEdit.IsEnabled = false;
2123.                   memberForceDisYEdit.IsEnabled = false;
2124.                   memberForceDisZEdit.IsEnabled = false;
2125.                   memberForceStartDayEdit.IsEnabled = false;
2126.                   memberForceEndDayEdit.IsEnabled = false;
2127.
2128.                   memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2129.
2130.                   EditMemberForceHeader.IsSelected = true;
2131.               }
2132.
2133.               public void
   EditMemberForcesComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
2134.               {
2135.                   if (!finishedLoading) return;
2136.
2137.                   if (memberForceEditComboBox.SelectedItem !=
   null)
2138.                   {
2139.                       memberForceNameEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().name;
2140.                       memberForceDisXEdit.Text =
   memberForces.Where(item => item.name ==
```

```
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.x.ToStr
   ing();
2141.                          memberForceDisYEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.y.ToStr
   ing();
2142.                          memberForceDisZEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.z.ToStr
   ing();
2143.                          memberForceStartDayEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().startDay.ToS
   tring();
2144.                          memberForceEndDayEdit.Text =
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().endDay.ToStr
   ing();
2145.
2146.                          memberForceNameEdit.IsEnabled = true;
2147.                          memberForceDisXEdit.IsEnabled = true;
2148.                          memberForceDisYEdit.IsEnabled = true;
2149.                          memberForceDisZEdit.IsEnabled = true;
2150.                          memberForceStartDayEdit.IsEnabled = true;
2151.                          memberForceEndDayEdit.IsEnabled = true;
2152.                          memberForcesEditSave.IsEnabled = true;
2153.                          memberForcesEditDelete.IsEnabled = true;
2154.
2155.                          memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2156.
2157.                          string listString = "";
2158.
2159.                          for (int i = 0; i < memberForces.Where(item
   => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members.Coun
   t; i++)
2160.                          {
2161.                              if (listString == "")
2162.                              {
2163.                                  listString +=
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members[i].T
   oString();
2164.                              }
2165.                              else
2166.                              {
2167.                                  listString += "&" +
   memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members[i].T
   oString();
2168.                              }
2169.                          }
2170.
2171.
   SendStringToInterface(string.Format("Mode&Member Force Creation&{0}",
   listString));
2172.                      }
2173.                      else
2174.                      {
2175.                          memberForceNameEdit.Text = "";
```

```
2176.                         memberForceDisXEdit.Text = "";
2177.                         memberForceDisYEdit.Text = "";
2178.                         memberForceDisZEdit.Text = "";
2179.                         memberForceStartDayEdit.Text = "";
2180.                         memberForceEndDayEdit.Text = "";
2181.
2182.                         memberForceNameEdit.IsEnabled = false;
2183.                         memberForceDisXEdit.IsEnabled = false;
2184.                         memberForceDisYEdit.IsEnabled = false;
2185.                         memberForceDisZEdit.IsEnabled = false;
2186.                         memberForceStartDayEdit.IsEnabled = false;
2187.                         memberForceEndDayEdit.IsEnabled = false;
2188.                         memberForcesEditSave.IsEnabled = false;
2189.                         memberForcesEditDelete.IsEnabled = false;
2190.
2191.                         memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2192.                     }
2193.                 }
2194.
2195.                 public void EditMemberForce(object sender,
   RoutedEventArgs e)
2196.                 {
2197.                     if (memberForceNameEdit.Text == "")
2198.                     {
2199.                         MemberForceErrorEdit();
2200.                         return;
2201.                     }
2202.
2203.                     if
   (memberForceEditComboBox.SelectedItem.ToString() !=
   memberForceNameEdit.Text && memberForces.Where(item => item.name ==
   memberForceNameEdit.Text).ToList().Count != 0)
2204.                     {
2205.                         MemberForceErrorEdit();
2206.                         return;
2207.                     }
2208.
2209.                     if (threadCommand == "No Member Force")
2210.                     {
2211.                         MemberForceErrorEdit();
2212.                         return;
2213.                     }
2214.
2215.                     try
2216.                     {
2217.                         if (memberForceDisXEdit.Text == "")
   memberForceDisXEdit.Text = "0";
2218.                         if (memberForceDisYEdit.Text == "")
   memberForceDisYEdit.Text = "0";
2219.                         if (memberForceDisZEdit.Text == "")
   memberForceDisZEdit.Text = "0";
2220.
2221.                         double dx =
   double.Parse(memberForceDisXEdit.Text);
2222.                         double dy =
   double.Parse(memberForceDisYEdit.Text);
2223.                         double dz =
   double.Parse(memberForceDisZEdit.Text);
2224.
```

```
2225.                       int startDay =
   int.Parse(memberForceStartDayEdit.Text);
2226.                       int endDay =
   int.Parse(memberForceEndDayEdit.Text);
2227.
2228.                       List<string> listString =
   threadCommand.Split('&').ToList<string>();
2229.
2230.                       List<int> list = new List<int>();
2231.
2232.                       for (int i = 1; i < listString.Count; i++)
2233.                       {
2234.                           if (listString[i] != "")
2235.                           {
2236.                               list.Add(int.Parse(listString[i]));
2237.                           }
2238.                       }
2239.
2240.                       if (dx == 0 && dy == 0 && dz == 0)
2241.                       {
2242.                           MemberForceErrorEdit();
2243.                       }
2244.
2245.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.x = dx;
2246.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.y = dy;
2247.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().disp.z = dz;
2248.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().startDay =
   startDay;
2249.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().endDay =
   endDay;
2250.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members.Clea
   r();
2251.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().members =
   list;
2252.
2253.                       memberForces.Where(item => item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First().name =
   memberForceNameEdit.Text;
2254.
2255.                       memberForceNameEdit.Text = "";
2256.                       memberForceDisXEdit.Text = "";
2257.                       memberForceDisYEdit.Text = "";
2258.                       memberForceDisZEdit.Text = "";
2259.                       memberForceStartDayEdit.Text = "";
2260.                       memberForceEndDayEdit.Text = "";
2261.
2262.                       memberForceNameEdit.IsEnabled = false;
2263.                       memberForceDisXEdit.IsEnabled = false;
2264.                       memberForceDisYEdit.IsEnabled = false;
2265.                       memberForceDisZEdit.IsEnabled = false;
2266.                       memberForceStartDayEdit.IsEnabled = false;
2267.                       memberForceEndDayEdit.IsEnabled = false;
2268.                       memberForcesEditSave.IsEnabled = false;
2269.                       memberForcesEditDelete.IsEnabled = false;
```

```
2270.
2271.                       memberForceEditComboBox.Items.Clear();
2272.
2273.                       for (int i = 0; i < memberForces.Count; i++)
2274.                       {
2275.
   memberForceEditComboBox.Items.Add(memberForces[i].name);
2276.                       }
2277.
2278.                       memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2279.
2280.
   SendStringToInterface(string.Format("Mode&Member Force Creation"));
2281.                   }
2282.                   catch
2283.                   {
2284.                       MemberForceErrorEdit();
2285.                   }
2286.               }
2287.
2288.               public void DeleteMemberForce(object sender,
   RoutedEventArgs e)
2289.               {
2290.                   memberForces.Remove(memberForces.Where(item =>
   item.name ==
   memberForceEditComboBox.SelectedItem.ToString()).First());
2291.
2292.                   memberForceEditComboBox.Items.Clear();
2293.
2294.                   for (int i = 0; i < memberForces.Count; i++)
2295.                   {
2296.
   memberForceEditComboBox.Items.Add(memberForces[i].name);
2297.                   }
2298.
2299.                   memberForceNameEdit.Text = "";
2300.                   memberForceDisXEdit.Text = "";
2301.                   memberForceDisYEdit.Text = "";
2302.                   memberForceDisZEdit.Text = "";
2303.                   memberForceStartDayEdit.Text = "";
2304.                   memberForceEndDayEdit.Text = "";
2305.
2306.                   memberForceNameEdit.IsEnabled = false;
2307.                   memberForceDisXEdit.IsEnabled = false;
2308.                   memberForceDisYEdit.IsEnabled = false;
2309.                   memberForceDisZEdit.IsEnabled = false;
2310.                   memberForceStartDayEdit.IsEnabled = false;
2311.                   memberForceEndDayEdit.IsEnabled = false;
2312.
2313.                   memberForceErrorEdit.Visibility =
   Visibility.Collapsed;
2314.
2315.                   EditMemberForceHeader.IsSelected = true;
2316.
2317.                   SendStringToInterface(string.Format("Mode&Member
   Force Creation"));
2318.               }
2319.
2320.               public void MemberForceError()
2321.               {
```

```
2322.                   nodalForceError.Visibility = Visibility.Visible;
2323.               }
2324.
2325.               public void MemberForceErrorEdit()
2326.               {
2327.                   nodalForceErrorEdit.Visibility =
   Visibility.Visible;
2328.               }
2329.
2330.               public void CreateNode(object sender,
   RoutedEventArgs e)
2331.               {
2332.                   try
2333.                   {
2334.                       nodes.Add(new Nodes(nodes.Count,
   xCoordNode.Text, yCoordNode.Text, zCoordNode.Text, xDisp.IsChecked,
   yDisp.IsChecked, zDisp.IsChecked, xRot.IsChecked, yRot.IsChecked,
   zRot.IsChecked));
2335.
2336.                       SendStringToInterface(string.Format("Create
   Node&{0}&{1}&{2}&{3}&{4}", nodes.Count - 1, xCoordNode.Text,
   yCoordNode.Text, zCoordNode.Text, xDisp.IsChecked.GetValueOrDefault()
   || yDisp.IsChecked.GetValueOrDefault() ||
   zDisp.IsChecked.GetValueOrDefault() ||
   xRot.IsChecked.GetValueOrDefault() ||
   yRot.IsChecked.GetValueOrDefault() ||
   zRot.IsChecked.GetValueOrDefault()));
2337.                   }
2338.                   catch
2339.                   {
2340.
2341.                   }
2342.               }
2343.
2344.               public void EditNode(object sender, RoutedEventArgs
   e)
2345.               {
2346.                   if (xCoordNodeEdit.Text == "0 node selected" ||
   xCoordNodeEdit.Text == "Multiple nodes")
2347.                   {
2348.                       if (xCoordNodeEdit.Text == "Multiple nodes")
2349.                       {
2350.                           List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2351.
2352.                           for (int i = 1; i < tempList.Count; i++)
2353.                           {
2354.
   nodes[int.Parse(tempList[i])].restriction = new
   Restrictions(xDispEdit.IsChecked.GetValueOrDefault(),
   yDispEdit.IsChecked.GetValueOrDefault(),
   zDispEdit.IsChecked.GetValueOrDefault(),
   xRotEdit.IsChecked.GetValueOrDefault(),
   yRotEdit.IsChecked.GetValueOrDefault(),
   zRotEdit.IsChecked.GetValueOrDefault());
2355.                           }
2356.                       }
2357.
2358.                       //SendStringToInterface(string.Format("Edit
   Node Support&{0}", xDispEdit.IsChecked.GetValueOrDefault() ||
   yDispEdit.IsChecked.GetValueOrDefault() ||
```

```
   zDispEdit.IsChecked.GetValueOrDefault() ||
   xRotEdit.IsChecked.GetValueOrDefault() ||
   yRotEdit.IsChecked.GetValueOrDefault() ||
   zRotEdit.IsChecked.GetValueOrDefault()));
                    }
                    else
                    {
                        List<string> tempList =
   threadCommand.Split('&').ToList<string>();

                        nodes[int.Parse(tempList[1])].position = new
   Vector3(xCoordNodeEdit.Text, yCoordNodeEdit.Text,
   zCoordNodeEdit.Text);
                        nodes[int.Parse(tempList[1])].restriction =
   new Restrictions(xDispEdit.IsChecked.GetValueOrDefault(),
   yDispEdit.IsChecked.GetValueOrDefault(),
   zDispEdit.IsChecked.GetValueOrDefault(),
   xRotEdit.IsChecked.GetValueOrDefault(),
   yRotEdit.IsChecked.GetValueOrDefault(),
   zRotEdit.IsChecked.GetValueOrDefault());

                        //SendStringToInterface(string.Format("Edit
   Node&{0}&{1}&{2}&{3}", xCoordNodeEdit.Text, yCoordNodeEdit.Text,
   zCoordNodeEdit.Text, xDispEdit.IsChecked.GetValueOrDefault() ||
   yDispEdit.IsChecked.GetValueOrDefault() ||
   zDispEdit.IsChecked.GetValueOrDefault() ||
   xRotEdit.IsChecked.GetValueOrDefault() ||
   yRotEdit.IsChecked.GetValueOrDefault() ||
   zRotEdit.IsChecked.GetValueOrDefault()));
                    }

                    MassiveUpdate3DUI();
                }

                public void DeleteNode(object sender,
   RoutedEventArgs e)
                {
                    try
                    {
                        List<string> tempList =
   threadCommand.Split('&').ToList<string>();

                        int count = 0;

                        for (int i = 1; i < tempList.Count; i++)
                        {
                            for (int j = 0; j < members.Count; j++)
                            {
                                bool remove = false;

                                if (members[j].n0 ==
   int.Parse(tempList[i]))
                                {
                                    remove = true;
                                }
                                if (members[j].n1 ==
   int.Parse(tempList[i]))
                                {
                                    remove = true;
                                }

```

```
2396.                                   if (remove)
2397.                                   {
2398.                                       members.RemoveAt(j);
2399.
2400.                                       for (int k = 0; k <
   memberForces.Count; k++)
2401.                                       {
2402.                                           if
   (memberForces[k].members.Contains(j))
2403.                                           {
2404.
   memberForces[k].members.Remove(j);
2405.
2406.                                               for (int l = 0; l <
   memberForces[k].members.Count; l++)
2407.                                               {
2408.                                                   if
   (memberForces[k].members[l] > j)
2409.                                                   {
2410.
   memberForces[k].members[l]--;
2411.                                                   }
2412.                                               }
2413.                                           }
2414.                                           else
2415.                                           {
2416.                                               for (int l = 0; l <
   memberForces[k].members.Count; l++)
2417.                                               {
2418.                                                   if
   (memberForces[k].members[l] > j)
2419.                                                   {
2420.
   memberForces[k].members[l]--;
2421.                                                   }
2422.                                               }
2423.                                           }
2424.
2425.
2426.                                       }
2427.
2428.                                       j--;
2429.                                   }
2430.                               }
2431.
2432.                               for (int j = 0; j < nodalForces.Count;
   j++)
2433.                               {
2434.                                   if
   (nodalForces[j].nodes.Contains(int.Parse(tempList[i])))
   nodalForces[j].nodes.Remove(int.Parse(tempList[i]));
2435.                               }
2436.
2437.                               for (int j = 0; j < memberForces.Count;
   j++)
2438.                               {
2439.                                   if (memberForces[j].members.Count ==
   0) memberForces.RemoveAt(j);
2440.                               }
2441.
```

```
2442.                             nodes.RemoveAt(int.Parse(tempList[i]) -
   count);
2443.                             count++;
2444.                         }
2445.
2446.                         for (int i = 0; i < nodes.Count; i++)
2447.                         {
2448.                             for (int j = 0; j < members.Count; j++)
2449.                             {
2450.                                 if (members[j].n0 ==
   nodes[i].number) members[j].n0 = i;
2451.                                 if (members[j].n1 ==
   nodes[i].number) members[j].n1 = i;
2452.                             }
2453.
2454.                             nodes[i].number = i;
2455.                         }
2456.
2457.                         for (int i = 0; i < members.Count; i++)
2458.                         {
2459.                             members[i].number = i;
2460.                         }
2461.
2462.                         for (int j = 0; j < nodalForces.Count; j++)
2463.                         {
2464.                             if (nodalForces[j].nodes.Count == 0)
2465.                             {
2466.                                 nodalForces.RemoveAt(j);
2467.                                 j--;
2468.                             }
2469.                         }
2470.
2471.                         xCoordNodeEdit.Text = "0 node selected";
2472.                         yCoordNodeEdit.Text = "0 node selected";
2473.                         zCoordNodeEdit.Text = "0 node selected";
2474.
2475.                         xCoordNodeEdit.IsEnabled = false;
2476.                         yCoordNodeEdit.IsEnabled = false;
2477.                         zCoordNodeEdit.IsEnabled = false;
2478.
2479.
   //SendStringToInterface(string.Format("Delete Node"));
2480.
2481.                         MassiveUpdate3DUI();
2482.                     }
2483.                     catch
2484.                     {
2485.
2486.                     }
2487.                 }
2488.
2489.                 public void MainHeaderSelectedMouse(object sender,
   MouseButtonEventArgs e)
2490.                 {
2491.                     MainHeader();
2492.                 }
2493.
2494.                 public void MainHeaderSelected(object sender,
   RoutedEventArgs e)
2495.                 {
2496.                     MainHeader();
```

```
2497.                 }
2498.
2499.                 public void MainHeader()
2500.                 {
2501.
   SendStringToInterface(string.Format("Mode&General"));
2502.
2503.                     if (GeometryMainHeader.IsSelected)
2504.                     {
2505.                         GeometryGeneralHeader.IsSelected = true;
2506.                     }
2507.
2508.                     if (PropertiesMainHeader.IsSelected)
2509.                     {
2510.                         PropertiesGeneralHeader.IsSelected = true;
2511.                     }
2512.
2513.                     if (ForcesMainHeader.IsSelected)
2514.                     {
2515.                         ForcesGeneralHeader.IsSelected = true;
2516.                     }
2517.
2518.                     if (SimulationMainHeader.IsSelected)
2519.                     {
2520.                         SimulationGeneral();
2521.                     }
2522.                 }
2523.
2524.                 public void SimulationGeneralHeaderSelected(object
   sender, RoutedEventArgs e)
2525.                 {
2526.                     SimulationGeneral();
2527.                 }
2528.
2529.                 public void SimulationClear(object sender,
   RoutedEventArgs e)
2530.                 {
2531.                     simulationSuccess = false;
2532.
2533.                     models.Clear();
2534.
2535.                     SimulationGeneral();
2536.                 }
2537.
2538.                 public void SimulationGeneral()
2539.                 {
2540.                     SimulationGeneralHeader.IsSelected = true;
2541.
2542.                     simulationDayStart.Text = startDay.ToString();
2543.                     simulationDayEnd.Text = endDay.ToString();
2544.
2545.                     if (!simulationSuccess)
2546.                     {
2547.                         SimSep1.Visibility = Visibility.Collapsed;
2548.                         SimLabelDay.Visibility =
   Visibility.Collapsed;
2549.                         SimShowDay.Visibility =
   Visibility.Collapsed;
2550.                         SimUpdate.Visibility = Visibility.Collapsed;
2551.                         SimSep2.Visibility = Visibility.Collapsed;
2552.                         SimSlider.Visibility = Visibility.Collapsed;
```

```
2553.                         SimLabelObject.Visibility =
   Visibility.Collapsed;
2554.                         SimPlot.Visibility = Visibility.Collapsed;
2555.                         SimulationElementResults.Visibility =
   Visibility.Collapsed;
2556.                     }
2557.                 }
2558.
2559.                 public void AddNodeHeaderSelected(object sender,
   RoutedEventArgs e)
2560.                 {
2561.                     SendStringToInterface(string.Format("Mode&Node
   Creation"));
2562.                     GeometryMainHeader.IsSelected = true;
2563.                     AddNodeHeader.IsSelected = true;
2564.                 }
2565.
2566.                 public void EditNodeHeaderSelected(object sender,
   RoutedEventArgs e)
2567.                 {
2568.                     SendStringToInterface(string.Format("Mode&Node
   Editor"));
2569.                     GeometryMainHeader.IsSelected = true;
2570.                     EditNodeHeader.IsSelected = true;
2571.
2572.                     xCoordNodeEdit.Text = "0 node selected";
2573.                     yCoordNodeEdit.Text = "0 node selected";
2574.                     zCoordNodeEdit.Text = "0 node selected";
2575.
2576.                     xCoordNodeEdit.IsEnabled = false;
2577.                     yCoordNodeEdit.IsEnabled = false;
2578.                     zCoordNodeEdit.IsEnabled = false;
2579.
2580.                     xDispEdit.IsChecked = false;
2581.                     yDispEdit.IsChecked = false;
2582.                     zDispEdit.IsChecked = false;
2583.                     xRotEdit.IsChecked = false;
2584.                     yRotEdit.IsChecked = false;
2585.                     zRotEdit.IsChecked = false;
2586.                 }
2587.
2588.                 public void OptionsHeaderSelected(object sender,
   RoutedEventArgs e)
2589.                 {
2590.
   SendStringToInterface(string.Format("Mode&Options"));
2591.                     OptionsHeader.IsSelected = true;
2592.                 }
2593.
2594.                 public void AddMaterialHeaderSelected(object sender,
   RoutedEventArgs e)
2595.                 {
2596.
   SendStringToInterface(string.Format("Mode&Material Creation"));
2597.                     PropertiesMainHeader.IsSelected = true;
2598.                     AddMaterialHeader.IsSelected = true;
2599.
2600.                     fckText.IsEnabled = false;
2601.                     labelWrongMatP.Visibility =
   Visibility.Collapsed;
2602.                 }
```

```
2603.
2604.                 public void AddMemberHeaderSelected(object sender,
   RoutedEventArgs e)
2605.                 {
2606.                     SendStringToInterface(string.Format("Mode&Member
   Creation"));
2607.                     GeometryMainHeader.IsSelected = true;
2608.                     AddMemberHeader.IsSelected = true;
2609.
2610.                     materialMemberComboBox.Items.Clear();
2611.                     propertyMemberComboBox.Items.Clear();
2612.
2613.                     for (int i = 0; i < materials.Count; i++)
2614.                     {
2615.
   materialMemberComboBox.Items.Add(materials[i].name);
2616.                     }
2617.
2618.                     for (int i = 0; i < properties.Count; i++)
2619.                     {
2620.
   propertyMemberComboBox.Items.Add(properties[i].name);
2621.                     }
2622.
2623.                     memberMatSecLabel.Content = "- Material /
   Selection";
2624.                     memberMatSecLabel.Foreground = Brushes.Red;
2625.
2626.                     memberFirstNodeLabel.Content = "- First Node";
2627.                     memberFirstNodeLabel.Foreground = Brushes.Red;
2628.
2629.                     memberSecondNodeLabel.Content = "- Second Node";
2630.                     memberSecondNodeLabel.Foreground = Brushes.Red;
2631.                 }
2632.
2633.                 public void AddMemberMaterialComboBoxChanged(object
   sender, SelectionChangedEventArgs e)
2634.                 {
2635.                     if (materialMemberComboBox.SelectedItem != null)
2636.                     {
2637.                         if (propertyMemberComboBox.SelectedItem !=
   null)
2638.                         {
2639.                             memberMatSecLabel.Content = "-
   Material/Selection OK";
2640.                             memberMatSecLabel.Foreground =
   Brushes.Black;
2641.                         }
2642.                     }
2643.                 }
2644.
2645.                 public void AddMemberPropertyComboBoxChanged(object
   sender, SelectionChangedEventArgs e)
2646.                 {
2647.                     if (propertyMemberComboBox.SelectedItem != null)
2648.                     {
2649.                         if (materialMemberComboBox.SelectedItem !=
   null)
2650.                         {
2651.                             memberMatSecLabel.Content = "-
   Material/Selection OK";
```

```
2652.                             memberMatSecLabel.Foreground =
   Brushes.Black;
2653.                         }
2654.                     }
2655.                 }
2656.
2657.                 public void EditMemberHeaderSelected(object sender,
   RoutedEventArgs e)
2658.                 {
2659.                     SendStringToInterface(string.Format("Mode&Member
   Editor"));
2660.                     GeometryMainHeader.IsSelected = true;
2661.                     EditMemberHeader.IsSelected = true;
2662.
2663.                     materialMemberEditComboBox.Items.Clear();
2664.                     propertyMemberEditComboBox.Items.Clear();
2665.
2666.                     for (int i = 0; i < materials.Count; i++)
2667.                     {
2668.
   materialMemberEditComboBox.Items.Add(materials[i].name);
2669.                     }
2670.
2671.                     for (int i = 0; i < properties.Count; i++)
2672.                     {
2673.
   propertyMemberEditComboBox.Items.Add(properties[i].name);
2674.                     }
2675.
2676.                     //SendStringToInterface(string.Format("Show
   Days&All"));
2677.                 }
2678.
2679.                 public void SplitMemberHeaderSelected(object sender,
   RoutedEventArgs e)
2680.                 {
2681.                     SendStringToInterface(string.Format("Mode&Split
   Member"));
2682.                     GeometryMainHeader.IsSelected = true;
2683.                     SplitMemberHeader.IsSelected = true;
2684.                 }
2685.
2686.                 public void EditMember(object sender,
   RoutedEventArgs e)
2687.                 {
2688.                     try
2689.                     {
2690.                         List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2691.
2692.                         if (tempList.Count > 1)
2693.                         {
2694.                             for (int i = 1; i < tempList.Count; i++)
2695.                             {
2696.
   members[int.Parse(tempList[i])].material =
   materialMemberEditComboBox.SelectedItem.ToString();
2697.
   members[int.Parse(tempList[i])].section =
   propertyMemberEditComboBox.SelectedItem.ToString();
```

```
2698.
   members[int.Parse(tempList[i])].startDay =
   int.Parse(dayMemberStartEditText.Text);
2699.
   members[int.Parse(tempList[i])].endDay =
   int.Parse(dayMemberEndEditText.Text);
2700.                               }
2701.                           }
2702.
2703.                           MassiveUpdate3DUI();
2704.                       }
2705.                       catch
2706.                       {
2707.
2708.                       }
2709.                   }
2710.
2711.                   public void DeleteMember(object sender,
   RoutedEventArgs e)
2712.                   {
2713.                       try
2714.                       {
2715.                           List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2716.
2717.                           if (tempList.Count > 1)
2718.                           {
2719.                               int count = 0;
2720.                               for (int i = 1; i < tempList.Count; i++)
2721.                               {
2722.
   members.RemoveAt(int.Parse(tempList[i]) - count);
2723.
2724.                                   for (int j = 0; j <
   memberForces.Count; j++)
2725.                                   {
2726.                                       if
   (memberForces[j].members.Contains(int.Parse(tempList[i]) - count))
2727.                                       {
2728.
   memberForces[j].members.Remove(int.Parse(tempList[i]) - count);
2729.
2730.                                           for (int k = 0; k <
   memberForces[j].members.Count; k++)
2731.                                           {
2732.                                               if
   (memberForces[j].members[k] > int.Parse(tempList[i]) - count)
2733.                                               {
2734.
   memberForces[j].members[k]--;
2735.                                               }
2736.                                           }
2737.                                       }
2738.                                       else
2739.                                       {
2740.                                           for (int k = 0; k <
   memberForces[j].members.Count; k++)
2741.                                           {
2742.                                               if
   (memberForces[j].members[k] > int.Parse(tempList[i]) - count)
2743.                                               {
```

```
2744.
   memberForces[j].members[k]--;
2745.                                                   }
2746.                                               }
2747.                                           }
2748.                                       }
2749.
2750.                                       count++;
2751.                                   }
2752.                               }
2753.
2754.                               for (int j = 0; j < memberForces.Count; j++)
2755.                               {
2756.                                   if (memberForces[j].members.Count == 0)
   memberForces.RemoveAt(j);
2757.                               }
2758.
2759.                               for (int i = 0; i < members.Count; i++)
2760.                               {
2761.                                   members[i].number = i;
2762.                               }
2763.
2764.                               MassiveUpdate3DUI();
2765.                           }
2766.                           catch
2767.                           {
2768.
2769.                           }
2770.                       }
2771.
2772.                       private void SplitMember(object sender,
   RoutedEventArgs e)
2773.                       {
2774.                           try
2775.                           {
2776.                               List<string> tempList =
   threadCommand.Split('&').ToList<string>();
2777.
2778.                               if (tempList[0] == "Selected Split:" &&
   tempList.Count > 1)
2779.                               {
2780.                                   int splitCount =
   int.Parse(splitPieces.Text);
2781.
2782.                                   if (splitCount < 1)
2783.                                   {
2784.                                       splitCount = 1;
2785.                                       splitPieces.Text = "1";
2786.                                   }
2787.
2788.                                   int originalMember =
   int.Parse(tempList[1]);
2789.
2790.                                   double diffX =
   nodes[members[originalMember].n1].position.x -
   nodes[members[originalMember].n0].position.x;
2791.                                   double diffY =
   nodes[members[originalMember].n1].position.y -
   nodes[members[originalMember].n0].position.y;
```

```
2792.                           double diffZ =
   nodes[members[originalMember].n1].position.z -
   nodes[members[originalMember].n0].position.z;
2793.
2794.                           diffX = diffX / splitCount;
2795.                           diffY = diffY / splitCount;
2796.                           diffZ = diffZ / splitCount;
2797.
2798.                           int originalEndNode =
   members[originalMember].n1;
2799.                           int lastNode = 0;
2800.
2801.                           for (int i = 1; i <= splitCount; i++)
2802.                           {
2803.                               if (i == 1)
2804.                               {
2805.                                   nodes.Add(new Nodes(nodes.Count,
2806.
   (nodes[members[originalMember].n0].position.x + i *
   diffX).ToString(),
2807.
   (nodes[members[originalMember].n0].position.y + i *
   diffY).ToString(),
2808.
   (nodes[members[originalMember].n0].position.z + i *
   diffZ).ToString(),
2809.                                       false, false, false, false,
   false, false));
2810.
2811.                                   lastNode = nodes.Count - 1;
2812.
2813.                                   members[originalMember].n1 =
   lastNode;
2814.                               }
2815.                               else if (i == splitCount)
2816.                               {
2817.                                   members.Add(new
   Member(members.Count, lastNode, originalEndNode,
   members[originalMember].material, members[originalMember].section,
2818.
   members[originalMember].startDay.ToString(),
   members[originalMember].endDay.ToString()));
2819.
2820.
   UpdateMemberForces(originalMember, members.Count - 1);
2821.                               }
2822.                               else
2823.                               {
2824.                                   nodes.Add(new Nodes(nodes.Count,
2825.
   (nodes[members[originalMember].n0].position.x + i *
   diffX).ToString(),
2826.
   (nodes[members[originalMember].n0].position.y + i *
   diffY).ToString(),
2827.
   (nodes[members[originalMember].n0].position.z + i *
   diffZ).ToString(),
2828.                                       false, false, false, false,
   false, false));
2829.
```

```
2830.                                       members.Add(new
   Member(members.Count, lastNode, nodes.Count - 1,
   members[originalMember].material, members[originalMember].section,
2831.
   members[originalMember].startDay.ToString(),
   members[originalMember].endDay.ToString()));
2832.
2833.                                       lastNode = nodes.Count - 1;
2834.
2835.
   UpdateMemberForces(originalMember, members.Count - 1);
2836.                                   }
2837.                               }
2838.                           }
2839.
2840.                           MassiveUpdate3DUI();
2841.                       }
2842.                       catch
2843.                       {
2844.
2845.                       }
2846.                   }
2847.
2848.                   public void NewFile(object sender, RoutedEventArgs
   e)
2849.                   {
2850.                       try
2851.                       {
2852.                           members.Clear();
2853.                           nodes.Clear();
2854.                           materials.Clear();
2855.                           properties.Clear();
2856.                           nodalForces.Clear();
2857.                           memberForces.Clear();
2858.
2859.                           PropertiesMainHeader.IsSelected = true;
2860.
2861.                           simulationSuccess = false;
2862.                           lastSimulatedSelected = "";
2863.
2864.
2865.                           MassiveUpdate3DUI();
2866.                       }
2867.                       catch
2868.                       {
2869.
2870.                       }
2871.                   }
2872.
2873.                   public void Exit(object sender, RoutedEventArgs e)
2874.                   {
2875.                       System.Windows.Application.Current.Shutdown();
2876.                   }
2877.
2878.
2879.                   public void IsTimeDependentChecked(object sender,
   RoutedEventArgs e)
2880.                   {
2881.                       fckText.IsEnabled = true;
2882.                   }
2883.
```

```
2884.                public void IsTimeDependentUnchecked(object sender,
   RoutedEventArgs e)
2885.                {
2886.                    fckText.IsEnabled = false;
2887.                    fckText.Text = "";
2888.                }
2889.
2890.                public void IsTimeDependentEditChecked(object
   sender, RoutedEventArgs e)
2891.                {
2892.                    fckTextEdit.IsEnabled = true;
2893.                }
2894.
2895.                public void IsTimeDependentEditUnchecked(object
   sender, RoutedEventArgs e)
2896.                {
2897.                    fckTextEdit.IsEnabled = false;
2898.                    fckTextEdit.Text = "";
2899.                }
2900.
2901.                public void EntireStructureChecked(object sender,
   RoutedEventArgs e)
2902.                {
2903.                    if (!finishedLoading) return;
2904.                    structureDays.IsEnabled = false;
2905.                }
2906.
2907.                public void EntireStructureUnchecked(object sender,
   RoutedEventArgs e)
2908.                {
2909.                    if (!finishedLoading) return;
2910.                    structureDays.IsEnabled = true;
2911.                }
2912.
2913.                public void UpdateStructureTime(object sender,
   RoutedEventArgs e)
2914.                {
2915.                    if (structureDays.IsEnabled)
2916.                    {
2917.                        try
2918.                        {
2919.                            int day = int.Parse(structureDays.Text);
2920.
   SendStringToInterface(string.Format("Show Days&{0}", day));
2921.                        }
2922.                        catch
2923.                        {
2924.
2925.                        }
2926.                    }
2927.                    else
2928.                    {
2929.                        SendStringToInterface(string.Format("Show
   Days&All"));
2930.                    }
2931.                }
2932.
2933.                public void EditMaterialHeaderSelected(object
   sender, RoutedEventArgs e)
2934.                {
```

```
2935.
   SendStringToInterface(string.Format("Mode&Material Editor"));
2936.                     EditMaterialHeader.IsSelected = true;
2937.                     PropertiesMainHeader.IsSelected = true;
2938.                     labelWrongMatPEdit.Visibility =
   Visibility.Collapsed;
2939.
2940.                     matComboBox.Items.Clear();
2941.
2942.                     for (int i = 0; i < materials.Count; i++)
2943.                     {
2944.                         matComboBox.Items.Add(materials[i].name);
2945.                     }
2946.                 }
2947.
2948.                 public void
   EditMaterialComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
2949.                 {
2950.                     if (matComboBox.SelectedItem != null)
2951.                     {
2952.                         matNameEdit.Text = materials.Where(item =>
   item.name == matComboBox.SelectedItem.ToString()).First().name;
2953.                         matEEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().E.ToString();
2954.                         matGEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().G.ToString();
2955.                         matVEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().v.ToString();
2956.                         fckTextEdit.Text = materials.Where(item =>
   item.name ==
   matComboBox.SelectedItem.ToString()).First().fck.ToString();
2957.
2958.                         isTimeDependentEdit.IsChecked =
   materials.Where(item => item.name ==
   matComboBox.SelectedItem.ToString()).First().timeDependent;
2959.
2960.
2961.                         matNameEdit.IsEnabled = true;
2962.                         matEEdit.IsEnabled = true;
2963.                         matGEdit.IsEnabled = true;
2964.                         matVEdit.IsEnabled = true;
2965.                         isTimeDependentEdit.IsEnabled = true;
2966.                         fckTextEdit.IsEnabled =
   isTimeDependentEdit.IsChecked.GetValueOrDefault();
2967.
2968.                         saveEditMaterial.IsEnabled = true;
2969.                         deleteEditMaterial.IsEnabled = true;
2970.                         calcGEdit.IsEnabled = true;
2971.                     }
2972.                     else
2973.                     {
2974.                         matNameEdit.Text = "";
2975.                         matEEdit.Text = "";
2976.                         matGEdit.Text = "";
2977.                         matVEdit.Text = "";
2978.                         fckTextEdit.Text = "";
2979.
```

```
2980.                         matNameEdit.IsEnabled = false;
2981.                         matEEdit.IsEnabled = false;
2982.                         matGEdit.IsEnabled = false;
2983.                         matVEdit.IsEnabled = false;
2984.                         isTimeDependentEdit.IsChecked = false;
2985.                         isTimeDependentEdit.IsEnabled = false;
2986.
2987.                         saveEditMaterial.IsEnabled = false;
2988.                         deleteEditMaterial.IsEnabled = false;
2989.                         calcGEdit.IsEnabled = false;
2990.                     }
2991.                 }
2992.
2993.                 public void SaveEditMaterial(object sender,
     RoutedEventArgs e)
2994.                 {
2995.                     try
2996.                     {
2997.                         string matName =
     matComboBox.SelectedItem.ToString();
2998.
2999.                         if (matName != matNameEdit.Text &&
     materials.Where(item => item.name == matNameEdit.Text).ToList().Count
     != 0)
3000.                         {
3001.                             labelWrongMatPEdit.Content = "Wrong
     Properties!";
3002.                             labelWrongMatPEdit.Visibility =
     Visibility.Visible;
3003.                         }
3004.                         else
3005.                         {
3006.                             materials.Where(item => item.name ==
     matName).First().E = double.Parse(matEEdit.Text);
3007.                             materials.Where(item => item.name ==
     matName).First().G = double.Parse(matGEdit.Text);
3008.                             materials.Where(item => item.name ==
     matName).First().v = double.Parse(matVEdit.Text);
3009.                             materials.Where(item => item.name ==
     matName).First().timeDependent =
     isTimeDependentEdit.IsChecked.GetValueOrDefault();
3010.                             if (fckTextEdit.Text == "")
     fckTextEdit.Text = "0";
3011.                             materials.Where(item => item.name ==
     matName).First().fck = double.Parse(fckTextEdit.Text);
3012.                             materials.Where(item => item.name ==
     matName).First().name = matNameEdit.Text;
3013.
3014.                             for (int i = 0; i < members.Count; i++)
3015.                             {
3016.                                 if (members[i].material ==
     matComboBox.SelectedItem.ToString()) members[i].material =
     matNameEdit.Text;
3017.                             }
3018.
3019.                             matComboBox.Items.Clear();
3020.
3021.                             for (int i = 0; i < materials.Count;
     i++)
3022.                             {
```

```
3023.
   matComboBox.Items.Add(materials[i].name);
3024.                               }
3025.
3026.                               labelWrongMatPEdit.Content = "Material
   Saved!";
3027.                               labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3028.                           }
3029.                       }
3030.                       catch
3031.                       {
3032.
3033.                       }
3034.                   }
3035.
3036.                   public void DeleteEditMaterial(object sender,
   RoutedEventArgs e)
3037.                   {
3038.                       try
3039.                       {
3040.                           materials.Remove(materials.Where(item =>
   item.name == matComboBox.SelectedItem.ToString()).First());
3041.
3042.                           matComboBox.Items.Clear();
3043.
3044.                           for (int i = 0; i < materials.Count; i++)
3045.                           {
3046.
   matComboBox.Items.Add(materials[i].name);
3047.                           }
3048.
3049.                           labelWrongMatPEdit.Content = "Material
   Removed!";
3050.                           labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3051.                       }
3052.                       catch
3053.                       {
3054.
3055.                       }
3056.                   }
3057.
3058.                   public void AddPropertyHeaderSelected(object sender,
   RoutedEventArgs e)
3059.                   {
3060.
   SendStringToInterface(string.Format("Mode&Property Creation"));
3061.                       AddPropertiesHeader.IsSelected = true;
3062.                       PropertiesMainHeader.IsSelected = true;
3063.                       labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3064.                       labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3065.                       labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3066.                   }
3067.
3068.                   public void
   EditPropertyComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
```

```
3069.                {
3070.                    if (!finishedLoading) return;
3071.
3072.                    if (propSecComboBox.SelectedIndex == 0)
3073.                    {
3074.                        propNoneSep.Visibility = Visibility.Visible;
3075.                        propNoneSave.Visibility =
   Visibility.Visible;
3076.                        labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3077.
3078.                        propDiamLabel.Visibility =
   Visibility.Collapsed;
3079.                        propDiam.Visibility = Visibility.Collapsed;
3080.                        propDiamCalc.Visibility =
   Visibility.Collapsed;
3081.                        propDiamSep.Visibility =
   Visibility.Collapsed;
3082.                        propDiamSave.Visibility =
   Visibility.Collapsed;
3083.                        labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3084.
3085.                        propHLabel.Visibility =
   Visibility.Collapsed;
3086.                        propH.Visibility = Visibility.Collapsed;
3087.                        propBLabel.Visibility =
   Visibility.Collapsed;
3088.                        propB.Visibility = Visibility.Collapsed;
3089.                        propRetCalc.Visibility =
   Visibility.Collapsed;
3090.                        propRetSep.Visibility =
   Visibility.Collapsed;
3091.                        propRetSave.Visibility =
   Visibility.Collapsed;
3092.                        labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3093.                    }
3094.
3095.                    if (propSecComboBox.SelectedIndex == 1)
3096.                    {
3097.                        propNoneSep.Visibility =
   Visibility.Collapsed;
3098.                        propNoneSave.Visibility =
   Visibility.Collapsed;
3099.                        labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3100.
3101.                        propDiamLabel.Visibility =
   Visibility.Visible;
3102.                        propDiam.Visibility = Visibility.Visible;
3103.                        propDiamCalc.Visibility =
   Visibility.Visible;
3104.                        propDiamSep.Visibility = Visibility.Visible;
3105.                        propDiamSave.Visibility =
   Visibility.Visible;
3106.                        labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3107.
3108.                        propHLabel.Visibility =
   Visibility.Collapsed;
```

```
3109.                       propH.Visibility = Visibility.Collapsed;
3110.                       propBLabel.Visibility =
   Visibility.Collapsed;
3111.                       propB.Visibility = Visibility.Collapsed;
3112.                       propRetCalc.Visibility =
   Visibility.Collapsed;
3113.                       propRetSep.Visibility =
   Visibility.Collapsed;
3114.                       propRetSave.Visibility =
   Visibility.Collapsed;
3115.                       labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3116.                   }
3117.
3118.                   if (propSecComboBox.SelectedIndex == 2)
3119.                   {
3120.                       propNoneSep.Visibility =
   Visibility.Collapsed;
3121.                       propNoneSave.Visibility =
   Visibility.Collapsed;
3122.                       labelWrongPropNone.Visibility =
   Visibility.Collapsed;
3123.
3124.                       propDiamLabel.Visibility =
   Visibility.Collapsed;
3125.                       propDiam.Visibility = Visibility.Collapsed;
3126.                       propDiamCalc.Visibility =
   Visibility.Collapsed;
3127.                       propDiamSep.Visibility =
   Visibility.Collapsed;
3128.                       propDiamSave.Visibility =
   Visibility.Collapsed;
3129.                       labelWrongPropDiam.Visibility =
   Visibility.Collapsed;
3130.
3131.                       propHLabel.Visibility = Visibility.Visible;
3132.                       propH.Visibility = Visibility.Visible;
3133.                       propBLabel.Visibility = Visibility.Visible;
3134.                       propB.Visibility = Visibility.Visible;
3135.                       propRetCalc.Visibility = Visibility.Visible;
3136.                       propRetSep.Visibility = Visibility.Visible;
3137.                       propRetSave.Visibility = Visibility.Visible;
3138.                       labelWrongPropRet.Visibility =
   Visibility.Collapsed;
3139.                   }
3140.               }
3141.
3142.               public void CalculateCircularProperty(object sender,
   RoutedEventArgs e)
3143.               {
3144.                   try
3145.                   {
3146.                       double A = Math.PI *
   Math.Pow(double.Parse(propDiam.Text) / 2, 2);
3147.                       double Ix = (Math.PI / 4) *
   Math.Pow(double.Parse(propDiam.Text) / 2, 4);
3148.                       double Iy = Ix;
3149.                       double Iz = (Math.PI / 2) *
   Math.Pow(double.Parse(propDiam.Text) / 2, 4);
3150.
3151.                       propA.Text = A.ToString();
```

```
3152.                       propIx.Text = Ix.ToString();
3153.                       propIy.Text = Iy.ToString();
3154.                       propIz.Text = Iz.ToString();
3155.                   }
3156.                   catch
3157.                   {
3158.
3159.                   }
3160.               }
3161.
3162.               public void CalculateRetangularProperty(object
   sender, RoutedEventArgs e)
3163.               {
3164.                   try
3165.                   {
3166.                       double A = double.Parse(propH.Text) *
   double.Parse(propB.Text);
3167.                       double Ix = double.Parse(propB.Text) *
   Math.Pow(double.Parse(propH.Text), 3) / 12;
3168.                       double Iy = double.Parse(propH.Text) *
   Math.Pow(double.Parse(propB.Text), 3) / 12;
3169.                       double Iz = Ix + Iy;
3170.
3171.                       propA.Text = A.ToString();
3172.                       propIx.Text = Ix.ToString();
3173.                       propIy.Text = Iy.ToString();
3174.                       propIz.Text = Iz.ToString();
3175.                   }
3176.                   catch
3177.                   {
3178.
3179.                   }
3180.               }
3181.
3182.               public void CreateProperty(object sender,
   RoutedEventArgs e)
3183.               {
3184.                   if (propName.Text != "" && propA.Text != "" &&
   propIx.Text != "" && propIy.Text != "" && propIz.Text != "")
3185.                   {
3186.                       try
3187.                       {
3188.                           if (properties.Where(item => item.name
   == propName.Text).ToList().Count == 0)
3189.                           {
3190.                               properties.Add(new
   Property(propName.Text, propA.Text, propIx.Text, propIy.Text,
   propIz.Text, propSecComboBox.SelectedIndex.ToString(), propDiam.Text,
   propH.Text, propB.Text));
3191.
3192.                               propName.Text = "";
3193.                               propA.Text = "";
3194.                               propIx.Text = "";
3195.                               propIy.Text = "";
3196.                               propIz.Text = "";
3197.                               propSecComboBox.SelectedIndex = 0;
3198.                               propDiam.Text = "";
3199.                               propH.Text = "";
3200.                               propB.Text = "";
3201.
```

```
3202.                                   labelWrongPropNone.Content =
   "Property Saved!";
3203.                                   labelWrongPropNone.Visibility =
   Visibility.Visible;
3204.
3205.                               }
3206.                               else
3207.                               {
3208.                                   if (propSecComboBox.SelectedIndex ==
   0)
3209.                                   {
3210.                                       labelWrongPropNone.Content =
   "Wrong Properties!";
3211.                                       labelWrongPropNone.Visibility =
   Visibility.Visible;
3212.                                   }
3213.                                   if (propSecComboBox.SelectedIndex ==
   1)
3214.                                   {
3215.                                       labelWrongPropDiam.Content =
   "Wrong Properties!";
3216.                                       labelWrongPropDiam.Visibility =
   Visibility.Visible;
3217.                                   }
3218.                                   if (propSecComboBox.SelectedIndex ==
   2)
3219.                                   {
3220.                                       labelWrongPropRet.Content =
   "Wrong Properties!";
3221.                                       labelWrongPropRet.Visibility =
   Visibility.Visible;
3222.                                   }
3223.                               }
3224.                           }
3225.                           catch
3226.                           {
3227.                               if (propSecComboBox.SelectedIndex == 0)
3228.                               {
3229.                                   labelWrongPropNone.Content = "Wrong
   Properties!";
3230.                                   labelWrongPropNone.Visibility =
   Visibility.Visible;
3231.                               }
3232.                               if (propSecComboBox.SelectedIndex == 1)
3233.                               {
3234.                                   labelWrongPropDiam.Content = "Wrong
   Properties!";
3235.                                   labelWrongPropDiam.Visibility =
   Visibility.Visible;
3236.                               }
3237.                               if (propSecComboBox.SelectedIndex == 2)
3238.                               {
3239.                                   labelWrongPropRet.Content = "Wrong
   Properties!";
3240.                                   labelWrongPropRet.Visibility =
   Visibility.Visible;
3241.                               }
3242.                           }
3243.                       }
3244.                       else
3245.                       {
```

```
3246.                         if (propSecComboBox.SelectedIndex == 0)
3247.                         {
3248.                             labelWrongPropNone.Content = "Wrong
   Properties!";
3249.                             labelWrongPropNone.Visibility =
   Visibility.Visible;
3250.                         }
3251.                         if (propSecComboBox.SelectedIndex == 1)
3252.                         {
3253.                             labelWrongPropDiam.Content = "Wrong
   Properties!";
3254.                             labelWrongPropDiam.Visibility =
   Visibility.Visible;
3255.                         }
3256.                         if (propSecComboBox.SelectedIndex == 2)
3257.                         {
3258.                             labelWrongPropRet.Content = "Wrong
   Properties!";
3259.                             labelWrongPropRet.Visibility =
   Visibility.Visible;
3260.                         }
3261.                     }
3262.                 }
3263.
3264.                 public void EditPropertyHeaderSelected(object
   sender, RoutedEventArgs e)
3265.                 {
3266.
   SendStringToInterface(string.Format("Mode&Property Editor"));
3267.                     EditPropertiesHeader.IsSelected = true;
3268.                     PropertiesMainHeader.IsSelected = true;
3269.                     labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3270.                     labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3271.                     labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3272.
3273.                     editComboBox.Items.Clear();
3274.
3275.                     for (int i = 0; i < properties.Count; i++)
3276.                     {
3277.                         editComboBox.Items.Add(properties[i].name);
3278.                     }
3279.
3280.                     propNameEdit.Text = "";
3281.                     propAEdit.Text = "";
3282.                     propIxEdit.Text = "";
3283.                     propIyEdit.Text = "";
3284.                     propIzEdit.Text = "";
3285.
3286.                     propNameEdit.IsEnabled = false;
3287.
3288.                     propAEdit.IsEnabled = false;
3289.                     propIxEdit.IsEnabled = false;
3290.                     propIyEdit.IsEnabled = false;
3291.                     propIzEdit.IsEnabled = false;
3292.
3293.                     editComboBox.IsEnabled = true;
3294.
3295.                     propSecComboBoxEdit.IsEnabled = false;
```

```
3296.                   propSecComboBoxEdit.SelectedIndex = 0;
3297.
3298.                   propNoneSaveEdit.IsEnabled = false;
3299.                   propNoneDeleteEdit.IsEnabled = false;
3300.
3301.                   propNoneSaveEdit.Visibility =
   Visibility.Visible;
3302.                   propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3303.                   propNoneSepEdit.Visibility = Visibility.Visible;
3304.                   labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3305.
3306.                   propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3307.                   propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3308.                   propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3309.                   propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3310.                   propDiamEdit.Visibility = Visibility.Collapsed;
3311.                   labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3312.
3313.                   propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3314.                   propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3315.                   propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3316.                   propRetSep.Visibility = Visibility.Collapsed;
3317.                   propHEdit.Visibility = Visibility.Collapsed;
3318.                   propBEdit.Visibility = Visibility.Collapsed;
3319.                   labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3320.               }
3321.
3322.               public void
   EditPropertyListComboBoxSelectionChanged(object sender,
   SelectionChangedEventArgs e)
3323.               {
3324.                   if (editComboBox.SelectedItem != null)
3325.                   {
3326.                       propNameEdit.Text = properties.Where(item =>
   item.name == editComboBox.SelectedItem.ToString()).First().name;
3327.                       propAEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().A.ToString();
3328.                       propIxEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Ix.ToString();
3329.                       propIyEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Iy.ToString();
3330.                       propIzEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().Iz.ToString();
3331.
3332.                       if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "None")
```

```
3333.                         {
3334.                             propSecComboBoxEdit.IsEnabled = true;
3335.                             propSecComboBoxEdit.SelectedIndex = 0;
3336.
3337.                             propNoneSaveEdit.IsEnabled = true;
3338.                             propNoneDeleteEdit.IsEnabled = true;
3339.
3340.                             propNoneSaveEdit.Visibility =
   Visibility.Visible;
3341.                             propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3342.                             propNoneSepEdit.Visibility =
   Visibility.Visible;
3343.                             labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3344.
3345.                             propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3346.                             propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3347.                             propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3348.                             propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3349.                             propDiamEdit.Visibility =
   Visibility.Collapsed;
3350.                             propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3351.                             labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3352.
3353.                             propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3354.                             propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3355.                             propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3356.                             propRetSepEdit.Visibility =
   Visibility.Collapsed;
3357.                             propHEdit.Visibility =
   Visibility.Collapsed;
3358.                             propBEdit.Visibility =
   Visibility.Collapsed;
3359.                             propHLabelEdit.Visibility =
   Visibility.Collapsed;
3360.                             propBLabelEdit.Visibility =
   Visibility.Collapsed;
3361.                             labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3362.
3363.                         }
3364.                         if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "Circular")
3365.                         {
3366.                             propSecComboBoxEdit.IsEnabled = true;
3367.                             propSecComboBoxEdit.SelectedIndex = 1;
3368.
3369.                             propNoneSaveEdit.IsEnabled = false;
3370.                             propNoneDeleteEdit.IsEnabled = false;
3371.
3372.                             propDiamCalcEdit.IsEnabled = true;
```

```
3373.                           propDiamSaveEdit.IsEnabled = true;
3374.                           propDiamDeleteEdit.IsEnabled = true;
3375.                           propDiamEdit.IsEnabled = true;
3376.                           propDiamSepEdit.IsEnabled = true;
3377.
3378.                           propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3379.                           propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3380.                           propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3381.                           labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3382.
3383.                           propDiamCalcEdit.Visibility =
   Visibility.Visible;
3384.                           propDiamSaveEdit.Visibility =
   Visibility.Visible;
3385.                           propDiamDeleteEdit.Visibility =
   Visibility.Visible;
3386.                           propDiamSepEdit.Visibility =
   Visibility.Visible;
3387.                           propDiamEdit.Visibility =
   Visibility.Visible;
3388.                           propDiamLabelEdit.Visibility =
   Visibility.Visible;
3389.                           labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3390.
3391.                           propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3392.                           propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3393.                           propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3394.                           propRetSepEdit.Visibility =
   Visibility.Collapsed;
3395.                           propHEdit.Visibility =
   Visibility.Collapsed;
3396.                           propBEdit.Visibility =
   Visibility.Collapsed;
3397.                           propHLabelEdit.Visibility =
   Visibility.Collapsed;
3398.                           propBLabelEdit.Visibility =
   Visibility.Collapsed;
3399.                           labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3400.                       }
3401.                       if (properties.Where(item => item.name ==
   editComboBox.SelectedItem.ToString()).First().type == "Retangular")
3402.                       {
3403.                           propSecComboBoxEdit.IsEnabled = true;
3404.                           propSecComboBoxEdit.SelectedIndex = 2;
3405.
3406.                           propNoneSaveEdit.IsEnabled = false;
3407.                           propNoneDeleteEdit.IsEnabled = false;
3408.
3409.                           propRetCalcEdit.IsEnabled = true;
3410.                           propRetSaveEdit.IsEnabled = true;
3411.                           propRetDeleteEdit.IsEnabled = true;
3412.                           propHEdit.IsEnabled = true;
```

```
3413.                           propBEdit.IsEnabled = true;
3414.                           propRetSepEdit.IsEnabled = true;
3415.
3416.                           propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3417.                           propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3418.                           propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3419.                           labelWrongPropNoneEdit.Visibility =
   Visibility.Collapsed;
3420.
3421.                           propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3422.                           propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3423.                           propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3424.                           propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3425.                           propDiamEdit.Visibility =
   Visibility.Collapsed;
3426.                           propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3427.                           labelWrongPropDiamEdit.Visibility =
   Visibility.Collapsed;
3428.
3429.                           propRetCalcEdit.Visibility =
   Visibility.Visible;
3430.                           propRetSaveEdit.Visibility =
   Visibility.Visible;
3431.                           propRetDeleteEdit.Visibility =
   Visibility.Visible;
3432.                           propRetSepEdit.Visibility =
   Visibility.Visible;
3433.                           propHEdit.Visibility =
   Visibility.Visible;
3434.                           propBEdit.Visibility =
   Visibility.Visible;
3435.                           propHLabelEdit.Visibility =
   Visibility.Visible;
3436.                           propBLabelEdit.Visibility =
   Visibility.Visible;
3437.                           labelWrongPropRetEdit.Visibility =
   Visibility.Collapsed;
3438.                       }
3439.
3440.                       propDiamEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().d.ToString();
3441.                       propHEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().h.ToString();
3442.                       propBEdit.Text = properties.Where(item =>
   item.name ==
   editComboBox.SelectedItem.ToString()).First().b.ToString();
3443.
3444.                       propNameEdit.IsEnabled = true;
3445.
3446.                       propAEdit.IsEnabled = true;
3447.                       propIxEdit.IsEnabled = true;
```

```
3448.                         propIyEdit.IsEnabled = true;
3449.                         propIzEdit.IsEnabled = true;
3450.
3451.                         editComboBox.IsEnabled = true;
3452.                     }
3453.                     else
3454.                     {
3455.
3456.                     }
3457.                 }
3458.
3459.                 public void
   PropSecComboBoxEditSelectionChanged(object sender,
   SelectionChangedEventArgs e)
3460.                 {
3461.                     if (!finishedLoading) return;
3462.
3463.                     if (propSecComboBoxEdit.SelectedIndex == 0)
3464.                     {
3465.                         propNoneSaveEdit.IsEnabled = true;
3466.                         propNoneDeleteEdit.IsEnabled = true;
3467.
3468.                         propNoneSaveEdit.Visibility =
   Visibility.Visible;
3469.                         propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3470.                         propNoneSepEdit.Visibility =
   Visibility.Visible;
3471.
3472.                         propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3473.                         propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3474.                         propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3475.                         propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3476.                         propDiamEdit.Visibility =
   Visibility.Collapsed;
3477.                         propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3478.
3479.                         propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3480.                         propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3481.                         propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3482.                         propRetSepEdit.Visibility =
   Visibility.Collapsed;
3483.                         propHEdit.Visibility = Visibility.Collapsed;
3484.                         propBEdit.Visibility = Visibility.Collapsed;
3485.                         propHLabelEdit.Visibility =
   Visibility.Collapsed;
3486.                         propBLabelEdit.Visibility =
   Visibility.Collapsed;
3487.                     }
3488.
3489.                     if (propSecComboBoxEdit.SelectedIndex == 1)
3490.                     {
3491.                         propNoneSaveEdit.IsEnabled = false;
```

```
3492.                       propNoneDeleteEdit.IsEnabled = false;
3493.
3494.                       propDiamCalcEdit.IsEnabled = true;
3495.                       propDiamSaveEdit.IsEnabled = true;
3496.                       propDiamDeleteEdit.IsEnabled = true;
3497.                       propDiamEdit.IsEnabled = true;
3498.                       propDiamSepEdit.IsEnabled = true;
3499.
3500.                       propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3501.                       propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
3502.                       propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3503.
3504.                       propDiamCalcEdit.Visibility =
   Visibility.Visible;
3505.                       propDiamSaveEdit.Visibility =
   Visibility.Visible;
3506.                       propDiamDeleteEdit.Visibility =
   Visibility.Visible;
3507.                       propDiamSepEdit.Visibility =
   Visibility.Visible;
3508.                       propDiamEdit.Visibility =
   Visibility.Visible;
3509.                       propDiamLabelEdit.Visibility =
   Visibility.Visible;
3510.
3511.                       propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3512.                       propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3513.                       propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3514.                       propRetSepEdit.Visibility =
   Visibility.Collapsed;
3515.                       propHEdit.Visibility = Visibility.Collapsed;
3516.                       propBEdit.Visibility = Visibility.Collapsed;
3517.                       propHLabelEdit.Visibility =
   Visibility.Collapsed;
3518.                       propBLabelEdit.Visibility =
   Visibility.Collapsed;
3519.                   }
3520.
3521.                   if (propSecComboBoxEdit.SelectedIndex == 2)
3522.                   {
3523.                       propNoneSaveEdit.IsEnabled = false;
3524.                       propNoneDeleteEdit.IsEnabled = false;
3525.
3526.                       propRetCalcEdit.IsEnabled = true;
3527.                       propRetSaveEdit.IsEnabled = true;
3528.                       propRetDeleteEdit.IsEnabled = true;
3529.                       propHEdit.IsEnabled = true;
3530.                       propBEdit.IsEnabled = true;
3531.                       propRetSepEdit.IsEnabled = true;
3532.
3533.                       propNoneSaveEdit.Visibility =
   Visibility.Collapsed;
3534.                       propNoneDeleteEdit.Visibility =
   Visibility.Collapsed;
```

```
3535.                         propNoneSepEdit.Visibility =
   Visibility.Collapsed;
3536.
3537.                         propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3538.                         propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3539.                         propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3540.                         propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3541.                         propDiamEdit.Visibility =
   Visibility.Collapsed;
3542.                         propDiamLabelEdit.Visibility =
   Visibility.Collapsed;
3543.
3544.                         propRetCalcEdit.Visibility =
   Visibility.Visible;
3545.                         propRetSaveEdit.Visibility =
   Visibility.Visible;
3546.                         propRetDeleteEdit.Visibility =
   Visibility.Visible;
3547.                         propRetSepEdit.Visibility =
   Visibility.Visible;
3548.                         propHEdit.Visibility = Visibility.Visible;
3549.                         propBEdit.Visibility = Visibility.Visible;
3550.                         propHLabelEdit.Visibility =
   Visibility.Visible;
3551.                         propBLabelEdit.Visibility =
   Visibility.Visible;
3552.                     }
3553.                 }
3554.
3555.                 public void DeleteEditProperty(object sender,
   RoutedEventArgs e)
3556.                 {
3557.                     try
3558.                     {
3559.                         properties.Remove(properties.Where(item =>
   item.name == editComboBox.SelectedItem.ToString()).First());
3560.
3561.                         editComboBox.Items.Clear();
3562.
3563.                         for (int i = 0; i < properties.Count; i++)
3564.                         {
3565.
   editComboBox.Items.Add(properties[i].name);
3566.                         }
3567.
3568.                         propNameEdit.Text = "";
3569.                         propAEdit.Text = "";
3570.                         propIxEdit.Text = "";
3571.                         propIyEdit.Text = "";
3572.                         propIzEdit.Text = "";
3573.
3574.                         propNameEdit.IsEnabled = false;
3575.
3576.                         propAEdit.IsEnabled = false;
3577.                         propIxEdit.IsEnabled = false;
3578.                         propIyEdit.IsEnabled = false;
3579.                         propIzEdit.IsEnabled = false;
```

```
3580.
3581.                       editComboBox.IsEnabled = true;
3582.
3583.                       propSecComboBoxEdit.IsEnabled = false;
3584.                       propSecComboBoxEdit.SelectedIndex = 0;
3585.
3586.                       propNoneSaveEdit.IsEnabled = false;
3587.                       propNoneDeleteEdit.IsEnabled = false;
3588.
3589.                       propNoneSaveEdit.Visibility =
   Visibility.Visible;
3590.                       propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3591.                       propNoneSepEdit.Visibility =
   Visibility.Visible;
3592.
3593.                       propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3594.                       propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3595.                       propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3596.                       propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3597.                       propDiamEdit.Visibility =
   Visibility.Collapsed;
3598.
3599.                       propRetCalcEdit.Visibility =
   Visibility.Collapsed;
3600.                       propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3601.                       propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3602.                       propRetSep.Visibility =
   Visibility.Collapsed;
3603.                       propHEdit.Visibility = Visibility.Collapsed;
3604.                       propBEdit.Visibility = Visibility.Collapsed;
3605.                   }
3606.                   catch
3607.                   {
3608.
3609.                   }
3610.               }
3611.
3612.               public void CalculateCircularEditProperty(object
   sender, RoutedEventArgs e)
3613.               {
3614.                   try
3615.                   {
3616.                       double A = Math.PI *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 2);
3617.                       double Ix = (Math.PI / 4) *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 4);
3618.                       double Iy = Ix;
3619.                       double Iz = (Math.PI / 2) *
   Math.Pow(double.Parse(propDiamEdit.Text) / 2, 4);
3620.
3621.                       propAEdit.Text = A.ToString();
3622.                       propIxEdit.Text = Ix.ToString();
3623.                       propIyEdit.Text = Iy.ToString();
3624.                       propIzEdit.Text = Iz.ToString();
```

```
3625.                     }
3626.                     catch
3627.                     {
3628.                         if (propSecComboBoxEdit.SelectedIndex == 0)
3629.                         {
3630.                             labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3631.                             labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3632.                         }
3633.                         if (propSecComboBoxEdit.SelectedIndex == 1)
3634.                         {
3635.                             labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3636.                             labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3637.                         }
3638.                         if (propSecComboBoxEdit.SelectedIndex == 2)
3639.                         {
3640.                             labelWrongPropRetEdit.Content =
   "Property Wrong!";
3641.                             labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3642.                         }
3643.                     }
3644.                 }
3645.
3646.                 public void CalculateRetangularEditProperty(object
   sender, RoutedEventArgs e)
3647.                 {
3648.                     try
3649.                     {
3650.                         double A = double.Parse(propHEdit.Text) *
   double.Parse(propBEdit.Text);
3651.                         double Ix = double.Parse(propBEdit.Text) *
   Math.Pow(double.Parse(propHEdit.Text), 3) / 12;
3652.                         double Iy = double.Parse(propHEdit.Text) *
   Math.Pow(double.Parse(propBEdit.Text), 3) / 12;
3653.                         double Iz = Ix + Iy;
3654.
3655.                         propAEdit.Text = A.ToString();
3656.                         propIxEdit.Text = Ix.ToString();
3657.                         propIyEdit.Text = Iy.ToString();
3658.                         propIzEdit.Text = Iz.ToString();
3659.                     }
3660.                     catch
3661.                     {
3662.                         if (propSecComboBoxEdit.SelectedIndex == 0)
3663.                         {
3664.                             labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3665.                             labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3666.                         }
3667.                         if (propSecComboBoxEdit.SelectedIndex == 1)
3668.                         {
3669.                             labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3670.                             labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3671.                         }
```

```
3672.                         if (propSecComboBoxEdit.SelectedIndex == 2)
3673.                         {
3674.                             labelWrongPropRetEdit.Content =
   "Property Wrong!";
3675.                             labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3676.                         }
3677.                     }
3678.                 }
3679.
3680.                 public void SaveEditProperty(object sender,
   RoutedEventArgs e)
3681.                 {
3682.                     if (propNameEdit.Text != "" && propAEdit.Text !=
   "" && propIxEdit.Text != "" && propIyEdit.Text != "" &&
   propIzEdit.Text != "")
3683.                     {
3684.                         try
3685.                         {
3686.
3687.                             string propName =
   editComboBox.SelectedItem.ToString();
3688.
3689.                             if (propName != propNameEdit.Text &&
   properties.Where(item => item.name ==
   propNameEdit.Text).ToList().Count != 0)
3690.                             {
3691.                                 //labelWrongMatPEdit.Content =
   "Wrong Properties!";
3692.                                 //labelWrongMatPEdit.Visibility =
   Visibility.Visible;
3693.                             }
3694.                             else
3695.                             {
3696.                                 properties.Where(item => item.name
   == propName).First().A = double.Parse(propAEdit.Text);
3697.                                 properties.Where(item => item.name
   == propName).First().Ix = double.Parse(propIxEdit.Text);
3698.                                 properties.Where(item => item.name
   == propName).First().Iy = double.Parse(propIyEdit.Text);
3699.                                 properties.Where(item => item.name
   == propName).First().Iz = double.Parse(propIzEdit.Text);
3700.                                 properties.Where(item => item.name
   == propName).First().d = double.Parse(propDiamEdit.Text);
3701.                                 properties.Where(item => item.name
   == propName).First().h = double.Parse(propHEdit.Text);
3702.                                 properties.Where(item => item.name
   == propName).First().b = double.Parse(propBEdit.Text);
3703.                                 if
   (propSecComboBoxEdit.SelectedIndex == 0) properties.Where(item =>
   item.name == propName).First().type = "None";
3704.                                 if
   (propSecComboBoxEdit.SelectedIndex == 1) properties.Where(item =>
   item.name == propName).First().type = "Circular";
3705.                                 if
   (propSecComboBoxEdit.SelectedIndex == 2) properties.Where(item =>
   item.name == propName).First().type = "Retangular";
3706.                                 properties.Where(item => item.name
   == propName).First().name = propNameEdit.Text;
3707.
```

```
3708.                                   for (int i = 0; i < members.Count;
   i++)
3709.                                   {
3710.                                       if (members[i].section ==
   editComboBox.SelectedItem.ToString()) members[i].section =
   propNameEdit.Text;
3711.                                   }
3712.
3713.                                   editComboBox.Items.Clear();
3714.
3715.                                   for (int i = 0; i <
   properties.Count; i++)
3716.                                   {
3717.
   editComboBox.Items.Add(properties[i].name);
3718.                                   }
3719.
3720.                                   propNameEdit.Text = "";
3721.                                   propAEdit.Text = "";
3722.                                   propIxEdit.Text = "";
3723.                                   propIyEdit.Text = "";
3724.                                   propIzEdit.Text = "";
3725.
3726.                                   propNameEdit.IsEnabled = false;
3727.
3728.                                   propAEdit.IsEnabled = false;
3729.                                   propIxEdit.IsEnabled = false;
3730.                                   propIyEdit.IsEnabled = false;
3731.                                   propIzEdit.IsEnabled = false;
3732.
3733.                                   editComboBox.IsEnabled = true;
3734.
3735.                                   propSecComboBoxEdit.IsEnabled =
   false;
3736.                                   propSecComboBoxEdit.SelectedIndex =
   0;
3737.
3738.                                   propNoneSaveEdit.IsEnabled = false;
3739.                                   propNoneDeleteEdit.IsEnabled =
   false;
3740.
3741.                                   propNoneSaveEdit.Visibility =
   Visibility.Visible;
3742.                                   propNoneDeleteEdit.Visibility =
   Visibility.Visible;
3743.                                   propNoneSepEdit.Visibility =
   Visibility.Visible;
3744.
3745.                                   propDiamCalcEdit.Visibility =
   Visibility.Collapsed;
3746.                                   propDiamSaveEdit.Visibility =
   Visibility.Collapsed;
3747.                                   propDiamDeleteEdit.Visibility =
   Visibility.Collapsed;
3748.                                   propDiamSepEdit.Visibility =
   Visibility.Collapsed;
3749.                                   propDiamEdit.Visibility =
   Visibility.Collapsed;
3750.
3751.                                   propRetCalcEdit.Visibility =
   Visibility.Collapsed;
```

```
3752.                                   propRetSaveEdit.Visibility =
   Visibility.Collapsed;
3753.                                   propRetDeleteEdit.Visibility =
   Visibility.Collapsed;
3754.                                   propRetSep.Visibility =
   Visibility.Collapsed;
3755.                                   propHEdit.Visibility =
   Visibility.Collapsed;
3756.                                   propBEdit.Visibility =
   Visibility.Collapsed;
3757.
3758.                                   labelWrongPropNoneEdit.Content =
   "Property Saved!";
3759.                                   labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3760.                               }
3761.                           }
3762.                           catch
3763.                           {
3764.                               if (propSecComboBoxEdit.SelectedIndex ==
   0)
3765.                               {
3766.                                   labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3767.                                   labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3768.                               }
3769.                               if (propSecComboBoxEdit.SelectedIndex ==
   1)
3770.                               {
3771.                                   labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3772.                                   labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3773.                               }
3774.                               if (propSecComboBoxEdit.SelectedIndex ==
   2)
3775.                               {
3776.                                   labelWrongPropRetEdit.Content =
   "Property Wrong!";
3777.                                   labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3778.                               }
3779.                           }
3780.                       }
3781.                       else
3782.                       {
3783.                           if (propSecComboBoxEdit.SelectedIndex == 0)
3784.                           {
3785.                               labelWrongPropNoneEdit.Content =
   "Property Wrong!";
3786.                               labelWrongPropNoneEdit.Visibility =
   Visibility.Visible;
3787.                           }
3788.                           if (propSecComboBoxEdit.SelectedIndex == 1)
3789.                           {
3790.                               labelWrongPropDiamEdit.Content =
   "Property Wrong!";
3791.                               labelWrongPropDiamEdit.Visibility =
   Visibility.Visible;
3792.                           }
```

```
3793.                       if (propSecComboBoxEdit.SelectedIndex == 2)
3794.                       {
3795.                           labelWrongPropRetEdit.Content =
   "Property Wrong!";
3796.                           labelWrongPropRetEdit.Visibility =
   Visibility.Visible;
3797.                       }
3798.                   }
3799.               }
3800.
3801.               public void Enable3DOrbit(object sender,
   RoutedEventArgs e)
3802.               {
3803.                   SendStringToInterface(string.Format("Orbit
   Status&True"));
3804.               }
3805.
3806.               public void Disable3DOrbit(object sender,
   RoutedEventArgs e)
3807.               {
3808.                   SendStringToInterface(string.Format("Orbit
   Status&False"));
3809.               }
3810.
3811.               public void EnableNodes(object sender,
   RoutedEventArgs e)
3812.               {
3813.                   SendStringToInterface(string.Format("Node
   Status&True"));
3814.               }
3815.
3816.               public void DisableNodes(object sender,
   RoutedEventArgs e)
3817.               {
3818.                   SendStringToInterface(string.Format("Node
   Status&False"));
3819.               }
3820.
3821.               public void EnableMembers(object sender,
   RoutedEventArgs e)
3822.               {
3823.                   SendStringToInterface(string.Format("Member
   Status&True"));
3824.               }
3825.
3826.               public void DisableMembers(object sender,
   RoutedEventArgs e)
3827.               {
3828.                   SendStringToInterface(string.Format("Member
   Status&False"));
3829.               }
3830.
3831.               public void ResetView(object sender, RoutedEventArgs
   e)
3832.               {
3833.                   SendStringToInterface(string.Format("Reset
   View"));
3834.               }
3835.
3836.               public void RunSimulationClick(object sender,
   RoutedEventArgs e)
```

```
3837.                {
3838.
   SendStringToInterface(string.Format("Mode&Mounting Simulation"));
3839.                    MountModels();
3840.
   SendStringToInterface(string.Format("Mode&Running Simulation"));
3841.                    RunSimulations();
3842.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   endDay));
3843.
3844.                    simulationSuccess = true;
3845.                    EnableSimulationSuccessfullUI();
3846.                }
3847.
3848.                public void UpdateUISimulationClick(object sender,
   RoutedEventArgs e)
3849.                {
3850.                    double diff = endDay - startDay;
3851.
3852.                    double value = double.Parse(SimShowDay.Text) -
   startDay;
3853.
3854.                    double percent = value / diff;
3855.
3856.                    percent = percent * 20d;
3857.
3858.                    SimSlider.Value = percent;
3859.
3860.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   SimShowDay.Text));
3861.                }
3862.
3863.                public void EnableSimulationSuccessfullUI()
3864.                {
3865.                    SimSep1.Visibility = Visibility.Visible;
3866.                    SimLabelDay.Visibility = Visibility.Visible;
3867.                    SimShowDay.Visibility = Visibility.Visible;
3868.                    SimUpdate.Visibility = Visibility.Visible;
3869.                    SimSep2.Visibility = Visibility.Visible;
3870.                    SimSlider.Visibility = Visibility.Visible;
3871.                    SimLabelObject.Visibility = Visibility.Visible;
3872.                    SimPlot.Visibility = Visibility.Visible;
3873.                    SimulationElementResults.Visibility =
   Visibility.Visible;
3874.
3875.                    SimShowDay.Text = endDay.ToString();
3876.                    SimulationElementResults.Text = "No object
   selected";
3877.                    SimPlot.IsEnabled = false;
3878.
3879.                    double diff = endDay - startDay;
3880.
3881.                    double value = double.Parse(SimShowDay.Text) -
   startDay;
3882.
3883.                    double percent = value / diff;
3884.
3885.                    percent = percent * 20d;
3886.
```

```
3887.                     SimSlider.Value = percent;
3888.                 }
3889.
3890.                 private void SimSliderValue(object sender,
   RoutedPropertyChangedEventArgs<double> e)
3891.                 {
3892.                     if (!finishedLoading) return;
3893.
3894.                     int diff = endDay - startDay;
3895.
3896.                     double value = SimSlider.Value / 20d;
3897.
3898.                     SimShowDay.Text = Convert.ToInt32(startDay +
   diff * value).ToString();
3899.
3900.
   SendStringToInterface(string.Format("Mode&Simulation Successful&{0}",
   SimShowDay.Text));
3901.                 }
3902.
3903.                 public void MountPlotForNode(object sender,
   RoutedEventArgs e)
3904.                 {
3905.                     Plot plot = new Plot();
3906.
3907.                     plot.Show();
3908.                 }
3909.
3910.                 public void About(object sender, RoutedEventArgs e)
3911.                 {
3912.                     About about = new About();
3913.
3914.                     about.Show();
3915.                 }
3916.
3917.                 public void SendStringToInterface(string text)
3918.                 {
3919.                     if (!finishedLoading) return;
3920.
3921.                     text = "%%&" + text + "&%%";
3922.
3923.                     ASCIIEncoding ASEn = new ASCIIEncoding();
3924.
3925.                     byte[] byteArray = ASEn.GetBytes(text);
3926.
3927.                     SendData(byteArray);
3928.                 }
3929.
3930.                 public void Save(object sender, RoutedEventArgs e)
3931.                 {
3932.                     SaveFileDialog saveFileDialog1 = new
   SaveFileDialog();
3933.                     saveFileDialog1.Filter = "Structure
   File|*.struct";
3934.                     saveFileDialog1.Title = "Save a Structure File";
3935.                     saveFileDialog1.ShowDialog();
3936.
3937.                     if (saveFileDialog1.FileName != "")
3938.                     {
3939.                         BinaryFormatter bf = new BinaryFormatter();
```

```
3940.                         FileStream file =
   File.Open(saveFileDialog1.FileName, FileMode.OpenOrCreate);
3941.
3942.                         SaveFile save = new SaveFile();
3943.                         save.nodes = nodes;
3944.                         save.members = members;
3945.                         save.materials = materials;
3946.                         save.properties = properties;
3947.                         save.nodalForces = nodalForces;
3948.                         save.memberForces = memberForces;
3949.                         save.startDay = startDay;
3950.                         save.endDay = endDay;
3951.
3952.                         bf.Serialize(file, save);
3953.                         file.Close();
3954.                     }
3955.                 }
3956.
3957.                 public void Load(object sender, RoutedEventArgs e)
3958.                 {
3959.                     OpenFileDialog openFileDialog1 = new
   OpenFileDialog();
3960.                     openFileDialog1.Filter = "Structure
   File|*.struct";
3961.                     openFileDialog1.Title = "Open a Structure File";
3962.                     openFileDialog1.ShowDialog();
3963.
3964.                     if (openFileDialog1.FileName != "")
3965.                     {
3966.                         BinaryFormatter bf = new BinaryFormatter();
3967.                         FileStream file =
   File.Open(openFileDialog1.FileName, FileMode.Open);
3968.
3969.                         SaveFile save =
   (SaveFile)bf.Deserialize(file);
3970.
3971.                         nodes = save.nodes;
3972.                         members = save.members;
3973.                         materials = save.materials;
3974.                         properties = save.properties;
3975.                         nodalForces = save.nodalForces;
3976.                         memberForces = save.memberForces;
3977.                         startDay = save.startDay;
3978.                         endDay = save.endDay;
3979.
3980.                         MassiveUpdate3DUI();
3981.
3982.                         file.Close();
3983.                     }
3984.                 }
3985.
3986.                 public void MassiveUpdate3DUI()
3987.                 {
3988.                     string command = "Open Massive Structure
   Changes";
3989.
3990.                     for (int i = 0; i < nodes.Count; i++)
3991.                     {
3992.                         command +=
   string.Format("&N&{0}&{1:0.00}&{2:0.00}&{3:0.00}&{4}",
   nodes[i].number, nodes[i].position.x, nodes[i].position.y,
```

```
   nodes[i].position.z, nodes[i].restriction.x1 ||
   nodes[i].restriction.y1 || nodes[i].restriction.z1 ||
   nodes[i].restriction.x2 || nodes[i].restriction.y2 ||
   nodes[i].restriction.z2);
3993.                   }
3994.
3995.                   for (int i = 0; i < members.Count; i++)
3996.                   {
3997.                       command +=
   string.Format("&M&{0}&{1}&{2}&{3}", members[i].n0, members[i].n1,
   members[i].startDay, members[i].endDay);
3998.                   }
3999.
4000.                   SendStringToInterface(command);
4001.               }
4002.           }
4003.
4004.           [System.Serializable]
4005.           public class SaveFile
4006.           {
4007.               public List<Nodes> nodes = new List<Nodes>();
4008.               public List<Member> members = new List<Member>();
4009.
4010.               public List<Material> materials = new
   List<Material>();
4011.               public List<Property> properties = new
   List<Property>();
4012.
4013.               public List<NodalForces> nodalForces = new
   List<NodalForces>();
4014.               public List<MemberForces> memberForces = new
   List<MemberForces>();
4015.
4016.               public int startDay;
4017.               public int endDay;
4018.           }
4019.       }
```