UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

"Uma Ferramenta de Visualização para Redes Neurais Artificiais do Tipo Neocognitron"

ORIENTADOR: Prof. Dr. José Hiroki Saito
ALUNO: Bruno Zanetti

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.
(Campo de pesquisa: Visualização Científica)

São Carlos
Abril/2004

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

Z28fv

Zanetti, Bruno.
Uma ferramenta de visualização para redes neurais artificiais do tipo neocognitron / Bruno Zanetti. -- São Carlos : UFSCar, 2004.
76 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2004.

1. Redes neurais (computação). 2. Visualização. 3. Rede neural neocognitron. 4. Sistemas de partículas. I. Título.

CDD: 006.32 (21ª)

Agradecimentos

Ao Prof. Dr. José Hiroki Saito, orientador, pela oportunidade de desenvolver este trabalho.

À CAPES, pela bolsa concedida.

Aos meus amigos e família, por me tolerar.

Sumário

1 Introdução 1
2 Redes Neurais 3
2.1 Redes Neurais Biológicas[17] 3
2.1.1 O neurônio 3
2.1.2 As sinapses 4
2.1.3 O cérebro 5
2.2 Redes Neurais Artificiais 7
2.2.1 Um modelo de neurônio artificial 7
2.2.2 As sinapses do modelo artificial 8
2.2.3 O cérebro e a rede neural artificial 9
2.2.4 Diferenças entre estruturas biológicas e Redes Neurais Artificiais [7] 9
2.3 Modelos de Redes Neurais 11
2.4 O Neocognitron [10,11] 13
2.4.1 Arquitetura do Neocognitron 15
2.4.2 Processamento de dados no neocognitron 17
2.4.2.1 Processamento nas células-S 17
2.4.2.2 Processamento nas células-C 20
2.4.2.3 Treinamento dos pesos das células-S 21
3 Visualização Científica 23
3.1 Introdução 23
3.2 Fundamentos 23
3.2.1 O sistema visual humano 24
3.2.1.1 Estruturas Visuais e Princípios Gestalt 24
3.2.2 Processo de visualização 26
3.3 Aplicações 27
3.3.1 Análise de dados meteorológicos 27
3.3.2 Imagens médicas 28
3.3.3 Visualização de documentos 29
4 Computação Gráfica 30
4.1 Definição 30
4.2 Bibliotecas Gráficas 30
4.2.1 Java 3D 31
4.2.2 Direct3D 32
4.2.2.1 Vantagens 32
4.2.2.2 Desvantagens 32
4.2.3 OpenGL 33
4.2.3.1 Vantagens 33
4.2.3.2 Desvantagens 34
4.3 Sistemas de Partículas 34
5 Trabalho Realizado 36
5.1 Objetivos 36
5.2 Conceitos utilizados 36
5.3 Sobre a Programação 36
5.4 A biblioteca 37
5.4.1 Simulador 37
5.4.1.1 O nó (CNeuralNode) 38
5.4.1.2 O Plano (CNeuralPlane) 41

5.4.1.3 A Camada (CNeuralLayer) 47
5.4.1.4 O Nível (CNeuralLevel) 50
5.4.1.5 A Rede (CNeocognitron) 51
5.4.2 Visualizador 52
5.4.2.1 CGraphicObject 53
5.4.2.2 CVisualNode 53
5.4.2.3 CVisualPlane 56
5.4.2.4 CVisualLayer 57
5.4.2.5 CVisuaLevel 59
5.4.2.6 CVisualNeocognitron 60
5.4.2.7 Classes de Janelas de Diálogo 60
5.5 A Ferramenta 61
5.6 Resultados Obtidos 63
5.6.1 Criação da rede 63
5.6.2 Treinamento da Rede 64
5.6.3 Teste da Rede 68
6 Conclusão e Propostas de Trabalhos Futuros 73
6.1 Conclusão 73
6.2 Trabalhos Futuros 73
7 Referências 75

Lista de Figuras

Figura 1 – Neurônio [17] 3
Figura 2 - Tipos de Sinapses. (a) Sinapse Axodendrítica (b) Sinapse Axossomática (c) Sinapse Axoaxônica [17] 4
Figura 3 - Divisões citoarquitetônicas do cérebro, segundo Brodmann. Áreas divididas através de símbolos e numeradas. (a) Vista Lateral (b) Vista Central [16] 6
Figura 4 - Estrutura do Nó [10] 7
Figura 5 - Estrutura hierárquica do Neocognitron. Imagem gerada pelo visualizador implementado neste trabalho 15
Figura 6 - Planos de uma Camada-S e o Plano inibidor V_C dessa mesma camada. Imagem gerada pelo visualizador implementado neste trabalho. 18
Figura 7- Cálculo da Saída das Células-S[10] 19
Figura 8- Colunas-S[11] 21
Figura 9 - Exemplos de ferramentas visuais. (a) Multiplicação (b) Mapa de navegação [3] 23
Figura 10 - Ciclo de visualização em aplicações científicas [2] 26
Figura 11 - Visualização de um modelo numérico de nuvem [20] 28
Figura 12 - Imageamento médico [4] 28
Figura 13 - Visualização de documentos pessoais. O tamanho das áreas dos assuntos indica sua importância ao pesquisador, enquanto que a proximidade entre as áreas indica proximidade dos assuntos quando pesquisados por esse pesquisador [18] 29
Figura 14 - Estrutura do sistema de fontes e pesos do nó 38
Figura 15 - Estrutura SLink 38
Figura 16 - Diagrama de classe da Classe CNeuralNode e estruturas auxiliares 39
Figura 17 - Estrutura hierárquica dos nós do neocognitron 40
Figura 18 - Estrutura SPlaneLink 41
Figura 19 - Diagrama de classe da Classe CNeuralPlane, estruturas auxiliares e relacionamento com a classe CNeuralNode 42
Figura 20 - Hierarquia de classes referentes aos planos S e V_C 44
Figura 21 - Hierarquia de classes referentes aos planos C e V_S 45
Figura 22 - Hierarquia de classes referentes aos planos auxiliares Retina e Redutor 46
Figura 23 - Estrutura connections 47
Figura 24 - Estrutura de resumo de plano 48
Figura 25 - Diagrama de classes da classe CNeuralLayer e estruturas auxiliares 48
Figura 26 - Hierarquia de classes das camadas derivadas 49
Figura 27 - Diagrama de classe da classe CNeuralLevel 51
Figura 28 - Diagrama de classe da classe CNeocognitron 52
Figura 29 - Hierarquia resumida das classes visuais 54
Figura 30 - Janela de diálogo da classe CVisualNode 54
Figura 31 - Janela de diálogo da classe CVisualCCell 55
Figura 32 - Janela de diálogo da classe CVisualSCell 56
Figura 33 - Janela de diálogo da classe CNeuralPlane 57
Figura 34 - Janela de diálogo da classe CVisualLayer 58
Figura 35 - Janela de diálogo da classe CVisualLevel 60
Figura 36 - Hierarquia de classes das classes de janelas de diálogo 61
Figura 37 - Imagem da rede criada para o teste 64
Figura 38 - Padrões de treinamento e teste 64

Figura 39 - Rede após treinamento com o primeiro padrão.....65
Figura 40 - Saída da rede após treinamento com o primeiro padrão 66
Figura 41 - Rede após treinamento completo, exibindo o segundo padrão 67
Figura 42 - Saída da rede após treinamento completo, exibindo o segundo padrão.....67
Figura 43 - Rede reconhecendo o a imagem (g) do primeiro padrão.....68
Figura 44 - Saída do reconhecimento da imagem (g) do primeiro padrão.....69
Figura 45 - Rede reconhecendo a imagem (f) do primeiro padrão 69
Figura 46 -Saída do reconhecimento da imagem (f) do primeiro padrão 70
Figura 47 - Rede reconhecendo a imagem (f) do segundo padrão.....70
Figura 48 - Saída do reconhecimento da imagem (f) do segundo padrão.....71
Figura 49 - Rede reconhecendo a imagem (e) do segundo padrão 71
Figura 50 - Saída do reconhecimento da imagem (e) do segundo padrão 72

Lista de Tabelas

Tabela 1 - Aplicações das Redes Neurais mais conhecidas [23] 12
Tabela 2 - Aplicações das Redes Neurais mais conhecidas (cont.) [23] 13

Resumo

As redes neurais artificiais possuem a capacidade de aumentarem significativamente de tamanho, tornando-se mais difícil para o ser humano estudá-las a contento.

As redes do tipo neocognitron são um exemplo desse tipo de rede, que pode ampliar em muitas vezes seu tamanho inicial durante o treinamento.

Utilizando-se de técnicas de visualização, conseguiu-se uma ferramenta que possibilita ver o funcionamento de uma rede neocognitron, tanto em seu treinamento quanto na fase de classificação, permitindo assim uma maior compreensão do modelo como um todo, bem como experimentação dos diversos parâmetros do modelo.

Por trás dessa ferramenta está uma biblioteca de classes que pode ser utilizada separadamente para a implementação de novas formas de visualização ou novos simuladores de modelos de redes neurais diferentes.

A utilização dessa biblioteca reduz o tempo de prototipação de uma rede neural do tipo Neocognitron e, com algumas expansões, de outros tipos de redes neurais. Isso propicia um maior tempo para o pesquisador voltar sua atenção à arquitetura da rede em si.

No campo da educação, a ferramenta de visualização fornece um laboratório sobre a rede neural Neoconitron, podendo ainda ser adaptado para se conformar a novas expansões da biblioteca, fazendo com que o aluno aprenda mais sobre a arquitetura da rede ao vê-la em funcionamento e ao alterar seus parâmetros interativamente.

Abstract

Some Artificial Neural Networks may increase their size significantly with its use, making it difficult for a human being study it well.

The networks of the Neocognitron model are an example of this kind of network, increasing its initial size many times before the end of their training.

Using Scientific Visualization techniques, we obtained a tool that allows the user to see in details a neocognitron network actually working, be it during its training time or the recognition process. This allows a better comprehension of the whole model , as it helps in the construction of a network by allowing experimentation with the parameters of the model.

Powering this tool is a library of classes, that can be used separately to the implementation of new ways of visualization or in new simulators of other models of neural networks.

1 Introdução

Vivemos na era da informação, e, portanto, recebemos cada vez mais informações, que precisamos analisar e entender, e em um ritmo cada vez mais rápido. O progresso da velocidade dos processadores não auxiliou em nossa tarefa de entender os dados, muito pelo contrário. Agora possuímos a capacidade de realizar simulações, cada vez mais complexas e próximas da realidade, que nos geram cada vez mais dados, e os computadores ainda não são capazes de interpretá-las em nosso lugar.

A visualização surgiu como uma proposta para resolver parte desse problema, aproveitando-se da capacidade cada vez maior dos computadores de processarem dados numéricos. A proposta é simples: modificar a representação dos dados, de forma que possamos enxergá-los não como tabelas gigantescas, mas de forma gráfica, facilitando a compreensão dos dados pelos seres humanos.

A idéia é se aproveitar dos recursos naturais de nossos cérebros no reconhecimento de padrões e associação visual, para que possamos entender os dados de uma forma global, observando o conjunto como um todo, a procura de características que nos escapam quando olhamos para tabelas numéricas.

Este trabalho traz um modelo de visualização para redes neurais artificiais, mais especificamente as redes do tipo neocognitron, um tipo de rede que tende a gerar um grande número de nós durante seu treinamento, o que torna difícil o estudo da rede de forma global.

O principal objetivo deste trabalho é amenizar, talvez resolver, essas dificuldades através de uma ferramenta interativa de visualização em 3D. Essa ferramenta propiciará uma maior facilidade para a implementação e teste de redes neurais, tornando mais rápido o tempo de aprendizado e pesquisa através do método construtivista de ensino.

Este trabalho se divide em quatro partes principais, dispostas da seguinte forma:

- Capítulo 2 : Redes Neurais – oferece uma introdução às redes neurais, com os princípios básicos que levaram à criação desse modelo de processamento, as principais redes já criadas e, como é um dos focos do trabalho, uma descrição mais detalhada da rede neocognitron. Este capítulo tem por objetivo demonstrar as bases sobre as quais os dados

serão gerados para a criação da visualização, o que nos leva ao próximo capítulo;

- Capítulo 3: Visualização Científica – oferece uma demonstração da utilidade dessa disciplina, citando seus fundamentos e aplicações, além de algumas teorias que foram utilizadas no desenvolvimento do trabalho;
- Capítulo 4: Computação Gráfica – quando se pretende implementar uma visualização, deve-se saber transformar dados em gráficos. Nessa parte entra a computação gráfica. Neste capítulo é apresentada uma definição de computação gráfica e a comparação entre as principais bibliotecas gráficas comerciais atualmente utilizadas, além de uma apresentação dos motores de partículas, uma das bases do modelo de visualização implementado nesse trabalho.
- Capítulo 5: Trabalho Realizado – neste capítulo se encontra a documentação do trabalho realizado, que contém a documentação tanto da biblioteca de classes criada para a implementação da visualização, quanto da ferramenta desenvolvida para manipular essa biblioteca. Neste capítulo também são apresentados exemplos de uso da ferramenta, com o resultado da criação e teste de uma rede.
- Capítulo 6: Conclusão e Trabalhos Futuros – aqui se encontram as conclusões tiradas deste trabalho, bem como sugestões de trabalhos futuros, para o aprimoramento de todo o sistema (biblioteca de classes e ferramenta de manipulação)

As contribuições desse trabalho se dão no campo de pesquisa e ensino de redes neurais, pois uma maior facilidade de construção e prototipação desse tipo de sistema diminui o tempo de aprendizado e coleta de dados para pesquisa, uma vez que tanto o pesquisador quanto o aluno , ao utilizarem as bibliotecas e ferramentas criadas, não gastarão muito tempo com detalhes da implementação do sistema, e sim com a arquitetura da rede neural estudada.

2 Redes Neurais

As Redes Neurais Artificiais (RNAs) são uma arquitetura de processamento paralelo inspiradas no cérebro humano, buscando uma forma de processamento altamente paralelo e com a capacidade de aprendizado pela experiência. As RNAs são uma tentativa de resolver problemas complexos, como os de reconhecimento de padrões, de forma rápida e eficaz a partir de elementos de processamento muito simples.

Uma vez que sua inspiração provém da neurofisiologia, apresentamos aqui alguns conceitos elementares, úteis para o melhor entendimento do funcionamento das redes neurais.

2.1 Redes Neurais Biológicas[17]

As redes neurais biológicas são formadas por diversos componentes biológicos, i.e. diversos tipos de células e sua intercomunicação. O sistema nervoso humano é constituído basicamente dessas redes neurais, portanto analisaremos alguns dos elementos mais importantes no processamento de informações dentro desse sistema.

2.1.1 O neurônio

O neurônio é a unidade sinalizadora do cérebro, também considerada o elemento de processamento do mesmo. É uma célula especializada, com vários prolongamentos para a recepção de sinais, chamados dendritos, e um único para a emissão de sinais, chamado axônio. Entre essas duas estruturas se encontra o núcleo da célula, denominado soma. Uma esquematização da estrutura do neurônio é apresentada na Figura 1.

Figura 1 – Neurônio [17]

A natureza integradora do neurônio é conferida por sua membrana plasmática, cuja característica mais importante é a presença de diferentes tipos de canais iônicos,

capazes de filtrar seletivamente as passagens de íons para dentro e para fora do neurônio.

O sinal elétrico que o neurônio utiliza como unidade de informação é o impulso nervoso, um episódio muito rápido de inversão da polaridade da membrana, produzido pela abertura seletiva e consecutiva de canais Na+ e K+, causando um caudaloso fluxo iônico através da membrana, que se propaga ao longo do axônio, conduzindo o impulso nervoso de uma extremidade à outra do neurônio.

Os impulsos nervosos são sempre transportados no sentido Axônio→Dendritos, passando do axônio de um neurônio ao dendrito do próximo, através das sinapses.

2.1.2 As sinapses

A sinapse é a unidade processadora de sinais do cérebro. Trata-se de uma estrutura microscópica de contato entre um neurônio e outra célula, através da qual se dá a transmissão de mensagens entre as duas. Ao serem transmitidas, essas mensagens podem ser modificadas no processo de passagem de uma célula à outra, e é justamente nisso que reside a grande flexibilidade do cérebro. As sinapses podem ocorrer de diversas maneiras, como exemplificado na Figura 2, podendo ser (a) Axodendrítica, (b) Axossomática ou (c) Axoaxônica, assim chamadas pelo fato de se situarem mais próximas dos dendritos, soma ou axônio, respectivamente.

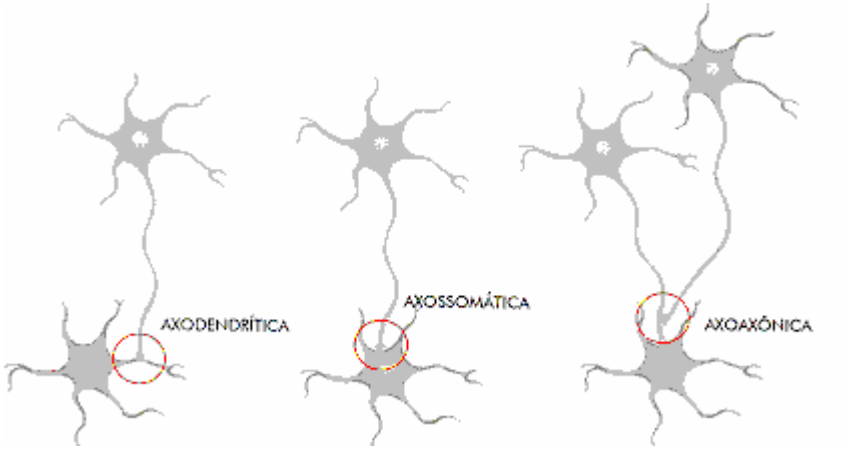


Figura 2 - Tipos de Sinapses. (a) Sinapse Axodendrítica (b) Sinapse Axossomática (c) Sinapse Axoaxônica [17]

Há dois tipos básicos de sinapses: as químicas e as elétricas. As sinapses elétricas – chamadas junções comunicantes – são sincronizadores celulares. Com

estrutura mais simples, transferem correntes iônicas e até mesmo pequenas moléculas entre células acopladas, fazendo uma transmissão rápida e de alta fidelidade, mas tendo, por outro lado, baixa capacidade de modulação. Esse tipo de sinapse é comum em recém nascidos e nas células cardíacas.

As sinapses químicas são verdadeiros *chips* biológicos porque podem modificar as mensagens que transmitem de acordo com inúmeras circunstâncias. Sua estrutura é especializada no armazenamento de substâncias neurotransmissoras e neuromoduladoras que, liberadas no exíguo espaço entre a membrana pré e a membrana pós-sináptica, provocam nesta última alterações de potencial elétrico que poderão influenciar o disparo de impulsos nervosos do neurônio pós-sináptico.

2.1.3 O cérebro

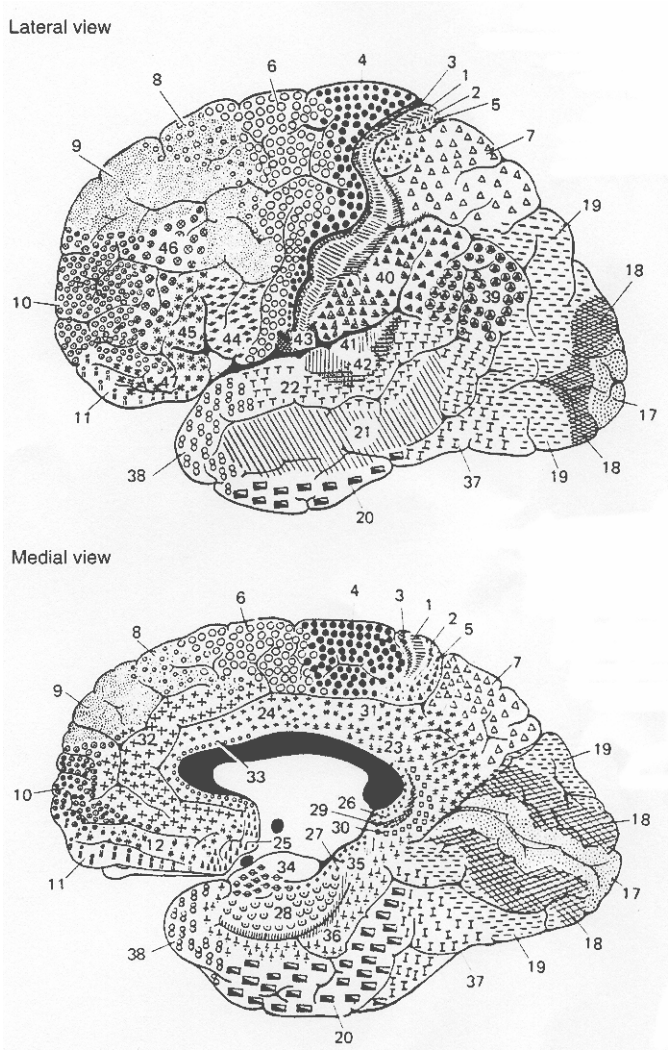
O cérebro humano possui um grande número de neurônios, ou elementos de processamento. Estimativas típicas são da ordem de 10-500 bilhões, com um número ainda maior de sinapses entre eles[7]. Um neurônio ou uma sinapse isolados teriam pouca utilidade, porque a capacidade de processamento de informações do sistema nervoso provém da integração entre as milhares de sinapses existentes em cada neurônio. Todas elas interagem: os efeitos excitatórios e inibitórios de cada uma delas sobre o potencial da membrana do neurônio pós-sináptico somam-se algebricamente, e o resultado dessa integração é que caracterizará a mensagem que emerge através do axônio do segundo neurônio, em direção a outros neurônios.

Há estimativas de que o cérebro é dividido em cerca de 1000 módulos, cada qual com cerca de 500 redes neurais. Cada rede possuiria cerca de 100000 neurônios. Como o axônio de cada neurônio se conecta a cerca de 100 (algumas vezes esse número sobe para a casa dos milhares) outros neurônios, podemos ter uma idéia da complexidade das redes formadas no cérebro, e de como, a partir de elementos que possuem cerca de 7 graus de magnitude de velocidade abaixo dos transistores atuais, o cérebro consegue resolver problemas complexos que um computador teria dificuldades em processar. É pensando nessa abordagem que modelamos esses problemas complexos para serem resolvidos através das RNAs[7].

Devido a esse número imenso de neurônios e a essa complexidade de interconexões ainda não é possível se ter um mapeamento completo de todas as redes do

cérebro. Por isso, quando tratamos do cérebro trabalhamos em termos de funções de alto nível, como que parte do cérebro é responsável pela visão ou audição.

Essa divisão do cérebro foi claramente mostrada no início do século 20 por Korbinian Brodmann, que dividiu o cérebro em áreas conforme a arquitetura das conexões dos neurônios. Ele dividiu o cérebro em cerca de 50 áreas citoarquitetônicas diferentes[16], como mostrado na Figura 3a (Vista Lateral) e na Figura 3b (Vista Central).



(a)

(b)

Figura 3 - Divisões citoarquitetônicas do cérebro, segundo Brodmann. Áreas divididas através de símbolos e numeradas. (a) Vista Lateral (b) Vista Central [16]

2.2 Redes Neurais Artificiais

Os trabalhos sobre RNAs foram motivados, desde sua concepção pelo reconhecimento de que o cérebro humano faz computações de uma maneira completamente diferente de um computador digital convencional[12]. Seguindo o princípio da analogia, A seguir é descrito um modelo pelo qual esse tipo de processamento biológico pode ser "imitado".

2.2.1 Um modelo de neurônio artificial

Descreveremos agora uma forma básica de neurônio utilizada na modelagem de redes neural, de forma a termos uma base de entendimento para as redes neurais específicas.

Essa forma básica de neurônio é esquematizada na Figura 4, com todos os seus componentes presentes.

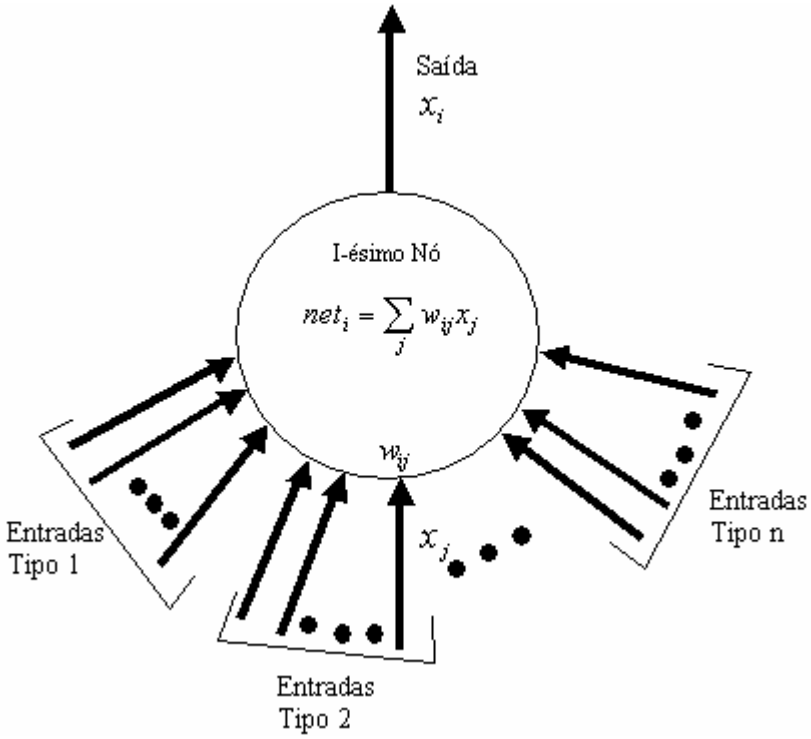


Figura 4 - Estrutura do Nó [10]

Cada nó em uma rede neural é numerado. Cada nó possui, de modo singular aos neurônios, diversas entradas, mas apenas uma saída, que, no entanto, pode ser conectada a diversos outros nós na rede.

A entrada que o i-ésimo nó recebe do j-ésimo nó é notada x_j (note que essa também é a notação da saída do nó j, assim como a saída do nó i será notada x_i). Cada conexão ao nó i é associada a uma força de conexão, chamada peso. O peso da conexão do nó j ao nó i é denominado w_{ij} [10].

As entradas de um nó são divididas em diversos tipos, o que indica que conexões de tipos diferentes podem possuir efeitos diferentes, como, por exemplo, conexões excitantes e inibidoras, que contribuem com valores positivos e negativos, respectivamente.

Cada nó possui um valor de entrada da rede, calculado como

$$net_i = \sum_j x_j w_{ij}$$

onde o índice j percorre todas as conexões ao nó i[10].

Uma vez que a entrada da rede é calculada, ela pode ser convertida para um valor de ativação, ou simplesmente ativação. Podemos escrever esse valor de ativação como

$$a_i(t) = F_i(a_i(t-1), net_i(t))^1$$

para denotar que a ativação é uma função explícita da entrada da rede[10].

Assim que a ativação do nó é calculada, podemos determinar o valor de saída aplicando uma função de saída:

$$x_i = f_i(a_i(t))$$

Dado que geralmente $a_i(t) = net_i(t)$, a função de saída é normalmente escrita como $x_i = f_i(net_i(t))$[10].

2.2.2 As sinapses do modelo artificial

No modelo de neurônio descrito acima, podemos considerar os pesos e as conexões com outros neurônios como sendo simplificações das sinapses que existem entre os neurônios biológicos.

[1] Quando tratamos de redes neurais, o tempo (t) é geralmente considerado uma variável medida em passos discretos, ao invés de uma variável contínua. Assim sendo, t-1 denota um passo de tempo anterior ao t atual.

Portanto, ao modificarmos a topologia das conexões, i.e. modificarmos qual neurônio se conecta a qual, ou os pesos dessas conexões, simulamos as funções das sinapses, com baixa complexidade.

2.2.3 O cérebro e a rede neural artificial

O cérebro humano é um sistema de processamento de informações altamente complexo, não linear e paralelizado.

Além disso, ele tem a capacidade de organizar seus componentes estruturais para poder realizar suas tarefas, permitindo que ele realize computações do tipo de reconhecimento de padrões, percepção e controle motor muitas vezes mais rápido que o computador digital mais rápido em existência[12].

As RNAs são sistemas deliberadamente construídos para fazer uso de alguns princípios organizacionais que são supostamente semelhantes aos do cérebro humano[1]. Elas são processadores extremamente paralelos e distribuídos, construídos a partir de unidades de processamento simples (neurônios), que possuem uma propensão natural a armazenar conhecimentos providos de sua experiência. As RNAs se assemelham ao cérebro em dois aspectos:

- Conhecimento é adquirido pela rede do seu ambiente, através de um processo de aprendizado;
- Forças de interconexão entre os neurônios (sinapses) são utilizadas para armazenar o conhecimento adquirido.

Portanto as RNAs derivam seu poder computacional de duas fontes, sua estrutura extremamente paralela e distribuída e sua capacidade de generalização proveniente do aprendizado[6]

2.2.4 Diferenças entre estruturas biológicas e Redes Neurais Artificiais [7]

Há diversas diferenças entre as estruturas biológicas e a representação ou implementação das RNAs. Enquanto mostramos essas diferenças, é importante manter em mente que os nós das RNAs geralmente são considerados análogos aos neurônios nas redes neurais do cérebro. Algumas dessas difereças se encontram enumeradas abaixo:

a. As conexões entre nós podem possuir pesos positivos ou negativos. Esses pesos correspondem a conexões excitatórias ou inibitórias entre neurônios, assim simplificando a complexidade inerente das sinapses.

b. Informações sobre o estado de ativação ou excitação de um nó são passadas a outros nós aos quais ele está conectado, através de um valor que corresponde a um nível de corrente direta (dc). Em redes neurais biológicas (RNBs), uma seqüência de pulsos através da sinapse é que carrega essa informação, e valores de ativação absolutos maiores correspondem a maiores taxas de pulsos, ou seja, a freqüência da corrente alternada (ac), ou a taxa de repetição do pulso é que geralmente indica o nível de ativação.

c. Há diversos tipos de neurônios nos sistemas biológicos. Uma RNA geralmente é implementada utilizando-se apenas um tipo de nó. Ocasionalmente, são utilizados dois ou três tipos de nós, e, conforme a tecnologia das RNAs evolui, redes mais sofisticadas podem fazer uso de diversos tipos de nós em suas implementações. Por outro lado, alguns estudos indicam que qualquer implementação pode ser feita com no máximo dois tipos de nós.

d. Neurônios nas RNBs possuem um ciclo individual de 10-100 milisegundos. A freqüência básica de clock de um processador comercial hoje em dia atinge a ordem de 1-2Ghz, que resultam em tempos de ciclo extremamente pequenos. Mas nesses casos, a velocidade de processamento é enganosa. Apesar de seu ciclo de processamento incrivelmente mais lento, o cérebro ainda é capaz de realizar tarefas em algumas ordens de magnitude mais rápido que os computadores, devido à sua arquitetura extremamente paralela.

e. Há uma diferença significativa entre o número de nós utilizados em uma RNA típica e o número de neurônios envolvidos em qualquer tarefa em uma RNB. RNAs típicas são implementadas com um número de nós que variam de uma dúzia a centenas de milhares de nós, enquanto que se considerarmos que cada um dos 1000 módulos do cérebro descrito acima contém cerca de 500 milhões de neurônios, e

que é quase certo de que uma tarefa simples envolva vários (talvez muitos) desses módulos, teremos uma idéia da quantidade de neurônios utilizados pelo cérebro. Para muitas aplicações práticas, não teríamos idéia de como utilizar efetivamente uma RNA com os 500 milhões de neurônios do primeiro módulo.

2.3 Modelos de Redes Neurais

Desde o início das pesquisas em redes neurais artificiais foram criadas redes neurais com os mais diversos propósitos, desde reconhecimento de padrões até processamento de sinais. Apresentadas a seguir, as Tabelas 1, 2 e 3 fazem um resumo dos principais modelos criados até então, com uma pequena descrição de suas características. A rede Neocognitron, uma rede neural artificial proposta a partir do modelo biológico de visão, será descrita com detalhes em separado na seção 2.4, pelo fato da mesma ser o objeto da ferramenta de visualização da presente dissertação.

Tabela 1 - Aplicações das Redes Neurais mais conhecidas [23]

Rede	Ano	Inventores/ Desenvolvedores	Aplicação Principal	Vantagens	Desvantagens
ADALINE/ MADALINE	1960	B. Widrow	Filtragem adaptativa de sinais, equalização adaptativa.	Rápida, fácil de implementar, pode ser implementada utilizando circuitos analógicos ou VLSI.	Assume relação linear entre entrada e saída. Somente classifica em espaços de classificação linearmente separáveis
Adaptative Resonance Theory	1983	G. Carpenter, S. Grossberg	Reconhecimento de padrões.	Capaz de aprender novos padrões, formar novas categorias de padrões e reter as categorias aprendidas.	A natureza dos exemplares de categoria podem mudar durante o treinamento.
Back-Propagating Perceptrons: Basic	1974 – 1986	P. J. Werbos, D. Parker, D. Rumelhart	Reconhecimento de padrões, filtragem de sinais, remoção de ruídos, segmentação de sinais/imagens, classificação, mapeamento, controle robótico adaptativo, compressão de dados.	Operação rápida. Boa em formar representações internas das características dos dados de entrada ou classificação e outras tarefas. Bem estudada. Muitas aplicações com êxito.	Longo período de aprendizado.
Recurrent	1987	Almeida, Pineda	Controle robótico, reconhecimento de voz, predição de elementos de seqüências.	Atualmente é a melhor rede para classificar e mapear informações variantes com o tempo.	Rede complexa, pode ser difícil de treinar e otimizar.

Tabela 2 - Aplicações das Redes Neurais mais conhecidas (cont.) [23]

Rede	Ano	Inventores/ Desenvolvedores	Aplicação Principal	Vantagens	Desvantagens
Recurrent	1987	Almeida, Pineda	Controle robótico, reconhecimento de voz, predição de elementos de seqüências.	Atualmente é a melhor rede para classificar e mapear informações variantes com o tempo.	Rede complexa, pode ser difícil de treinar e otimizar.
Time-Delay	1987	D. W. Tank, J. J. Hopfield	Reconhecimento de voz.	Desempenho equivalente aos melhores métodos convencionais, operação mais rápida.	Janela de atividade temporal representada fixa, responde desastradamente em diferenças na escala da entrada.
Functional-Link Network	1988	Y. H. Pão	Classificação, Mapeamento.	Somente duas camadas (entrada e saída) necessárias; treinamento rápido.	Não existe meio claro de identificar funções ou links funcionais.
Radial Basis Function Network	1987 – 1988	Múltiplos pesquisadores	Classificação, Mapeamento.	Rede com uma única camada oculta de neurônios deste tipo tem desempenho equivalente à rede de Retropropagação (back-propagation) básica com duas camadas ocultas.	Desconhecidas.
Back-Propagation of Utility Function Through Time	1974	P. J. Werbos	Maximizar índice de desempenho ou função de utilidade sobre o tempo, neurocontrole (robótica).	Abordagem neural mais abrangente para controle e/ou predição baseada em modelos.	Só pode ser utilizada após a identificação do modelo diferenciável, deve ser adaptada externamente se o modelo é dinâmico, e assume que o modelo é exato.
Bidirecional Associative Memory	1987	B. Kosko	Memória Hetero-associativa (endereçável através do conteúdo).	Arquitetura e dinâmica simples, com regras claras de aprendizado. Possui prova clara de estabilidade dinâmica.	Pouca capacidade de armazenamento, pouca precisão de recuperação da informação.
Boltzmann machine, Cauchy Machine	1984, 1986	G. Hinton, T. Sejnowski, D. Ackley; H. Szu	Reconhecimento de padrões (imagens, radar, sonar), otimização.	Capaz de formar representação ótima de características do padrão. Segue superfície de energia para obter otimização mínima.	Boltzmann Machine – tempo de aprendizado muito longo. As Cauchy Machines oferecem aprendizado mais rápido.

Tabela 3 - Aplicações das Redes Neurais mais conhecidas (cont.) [23]

Rede	Ano	Inventores/ Desenvolvedores	Aplicação Principal	Vantagens	Desvantagens
Boundary Contour System	1985	S. Grossberg, E. Mingolla	Processamento de imagens de baixo nível.	Abordagem biológica para excelente segmentação.	Arquitetura multicamada complexa.
Brain-State-in-a-Box	1977	J. Anderson	Retorno auto-associativo.	Desempenho possivelmente melhor que a rede Hopfield.	Não foi completamente explorada em termos de desempenho e potencial de aplicação.
Hopfield	1982	J. Hopfield	Retorno auto-associativo, otimização.	Conceito simples, estabilidade dinâmica comprovada, de fácil implementação em VLSI.	Incapaz de aprender novos estados, pouca capacidade de armazenamento de memória, muitos estados falsos retornados.
Learning Vector Quantization	1981	T. Kohonen	Retorno auto-associativo (completa o padrão dado), compressão de dados	Capaz de auto-organizar representações vetoriais de distribuições de probabilidades nos dados. Rápida execução uma vez terminado o treinamento.	Problemas não resolvidos em selecionar o número de vetores a serem utilizados e a quantidade de tempo requerido para o treinamento.apropriado.
Self-Organizing Topology-Preserving Map	1981	T. Kohonen	Mapeamento complexo (envolvendo relacionamento de vizinhos), compressão de dados, otimização	Capaz de auto-organizar representações vetoriais de dados com uma ordenação significativa entre as representações.	Problemas não resolvidos em selecionar o número de vetores a serem utilizados e o tempo requerido para o treinamento.

2.4 O Neocognitron [10,11]

O neoconitron é um modelo de rede neural proposto por Fukushima com um propósito prático específico: o reconhecimento de caracteres escritos a mão.

Ao propor o neocognitron, Fukushima e seus colegas estavam interessados em desenvolver um modelo do cérebro e seu sistema visual. Para um melhor entendimento de seu trabalho, faremos uma breve e simplificada explicação sobre o sistema visual.

O nervo ótico é formado por axônios de células nervosas chamadas de gânglios retinais. Esses gânglios são estimulados indiretamente pelos receptores de luz do olho (cones e bastonetes) através de diversos neurônios intermediários.

Através das experiências de Hubel e Wiesel, descobriu-se como determinar a qual estímulo em particular um determinado neurônio é mais sensível. Tomaremos como exemplo as células do Núcleo Lateral Geniculado.

Essas células possuem campos receptivos circulares, ou seja, eles respondem mais fortemente a estímulos contidos em áreas circulares de um determinado tamanho em uma parte particular da retina. A parte da retina responsável por estimular um gânglio em particular é chamada de campo receptivo do gânglio.

Alguns desses campos receptivos possuem uma resposta excitante para pontos de luz focados, e uma resposta inibidora para pontos de luz maiores e mais difusos, enquanto que outros possuem uma característica oposta, com uma resposta inibidora para o ponto de luz focado e excitante para o ponto de luz difuso.

O córtex visual (conhecido também como área 17 do cérebro) em si é composto por seis camadas de neurônios (as células descritas como exemplo acima se conectam, em sua maioria, à quarta camada), e a cada camada a característica das respostas dos neurônios aumenta em complexidade.

Seguindo adiante com o exemplo, as células da quarta camada projetam suas respostas em um conjunto de células diretamente acima delas, chamadas células simples. As células simples respondem a segmentos de linha possuindo uma orientação em particular.

As células simples projetam suas saídas para células conhecidas como células complexas. As células complexas também respondem a segmentos de linha com a mesma orientação que as simples, mas são menos afetadas pela posição dos mesmos na retina, pois integram uma perspectiva – ou campo receptivo – mais ampla, baseadas nas respostas vindas das células simples. Algumas células complexas são sensíveis a segmentos de linha em uma determinada orientação se movendo em uma determinada direção.

Células em diferentes posições no córtex visual se projetam a diferentes áreas do cérebro. As células das camadas 2 e 3 se projetam a células nas áreas 18 e 19 do cérebro. Essas áreas possuem células chamadas de hipercomplexas.

As células hipercomplexas respondem a linhas que formam ângulos ou dobras e que se movem em várias direções ao longo do campo receptor.

Desse estudo podemos perceber que existe uma hierarquia de células com resposta de características progressivamente mais complexas. A partir da existência dessa hierarquia, Fukushima foi capaz de extrapolar essa idéia em uma hierarquia de rede neural em que o nível de abstração dos dados aumenta de acordo com a

profundidade da rede. O neocognitron, portanto, adota uma estrutura hierárquica em uma arquitetura de camadas.

2.4.1 Arquitetura do Neocognitron

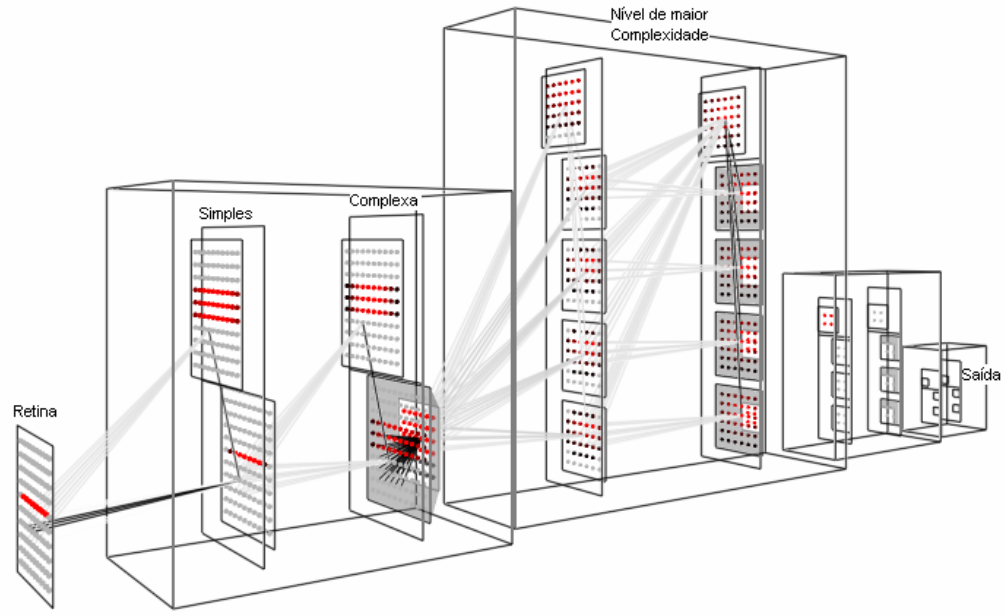


Figura 5 - Estrutura hierárquica do Neocognitron. Imagem gerada pelo visualizador implementado neste trabalho.

O neocognitron é organizado em módulos chamados níveis, conforme a Figura 5. Cada nível consiste de duas camadas: uma camada de células simples ou células-S, seguida por uma camada de células complexas, ou células-C.

Cada camada, por sua vez, é dividida em um número de planos, cada qual consistindo de um conjunto retangular de nós. Em um dado nível, a camada-C e a camada-S podem ou não ter o mesmo número de planos. Todos os planos em uma mesma camada possuem o mesmo tamanho em termos de número de nós. No entanto o número de nós nos planos da camada-S, ou planos-S, de um nível pode diferir do número de nós nos planos da camada-C, ou planos-C no mesmo nível. Além disso, o número de nós por plano pode variar de nível para nível.

Para se construir uma rede completa, combinamos uma camada de entrada, chamada de retina, com um número de níveis de modo hierárquico, e não havendo, a princípio, nada que limite o tamanho da rede em termos de número de níveis.

A estratégia de interconexão entre os planos também é diferente de outros tipos de redes completamente interconectadas. Cada camada de células simples age como um sistema de extração de atributos, que utiliza a camada que a precede como entrada.

Na primeira camada-S, as células de cada plano são sensíveis a características simples na retina, como, por exemplo, segmentos de linhas em diversos ângulos. Cada célula-S em um mesmo plano é sensível à mesma característica, mas em uma localidade diferente na camada de entrada. Células-S em planos diferentes respondem a diferentes características.

Conforme examinamos a rede mais profundamente, as células-S respondem a características com graus mais altos de abstração, como intersecção entre segmentos de linha em vários ângulos e orientações.

As células-C integram as respostas de grupos de células-S dentro de um mesmo plano. Devido ao fato de que cada célula-S dentro de um mesmo plano está procurando a mesma característica em uma localidade diferente, a resposta das células-C é menos sensível à localização exata da característica na camada de entrada.

Este comportamento das células-C é que dá ao neocognitron a capacidade de reconhecer caracteres sem levar em conta sua posição exata no campo da retina. Com essa função, ao chegarmos à última camada de células-C, a camada de saída, o campo receptivo efetivo de cada célula é a retina inteira, ou seja, a camada de entrada como um todo.

Quanto à conexão entre os níveis, é importante notar que apesar de que os planos-S da primeira camada recebem estímulos apenas de um plano (a retina), nas camadas subseqüentes cada plano-S recebe estímulos de todos os planos-C do nível anterior.

Os pesos das conexões para as células-S são determinados por um processo de treinamento descrito mais adiante. Ao contrário da maioria das outras arquiteturas, onde cada nó possui um conjunto de pesos diferentes, todas as células-S de um mesmo plano respondem, como já foi descrito, a uma única característica.

Devido a esse fato, é necessário treinar apenas uma célula-S em cada plano e depois distribuir os pesos resultantes para outras células.

Os pesos nas conexões para as células-C não são modificados por treinamento, sendo geralmente atribuídos de acordo com a arquitetura específica desejada.

2.4.2 Processamento de dados no neocognitron

2.4.2.1 Processamento nas células-S

Cada célula-S, como descrito anteriormente, recebe entradas de uma determinada região dos planos-C do nível anterior. Além disso, a camada-S de um nível está associado a um plano-V_C, composto por um número de células-V_C igual ao número de células-S em cada plano-S existente na camada, conforme demonstrado na Figura 6.

As células-V_C possuem os mesmos campos receptivos que as células-S nas posições correspondentes dos planos. A saída de uma célula-V_C se liga a uma única célula-S, que ocupa a mesma posição no plano correspondente à célula-V_C, em cada um dos planos na camada. Essa saída possui um efeito inibidor nas celulas-S.

Sendo o índice k_l uma referência ao k-ésimo plano no nível *l*, podemos identificar cada célula no plano com um vetor bidimensional, com n indicando sua posição no plano; então definimos o vetor v como a posição relativa de uma célula na camada anterior que fica no campo receptivo da unidade n. Com essas definições, podemos escrever a função de saída de uma célula-S como:

$$u_{S_l}(k_l,n)=r_l\cdot\phi\left[\frac{1+\sum_{k_{l-1}=1}^{K_{l-1}}\sum_{v\in A_l}a_l(k_{l-1},v,k_l)\cdot u_{C_{l-1}}(k_{l-1},n+v)}{1+\frac{r_l}{1+r_l}b_l(k_l)\cdot v_{C_l}(n)}-1\right] \quad \textbf{(1)}$$

onde a função ϕ é um função de limite linear dada por

$$\phi(x)=\begin{Bmatrix}x & x\geq 0\\ 0 & x<0\end{Bmatrix}$$

O somatório interno é a soma de produtos usual do cálculo das entradas, $u_{C_l}(k_{l-1},n+v)$, e pesos, $a_l(k_{l-1},v,k_l)$. A soma se estende por todas as unidades na camada-C anterior que se encontram dentro do campo receptivo da unidade n. Esses são designados pelo vetor $n+v$. Essa soma mede o quanto o padrão de entrada está próximo do vetor de pesos. A_l indica o campo receptivo, que tem uma geometria idêntica para todos os nós de uma mesma camada. O somatório externo se estende por todos os K_{l-1} planos da camada-C anterior.

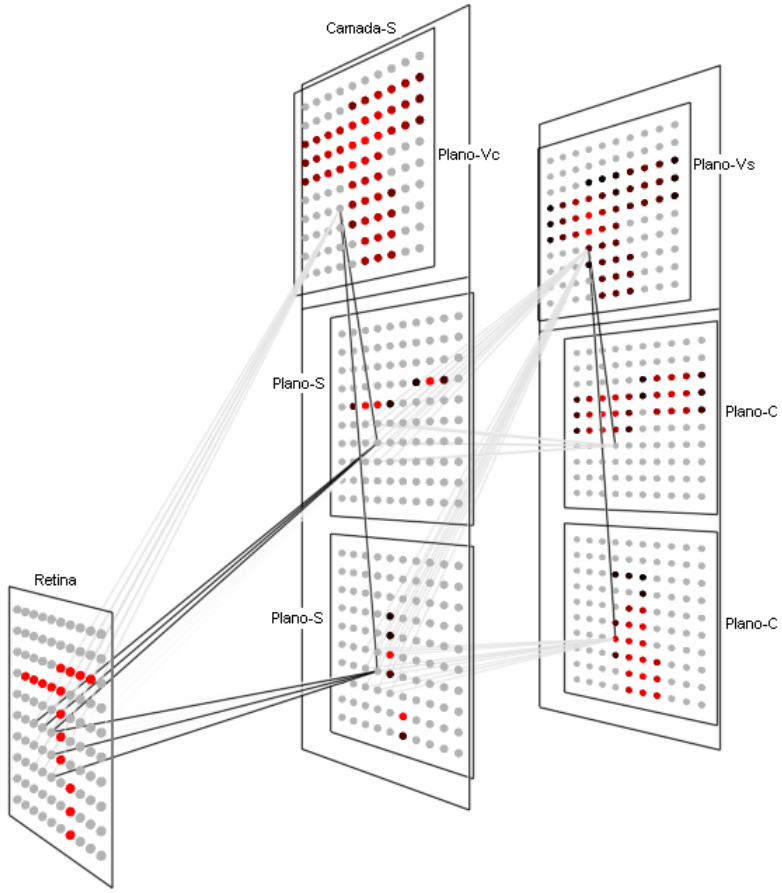


Figura 6 - Planos de uma Camada-S e o Plano inibidor V_C dessa mesma camada. Imagem gerada pelo visualizador implementado neste trabalho.

O produto $b_l(k_l)\cdot v_{C_l}(n)$, no denominador, representa a contribuição inibidora da célula-V_C. O parâmetro r_l, onde $o \le r_l < \infty$, determina a seletividade da célula a um determinado padrão, sendo que um maior r_l exige uma maior excitação, em relação à inibição, para gerar saídas diferentes de 0.

A célula-V_C na posição n tem um valor de saída de

$$v_{C_l}(n) = \sqrt{\sum_{k_{l-1}=1}^{K_{l-1}} \sum_{v \in A_l} c_l(v)\cdot u^2_{C_{l-1}}(k_{l-1}, n+v)} \qquad \textbf{(2)}$$

onde $c_l(v)$ é o peso na conexão de uma célula na posição v do campo receptivo da célula-V_C [11]. Esses pesos não estão sujeitos a treinamento. Eles tomam a forma de uma função normalizada monotonicamente decrescente conforme a magnitude de v aumenta.

Um exemplo desse tipo de função seria:

$$c_l(v) = \frac{1}{C(l)} \alpha_l^{r'(v)} \quad \textbf{(3)}$$

onde $r'(v)$ é a distância normalizada entre a célula localizada na posição v e o centro do campo receptivo, e α_l é uma constante menor que 1 que determina a taxa de diminuição com o aumento da distância.

O fator $C(l)$ é uma constante de normalização:

$$C(l) = \sum_{k_{l-1}}^{K_{l-1}} \sum_{v \in A_l} \alpha_l^{r'(v)} \quad \textbf{(4)}$$

A condição para que os pesos sejam normalizados pode ser descrita como

$$\sum_{k_{l-1}}^{K_{l-1}} \sum_{v \in A_l} c_l(v) = 1 \quad \textbf{(5)}$$

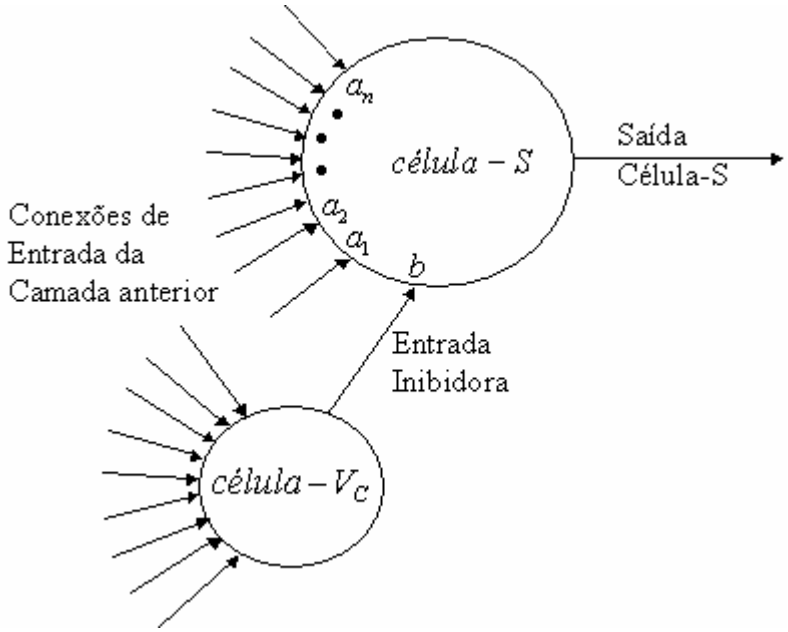


Figura 7- Cálculo da Saída das Células-S[10]

Com essas equações tanto as entradas excitantes como as inibidoras serão mais fortes se o padrão de entrada estiver centralizado no campo receptivo da célula. O esquema da relação entre cada célula-S e sua célula-V_C pode ser visto na Figura 7, acima.

2.4.2.2 Processamento nas células-C

As funções que descrevem o processamento das células-C são similares em forma àquelas das células-S. Assim como na camada-S, cada camada-C tem associada a ela um único plano de unidades inibidoras que funcionam de modo similar às células-V_C na camada-S. Estas unidades recebem o nome de $v_{S_l}(n)$.

Geralmente as unidades em um dado plano-C recebe conexões de entrada de um, ou no máximo um pequeno número, de planos-S da camada anterior. As células-V_S recebem conexões de entrada de todos os planos-S presentes na camada anterior.

A saída de uma célula-C é dada por

$$u_{C_l}(k_l,n)=\psi\left[\frac{1+\sum_{\kappa_l=1}^{K_l} j_l(\kappa_l,k_l)\sum_{v\in D_l} d_l(v)\cdot u_{S_l}(k_l,n+v)}{1+v_{S_l}(n)}-1\right] \qquad \textbf{(6)}$$

onde K_l é o número de planos-S no nível l, $j_l(\kappa_l,k_l)$ é 1 ou 0 dependendo de se o plano-S κ_l é ou não conectado ao plano-C k_l. $d_l(v)$ é o peso na conexão da célula-S na posição v no campo receptivo da célula-C, e D_l define a geometria do campo receptivo da célula-C.

A função ψ é definida por

$$\psi(x)=\begin{cases}\dfrac{x}{\beta+x} & x\geq 0\\ 0 & x<0\end{cases} \qquad \textbf{(7)}$$

onde β é uma constante. A saída das células-V_S é dada por

$$v_{S_l}(n)=\frac{1}{K_l}\sum_{\kappa_l=1}^{K_l}\sum_{v\in D_l} u_{S_l}(k_l,n+v)\cdot d_l(v) \qquad \textbf{(8)}$$

Os pesos, $d_l(v)$ são valores fixos, com a mesma forma geral que os $c_l(v)$ descritos para as células-S, apesar de que pode-se atingir resultados satisfatórios se $d_l(v)$ for um valor uniforme.

Similarmente às células-S, uma célula-C só emitirá saída excitante se o valor de excitação da célula for maior que a média.

2.4.2.3 Treinamento dos pesos das células-S

Apesar de existirem diversos modos de treinamento para o neocognitron, descreveremos aqui o método projetado originalmente, que é o aprendizado sem supervisão.

A princípio, o treinamento segue conforme a maioria das redes neurais, ou seja, apresentamos uma amostra à rede, e os dados são propagados pela rede, permitindo que os pesos das conexões se ajustem progressivamente de acordo com um algoritmo determinado. Depois de os pesos serem atualizados, é apresentado um segundo padrão na camada de entrada, e o processo se repete com todas as amostras de treinamento, até que a rede esteja classificando corretamente os padrões.

O neocognitron possui uma característica que diferencia o processo: todas as células em um mesmo plano compartilham o mesmo conjunto de pesos. Portanto, apenas uma única célula de cada plano precisa participar do treinamento, e, após isso, distribuir o seu conjunto de pesos para as demais células.

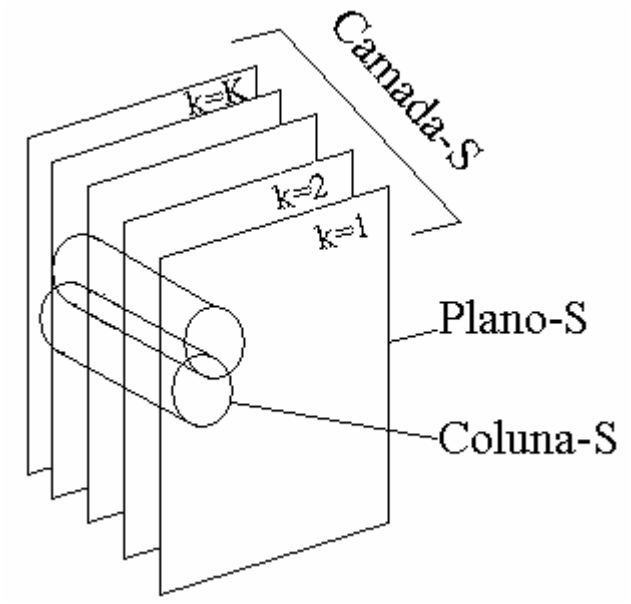


Figura 8- Colunas-S[11]

Para entender melhor o funcionamento, podemos imaginar todos os planos-S de uma dada camada empilhados uns sobre os outros, alinhados de tal modo a que as células correspondentes a uma determinada localidade estejam diretamente umas acima das outras, como exemplificado na Figura 8. Com isso conseguimos imaginar diversas colunas, correndo perpendicularmente aos planos. Essas colunas criam grupos de

células-S, onde todos os membros do grupo têm campos receptivos aproximadamente na mesma localidade na camada de entrada.

Com esse modelo em mente, podemos agora aplicar um padrão de entrada e examinar a resposta das células-S em cada coluna. Para garantir que cada célula-S forneça uma resposta distinta, podemos iniciar os pesos a_l com valores aleatórios pequenos e positivos e os pesos inibidores b_l com zero.

Primeiro anotamos o plano e posição da célula-S cuja resposta é a mais forte em cada coluna. Então examinamos os planos individuais de modo que, caso um plano possua duas ou mais dessas células-S, escolhamos somente a célula-S com a resposta mais forte, sujeita à condição de que cada uma das células esteja em uma coluna-S diferente.

Essas células-S se tornam os protótipos, ou representantes, de todas as células em seus respectivos planos. De maneira semelhante, a célula-V_C com a resposta mais forte é escolhida como representante para as outras células no plano-V_C.

Uma vez escolhidos os representantes, as atualizações dos pesos são feitas de acordo com as seguintes equações:

$$\Delta a_l\left(k_{l-1}, v, \hat{k}_l\right) = q_l c_{l-1}(v) u_{C_{l-1}}\left(k_{l-1}, \hat{n} + v\right) \qquad \textbf{(9)}$$

$$\Delta b_l\left(\hat{k}_l\right) = q_l v_{C_{l-1}}\left(\hat{n}\right) \qquad \textbf{(10)}$$

onde q_l é o parâmetro da taxa de aprendizado, $c_{l-1}(v)$ é a função monotonicamente decrescente descrita anteriormente (Equação 3) e a localização do representante do plano $\hat{k}$ é $\hat{n}$.

Com esse algoritmo, uma vez que as células de um plano passem a responder a uma determinada característica, elas passam a emitir respostas menores em relação a outras características. Os níveis mais profundos, como dito anteriormente, passarão a responder a características mais complexas.

3 Visualização Científica

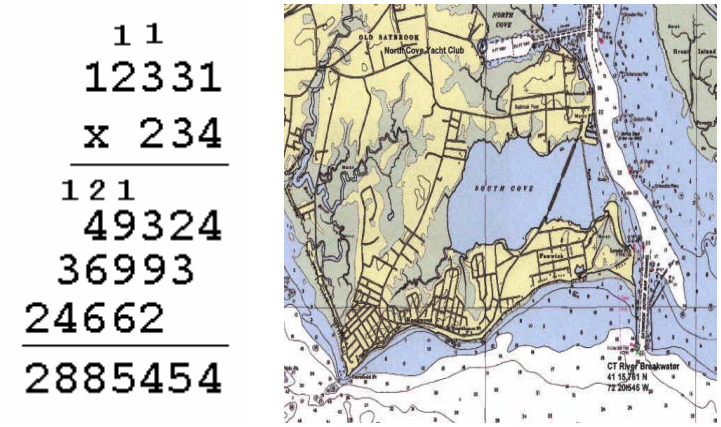
3.1 Introdução

Uma constante encontrada na literatura sobre visualização científica é a palavra transformação. A visualização científica trata exatamente sobre isso: a transformação de uma idéia, modelo, equação, etc. em uma representação, geralmente gráfica, de mais fácil entendimento ou simplesmente em uma forma diferente, para obter um outro ponto de vista sobre o assunto em questão.

Ao tentarmos essa transformação, devemos ter em mente diversos tópicos como, por exemplo, qual o grau de alteração podemos impor sobre os dados representados, a qualidade estética da representação, qual o público alvo da representação, etc.

3.2 Fundamentos

O ser humano se utiliza de ferramentas visuais há muito tempo, mesmo quando não se dá conta disso, como nos casos em que utilizamos papel e caneta ou uma régua de cálculo para realizar uma conta, ou utilizamos um mapa para nos orientarmos em uma cidade ou viagem marítima.



(a) (b)

Figura 9 - Exemplos de ferramentas visuais. (a) Multiplicação (b) Mapa de navegação [3]

No caso do papel e caneta, utilizamos o modelo que nos ensinaram na escola para estender nossa memória através da escrita dos algarismos da conta no papel e ao mesmo tempo para agilizar a conta ao realizarmos diversas contas rápidas ao invés de uma única maior, como demonstrado no exemplo da Figura 9a.

No caso do mapa na Figura 9b, utilizamos o modelo como um banco de dados, organizado de forma que as informações estão mais próximas do ponto onde necessitamos da mesma. Os diversos modelos de mapas marítimos permitem que os cursos sejam traçados de maneira rápida e precisa nos mais diversos pontos da Terra. [3]

Com toda essa diversidade de funções, podemos chegar a alguns fundamentos e tópicos que são úteis ao desenvolvimento de um modelo de visualização.

3.2.1 O sistema visual humano

É estimado que metade dos neurônios do cérebro estão de alguma forma ligados à atividade de processar e entender um estímulo visual[2]. Dessa maneira, se, ao desenvolvermos um modelo de visualização, utilizarmos técnicas que permitam ao cérebro utilizar seus próprios recursos para identificar as informações estaremos nos aproveitando de mais esse processamento de informações para que nossas idéias possam ser compreendidas pelos nosso público.

Segundo [3], a informação visual pode ser processada de dois modos:

- Processamento controlado, onde o processamento é detalhado, lento, consciente, serial e de baixa capacidade. Exemplo: Leitura.
- Processamento automático, onde ele passa a ser superficial, paralelo, tem alta capacidade e é inconsciente. Exemplo: Visão durante a condução de um veículo.

Utilizando o processamento automático, podemos fazer com que o sistema visual detecte padrões de codificação rapidamente, sem termos que nos referir a textos, aumentando a capacidade de absorção de dados pelo observador.

Baseados nessas características foram definidos alguns princípios e estruturas úteis ao desenvolvimento de uma visualização.

3.2.1.1 Estruturas Visuais e Princípios Gestalt

Existem certas estruturas visuais que são detectadas pelo processamento automático do sistema visual descrito acima. São elas:

- Números;
- Orientação de linhas;
- Comprimento;

- Largura;
- Tamanho;
- Curvatura;
- Terminadores;
- Intersecção;
- Fechamento;
- Cor;
- Intensidade;
- Aparecimento/Desaparecimento de objetos;
- Direção de movimento;
- Profundidade e
- Direção da iluminação. [13]

Ao mapearmos dados em uma dessas características, estamos garantindo que eles serão prontamente processados e os padrões que eles formam poderão ser mais rapidamente acessados.

Além dessas estruturas básicas, também foram definidos alguns princípios de organização que devemos ter em mente para construir uma visualização eficiente, chamados princípios Gestalt[22]:

- Pragnanz: Todo padrão de estímulo é visto de tal modo que a estrutura resultante seja a mais simples possível;
- Proximidade: A tendência de que objetos próximos se agrupem em uma mesma unidade de percepção;
- Similaridade: Se diversos estímulos são apresentados juntos, há a tendência de ver a forma de tal modo que itens similares sejam agrupados;
- Fechamento: A tendência de unir contornos que estão muito próximos uns dos outros;
- Boa Continuidade: Elementos vizinhos são unidos quando eles estão potencialmente conectados por uma linha reta ou levemente curvada;
- Destino Comum: Elementos se movendo na mesma direção parecem estar agrupados;

- Familiaridade: Há mais chance de que os elementos formem grupos se os grupos parecem familiares ou significativos.

3.2.2 Processo de visualização

Para se obter uma representação gráfica de qualquer tipo de dados, são necessários:

- Dados crus: vindos de uma base de dados, simulação, experimento, etc;
- Modelo de Visualização: que tipo de efeito gráfico será gerado baseado nesses dados; e
- Um mapeamento entre esses dados e o modelo de visualização.

Entao podemos definir um processo de visualização como sendo composto das seguintes etapas[2]:

- Extração/Análise dos dados;
- Mapeamento dos dados em estruturas visuais ou modelos geométricos,
- Renderização, ou criação da imagem.

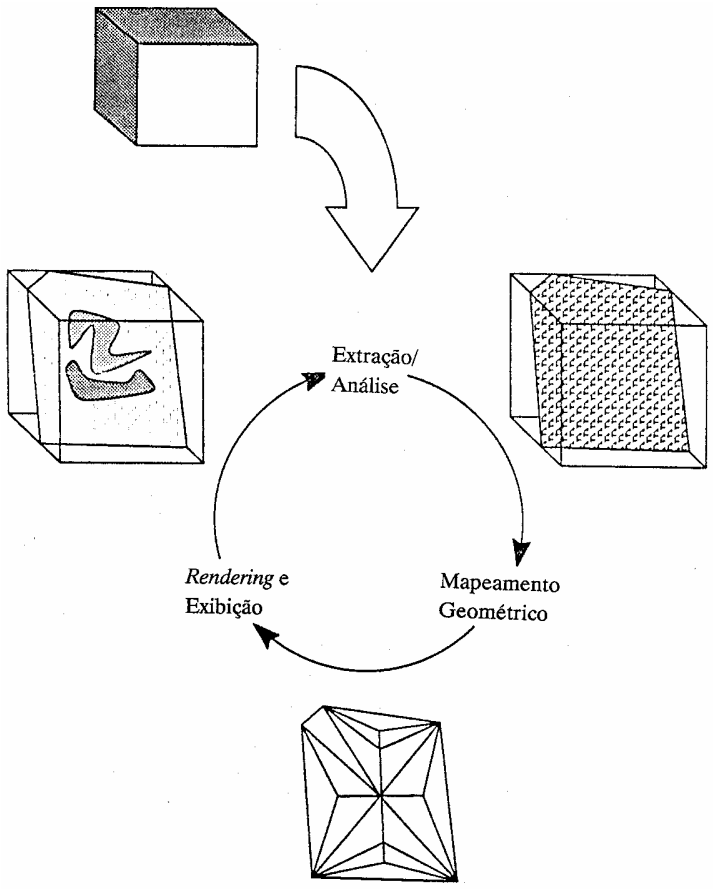


Figura 10 - Ciclo de visualização em aplicações científicas [2]

Ao longo desse processo, esquematizado na Figura 10, é desejável que o controle das transformações possa ser feito pelo usuário da visualização[3], permitindo se extrair o máximo dos dados com poucas alterações e aumentando assim a interatividade da visualização.

Para se implementar uma visualização de uma simulação, é usual a integração entre a fonte de dados e a visualização. Uma alternativa é a separação entre essas duas partes, visando a melhoria de ambas por especialistas diferentes[23]. Essa separação permite que cada especialista se preocupe com os problemas de sua área, seja ela uma simulação que gera os dados utilizados na visualização, ou a própria visualização. Essa forma de trabalho gera algumas vantagens, como a possibilidade de se trocar o modelo da visualização sem ser necessária nenhuma mudança na simulação.

3.3 Aplicações

Visualizações já foram criadas sobre os mais variados conjuntos de dados, normalmente quando se quer um entendimento mais profundo sobre um conjunto muito grande de dados ou descobrir padrões de alto nível existentes nesses conjuntos, tarefa quase impossível de se realizar olhando apenas para as tabelas.

Nesta seção serão apresentados alguns exemplos da variedade de aplicações das visualizações.

3.3.1 Análise de dados meteorológicos

Dado que o meteorologista trabalha normalmente com uma grande quantidade de dados, a representação gráfica dos dados obtidos de satélites ou radares é um recurso muito importante para o estudo do comportamento do tempo. Assim, muitos dos sistemas de Visualização de Dados Meteorológicos tem provado serem ferramentas poderosas para os meteorologistas[2]. Um exemplo dessa utilidade é demonstrado na Figura 11, onde um modelo matemático de nuvens foi transformado em uma imagem para o melhor entendimento dos dados presentes no modelo.

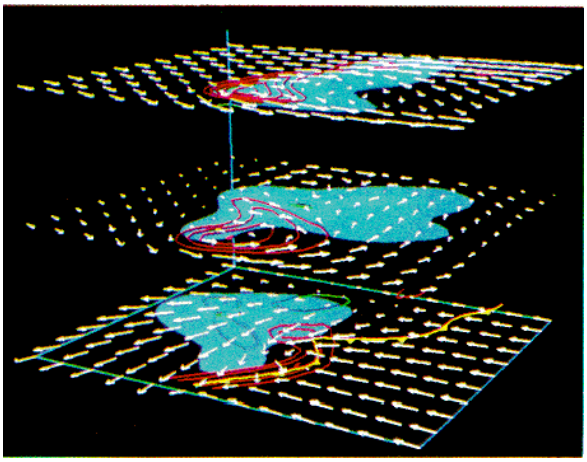
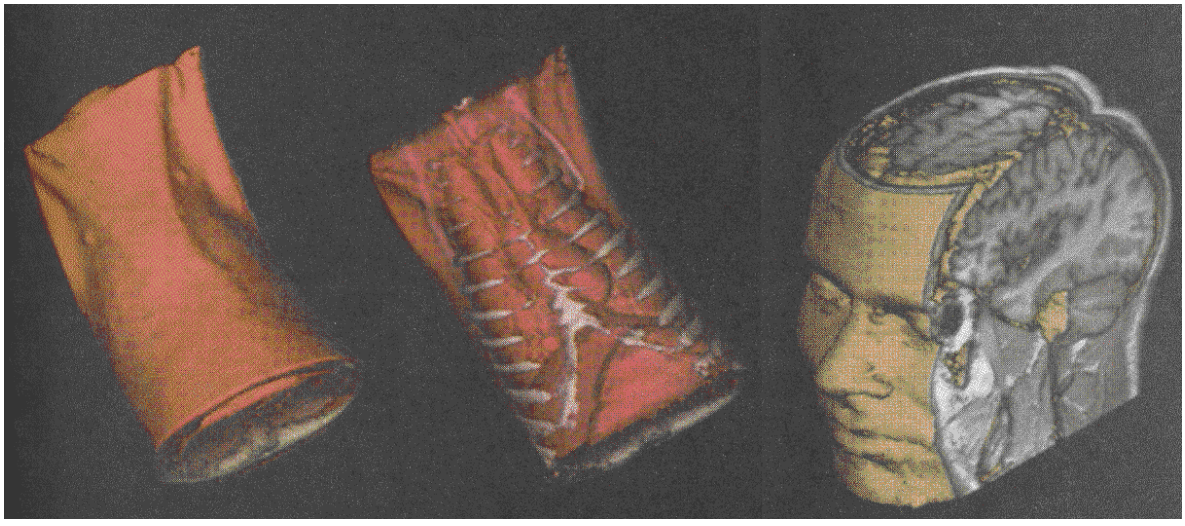


Figura 11 - Visualização de um modelo numérico de nuvem [20]

3.3.2 Imagens médicas

Computações aplicadas a imagens médicas criaram oportunidades em medicina diagnóstica, planejamento cirúrgico para próteses ortopédicas e planejamento de radioterapia. Em cada um desses casos, essas oportunidades vieram a tona com visualizações de 2 e 3 dimensões de partes do corpo previamente inacessíveis à visão[4]. A Figura 12 exemplifica essa capacidade de manipulação das imagens formadas a partir de sensores médicos. Um médico pode examinar a aparência externa de uma parte do corpo (Figura 12a) ou retirar camadas externas e observar detalhes interiores (Figura 12b). Em outro caso, é possível fazer cortes diversos na imagem para observar diversas camadas ao mesmo tempo (Figura 12c).



(a) **(b)** **(c)**

Figura 12 - Imagens médicas [4]

3.3.3 Visualização de documentos

Documentos de texto estão intimamente ligados aos nossos trabalhos. Nós nos comunicamos com eles e os tratamos como extensões de nossa memória. Nesse aspecto, podemos estar interessados em como os conteúdos de um único documento se relacionam e como diversos documentos se relacionam entre si[3]. Esse tipo de estudo pode agilizar a busca por informações em espaços tão restritos quanto um programa de computador ou tão amplos quanto a internet. Na Figura 13 observamos um exemplo de estudo das áreas de importância para um pesquisador baseados em seus documentos pessoais e publicações. Com a visualização é mais fácil perceber onde os assuntos são interligados, e qual a freqüência em que o pesquisador se envolve com cada assunto.

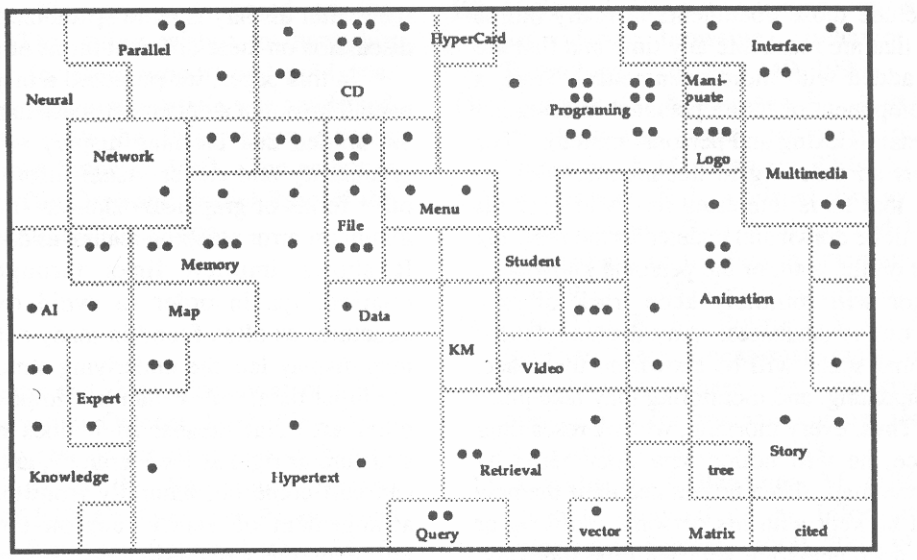


Figura 13 - Visualização de documentos pessoais. O tamanho das áreas dos assuntos indica sua importância ao pesquisador, enquanto que a proximidade entre as áreas indica proximidade dos assuntos quando pesquisados por esse pesquisador [18]

4 Computação Gráfica

A computação gráfica sempre foi um dos ramos mais visualmente espetaculares da tecnologia da computação, produzindo imagens cuja aparência e movimento as tornam bem diferentes das outras formas de saídas computacionais. A computação gráfica também é um meio extremamente efetivo para a comunicação entre homem e computador[19]; como visto no capítulo sobre visualização, o olho humano pode absorver o conteúdo da informação mostrada em um diagrama ou visão perspectiva muito mais rápido do que em uma tabela de números.

4.1 Definição

O termo "COMPUTER GRAPHICS" surgiu provavelmente em 1959, criado por Verne L. Hudson, quando este coordenava um projeto na BOEING para simulação de fatores humanos em aviões. Hoje "COMPUTER GRAPHICS" é definido pela ISO (International Standards Organization) como sendo: "Métodos e técnicas para conversão de dados de e para dispositivos gráficos, através do computador". Portanto, tal definição abrange toda e qualquer aplicação que envolva imagens tratadas por computador, desde a geração de gráficos simples por programas de auxílio à engenharia até imagens tão realistas a ponto de substituírem cenários de filmes[8].

4.2 Bibliotecas Gráficas

Bibliotecas ou Pacotes Gráficos são conjuntos de rotinas gráficas básicas, com as quais pode-se desenvolver programas aplicativos sofisticados sem necessidade de se preocupar com detalhes particulares dos dispositivos gráficos, como, por exemplo, a forma de se gerar uma reta em um determinado terminal de vídeo.

Assim como para programação matemática foram criadas bibliotecas de rotinas que executam, de maneira eficiente, funções aritméticas do tipo seno, raiz quadrada, módulo e outras funções, para os programadores que se utilizam da computação gráfica se fizeram necessárias rotinas que traçam retas, arcos, polígonos e realizam diversas operações gráficas[8].

Características desejáveis em uma boa biblioteca gráfica são:

- Simplicidade: Características muito complexas para o programador entender não devem ser utilizadas.

- Consistência: A biblioteca deve se comportar de maneira previsível. Nomes de funções, sequências de comandos, tratamento de erros e sistemas de coordenadas devem todos seguir padrões simples e consistentes, sem exceções.
- Completude: não devem haver omissões no conjunto de funções providos pela biblioteca. A biblioteca deve possuir um conjunto razoavelmente pequeno de funções que convenientemente tratem de um grande grupo de aplicações.
- Robustez: Erros triviais no uso da biblioteca devem ser corrigidos sem comentários, enquanto que os erros graves devem ser reportados da maneira mais prestativa possível. Somente erros muito graves devem parar a execução do programa.
- Performance: A biblioteca deve manter uma performance consistente, ou seja, a performance deve ser a mesma qualquer que seja a implementação que a utilize[19].

Considerando essas características, foram criadas diversas bibliotecas gráficas ao longo dos anos, mas atualmente três se destacam como bibliotecas utilizadas amplamente na área comercial, que serão descritas nas próximas sessões.

4.2.1 Java 3D

A API (Application Program Interface) Java 3D é uma hierarquia de classes Java que serve como interface para um sistema sofisticado de renderização de gráficos tridimensionais e sons. Java 3D provê construções de alto nível para criar e manipular geometria 3D, e para criar as estruturas utilizadas para renderizar tal geometria. Utilizando essa API, desenvolvedores podem criar eficientemente universos virtuais precisos, em uma grande variedade de tamanhos[14].

Porém, essa API utiliza OpenGL e Direct3D (bibliotecas gráficas descritas a seguir) como suas API´s de renderização de baixo nível. Ela depende dos drivers de OpenGL e Direct3D para aceleração da renderização de baixo nível. Portanto, utilizar hardware que oferece aceleração para uma dessas duas API´s é o melhor modo de aumentar o desempenho geral da renderização da API Java 3D[15].

4.2.2 Direct3D

Direct 3D é uma API da Microsoft para gráficos 3D baseada em COM (Component Object Model), originada de uma API para gráficos 3D de terceiros, voltada para a arquitetura x86. A Microsoft adquiriu a tecnologia em 1996 e a oferece como um componente de sua tecnologia DirectX. A API contém uma interface de modo imediato e uma interface em forma de grafo estruturado hierárquico de cena.

A Microsoft define a API Direct3D por suas interfaces COM, enquanto que a semântica é definida nos arquivos de ajuda e códigos de exemplo fornecidos no kit de desenvolvimento necessário para se utilizar a API[5].

4.2.2.1 Vantagens

A biblioteca provê recursos como Programmable Pixel e Vertex Shaders, que são recursos avançados que permitem repor partes do procedimento normal de renderização por códigos próprios; além disso, ela também possui funções que permitem enumerar exatamente quais são os hardwares gráficos disponíveis ao sistema.

Outra vantagem é a sua estruturação, que é orientada a objetos, e, nas versões recentes, foi tornada mais intuitiva e de uso mais fácil aos desenvolvedores. Sua interface COM permite que sejam introduzidas mudanças no código sem que se quebre a compatibilidade com códigos antigos[6].

4.2.2.2 Desvantagens

Sua atualização ocorre aproximadamente anualmente, o que é muito lento quando se trata de desenvolvimento na indústria da computação gráfica. A Microsoft diminui esse atraso trabalhando junto às empresas de hardware e disponibilizando funções em sua API antes mesmo que elas estejam disponíveis nas placas de vídeo.

Outra desvantagem é a quantidade de código necessária para se iniciar um programa em Direct3D: na versão 8.0 da API, são necessárias cerca de 200 linhas de código para se desenhar o primeiro triângulo, além do que sua interface COM torna difícil a programação em C.

A maior desvantagem, no entanto, é a falta de portabilidade, pois essa API pode ser utilizada somente no sistema operacional Windows, da Microsoft e não é um padrão aberto, fazendo com que decisões errôneas por parte da Microsoft sobre qual característica suportar permaneçam erradas por aproximadamente um ano[6].

4.2.3 OpenGL

OpenGL é uma API para gráficos 3D baseada em procedimentos da SGI (Silicon Graphics, Inc), sucessora da biblioteca IrisGL, originada para o mercado de workstations para Unix.

As definições da OpenGL são controladas pelo OpenGL Architectural Review Board (ARB). O ARB controla a especificação pela qual uma implementação pode se chamar de OpenGL. Essa especificação é um documento em prosa em inglês com a matemática associada que define a semântica da API. Ele também define um conjunto de teste de conformidade que serve como validação para uma implementação.

O ARB foi formado em 1992 e segue um conjunto de regras para revisar e supervisionar a especificação da API.

A OpenGL define uma máquina de estados que controla o procedimento de renderização. Os atributos da máquina de estados são modificados através de chamadas de procedimentos e os vértices definindo as primitivas gráficas são especificados para o procedimento de renderização através de chamadas de procedimento[5].

Cada fabricante de hardware também pode criar extensões da API para introduzir recursos exclusivos existentes em seu hardware.

4.2.3.1 Vantagens

OpenGL é muito portátil. Ela pode executar eficientemente em quase todas as plataformas em existência. A razão para isso é que OpenGL é um padrão aberto. Isso significa que qualquer companhia com uma plataforma que deseje suportar OpenGL pode comprar uma licença da SGI e então implementar todo o conjunto de características definido pela OpenGL para aquela plataforma.

Outra ventagem é a grande variedade de características presentes na biblioteca, tanto em seu núcleo quanto em suas extensões. A capacidade de extensão permite que ela esteja imediatamente atualizada com as novas características de um novo hardware, apesar da confusão que isso causa. Devido à formação do ARB se dever a diversas companhias diferentes, as características disponíveis na OpenGL representam um grande grupo de interesses, tornando-a, portanto, útil em muitas aplicações diferentes.

OpenGL também é utilizada em diversos setores. Sua estabilidade e suporte a diversas plataformas atraem setores de tecnologias fora da indústria dos jogos. Ela é reconhecida como o padrão da indústria para gráficos em todos os setores, exceto no de

jogos, onde a Direct3D provê uma competição viável, uma vez que está implementada visando o sistema operacional mais utilizado por usuários domésticos[6].

4.2.3.2 Desvantagens

As extensões devem ser mencionadas também como desvantagens, porque, apesar de poderosas, elas podem tornar o código confuso. O desenvolvedor também pode se confundir com qualquer compilador que não possua rastreamento de referências. A pior parte é o fato de que muitas extensões recentes são completamente específicas a um novo hardware de empresas específicas. Apesar de servirem para testar as capacidades desse novo hardware, esse tipo de extensão não é utilizado comercialmente.

As convenções de nome também podem parecer exageradas, uma vez que muitas IDEs (Integrated Development Environment) possuem ajuda sensível ao contexto que pode mostrar os parâmetros requeridos às funções. Além disso, ter 12 nomes diferentes para uma única função pode parecer estranho à programação em C++, onde o recurso de sobrecarga de funções evitaria isso[6].

4.3 Sistemas de Partículas

Sistemas de partículas são utilizados para representar objetos complexos, como fogo, água, explosões, etc. Esses sistemas possuem como primitivas não vértices e polígonos definindo suas bordas, mas por nuvens de pontos, que definem seu volume.

Além disso, um sistema de partículas não é uma entidade estática. Suas partículas mudam de forma e se movem com a passagem do tempo; novas partículas “nascem”, enquanto as velhas “morrem”.

Objetos representados por um sistema de partículas não são determinísticos, uma vez que sua forma não é completamente especificada. Ao invés disso, eles são dinâmicos, com suas formas e aparências definidas através de métodos estocásticos[21].

Além disso, pode-se associar formas às partículas, como esferas, retângulos, e até mesmo aves e peixes, esses últimos utilizados na criação de vida virtual. De fato, um sistema de partículas pode ser construído através de um conjunto de partículas ou de um conjunto de sistemas de partículas mais simples, de forma a gerar uma hierarquia[9].

Devido a essa capacidade de representar diversos objetos parecidos como um único sistema, e a capacidade de se formar hierarquias de sistemas de partículas é que

este trabalho utiliza esses sistemas para simulação e visualização de redes neurais, como explicado no Capítulo 5.

5 Trabalho Realizado

5.1 Objetivos

Este trabalho possui diversos objetivos, sendo o principal o estudo da aplicação da visualização no estudo da rede neural Neocognitron, visando a melhoria do entendimento sobre o comportamento desta rede específica.

Ainda sobre o aspecto do aprendizado dessa rede neural, outro objetivo foi a criação de uma ferramenta que permitisse manipular essa visualização livremente, de modo a ser utilizada no estudo da rede neural.

5.2 Conceitos utilizados

O meio para se fazer a visualização do Neocognitron em três dimensões foi utilizar uma hierarquia de sistemas de partículas, como descrito na seção 4.3.

Utilizando-se do conceito de sistemas hierárquicos de partículas, foi definido um sistema de partículas composto da seguinte forma:

As partículas mais primitivas do sistema representam os nós de processamento da rede neural, existindo, a exemplo do Neocognitron (Seção 2.4), diversos tipos de partículas diferentes, com formas de processamento diferente.

Essas partículas são agrupadas em sistemas maiores, os planos, que por sua vez são agrupados em camadas. Ainda seguindo o modelo do Neocognitron, essas camadas são agrupadas para formar níveis, enquanto que a rede neural é composta de uma série desses níveis. Os detalhes precisos são fornecidos na seção 5.4.1.

Esse sistema de partículas foi desenvolvido na forma de uma biblioteca de classes, fazendo-se o uso do que foi dito em 3.2.2, ou seja, a separação entre o simulador da rede neural e a visualização do mesmo, utilizando-se dos recursos de orientação a objetos para permitir que a visualização se acople de forma transparente à simulação, e que tanto a visualização quanto a simulação possam ser estendidas.

5.3 Sobre a Programação

Tanto a aplicação quanto a biblioteca foram desenvolvidos na linguagem C++, utilizando o compilador Microsoft Visual C++ e sua biblioteca de auxílio MFC (Microsoft Foundation Classes), no sistema operacional Windows.

O compilador foi escolhido devido à facilidade que dispõe para manipular o grande número de classes utilizadas, e por possuir recursos (a biblioteca MFC) para trabalhar com o sistema operacional de forma fácil e rápida.

A biblioteca MFC foi utilizada, não só pela facilidade em tratar com o sistema operacional mas também por implementar um modelo de documento/visão semelhante ao implementado na biblioteca desenvolvida.

5.4 A biblioteca

Foi criada uma biblioteca de classes, com a função de simular o comportamento de uma rede neural, o modelo Neocognitron no caso, e de transformar os dados obtidos com essa simulação em comandos da biblioteca gráfica OpenGL, a serem executados pela ferramenta.

Como dito acima, optou-se pela separação, ainda que parcial, entre simulador e visualizador, para maior flexibilidade e modularização. Ambos os módulos serão descritos a seguir.

Na escrita do código, procurou-se seguir a notação utilizada na biblioteca MFC (Microsoft Foundation Classes), ou seja, no início de cada nome de variáveis, membros de classes ou argumentos de funções coloca-se, em letras minúsculas, o indicador do tipo de dado. Com essa notação é mais fácil e rápido reconhecer os tipos de dados envolvidos nas operações.

5.4.1 Simulador

O simulador é a parte da biblioteca responsável pela geração dos dados "crus" da rede neural. Como dito anteriormente, ele é um sistema de partículas hierárquico, com os nós como partícula mais primitiva e a rede como o sistema mais avançado. A seguir será explicado o papel de cada uma dessas classes na biblioteca, a partir da mais básica para as estruturas mais complexas. Algumas estruturas, como os nós e os planos, podem ser utilizadas para a implementação de outras redes neurais, enquanto que as estruturas de camada, níveis e rede foram especificamente criadas para a arquitetura do neocognitron.

5.4.1.1 O nó (CNeuralNode)

Esta é a classe básica do sistema de partículas do simulador. Ela age como um retentor de dados e como unidade de processamento. Cada vez que é ordenada, ela calcula a nova saída com base em suas entradas dispõe o resultado em sua variável de saída. Seu sistema de fontes e pesos é organizado em forma de um arranjo de matrizes retangulares, que são utilizadas, no neocognitron, para representar o campo de visão das células em cada conexão, como demonstrado na Figura 14.

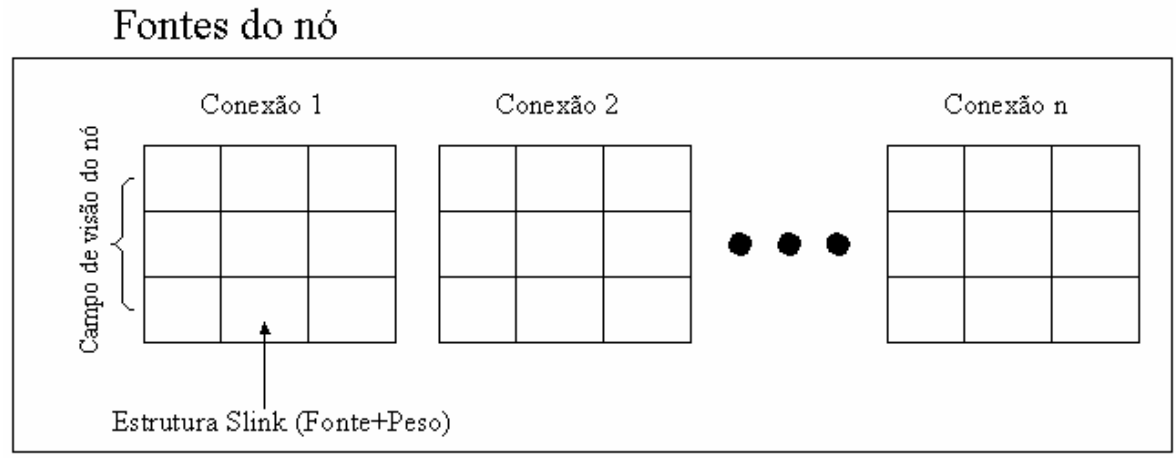


Figura 14 - Estrutura do sistema de fontes e pesos do nó

Essa estrutura de campos de visão é implementada utilizando uma unidade de dados simples, chamada de SLink. Essa estrutura tem seu código exposto na Figura 15.

```
typedef struct
{
    double m_dWeight;            //Peso da conexão
    CNeuralNode * m_pnnSource;   //Aponta para o nó fonte da conexão
} SLink;
```

Figura 15 - Estrutura SLink

Além dessa estrutura, o nó possui métodos com funções importantes, como calcular o valor de saída e manipular as suas fontes de dados, adicionando ou removendo conexões com outros nós, além de métodos com funções secundárias de acesso aos dados do nó, como nome, valor de saída já calculado e situação inibidora.

A Figura 16 demonstra o diagrama da classe CNeuralNode, expondo seus atributos e métodos.

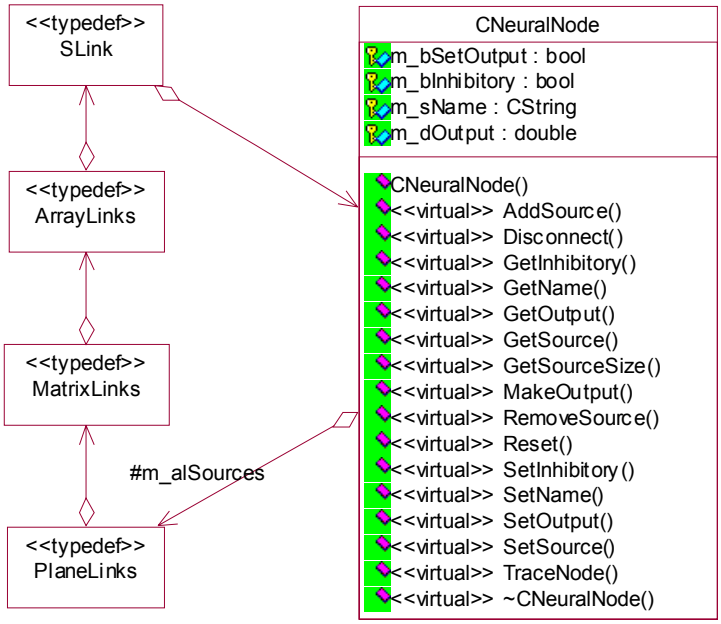


Figura 16 - Diagrama de classe da Classe CNeuralNode e estruturas auxiliares

Classes Derivadas

Para implementar o neocognitron, com seus diversos tipos de células, foram criadas subclasses, uma para cada tipo de célula, de modo a especializar alguns de seus métodos. A Figura 17 demonstra esta hierarquia de classes criada para a implementação do neocognitron.

A funcionalidade de cada classe derivada será descrita a seguir.

Células S (CSCell)

As células-S são descritas em detalhes na seção 2.4.2.1(Processamento nas células-S). Elas são responsáveis pela extração de características das células-C do nível anterior. Para sua implementação foi adicionado um atributo que representa sua ligação com a célula inibidora V_C e métodos para a manipulação desse novo atributo, além da sobrecarga de outros métodos da classe CNeuralNode para configurar seu comportamento, como o método que calcula o valor da saída do nó e o método que inicia as conexões do nó com valores aleatórios pequenos quando pedido. O restante de seu comportamento foi herdado da classe CNeuralNode.

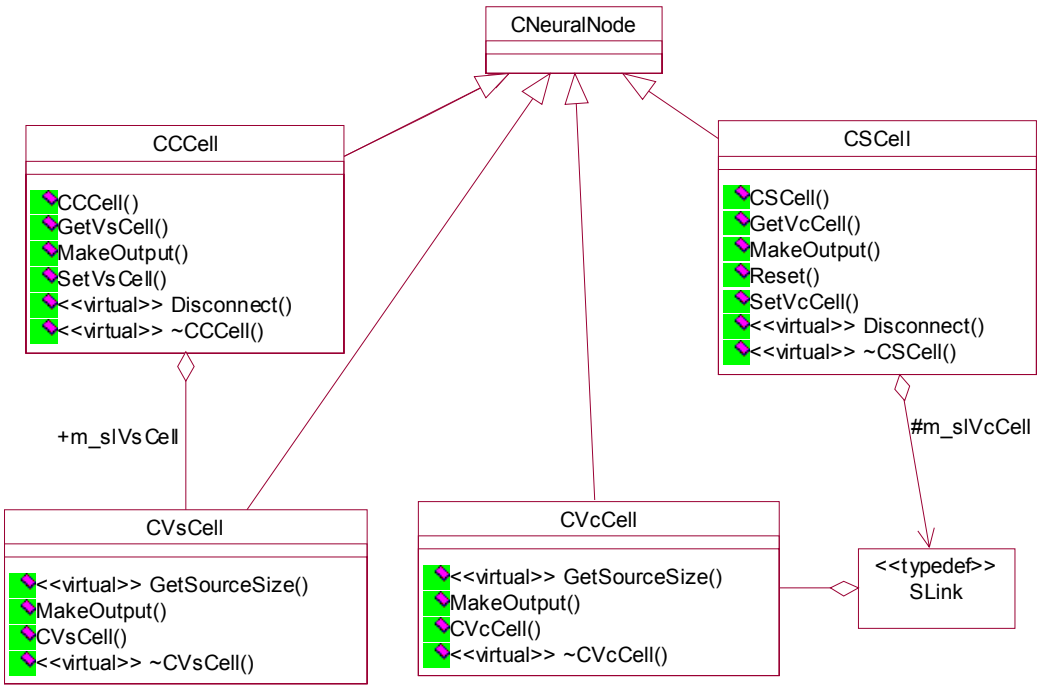


Figura 17 - Estrutura hierárquica dos nós do neocognitron

Células V_C (CVcCell)

As células-V_C são descritas em detalhes na seção 2.4.2.1(Processamento nas células-S). Elas são responsáveis pela inibição das células-S de sua camada, fazendo com que somente células-S com resposta acima da média sejam realmente acionadas. Para sua implementação, foram sobrecarregados os métodos da classe CNeuralNode responsáveis pelo cálculo do valor da saída do nó e pela manipulação de fontes, que devem ignorar a primeira posição do conjunto. O restante de seu comportamento foi herdado da classe CNeuralNode.

Células C (CCCell)

As células-C são descritas em detalhes na seção 2.4.2.2 (Processamento nas células-C). Elas são responsáveis pela integração das respostas das células-S da camada anterior. Para sua implementação foi adicionado, como no caso da célula-S, um atributo representando sua conexão a uma célula V_S, e métodos para manipulação desse novo atributo, além da sobrecarga do método da classe CNeuralNode que calcula o valor da saída do nó. O restante de seu comportamento foi herdado da classe CNeuralNode.

Células Vs (CVsCell)

As células-V_S são descritas em detalhes na seção 2.4.2.2 (Processamento nas células-C). Elas são responsáveis pela inibição das células-C de sua camada, fazendo com que somente células-C com resposta acima da média sejam realmente acionadas. Para sua implementação, foram sobrecarregados os métodos da classe CNeuralNode responsáveis pelo cálculo do valor da saída do nó e pela manipulação de fontes, que devem ignorar a primeira posição do conjunto. O restante de seu comportamento foi herdado da classe CNeuralNode.

5.4.1.2 O Plano (CNeuralPlane)

O plano é um sistema de partículas por si só, tendo como partículas os nós do tipo CNeuralNode. Além disso, ele também é a partícula do sistema criado pelas camadas, que serão vistas mais adiante. Os nós são armazenados em uma matriz retangular, e criados ou destruídos quando se altera o tamanho do plano. O plano foi projetado para o uso na construção do neocognitron, mas poderia ser ajustado para o uso com outros tipos de rede, talvez fazendo o papel das camadas da maioria das arquiteturas.

Os comandos dados ao plano se refletem em todos os seus nós, ele determina o tamanho do campo de visão dos seus nós e os conecta aos nós corretos do plano que serve de fonte de dados para ele. O plano possui uma lista de conexões, que armazenam uma indicação dos planos aos quais ele está conectado e uma matriz de pesos correspondente aos pesos que serão passados aos seus nós quando é feita a conexão. A unidade de dados armazenada nessa lista é do tipo SPlaneLink, cujo código se encontra na Figura 18.

```
typedef struct {
    CDoubleMatrix* m_Weight;            //Matriz quadrada de pesos
    CNeuralPlane* m_pnpaSourcePlane;    //Plano fonte
} SPlaneLink;
```

Figura 18 - Estrutura SPlaneLink

O plano possui métodos que retornam um nó do tipo do qual ele é composto, de modo a tornar mais flexível e fácil a especialização do plano, bastando geralmente à classes derivadas somente sobrecarregar esse método, alterando com isso o tipo de nó

utilizado no plano, mas mantendo o mesmo comportamento em relação ao conjunto de nós.

A Figura 19 demonstra o diagrama da classe CNeuralPlane, expondo seus atributos e métodos.

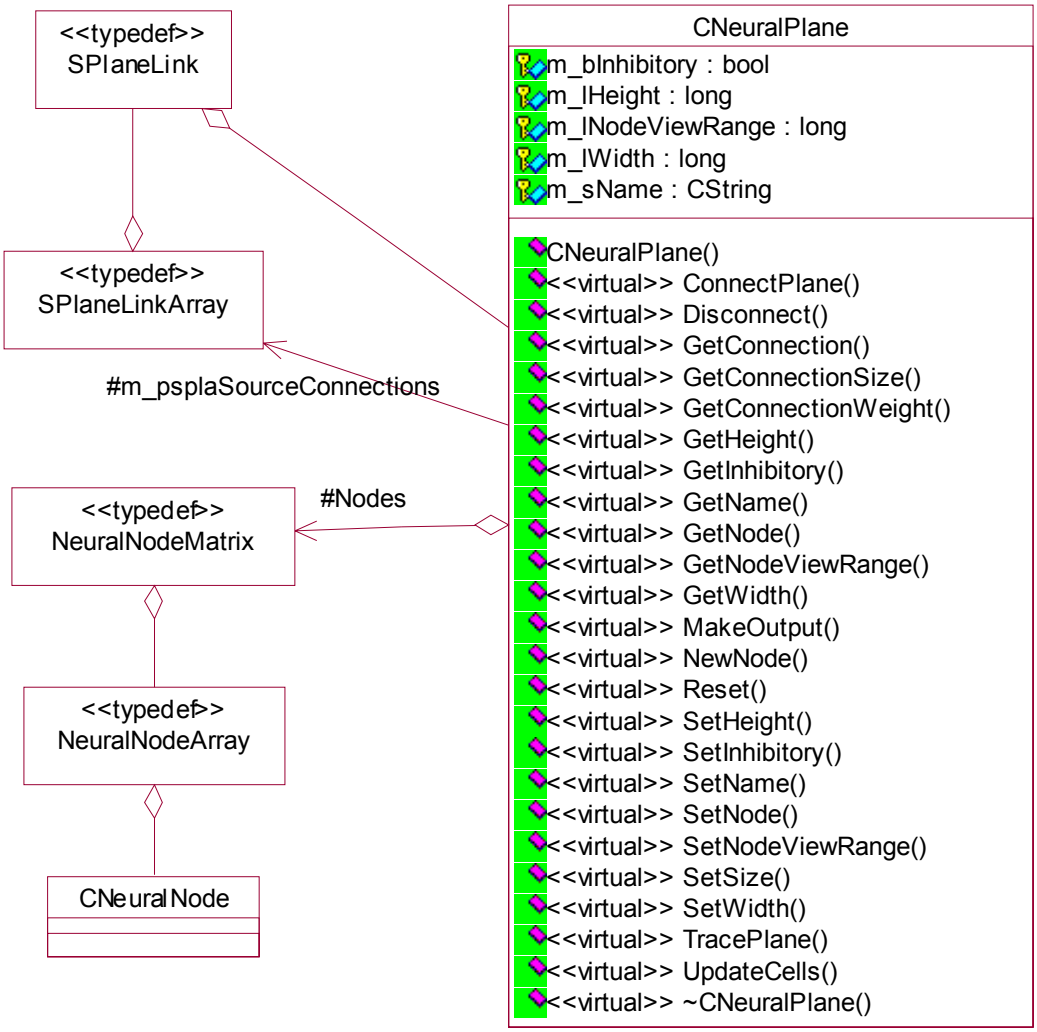


Figura 19 - Diagrama de classe da Classe CNeuralPlane, estruturas auxiliares e relacionamento com a classe CNeuralNode

Classes Derivadas

Para implementar o neocognitron, com seus diversos tipos de planos, foram criadas subclasses, uma para cada tipo de plano, de modo a especializar alguns de seus métodos. A hierarquia de classes de planos criada para a implementação do neocognitron será apresentada por partes antes dos planos derivados.

A funcionalidade de cada classe derivada será descrita a seguir.

Plano S (CSCellPlane)

Os planos-S são compostos por céluas-S, implementadas pela classe CSCell, já descrita. Eles são diferentes do plano genérico por possuírem uma conexão especial: um plano inibidor do tipo V_C (Seção 2.4.2.1 - Processamento nas células-S). Essa conexão possui um peso específico, a ser alterado durante o treinamento (Seção 2.4.2.3 - Treinamento dos pesos das células-S). Além do plano inibidor, eles também possuem o parâmetro r_l, que age como uma medida de sensibilidade do plano(Seção 2.4.2.1 - Processamento nas células-S).

O posicionamento desta classe na hierarquia dos planos pode ser vista na Figura 20.

Plano V_C (CVcCellPlane)

Os planos-V_C são compostos por céluas-V_C, implementadas pela classe CVcCell, já descrita. Eles são diferentes do plano genérico por possuírem uma conexão especial: em um plano inibidor do tipo V_C a primeira conexão armazena o peso a ser utilizado para a conexão com todos os planos fonte. Essa conexão possui um peso específico, não alterado durante o treinamento, e calculado de antemão, com o auxílio do fator α_l. Esse peso é calculado baseando-se no raio de visão do nó, como descrito em 2.4.2.1 - Processamento nas células-S.

O posicionamento desta classe na hierarquia dos planos pode ser vista na Figura 20.

Figura 20 - Hierarquia de classes referentes aos planos S e V_C

Plano C (CCCellPlane)

Os planos-C são compostos por céluas-C, implementadas pela classe CCCell, já descrita. Eles são diferentes do plano genérico por possuírem uma conexão especial: um plano inibidor do tipo V_S (Seção 2.4.2.2 - Processamento nas células-C). Essa conexão possui um peso fixo de 1, sendo portanto desnecessário o armazenamento de tal fator. Outra diferença são os pesos de suas conexões, que se comportam de forma semelhante aos planos-V_C, ou seja, são calculadas baseando-se no raio de visão do nó. Para o cálculo das saídas dos nós, são armazenados os valores da função d_l e dos fatores α_l,

necessário ao cálculo dessa função, e β, necessário ao cálculo da saída das células-C do plano.

O posicionamento desta classe na hierarquia dos planos pode ser vista na Figura 21.

Figura 21 - Hierarquia de classes referentes aos planos C e V_S

Plano V_S (CVsCellPlane)

Os planos-V_S são compostos por céluas-V_S, implementadas pela classe CVsCell, já descrita. Eles são diferentes do plano genérico por possuírem uma conexão especial: em um plano inibidor do tipo V_S a primeira conexão armazena o peso a ser utilizado para a conexão com todos os planos fonte. Essa conexão possui um peso específico, não alterado durante o treinamento, e calculado de antemão, com o auxílio

do fator α_l. Esse peso é calculado baseando-se no raio de visão do nó, como descrito em 2.4.2.2 - Processamento nas células-C.

O posicionamento desta classe na hierarquia dos planos pode ser vista na Figura 21.

Retina (CRetina)

Para a implementação do neocognitron, é necessário um plano especial de entrada, chamado retina, que abre as imagens a serem utilizadas pela rede. Esse plano possui facilidades para a abertura de imagens, permitindo o carregamento da imagem em tons de cinza (refletindo a intensidade das cores) ou com apenas uma das três bandas de cores (vermelho, verde ou azul), sendo pouco diferente do plano padrão (CneuralPlane) em outros aspectos. Sua posição na hierarquia dos planos pode ser vista na Figura 22.

Figura 22 - Hierarquia de classes referentes aos planos auxiliares Retina e Redutor

Redutor (CAverageReducingPlane)

Plano auxiliar criado para realizar a redução de tamanho dos planos-C, de modo a servir de entrada para os planos-S do nível posterior ao seu. Estes planos auxiliares possuem uma alteração no algoritmo de conexão, fazendo com que se conectem aos planos-C com o campo de visão necessário para a redução e matriz de pesos que faz a média entre os valores do campo de visão de cada nó. Essa conexão ocorre no "centro" do plano fonte, ou seja, a diferença entre os dois planos é dividida igualmente ao redor do plano redutor. Sua posição na hierarquia de planos pode ser vista na Figura 22.

5.4.1.3 A Camada (CNeuralLayer)

A camada é o próximo passo na hierarquia de sistemas de partículas, tendo como partículas os planos definidos anteriormente. A utilidade desta estrutura segue o mesmo padrão do plano, ou seja, fornece um controle de nível mais alto sobre suas partículas, mas permitindo que cada partícula seja acessada separadamente para um controle mais fino. A camada é construída com base em um conjunto de planos e um plano inibidor, e possui, diferentemente dos planos, funções para o treinamento da rede, ainda que restritas à camada atual, uma vez que o treinamento envolve a competição entre os planos.

Outro tipo de método envolve a conexão entre duas camadas, que pode conectar duas camadas já existentes ou criar uma camada já conectada a outra camada existente. Para conectar duas camadas, pode-se indicar com qual camada se deseja conectar e deixar o comportamento padrão da camada determinar o modo de conexão ou passar uma estrutura Connections (como definida na Figura 23) como argumento para o método que faz a conexão da camada, que indica como deve ser feita a conexão com a outra camada.

```
typedef struct {
    CLongArray* lSourceIndex;                //Índice do plano na camada fonte
    long lLocalIndex;                        //Índice do plano nesta camada
    CDoubleMatrixArray* pdmConnectionWeight;//Matriz de pesos da conexão
} LayerToLayerConnection;

typedef CArray <LayerToLayerConnection*, LayerToLayerConnection*> Connections;
```

Figura 23 - Estrutura connections

A criação de uma camada já completamente configurada com seus planos e conexões, pode ser feita através da passagem de uma lista com os dados resumidos dos

planos presentes na mesma, incluindo uma lista das conexões desse plano com a camada fonte. Essa estrutura de resumo é descrita na Figura 24.

```
typedef struct {
    CString sName;                          //Nome do plano
    bool bInhibitory;                       //Capacidade inibidora do plano
    long lWidth;                            //Largura do plano
    long lHeight;                           //Altura do plano
    long lNodeViewRange;                    //Raio de visão dos nós
    SPlaneLinkArray* pspla Connections;//Conexões do plano
} ResumoPlano;

typedef CArray <ResumoPlano*, ResumoPlano*> ResumoPlanoArray;
```

Figura 24 - Estrutura de resumo de plano

Mais adiante, essa camada genérica será especializada nas duas camadas do neocognitron, a camada-S e a camada-C. O diagrama de classes da camada pode ser visto na Figura 25.

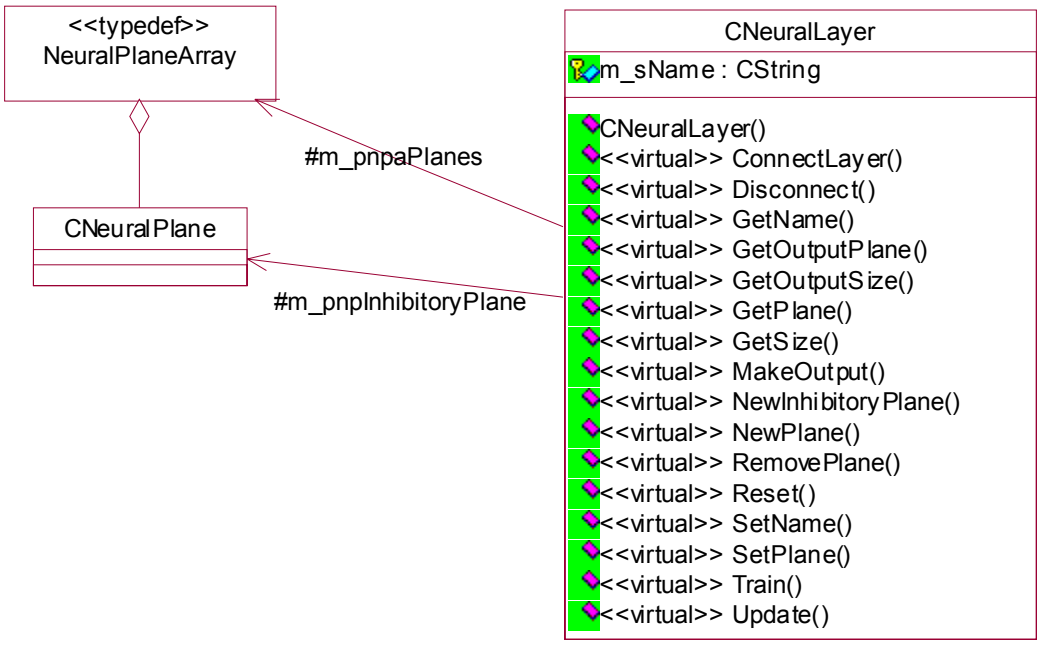


Figura 25 - Diagrama de classes da classe CNeuralLayer e estruturas auxiliares

Classes Derivadas

No neocognitron existem apenas dois tipos de camadas, as camadas-S e as camadas-C, conectando-se em seqüência. Para representar este fato, foram derivadas duas classes da camada descrita anteriormente, de acordo coma Figura 26. Sua funcionalidade será descrita a seguir.

Camada S (CSLayer)

Para a implementação da camada-S do neocognitron foram necessárias pequenas mudanças na classe mãe, como a inclusão do parâmetro q_l (2.4.2.3 - Treinamento dos pesos das células-S) na classe e a implementação do método de treinamento. Como não há a menção de criação dinâmica de planos durante o treinamento na definição da rede, isto não é implementado. A camada-S foi implementada com a classe CSLayer, cuja relação com as outras camadas pode ser vista na Figura 26.

Figura 26 - Hierarquia de classes das camadas derivadas

Camada C (CCLayer)

Para a implementação da camada-C foram necessárias alterações para a implementação da capacidade de redução de tamanho para a saída. Isso foi feito acrescentando-se uma segunda lista de planos, estes agora com planos do tipo CAverageReducingPlane e de métodos para o controle do tamanho de entrada e saída da

camada, através de tamanhos absolutos ou da definição da relação entre os dois tamanhos (diferença entre o tamanho de entrada e o de saída). A camada-C foi implementada com a classe CCLayer, cuja relação com as outras camadas pode ser vista na Figura 26.

5.4.1.4 O Nível (CNeuralLevel)

O nível representa uma nova camada de abstração e um nível mais alto de comando da rede. Para implementá-lo, foi criada uma classe, a CNeuralLevel, que possui comandos de nível mais alto sobre as camadas e planos da rede. O nível, sendo particular ao neocognitron, possui duas camadas, uma camada-S e uma camada-C, interconectadas entre si enquanto a camada-S conecta o nível ao nível anterior e a camada-C é conectada pela camada–S do nível posterior.

O nível possui um mecanismo semelhante ao da camada-C quando se trata do tamanho de entrada e saída do plano, controlando a relação entre os dois tamanhos através do armazenamento e alteração da diferença entre os tamanhos dos planos.

Além disso, possui também métodos para criação e inserção rápida de camadas e planos padrão (a serem configurados após a criação) no nível, o que simplifica as operações de alto nível.

A relação desta classe com as outras já apresentadas pode ser analisada na Figura 27.

Figura 27 - Diagrama de classe da classe CNeuralLevel

5.4.1.5 A Rede (CNeocognitron)

Para tornar a utilização da biblioteca mais fácil, foi construída a classe CNeocognitron, que implementa a rede como um todo, desde a retina até uma camada de saída em separado, com métodos que simplificam o comando da rede e ao mesmo tempo permitem um aprofundamento e controle fino até mesmo de cada nó, ao se navegar pelas estruturas que a compõem.

A rede é composta basicamente por um plano do tipo CRetina, que faz as vezes de retina, uma lista de níveis (objetos da classe CNeuralLevel) e uma camada simples de saída, um objeto do tipo CNeuralLayer com planos de apenas um nó. Sua estrutura pode ser vista na Figura 28.

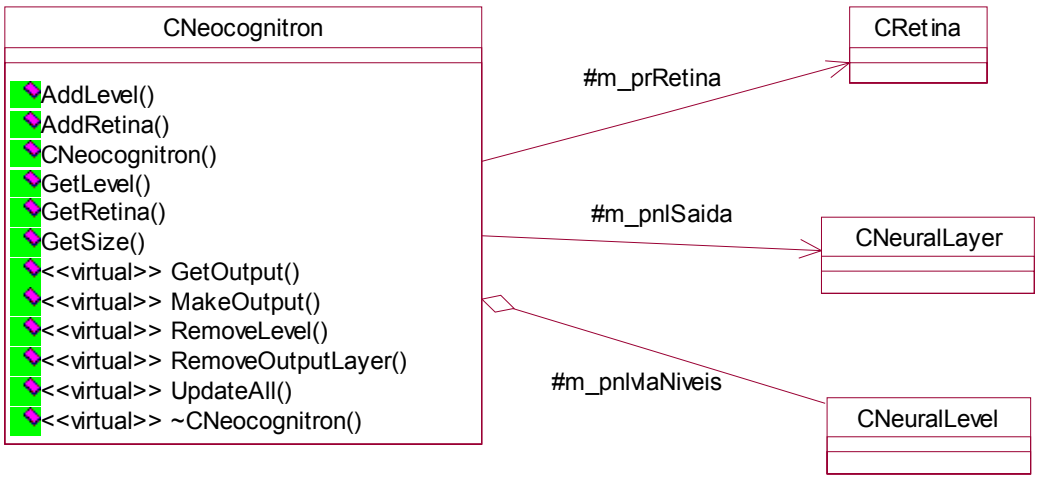


Figura 28 - Diagrama de classe da classe CNeocognitron

5.4.2 Visualizador

O visualizador é a segunda parte da biblioteca de classes, possuindo a capacidade de mostrar os dados gerados pelo simulador com desenhos e de alterar suas configurações com janelas de diálogo, fazendo assim o papel de interface.

Como dito anteriormente, o simulador foi implementado com base em um sistema de partículas, e o visualizador não é diferente. A estrutura do visualizador se assemelha à do simulador, existindo uma classe no visualizador para cada classe no simulador. Isso ocorre porque as classes do visualizador foram criadas para se "acoplarem" às classes do simulador, havendo em tempo de execução um objeto do visualizador para cada objeto do simulador. Essa estrutura permite que a parte da visualização possa ser trocada em tempo de execução, bastando para isso trocar os objetos do visualizador por objetos que implementem uma visualização diferente.

Esse acoplamento ocorre devido a um atributo que todas as classes do visualizador possuem, que indica qual é a classe de dados dela. Ao utilizar o método SetData() das classes, a classe visual passa a receber informações desse objeto de dados (proveniente do simulador) e a se comportar e construir de acordo com os dados recebidos.

Todas as classes do visualizador (com exceção das classes que implementam as janelas de diálogo) vêm de uma única classe mãe, que determina a interface comum a que todas têm acesso. Utilizando essa interface, uma aplicação que utilize este

visualizador pode obrigar qualquer parte dele a se desenhar ou exibir a janela de diálogo, sem se importar com qual o tipo de objeto está trabalhando.

Todas as classes possuem três métodos principais: SetData(), que associa o objeto da classe visual a um objeto do simulador; Draw(), que executa a função primária do objeto, que é se desenhar na tela; e ShowDetails(), que exibe a janela de diálogo da classe, repassando os dados do objeto do simulador para a tela e alterando-os conforme o usuário interage com a janela. A seguir serão documentadas as classes presentes no visualizador.

5.4.2.1 CGraphicObject

Esta é a classe mãe para todas as classes do visualizador, servindo como uma interface comum a todas. Além dessa interface, ela é que armazena dados como a posição do objeto e do seu tamanho, entre outras informações. Muitos de seus métodos não são implementados, servindo apenas de interface. A Figura 29, resume a hierarquia abaixo desta classe.

5.4.2.2 CVisualNode

Partícula básica de todo o sistema de partículas, ela se acopla a nós da hierarquia da classe CNeuralNode através do método SetData(), já citado. Ele se desenha na tela como uma esfera, de cor vermelha para saídas positivas dos nós, azul para saídas negativas e cinza para saídas iguais a zero. A intensidade da cor varia de acordo com o valor, quanto maior o valor maior a intensidade.

Esta classe possui um registro estático (ou seja, existe apenas uma cópia desse atributo para todos os objetos da classe) de todos os nós pertencentes a ela e com quais nós do simulador eles se acoplam, eliminando assim a necessidade de uma cópia da estrutura do CNeuralNode com suas conexões.

Os objetos dessa classe também armazenam a localização de um objeto de uma das classes de janelas de diálogo, o tipo dependendo do construtor da classe de nó. Isso permite uma flexibilidade de interface para a classe.

Na Figura 29 pode ser observada a hierarquia de classes dos nós visuais. A Figura 30 exibe a janela de diálogo da classe.

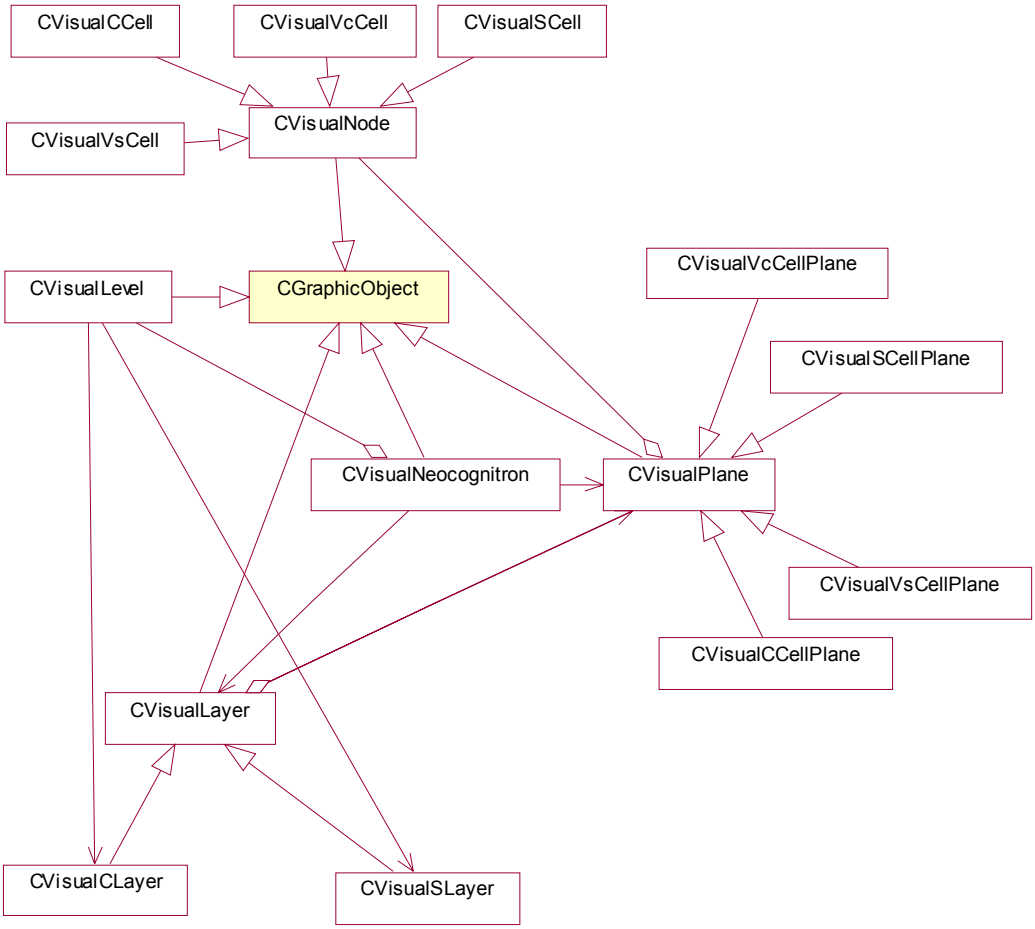


Figura 29 - Hierarquia resumida das classes visuais

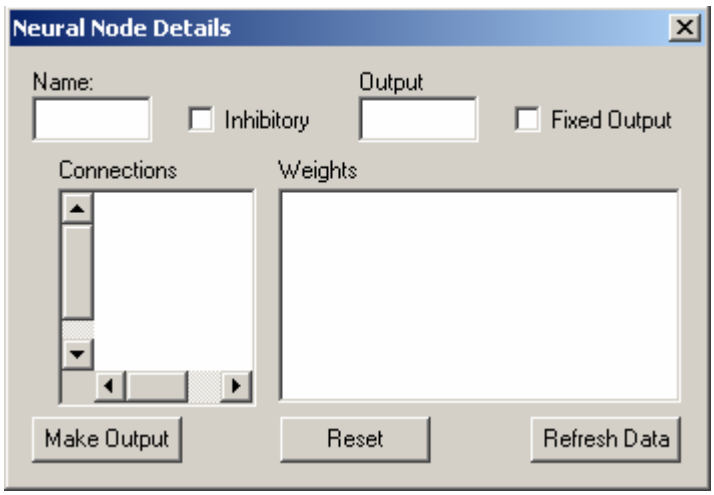


Figura 30 - Janela de diálogo da classe CVisualNode

Classes Derivadas

CVisualCCell

Nós desta classe se conectam às céluas-C do neocognitron, e possuem um funcionamento muito semelhante à classe mãe, com a diferença de que eles levam em conta a existência do nó inibidor nestas classes, e desenham a conexão corretamente. Outro detalhe diferente é que a classe de janela de diálogo com o qual o nó é construído é diferente da classe mãe, fazendo assim a personalização dos detalhes do nó (Figura 31).

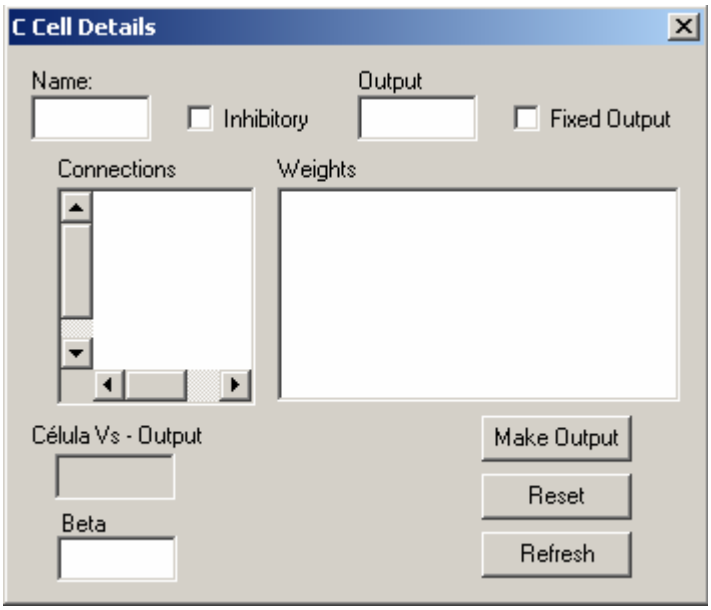


Figura 31 - Janela de diálogo da classe CVisualCCell

CVisualSCell

Como na classe CVisualCCell, as diferenças são a representação gráfica da conexão inibidora, não presente na classe mãe, e uma janela de diálogo particular a essa classe, mostrada na Figura 32.

CVisualVcCell e CVisualVsCell

Com ainda menos modificações que as classes derivadas anteriores, estas classes diferem da classe mãe somente pela janela de diálogo, que possui o mesmo modelo para as duas, igual ao da classe mãe, sendo modificado apenas o texto do título da mesma e a forma de preenchimento dos detalhes.

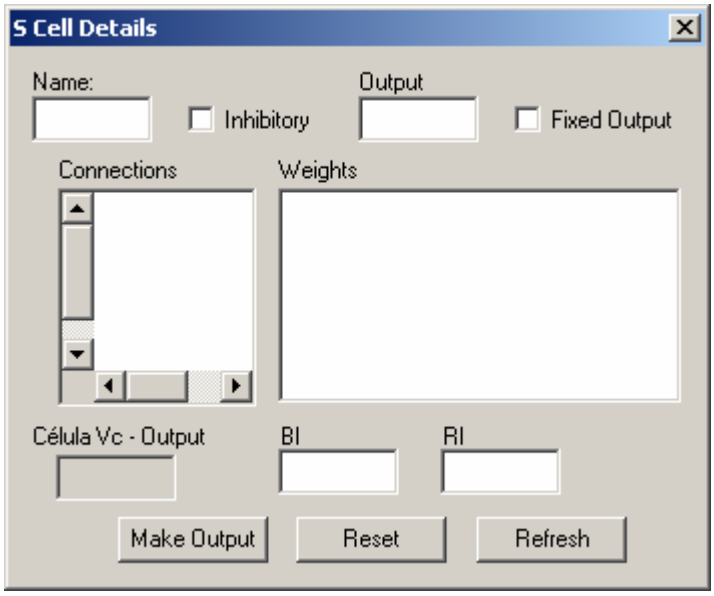


Figura 32 - Janela de diálogo da classe CVisualSCell

5.4.2.3 CVisualPlane

Armazena um conjunto de nós visuais (da classe CVisualNode), representando um plano da hierarquia da classe CNeuralPlane. Por possuir uma visão de todos os nós do plano, possui métodos para normalizar os valores dos nós antes de desenha-los, de modo a representar melhor a diferença de valores. O plano é representado por um quadrilátero envolvendo o conjunto de nós, dando uma maior sensação de agrupamento. A maior parte de seus métodos se resume a manipular diversos nós de uma única vez, disseminando as mudanças feitas no plano para os nós.

Para flexibilizar a classe, quando necessária a criação de um nó é utilizado o método NewNode(), que, ao ser sobrecarregado nas classes derivadas, permite a criação de qualquer tipo de nó da hierarquia da classe CVisualNode. Os planos visuais também possuem objetos de janela de diálogo próprios, para a manipulação dos dados do plano a que são acoplados. A hierarquia de classes dos planos visuais pode ser vista na Figura 29, enquanto que o modelo de janela de diálogo pode ser vista na Figura 33.

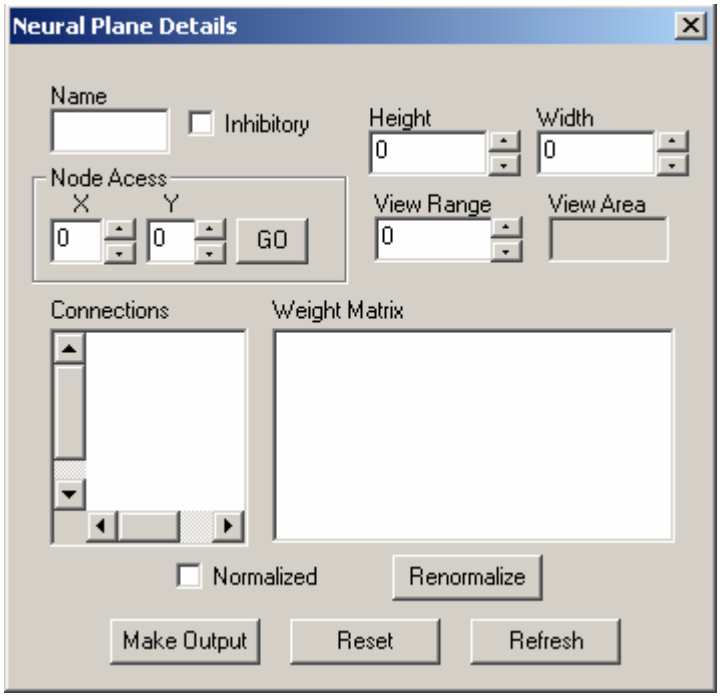


Figura 33 - Janela de diálogo da classe CNeuralPlane

Classes Derivadas

Em todas as classes derivadas do plano visual (**CVisualCCellPlane, CVisualSCellPlane, CVisualVcCellPlane e CVisualVsCellPlane**), foi somente necessário sobrecarregar o método NewNode(), para que seu comportamento fosse alterado através de seus novos nós, e o construtor, para que o objeto fosse construído com a janela de diálogo adequada ao plano que representaria. Apesar de possível, não foi necessária mais nenhuma alteração.

5.4.2.4 CVisualLayer

Armazena um conjunto de planos visuais (da classe CVisualPlane), representando um plano da hierarquia da classe CNeuralLayer.

A camada é representada por um quadrilátero envolvendo o conjunto de planos, com um segmento de reta separando os planos regulares do plano inibidor.Além disso, o plano inibidor se encontra um pouco à frente da camada, representando o fato de que ele é processado antes do resto dos planos. Os planos regulares se encontram um pouco atrás da camada, para podermos perceber com clareza a conexão de seus nós ao plano inibidor.

A maior parte de seus métodos se resume a manipular diversos planos de uma única vez, disseminando as mudanças feitas na camada para os planos. Para flexibilizar a classe, quando necessária a criação de um plano é utilizado o método NewPlane() ou NewInhibitoryPlane(), que, ao serem sobrecarregados nas classes derivadas, permitem a criação de qualquer tipo de plano da hierarquia da classe CVisualPlane. As camadas visuais também possuem objetos de janela de diálogo próprios, para a manipulação dos dados da camada a que são acopladas.

A hierarquia de classes dos planos visuais pode ser vista na Figura 29, enquanto que o modelo de janela de diálogo pode ser vista na Figura 34.

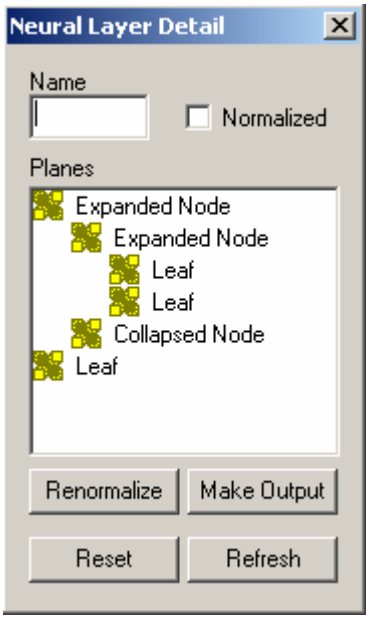


Figura 34 - Janela de diálogo da classe CVisualLayer

Classes Derivadas

As classes derivadas das camadas visuais precisam, a exemplo dos planos visuais, sobrecarregar somente os métodos construtores, de modo a construir o tipo correto de janela de diálogos e os métodos que criam os planos, NewPlane e NewInhibitoryPlane(), de modo a preencher a camada corretamente. O resto do trabalho é feito pelo método SetData() da classe mãe.

Desse modo foi implementada a classe **CVisualSLayer**, utilizada para representar a camada-S (CSLayer) do neocognitron.

Para a implementação da camada-C, devido a seus planos redutores, foi necessária a criação do atributo que armazenaria esta lista de planos, deixando a classe

um pouco mais complexa, e portanto, exigindo uma documentação em separado, a seguir.

CVisualCLayer

A camada-C, ao necessitar de um segundo conjunto de planos, exigiu uma representação um pouco diferente da camada padrão. Ela continua sendo representada por um quadrilátero dividido em setores de planos regulares e plano inibidor, com o plano inibidor à frente e os planos mais atrás. Atrás de cada plano regular está posicionado seu plano redutor, unido a ele por planos semitransparentes que se originam de suas bordas. Isso dá aos planos um efeito de afunilamento, o que reforça a idéia de redução de tamanho dos planos-C da camada.

Após a criação da lista de planos redutores, foi necessária a sobrecarga dos métodos SetData(), Draw() e NewPlane(), de modo a controlarem também a lista de planos redutores.

5.4.2.5 CVisuaLevel

Os níveis visuais, a exemplo dos níveis do simulador, armazenam duas camadas em seu interior, uma camada-S e uma camada-C.

Os níveis são representados por paralelepípedos envolvendo todo o seu conteúdo, sem divisões. As camadas-S (CVisualSLayer) apresentam-se à frente, tendo após elas as camadas-C.

Também como no caso dos planos do simulador, esta classe age mais como um facilitador de controle do que uma parte específica da representação. Seus métodos disseminam mudanças por todos os seus componentes, como mudanças de posição, mudança de fonte de dados (objetos da classe CNeuralLevel), etc.

Como as outras estruturas visuais, o nível visual também possui um tipo de janela de diálogo específico, que permite o acesso a seus dados pelo usuário.

Enquanto a posição da classe na hierarquia de objetos visuais pode ser vista na Figura 29, a janela de diálogo associada a ela pode ser vista na Figura 35.

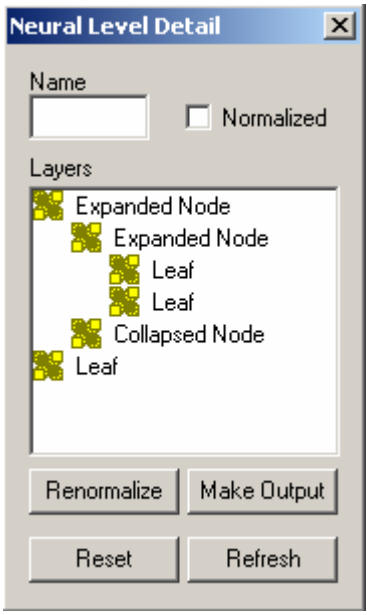


Figura 35 - Janela de diálogo da classe CVisualLevel

5.4.2.6 CVisualNeocognitron

Esta classe visual é o nível mais alto de controle da visualização. A rede completa é envolvida em um paralelepípedo para a representação gráfica de um todo conciso e completo.

Esta classe tem estrutura similar ao neocognitron do simulador, contendo a retina, a lista de níveis e a camada de saída. Todas essas estruturas são criadas quando se associa um objeto desta classe com um objeto da classe CNeogonitron do visualizador através do método SetData().

Como no caso do nível, todas as alterações feitas neste nível são propagadas pela rede inteira, alterando seus elementos de acordo com a mudança.

A rede visual não possui janela de diálogo, uma vez que age como um aglomerado de outros objetos visuais, possuindo pouca funcionalidade além do controle desses objetos. Sua posição na hierarquia de classes pode ser vista na Figura 29.

5.4.2.7 Classes de Janelas de Diálogo

Para criar uma interface mais intuitiva para a alteração de parâmetros das classes do simulador, foram criadas janelas de diálogo. Os modelos foram criados no próprio ambiente de desenvolvimento do Visual C++ e associados a classes de controle.

Como a implementação da classe CDialog (classe básica da biblioteca MFC para controle de caixas de diálogo) permite somente a implementação de janelas de exibição modal (i.e. só se pode voltar a manipular a aplicação após o fechamento da mesma), foi criada uma classe, derivada da CDialog, que permitisse a apresentação modeless (i.e. pode-se trocar o foco entre a aplicação e a janela sem problemas).

Essa Classe, a CbaseModelessDialog, além de implementar esse modo de exibição, permite que suas classes filhas alterem seu modelo de caixa de diálogo e fornece uma interface comum entre elas, onde define um atributo que aponta para o alvo da janela, ou seja, pode-se modificar a fonte de dados que preenche os campo da janela em tempo de execução.

A partir dela, foram geradas diversas classes que implementam o controle das caixas de diálogos apresentadas nas sessões acima, junto às classes visuais. O diagrama de classes dessas classes é apresentado abaixo, na Figura 36.

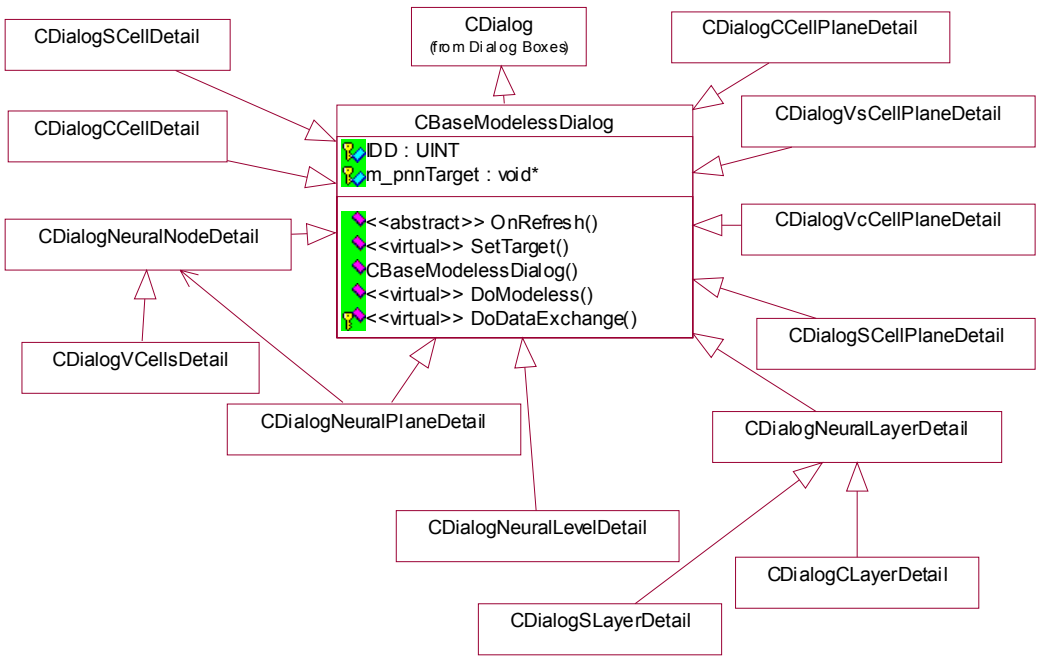


Figura 36 - Hierarquia de classes das classes de janelas de diálogo

5.5 A Ferramenta

Uma vez definida a biblioteca, a ferramenta de visualização passa a ser somente uma interface para o uso da primeira, necessitando apenas de que se crie liberdade para a manipulação das duas partes da biblioteca.

Para a implementação dessa interface, foi utilizada a arquitetura Document/View da biblioteca MFC, onde os dados são armazenados em uma classe de documento e a classe de visão gera a interface com a tela. Essa arquitetura, já muito parecida com a arquitetura da biblioteca, deixa claro onde cada parte da biblioteca deve ficar.

Outro recurso utilizado da biblioteca MFC foi a arquitetura de aplicações MDI (Multiple Documents Interface), que permite a manipulação de diversos documentos de uma única vez, o que permite, no caso, criar diversas redes do tipo neocognitron e observa-las ao mesmo tempo, na mesma aplicação.

Para a implementação, foi declarada uma classe de documento, que contém como dado uma rede neural da classe CNeocognitron, que fica à disposição da interface para ser manipulada. Na classe de visão, foram implementadas rotinas de tratamento de mensagens do sistema operacional, para a manipulação do mouse, teclado, menus e janelas de diálogo.

Nessa classe de visão foi configurado o ambiente da biblioteca gráfica OpenGL, deixando-o pronto para que os objetos visuais sejam desenhados sem maiores configurações. Foi utilizada uma projeção perspectiva para o desenho, de modo a criar a sensação de profundidade.

Através do sistema de nomes do OpenGL, é possível, com o auxílio do mouse, selecionar uma área na tela e identificar qual estrutura visual foi selecionada. Utilizando-se disso, foi criado um sistema de menus pop-up que responde diferentemente a cada objeto selecionado com o botão direito do mouse.

Quando a seleção é feita com o botão esquerdo, o objeto visual selecionado exibe sua janela de diálogo, tornando assim possível alterar os aspectos do simulador.

A manipulação do ângulo de visão se dá também através do mouse ou do teclado. Quando não seleciona nenhum objeto, o botão esquerdo do mouse cuida da translação da imagem nos eixos X e Y (X e Z caso a tecla T seja mantida pressionada). Enquanto estiver pressionado, o movimento do mouse é traduzido em comandos de translação para a ferramenta. No caso do botão direito não selecionar nada ele controla as rotações da câmera, nos eixos X e Y (X e Z caso a tecla T seja pressionada) de modo semelhante à translação.

5.6 Resultados Obtidos

A ferramenta implementada permite uma boa manipulação do espaço 3D aonde se encontra a visualização, permitindo um controle intuitivo das translações e rotações do espaço, como descrito acima.

A seguir, são apresentados alguns exemplos da funcionalidade da ferramenta, demonstrando a criação e treinamento de uma rede neocognitron para o reconhecimento de dois caracteres.

5.6.1 Criação da rede

A rede criada segue o exemplo dos primeiros testes com o neocognitron, feitos por Fukushima [11]. É criada uma rede com três níveis, cada camada possuindo 5 planos (no teste original, a rede reconhecia 5 padrões, e portanto possuía mais planos – 24 por camada).

A entrada da rede tem o tamanho de 16x16 pixels, e o tamanho dos planos vai reduzindo de acordo com a sua posição na rede (Planos na camada Us1 têm 16x16 de tamanho, os da camada Uc1 têm 10x10, os da camada Us2 têm 8x8, da camada Uc2 têm 6x6, os da camada Us3 têm 4x4 e os da camada Uc3 têm 1x1).

A Figura 37 mostra uma visualização da rede criada. Para construir a rede, foi criado um procedimento que a construía completamente, para não ser necessária a construção através de menus popup. Esse procedimento consiste de utilizar os métodos da classe CNeocognitron e adicionar uma retina (com uma imagem de tamanho 16x16 em branco), configurar cada nível com camadas criadas automaticamente e adiciona-los à rede.

Mesmo uma rede pequena como essa gera muitos nós, o processo levando em média 6 segundos para se completar. Aumentando o número de planos para 10 em cada camada, esse tempo sobe para 25 segundos.

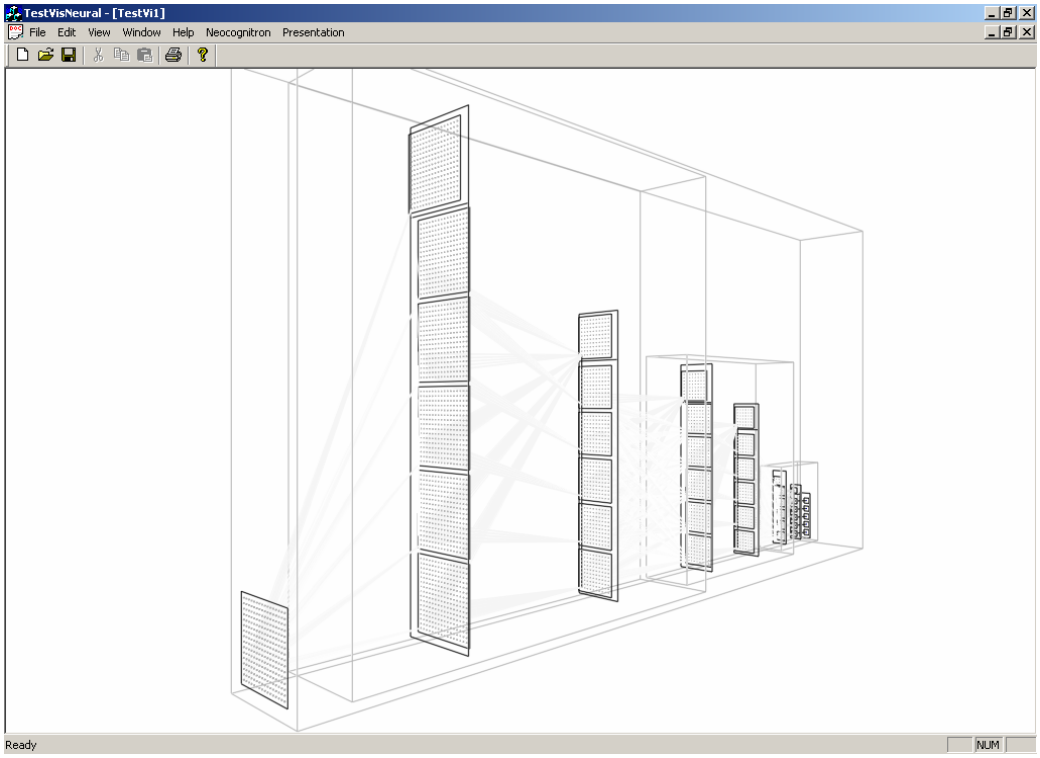
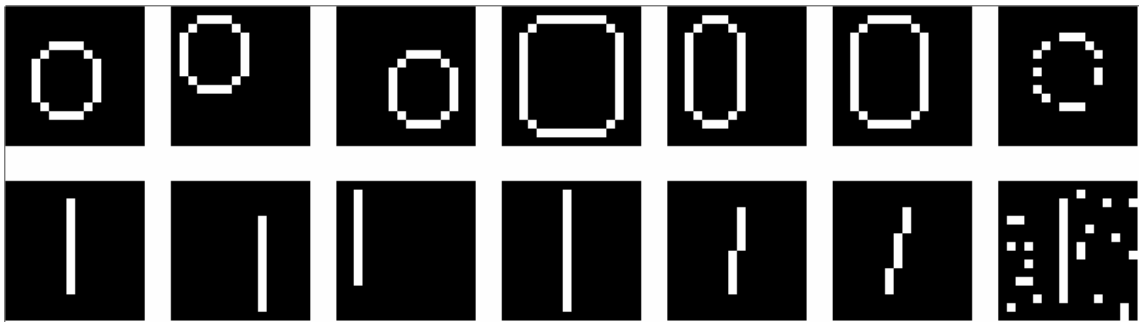


Figura 37 - Imagem da rede criada para o teste

5.6.2 Treinamento da Rede

O próximo passo após a criação da rede é seu treinamento. Para tanto, foram utilizados os padrões demonstrados na Figura 38. Os padrões de (a) a (c) apenas modificam a posição do padrão na retina, e foram utilizados para o treinamento. Os padrões (d) a (g) representam deformações nos padrões originais, e foram utilizados para reconhecimento, como mostrado adiante.



(a) (b) (c) (d) (e) (f) (g)

Figura 38 - Padrões de treinamento e teste

Para o treinamento, as imagens de (a) a (c) foram exibidas à rede repetidas vezes, totalizando um número de 40 exibições de imagens de cada padrão. Um exemplo do

resultado obtido com o treinamento da rede após 20 exibições do primeiro padrão pode ser vista na Figura 39.

A Figura 40, mostra as caixas de diálogo com a saída da rede. Cada um dos planos de saída é representado.

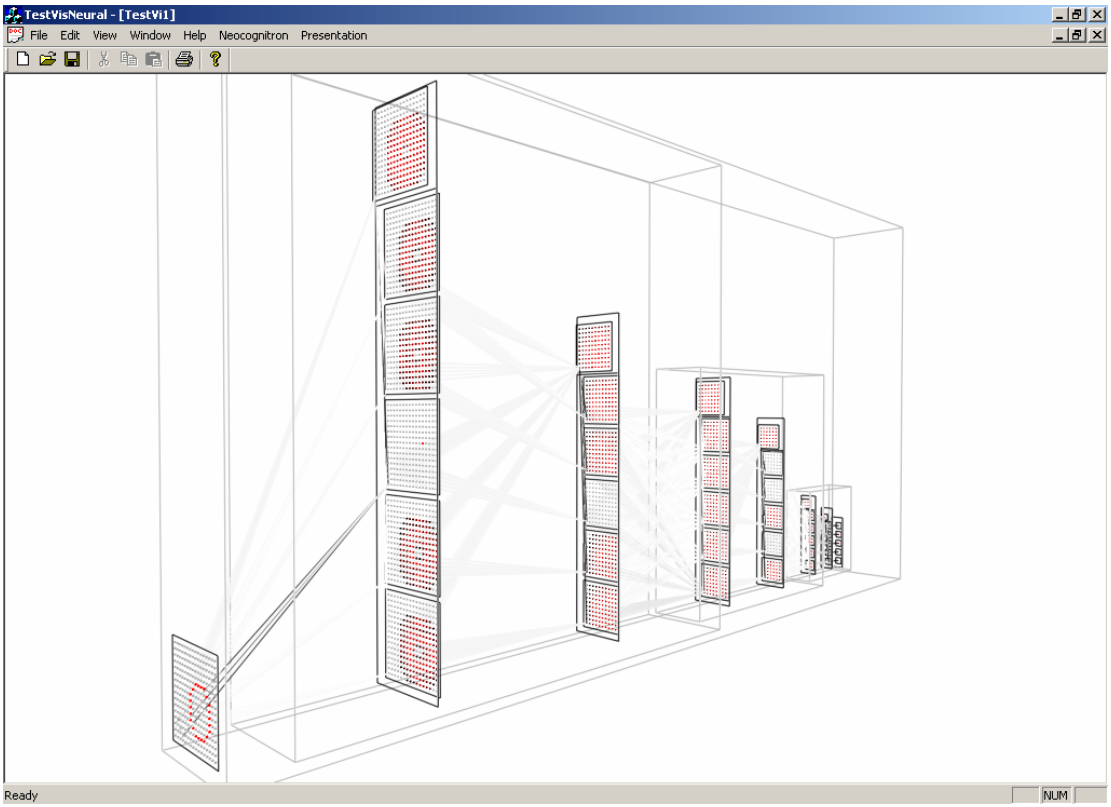


Figura 39 - Rede após treinamento com o primeiro padrão

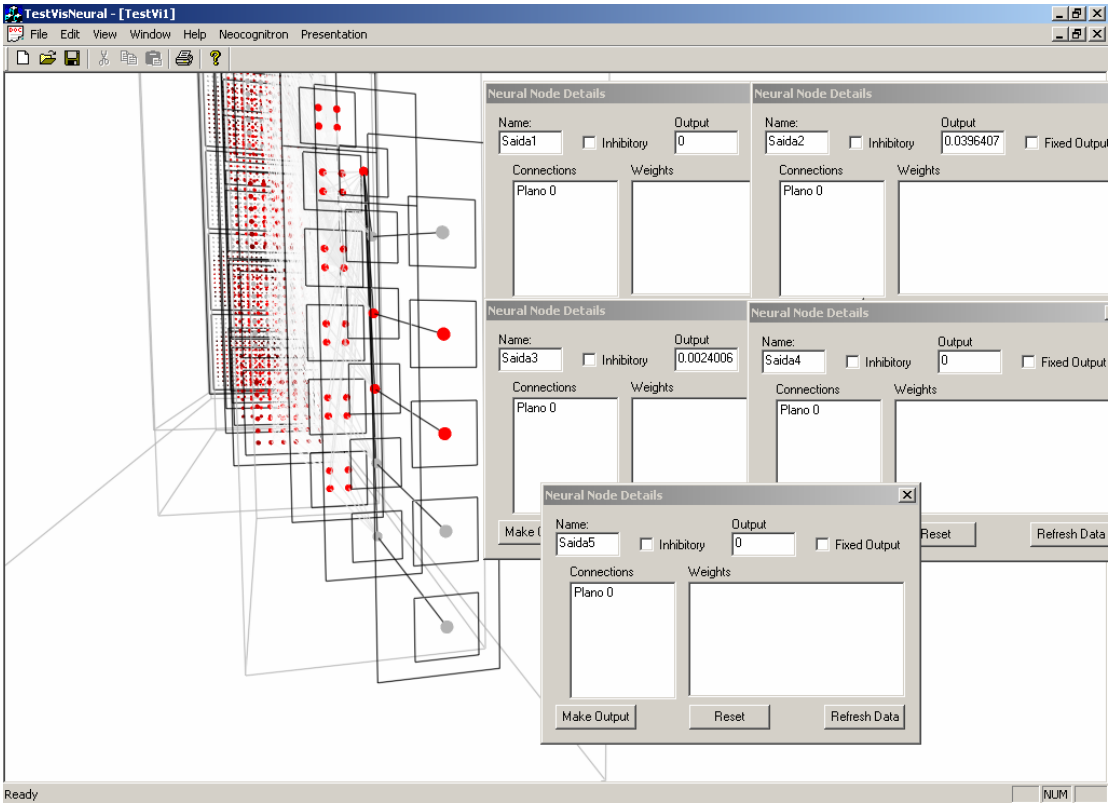


Figura 40 - Saída da rede após treinamento com o primeiro padrão

Com a apresentação do resto das imagens à rede, ela foi treinada para diferenciar os dois padrões mostrados. O resultado desse treinamento pode ser visto na Figura 41, com sua saída demonstrada na Figura 42.

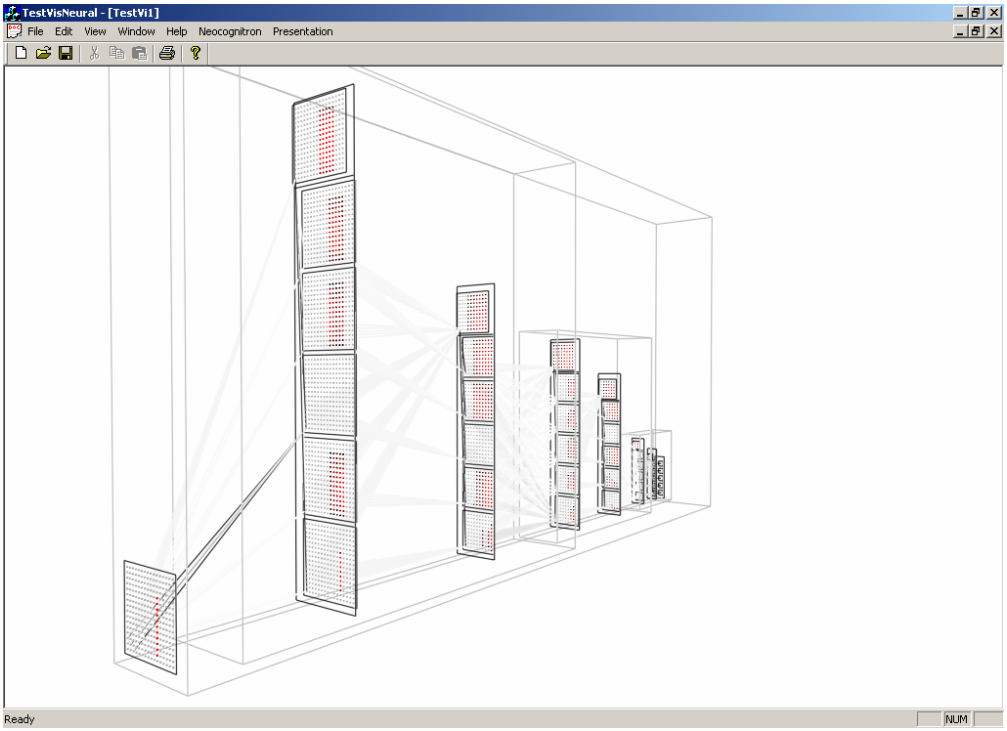


Figura 41 - Rede após treinamento completo, exibindo o segundo padrão

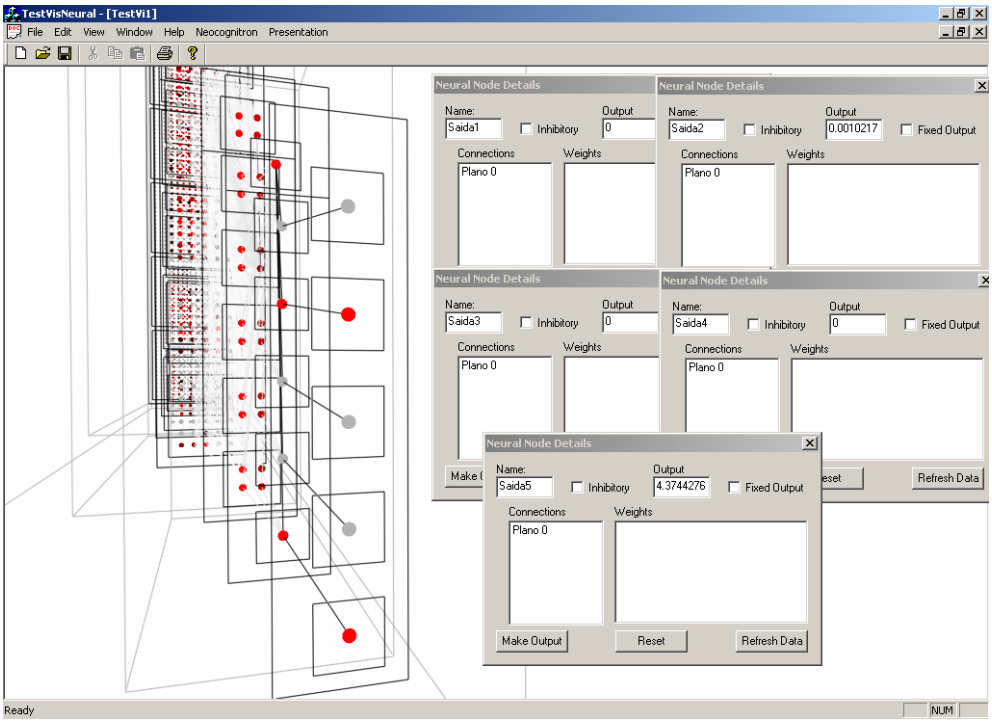


Figura 42 - Saída da rede após treinamento completo, exibindo o segundo padrão

5.6.3 Teste da Rede

Após esse treinamento, como teste, foram apresentadas à rede as imagens (f) e (g) do primeiro padrão e as imagens (e) e (f) do segundo padrão, para realizar o reconhecimento.

O resultado foi satisfatório, como demonstrado na seqüência de imagens Figura 43 a Figura 50 a seguir, demonstrando a rede e as saídas da rede para cada imagem apresentada à ela.

Esse resultado mostra que o simulador foi implementado corretamente, e que a visualização gerada é significativa, apesar de serem necessários alguns ajustes, como a normalização das cores dos nós por camada, e não por plano, como visto na camada de saída, onde todos os nós possuem a mesma cor, independente do valor de saída.

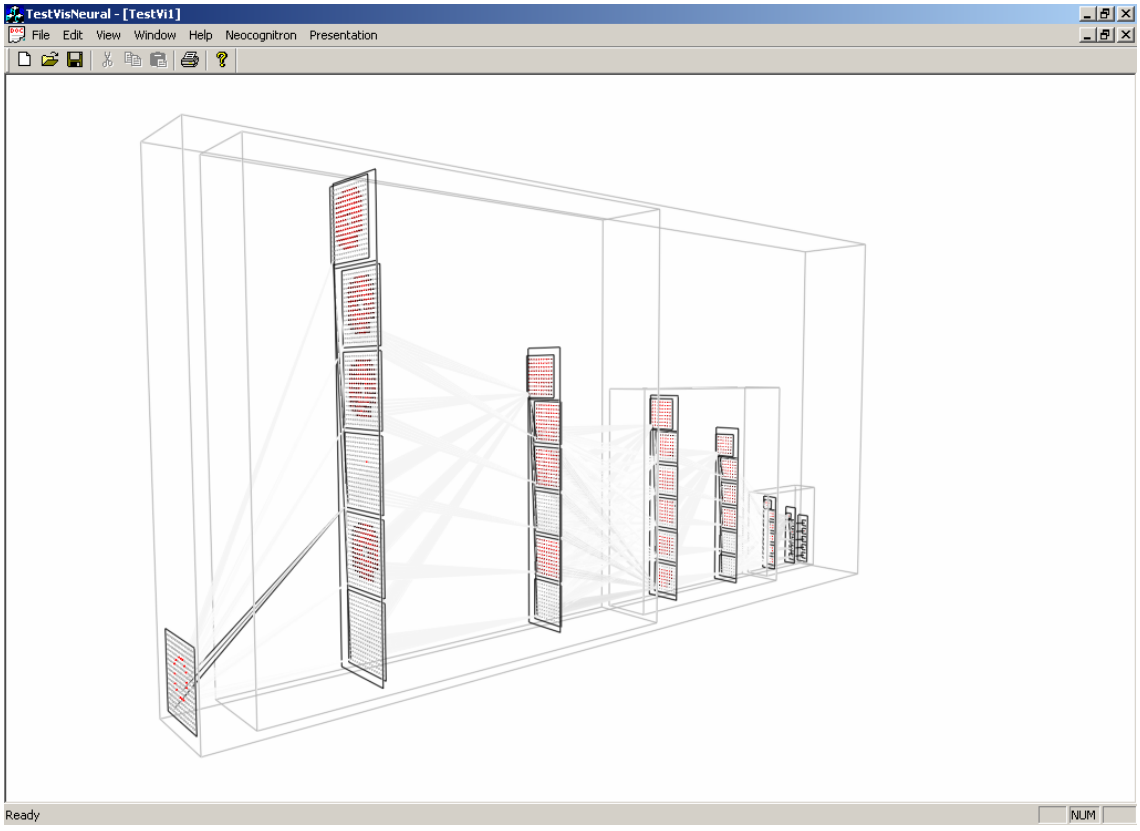


Figura 43 - Rede reconhecendo a imagem (g) do primeiro padrão

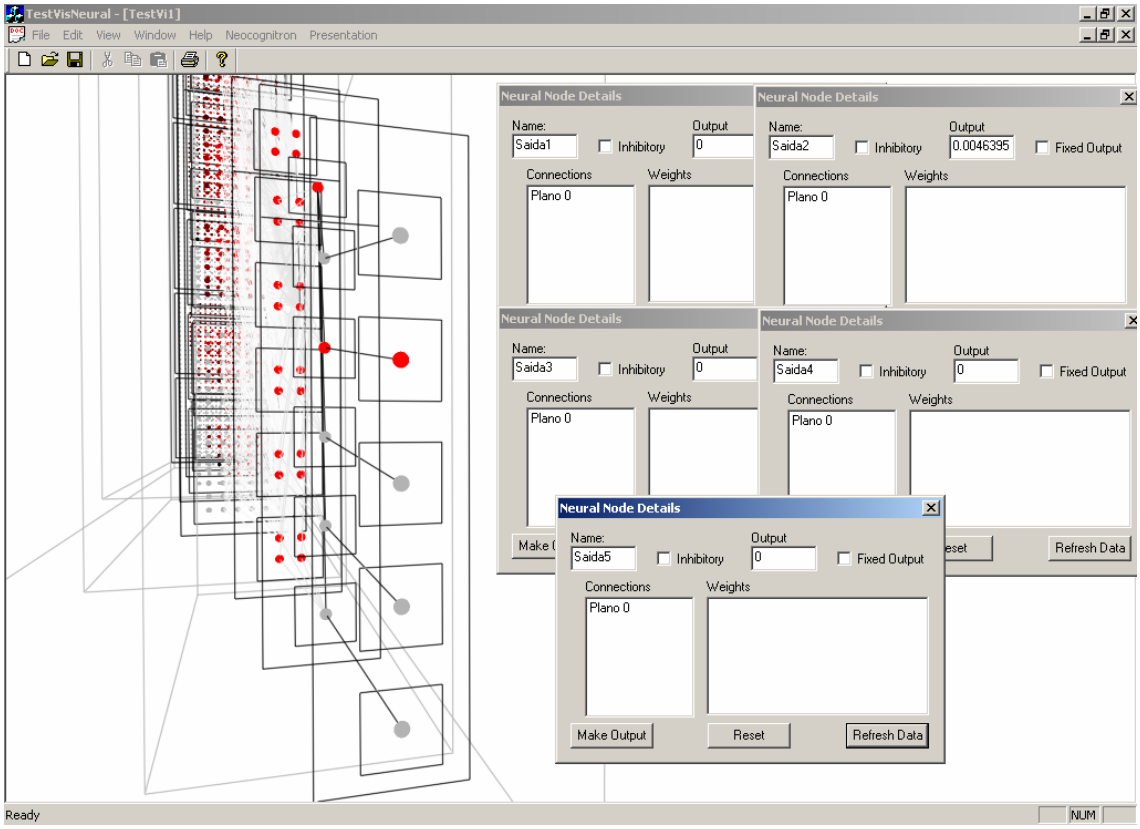


Figura 44 - Saída do reconhecimento da imagem (g) do primeiro padrão

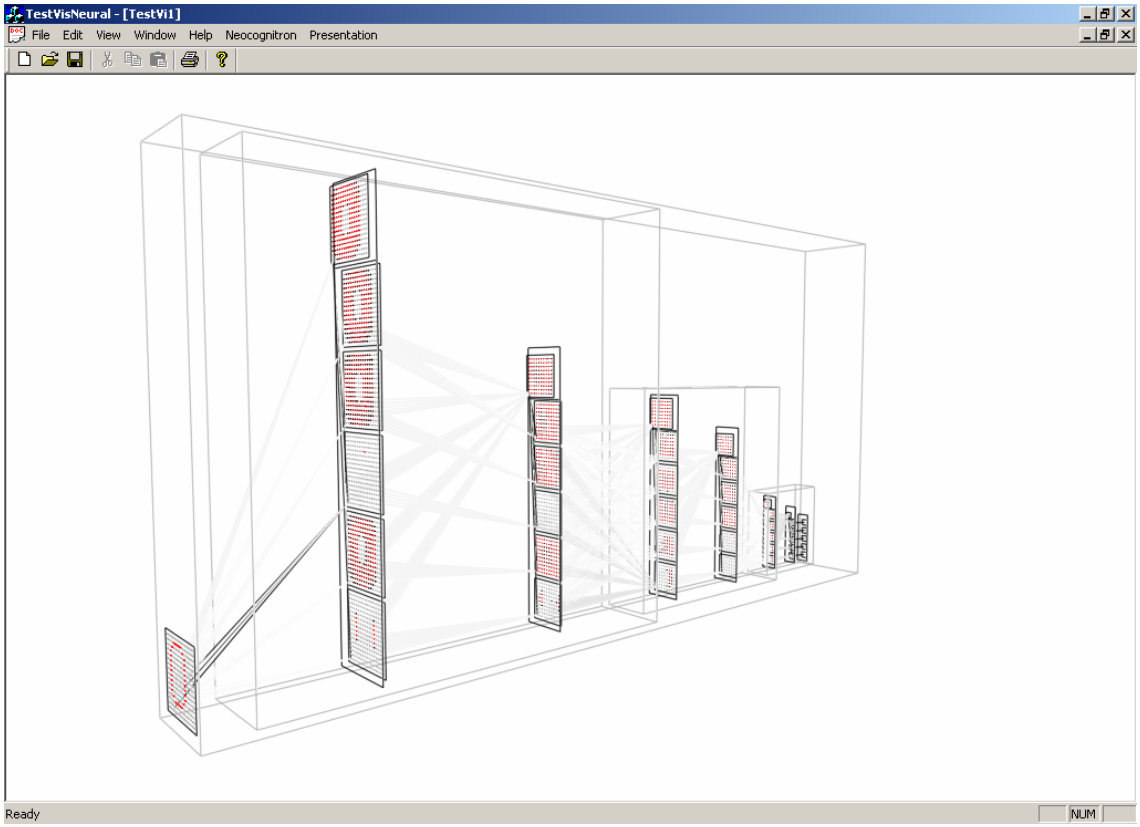


Figura 45 - Rede reconhecendo a imagem (f) do primeiro padrão

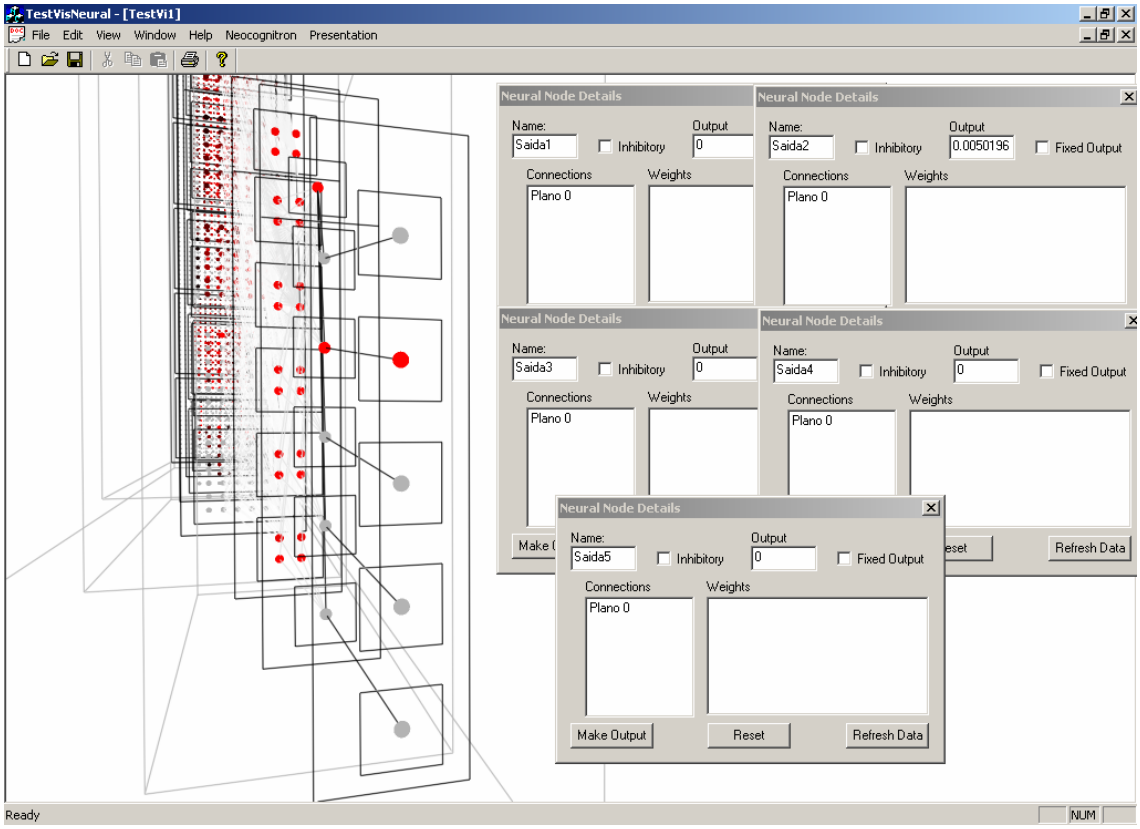


Figura 46 -Saída do reconhecimento da imagem (f) do primeiro padrão

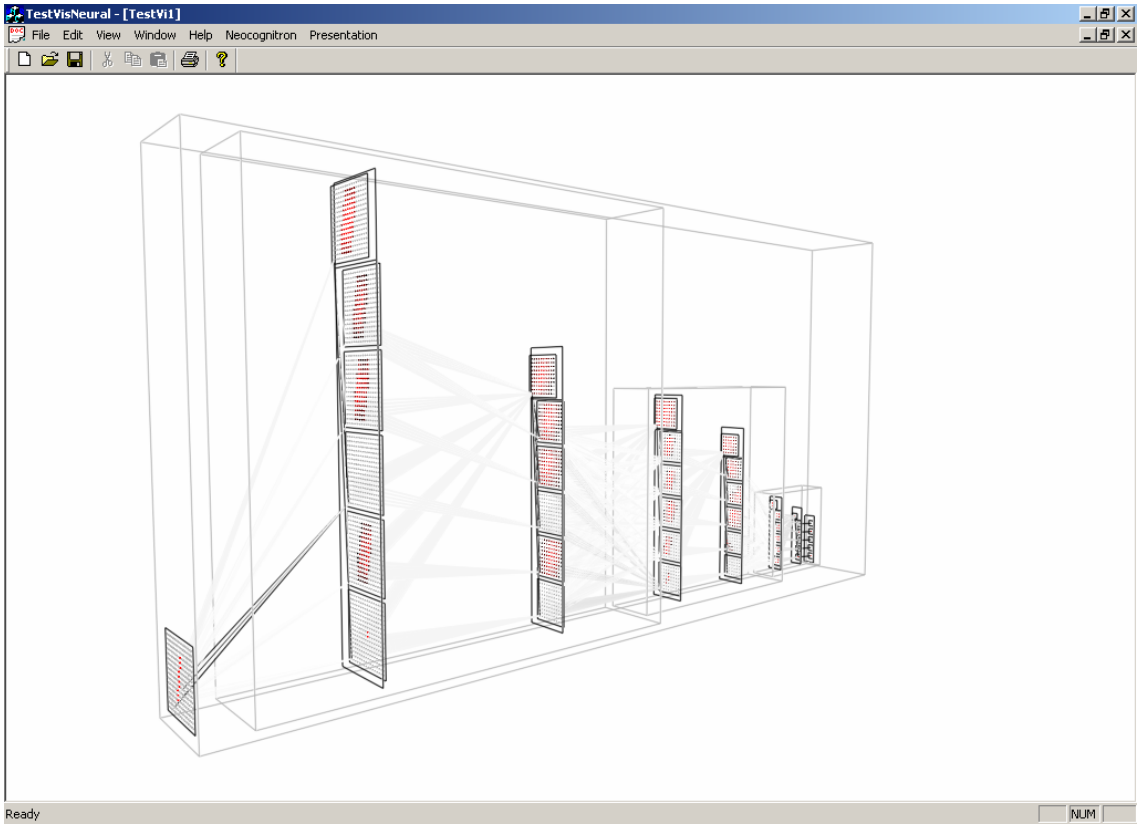


Figura 47 - Rede reconhecendo a imagem (f) do segundo padrão

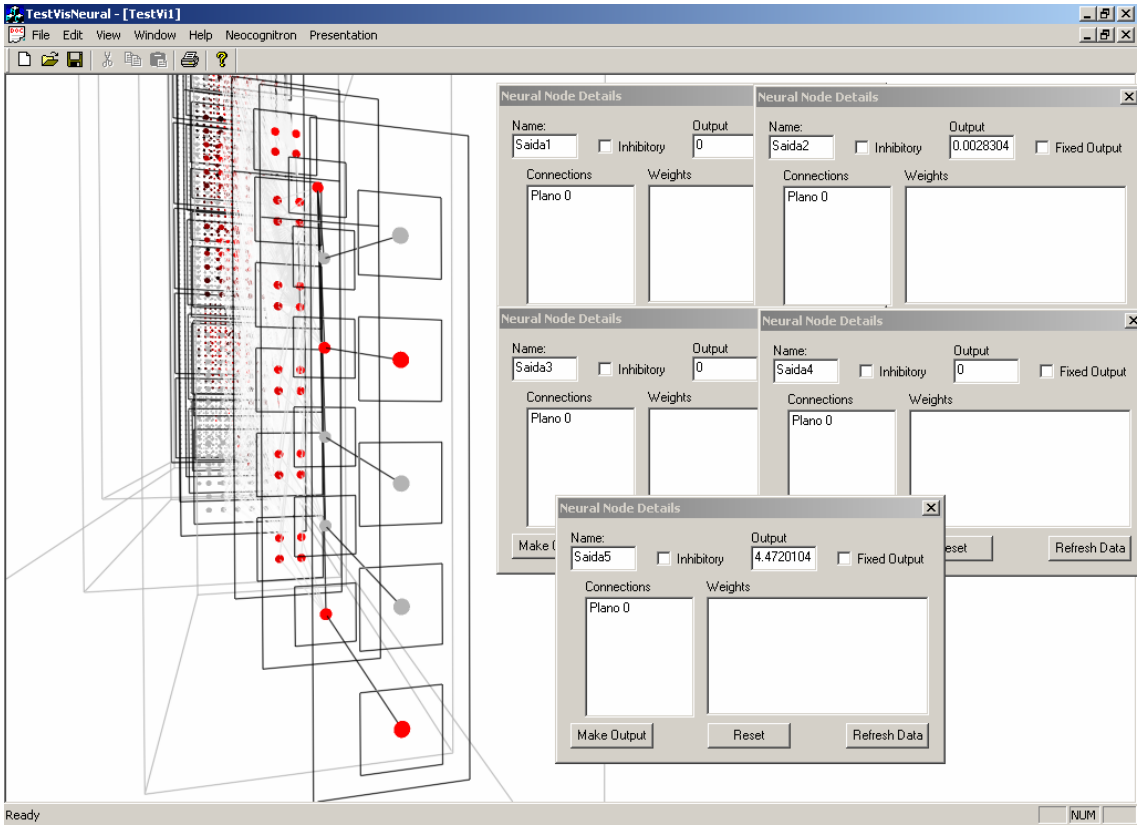


Figura 48 - Saída do reconhecimento da imagem (f) do segundo padrão

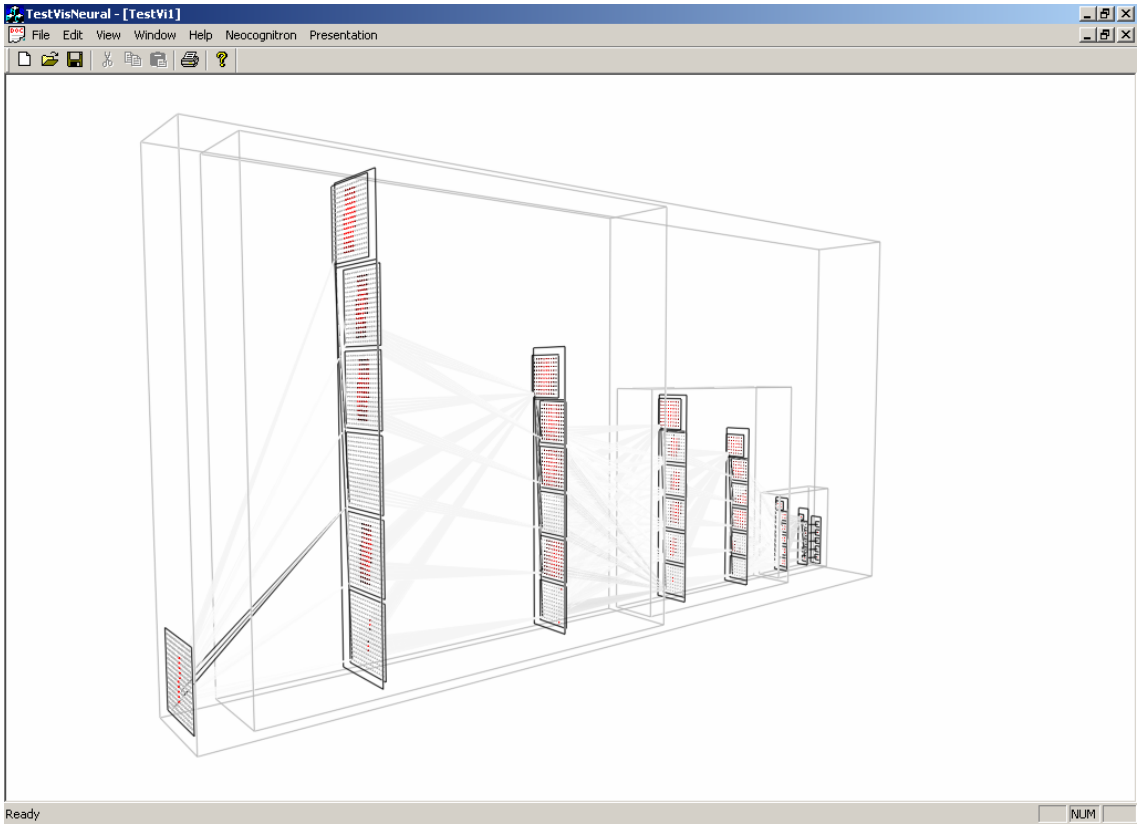


Figura 49 - Rede reconhecendo a imagem (e) do segundo padrão

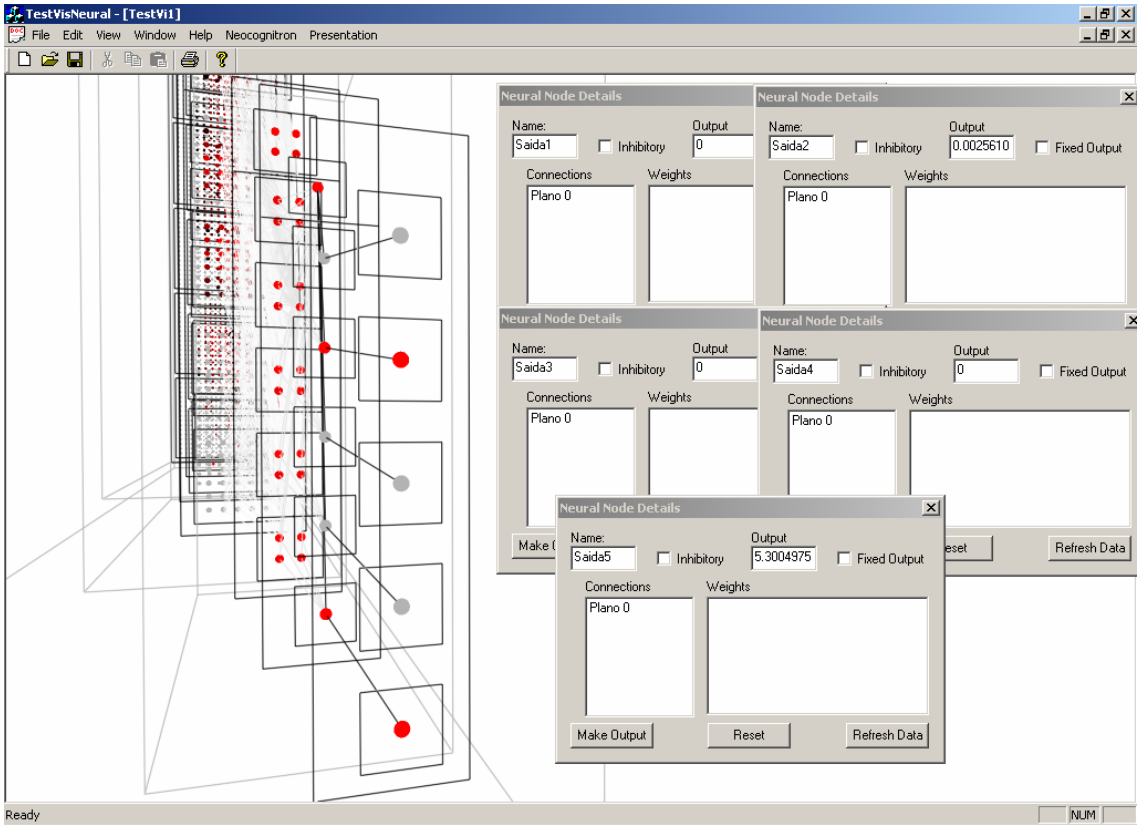


Figura 50 - Saída do reconhecimento da imagem (e) do segundo padrão

6 Conclusão e Propostas de Trabalhos Futuros

6.1 Conclusão

Devido à necessidade de possuir uma grande flexibilidade, a biblioteca tornou-se um pouco lenta, não sendo otimizada para a simulação da rede neural e nem para a apresentação gráfica, gerando um grande número de objetos que pode deixar o computador que a executa lento se houver pouca memória.

Outra causa para lentidão foi a própria rede neural escolhida, que gera um imenso número de nós, que associado à falta de otimização do simulador causa um atraso muito grande no caso de entradas um pouco maiores do que meros testes.

Apesar desses problemas, a estrutura criada para a biblioteca permite que cada uma das partes dela (simulador e visualizador) seja aprimorada e otimizada em separado, desde que se mantenham as interfaces intactas, permitindo assim que trabalhos futuros tornem praticáveis a implementação e estudo de redes maiores.

Outra grande vantagem da biblioteca é que ela torna possível, devido à sua modularidade, a implementação de outros tipos de redes neurais alterando-se somente o simulador, ou de outros tipos de visualização da mesma rede, alterando-se o visualizador.

Essas alterações devem ser feitas em sua maioria com criação de novas classes que especializam as já documentadas, sendo necessária a sobrecarga de poucos métodos para uma grande mudança do comportamento da rede.

6.2 Trabalhos Futuros

A ferramenta gerada para a utilização da biblioteca não foi concluída completamente, faltando a implementação de alguns procedimentos, como impressão de imagens, gravação em arquivo da rede criada e processamento em lote, que criaria animações do funcionamento da rede. Apesar de não concluídos, essas alterações são de simples implementação através da biblioteca MFC, que possui facilidades para impressão e gravação em arquivos.

A interface da ferramenta pode ainda ser melhorada, com uma melhor escolhe de menus e de janelas de diálogo, bem como uma flexibilização da mesma, de modo a permitir novas opções de menu com a inclusão de novas classes visuais (essa flexibilidade já existe em termos de janelas de diálogo).

Uma característica interessante a ser acrescida é um interpretador de expressões matemáticas, de modo a ser possível alterar a função de resposta do nó durante o tempo de execução, tornando assim a ferramenta mais genérica, e não apenas útil para programadores com acesso ao código fonte.

Essa modificação, associada a comandos para inclusão de nós independentes e de tipos diferentes no ambiente, tornariam a ferramenta útil não apenas para visualização de redes implementadas, mas para a criação de redes de modo mais simples e intuitivo.

7 Referências

[1] ANDERSON, J. A.; ROSENFELD, E. **Neurocomputing**: foundations of research. The MIT Press, 1988.

[2] BATTAIOLA, A. L. **Análise de conceitos relacionados à implementação de sistemas de visualização tridimensional de dados meteorológicos**. Tese de Doutorado, INPE, São José dos Campos, 1992.

[3] CARD, S. K.; MACKINLAY, J. D.; SHNEIDERMAN, B. **Readings in Information Visualization**: Using Vision to Think. Academic Press, 1999.

[4] DEFANTI, T. A.; BROWN, M. D.; McCORMICK, B. H. **Visualization:** Expanding Scientific and Engineering Research Opportunities. IEEE Computer, 22(8), 12-25, 1989.

[5] **Direct3D vs. OpenGL: A Comparison**. Disponível em <http://www.xmission.com/~legalize/d3d-vs-opengl.html>, Acesso em Abril, 2004.

[6] ROY, P. **Direct3D vs. OpenGL: Which API to Use When, Where, and Why**. Disponível em <http://www.gamedev.net/reference/articles/article1775.asp>, Acesso em Fevereiro 2004.

[7] EBERHART, R. C.; DOBBINS, R. W. **Neural networks PC Tools**: A Practical Guide. Academic Press, 1990.

[8] FILGUEIRAS, L. V., et al. **Fundamentos de Computação Gráfica**: Compugrafia. Livros técnicos e Científicos Editora S.A., 1987.

[9] FRANÇON, J.; LIENHARDT, P. **Basic Principles of Topology-Based Methods for Simulating Metamorphoses of Natural Objects.** In: THALMANN, N. M.(Ed.); THALMANN, D. (Ed.) **Artificial Life and Virtual Reality**. John Wiley & Sons, 1994.

[10] FREEMAN, J. A., SKAPURA, D. M. **Neural networks**: Algorithms, Applications, and Programming Techniques. Addinson-Wesley Publishing Company Inc., 1992.

[11] FUKUSHIMA, K. **Neocognitron:** A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Biological Cybernetics, 36, pp. 193-202 (April 1980).

[12] HAYKIN, S. **Neural Networks:** a comprehensive foundation – 2nd Edition. Prentice Hall, 1999.

[13] HEALEY, C. G.; BOOTH, K. S.; ENNS, J. T. **High-Speed Visual Estimation Using Preattentive Processing.** ACM Transactions on Computer-Human Interaction, 3(2), 107-135, 1995.

[14] SUN MICROSYSTEMS JAVA 3D ENGENEERING TEAM. **Java 3D API Tutorial.** Disponível em http://java.sun.com/developer/onlineTraining/java3d/. Acesso em Abril 2004.

[15] SUN MICROSYSTEMS. **Java 3D Performance Guide**. Disponível em (http://java.sun.com/products/java-media/3D/collateral/j3d_perfguide.html). Acesso em Abril 2004.

[16] KELLY, J. P. **The Neural Basis of Perception and Movement**. In: KENDEL, E. R. (Ed.), et al. **Principles of Neural Science**, Chapter 20. Appleton & Lange, 1991.

[17] LENT, R. **Cem Bilhões de Neurônios**: Conceitos Fundamentais de Neurociência. Editora Atheneu, 2001.

[18] LIN, X. **Visualization for the Document Space**. Proceedings of IEEE Vizualization'92 Conference, Boston. 274-281, 1992.

[19] NEWMAN, W. M., SPROULL, R. F. **Principles of Interactive Computer Graphics.** McGraw-Hill, 1979.

[20] PAPATHOMAS, T. V.; SCHIAVONE, J. A.; JULESZ, B. **Applications of computer graphics to the visualization of meteorological data.** ACM SIGGRAPH Computer Graphics, Proceedings of the 15th annual conference on computer graphics and interactive techniques, 22 (4), 1988.

[21] REEVES, W. T. **Particle Systems**: A Technique for Modeling a Class of Fuzzy Objects. ACM Computer Graphics, Volume 2 number 2, April 1983

[22] TONÉE, M. J. **An Itroduction to the Visual System**. Cambridge, UK: Cambridge University Press, 1996.

[23] HELBING, R.; RÜGER, M., ILGENSTEIN, Uwe. **A Flexible Approach to Modelling Computer Visualization using Simulation Traces**. Institut für Simulation und Graphik, Otto-von-Guericke-Universität Magdeburg, 1997.