Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica

Uma Bancada Para Processamento Concorrente Dedicada à Computação de Imagem

autor: José Raimundo de Oliveira

orientador: Prof.Dr. Léo Pini Magalhães

co-orientador: Prof.Dr. Paulo Cesar Bezerra

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas para a obtenção do título de Doutor em Engenharia Elétrica (área Automação).

Campinas - SP
dezembro 1995

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

OL4b

Oliveira, José Raimundo de
Uma bancada para processamento concorrente dedicada à computação de imagem / José Raimundo de Oliveira.-- Campinas, SP: [s.n.], 1995.

Orientadores: Léo Pini Magalhães, Paulo Cesar Bezerra.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Hardware - Linguagens descritivas. 2. Arquitetura de computador. 3. Computação gráfica. 4. Processamento de imagens. I. Magalhães, Léo Pini. II. Bezerra, Paulo Cesar. III.Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. IV. Título.

Uma Bancada Para Processamento Concorrente Dedicada à Computação de Imagem

autor: José Raimundo de Oliveira

orientador: Prof.Dr. Léo Pini Magalhães

co-orientador: Prof.Dr. Paulo Cesar Bezerra

Data: 11/12/1995

Banca:

Presidente:
Prof.Dr. Léo Pini Magalhães

Membros Titulares:
Prof.Dr. Márcio L. Andrade Netto (DCA/FEE/UNICAMP),
Prof.Dr. Clésio L. Tozzi (DCA/FEE/UNICAMP),
Prof.Dr. Cláudio Zamitti Mammana (USP SP),
Prof.Dr. Caetano Traina Junior (USP SC).

Membros Suplentes:
Prof.Dr. Ivan Ricarte (DCA/FEE/UNICAMP),
Prof.Dr. Furio Damiani (DSIF/FEE/UNICAMP).

Este exemplar corresponde à redação final da tese defendida por José Raimundo de Oliveira e aprovada pela Comissão Julgadora em 11 12 95

Orientador

Dedico este trabalho

Às minhas filhas ***Ana Paula*** *e* ***Fernanda****, principalmente, pela* ***alegria;***

à minha querida esposa ***Lena****, principalmente, pelo* ***amor;***

e a meus pais ***Benigno*** *e* ***Efigênia****, principalmente, pelo* ***apoio****.*

Agradecimentos

A conclusão deste trabalho se deve em muito a:

o ***Prof. Léo Pini Magalhães****, pela amizade e paciente orientação;*

o ***Prof. Paulo Cesar Bezerra****, pela amizade, discussões técnicas e orientação;*

o ***Prof. Márcio L. Andrade Netto****, pela amizade e ajuda na definição do tema;*

o ***Prof. Clésio L. Tozzi****, pela amizade e ajuda na definição do tema;*

o ***Prof. Roberto de Alencar Lotufo e família****, pela amizade e hospitalidade em Bristol;*

os ***Prof. Carlos A. Reis****,* ***Furio Damiani*** *e* ***Ivanil S. Bonatti*** *companheiros, juntos com o Prof. P.C.Bezerra, na implantação do LCAEe;*

o acadêmico ***Miguel Francisco Augusto****, pela competente administração do sistema do LCAEe e amizade;*

os engenheiros, ***Norbert Chien*** *da Mentor Graphics e* ***José Carlos Panhota*** *da HICAD, que suportaram a implantação do LCAEe desde o seu início;*

os membros da banca, ***Prof. Cláudio Mammana*** *e* ***Prof. Caetano Traina****, que juntos com os professores Márcio e Clésio contribuiram com sugestões enriquecedoras para o presente texto;*

e a todos os amigos do ***corpo docente****, do* ***corpo discente****, e do* ***corpo tecno-administrativo*** *da Faculdade de Engenharia Elétrica da Unicamp*

Uma Bancada para Processamento Concorrente Dedicada à Computação de Imagens

Resumo:

Sistemas de Computadores dedicados a Computação de Imagem tem sido o alvo de diversos trabalhos de pesquisa e desenvolvimento em universidades e indústrias por todo o mundo. A maior parte dos trabalhos disponíveis na literatura visa a implementação de algoritmos específicos em circuitos dedicados em VLSI.

Neste trabalho evitou-se o desenvolvimento de uma arquitetura muito rígida, presa a um único algoritmo. No lugar disto, procurou-se integrar numa bancada uma memória de quadro ligada a um sistema de interconexão de multiprocessadores, chamada de VAM (Via de Acessos Múltiplos). A VAM permite a interligação de processadores elementares (PE) de um arranjo de processamento que trata de forma paralela todas as tarefas de um sistema de computação de imagem. Cada PE pode implementar as suas tarefas por *software*, por *firmware* ou mesmo por *hardware* dedicado. Trata-se, portanto, de uma bancada para experimentos que permite o desenvolvimento de atividades de pesquisa em arquitetura de computadores, em circuitos VLSI dedicados, interconexão de multiprocessadores, *software* básico e de aplicação em computação de imagem.

Para o desenvolvimento deste trabalho foram necessários estudos nas áreas de síntese de imagem, de circuitos de exibição e na área de arquiteturas aplicadas à computação de imagem. Estes estudos foram baseados num amplo levantamento bibliográfico. Com base nestes estudos, foi proposta uma especificação da bancada. Em cima desta especificação foram estudados exemplos de aplicação da bancada. Para este projeto foram utilizados recursos de engenharia concorrente disponíveis na Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas.

Palavras Chaves:

Arquiteturas em *Hardware*, Tratamento de imagem em paralelo, Dispositivos de Exibição *Raster*, Geração de Imagens com Realismo.

A Workbench for Concurrent Image/Graphic Processing

Abstract:

Computer systems dedicated to Image Processing, Computer Graphics and Computer Vision have been the subject of several research and development works at universities and industries all over the world. Most of these works describes specific VLSI implementations of algorithms.

This work purposely avoids the development of an architecture restricted to a single algorithm. Instead, we integrate a frame buffer connected to a multiprocessor interconnection structure referred as VAM (this acronym comes from the portuguese denomination: "Via de Acessos Múltiplos" - Multiple Access Bus). A VAM allows the interconnection of processing elements (PE) that can execute in parallel an image algorithm. Each PE can implement its tasks by software, by firmware, or by dedicated hardware. This architecture works as an experimental workbench that allows research and development in computer architecture, multiprocessor interconnection, application specific VLSI IC and software.

The features of the proposed VAM have been specified as a result from a large library search for references on image synthesis, display circuits and computer graphic specific architectures. Also based on this search, we discuss two examples of applications of VAM to specific problems. This design was developed using facilities of Concurrent Engineering available in Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Brazil.

Key-words:

Hardware Architecture, Parallel rendering, Raster Display Devices, High-Reality Image Generation.

Tabela de Conteúdo

Capítulo 3

Capítulo 4

Capítulo 7

Capítulo 8

Lista de Figuras

Lista de Tabela

Capítulo 1

Introdução ao Trabalho

1.1 - Contexto no qual o trabalho se situa.

Computação Gráfica e **Processamento de Imagens**. Estes termos são utilizados na literatura de forma dúbia. Alguns autores chamam de Computação Gráfica (CG) as atividades referentes a Geração e Síntese de Imagens e de Processamento de Imagem (PI), aquelas atividades de Análise e Tratamento de Imagens. Enquanto que outros autores, tratam Processamento de Imagem como sendo uma linha da Computação Gráfica. O fato é que existem duas linhas de trabalho cujas diferenças principais se dão no nível de Entrada/Saída (E/S), pois os requisitos de processamento feitos por elas são básicamente os mesmos. Assim sendo, sob o ponto de vista da arquitetura *hardware* não existe fronteira entre estas duas áreas de trabalho. É só uma questão de adicionar recursos para entrada da imagem, para ser possível fazer Processamento de Imagem. Ou então adicionar recursos de saída de imagem para se ter um sistema de Computação Gráfica. Para cobrir estas duas aplicações neste trabalho utilizar-se-á o termo **Computação de Imagem**.

Entenda-se então por Computação de Imagem, a área de trabalho que abrange desde a Aquisição e Análise de Imagens até a Síntese e Exibição de Imagens geradas a partir do usuário.

Desta forma Computação de Imagem engloba técnicas e ferramentas computacionais na reprodução, captura e registro de imagens de um cenário já existente ou para expressão de idéias por meio de imagens (geração de imagens, apoio a projeto/desenho). Pode-se representar todas as atividades cobertas pela Computação de Imagem no diagrama da figura 1.1, onde:

- **Aquisição** representa as técnicas de captura de imagem. Abrange desde ótica até algoritmos computacionais de filtragem e pré-processamento da imagem.

- **Análise** envolve as questões de manipulação de imagem. Uma vez adquirida a imagem e armazenada de acordo com um certo formato de dados, esta imagem pode ser operada de acordo com determinadas aplicações.

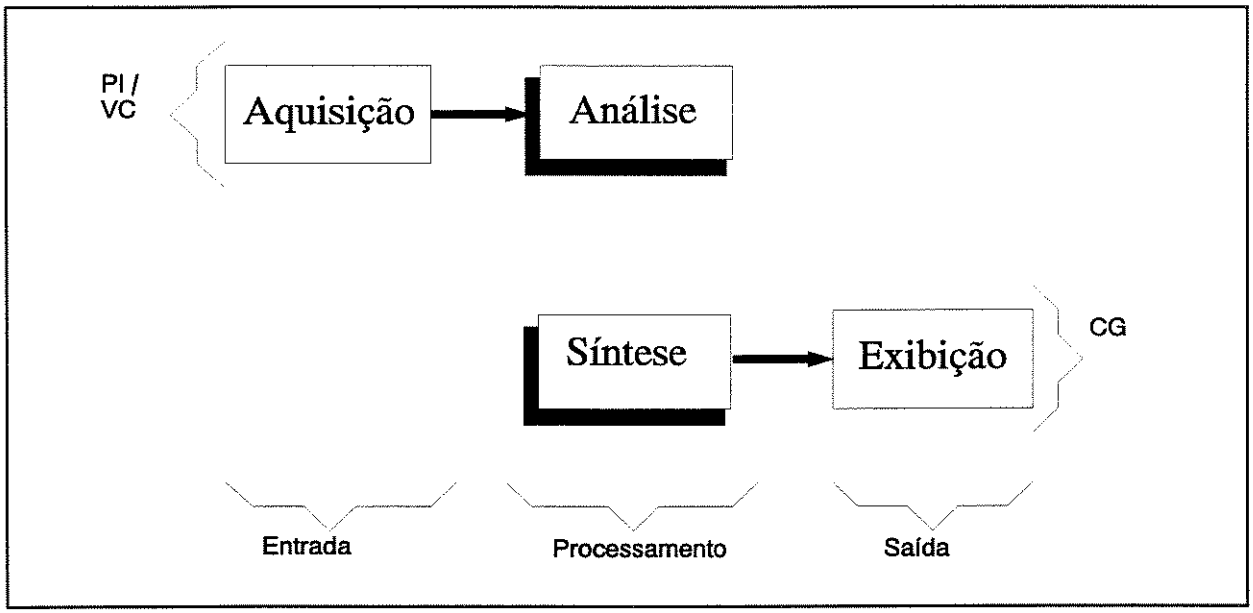


Fig. 1.1 - Computação de Imagem

- **Síntese** envolve as questões de geração de imagens por computador, seja para apoio a projeto, seja para artes. Envolvida por exemplo com os algoritmos de simulação da vista humana para a geração de uma imagem com realismo.

- **Exibição** explora os aspectos pertinentes a exposição de uma imagem ao ser humano, seja num monitor de vídeo, seja em uma impressora (fotoplotadora, laser, matricial...etc).

Dentro da Computação de Imagem, a sub-área de trabalho que abrange tanto a Síntese quanto a Exibição de Imagens é chamada de **Computação Gráfica**. [Jolu 94], [FDFH 90]. E a sub-área de trabalho que envolve tanto Aquisição quanto a Análise de Imagens pode ser sub dividida em **Processamento de Imagem** (PI) e em **Visão Computacional** (VC). PI é a designação das atividades relacionadas com a transformação Imagem - Imagem, ou seja, a geração de uma imagem a partir de uma imagem fonte. Este processamento procura, por exemplo, remover ruídos e manchas gerando uma imagem melhorada sob certo ponto de vista. Isto pode ser útil em aplicações como, por exemplo, sensoriamento remoto, onde os dados adquiridos por um sistema de armazenamento são analisados para prospecção mineral, agricultura, metereologia, etc. VC designa as atividades que procuram o entendimento de uma imagem visando obter informações geométricas, topológicas ou mesmo físicas de objetos numa ou em mais de uma imagem. Exemplo mais ilustrativo de aplicação de VC é encontrado na área de robótica, quando uma imagem é

adquirida e analisada em tempo real. Outro exemplo de aplicação é a inspeção visual automatizada [Horn 86], [BaBr 82].

O DCA-FEE-UNICAMP reúne um grupo de pesquisadores que, desde 1976, vêm desenvolvendo trabalhos em diversas áreas ligadas à Computação de Imagens. Os trabalhos de pesquisa do DCA nesta área abrangem desde Sistemas de Auxílio a Projetos de Engenharia, Padronização de Pacotes Gráficos, Análise de Imagens por Computador, Interface Homem-Máquina, Visão por Computador e Síntese de Imagens por Computador. Por outro lado existe no DCA, também desde 1976, outra linha de pesquisa e desenvolvimento que se interessa pelo estudo de sistemas de processadores.

Uma grande intersecção entre estas duas linhas também sempre existiu dentro do DCA e diversos trabalhos envolvendo microcomputadores e Computação de Imagens foram realizados, seja a nível de *Software*, seja a nível de *Hardware*. O presente trabalho também se enquadra na interseção destas duas linhas de pesquisa.

1.2 - Motivações

Considerando os interesses em Arquitetura de Computadores e Computação de Imagem foi proposto o presente tema de trabalho. O objetivo deste trabalho é propor uma arquitetura de multiprocessamento dedicada a geração de imagens com realismo para, quando implementada, facilitar os trabalhos na área.

Neste trabalho procura-se enfatizar os aspectos de síntese e exibição de imagens, discutindo as técnicas de síntese (algoritmos) e as suas implicações na estrutura de processamento digital. Procura-se ainda discutir, analisar e propor alternativas para a exibição de imagens.

Para a síntese de imagens é necessária uma alta capacidade de processamento número rico e uma alta capacidade de armazenamento. É importante que este processamento seja o mais rápido possível pois além da necessidade de tratar uma grande massa de informações, é imperativo que este processamento não implique num desconforto para o usuário. Mesmo com a disponibilidade de sistemas de processamento de alto desempenho como os chamados Supercomputadores, a comunicação Processador - Monitor de Vídeo da Estação de Trabalho tem inibido algumas aplicações como por exemplo: - Animação em Tempo Real - quando a taxa de transferência pode atingir a ordem de 30 megabytes por segundo. Esta transferência pode ser melhorada se parte do (ou

todo o) processamento gráfico for deslocado para próximo da unidade de exibição, de forma a tornar esta unidade, parte integrante do processador da imagem.
As características essencialmente concorrentes dos processos de Síntese de Imagem sugerem que Estruturas de Multi-Processamento são adequadas para o aumento do desempenho computacional de uma arquitetura para computação gráfica.

Um tendência atual investiga algoritmos de síntese voltados a estruturas de processadores maciçamente paralelos [MCEF 94], [Ells 94], [MPHK 94], e [BBPr 94], bem como os aspectos estruturais do *rendering* em paralelo [Whit 94] e [Neum 94]. É neste contexto que se enquadra o presente trabalho.

1.3 - Resultados esperados

O resultado esperado mais importante deste trabalho é a definição de uma bancada experimental para trabalhos de pesquisas na área de Computação de Imagem.

Adicionalmente esta definição permitirá a aquisição de experiência nas áreas de Processamento Concorrente aplicado à Computação Gráfica e de Projeto Digital Especializado.

Outro benefício esperado com o desenvolvimento deste trabalho advém da própria sistemática de projeto adotada, que proporcionará a utilização desta em novos trabalhos.

1.4 - Introdução aos demais capítulos

O capítulo 2 é dedicado ao estudo da aplicação **Geração de Imagens com Realismo** descrevendo de forma tutorial os problemas desta aplicação e as suas implicações sobre a arquitetura do sistema de computador.

No capítulo 3 são descritas as principais estruturas de sistemas de **Exibição de Imagens** por computador.

No capítulo 4 são levantadas as principais técnicas de processamento aplicadas à Computação de Imagens.

Com base nos estudos e levantamentos destes capítulos, no capítulo 5 é feita a especificação *Hardware* de uma bancada para Geração de Imagens com Realismo. Para caracterizá-la como uma unidade de processamento voltada a aplicações em geração de imagem, foi dada maior ênfase aos aspectos de circuitos de processamento e exibição de imagens.

No capítulo 6 são abordados os aspectos da aplicação da arquitetura proposta. São realçados os aspectos da estrutura de dados e dos algoritmos aplicados.

No capítulo 7 é feita uma análise dos resultados obtidos, comparando-os com outras soluções, e são apresentadas propostas para a continuação das atividades na área.

No capítulo 8 é arrolada a bibliografia considerada no desenvolvimento deste trabalho.

Capítulo 2

Geração de Imagens com Realismo

Este capítulo objetiva esclarecer os conceitos e conhecimentos utilizados durante o texto. Está fartamente documentado de forma a permitir ao leitor um aprofundamento nos tópicos abordados.

2.1 - Geração de Imagem com Realismo

Geração de Imagem com Realismo é a arte de criar cenas com detalhes reais a partir da imaginação de um artista ou a partir da especificação formal de um projeto. Nela não se inclui a reprodução e o tratamento de cenas reais, onde a captura da imagem pode ser feita por métodos fotográficos (químicos ou eletrônicos). A geração por computador de imagens com realismo permite a criação mais rápida de imagens, e que as alterações (modificação, inclusão e exclusão) numa imagem já criada possam ser feitas sem causar grande fadiga ao artista. Some-se a isto uma melhoria da qualidade e da precisão dos detalhes da imagem.

A Geração de Imagem com Realismo por Computador se aplica a todas as áreas que utilizaram, ou ainda utilizam, representação gráfica por desenho artístico, seja para a visualização de objetos em desenvolvimento, seja para fins exclusivamente artísticos. Dentre as diversas áreas que se beneficiam com a adoção de computadores para a geração de imagens pode-se destacar:

- Engenharia mecânica — o computador permite, por exemplo, uma rápida visualização de elementos mecânicos em desenvolvimento. Esta visualização pode ser feita com base em informações de projeto, o que garante sua precisão. O computador permite ainda a visualização de engrenagens, de encaixes mecânicos em cortes, impossíveis de serem implementados físicamente.

- Engenharia civil e arquitetura — um sistema computacional de apoio a projetos de engenharia e arquitetura pode exibir rapidamente o modelo de uma edificação em desenvolvimento com muito mais

detalhe e precisão que as antigas maquetes. Na exibição deste modelo é possível, através da simples mudança de parâmetros de visualização, obter imagens do interior da edificação antes mesmo de se encerrar o projeto, bem como de detalhes, perspectivas, etc.

- Artes plásticas e publicidade — o computador permite uma rápida obtenção tanto de imagens realistas quanto a criação de cenas fictícias envolvendo objetos reais, o que em publicidade é de muita importância. A inclusão e a retirada destes objetos podem ser feitas muito facilmente, ampliando assim as opções do artista na concepção.

- Ciências físicas, químicas e bio-médicas — no campo científico a modelagem e a visualização por computador auxiliam na análise e simulação, desde moléculas até órgãos complexos.

Existem diversas técnicas para gerar imagens com realismo por computador. O maior desafio destas técnicas é compatibilizar o compromisso de alta velocidade de tratamento com uma alta qualidade de imagem. Quanto maior o realismo na imagem, maior tempo de processamento é requerido. As soluções encontradas procuram se basear no aumento da capacidade de processamento dos computadores ou na simplificação de métodos de processamento (melhoria ou criação de novos algoritmos). A grande estratégia para a agilização do tratamento tem sido a transferência de etapas completas de processamento para o *hardware*. As crescentes facilidades no desenvolvimento de circuitos integrados de processamento específico (ASICs - *Application Specific Integrated Circuits*) viabilizam esta transferência.

O presente capítulo descreve de forma introdutória as diversas tarefas necessárias ao processo de geração de imagem com realismo por computador. Nesta descrição não existe a preocupação de ordenar estas tarefas. Um sistema de geração de imagens por computador deve suportar a execução das seguintes tarefas:

. Modelagem Geométrica;
. Transformações com Objetos na Imagem;
. Recorte (*Clipping*);
. Projeção no Plano da Tela e Perspectiva;
. Visibilidade;
. Iluminação;
. Tonalização (*Shading*);
. *Aliasing*;
. Textura.

Nas seções seguintes deste capítulo cada uma destas tarefas é descrita de uma forma resumida. Para um estudo mais detalhado recomenda-se as referências [Roge 85], [HeBa 86], [Hill 90] e [FDFH 90].

2.2 - Modelagem Geométrica

Modelagem, em seu sentido mais amplo, é a técnica de representar comportamentos ou estruturas de entidades e fenômenos físicos e abstratos. A sua utilização permite o ensaio e testes através de simulações para fins de compreensão ou simples visualização.

No caso específico de geração de imagens por computador, utiliza-se modelagem geométrica para descrever os objetos de uma cena a ser sintetizada para exibição. São utilizados para isto modelos matemáticos adequados ao processamento computacional. Os objetos são usualmente modelados pela descrição de suas formas (volume ou superfície). Várias formas podem ser usadas: polígonos, esferas, cilindros, linhas parametrizadas e equações implícitas. Para uma aproximação da forma desejada do objeto real, deve haver um compromisso entre a melhoria do realismo ao usar um modelo mais apurado e os custos adicionais que isto implica. Esta tarefa inclui ainda a definição da localização e orientação de cada objeto no espaço.

A seguir serão analisadas sucintamente as modelagens:

- por *Superfícies Planas* que aproximam a superfície de um objeto por um conjunto de polígonos;
- por *Superfícies Curvas*, que são representadas por curvas paramétricas;
- por *Varredura de Curvas*, onde as descrições dos sólidos são feitas através de operações de rotação e de translação no espaço tridimensional com curvas simples que definem os perfís dos objetos representados;
- Pelo método de *Construtive Solid Geometry — CSG*, que descreve os objetos pela combinação booleana de elementos geométricos primitivos, como por exemplo esferas, cubos, pirâmides e cilindros para gerar elementos mais complexos;
- por códigos *OCTREES*, que definem as propriedades de cada elemento volumétrico do espaço tri-dimensional (VOXEL - *VOlume ELement*) que contém o sólido.

2.2.1 - Superfícies Planas

A superfície de qualquer objeto tridimensional pode ser representada por um conjunto de planos ou seja por superfícies poligonais. Para alguns objetos esta representação é muito precisa, por exemplo para poliedros. Noutros casos, entretanto, esta representação é aproximada. A representação de superfícies por planos é muito simples de ser implementada, inclusive por *hardware*. A sua simplicidade permite uma alta velocidade na exibição dos objetos. Isto é muito útil em aplicações relacionadas com o desenvolvimento de projetos, pois permite uma rápida visualização da estrutura dos objetos. Esta visualização poderá ser melhorada se a superfície dos objetos for dividida em mais faces de polígonos menores.

A descrição de objetos por polígonos pode ser feita através de *Tabelas Descritoras de Polígonos* [HeBa 86]. O formato destas tabelas varia muito em função da aplicação e do pacote gráfico a que será submetida. Basicamente elas possuem informações geométricas, ou seja das coordenadas dos vértices, de arestas e de superfície. Um exemplo de tabela de dados geométricos é dado na figura 2.1. Outras informações podem ser anexadas a estas tabelas como por exemplo: orientação dos planos no espaço, informação de cor, de textura, etc.

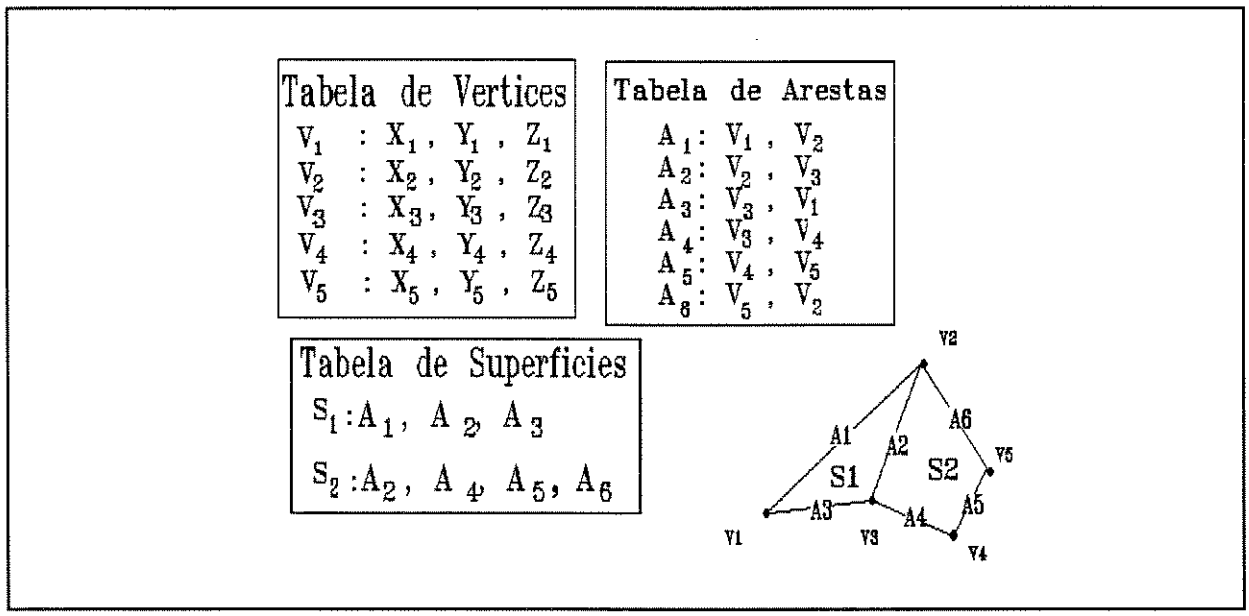


Fig. 2.1 - Descrição do contorno de um objeto composto por duas superfícies planas.

Para determinadas operações sobre os objetos na cena, como por exemplo: - operação de transformações de visibilidade, de superfícies escondidas e

tonalização (*shading*), são necessárias informações adicionais sobre a orientação dos polígonos que formam as superfícies. Estas informações podem ser obtidas a partir da descrição dos polígonos por meio da equação do plano.

Uma superfície plana pode ser expressa da seguinte forma:

$$Ax + By + Cz + D = 0 \qquad \textbf{(2.1)}$$

onde x, y e z representam qualquer ponto na superfície do plano. Os coeficientes A, B, C e D são constantes que podem ser calculadas em função dos valores das coordenadas de três pontos não colineares no plano. A escolha destes três pontos pode ser feita de diversas formas. A mais conveniente para garantir a não-colinearidade utiliza três vértices do polígono. A solução deste cálculo, dados três pontos não co-lineares (x_1, y_1, z_1), (x_2, y_2, z_2) e (x_3, y_3, z_3), é

$$\begin{aligned} A &= y_1(z_2-z_3)+y_2(z_3-z_1)+y_3(z_1-z_2) \\ B &= z_1(x_2-x_3)+z_2(x_3-x_1)+z_3(x_1-x_2) \\ C &= x_1(y_2-y_3)+x_2(y_3-y_1)+x_3(y_1-y_2) \\ D &= -x_1(y_2z_3-y_3z_2)-x_2(y_3z_1-y_1z_3)-x_3(y_1z_2-y_2z_1) \end{aligned} \qquad \textbf{(2.2)}$$

A orientação de um plano no espaço é dada pelo seu vetor normal. As coordenadas cartesianas deste vetor são (A, B, C).

Se a seleção dos vértices do polígono para o cálculo do vetor normal obedecer ao sentido anti-horário visto de fora para dentro num sistema de coordenada da mão direita (descrito em 2.3.4), a direção do vetor normal será de dentro para fora. Os pontos no espaço poderão ser separados em pontos internos ou externos a este plano de acordo com as inequações (2.3) e (2.4).

Os pontos serão externos se

$$Ax + By + Cy + D > 0 \qquad \textbf{(2.3)}$$

Os pontos serão internos se

$$Ax + By + Cz + D < 0 \qquad \textbf{(2.4)}$$

2.2.2 - Superfícies Curvas

A representação de superfícies curvas pode ser feita por meio de Equações Paramétricas [HeBa 86]. As equações paramétricas para superfícies são formuladas com dois parâmetros u e v. Um ponto na superfície é representado pelo vetor de funções paramétricas:

$$P(u,v) = (x(u,v),\ y(u,v),\ z(u,v)) \tag{2.5}$$

As equações paramétricas são normalizadas de forma que os parâmetros u e v sejam definidos dentro do intervalo 0 até 1. Como exemplo, uma superfície esférica pode ser descrita pelas equações:

$$\begin{aligned} x(u,v) &= r\sin(\pi u)\cos(2\pi v) \\ y(u,v) &= r\sin(\pi u)\sin(2\pi u) \\ z(u,v) &= r\cos(\pi u) \end{aligned} \tag{2.6}$$

Onde r é raio da esfera. O parâmetro u descreve linhas de latitude constante e o parâmetro v descreve linhas de longitude constante.

A descrição de uma mesma superfície pode ser feita de diversas formas. Uma forma conveniente para manipulação por computador é a forma polinomial. O método B-Spline ou o seu caso particular Bézier, são dois exemplos de descrição de superfícies curvas por polinômios.

Superfícies de Bézier

A formulação das curvas de Bézier pode ser estendida facilmente para descrever superfícies tridimensionais. Isto é feito pelo produto cartesiano de duas curvas de Bézier. São usadas duas funções similares, uma para cada parâmetro:

$$P(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} p_{i,j}\ B_{i,n}(u)\ B_{j,m}(v) \tag{2.7}$$

Superfícies B-Splines

A extensão do método de B-Spline para a descrição de superfícies é similar a feita para o método de Bézier, ou seja através de um produto cartesiano de duas funções *Blending*.

$$P(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m} p_{i,j}\ N_{i,k}(u)\ N_{j,l}(v) \qquad (2.8)$$

2.2.3 - Representação por Varredura de Curvas

Alguns objetos com formato de superfície simétrica podem ser descritos em função de uma curva geradora e de uma operação geométrica no espaço tridimensional com esta curva [HeBa 86]. As operações mais comuns para a geração de imagens são a translação de uma curva e a rotação em torno de um eixo.

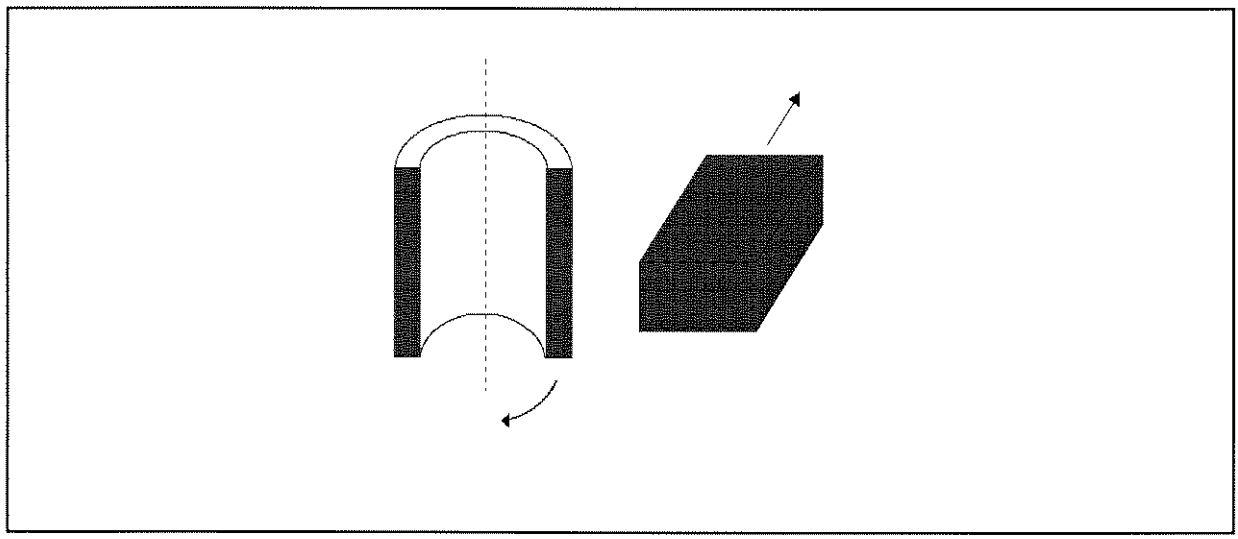


Fig. 2.2 - Exemplos de sólidos gerados a partir de operações com curvas.

Por exemplo, dado um eixo de rotação e um retângulo como na figura 2.2 Uma operação de rotação em torno do eixo pode gerar um cano com a parede espessa.

Outro exemplo: Dado um retângulo no plano xy, uma operação de translação deste retângulo no sentido do eixo z gera um paralelepípedo.

2.2.4 - Métodos Constructive Solid-Geometry

O método CSG é utilizado por diversos pacotes gráficos. Ele baseia-se na utilização de uma biblioteca de sólidos primitivos simples e num conjunto de operadores regulares de sólidos, União (+), Diferença (-) e Interseção (&). O efeito da aplicação dos operadores sobre dois sólidos é mostrado na figura 2.3.

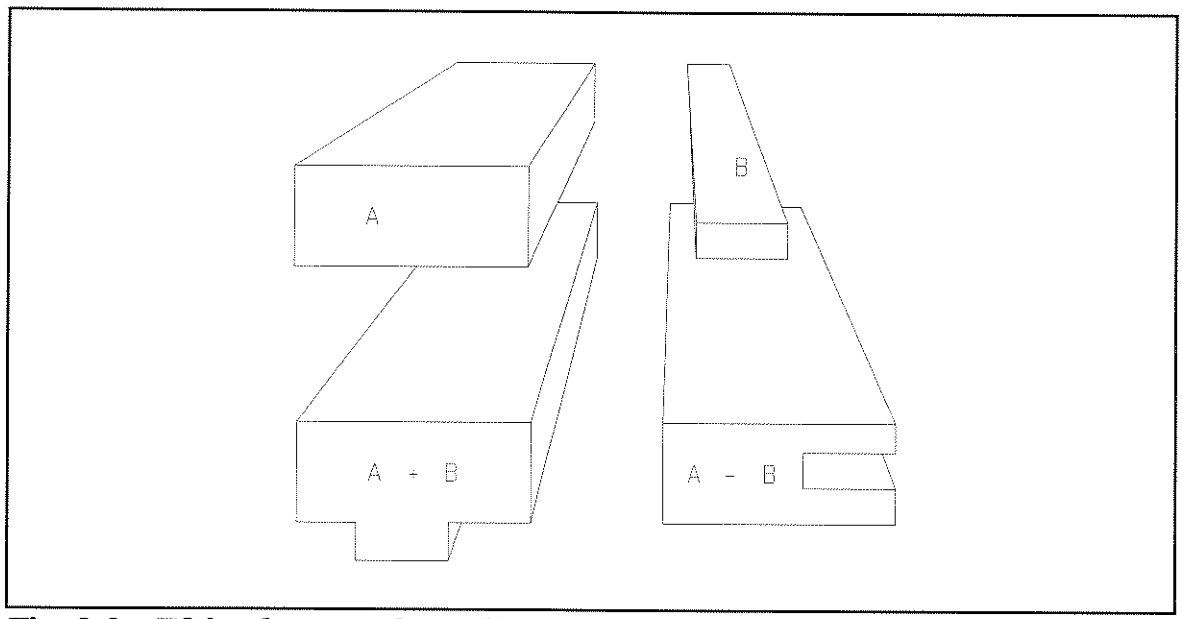


Fig. 2.3 - Efeito dos operadores CSG sobre sólidos primitivos

2.2.5 - Octrees

Octrees são estruturas hierárquicas em árvores que são utilizadas para representar sólidos em alguns sistemas gráficos. A estrutura é organizada de forma que cada nó corresponde a uma região no espaço tri-dimensional. Esta representação aproveita as coerências espaciais para reduzir os requisitos de área em memória para armazenar os objetos descritos. Esta estrutura permite ainda uma representação conveniente para objetos com superfícies removidas e para o desempenho de outras manipulações de objetos.

A codificação em *octrees* é uma extensão para o espaço tri-dimensional da codificação *quadtree* para o espaço em 2D.

A codificação *quadtree* de uma região plana é gerada pela divisão sucessiva desta região em quadrantes. Cada nó no *quadtree* tem quatro elementos de

dados, um para cada quadrante na região. Se todos os pixels naquele quadrante tiverem a mesma cor a divisão termina. Se for preciso nova divisão do quadrante é feita até que todos os quadrantes de uma divisão sejam da mesma cor.

A codificação octree de um objeto sólido é feita dividindo o objeto em octantes. Cada nó no *octree* possue oito elementos de dados. Se cada elemento de volume dentro de um octante tiver a mesma característica, esta é armazenada no seu elemento de dado correspondente. Se houver alguma diferença nova subdivisão em octantes é feita até que não haja mais diferenças. Um exemplo da divisão de uma região tridimensional é dada na figura 2.4.

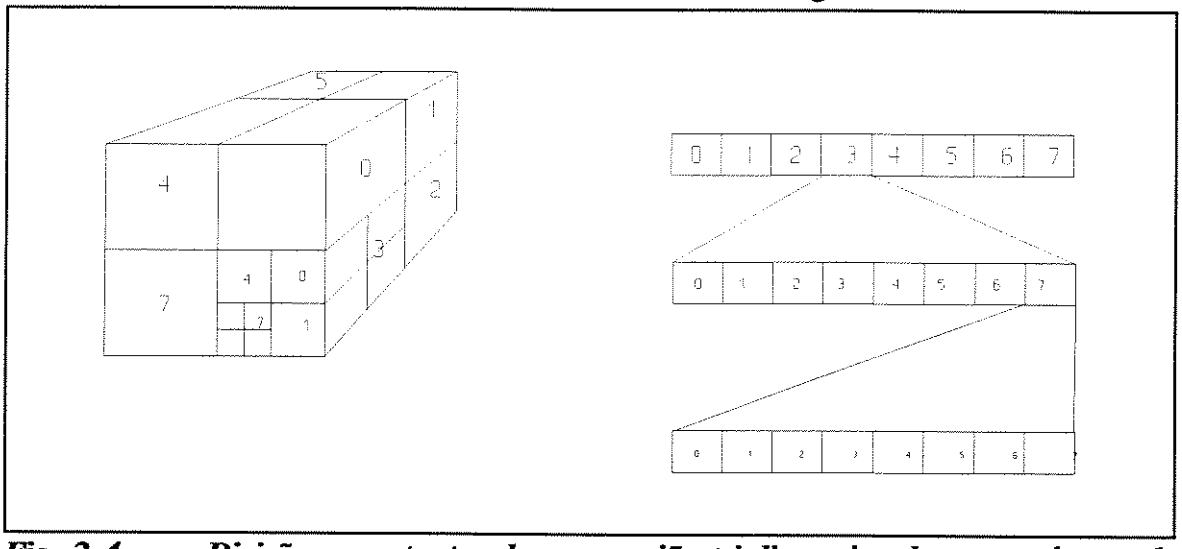


Fig. 2.4 - Divisão em octantes de uma região tri-dimensional e a sua árvore de dados.

Os algoritmos para geração de *octrees* podem ser estruturados para aceitar descrição de objetos em qualquer forma, como poligonal, superfícies curvas e CSG. Usando as coordenadas máxima e mínima do objeto um cubo pode ser traçado em volta dele. Esta região contendo o objeto é então testada octante por octante para gerar a representação em *octree*.

2.3 - Operações com objetos na Imagem

Dada uma descrição de uma cena, é desejável poder executar operações sobre os pontos que descrevem os objetos nela contidos. Estas operações permitem,

por exemplo, deslocar objetos numa cena, alterar o tamanho destes objetos, etc.

Matrizes de transformação

As coordenadas de um ponto num sistema de coordenadas podem sofrer diversos tipos de operação, ou de transformação. Estas transformações são equacionadas em notação matricial. Esta notação permite que cada transformação seja representada por um único nome ou símbolo e que duas transformações possam ser combinadas, ou concatenadas, para resultar numa terceira transformação com o mesmo efeito da aplicação em sequência das duas primeiras. A seguir tem-se um exemplo de operações de transformação sobre um ponto numa cena.

Exemplo de transformações sobre um ponto

seja p um ponto de coord. x, y e z;

seja **T** uma matriz de translação a ser aplicada em p, define-se p_T, o novo ponto transladado, como

$$p_T = p\,\mathbf{T}\ ;$$

seja **S** uma matriz de escala a ser aplicada em p_T, define-se p_{TS}, o novo ponto escalado, como

$$p_{TS} = p_T\,\mathbf{S}\ .$$

O mesmo resultado pode ser atingido definindo-se **U** o produto matricial de **T** e **S**.

$$\mathbf{U} = \mathbf{T}\,\mathbf{S},$$

U é a matriz de transformação equivalente a aplicação de uma translação seguida de uma escala.

Tem-se, portanto,

$$p_{TS} = p_T\,\mathbf{U}\ .$$

Dentre os diversos tipos básicos de transformação, destacam-se três, à saber: Translação, Rotação e Escala. As matrizes destas transformações em coordenadas homogêneas são dadas a seguir.

2.3.1 - Transformação de Translação

A transformação que desloca ou translada um ponto (x, y, z) para um novo ponto (x', y', z') é

$$[x' \; y' \; z' \; 1] = [x \; y \; z \; 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} \quad \textbf{(2.9)}$$

Onde T_x, T_y e T_z são as componentes de translação nas direções X, Y e Z respectivamente.

2.3.2 - Transformação de Rotação

A transformação de rotação em 3D é mais complexa que a mesma operação em 2D. Na transformação em 3D é necessário informar em torno de que eixo vai se dar a rotação. É fácil equacionar a rotação em torno de um dos eixos de coordenadas 3D, entretanto nem sempre isto é o desejado. Neste caso é necessário executar uma série de transformações de rotação e translação para que se possa aplicar a transformação de rotação em torno de um dos eixos. As equações para as Transformações de Rotação em torno dos três eixos são representadas nas equações (2.10), (2.11) e (2.12) onde o parâmetro Θ especifica o ângulo de rotação em torno do eixo no sentido horário. Estas equações consideram que o observado está alinhado com o eixo da rotação e olhando para a origem.

Transformação de Rotação em torno do eixo z

A rotação em torno do eixo z pode ser equacionada como abaixo. O sentido positivo da rotação é ilustrado na figura 2.5a

$$[x' \; y' \; z' \; 1] = [x \; y \; z \; 1] \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.10)}$$

Transformação de Rotação em torno do eixo y

O sentido positivo de rotação em torno do eixo y é ilustrado na figura 2.5b. O resultado desta transformação pode ser conseguido pela equação:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \cdot \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.11)}$$

Transformação de Rotação em torno do eixo x

A equação abaixo transforma as coordenadas dos pontos de um objeto de acordo com uma rotação em torno do eixo x cujo sentido positivo é mostrado na figura 2.5c.

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.12)}$$

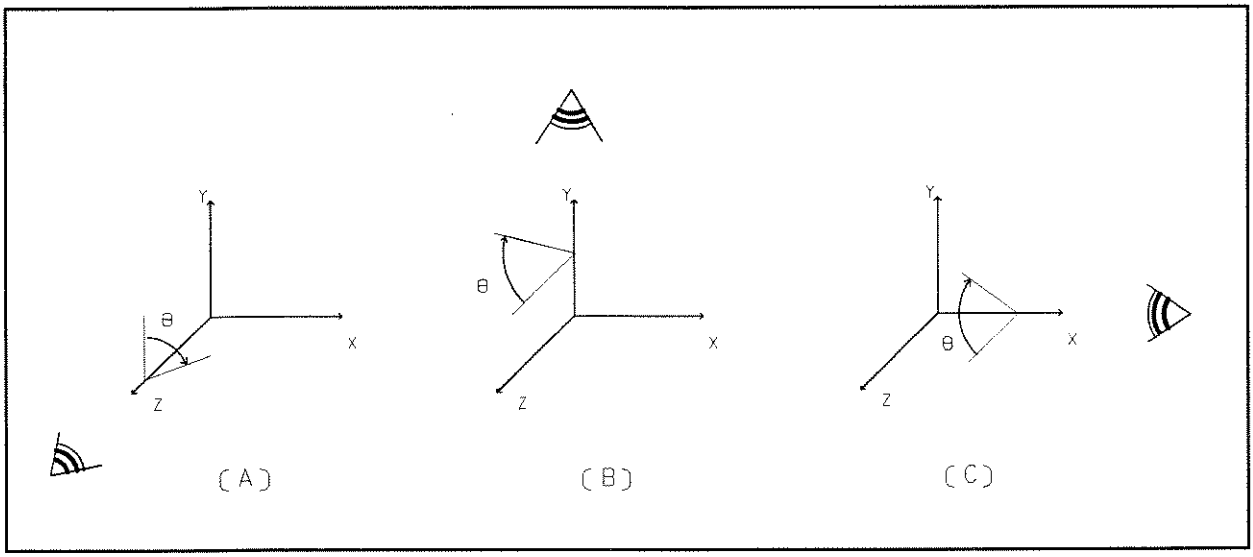


Fig. 2.5 - Rotações em torno dos eixos de coordenadas. Os ângulos são medidos no sentido horário sobre o eixo olhando para a origem.

2.3.3 - Transformação de Escala

A Transformação de Escala permite alterar o tamanho de objetos a serem exibidos. Isto é feito pela multiplicação de cada valor de coordenada por um fator de escala. Esta transformação é representada matricialmente de acordo com a seguinte equação:

$$[x' \; y' \; z' \; 1] = [x \; y \; z \; 1] \cdot \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.13)}$$

2.3.4 - Outras Transformações importantes

Além das transformações de Translação, Rotação e Escala é importante destacar as transformações de Visão, de Deslizamento e de Reflexão. Estas transformações apresentam características lineares que permitem a sua representação em termos de matrizes.

Transformação de Visão

Considere um universo a ser observado. Todos os objetos neste universo são descritos mais facilmente em termos de um sistema de coordenadas em três dimensões cujas referências estão dentro deste universo. Este sistema de coordenadas será chamado de Sistema de Coordenadas do Mundo Real - CM e se referencia a um sistema de coordenadas cartesianas convencional, como o mostrado na figura 2.6. Este sistema é conveniente para representar objetos no mundo real, pois pode-se utilizar como origem de referência (ponto 0, 0, 0) um dos objetos da cena e descrever todos os demais objetos em função deste. Quanto maior o valor das coordenadas de um objeto na cena, mais distante do objeto na origem vai estar o objeto descrito. A este sistema de coordenadas é dado também o nome de Sistema de Coordenadas da Mão Direita. Isto porque associando-se os dedos polegar, indicador e médio da mão direita, respectivamente, aos eixos X, Y e Z e colocando-os de forma ortogonal, um com respeito ao outro, obtém-se a representação dos eixos de coordenadas.
Considere agora um observador deste universo. As referências deste observador são mais facilmente descritas em termos de outro sistema de coordenadas que tem como origem o Ponto de Observação ou Ponto de Vista. Este sistema será chamado Sistema de Coordenadas do Observador (CV) e é conveniente pois, quanto maior o valor das coordenadas de um objeto na cena, mais distante do ponto de vista do observador vai estar o objeto descrito, como mostrado na figura 2.7. Este sistema de coordenadas é também lembrado com o nome de Sistema de Coordenadas da Mão Esquerda. Isto porque utilizando-se os dedos polegar, indicador e médio da mão esquerda para representar respectivamente os eixos X, Y e Z, e colocando-os de forma ortogonal, um

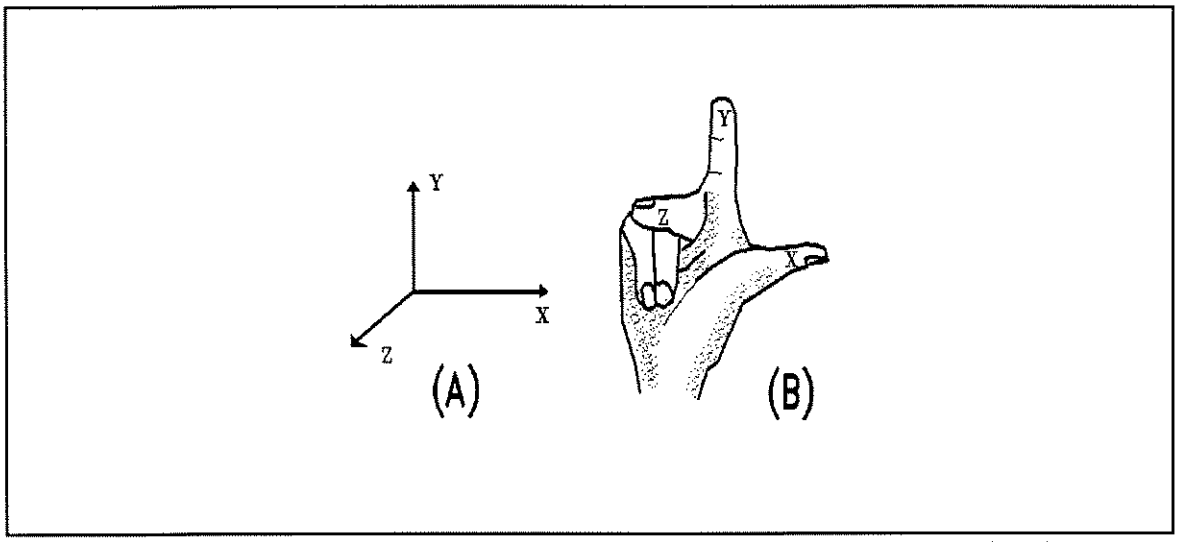


Fig. 2.6 - ***(a)-Sistema de coordenadas do mundo real - CM. (b)-Sistema de coordenadas da mão direita.***

com respeito ao outro, é possível visualizar uma representação dos eixos de coordenadas.

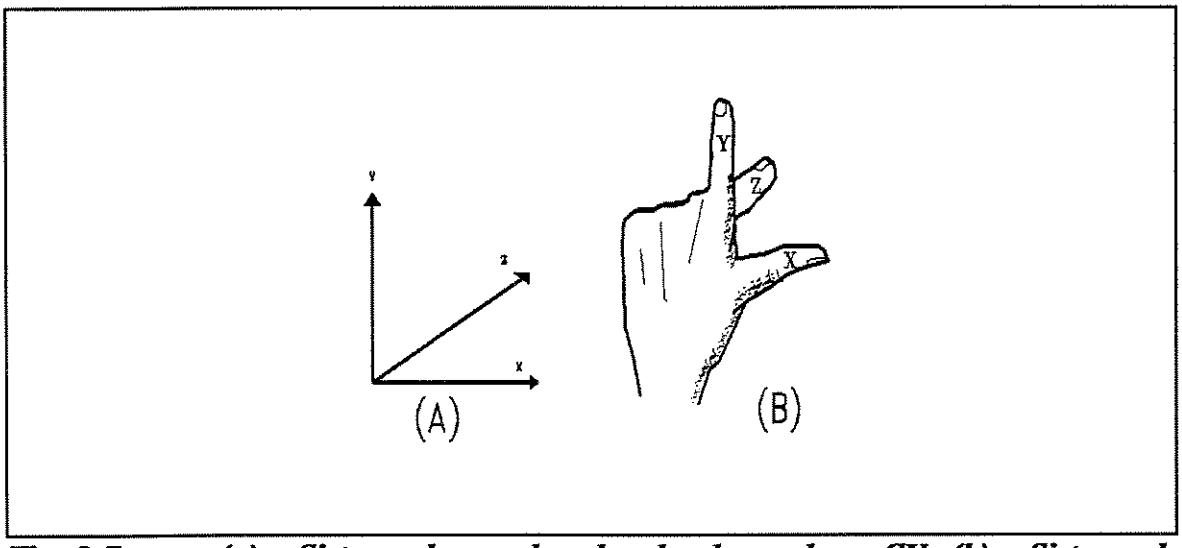


Fig. 2.7 - ***(a) - Sistema de coordenadas do observador - CV. (b) - Sistema de coordenadas da mão esquerda.***

Uma operação importante é transformar as referências das coordenadas CM para as coord. CV. Esta transformação é chamada Transformada de Visão. Um caso especialmente simples de transformada de Visão é quando o eixo Z_m é colinear com o eixo Z_v e os outros eixos são paralelos e possuem o mesmo sentido. Neste caso a operação de transformação é simplesmente uma reflexão no plano $Z_m = 0$. Para qualquer ponto de observação, entretanto, é necessário aplicar em sequências diversas transformadas de rotação e translação nos elementos descritos no CM para obter-se a descrição em CV.

Deslizamento (*shearing*)

A aplicação desta transformação sobre os pontos de um objeto produz uma distorção no formato do seu volume. De uma forma geral a matriz de transformação de deslizamento é caracterizada por possuir elementos não nulos fora da diagonal. A posição e o valor destes elementos levam a tipos e intensidade de distorções diferentes. Um exemplo de matriz de transformação de deslizamento é

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(2.14)}$$

Os parâmetros a e b podem assumir qualquer valor real. O efeito desta matriz é alterar os valores das coordenadas x e y por uma quantidade proporcional ao valor da coordenada z, mantendo o valor desta última. Planos perpendiculares ao eixo z são deslocados por uma quantidade proporcional ao valor de z. Fazendo $a = b = 1$ e aplicando a matriz de transformação sobre o cubo da figura 2.8(a) tem-se o objeto da figura 2.8(b).

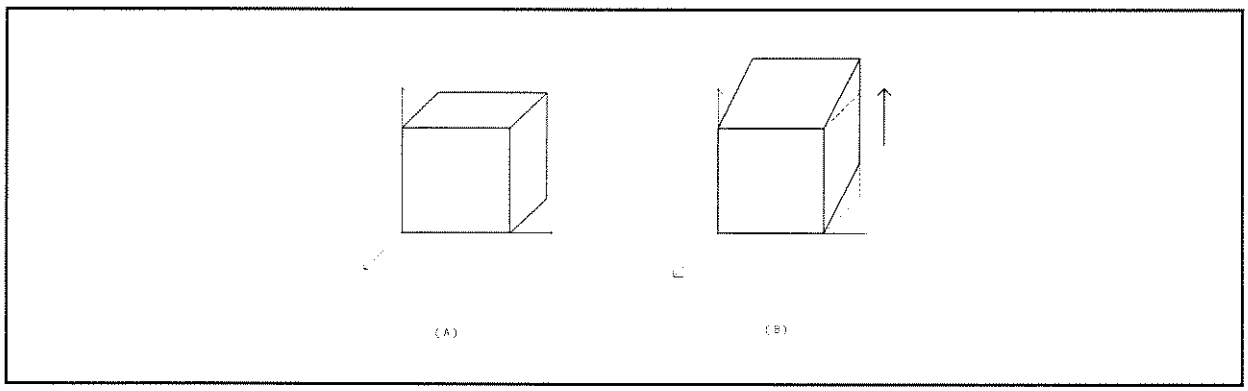


Fig. 2.8 - (a) - Cubo unitário. (b) - Cubo deslizado.

Reflexão

Este tipo de transformação produz reflexão de coordenadas sobre um plano refletor específico. A transformação de coordenadas CM em coordenadas CV que tem os eixos Z_m e Z_v colineares é um exemplo de reflexão utilizando o plano xy como refletor. Neste exemplo a matriz de transformação é dada abaixo.

$$RF_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.15)$$

A reflexão relativa aos planos y_z e x_z é obtida através de matrizes similares. A reflexão sobre outro plano qualquer é obtida pela concatenação de matrizes de transformação de rotação e reflexão.

2.4 - Recorte (*Clipping*)

A partir de um ponto de vista de observação, é preciso determinar qual parte da cena a ser gerada pertence ao campo visual do observador. Com isto é possível eliminar o processamento de partes não visíveis, agilizando a geração da imagem visível. O processo de separar os objetos ou parte dos objetos, que são visíveis em função de um ponto de vista é chamado de Recorte (*Clipping*).

O campo visual do observador forma um volume que pode ter vários formatos, considerando-se o meio através do qual é vista a cena. Por exemplo, se uma cena é observada através de uma janela circular o campo visual é um cone cujo ápice está no olho do observador. Quando a cena é observada através de um quadrilátero, como no caso de uma tela de vídeo de computador, este volume é uma pirâmide.

Representando esta pirâmide no sistema de coordenadas CV, o seu ápice é colocado na origem, ela é simétrica ao eixo Z_v e cada um das suas quatro faces é perpendicular ao plano definido por um par de eixos CV. Ela é chamada de Pirâmide de Visão ou Pirâmide de Recorte (*Viewing or Clipping Pyramid*).

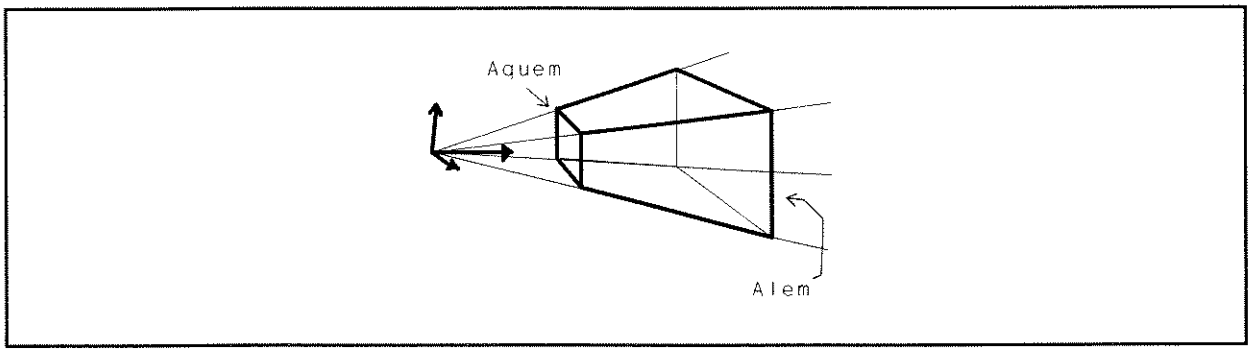


Fig. 2.9 - Volume de Visão

Além dos quatro planos de recorte definidos pelas quatro faces da pirâmide tem-se ainda dois outros planos de recorte, que são perpendiculares ao eixo Z_v. Estes planos impõem restrições de profundidade. O primeiro plano (a partir da origem CV) é chamado de plano AQUÉM (*hither*) ou plano Frontal e o segundo é chamado plano ALÉM (*yon*) ou plano de Fundo. Para lembrar, os objetos serão invisíveis se estiverem aquém do plano AQUÉM ou além do plano ALÉM. O volume formado então por estes seis planos recebe o nome de FRUSTO (do latim: *Frustum*), ou simplesmente Volume de Visão (*View Volume*). A figura 2.9 mostra o volume de visão considerado.

A tela 2D é considerada como se estivesse posicionada de forma que seu plano é perpendicular ao eixo Z, e seus quatro cantos coincidem com as quatro arestas da pirâmide de recorte. Os seus eixos X_s e Y_s são paralelos e tem a mesma direção que os eixos X_v e Y_v , respectivamente.

Pode-se exprimir os limites de recorte em termos da distância da tela ao ponto de observação *d*, do comprimento da metade da tela *s*, e da coordenada Z_v do ponto que esta sendo tratado.

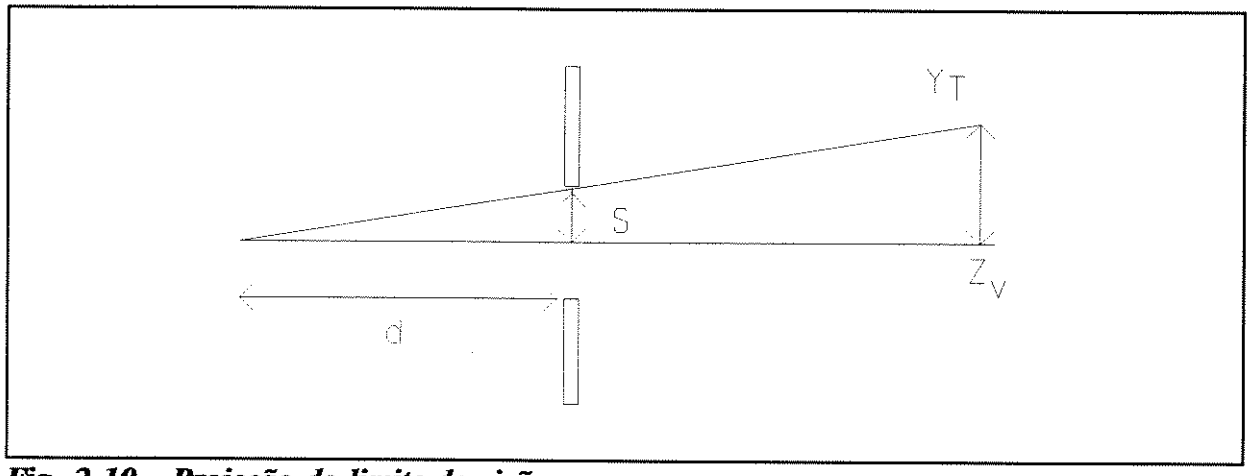


Fig. 2.10 - Projeção do limite de visão

Considere a vista lateral do plano de recorte superior mostrada na figura 2.10. Para um ponto $P(X_v, Y_T, Z_v)$ situado no limite de recorte superior Y_T, podemos ver, por similaridade de triângulos, que:

$$\frac{d}{s} = \frac{Z_v}{Y_T} \qquad \textbf{(2.16)}$$

Ou seja o limite de recorte em Y_v chamado de W_y é

$$W_y(Z_v) = Y_T = Z_v(\frac{s}{d}) \qquad \textbf{(2.17)}$$

De forma que, para um ponto qualquer $P(X_v, Y_v, Z_v)$ estar abaixo do plano de recorte superior (ou seja, possivelmente visível), basta satisfazer

$$Y_v < W_y(Z_v) \qquad (2.18)$$

O limite de recorte W_x pode ser determinado de forma similar. Os parâmetros W_y e W_x podem ser calculados logo que as coordenadas do ponto a ser tratado forem conhecidas.

De uma forma geral para determinar se um ponto está dentro do volume de visão basta comparar a coordenada apropriada com o parâmetro W correspondente, como mostrado na tabela 2.1. Para os recortes de profundidade a compararação é feita da coordenada Z_v do ponto com os valores k_1 e k_2, respectivamente, coordenada Z_v do plano Aquém e coordenada Z_v do plano Além.

Tab. 2.1 - Comparações para o Recorte.

Está dentro	**se**
Superior	$Y_v <= W_y$
Inferior	$Y_v >= - W_y$
Direita	$X_v <= W_x$
Esquerda	$X_v >= - W_x$
Além	$Z_v <= k_2$
Aquém	$Z_v >= k_1$

Um caso particularmente interessante é o da tela quadrada, quando $W_y = W_x$.

2.4.1 - Algoritmos de Recorte.

Existem algoritmos de recorte para linhas retas, para polígonos e para caracteres.

O recorte de linhas retas.

Seguramente o método mais conhecido de recorte de linhas em 3D é baseado no algoritmo de **Cohen-Sutherland** para recorte em 2D descrito em [NeSp 79]. A principal diferença é que o Código de Recorte tem agora 6 bits, um para cada plano de recorte. Este código tem o seguinte formato:

bit 1 - o ponto está à esquerda da volume de visão
bit 2 - o ponto está à direita da volume de visão
bit 3 - o ponto está abaixo da volume de visão
bit 4 - o ponto está sobre a volume de visão
bit 5 - o ponto está aquém da volume de visão
bit 6 - o ponto está além da volume de visão

Como no algoritmo para 2D, uma linha será, trivialmente, aceita se seus dois pontos finais têm os códigos de recorte iguais a ZERO. Ela será, trivalmente, recortada se o AND lógico dos códigos dos seus pontos finais for todo igual a UM. Para todos os outros valores do AND é necessário subdividir a linha.

Esta subdivisão pode ser feita de acordo com o algoritmo de **Subdivisão pelo Ponto-médio** proposto por Sproull e Sutherland descrito em [Roge 85].

Outro algoritmo importante é o **Algoritmo de Cyrus-Beck**, descrito em [CyBe 78]. A sua importância se deve ao fato que ele pode ser aplicado a um volume de recorte com qualquer formato convexo. A limitação deste algoritmo é que ele só se aplica a linhas totalmente contidas em frente do ponto de vista do observador. Se a linha começar ou ultrapassar este ponto, o algoritmo rejeita a linha toda, mesmo que ela seja parcialmente visível. Para contornar este problema o algoritmo é aplicado sob um volume de recorte em coordenadas ordinárias e depois é aplicada ao resultado a transformação de perspectiva.

Liang e Barsky apresentaram em [LiBa 84] um novo algoritmo que passou a ser referenciado como **Algoritmo de Liang-Barsky**. Eles introduziram um novo tipo de teste de rejeição trivial, mais eficiente, para volumes de recorte com qualquer formato. Este algoritmo utiliza coordenadas homogêneas e evita de forma mais eficiente o problema de linhas não totalmente contidas à frente do ponto de vista do observador.

O recorte de polígonos.

Um polígono pode ser representado por um conjunto de linhas retas, e assim pode-se aplicar os algoritmos de recorte de retas para estes polígonos. Entretanto se um polígono fechado é recortado de acordo com os algoritmos de

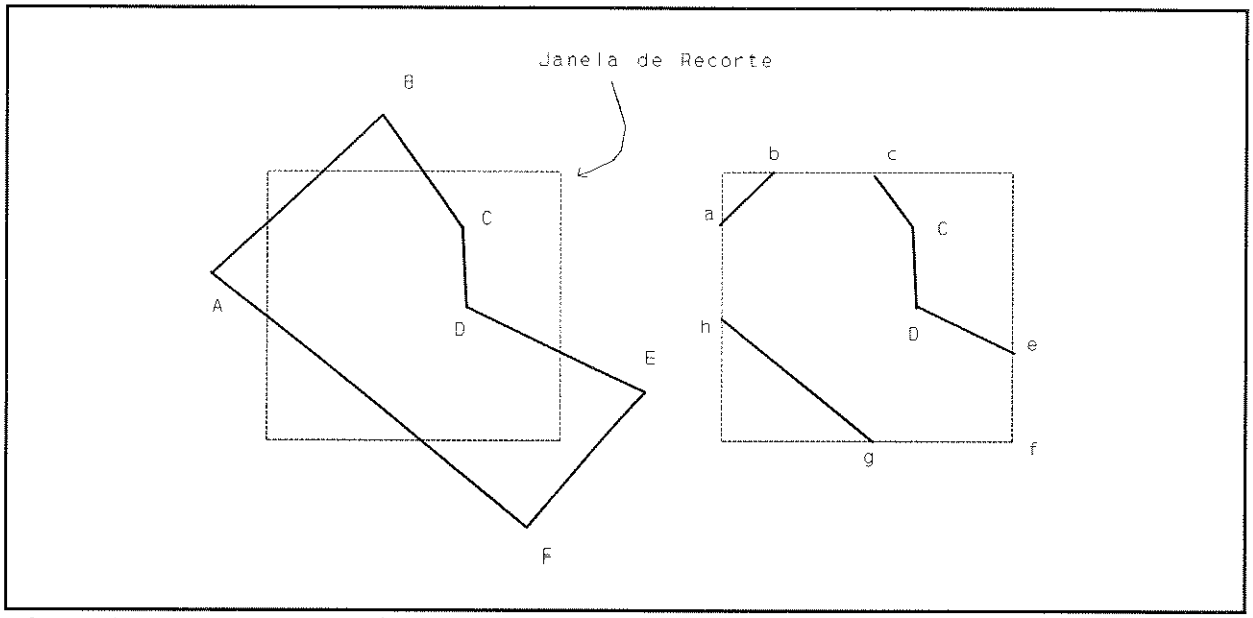


Fig. 2.11 - Recorte de polígono gerando polígonos abertos.

linha, isto pode resultar em um ou mais polígonos abertos como mostrado na figura 2.11. Neste recorte é necessário incluir na descrição do polígono resultante os segmentos h-a b-c e-f e f-g. Mais complicado é o recorte de um polígono concavo como aquele da figura 2.11, que resulta em diversos polígonos disjuntos. As descrições destes polígonos têm que ser geradas corretamente.

O primeiro algoritmo de recorte de polígonos a ser destacado é conhecido como Algoritmo Reentrante de Recorte de Polígonos ou **Algoritmo de Sutherland-Hodgman** descrito em [SuHo 74] que é baseado na idéia de que é fácil recortar um polígono contra um único plano de recorte. O algoritmo se aplica sucessivamente para cada um dos planos de recorte. Ele pode ser aplicado a qualquer polígono planar ou não planar.

Liang e Barsky apresentaram em [LiBa 83] um novo algoritmo para recorte de polígonos. O **Algoritmo de Liang-Barsky** para Polígonos é baseado nos conceitos usados no algoritmo Liang-Barsky para linhas (citado anteriormente nesta seção). Testes indicam que este algoritmo é duas vezes mais rápido que

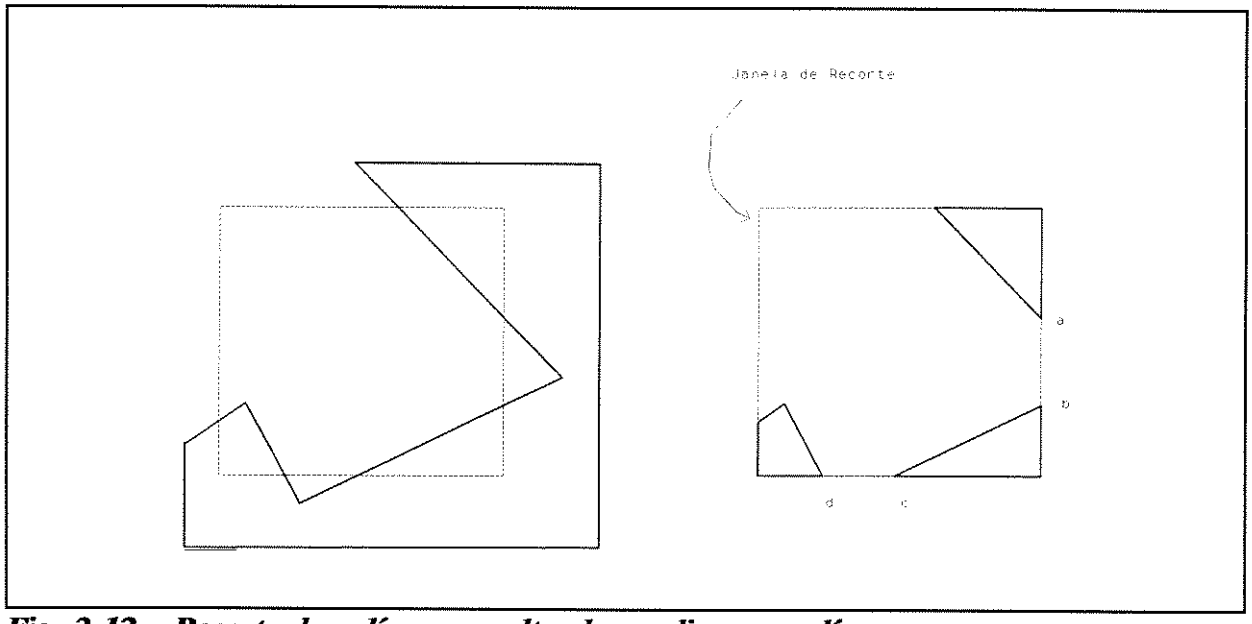


Fig. 2.12 - Recorte de polígono resultando em diversos polígonos.

o algoritmo de Sutherland-Hodgman, segundo [Roge 85].

O último algoritmo de recorte a ser citado é chamado de **Algoritmo de Weiler-Atherton** e é indicado para o recorte de polígonos sob uma região de recorte concava. Este tipo de recorte pode ser aplicado a tarefa de remoção de superfícies escondidas, para evitar o processamento de superfícies colocadas atrás de objetos concavos ou com furos [Roge 85].

O recorte de caracteres.

O recorte de caracteres depende da forma como eles são gerados. Se forem gerados por *software* numa tela em bitmap eles podem ser tratados de acordo com os algoritmos citados anteriormente. Se forem gerados por *hardware* ou *firmware* é necessário a existência de *hardware* para este tratamento.

2.5 - Projeção no Plano da Tela e Perspectiva

Como as imagens geradas a partir de uma cena tridimensional serão exibidas num meio bidimensional, normalmente uma tela de vídeo, é necessário executar sobre os elementos desta cena a conversão da representação tridimensional em bidimensional. Estas conversão recebe o nome de Projeção.

Existem dois tipos de projeção, a Projeção Paralela que permite a descrição exata de detalhes geométricos de objetos e a Projeção Perspectiva, que considera o efeito visual da diminuição do tamanho da imagem de objetos distantes. O tipo de projeção que mais interessa ao processo de geração de imagem com realismo é a chamada Projeção em Perspectiva.

A projeção em perspectiva considera que duas linhas em paralelo no espaço tridimensional quando projetadas num plano de visão parecem convergir para um único ponto, chamado pelos artistas de Ponto de Fuga. Como no espaço 3D linhas paralelas só se encontram no infinito, o ponto de fuga pode ser considerado uma representação de um ponto no infinito. Uma imagem pode ter até 3 pontos de fuga considerando a posição do observador no espaço tridimensional.

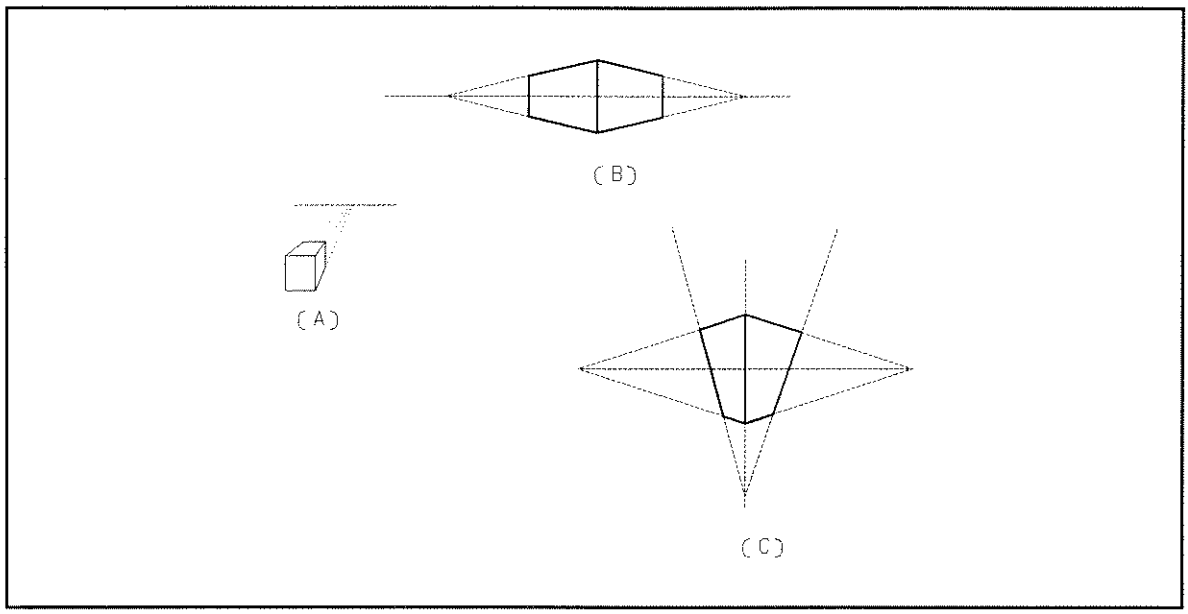


Fig. 2.13 - Projeções em perspectiva. a) Com um único ponto de fuga; b) com dois pontos de fuga; c) com três pontos de fuga.

Na figura 2.13a é mostrado um exemplo de projeção em perspectiva com um único ponto de fuga. O ponto de fuga, neste exemplo, pode ser entendido como sendo o ponto $X_m = -\infty$ onde as linhas paralelas ao eixo X_m se encontram.

Na figura 2.13b tem-se um exemplo de projeção em perspectiva com dois pontos de fuga. Por motivo similar ao caso de um único ponto de fuga, os pontos de fuga neste exemplo podem ser interpretados como sendo $X_m = -\infty$ e $Z_m = -\infty$.

A figura 2.13c mostra um exemplo de projeção em perspectiva com três pontos de fuga.

O cálculo da transformada de projeção em perspectiva, considerando qualquer ponto de vista pode requerer transformações de um conjunto muito grande de pontos. Para agilizar estes cálculos algumas simplificações são feitas. Por exemplo, restringe-se somente a projeção central ou mesmo, utiliza-se de projeções paralelas ou ortográficas que são bem mais simples de serem calculadas. A projeção central é feita considerando que o ponto de vista está colocado sobre um dos eixos do sistema de coordenadas. A projeção ortográfica é uma forma especial de projeção paralela, na qual as linhas paralelas no espaço tridimensional são projetadas como linhas paralelas no plano de visão.

Para ilustrar como pode ser determinada a projeção perspectiva considere que as descrições tridimensionais são feitas no sistema de coordenadas do observador - CV. Neste espaço a tela está colocada de forma paralela ao plano X_vY_v no ponto $Z_v = d$ e o eixo Z_v passa pelo centro da tela. Outra consideração é que a tela é quadrada. A figura 2.14 mostra a posição de um ponto $p_s(X_s,Y_s)$na tela que é o resultado da projeção em perspectiva do ponto $p_v(X_v,Y_v,Z_v)$.

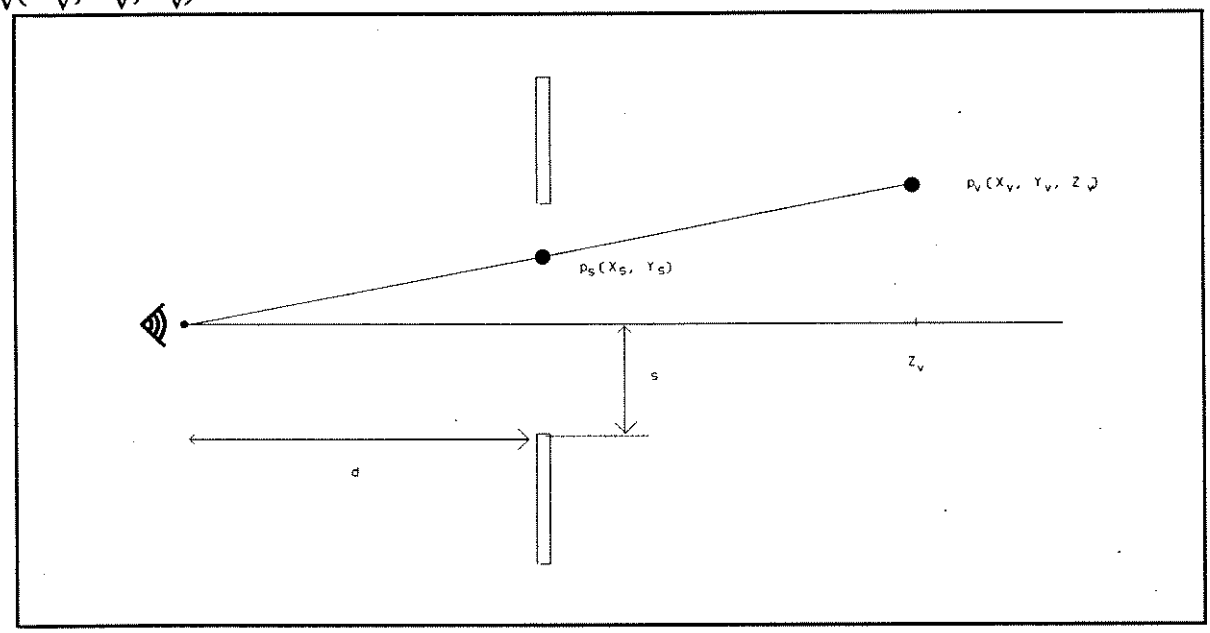


Fig. 2.14 - Projeção em perspectiva

Calcula-se então as coordenadas da tela 2D por triângulos semelhantes:

$$\frac{Y_s}{d} = \frac{Y_v}{Z_v} \qquad \textbf{(2.19)}$$

Então:

$$Y_s = \frac{Y_v}{Z_v} d \qquad \textbf{(2.20)}$$

Por similaridade para X:

$$X_s = \frac{X_v}{Z_v} d \tag{2.21}$$

Observe que X_S e Y_S são expressas na mesma unidade de d, é preferível expressá-los em frações sem dimensão dividindo, cada uma por s .

$$\begin{aligned} Y_s &= \frac{Y_v}{Z_v} \frac{d}{s} \\ X_s &= \frac{X_s}{Z_v} \frac{d}{s} \\ Z_s &= d \end{aligned} \tag{2.22}$$

Este tipo de coordenada de tela sem dimensão é bem útil para estabelecer-se uma independência do sistema de coordenadas do dispositivo de exibição. Com isto é possível executar todo sistema de geração de imagens e somente na última etapa converter para o sistema de coordenadas físico da tela usada.

A transformação de projeção em perspectiva é bem diferente das outras transformações até aqui analisadas. A principal diferença é a sua característica não linear o que impede uma representação por matriz. Para a transformada de perspectiva é utilizado um sistema de coordenadas homogêneas X_h, Y_h, Z_h, W que estabelece a linearidade necessária ao cálculo matricial. Após todas as operações matriciais é feita então a conversão de coordenadas homogêneas para coordenadas cartesianas da tela através das divisões:

$$\begin{aligned} X_s &= \frac{X_h}{W} \\ Y_s &= \frac{Y_h}{W} \end{aligned} \tag{2.23}$$

As equações (2.22) podem ser reescritas na forma matricial usando coordenadas homogêneas

$$[X_h, Y_h, Z_h, W] = [X_v, Y_v, Z_v, 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s & \frac{sZ_V - d}{dZ_v} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

de onde

$$X_h = X_v \tag{2.25}$$

$$Y_h = Y_v \tag{2.26}$$

$$Z_h = sZ_v \tag{2.27}$$

$$W = \left(\frac{sZ_v - d}{dZ_v}\right) Z_v + 1 = \frac{sZ_v}{d} \tag{2.28}$$

convertendo para coordenadas cartesianas da tela, de acordo com (2.23) têm-se as expressões de (2.22).

2.6 - Visibilidade

Trata-se da tarefa de resolver o problema de determinação de visibilidade de uma superfície a partir de um observador. Esta visibilidade pode ser **direta** ou **indireta**. A determinação da visibilidade direta parte da descrição de um conjunto de objetos no espaço e da definição de um ponto de vista do observador para definir que faces ou superfícies dos objetos não estarão expostas ao observador. Uma vez determinadas, estas faces não necessitarão de processamento nas etapas seguintes do processo de geração de imagem, economizando assim recursos. Esta tarefa também recebe o nome de Eliminação de Superfícies Escondidas (*hidden surface elimination*).

A determinação da visibilidade indireta inclue, no processo de geração de imagem, os objetos ou partes de objetos que, apesar de diretamente invisíveis, tornam-se visíveis em função do ponto de vista do observador e da presença de outros objetos em cena, como espelhos e transparências.

Determinação da Visibilidade Direta.

O problema de visibilidade direta é essencialmente um problema de ordenação, em relação ao observador, dos objetos de uma cena. De acordo com [SSSc 74], o que distingue os diversos métodos de tratamento deste problema é a forma de realizar esta ordenação.
O problema de visibilidade simples, ou seja, com um único ponto de vista fixo num único momento no tempo, é o clássico. É o problema de formação de imagem direta com iluminação direta de diversos pontos de luz. O algoritmo

para resolver este problema de visibilidade, forma a base para extensões que resolvem problemas de visibilidade mais complexos, como aqueles que consideram borrões de movimento, profundidade de campo ou efeitos de iluminação indireta. O problema da superfície visível para este caso simples de formação de imagem direta pode ser colocado como mostrado no quadro a seguir.

Algoritmo de determinação de visibilidade

```
DADOS
        um conjunto de superfícies em 3D;
        um ponto de vista;
        um plano de imagem orientado;
        e um volume de visão;
PARA cada ponto no plano de imagem dentro do volume de visão FAÇA{
        Determine qual ponto em qual superfície está mais próximo do ponto de vista numa linha do ponto de vista
        até o ponto no plano de imagem
}
```

O tratamento do Problema de Visibilidade.

Existem duas classes de algoritmos para resolver este problema:

- algoritmos contínuos e
- algoritmos discretos ou algoritmos que usam amostragem de pontos.

Na prática a maior parte dos algoritmos de tratamento de visibilidade combinam estas duas classes. Por exemplo, pode-se começar utilizando técnicas de subdivisão de algoritmos discretos, para depois aplicar técnicas contínuas para resolver subproblemas de apresentação como o de *Aliasing*.

Algoritmos Contínuos.

Os algoritmos contínuos operam fazendo a determinação da visibilidade sobre áreas contínuas cobrindo o plano de imagem inteiro. Cada pedaço de cada superfície será detectado independentemente da localização, tamanho, ...etc. Algoritmos contínuos de filtragem e sombreamento são usados para processar o que deve ser exibido. Os algoritmos contínuos são muito complexos e os objetos primitivos e efeitos que eles podem tratar são muito poucos, segundo [JGMH 88]. No capítulo 1 desta referência é apresentada uma taxonomia mais completa dos algoritmos de visibilidade.

Algoritmos discretos

Algoritmos discretos formam uma solução aproximada para o problema de visibilidade. Estes algoritmos determinam a visibilidade para um número finito de pontos amostrados e estimam a visibilidade dos objetos entre estes pontos. Dentre os algoritmos discretos destacam-se:

Algoritmo *Z-buffer*
Algoritmo *Ray-Tracing*
Algoritmo do Pintor
Algoritmo de *Scan-line*

Estes algoritmos operam com um modelo de propagação de luz muito simples. Admite-se que a luz viaja num meio homogêneo, em raios retos, e interage com objetos somente em suas superfícies de acordo com as regras da óptica geométrica. Não são considerados os efeitos de difração, fase, polarização e qualquer relação entre o tamanho do objeto e o comprimento de onda da luz. Não há nenhuma dependência no tempo em qualquer aspecto dos raios de luz e um número infinito de raios preenche a cena.

2.6.1 - Algoritmo Z-BUFFER

Este algoritmo foi inicialmente proposto na tese de doutoramento de E.Catmull [Catm 74] na universidade de Utah, entretanto a maior parte das referências da literatura é feita ao artigo [Catm 75], que apresenta um resumo do trabalho de tese. É seguramente o algoritmo mais simples para a remoção de superfícies escondidas. Ele requer um *Z-buffer* (ou *Depth-buffer*) ou seja uma memória que retenha os valores de profundidade para cada pixel da imagem que está sendo exibida. A escrita de um novo pixel na memória de vídeo é condicionada a uma comparação do seu valor de profundidade com o valor armazenado anteriormente no *Z-buffer*. Se o novo pixel estiver na frente do pixel já contido na memória de vídeo (ou seja, se o novo valor de profundidade for menor que o valor de profundidade anterior, em coordenadas do observador), o novo pixel é escrito na memória de vídeo. Se não for este o caso, nada é escrito. O algoritmo Z-BUFFER pode ser descrito resumidamente pela sequência mostrada no quadro a seguir. Por sua simplicidade, este algoritmo é o que está mais disponível em implementações em *hardware*.

Algoritmo Z-buffer

```
DADOS
 LISTA de poligonos {P1, P2, ....Pn}
 ARRAY z-buffer{x,y} /* iniciado com +∞ */
 ARRAY Intensidade{x,y}

INICIO
 PARA cada poligono P da LISTA de poligonos, FAÇA{
  PARA cada pixel(x,y) que intercepta P FAÇA{
   Calcule a profundidade em z de P em (x,y)
   SE a profundidade em z ( z-buffer [x,y] ENTAO{
    Intensidade[x,y] = intensidade de P em (x,y)
    z-buffer[x,y] = profundidade em z
    }
   }
  }
  Exiba o ARRAY Intensidade
FIM
```

2.6.2 - Algoritmo do Pintor

Trata-se um dos algoritmos mais antigos de eliminação de superfície escondida. Os primeiros trabalhos que descrevem este algoritmo são [Schu 69] e [NNSa 72]. O seu nome é uma analogia a forma de trabalho de um pintor criando uma obra de arte. Este algoritmo é também chamado Algoritmo *List-Priority*. A idéia básica consiste em classificar os elementos da cena em ordem de profundidade relativa à distância ao ponto de vista. Quando esta classificação estiver pronta, os elementos são mostrados nesta ordem. A técnica parece muito simples, porém a operação de classificação pode ser bem complexa, particularmente em dois casos:

1 - Quando houver sobreposição cíclica (*Cyclic Overlapping*), como mostrado na figura 2.15

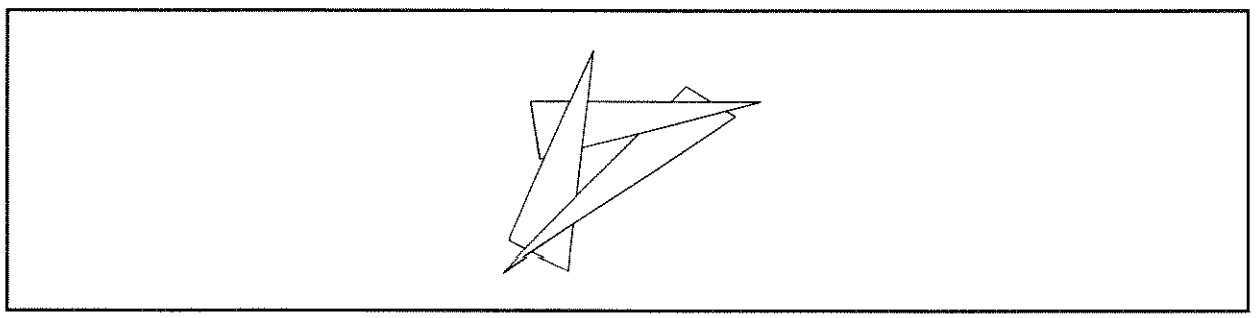


Fig. 2.15 - Polígonos com sobreposição cíclica.

2 - Quando os polígonos estão dispostos como na figura 2.16 e a classificação é feita por valores mínimos de Z.

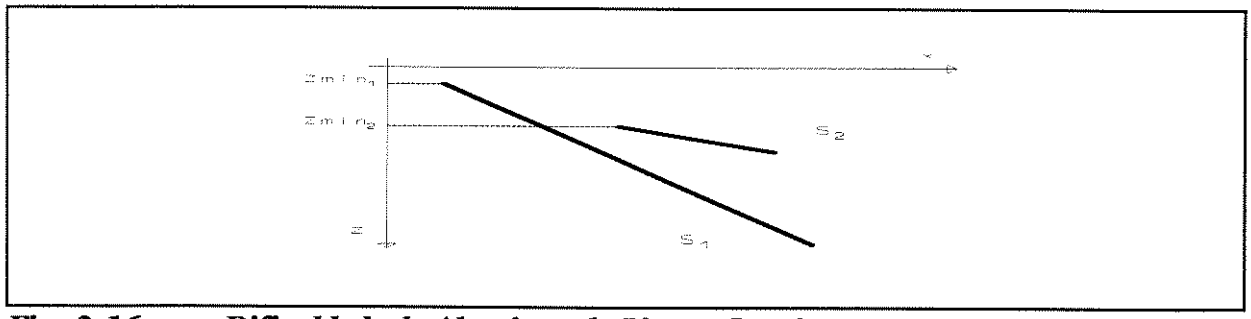


Fig. 2.16 - Dificuldade do Algoritmo do Pintor. S_1 vai aparecer antes de S_2 pois $Zmin_1 < Zmin_2$.

Para resolver estes problemas a classificação tem que ser precedida de uma divisão dos polígonos em fatias a serem classificadas. Uma técnica para esta divisão é apresentada em [NNSa 72].

O algoritmo básico do pintor pode ser resumido pela sequência de operações abaixo:

Algoritmo do pintor

```
DADOS
 LISTA de poligonos {P1, P2, ..., Pn}
 ARRAY de intensidades[x,y]
INICIO
 Ordene a LISTA de poligonos do fundo para frente
 PARA cada poligono P na lista de poligonos FAÇA{
  PARA cada pixel(x,y) que intercepta P FAÇA{
   intensidade[x,y] = intensidade de P em (x,y)
  }
 }
 Exiba o array intensidades
FIM
```

2.6.3 - Algoritmo Scan-Line

Os algoritmos *Scan-Line* de remoção de superfícies escondidas são baseados nos mesmos princípios de preenchimento de áreas para 2D. Basicamente é constituido dos seguintes passos:

- Cálculo da intercessão de uma linha da tela (*Scan-line*) com o polígono;
- Classificação das intercessões em ordem crescente das suas coordenadas X;

. Preenchimento de todos os pixels entre os pares de intercessões.

O cálculo de intercessões pode levar muito tempo, todavia é possível utilizar o fato de que somente algumas bordas do polígono interceptam uma linha da tela; também pode ser levado em conta que se um polígono intercepta uma linha é altamente provável que ele intercepte a linha seguinte. E mais, é possível calcular a coordenada X_{i+1} da intercessão na linha seguinte a partir da coordenada X_i da intercessão atual e da inclinação m da borda do polígono ela será

$$X_{i+1} = x_i + \frac{1}{m} \tag{2.29}$$

Os algoritmos *Scan-Line* podem ser de dois tipos, *Z-buffer Scan-Line* ou *Spanning Scan-Line*.

O algoritmo *Z-buffer Scan-Line* é uma simplificação do algoritmo Z-buffer pois a memória de profundidade (*Z-buffer*) é limitada à quantidade de pixel de uma linha de tela. Este algoritmo *Z-buffer scan-line* foi introduzido em [Myer 75].

O algoritmo *Spanning Scan-Line*, introduzido em [Watk 70], evita a determinação do valor de profundidade para cada pixel de toda uma linha através da introdução do conceito de *SPAN*. A figura 2.17 mostra a intercessão de dois polígonos com um plano *scan* (perpendicular ao plano da tela passando por uma

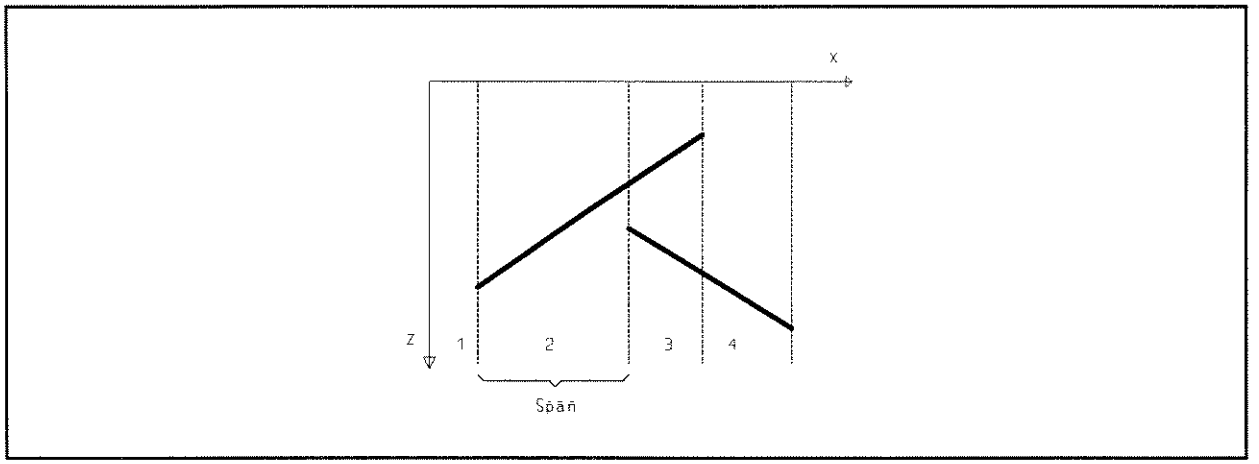


Fig. 2.17 - Segmentos em Spans.

linha horizontal na tela). Dividindo a linha da tela a partir de linhas perpendiculares que passam pelos extremos dos polígonos obtem-se o que é chamado *span*. Daquela mesma figura pode-se identificar 3 tipos de *spans*:

span vazio, como o *span* 1 da figura 2.17, quando deve ser exibida a cor de fundo.

span contendo um único segmento de intercessão com polígono, como os *spans* 2 e 4 da figura 2.17, neste caso os atributos do polígono correspondente deve ser mostrado na tela.

span contendo diversos segmentos de intercessão com polígonos, como o *span* 3 da figura 2.17. Somente nestes os valores de profundidade dos pixels são calculados. O segmento que estiver mais próximo terá os atributos do seu polígono exibidos.

O núcleo do algoritmo *Scan-Line* básico pode ser descrito resumidamente como mostrado no quadro abaixo.

Algoritmo Scan-line

```
DADOS
 LISTA de poligonos(P1, P2, ..., Pn)
 ARRAY Intensidade[x, y]
 ARRAY y-bucket[(top-scanline, ..., bottom-scanline), ]
 ARRAY pixel[pixel-a-esquerda, ..., pixel-a-direita]

INICIO
 PARA cada poligono P na LISTA de poligono FAÇA{
  Insira P num y-bucket de acordo com o seu maior valor em y
  }
 PARA cada scanline DESDE top-scanline ATE bottom-scanline FACA{
  SE o y-bucket[scanline] não estiver vazio FAÇA{
   PARA cada poligono P no y-bucket[[scanline] FAÇA{
    calcule os pontos extremos de P que interceptam a scanline
    insira os pontos extremos na LISTA x-ordenado
    }
   }
  PARA cada pixel DESDE pixel-a-esquerda ATE pixel-a-direita FAÇA{
   Contrua uma LISTA z-depth a partir da LISTA x-ordenado
   Construa os elementos visiveis a partir da LISTA z-depth
   Determine a intensidade do pixel a partir dos elementos visiveis
   }
  Atualize a LISTA x-ordenado para a proxima scanline
  }
FIM
```

2.6.4 - Algoritmo Ray-Tracing

O algoritmo *ray-tracing* é baseado na idéia de simular os efeitos óticos produzidos numa cena por raios de luz oriundos de uma ou mais fontes. Esta idéia sempre foi conhecida e inicialmente foi evitada por requerer muitos cálculos de intercessão dos raios com os objetos da cena.

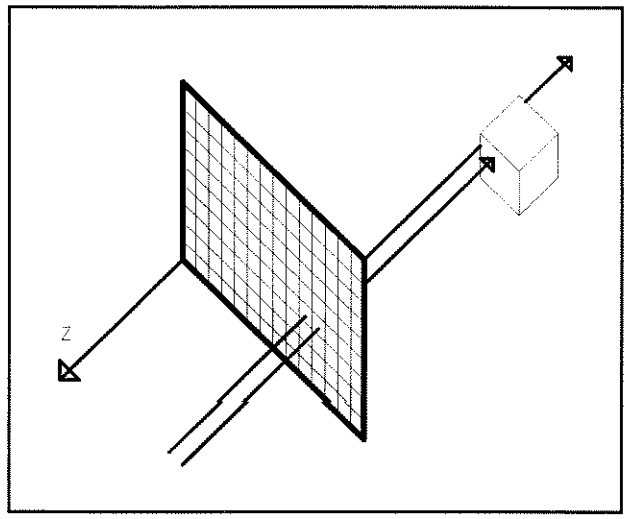


Fig. 2.18 - Ray-tracing simples.

Se a simulação partisse de uma fonte de luz e determinasse os caminhos de todos os seus raios, muitos raios não atingiriam o observador, ou seja o processamento seria ineficiente. [Appe 68] sugeriu que a simulação dos raios deveria ser feita seguindo o caminho inverso, ou seja a partir do observador até a fonte. Mesmo assim, até o início dos anos 80 o uso deste tipo de algoritmo era bastante limitado por ser considerado um "algoritmo de força bruta" [SSSc 74]. Os trabalhos [KaGr 78] e [Whit 80] apresentam a implementação do algoritmo de *ray-tracing* em conjunção com novos modelos de iluminação e de tonalização. Os efeitos obtidos nas imagens geradas segundo o algoritmo de *ray-tracing* aplicado ao modelo de iluminação proposto por Whitted, estimularam a maior parte dos trabalhos na área de geração de imagens na década de 80. A figura 2.18 mostra a forma mais simples do algoritmo de remoção de superfícies escondidas por *ray-tracing*. Neste caso, a cena está descrita em coordenadas do mundo e a transformação de perspectiva não foi aplicada. É considerado que o observador está posicionado no infinito do eixo Z. Assim todos os raios atravessam a tela paralelamente ao eixo Z. Cada raio sai do observador atravessa o centro de cada pixel na tela e vai para a cena. O caminho de cada raio é então seguido para determinar quais objetos na cena são interceptados por ele. Todos os objetos na cena devem ser examinados para todos os raios. Se um raio atinge um objeto todas as possíveis intercessões do raio e do objeto são determinadas. Isto pode levar a diversas intercessões em muitos objetos. As intercessões são então classificadas de acordo com a sua profundidade. As características da superfície do objeto mais próximo da tela são usadas para definir a tonalidade do pixel.

O algoritmo de *Ray-Tracing* de remoção de superfície escondida, sem considerar o modelo de iluminação adotado, pode ser descrito resumidamente pela sequência mostrada no quadro a seguir.

Algoritmo Ray-tracing

```
INICIO
 PARA cada pixel P FAÇA{
  forme um raio R no espaço de objeto a partir da posição da camera e atravessando o pixel
  intensidade = trace (R)
  use intensidade para colorir o pixel P
  }
FIM

PROCEDURE trace (raio)
 INICIO
  PARA cada entidade E na cena FAÇA{
   calcule a intercessão de E com o raio
   }
  SE o raio não atinge entidades na cena ENTÃO{
   RETORNE(intensidade do fundo)
   }
   SENÃO{
    Ache a entidade E com a intercessão mais próxima
    calcule a intensidade I no ponto da intercessão
    RETORNE(modelo-intensidade(I, trace(raio-refletido), trace(raio-refratado))
    }
 FIM
```

2.7 - Iluminação

A determinação dos componentes da imagem que são dependentes da propagação da luz através da cena e independentes da posição do observador, incluindo o processo de distribuição da luz incidente das fontes até a superfície dos objetos (**iluminação direta**) e do cálculo da luz incidente em cada objeto que não é devido à propagação direta da fonte de luz para o objeto (**iluminação indireta**) é denominada **iluminação**. A iluminação direta é referenciada como o problema de sombreamento (*shadowing*), o qual envolve a determinação da visibilidade a partir da fonte de luz. A iluminação indireta pode incluir reflexões (difusa e especular) de outros objetos e luz atravessando objetos translúcidos e transparentes.

No tratamento da iluminação direta pode-se aplicar, de certa forma, os mesmos algoritmos que são usados para a questão da visibilidade, considerando agora a visibilidade do objeto pelo ponto de iluminação. Estes algoritmos são aplicados como um pré-processamento do algoritmo de visualização propriamente dito. Como exemplo deste pré-processamento pode-se citar [AWGr 78] que sugere a pré-definição de "polígonos de sombra" pela aplicação de algoritmo de visualização a partir da fonte luminosa. Estes polígonos de sombra são, então, combinados com os polígonos originais na aplicação de um dos algoritmos de visualização na formação da imagem, diminuindo a intensidade de iluminação dos objetos sob a sombra.

Outro exemplo deste preprocessamento é [Will 78] que define um "*Depth map*" que é um *Z-buffer* computado a partir do ponto de vista da fonte de luz.

O último exemplo, aqui considerado é [Crow 77a], que introduz o conceito de "volume de sombra" que contorna os elementos sob sombra. Estes volumes servem como dados de entrada para um algoritmo do tipo *Scan-line*. Este pré-processamento é discutido e melhorado em [Berg 86].

Neste tratamento deve ser considerado, ainda, o tipo de iluminação. A cena pode ser iluminada de forma uniforme em todas as direções, num tipo de iluminação chamada Difusa. A cena pode ter diversas fontes de luz, o que pode levar a efeitos diferentes sobre as superfícies dos objetos.

2.8 - Tonalização (Shading)

É a tarefa que determina a intensidade de luz que deixa um objeto, dadas a distribuição de luz incidente e as propriedades óticas das superfícies do objeto. É portanto a tarefa que estabelece a intensidade e as cores das superfícies a serem exibidas dos objetos em cena.

Das propriedades óticas das superfícies do objeto destacam-se a **Refletância** e a **Transparência**. A refletância determina quanto da luz incidente é refletida pela superfície. Se uma superfície tem diferentes refletâncias para luz com diferentes comprimentos de onda, ela irá parecer colorida para o observador. A transparência define quanto uma superfície deixa passar de luz através dela.

2.8.1 - O Processamento de Tonalização

Um modelo de tonalidade é usado para calcular a intensidade de luz que será vista quando se observa um objeto. Este modelo deve considerar as propriedades de **Refletância** e de **Transparência** (ou **Refratância**).

Refletância

A luz refletida numa superfície pode ser dividida em duas componentes: Difusa e Especular. Quando a luz incide sobre uma superfície difusa, ela é re-irradiada igualmente em todas as direções. Exemplos de superfícies difusas são parede e folha de papel. Uma superfície especular ideal re-irradia a luz em somente uma direção. Exemplos de superfícies especulares são espelhos e superfícies metálicas.

Para a geração de imagens com realismo é necessário então modelar, igualmente, estes dois tipos de componentes da luz. A componente difusa é a mais fácil de calcular pois, como ela é igual em todas as direções, basta calcular o seu valor na direção normal à superfície que recebe a luz. Isto é feito pela lei de Lambert

$$N \bullet L \qquad (2.30)$$

onde: N é o vetor-normal à superfície no ponto P,
L é o vetor que aponta para a fonte de luz, e
• significa produto escalar.

A componente especular é mais difícil de ser calculada pois depende também da posição de observação.

Além das componentes especular e difusa, uma terceira componente é usada em computação gráfica para modelar a luz vinda de diversas fontes de luz diretas ou indiretas. É a chamada componente de luz ambiental. O uso desta componente significa que objetos colocados na sombra de outros objetos não são simplesmente pretos.

Transparência

O modelo de tonalidade deve considerar ainda a questão de refração nos objetos. A luz a ser refratada pode ser dividida em duas componentes uma especular e outra difusa. A componente difusa é útil para a modelagem de objetos translúcidos, onde a luz ao atravessar o objeto se espalha e perde intensidade. A componente especular permite o modelagem de vidros limpos. A refração difusa pode ser gerada decrescendo a intensidade da luz refratada e espalhando a contribuição da intensidade a cada ponto da superfície refratária numa área finita. Este cálculos consomem muito tempo de processamento e os modelos de tonalidade em geral empregam somente o efeito especular. Por isto a modelagem da Refratância é referenciado somente como modelagem da Transparência. A modelagem de superfícies transparentes pode ser feita adicionando a componente especular da refração ao cálculo da intensidade de luz refletida.

Para a geração de imagens com realismo, a forma ideal de tonalizar os objetos seria aplicar sobre cada ponto da superfície destes, os modelos de iluminação pontuais. Obviamente isto consumiria muito tempo. Utilizam-se então técnicas que reduzam este tempo, sem prejudicar visivelmente a qualidade da imagem.

A escolha da técnica a ser utilizada depende do tipo de superfície e da aplicação específica.

Muitas vezes, objetos com superfície plana podem ser tonalizados com alto grau de realismo, utilizando uma única intensidade para todos os pontos da superfície.

As superfícies planas podem ser representadas por um conjunto de polígonos, em cada um destes pode ser aplicado então um modelo de tonalização. Cada um destes polígonos pode ser tonalizado com uma intensidade constante, ou pode-se variar o tom usando valores interpolados da intensidade.

2.8.2 - Tonalização por Intensidade Constante

Sob certas condições, um objeto com superfícies planas pode ser tonalizado realisticamente usando intensidades constantes. No caso onde a superfície é exposta unicamente à luz ambiente, com tom constante, sem desenhos na superfície e sem textura ou sombras pode gerar uma representação bem apurada da superfície. Para superfícies iluminadas por uma fonte pontual, a superfície pode ser tonalizada com intensidade constante se a fonte de luz e o ponto de observação estiverem muito distantes da superfície.

Uma superfície curva que é representada por um conjunto de planos pode ser tonalizada com intensidades constantes nos planos se estes forem bem pequenos. Isto pode gerar imagens bem realisticas, especialmente se a fonte de luz e o ponto de observação estiverem bem distantes do objeto.

Com este método, a intensidade é calculada num ponto interior de cada superfície plana e toda a superfície é tonalizada com este valor de intensidade.

Este método é também referenciado como *Poligonal Shading*, ou ainda, *Flat Shading* e foi proposto por Bouknight em [BoKe 70].

O método de tonalização por intensidade constante é o mais simples de ser implementado, entretanto ele apresenta algumas deficiências. A primeira e seguramente a mais marcante, é que quando as orientações dos planos adjacentes mudam abruptamente, a diferença nas intensidades de cores de cada plano produz efeitos irreais de descontinuidade de tons. Estes efeitos podem ser minorados se a superfície for dividida em um número maior de polígonos. Entretanto esta solução eleva o custo do processamento.

Outra deficiência deste método é que as diferenças das tonalidades de duas superfícies planas adjacentes são acentuadas na fronteira entre elas pelo efeito chamado de Faixas de Mach (*Mach-bands*). Este efeito é visualizado por apresentar listras escuras e claras nas bordas onde houver uma discontinuidade na magnitude ou na derivada da intensidade.

Para reduzir os efeitos de descontinuidade de tonalidade entre superfícies adjacentes deve ser adotada alguma técnica de interpolação dos tons. Existem duas técnicas de interpolação, a primeira interpola a intensidade das cores (método Gouraud) e a segunda interpola o vetor normal (método Phong). Estes dois métodos são descritos a seguir.

2.8.3 - Tonalização por GOURAUD

Este método de interpolação de intensidade de tons foi desenvolvido por Gouraud [Gour 71a] [Gour 71b]. Ele remove as descontinuidades de intensidade entre as superfícies planas adjacentes pela variação linear da intensidade sobre cada superfície plana de forma que os valores da intensidade coincidam nas bordas. Neste método, os valores de intensidade ao longo de cada linha de varredura que atravessa uma dada superfície plano são interpolados a partir da intensidade nos vértices do polígono que representa a superfície.

Este método de tonalização parte das intensidades de cores nos vértices do polígono que descreve a superfície a ser tonalizada. Estas intensidades são calculadas considerando os vetores normais em cada um dos vértices. Se estes vetores não forem conhecidos eles devem ser calculados pela média das normais de cada superfície que contem o vértice. Estes vetores normais são então usados no modelo de iluminação para gerar os valores de intensidade em cada vértice. Quando a tonalização for feita em cores, a intensidade de cada componente da cor deve ser calculada nos vértices. O método pode ser combinado com um algoritmo de superfície escondida para preencher os poligonos visíveis em cada linha de varredura.

Para ilustrar o funcionamento do método de Gouraud, considere a superfície poligonal mostrada na figura 2.19. A intensidade no ponto P é determinada pela seguinte sequência de interpolações lineares:

- Primeiro, obtêm-se as intensidades nas intercessões do polígono com a linha de varredura onde está o ponto P, respectivamente, os pontos Q e R.

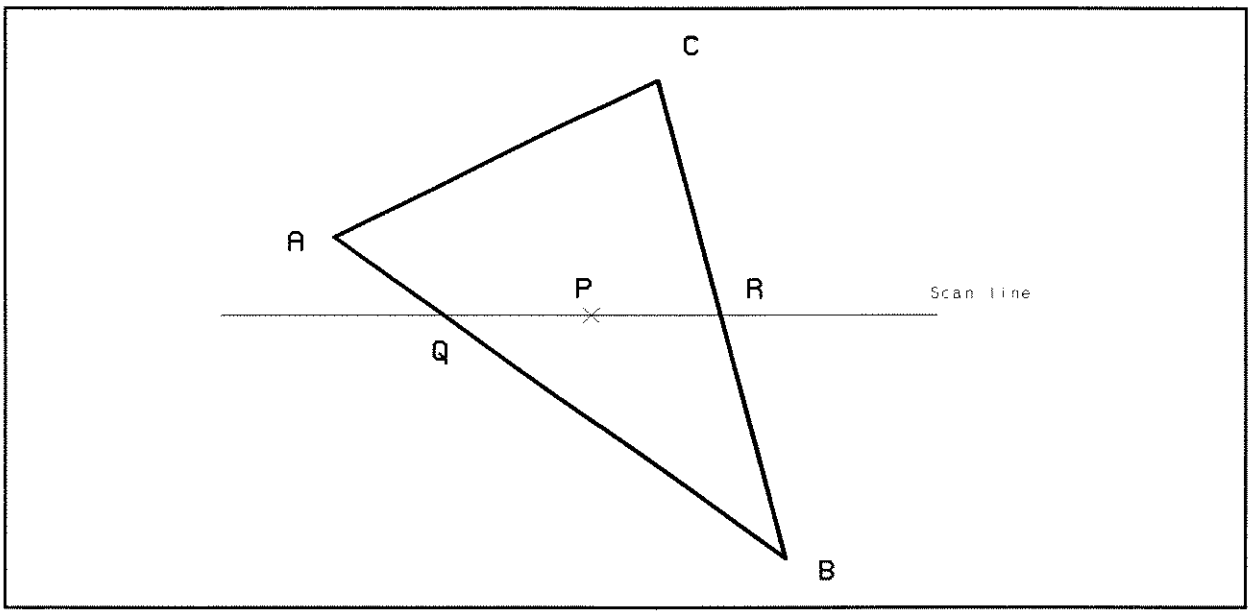


Fig. 2.19 - Interpolação de tonalização

A intensidade no ponto Q é determinada pela interpolação linear das intensidades nos vértices A e B do polígono.

$$I_Q = uI_A + (1-u)I_B \qquad 0 \leq u \leq 1 \tag{2.31}$$

onde $u = AQ/AB$

Da mesma forma é determinada a intensidade no ponto R, interpolando a intensidade dos vértices B e C.

$$I_R = wI_B + (1-w)I_C \qquad 0 \leq w \leq 1 \tag{2.32}$$

onde $w = RC/BC$

- E por último, a intensidade no ponto P é determinada pela interpolação linear das intensidades em Q e R.

$$I_P = tI_Q + (1-t)I_R \qquad 0 \leq t \leq 1 \tag{2.33}$$

onde $t = QP/QR$

O cálculo da intensidade ao longo da linha de varredura pode ser feito de forma incremental. Para dois pixels em t_1 e t_2 numa linha de varredura têm-se que:

$$I_{p_2} = t_2 I_Q + (1-t_2)I_R \tag{2.34}$$

e

$$I_{p_1} = t_1 I_Q + (1-t_1) I_R \qquad \textbf{(2.35)}$$

subtraindo (2.35) de (2.34) têm-se:

$$I_{p_2} - I_{p_1} = \Delta t \times \Delta I \qquad \textbf{(2.36)}$$

o que permite que

$$I_{p_2} = I_{p_1} + \Delta t \Delta I \qquad \textbf{(2.37)}$$

ao longo da linha de varredura.

A tonalização por Gouraud remove as descontinuidades associadas com o método de Intensidade constante, mas tem algumas deficiências. Brilhos na superfícies são as vezes mostrados com formas anômalas e a interpolação linear da intensidade não elimina de forma satisfatória as faixas de Mach.

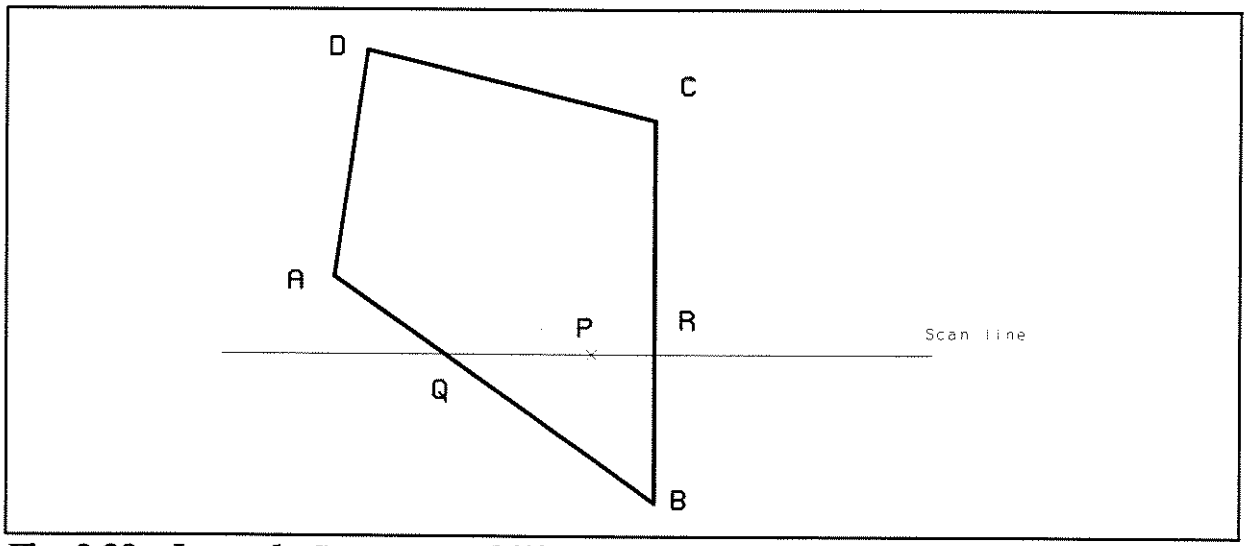


Fig. 2.20 - Interpolação num quadrilátero.

Outra deficiência está associada ao formato do polígono a ser tonalizado. Nos polígonos triangulares, todas as intensidades de cor dos vértices participam do cálculo das intensidades de cor das arestas na primeira interpolação linear e estas, por sua vez, influenciam a tonalização dos pontos internos ao polígono na segunda interpolação linear. Este é o caso mostrado na figura 2.19. No caso de polígonos com quatro ou mais vértices, a intensidade de cor de um dos vértices pode não interferir no cálculo da tonalização numa dada linha de varredura. Este é o caso mostrado na figura 2.20, onde o vértice *D* não influencia na determinação da tonalização do ponto *P*.

Para se evitar este último problema e melhorar a tonalização dos polígonos com mais de três vértices é recomendado fazer uma triangularização do polígono antes de aplicar o algoritmo.

Algoritmo de Tonalização por Gouraud

```
I = Itop;
PARA cada linha de varredura entre Ytop e Ytop+n, FAÇA{
        /* determine xe o ponto de intercessão a esquerda */
        xe = xe + delta_X(Aresta Esquerda);
        /* determine I(xe,y) a intensidade na intercessão esquerda */
        I(xe,y) = I = I + delta_S(Aresta Esquerda);

        /* determine xd o ponto de intercessão a direita */
        xd = xd + delta_X(Aresta Direita);
        /* determine I(xd,y) a intensidade na intercessão direita */
        I(xd,y) = I = I + delta_S(Aresta Direita);

        /* determine delta_int a variação de intensidade na linha */
        var_int = I(xe,y) - I(xd,y);
        tamx = xd - xe;
        delta_int = var_int/tamx;

        /* Tonaliza a linha */
        Iy = I(xe,y)
        PARA x entre xe+1 e xd, FAÇA{
            I(x,y) = Iy = Iy + delta_int
        }
}
```

Algoritmo Incremental de determinação da tonalização.

O algoritmo de tonalização a ser descrito nesta seção considera que um quadro de imagem é tratado linha a linha de varredura (*scan-line*) de cima para baixo. Ele supõe que a cena a ser exibida é descrita por um conjunto de polígonos côncavos já projetados na *viewport* e que o cálculo de tonalização dos vértices de cada polígono já foi feito preliminarmente.

Assim sendo, inicia-se o algoritmo determinando o conjunto de arestas visíveis e para cada aresta calcula-se:

1 - Y_{top} o número da primeira linha de varredura que intercepta a aresta;
2 - n, a quantidade de linhas seguintes que a interceptarão;
3 - as coordenadas X_{top} e Z_{top} do ponto mais alto da aresta;
4 - as inclinações ΔX e ΔZ da aresta;
5 - a intensidade I_{top} da superfície no ponto mais alto da aresta; e
6 - a variação da tonalização é calculada como sendo:

$$\Delta S = \frac{S_A - S_B}{n}$$

onde S_A e S_B são as intensidades de cor nos dois extremos da aresta e n o número de linhas que interceptam esta aresta.

Com estas informações a tonalização é feita linha por linha de varredura de forma mais fácil. Para uma dada linha de varredura, basta considerar os dados acima listados das arestas atingidas pela linha de varredura, aqui chamadas de *Aresta Esquerda e Aresta Direita*.

2.8.4 - Tonalização por PHONG

Uma melhoria na tonalização pode ser conseguida, aproximando o valor da normal à superfície em cada ponto ao longo de uma linha de varredura e depois calculando o valor da intensidade usando o valor aproximado da normal em cada ponto. Este método desenvolvido por Bui Tuong Phong [Phon 75], também referenciado como Método de Interpolação do Vetor Normal, exibe brilhos mais realísticos e reduz enormemente os efeitos de *Mach-band*.

O cálculo de intensidade em cada ponto da linha de varredura usando um valor aproximado do vetor-normal (modelo de Phong) produz um resultado mais realístico que a interpolação direta da intensidade a partir das intensidades nos vértices (modelo de Gouraud). Entretanto, o compromisso é que o método de Phong requer um maior número de cálculos.

Uma maneira de produzir imagens com mais realismo sem contudo aumentar muito a quantidade de cálculos, é aplicar o modelo de Phong somente nos polígonos em que se espera haver problemas de brilho (*highlight*). Nos demais polígonos aplica-se o modelo de Gouraud.

2.9 - Aliasing

A principal saída de um processo de síntese de imagem é um monitor de vídeo segundo a tecnologia Raster. Neste monitor a imagem exibida é uma representação discretizada de uma imagem contínua. O processo de converter um sinal contínuo num sinal discreto é chamado **Amostragem** (*sampling*). O espaçamento entre as amostras limita a quantidade de detalhes que podem ser representados num sinal discreto. Qualquer tentativa de representar detalhes mais finos pode resultar na introdução de um sinal distorcido. Este sinal

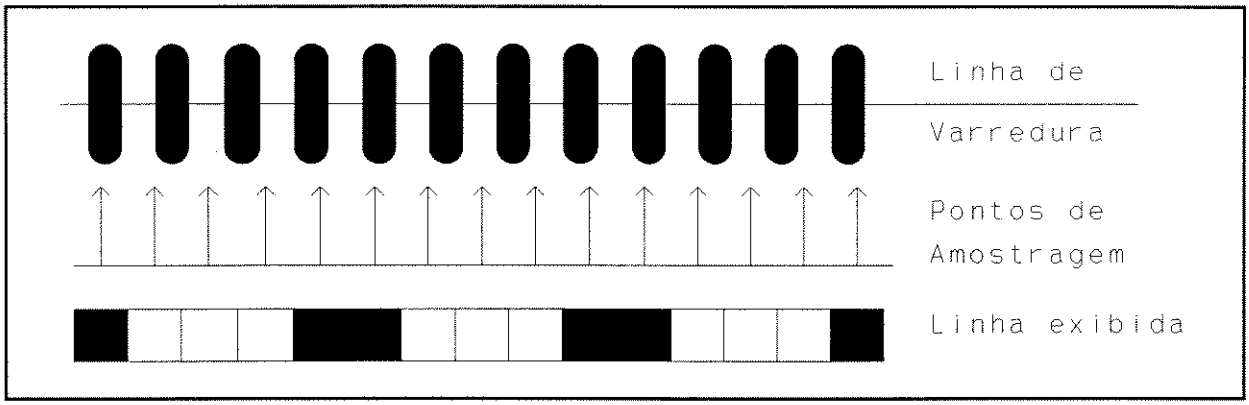


Fig. 2.21 Efeito Aliasing numa linha de varredura.

distorcido é chamado de ***Alias*** e o processo de geração de *alias* é conhecido como ***Aliasing***. O processo para prevenir aliasing é chamado de ***Anti-aliasing***. As imagens de tela de vídeo nas quais as bordas aparecem com degraus ou serrilhado são um exemplo de *aliasing* espacial causado pela amostragem sem filtragem. Um exemplo do efeito de *aliasing* numa imagem é mostrado na figura 2.21. Nesta figura têm-se uma fileira de barras sendo representadas nos pontos de amostragem correspondentes aos pixels. A linha exibida é mostrada com a distorção resultante.

De acordo com a teoria da amostragem é possível recontruir um sinal contínuo $I(x)$ a partir de sua amostragem em n pontos regularmente espaçados em algum intervalo em x. Para tanto, o sinal original não deve conter componentes de frequência maiores que $n/2$ ciclos por intervalo de amostragem; se ele contiver frequências maiores que $n/2$ o sinal reconstruido será sempre incorreto. Em outras palavras, um sinal discreto de n pontos pode representar frequências abaixo de $n/2$ ciclos por intervalo de amostragem; todas altas frequências no sinal original serão representadas incorretamente (ou causarão *aliasing*). Se forem olhadas as componentes de frequência do sinal reconstruido, não será possível dizer se elas são legítimas ou são distorções introduzidas pela sub-amostragem do sinal original.

Em computação gráfica, o sinal $I(x,y)$ é uma função bi-dimensional que representa a intensidade de luz que atravessa o plano da tela. Pode-se amostrar esta intensidade para obter sua representação baseada nos pontos da tela (pixel). Estas amostras são armazenadas na memória de quadro, e a função intensidade é reconstruida pelo *hardware* do monitor. Eventualmente $I(x,y)$ contém componentes de alta frequência. Isto ocorrendo, quando $I(x,y)$ for amostrado para determinar as intensidades dos pixels, problemas de *aliasing* serão inevitáveis. A solução mais simples é aumentar o número de amostras, pois à medida que n aumenta, podem ser representadas frequências mais altas. Isto tem

o inconveniente de que o tempo de processamento também cresce. Além disso, não adianta aumentar o número de amostra, pois é o *hardware* que limita o número de pixels que podem ser exibidos. Esta limitação do *hardware* pode ser contornada se for feita uma super-amostragem (amostra-se numa resolução maior que a da tela e depois para cada pixel é tirada a média dos valores de intensidade). Da mesma forma, isto também consome muito tempo de processamento. O que deve ser feito, então, é arranjar uma forma de remover os componentes de alta frequência em $I(x,y)$ antes de amostrar.

Novamente, da teoria de amostragem tem-se que: dada uma função $I(x,y)$, é possível gerar uma função $I'(x,y)$, que tem as mesmas componentes de frequência que $I(x,y)$ para frequências menores que uma certa frequência ω_0 e não possue componentes de frequência maior que ω_0. Para tanto deve ser feita a convolução

$$I'(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} I(\alpha,\beta)H(x-\alpha)H(y-\beta)d\alpha d\beta \qquad \textbf{(2.39)}$$

com um filtro apropriado H

$$H(u) = \frac{\sin(\omega_0 u)}{\pi u} \qquad \textbf{(2.40)}$$

Fazendo-se assim, pode-se amostrar $I(x,y)$ para obter a intensidade do pixel (x,y), sabendo que nenhum *aliasing* irá ocorrer. Esta convolução pode ser entendida como a média ponderada da função intensidade onde H indica o pêso relativo de um ponto como função da distância (u) do centro do pixel. Infelizmente esta convolução é impossível na prática, pois seria necessário o cálculo de uma integral dupla com campos infinitos para cada pixel. Também a função intensidade é muito complicada para ser convolucionada analiticamente. Assim sendo, devem ser usadas simplificações para calcular esta convolução num tempo razoável.

As duas maiores simplificações são: - utilizar um filtro de amostragem mais simples, e - aproveitar as propriedades de coerência da função intensidade. Os filtros de amostragem mais comumente utilizados são *box filter* e Gaussiana truncada. Em ambos, o filtro não se estende sobre uma área maior que três por três pixels (uma aproximação do maior lóbulo de um filtro de amostragem ideal quando a frequência de corte é n/2) e muitas vezes não é maior que a área de um único pixel.

A simplificação da função intensidade começa com a observação de que à medida que é amostrado através de um polígono, o sinal vai mudar localmente

muito lentamente (propriedade da coerência). Somente quando forem tratadas as bordas do polígono é que pode ocorrer uma grande mudança na intensidade. Assim, pode ser admitido que dentro de um pixel a intensidade de cada polígono é constante. Isto leva a simplificação de que é necessário calcular uma única vez a tonalidade de cada polígono dentro de um pixel, economizando assim processamento. Consequentemente, a área e a posição que um polígono cobre dentro de um pixel, juntamente com um único cálculo de tonalização para cada polígono, são suficientes para determinar a contribuição para a função intensidade naquele pixel.

A aplicação da teoria da amostragem em computação gráfica, como descrita nesta seção, foi inicialmente proposta por [Crow 77b]. Diversos trabalhos surgiram a partir daí, modificando principalmente o tipo de filtro a ser aplicado e na aplicação da coerência para a simplificação da função intensidade. Um estudo mais detalhado da aplicação da teoria da amostragem no tratamento de *Alias*, com uma sequência interessante de imagens pode ser feito no capítulo 14 da referência [FDFH 90] e uma comparação de diversos métodos de tratamento de *alias* é feita em [Crow 81].

2.10 - Textura

Para fornecer a ilusão de realismo, os sistemas de geração de imagem devem estar aptos a exibir cenas complexas. Por exemplo, se estiver sendo modelada uma sala de visitas, deve ser permitido incluir retratos na parede ou tapetes persas no chão. Estes objetos, ricos em detalhes (textura), podem ser modelados, por exemplo, por diversos polígonos individuais. Mas modelando assim, o número de polígonos a serem tratados cresce muito, e pode comprometer o desempenho do sistema de geração e exibição da imagem. As técnicas computacionais de tratamento eficiente destes detalhes recebem o nome de **Tratamento da Textura do Objeto**. Existem dois métodos tradicionais de tratamento da textura:

- O mapeamento da textura (*texture mapping*), e
- Textura procedural (*Procedural Texture*).

O **Mapeamento de Textura** foi introduzido na tese de doutorado de Catmull [Catm 74] para fornecer a ilusão de complexidade a um custo razoável. Este método parece com a cobertura com papel de parede dos polígonos existentes na cena. Cada vértice nos polígonos com textura contém coordenadas num espaço de textura em 2-D. A medida que cada pixel é tonalizado, as coordenadas de textura são interpoladas e feita uma procura numa matriz de cores que

contém a informação de textura (Mapa de Textura). A cor na matriz é usada como a cor do polígono no pixel específico.

Os detalhes de textura são componentes de alta frequência na cena, fazendo com que o tratamento de textura seja muito sensível ao *aliasing*. O método de mapeamento de textura foi aperfeiçoado por Blinn e Newell [BlNe 76] que introduziram o uso de filtros para o *antialiasing*.

Um estudo mais detalhado das técnicas de mapeamento de textura pode ser encontrado em [Heck 86].

O mapeamento da textura pode gerar grandes tabelas de cores, elevando assim as necessidades de espaço em memória. Uma forma de minimizar este espaço é o uso do tratamento da textura por procedimentos computacionais de acordo com o método chamado de **Textura Procedural**. Neste método, a textura é representada por uma função que calcula diretamente o valor de textura sempre que for chamada, ao invés de obtê-lo de uma tabela. Como exemplos de trabalhos com textura procedural podem ser citados [Gard 84], [FFCa 82], [Perl 85], [Peac 85].

Entretanto, segundo [Aman 87], o uso de procedimentos para a determinação da textura tem sido evitado, pois através deles é difícil eliminar os efeitos de *aliasing*.

Além da informação de cor, o mapa de textura pode fornecer ainda informações sobre as propriedades da superfície do objeto. [Blin 78] sugere que as perturbações ao vetor normal da superfície podem ser armazenadas no mapa de textura, permitindo assim o modelamento de superfícies rugosas.

2.11 - Composição de Imagens

Composição de Imagens é a combinação de duas ou mais imagens para criar uma nova. Com a Composição de Imagens, se for preciso alterar parte de uma imagem, não é necessário regerar toda a imagem, basta gerar isoladamente a porção a ser modificada e depois COMPOR com a imagem complementar. Outra utilidade da Composição de Imagem (talvez a mais importante) é que se alguma porção da imagem a ser exibida não tiver sido gerada (por exemplo, foi capturada via camera diretamente para a memória de vídeo), a Composição de Imagens é o único meio de incorporá-la à imagem.

Que espécies de operações podem ser feitas na composição?

O valor de cada pixel na imagem composta é determinado a partir das imagens componentes de várias formas:

Numa Sobreposição, os pixels da imagem dianteira devem ter definidos atributos de Transparências da mesma forma que são definidos outros atributos (por exemplo: valores de cores primitivas RGB).

O valor do pixel na imagem composta é tomado da imagem trazeira, a menos que a imagem dianteira tenha um atributo de Não Transparência naquele ponto. Neste caso, o valor é retirado da imagem dianteira.

Na mistura de duas imagens, o pixel resultante é uma combinação linear dos valores dos dois pixels componentes.

Porter & Duff[PoDu 84] apresentaram um mecanismo para a composição de imagens usando valores de combinação e transparência que podem ser armazenados e operados diretamente na memória de quadro. Este mecanismo é aplicado em diversos projetos de circuitos de memória de quadro (por exemplo, Silicon Graphics [AkJe 88]).

2.12 - Consideração sobre novos modelos e conclusão

O objetivo do estudo descrito neste capítulo é servir de base a uma proposta de arquitetura que possa ser aplicada às tarefas descritas. Com isto em mente, observou-se que é importante que a arquitetura de um sistema de processamento para computação de imagem não seja rigidamente organizada para tratar as informações de imagem somente de uma forma, pois a medida que os recursos computacionais se tornam mais poderosos, novos modelos e novos conceitos são introduzidos para a geração de imagens. Da mesma forma, conceitos já conhecidos teoricamente passam a ter a aplicação viabilizada pelos novos equipamentos. Exemplo disso é a aplicação do modelo de transferência de luz por propagação de ondas, como em [Mora 81]. A aplicação desta teoria esbarrou na grande necessidade de memória e no grande número de cálculo requerido.

Em 1984, [GTGr 84] introduziu um novo modelo de iluminação para a síntese de imagens em 3D. Neste modelo, a interação de luz entre as superfícies refletindo difusamente é calculada resolvendo equações de equilíbrio de campos de radiação. Este método é chamado de **Radiosidade**. O método de Radiosida-

de calcula a reflexão difusa da luz entre os objetos. A superfície dos objetos é considerada como tendo uma difusão de luz "ideal" ou, como é normalmente chamada de, Reflexão Lambertiana, isto é, a luz incidente é refletida de uma superfície em todas as direções com igual intensidade. Cada superfície é tratada como uma fonte de luz secundária de forma que após ter sido refletido de uma superfície, o passado do raio não é mais considerado. A radiosidade de uma superfície *b* é definida como sendo a taxa total de energia que deixa a superfície e é igual a soma da energia emitida *e* e da energia refletida e_r. Este método de **Radiosidade** dá uma boa descrição dos efeitos globais de iluminação e gerou diversos trabalhos de aplicação, como por exemplo: - Em [BuDe 89] é descrita uma arquitetura em VLSI para o tratamento de geração de imagens usando o modelo de radiosidade para luz. - Em [BaWi 90] é descrita a implementação de um algoritmo baseado no modelo de iluminação por radiosidade para máquinas multiprocessadoras. Todos os trabalhos aplicando radiosidade procuram agilizar o seu tratamento.

Outro exemplo de novos conceitos é descrito em [BaCh 90]. Ali é descrito um modelo de reflexão (*Full Wave Reflection model*) que é indicado para ser aplicado principalmente em objetos de superfície metálica. O modelo é baseado na análise do espalhamento eletromagnético por superfícies rugosas. A análise *Full Wave* considera as componentes óticas-físicas, da reflexão especular e também do espalhamento difuso oriundos de uma superfície aleatoriamente rugosa.

Além dos novos conceitos e modelos, novos algoritmos são sempre propostos para as tarefas de computação gráfica, como por exemplo [Mulm 89] que propôs um novo (à época) e, segundo ele, eficiente algoritmo para a remoção de linhas escondidas.

Uma forma de estar aberto à todas estas possíveis mudanças é através da utilização de sistemas de propósito geral e executá-las através de programação (*software*).

Portanto, o projeto de uma arquitetura para aplicação em computação gráfica deve considerar uma grande facilidade de programação em alto nível. Todo o esforço para a implementação de uma arquitetura que previlegie uma determinada tarefa pode ser desperdiçado com o surgimento de um novo algoritmo ou pela aplicação de um novo conceito.

No projeto de uma arquitetura para computação de imagem deve-se aproveitar os conceitos e métodos de geração de imagens já consagrados. Deve-se

identificar algumas tarefas que podem ser agilizadas por *hardware* e por último permitir que novos métodos de processamento sejam aí programados.

Este capítulo procurou apresentar, resumidamente, os aspectos mais importantes envolvidos na Síntese de Imagens Realísticas, de forma a fornecer ao leitor uma visão introdutória porém abrangente do tópico.

Capítulo 3

Circuitos de Exibição de Imagem

Uma vez introduzidos os conceitos básicos, estuda-se aqui os pontos relativos à exibição final da imagem. Com isto pretende-se fornecer ao leitor uma visão da importância deste item no processo global de geração de imagens. Este estudo é realizado a partir do monitor de vídeo (segundo a tecnologia *raster*), abrangendo diversas técnicas de circuitos de saída gráfica, e nestas procura-se evidenciar o comprometimento de cada uma com o desempenho do processamento gráfico.

3.1 - Unidade de Exibição de Imagem

Unidade de Exibição de Imagem é o conjunto de elementos que permite a exibição de informações textuais, gráficas e pictóricas para o usuário. É formada por um circuito de Saída Gráfica e pelo Monitor de Vídeo, como mostrado na figura 3.1

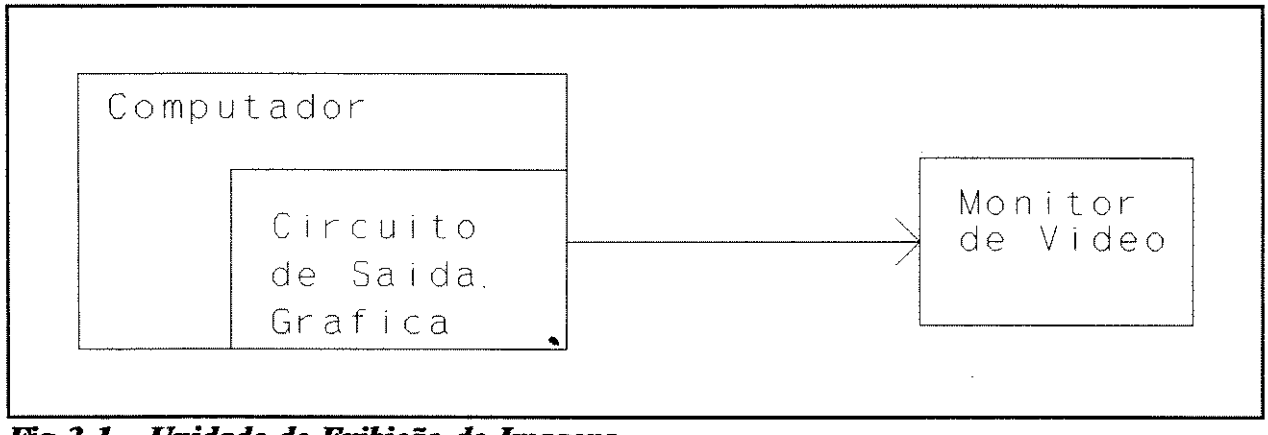


Fig.3.1 - Unidade de Exibição de Imagens

O circuito de Saída Gráfica ou simplesmente *Interface* Gráfica é um circuito, interno ao sistema de computação, que reúne os circuitos de ligação com o monitor de vídeo. Nestes incluem-se os circuitos de adequação elétrica, como por exemplo: — conversores de nível e os circuitos de adequação temporal, principalmente: — memória de quadro e registradores de deslocamentos. Além destes, este circuito pode ser composto ainda por uma unidade de processamen-

to gráfico, que permite a execução de funções gráficas, independentemente do processador principal do sistema de computação.

Monitor de vídeo é o elemento de exibição de imagem. Os monitores de vídeo podem ser de diversas tecnologias: *Raster*, *Storage Tube*, Plasma, Cristal Líquido e outras. Vide [Mamm 91] para um estudo mais específico sobre as diversas tecnologias para dispositivos de exibição. No contexto deste trabalho considera-se somente a tecnologia de monitor de vídeo *raster*. Estes podem ser:

- **Monocromáticos** — quando só dispõem de um canhão de elétrons que incide sobre uma tela revestida de fósforo. Neste tipo de monitor, além da ausência de cor (preto), só é possível exibir uma única cor. As cores mais comuns são verde, branco, azul ou ambar.

- **Policromáticos** — quando dispõem de mais de um canhão de elétrons, com cada um associado a uma das cores aditivas primárias - Verde, Azul e Vermelho. A combinação de diversas intensidades de cada um destes feixes permite exibir diversas cores além do preto.

Neste capítulo são descritas as alternativas de projeto de uma unidade de exibição de imagem.

3.2 - A Ligação Circuito de Saída Gráfica - Monitor de Vídeo.

A ligação do circuito de saída gráfica com o monitor de vídeo é feita através de sinais de vídeo e de sinais de sincronismo. Os **Sinais de Vídeo** indicam o nível de luminância dos pontos na tela do monitor de vídeo. Os **Sinais de Sincronismo são dois**, sincronismo horizontal e sincronismo vertical. Eles disparam a deflexão do feixe de elétrons no tubo no sentido horizontal e no vertical, respectivamente.

A ligação com monitores monocromáticos

Para esta ligação o sinal de vídeo pode ser digital ou analógico.

O **sinal de vídeo digital** pode ser em ponto aceso-apagado ou em escala de cinza.

Para usar o sinal de vídeo digital ponto aceso-apagado, o monitor deve possuir uma única entrada de sinal digital. Esta entrada reconhece dois níveis de tensão, um nível indica o ponto na tela aceso e o outro nível indica o ponto apagado na tela. Um exemplo deste tipo de ligação é a utilizada pelo cartão controlador de vídeo Hercules para computadores pessoais compatíveis com o PC-IBM.

Em alguns monitores existe mais que uma entrada digital de sinal de vídeo, de forma que a combinação lógica destas entradas define diversos níveis de acendimento do ponto na tela. É estabelecida assim uma escala discretizada de níveis de cinza.

O **sinal de vídeo analógico** tem uma variação contínua numa faixa de tensão padronizada. Cada nível deste sinal corresponde a um nível de acendimento do ponto na tela.

Os dois sinais de sincronismo podem ser enviados de três formas para o monitor: separados, compostos entre si ou compostos com o sinal de vídeo.

a) Alguns monitores requerem a ligação dos **sinais de sincronismo separados**, ou seja é necessário ligar dois fios, um para o sincronismo horizontal e outro para o sincronismo vertical.

b) Outros monitores requerem que os sinais de sincronismo sejam enviados somados num único sinal chamado **sinal de sincronismo composto**. Dentro do monitor há um circuito de filtro para a separação dos mesmos.

c) **Sinal de vídeo composto**. Em alguns monitores monocromáticos os sinais de sincronismo e o sinal de vídeo podem ser enviados somados através de um único fio. Dentro do monitor há o circuito de separação dos mesmos.

A ligação com monitores policromáticos

Esta ligação é feita através de 3 sinais de vídeo, correspondentes às cores primárias aditivas, a saber: vermelha (R), verde (G) e azul (B). Cada um destes sinais aciona um dos canhões de elétron no monitor. Estes sinais podem ser digitais ou analógicos.

Sinais de vídeo RGB digital. Em alguns monitores policromáticos só é possível dizer se a cor de um determinado ponto possue ou não a componente de uma determinada cor primária. Ou seja, a entrada dos sinais de cada cor primária (RGB) é digital. Isto limita a quantidade de cores possíveis na tela a somente oito cores. O conjunto de cores possível é dado na tabela 3.1.

Tab. 3.1 - Cores do RGB digital

rgb	preto
rgB	azul
rGb	verde
rGB	ciano
Rgb	vermelho
RgB	magenta
RGb	amarelo
RGB	branco

onde:

r = sem vermelho R = com vermelho

g = sem verde G = com verde

b = sem azul B = com azul

Sinais de vídeo RGB digital e de intensidade - RGBI. Em alguns monitores além das entradas digitais RGB de vídeo, é preciso enviar mais uma linha de sinal de vídeo com a informação de intensidade. Isto implica em estabelecer dois níveis de intensidade para cada uma da oito combinações possíveis de cores primárias. Elevando, assim, para 16 o número de cores possíveis. Este tipo de interface de sinal é usado nos sistemas pessoais PC-IBM para uso com o cartão controlador CGA. O conjunto de cores possível é dado na tabela 3.2.

Tab. 3.2 - Cores do RGBI Digital

	I	i
rgb	cinza	preto
rgB	azul intenso	azul
rGb	verde intenso	verde
rGB	ciano intenso	ciano
Rgb	vermelho intenso	vermelho
RgB	magenta intenso	magenta
RGb	amarelo intenso	amarelo
RGB	branco intenso	branco

onde:

r = sem vermelho

g = sem verde

b = sem azul

i = sem intensidade

R = com vermelho

G = com verde

B = com azul

I = com intensidade

Sinais de vídeo RGB digital multinível. Em alguns monitores existe mais de uma entrada digital para cada cor primária. Isto implica em estabelecer diversos níveis de intensidade para cada uma das cores primárias. Um exemplo deste

tipo de ligação é a utilizada com o cartão controlador EGA para microcomputadores pessoais PC-IBM. Os monitores para este tipo de interface possuem duas entradas para cada cor primária. Desta forma é possível estabelecer 4 níveis para cada cor primária. A combinação destas eleva para 64 o total de cores possíveis.

Sinais de vídeo RGB Analógico. Em outros monitores as entradas dos sinais de cores primárias é analógica. Desta forma é possível estabelecer qualquer nível de intensidade para cada uma das cores primárias. Isto permite uma melhor qualidade de imagem na tela, por causa da maior quantidade de combinações de cores primárias.

O sincronismo para monitores coloridos. Os sinais de sincronismo podem ser enviados de forma idêntica aos monitores monocromáticos, a saber: - sincronismos separados e sincronismo composto. Uma terceira forma de envio dos sinais de sincronismo é através da composição destes com o sinal de cor verde.

Os sinais de sincronismo e os sinais de vídeo digitais normalmente são compatíveis com TTL. Os sinais de vídeo analógicos obedecem aos níveis de tensão definidos numa das normas descritas a seguir na seção 3.3.

3.3 - Normas para a Ligação de Monitores.

A ligação entre o monitor de vídeo e o circuito de saída gráfica obedece a uma das normas existentes para os sistemas de televisão comercial. Estas normas definem a faixa de nível de tensão e a banda de passagem para os sinais de vídeo, o número de linhas de varredura, a quantidade de quadros por segundo, etc. Existem diversas normas internacionais adotadas nos diversos países. Na América do Norte e Japão é adotado como padrão o formato NTSC (National Television System Committee), na França é usado o formato SECAM (Sequentiel Coleur à Memoire). Na maioria dos países entretanto é adotado o formato PAL (Phase Alternation by Line). Deve-se frisar que estes formatos admitem variações diferentes (que são designadas pelas letras: A, B, C, E, G, H, I, K, K1, L, M e N) que definem praticamente um novo formato, por exemplo, no Brasil é adotado o formato PAL/M. O entendimento destas normas é importante para o projeto do circuito de saída gráfica para sistemas computacionais, pois muitas vezes os manuais dos circuitos de conversão D/A (digital/analógico) para vídeo referem-se a estas normas, como por exemplo em [Brok 85].

Das especificações destas normas destacam-se as diferenças existentes nos formatos, ou seja na amplitude e na forma de onda, dos sinais de vídeo definidos por cada uma delas para ser usado em televisores e em monitores de vídeo.

Todas as normas definem o sinal de vídeo composto com o formato mostrado na figura 3.2. Em torno de um nível de referência igual a zero, chamado de nível de **Apagamento**, o sinal de vídeo varia. No sentido positivo, a variação indica mudança na intensidade da informação de imagem. No sentido negativo a variação indica o disparo de sincronismo. Nesta forma de onda, destacam-se alguns níveis:

- o nível **Branco de Referência** indica o nível de tensão associado ao branco mais intenso; e
- o nível **Preto de Referência** corresponde ao nível de preto na tela.

Os nomes destes níveis são derivados dos seus efeitos em monitores Preto e - Branco. Ainda nesta forma de onda define-se como ***set-up*** a diferença entre o nível de preto de referência e o nível de apagamento. Esta diferença é usada para garantir o apagamento do feixe de elétrons durante os intervalos de retraço.

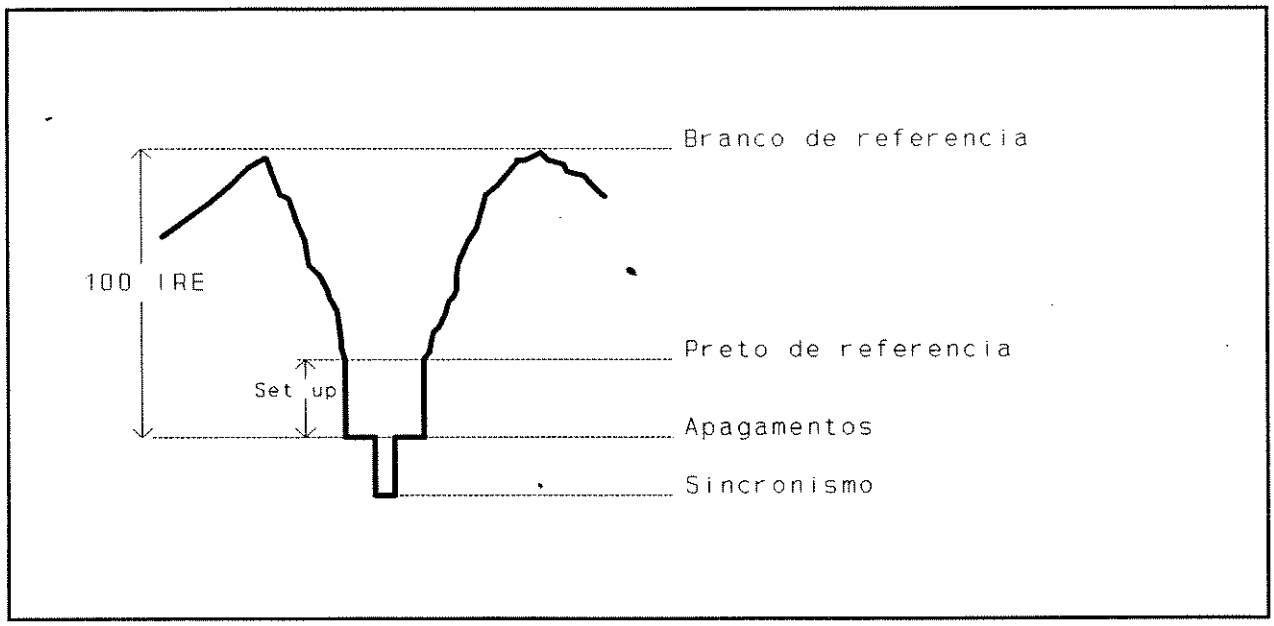


Fig.3.2 - Forma de onda de sinal de vídeo composto.

Para fins de comparação entre as diversas normas, os níveis do sinal de vídeo são descritos em relação à amplitude da parte de informação de imagem do sinal de vídeo, chamado de 100 IRE. Por exemplo, na norma NTSC o sinal de

sincronismo possue uma amplitude de -40 IRE, o que significa que ele tem uma amplitude negativa igual a 40% da amplitude da informação de imagem. A amplitude total do sinal de vídeo no formato NTSC é portanto de 140 IRE. Um resumo da comparação entre os níveis relativos de tensão utilizados nas principais normas é mostrado na tabela 3.3. Desta tabela pode-se notar que as diferenças entre as principais normas estão na amplitude relativa do sincronismo e no valor de *set-up*.

Tab. 3.3 - Níveis IRE

Níveis	NTSC	PAL	SECAM
Apagamento	0	0	0
Branco de Referência	100	100	100
Sincronismo	-40	-43	-43
Set-up	5-10	0	0 (colorido) 0-7 (monocromatico)

A maior parte das normas internacionais define uma amplitude de tensão de 0,714 volts entre o nível de apagamento e o nível de branco de referência, ou seja para a parte da forma de onda correspondente ao sinal de vídeo propriamente dito.

A tabela 3.4 resume a comparação entre os principais formatos de sinais de vídeo em termos de níveis de tensão e de corrente para cargas de 75 ohms e 37,5 ohms.

Desta tabela destacam-se duas normas estabelecidas pela *EIA - Electronic Industries Association* para o formato NTSC. A norma EIA RS170-A define os sinais para televisão em preto e branco e colorida e se aplica a sistemas que exibem imagens com 525 linhas a uma taxa de 30 quadros entrelaçados por segundo, de acordo com [EIA 57]. A segunda norma é a norma EIA RS343-A, que se aplica a sistemas de televisão com resolução mais alta, como por exemplo sistemas com 1024 linhas de 1024 pontos a uma taxa de 30 quadros entrelaçados por segundo, ou sistemas com 512 linhas de 512 pontos a uma taxa de 60 quadros por segundo sem entrelaçamento [EIA 69]. Esta norma estabelece a temporização para um número limitado de formatos de tela, mas indica como estabelecer a temporização para outros formatos.

Tab. 3.4 - Comparação entre os diversos formatos de vídeo.

Formatos	Niveis de tensão	Volts	mA (75ohms)	mA (37,5ohms)
NTSC RS170-a	Branco de Referência	1.00	13.3	26.6
	Preto de Referência	0.075	1.00	2.00
	Apagamento	0.0	0.0	0.0
	Sincronismo	-0.40	-5.33	-10.66
NTSC RS343-a	Branco de Referência	0.714	9.52	19.1
	Preto de Referência	0.071	0.952	1.89
	Apagamento	0.0	0.0	0.0
	Sincronismo	-0.286	-3.81	-7.63
PAL	Branco de Referência	0.714	9.52	19.1
	Preto de Referência	0.0	0.0	0.0
	Apagamento	0.0	0.0	0.0
	Sincronismo	-0.307	-4.09	-8.19
SECAM	Branco de Referência	0.714	9.52	19.1
	Preto de Referência	0.049	0.653	1.3
	Apagamento	0.0	0.0	0.0
	Sincronismo	-0.307	-4.09	-8.19

Estas duas normas diferem entre si, ainda, na tensão entre o nível de apagamento e o nível de branco de referência. Segundo a norma RS170 o nível de branco é 1,00 volt, na norma RS343-A este nível é de 0,714 volts. Um cuidado especial com respeito a esta diferença deve ser tomado, pois muitos equipamentos comerciais, em particular os VCR's (*Video Cassete Recorder*), obedecem a norma RS170-A.

3.4 - Circuito de Saída Gráfica.

Este circuito envia para o monitor de vídeo os sinais de vídeo e sincronismo. As principais partes deste circuito são: Memória de quadro, controle de vídeo e interface com o monitor.

A memória de quadro retem o conjunto de valores que representa a imagem. O sinal de vídeo é formado em função do seu conteúdo. Cada ponto na tela tem uma posição de memória associada. Alterando o conteúdo desta posição o estado do ponto na tela será modificado de forma correspondente.

O controle de vídeo gera os sinais de sincronismo, controla a formação do sinal de vídeo e coordena o acesso pelo processador para alterar o conteúdo da memória de quadro.

A seção de interface com o monitor é composta principalmente por conversores de sinais de acordo com os padrões aceitos pelo monitor.

O circuito de saída gráfica pode ser de duas formas básicas, a saber:

Bit-mapped. Cada ponto na tela tem uma posição de memória de quadro associada e cada posição tem um ponto na tela. Com isto é possível alterar cada um dos pontos elementares (**pixel**) da tela do monitor de vídeo. Neste tipo de circuito de controle é necessário dispor de pelo menos um *bit* de memória para cada ponto da tela.

Caractere. Cada ponto na tela tem uma posição de memória de quadro associada, mas cada posição pode corresponder a diversos pontos. Ou seja os pontos da tela são agrupados em conjuntos indivisíveis para alteração (caracteres). Isto significa que este conjunto é codificado e armazenado numa única posição de memória. A alteração desta posição de memória implica em modificar todos os pontos da tela associados a esta posição. As vantagens principais deste tipo de controle são: - a economia de posições de memória - e a velocidade de alteração da tela, pois modificando uma única posição de memória modifica diversos pontos na tela ao mesmo tempo.
A seguir é feita a descrição das principais estruturas de circuitos de controle de vídeo.

3.5 - Circuito de Vídeo Bit-Mapped Monocromático Digital(pontos acesos e apagados).

Neste tipo de circuito cada ponto está associado diretamente a um *bit* de memória. O estado deste *bit* vai determinar se o ponto estará aceso ou apagado. É o esquema mais simples de circuito de vídeo e pode ser representado de uma forma genérica pela figura 3.3

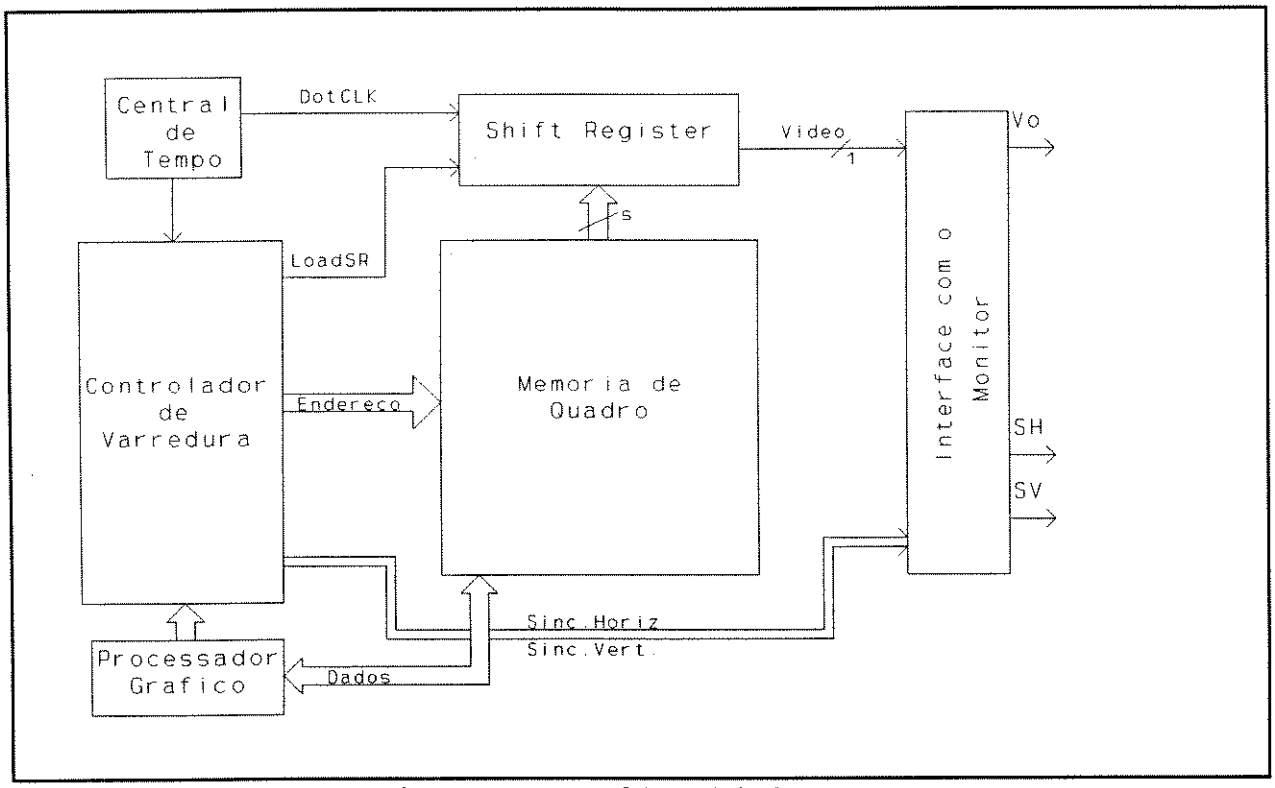


Fig.3.3 - Controlador de vídeo monocromático digital

Nesta representação tem-se os seguintes blocos:

Interface com o monitor. Representa todos os elementos de ligação com o monitor. Este bloco deverá tratar os sinais de vídeo e sincronismos e enviá-los somados ou separados para o monitor. Na figura, este bloco tem como saída os sinais **SH**, **SV**, e **Vo** que representam respectivamente as saídas de sincronismo horizontal e vertical e a saída de vídeo convertidas para os níveis elétricos compatíveis com o monitor utilizado. Este bloco possui como entrada os sinais digitais de vídeo serial, e de sincronismo horizontal e vertical.

Shift register. Representa todos os circuitos responsáveis pela serialização da informação gráfica para envio para o monitor. O nome do bloco evidencia o seu componente principal, a saber, um registrador de deslocamento (*shift-register*). Na figura, este bloco gera o sinal serial de vídeo - **Video**, com base na temporização do sinal **DotCLK** (relógio de pontos - Dot-Clock). A informação dos valores dos *bits* a serem deslocados (representados por **s** na figura) é carregada neste bloco a partir do sinal **LoadSR**, proveniente do bloco controlador de varredura.

Memória de Quadro. Representa todos os circuitos de armazenamento da informação gráfica. Cada *bit* desta memória corresponde diretamente a um elemento (ponto) da imagem na tela - *pixel*. Na figura, o controlador de

varredura fornece constantemente o endereço da posição (barramento **Endereco**) de onde será retirado o dado para o carregamento do *Shift-register*. O processador gráfico pode ter acesso a este bloco através do barramento de dados, para atualização dos *bits* correspondentes aos *pixels*.

Controlador de varredura. Representa todos os circuitos que controlam os acessos à memória de quadro para o carregamento do *Shift register* e para a alteração da informação armazenada. Neste bloco também estão representados os circuitos de geração dos sinais de sincronismo para a interface com o monitor.

Central de tempo. Representa os circuitos da base de tempo de todo o cartão. Composto por contadores e osciladores. Na figura é destacado o sinal **DotCLK**, gerado neste bloco.

Processador gráfico. Representa todos os circuitos de processamento gráfico local, principalmente no tratamento de primitivas gráficas. Nesta representação genérica, deve-se ressaltar que em alguns projetos este processador está integrado ao circuito controlador de varredura. Os sinais de controle do acesso do processador gráfico, bem como os *bits* de endereços são enviados ao controlador de varredura para que este coordene os acessos a memória de quadro.

O funcionamento do circuito de vídeo monocromático digital.

A informação da imagem é retirada da memória de quadro, agrupada em *s bits* em paralelo. No circuito monocromático digital, cada *bit* retirado corresponde a um *pixel* numa linha de varredura. Estes *bits* carregam um registrador de deslocamento (*shift register*).

Uma vez carregados no *shift register* estes *bits* são enviados em série para a interface com o monitor numa taxa que é definida pela frequência de varredura vertical, pelo número de linhas por quadro, pela resolução de pontos na linha e pelos intervalos de retraço, horizontal e vertical. Esta taxa é normalmente chamada de *Dot Clock*.

O carregamento do *shift register* é comandado pelo controlador de varredura através do sinal LoadSR, de forma que numa linha de varredura com *l* pontos são feitos *l/s* acessos.

Todo o sequenciamento destas operações é baseado num circuito de base de tempo que fornece os sinais de relógio para todo o circuito. A temporização

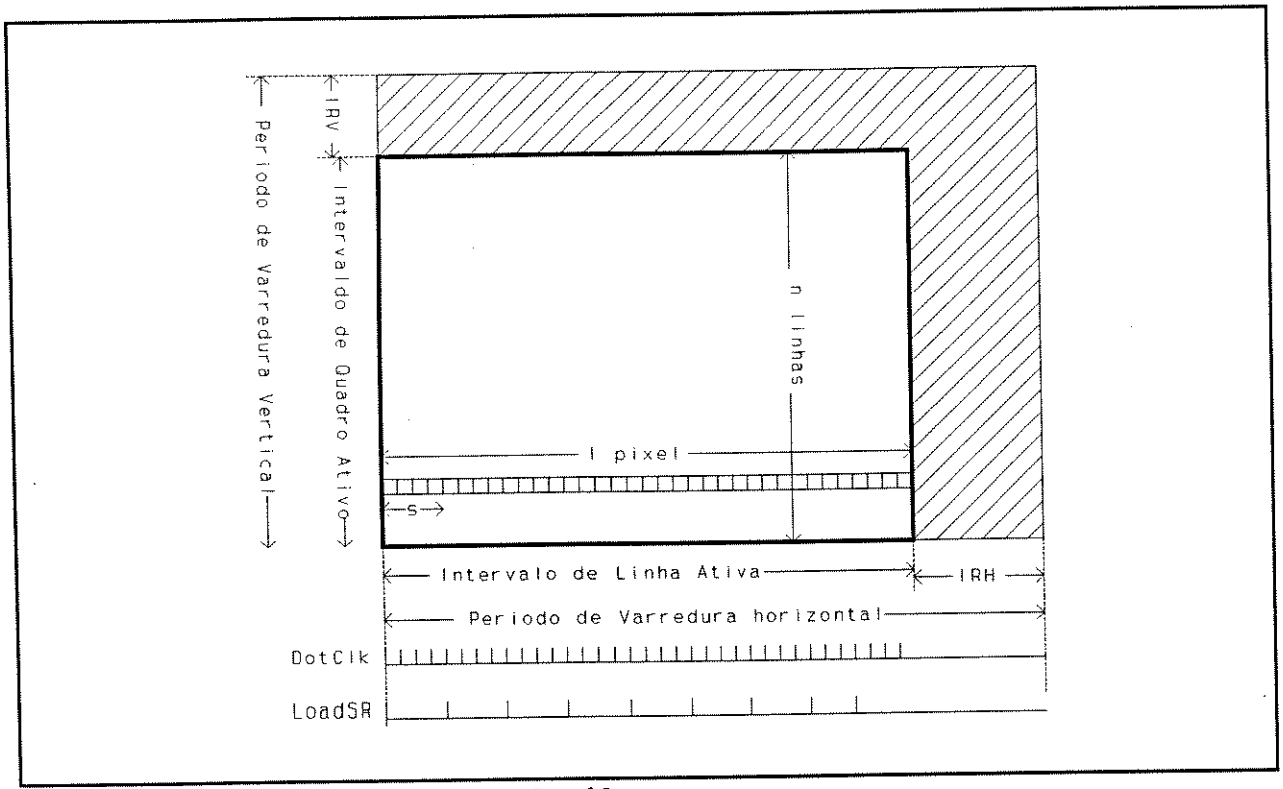


Fig. 3.4 - Distribuição dos tempos de vídeo.

de todo o circuito é baseada nas relações de tempo mostradas na figura 3.4. Nesta figura o período de varredura vertical é dividido em intervalo de retraço vertical (IRV), quando a tela está apagada e intervalo de quadro ativo, quando a tela exibe o conteúdo da memória de quadro. Da mesma forma o período de varredura horizontal é dividido em intervalo de retraço horizontal (IRH), quando a linha é apagada e intervalo de linha ativa, quando é exibido o conteúdo da linha que está sendo varrida.

Os valores dos intervalos de retraço e dos períodos de varredura, vertical e horizontal, são parâmetros do monitor de vídeo utilizado. De posse destes valores é possível determinar a frequência dos principais sinais de relógio. Por exemplo, o *dot clock* é calculado como

$$DotClk = (l + lr) \times F_h \qquad (3.1)$$

onde:

l é o número de pontos visíveis na linha ativa,
lr é o número de pontos invisíveis no retraço horizontal, e
F_h é a frequência de varredura de linha

O valor de lr é determinado em função da relação percentual do intervalo de retraço horizontal e o intervalo de linha ativa especificada pelo monitor utilizado.

3.6 - O Acesso do Processador Gráfico à Memória de Quadro.

Além do acesso do controlador de varredura à memória para a manutenção da informação na tela, esta deve permitir acessos pelo processador de tratamento gráfico.

Como os acessos pelo controlador têm que ser atendidos sempre, sob pena de gerar distorções na imagem na tela, a taxa de transferência entre o processador e a memória de quadro é extremamente sacrificada. Para que o acesso pelo processador não entre em conflito com os acessos pelo controlador, algumas técnicas são utilizadas. As mais comuns são:

- **Permitir acessos pelo processador só no retraço**. É a técnica mais fácil de ser implementada e mais barata, mas implica num maior comprometimento da taxa de transferência. De acordo com esta técnica o processador deve esperar os intervalos de retraço e somente nestes instantes executar suas transferências. Tipicamente a memória fica disponível entre 30% a 43% do tempo para o processador.

- **Compartilhamento de Acessos**. É possível ainda compartilhar os acessos do controlador com os acessos do processador durante os intervalos de quadro ativo e linha ativa. Ou seja, dentro dos ciclos de carregamento do *Shift Register* pode ser colocado um ciclo de acesso pelo processador. Esta técnica requer um circuito arbitrador dos acessos que privilegie os acessos do controlador. De acordo como esta técnica é possível elevar para 50% até 60% o tempo disponível para o processador. Com o uso de memórias VRAMs esta disponibilidade sobe para cerca de 95%, como será visto na seção 3.10.

- **Dupla Página de Memória.** A solução mais adotada pelos sistemas gráficos de mais alto desempenho é a duplicação de toda a memória de quadro, de forma que enquanto o controlador de varredura exibe o conteúdo de uma página a outra está completamente disponível para o processador.

- **Memória Virtual de Quadro**. É baseada no princípio de memória *cache*, ou seja, além da memória de quadro lenta e grande existe uma memória de rápido acesso com o tamanho correspondente a uma fração da tela.

Enquanto um determinado elemento gráfico está sendo tratado pelo processador ele ocupa uma região na memória virtual que é um espelho da região onde ele ficará na memória de quadro. Quando este tratamento termina toda a região é transferida para a memória de quadro. Esta técnica é também chamada de *Pixel-cache* [ABMa 88].

A taxa de transferência entre o processador gráfico e a memória de quadro é um grande "gargalo" para um sistema de geração de imagem com realismo. Um *chip* de memória típico para aplicação em projeto de memória de quadro possue um tempo de acesso à posição aleatória (t_{cyc}) da ordem de 250 nseg. Para acessos em posições consecutivas dentro de uma mesma linha (*row*), algumas memórias permitem acessos no modo paginado (*page mode*), a duração deste tipo de acesso, chamada de t_{pm}, é cerca de 125 nseg. Estes tempos limitam a taxa máxima de acesso à memória de quadro entre 4 até 8 milhões de acessos por segundo. Considerando a resolução da tela e os acessos do controlador, este é um valor muito baixo para diversas aplicações. As técnicas de processamento e o desenvolvimento da tecnologia de construção dos processadores já permitem atingir velocidade de processamento que esbarram nesta taxa de transferência.

3.6.1 - Formas de Acesso

A forma mais usada para aumentar a taxa de transferência para a memória de quadro é fazer com que num único acesso sejam operados diversos pontos na tela. Por exemplo, num circuito monocromático como o da figura 3.3, se o processador tiver acesso à memória de vídeo com 16 *bits* será possível operar 16 *pixels* por acesso. A disposição destes pontos na tela depende de como está organizada a memória de quadro. São três as formas de organizar a memória de quadro para os acessos do processador, a saber:

- **Pontual** - nesta forma somente um *pixel* é operado por acesso;
- **Linear** - nesta forma um número fixo de *pixels* na mesma linha horizontal é operado por vez;
- **Retangular** - nesta forma um número fixo de *pixels* formando um retângulo na tela é operado por acesso.

A forma Pontual é mais usada para circuitos nos quais os pontos na tela são representados por mais de um bit. A taxa de acesso para cada *pixel* é sempre a mesma e é igual à taxa de transferência para a memória de quadro. Os outros modos permitem o aumento desta taxa mas este aumento depende do que está sendo desenhado, como será visto a seguir.

A forma Linear favorece os acessos para operações em blocos, como por

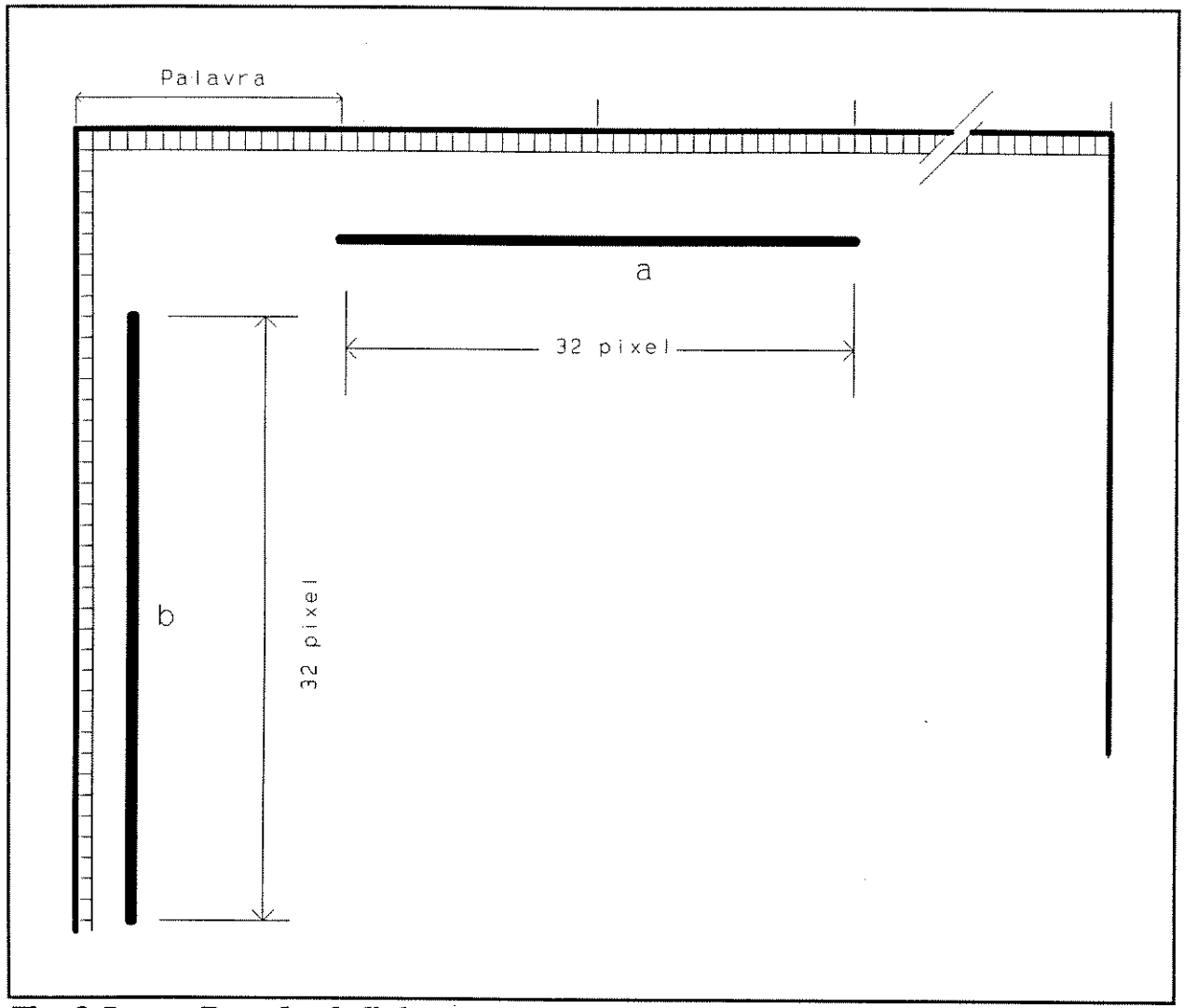


Fig. 3.5 - Desenho de linhas retas por acessos lineares.

exemplo, preenchimento de área, plotagem de caracteres e operações com *bits* em bloco — *bitblt*, também chamadas de *RasterOp*. De acordo com esta forma de acesso uma linha de varredura é dividida em diversas palavras, cada uma contendo um número fixo de pontos. No caso de um circuito monocromático ponto aceso-apagado, este número corresponde à largura do barramento de dados de acesso a memória. Ao traçar uma reta horizontal, começando no ponto correspondente ao início de uma palavra (num dos limites de palavra), o ganho na taxa de transferência é n, onde n é o número de *pixels* atingidos por vez. Entretanto para traçar uma reta horizontal a partir de qualquer ponto interno a palavra (num dos limites de ponto) são necessárias operações de mascaramento. Estas operações são necessárias para qualquer tratamento de um único ponto isolado. No exemplo da figura 3.5, numa tela com a resolução na linha de 256

pontos, as linhas foram divididas em 16 palavras de 16 pontos, pois para traçar a linha "a" com 32 pontos foram gastos dois acessos pelo processador. Para traçar a linha "b" também com 32 pontos foram gastos 32 acessos com mascaramento. Assim, apesar da forma linear de acesso aos pontos ter um potencial de multiplicar por *n* a taxa de transferência, este ganho depende não só de *n* mas também do número médio de linhas horizontais contido no objeto que está sendo desenhado.

A forma Retangular favorece mais as operações em bloco. Por exemplo, para plotar um caractere com as dimensões de 8 x 8 pontos usando a forma linear com uma palavra de 64 pontos serão necessários 8 acessos com mascaramento. Para acessos na forma retangular isto pode ser feito num único acesso. A tela é dividida num quadriculado que corresponde aos limites de início de uma palavra. As operações que são executadas a partir destes limites são aceleradas, entretanto as operações com início em qualquer outro ponto podem requerer até 4 operações de mascaramento.

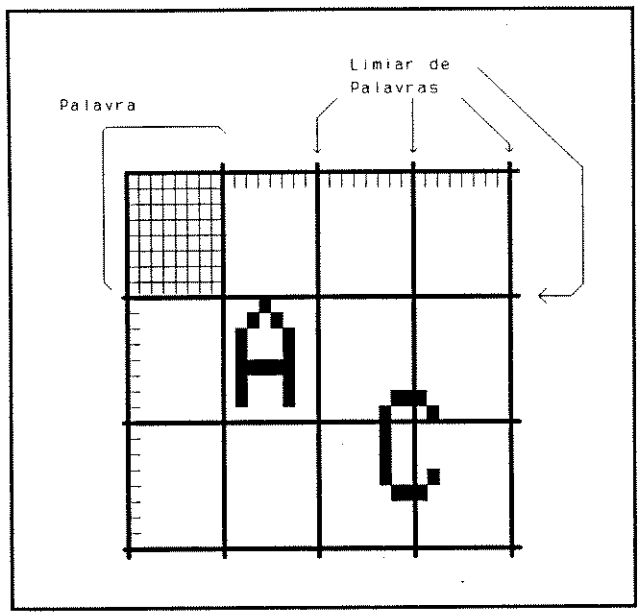


Fig. 3.6 - Desenho de caracteres por acessos retangulares.

Considere, por exemplo, a tela mostrada na figura 3.6. Ela está dividida em retângulos de 8 x 8 correspondentes a uma palavra de acesso pelo processador de 64 *bits*. Para plotar a letra "A" foi necessário um único acesso. Entretanto para plotar a letra "C" foram necessários 4 acessos, por que ela está posicionada num ponto interno a uma palavra.

De uma forma geral os acessos linear e retangular apesar de potencialmente acelerarem a transferência com a memória de quadro, prejudicam operações que necessitam atingir pontos isolados.

Para a alteração de um único pixel, um barramento largo exige que seja feita uma sequência de operações do tipo leitura-modificação-escrita (*read-modify-write*) por *software* ou por *hardware*. Estas operações quando aplicadas para permitir o acesso individual a um *bit* dentro de uma palavra com vários *bits* são chamadas de *bitblt (bit block transfer)* ou de *RasterOp (Raster Operation)*.

Por *software*, antes de um acesso de alteração de um pixel, deverá ser feita uma leitura do conteúdo da palavra que contenha o *bit* correspondente. Nesta palavra é efetuada uma operação de mascaramento que altere somente o *bit* correspondente ao *pixel* e preserve os demais *bits*. E por último é efetuada a escrita da palavra alterada na memória. O quadro abaixo mostra de forma simplificada os trechos de microprogramação do apagamento (a) e do acionamento (b) de um *pixel* na memória de quadro. Nestes trechos de programa, RegA representa um registrador no processador gráfico; MEM representa a posição de memória de quadro onde está o *bit* correspondente ao *pixel* a ser operado e MASC é uma variável com somente um *bit* igual a 1, aquele correspondente ao *pixel* a ser operado.

Microprogramas de apagamento e acionamento de pixel.

```
RegA <-- MEM
RegA <-- RegA and not(MASC)
MEM <-- RegA
```

a) Apagamento de pixel.

```
RegA <-- MEM
RegA <-- RegA or MASC
MEM <-- RegA
```

b) Acionamento de Pixel

Por *hardware*, será necessário utilizar memórias que permitam acessos em ciclo de *Read-Modify-Write* e circuitos de mascaramento de *bits*. Desta forma é possível "esconder" para o processador, um ciclo de leitura e outro de mascaramento em cada acesso de escrita.

A figura 3.7 mostra um esboço de um circuito de acionamento de *pixel* baseado em ciclos de *Read-Modify-Write*. Nesta figura o acesso para o acionamento de *pixel* num barramento de dados de largura - **n** - é iniciado pelo sinal de controle de escrita - **write**, este sinal não chega imediatamente à memória, passa por uma linha de atraso. Como o acesso de leitura só depende do endereçamento, assim que o endereçamento é feito inicia-se a leitura do conteúdo anterior. Este conteúdo é operado por um conjunto de **n** gates **OU**, gerando o dado a ser escrito na posição de memória endereçada. Assim para o programador acionar um pixel, basta escrever na posição de memória uma palavra com todos os *bits* em zero menos o *bit* correspondente ao *pixel* a ser acionado que deverá estar em 1. No exemplo acima foi usada uma única operação lógica (**OU**), entretanto é possível generalizar esta lógica colocando no lugar dos *gates*, circuitos do tipo ALU e programando previamente a lógica a ser utilizada.

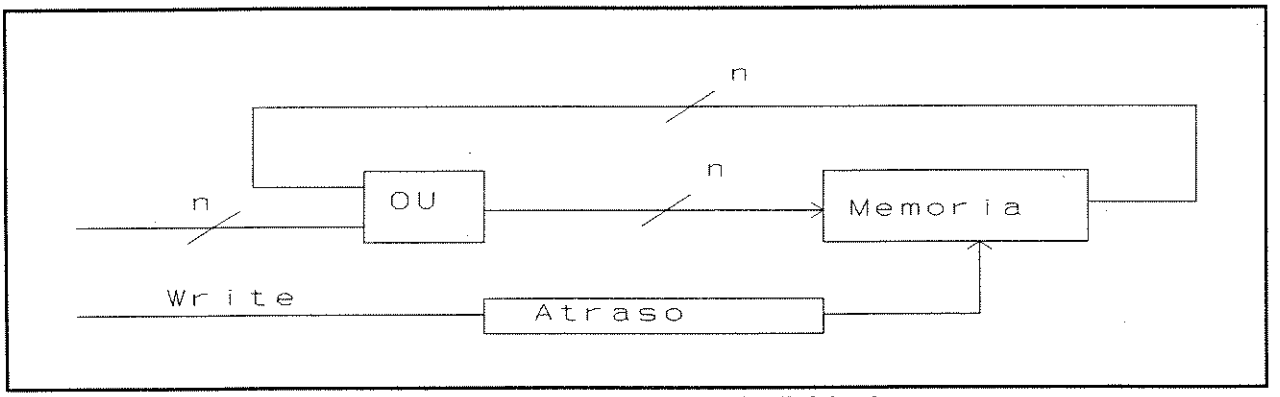


Fig.3.7 - Exemplo do hardware de acesso a pixels individuais

3.6.2 - A Escolha de Chip para a Memória de Quadro.

Na escolha do chip de memória para a memória de quadro deve ser considerado o compromisso **Capacidade de Armazenamento** (Empacotamento) x **Velocidade de Acesso** da memória pelo processador.

Uma alta capacidade de armazenamento por *chip* resolve a questão do empacotamento, ou seja a área no circuito implementado, pois quanto mais densa a memória menor a área no circuito impresso. As memórias mais densas são as memórias dinâmicas. O uso de memória dinâmica leva ao surgimento de uma outra questão - os ciclos de *refresh* do conteúdo da memória dinâmica - que vem a ser mais um concorrente na disputa do acesso à memória de quadro. Estes ciclos podem ficar transparentes se for tomado o cuidado de organizar a memória de forma que a cada ciclo de carga do *shift register* seja feito numa linha (*row*) de memória diferente, de forma a cobrir para *refreshing* toda a memória.

O outro lado deste compromisso é a necessidade de dispor de *chips* de memória diferentes para atingir diversos *pixels* simultaneamente, para aumentar a velocidade de acesso da memória pelo processador. Esta questão pode ser melhor colocada se for considerado o exemplo do projeto de uma memória de quadro monocromática pontos-acesos-apagados com 1024 linhas de 1024 pontos, ou seja uma memória com 1 megapixel. Atualmente existem no mercado diversos *chips* de memória com 1 mega*bits*. Dependendo da organização deste chip ele não pode ser usado para a implementação da memória de quadro. Um único *chip* com a organização 1 mega x 1 *bit* não é indicado, pois seria impossível compartilhar os acessos do controlador de varredura e o processador através de uma única porta de entrada para todas as suas 1 mega posições de memória. Uma solução mais indicada seria utilizar mais de um *chip*

de memória menos densos, formando a mesma capacidade de armazenamento. Desta forma, a cada acesso para a varredura mais de um *pixel* é retirado da memória, permitindo que durante o deslocamento destes *pixels* o processador possa ter acesso a esta memória.

3.6.3 - A determinação do ciclo de acesso do controlador de varredura.

Com já foi visto, é importante que os acessos do processador possam concorrer com os acessos do controlador de varredura. Para isto é necessário estabelecer o ciclo de acesso para o carregamento do *shift Register* de forma a permitir que dentro deste ciclo possa caber um ciclo de acesso pelo processador. Desta forma é definido o t_{cyc} máximo a ser exigido do chip de memória. Para determinar o ciclo de acesso do controlador de varredura - CMR - devem ser considerados os seguintes parâmetros:

FV = frequência de varredura vertical em Hz
NL = número de linhas,
NP = número de pontos na linha,
SR = comprimento do *shift register*,
VA = razão período de quadro aceso/período total do quadro,
HA = razão período de linha acesa/período total da linha.

Relacionados na seguinte fórmula:

$$CMR = \frac{VA \cdot HA \cdot SR}{FV \cdot NL \cdot NP} \cdot 10^6 \mu s \qquad \textbf{(3.2)}$$

Exemplo de Cálculo do CMR

Aplicando na formula (3.2), como exemplo ilustrativo, os dados do monitor TEK GMA 303 e considerando a tela de 1024 linhas de 1280 pontos, com 60hz de varredura vertical tem-se:

FV = 60
NL = 1024
NP = 1280
VA = .967 (pior caso)
HA = .712 (pior caso)

supondo um *shift register* de 16 bits tem-se:

CMR = 140 nseg

Ou seja, o *shift register* deve ser carregado a cada 140nseg.

Supondo ainda que se deseja ter um ciclo de acesso pelo processador para cada acesso do controlador de varredura, deveria ser escolhida uma memória com ciclos de acesso de no máximo 70 nseg. Se for aumentado o comprimento do *shift register* (SR) para 32 bits, é possível escolher memórias com ciclos de acesso de 140 nseg. Ou então, para SR = 64 bits permitindo escolher memórias com ciclos de acesso de 280 nseg. Nesta ultima opção chegou-se a um tempo de acesso bem viável, mas a escolha do componente pode esbarrar no outro problema - a quantidade de componentes de memória (empacotamento), Pois, quanto mais bits em paralelo for necessário, mais *chips* de memória serão necessários.

Neste caso, quantos *chips* serão necessários?

O tamanho da memória de quadro é:

1 280 x 1 024 = 1 310 720 bits

O *shift register* de 64 bits requer 64 ligações a esta memória, e cada uma destas ligações terá:

1 310 720 / 64 = 20 480 bits

Pode-se optar por diversos esquemas de ligação de memória, como por exemplo:

- Memória Dinâmica com 64k x 1 bit. Serão necessários, assim, 64 *chips* de memória.
- Memória DRAM de 16k x 4 bit. Serão necessários, assim, 32 *chips* de memória.
- Memória Estática 8k x 8 bits. Serão necessários, assim 24 *chips* de memória.

A decisão deverá ficar para aquela que implicar numa menor área de circuito e permitir uma ligação mais simples com o processador gráfico.

3.7 - Circuito de Vídeo Bit-Mapped Monocromático com Escala de Cinza.

Neste tipo de circuito cada ponto na tela está associado a mais de um *bit* de memória. O conteúdo da palavra formada por estes *bits* vai determinar o nível do brilho do ponto na tela. Isto é feito passando os *bits* desta palavra por um conversor digital-analógico. Existem dois tipos de circuitos bit-mapped monocromáticos com escala de cinzas, o Analógico e o Digital. A diferença entre eles está na ligação com o monitor. No analógico, o conversor D-A é colocado no próprio circuito, e o sinal de vídeo enviado para o monitor é analógico. No digital, o conversor D-A é colocado no monitor de vídeo, de forma que são enviados para o monitor vários sinais digitais com o código do nível de cinza a ser exibido. Este código é então convertido no monitor.

Para a descrição deste circuito, considere o esquema generalizado de um circuito monocromático com escala de cinza analógico, como mostrado na figura 3.8. As diferenças para o esquema monocromático pontos-acesos-

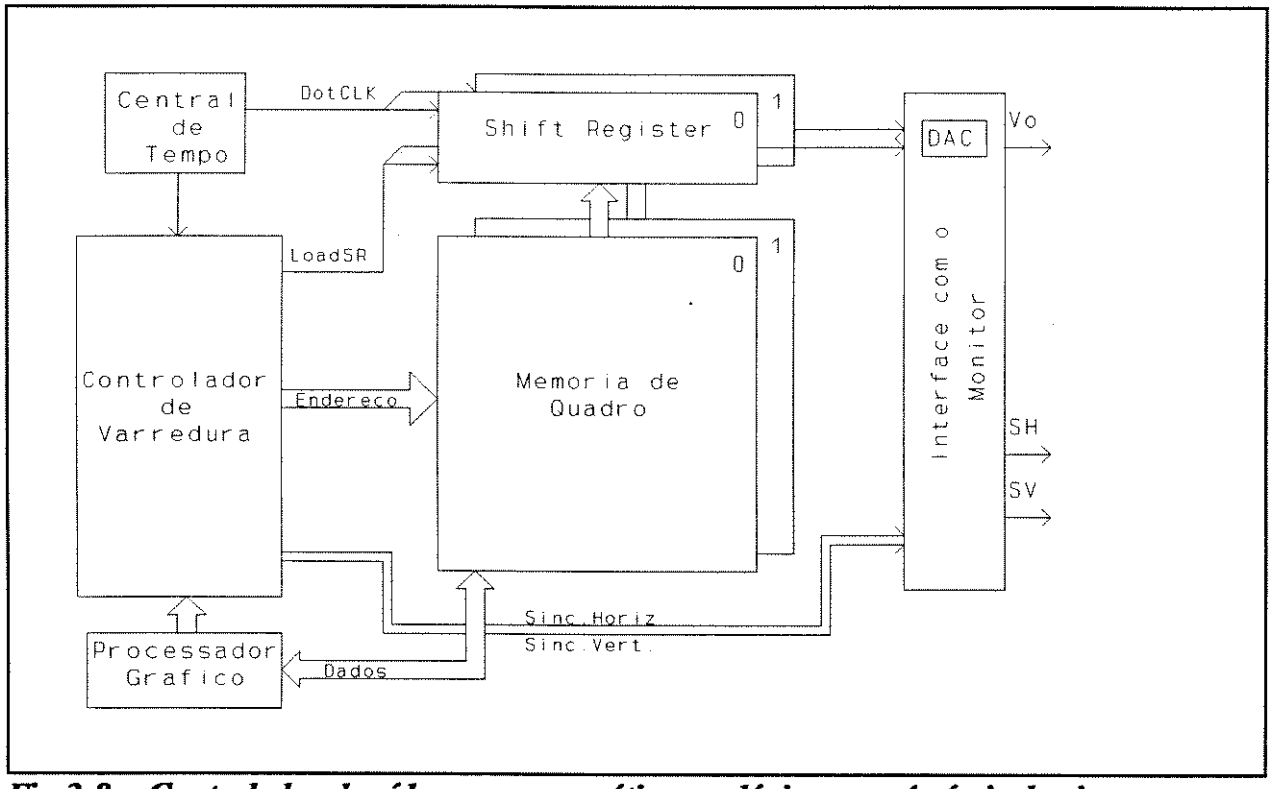


Fig.3.8 - Controlador de vídeo monocromático analógico com 4 níveis de cinza.

apagados são:

- a multiplicação da quantidade de memória de quadro em função do número de níveis possíveis de luminância dos *pixels* na tela. Esta repetição da memória de quadro pode ser visualizada no esquema acima como se fossem estabelecidos planos de memória de quadro;

- que cada plano possui o seu próprio shift register;

- e a inclusão de um conversor digital-analógico na interface com o monitor.

O controle e a temporização são essencialmente os mesmos do circuito monocromático pontos-acesos-apagados.

Os acessos para o *shift register* são feitos sincronizados em todos os planos, de forma que os *bits* de cada plano são enviados em paralelo para a interface do monitor. A palavra formada por estes *bits* é convertida para um nível analógico e enviada ao monitor.

Os problemas de especificação do chip de memória são tratados da mesma forma que antes, com o agravante de que a quantidade dos componentes de memória é também multiplicada.

3.7.1 - O Acesso do Processador Gráfico à Memória de Vídeo.

Num circuito de vídeo *bit-mapped* monocromático analógico os acessos do processador podem ser feitos de duas formas básicas, a saber:

- **Acesso Multiplanar,**
- **Acesso Planar.**

O **Acesso Multiplanar** dá-se quando uma palavra de acesso pelo processador está ligada a um *bit* em cada plano. Considere o exemplo de um processador de 8 *bits* e um esquema de memória de quadro com 8 planos. Para usar esta forma de acesso neste exemplo, a cada *bit* do barramento de dados do processador será ligado um *bit* em cada plano. De forma que em cada acesso será operado um único *pixel*. Esta forma de acesso é bem interessante para operações de plotagem de pontos isolados, pois permite uma associação entre a posição de memória e o *pixel* a ser operado. De acordo com a classificação descrita na seção 3.6.1, os acessos pontuais são acessos multiplanares.

O **Acesso Planar** dá-se quando a palavra de acesso pelo processador está toda ligada a um mesmo plano, como no caso do circuito pontos-acesos-apagados. O acesso a outros planos é feito endereçando outra posição de memória. No acesso planar, os acessos podem ser ainda lineares ou retangulares, de acordo com a classificação descrita na seção 3.6.1. Considere como exemplo um processador de 8 *bits* e uma memória de quadro organizada em 8 planos. A cada acesso serão operados 8 *pixels* num dos seus 8 planos.

Quando o barramento de dados possue mais *bits* que o número de planos, os acessos do processador misturam os acessos multiplanar e planar. De forma que numa mesma palavra têm-se *bits* associados aos planos e diversos *pixel*s na mesma palavra. Como exemplo, suponha um barramento de 8 *bits* tendo acesso a uma memória com dois planos. Neste exemplo, num único *byte* é possível ter acesso a 4 *pixels* como mostrado na figura 3.9. Este é um acesso linear de 4 *pixels*.

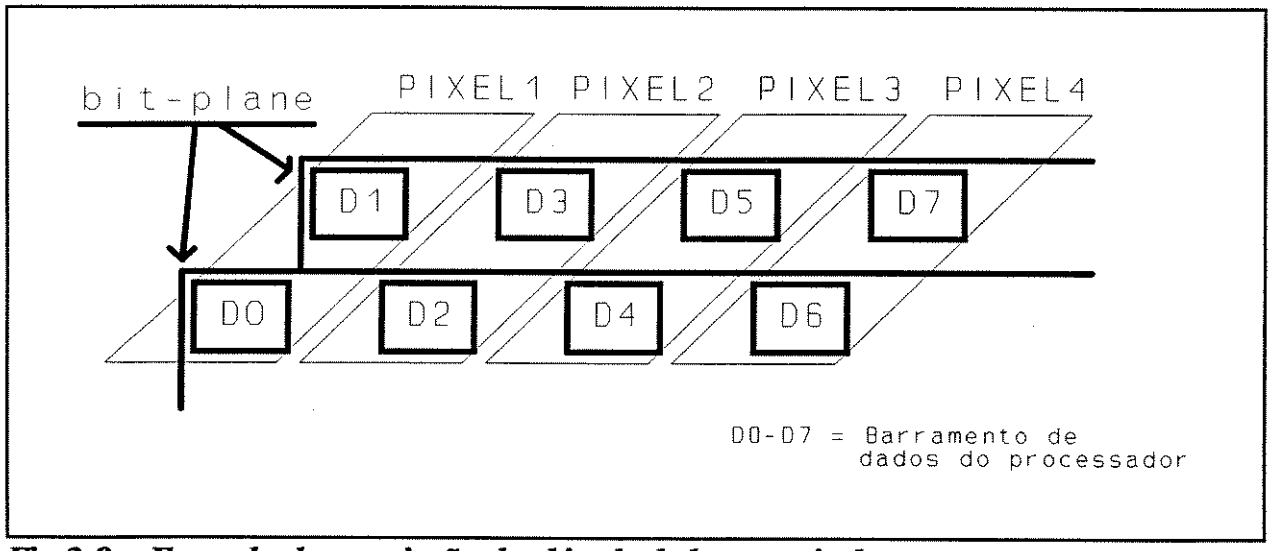


Fig.3.9 - Exemplo de associação dos bits de dados aos pixels

3.8 - Circuito *bit-mapped* Policromático

Os circuitos de vídeo policromáticos podem ser divididos em circuitos digitais e analógicos. Os circuitos digitais normalmente são limitados a exibição de poucas cores, os circuitos analógicos permitem um maior número de cores e são normalmente utilizados em estações de trabalho.

3.8.1 - Circuito de vídeo *bit-mapped* policromático digital (RGB)

Este esquema de circuito de vídeo associa a cada uma das cores primárias, vermelho, verde e azul (RGB) um *shift register* e um plano da memória de quadro. Todo o controle e temporização são essencialmente os mesmos do esquema anterior. Devem ser observados os mesmos problemas de empacotamento e de carga sincronizada do *shift register*. Com o circuito de vídeo *bit-mapped* policromático digital (RGB) é possível definir 8 cores diferentes na tela.

O Acesso do Processador Gráfico à Memória de Quadro.

Essencialmente, esta questão é tratada da mesma forma do esquema anterior, ou seja com acessos planares ou multiplanares. A diferença é que agora os planos estão associados a cores primárias, são **Planos de cores**. A associação

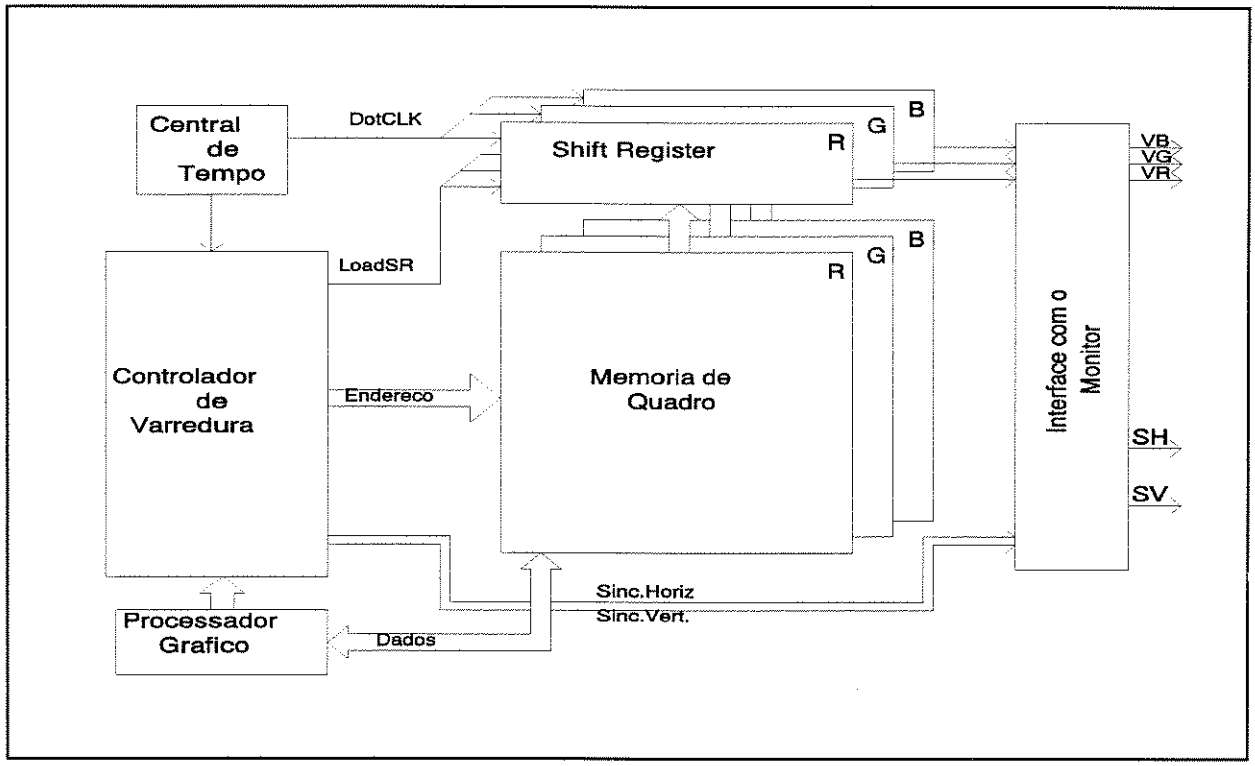


Fig. 3.10 - Controlador de vídeo RGB digital

das cores primárias aos *bits* do barramento de dados é feita diretamente. Como a largura dos barramentos de dados dos computadores é em geral um múltiplo de 8 *bits*, é difícil evitar o disperdício de *bits* para a representação de cores. É mais fácil implementar, e é mais fácil descrever para o programador uma associação que despreza alguns *bits* na palavra. Como exemplo, considere um barramento de oito *bits* que tem acesso a uma memória de quadro organizada com dois *pixels* por *byte*. Neste caso há um desperdício de 2 *bits* por *byte*.

	R_1	G_1	B_1		R_2	G_2	B_2

Circuito de Vídeo Bit-Mapped RGBI

Trata-se de um esquema parecido com o anterior, onde foi adicionado mais um plano. Os *bits* deste plano são enviados juntamente com os *bits* de cores primárias ao monitor e estabelecem uma condição de intensificação de nível de cores. Com isto o número de cores disponível sobe para 16. A associação dos planos de cores aos *bits* na palavra é direta, e aproveita os *bits* que eram desprezados no esquema RGB. Esta associação é compatível com a placa CGA para microcomputadores IBM-PC.

I_1	R_1	G_1	B_1	I_2	R_2	G_2	B_2

Circuito de Vídeo Bit-Mapped RGB DIGITAL Multinível.

Trata-se de um esquema onde para cada cor primária foi estabelecido mais de um plano de *bits*. Estes planos formam então os planos de cores. O número de planos de *bits* define a quantidade de níveis de cada cor primária. Como exemplo desta organização, considere que para cor primária sejam definido 2 planos de *bits*. Com isto o número de cores disponível sobe para 64. A associação dos planos de cores aos *bits* na palavra pode ser feita como mostrado abaixo. Esta associação é compatível com a placa EGA para microcomputadores IBM-PC.

		r	g	b	R	G	B

onde: rR = dois *bits* que definem o nível de vermelho;
gG = dois *bits* que definem o nível de verde; e
bB = dois *bits* que definem o nível de azul.

3.8.2 - Circuito de Vídeo Bit-Mapped Policromático Analógico

Este circuito pode ser visualizado como uma extensão do circuito monocromático analógico, onde para cada cor primária são agrupados diversos planos. Os *bits* de um conjunto de planos passam por um conversor digital analógico que estará ligado a uma saída de cor primária, como mostrado na figura 3.11.

Este esquema exige um grande espaço de memória de quadro. As associações mais comuns de planos às cores primárias são:

- 4 planos por componente primário de cor, que dá uma definição de 4096 cores possíveis.

- 8 planos por componente primário de cor, que dá uma definição de 16777216 cores possíveis.

Apesar destes números parecerem altos, é importante destacar que nas aplicações que exigem alta definição de cores deve ser considerada, também, a quantidade de níveis de cinza disponível. Isto pode ser determinado pelo número de níveis que existirão com as componentes de cores primárias com a mesma intensidade. Se este número for baixo, isto implicará num *aliasing* pronunciado de cores. No caso de 4 planos por cor primária existirão no

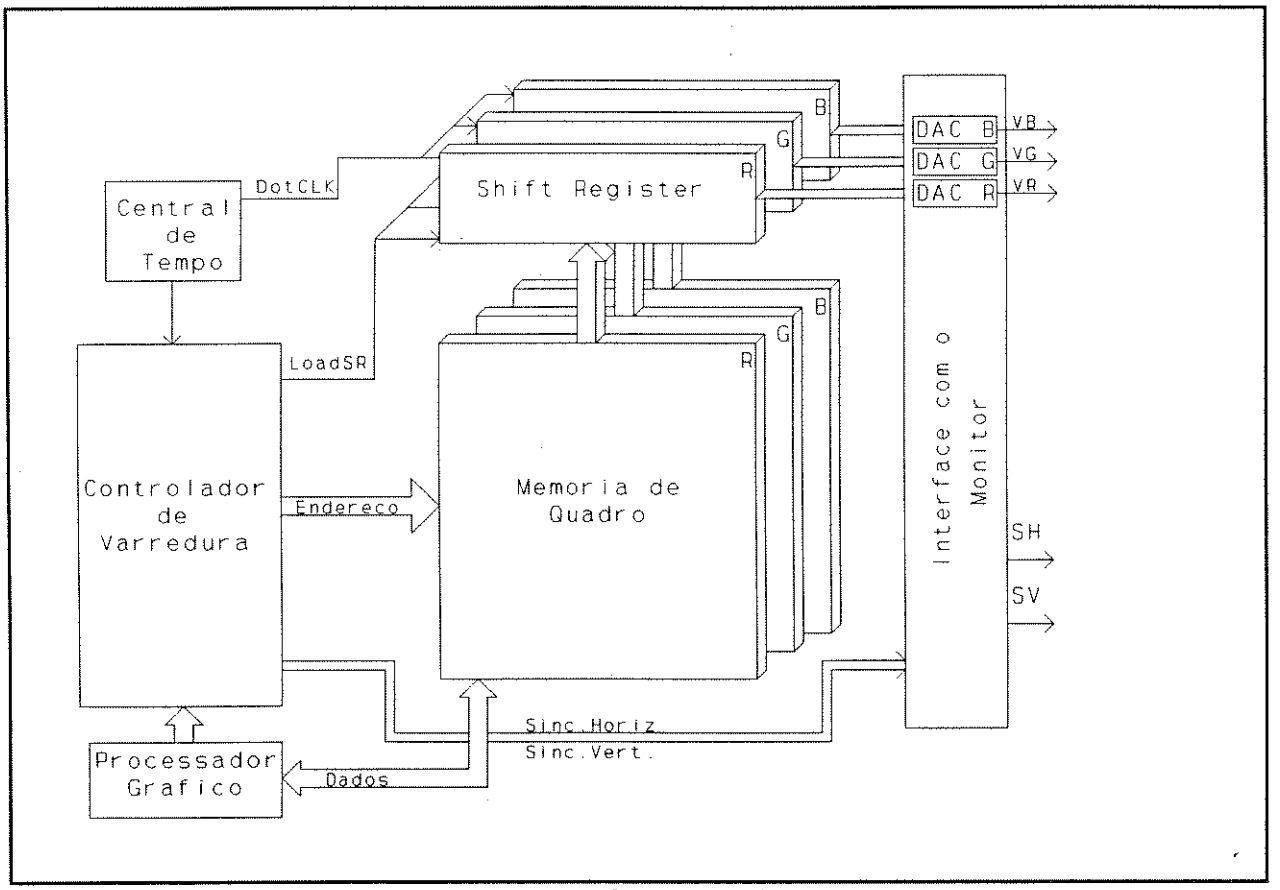


Fig.3.11 - Controlador de vídeo Policromático Analógico

máximo 16 níveis de cinza. Isto é muito pouco para aplicações que envolvem sombreamento. No caso de 8 planos por cor primária existirão no máximo 256 níveis de cinza. Isto é bem aceitável. Esta tem sido a definição utilizada nos cartões dedicados à aplicação de Geração de Imagens com Realismo. Este tipo de circuito recebe o nome de circuito de exibição de imagem com cor plena (*true color*).

3.8.3 - Circuito Bit-Mapped com Palheta

Na maioria das aplicações, entrètanto, o uso de tanta memória para exibir tantas cores não se justifica. O circuito de vídeo policromático com palheta mantém uma quantidade pequena de memória e permite um grande conjunto de opções de cores. Este tipo de circuito pode ser descrito com auxílio da figura 3.12, que mostra o detalhe da interface dos sinais de vídeo. A memória de vídeo conserva **códigos de cores** com *n bits* que são serializados como nos circuitos anteriores. Na saída do *Shift Register,* estes códigos endereçam uma memória de alta velocidade, chamada de ***Look-up Table***. Em cada posição

desta memória têm-se valores com *m bits* para cada uma das cores primárias. Estes *m bits* são enviados para conversores Digital-Analógicos ligados às saídas de cores primárias para o monitor de vídeo. Os valores para cada uma das cores primárias são previamente carregados em cada uma das 2^n posições da *look-up table*, sendo possíveis 2^{3m} opções de cor. Desta forma num dado instante se trabalha com 2^n cores escolhidas de um conjunto de 2^{3m} cores possíveis. O nome deste circuito leva em consideração a semelhança de funções dele com uma palheta de pintura que permite ao pintor misturar e escolher as cores das tintas mais adequadas para o seu trabalho. Um exemplo muito comum de definição de um circuito de palheta é aquela que permite o trabalho com 256 cores simultâneas escolhidas em 16 milhões de cores possíveis. Neste exemplo, os códigos de cores têm 8 *bits*, ou seja a memória tem 8 planos e a *look-up table* tem 8 *bits* para cada cor primária.

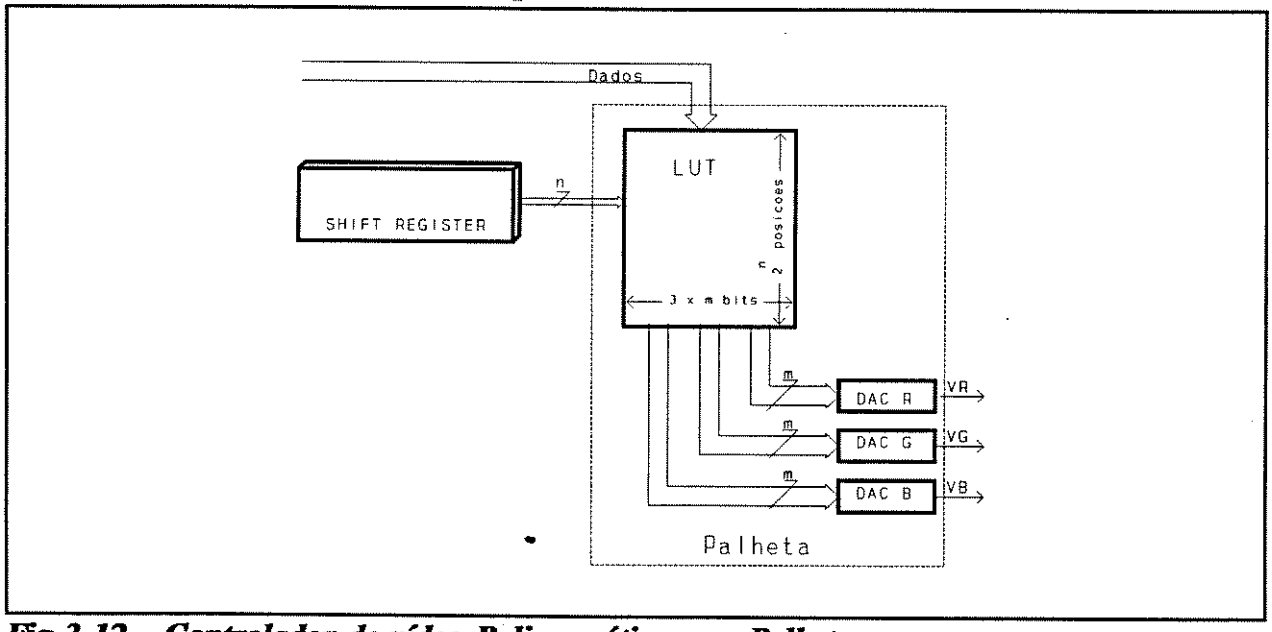


Fig.3.12 - Controlador de vídeo Policromático com Palheta

3.9 - Circuito Exibidor de Caracteres

O circuito exibidor de caracteres é baseado na definição de códigos para os caracteres alfanuméricos. Um esquema bem simplificado deste tipo de circuito é mostrado na figura 3.13. A memória de quadro armazena os códigos relativos ao texto a ser exibido juntamente com códigos de atributos de exibição dos caracteres. Dentre estes atributos podem ser destacados os que definem a cor de fundo do texto (*Background*), a cor do caractere a ser exibido (*Foreground*) e o caractere piscando (*Blinking*). O código do caractere e mais o número da linha que está sendo varrida são enviados para um circuito Gerador de Caracteres. Este circuito é essencialmente uma memória não volátil que armazena linha

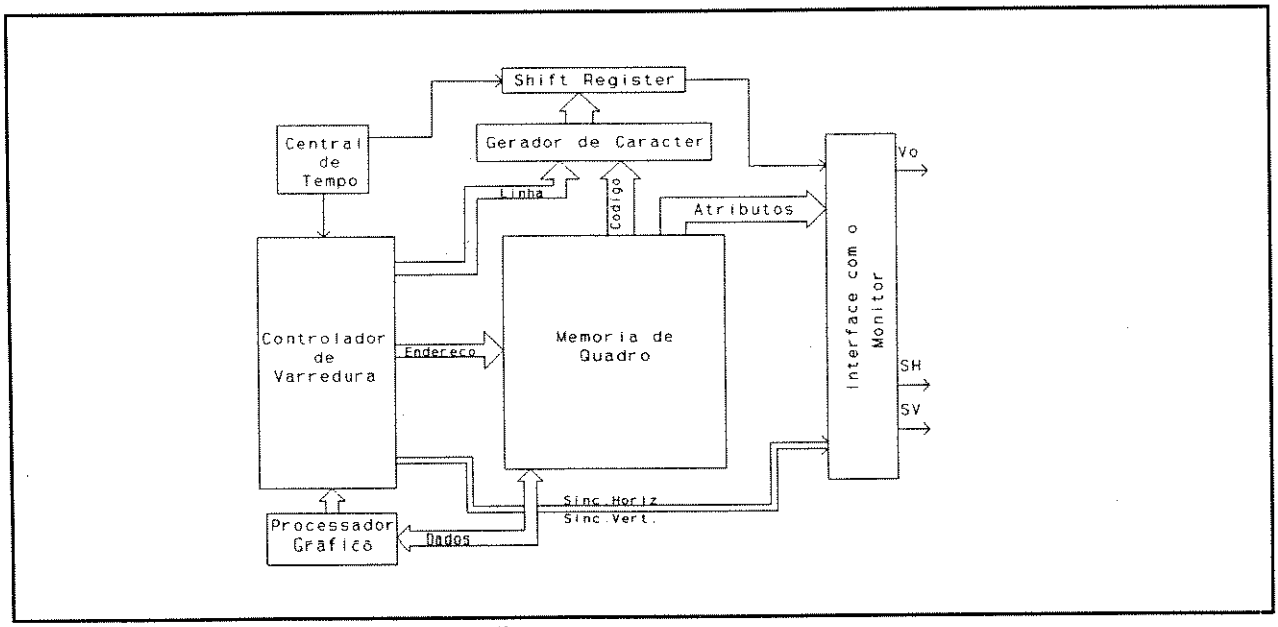


Fig. 3.13 - Circuito Exibidor de Caracteres

a linha o formato do caractere, como no exemplo da figura 3.14, onde é mostrado um trecho desta memória onde está armazenado o formato do caractere "A". Este circuito envia então para um circuito de *Shift register* os *bits* que compõem o formato do caractere numa dada linha.

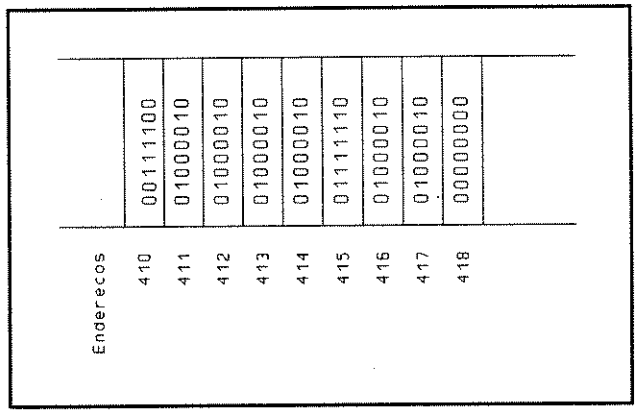


Fig. 3.14 - Conteúdo da memória de Gerador de Caractere, com o formato da letra "A".

O código de atributo é enviado diretamente para a interface com o monitor que o decodifica e o sincroniza com os *bits* deslocados a partir do código do caractere. Os *bits* correspondentes ao formato do caractere produzem uma saída na cor *Foreground* para o monitor, os *bits* correspondentes ao fundo do caractere produzem uma saída na cor *background* para o monitor.

Este tipo de circuito é usado, por exemplo, em terminais alfanúmericos, em microcomputadores e em jogos eletrônicos. Exemplos de projeto de circuito de vídeo deste tipo são [Oliv 83] e [Brag 85], nestas referências são feitas descrições com maiores detalhes deste tipo de circuito.

3.10 - Memórias de Vídeo - VRAM

A partir de 1985 o projeto de circuitos controladores de vídeo ficou mais fácil com o lançamento no mercado das chamadas Memórias Dinâmicas de Dupla

Porta Específicas para Circuitos de Vídeo, ou simplesmente Video-RAM (VRAM).

Estas memórias reúnem num único chip o bloco de memória e o registrador de deslocamento. Um diagrama da organização típica de uma memória VRAM é mostrado na figura 3.15.

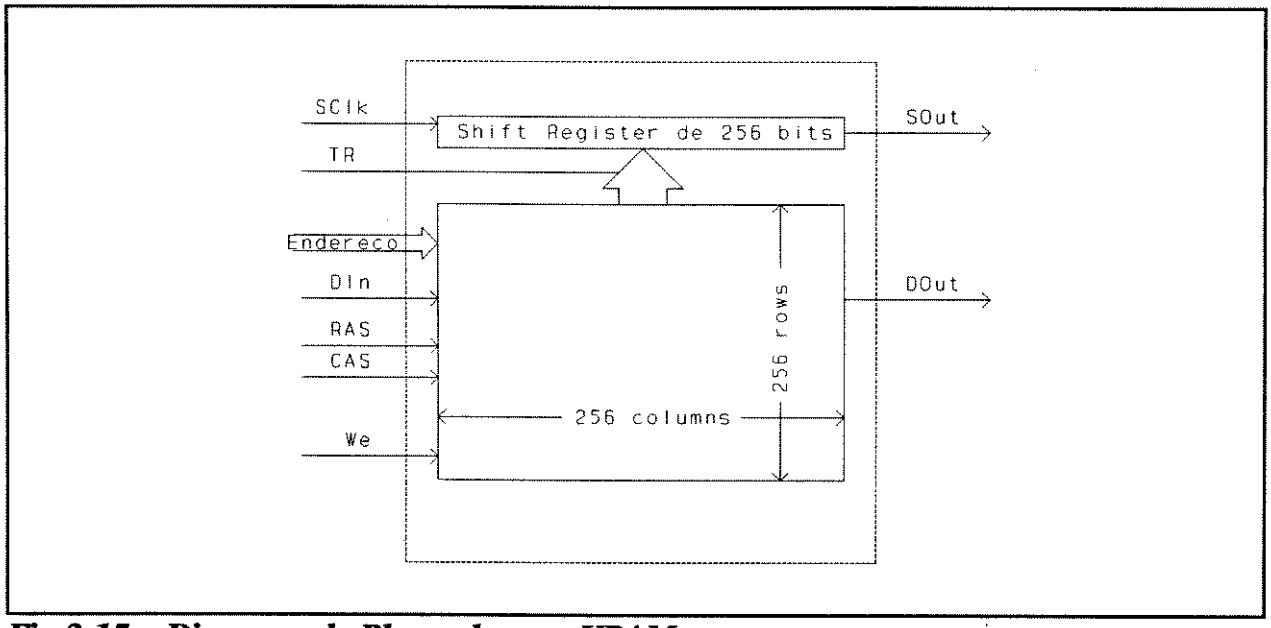


Fig.3.15 - Diagrama de Blocos de uma VRAM

A memória tem duas portas, uma para a ligação com computador (porta paralela) e outra para a ligação com o circuito de saída de vídeo (porta serial). A porta paralela possue dois sinais de dados, **Din** (entrada de dados) e **Dout** (saída de dados), permitindo o acesso como numa memória dinâmica normal, ou seja através dos sinais **RAS** (*Row Address Strobe*), **CAS** (*Column Address Strobe*), **WE** (*Write Enable*) e *bits* de endereçamento - **A0...An** - . A porta serial, sinal **Sout** (*Serial Output*), corresponde à saída de um registrador de deslocamento (*shift register*) com a largura de uma linha (*row*) da memória dinâmica. O deslocamento neste registrador é feito com base num sinal de relógio **SClk** (*Serial Clock*). O carregamento do registrador de deslocamento é feito através do sinal de controle **TR** (*TRansfer*). Este sinal transfere o contéudo das células da linha (*row*), endereçada pelos *bits* A0...An, para o registrador de deslocamento.

Além de facilitar o projeto de circuitos controladores de vídeo, este tipo de memória apresenta, o que pode ser considerada como a sua característica mais atrativa, uma redução substancial da concorrência entre os acessos de renovação dos dados na tela (*screen refreshing*) e os acessos pelo processador gráfico. O único instante de competição pelo acesso se dá durante o carregamento do registrador de deslocamento. Uma vez carregado, o acesso de

refreshing se dá pelo pulsar do SClk e toda a memória pode ficar a disposição do processador. O uso deste tipo de memória permite uma disponibilidade para acesso pelo processador da ordem de 94% do tempo. As estruturas que utilizam memórias dinâmicas comuns conseguem uma disponibilidade típica da ordem de 50%.

3.10.1 - Exemplo de Aplicação de VRAM

Um exemplo de aplicação em projeto deste tipo de memória é mostrado a seguir. A figura 3.16 mostra o esquemático simplificado de um circuito de memória de vídeo monocromática com a resolução de 512 x 512 *pixels*.

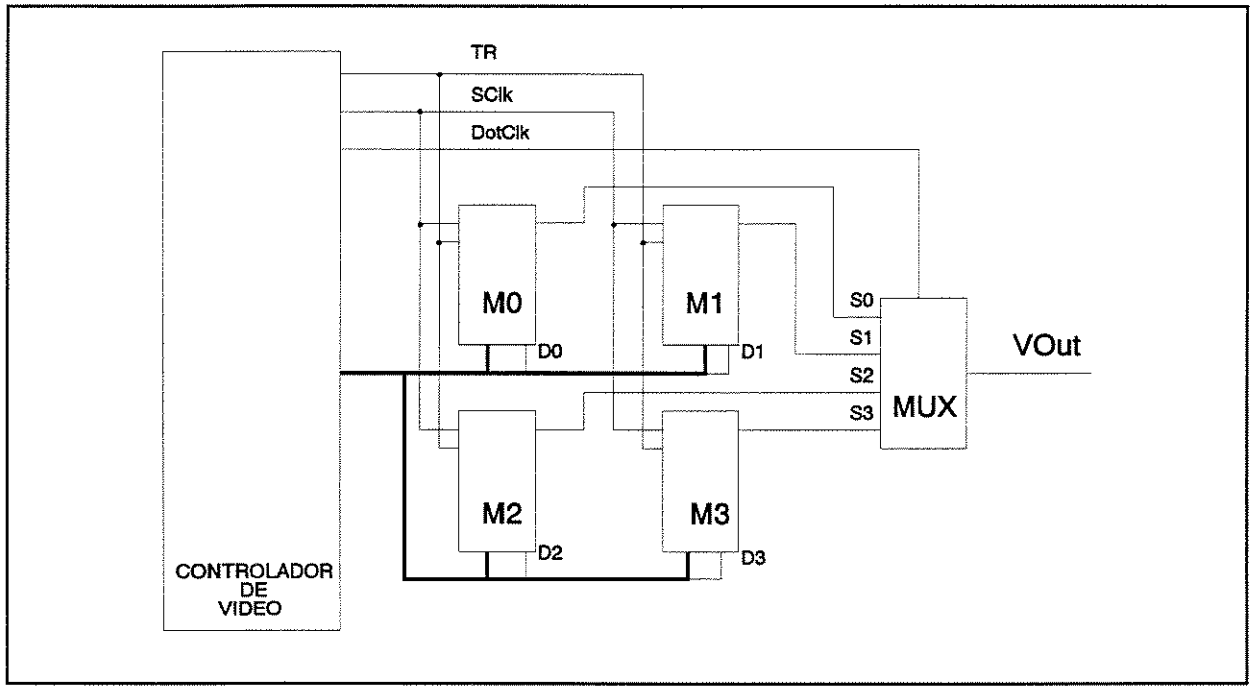


Fig.3.16 - Circuito de vídeo monocromático baseado em VRAM

Neste esquemático é utilizada uma memória VRAM com 64 kilo*bits* organizados em 256 filas (*rows*) de 256 colunas. A interface do chip de memória é feita com 1 *bit* de dado para ligação com o processador e uma saída serial para a ligação com a saída de vídeo. Para a resolução pretendida são necessários 4 chips de memória. O esquemático da figura 3.16 mostra, além dos chips de memória, um controlador de vídeo e um circuito multiplexador. Destacam-se ainda os sinais Vout, DotClk e SClk.

O sinal DotClk corresponde a frequência do sinal de saída de vídeo - Vout. O sinal SClk, que é igual a 1/4 da frequência do sinal DotClk, é ligado a todos os chips de memória. A cada pulso de SClk, cada chip de memória desloca um *pixel* para o circuito multiplexador.

O circuito multiplexador, com base no sinal DotClk, seleciona ordenadamente um único *pixel* para enviar para a tela pelo sinal Vout como mostrado na figura 3.17.

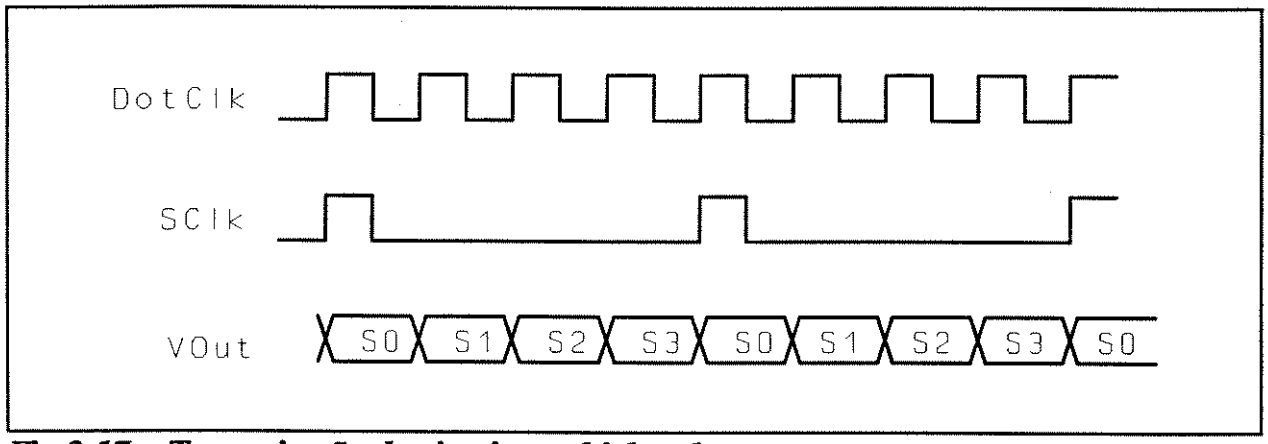


Fig.3.17 - Temporização do circuito multiplexador.

O circuito controlador fornece a temporização dos sinais SClk e TR, comum a todos os chips de memória e gerencia os acessos à memória pelo processador gráfico.

Neste circuito a carga simultânea destes registradores é feita a cada 2 linhas de varredura. Supondo que cada linha dure 16 μseg, e que o tempo de carga do registrador seja 500 nseg tem-se

$$\frac{500}{2 \cdot 16000} \cdot 100 = 1{,}5625\% \textit{ do tempo ocupado}$$

A este tempo de ocupação pode ser acrescentado o tempo de renovação dos conteúdos das células de memória dinâmica (*refreshing*) que exige um acesso a cada 16 μseg. Supondo uma duração de 500 nseg. para este acesso, tem-se

$$\frac{500}{16000} \cdot 100 = 3.125\% \textit{ do tempo ocupado}$$

O tempo livre para acessos pelo processador gráfico é, então, de aproximadamente 95%.

3.11 - O tratamento de Janelas

O tratamento de janelas é importante para uma boa interação com o usuário. Através de janelas é possível ter na mesma tela diversos processos, que rodam simultaneamente numa mesma máquina. É possível, por exemplo num sistema de geração de imagem, ter numa janela um editor de texto, atráves do qual é modificado o conteúdo de uma tabela de entrada de imagem e ter exibido o resultado da geração numa outra janela. Este tipo de recurso surge como uma

substituição à necessidade de se ter dois monitores de vídeo sobre a mesma mesa.

Os sistemas de *software* que gerenciam janelas estão se tornando mandatórios. Pode-se dizer que o surgimento deste recurso definiu uma nova forma de operação de sistemas de computadores. Com um bom sistema de gerência de janelas é possível ter sobre um mesmo meio (monitor de vídeo) muito mais de uma ou duas telas. Nestes sistemas, o usuário pode escolher entre dividir a tela do monitor entre diversas janelas e exibí-las simultaneamente, ou sobrepô-las colocando à frente aquela correspondente ao processo que estiver sendo operado. A troca de janela à frente é feita por um simples comando do usuário. A disponibilidade de recursos para o tratamento de janelas tornou-se uma imposição de diversos aplicativos, principalmente na área de CAD.

Uma comparação dos diversos sistemas de gerência de janela realizados por *software* é feita em [Ster 87]. Uma descrição específica sobre o sistema *X Window* é feita em [ScGe 86].

O principal problema com um sistema de gerência de janelas é fazer com que as manipulações (inclusão, sobreposição, apagamento, movimento e alteração) sejam feitas num tempo confortável. Devem durar um intervalo de tempo equivalente, por exemplo, ao folhear de um livro, para que a operação seja a mais natural possível. Para que estas operações sejam rápidas é necessário estabelecer algum artifício em *hardware* de tratamento de janelas.

3.11.1 - A Gerência de Janelas por *Hardware*.

Os princípios e os problemas da gerência de janelas podem ser introduzidos considerando uma saída gráfica *bit-mapped* policromática com 8 *bits* por *pixel* e uma resolução de 1024x1024 *pixels*, correspondendo a uma memória de vídeo de 1 mega*bytes*. Neste exemplo, o endereço inicial de memória é 00000h e corresponde ao primeiro *pixel* da linha inferior. Tem-se acesso à memória em palavras de 8 *bits*, cada linha corresponderá a um endereço *byte* múltiplo de 1024, de forma que o endereçamento de cada *pixel* seja conseguido simplesmente pela operação

$$(1024 \cdot Y) + X \qquad (3.5)$$

Sobre este exemplo, considere que a memória de vídeo está inicialmente toda preenchida com o conteúdo correspondente a uma imagem qualquer. A **inclusão** de um janela sobre esta imagem corresponde a escrever, por cima de

uma determinada região de memória, o conteúdo relativo à imagem do novo janela. Para que na eventual retirada desta nova janela, a imagem anterior seja totalmente recuperada, é necessário que antes da escrita, o conteúdo atual seja conservado numa outra área de memória.

Pode-se resumir as operações de inclusão e retirada de janela pelas microoperações descritas no quadro a seguir.

Microprogramas de Inclusão e Retirada de Janelas

```
INCLUSAO
    M(x,y) -> S(i)
    W(x,y) -> M(x,y)

RETIRADA
    S(i) -> M(x,y)

onde: M(x,y) é a área de memória correspondente ao window.
      S(i) é um vetor de rascunho onde serão armazenados
      temporáriamente os pixels.
      W(x,y) é o conjunto de pixels do novo window.
```

Se este janela tiver a dimensão de 512x512, a operação de inclusão necessitará de 512K microperações de transferência. Se cada transferência durar 500 nS têm-se um total de 256 mS para incluir uma janela de 1/4 da tela. Este valor não considera o conflito pelo acesso com o *refresh* da tela e nem o *overhead* do programa de manipulação.

Estas operações quando executadas por processadores convencionais, como por exemplo 680xx e 80x86, podem consumir mais que um segundo. A solução trivial é utilizar recursos de acesso direto a memória (DMA) para realizar estas operações de transferência.

Outro problema, a ser considerado na gerência de janelas, é relativo ao tratamento das bordas da janela. Se no exemplo acima os acessos à memória de vídeo fossem feitos com palavras de 32 *bits*, haveria a necessidade de tratar especialmente as bordas de janelas que não coincidissem com os limites de endereçamento *Long-word*. Este tratamento é feito por operações de mascaramento de *pixels*, em sequências de *Read-Modify-Write*. A solução adotada é executar o tratamento de bordas por programação e o tratamento do "miolo" por DMA. Entretanto este procedimento penaliza o tratamento de janelas muito estreitas, que podem acabar sendo tratadas mais lentamente que aquelas mais largas.

De qualquer maneira, estas formas de gerência acabam sempre ocupando o barramento de dados para a transferência de *pixels*, o que pode ser muito crítico se não houver recursos de DMA com ciclos escondidos (*cycle stealing*). As soluções mais recentes para a gerência por *hardware* evitam ciclos de transferências de *pixel* no barramento de dados do processador. Nas seções a seguir são descritas duas soluções diferentes para esta gerência.

A referência [CSLe 86] descreve um sistema de gerência de janelas baseado na divisão da tela em faixas de mosaicos (*tiles*). Já [RhWi 89] apresenta uma visão de diversos esquemas de gerência de janelas por *hardware*.

3.11.2 - Gerência de Janelas do i80786

O processador de exibição do i80786 tem integrado um sistema de gerência de janelas baseado em Tabela de Descritores de Faixas de Endereçamento (*Strips Descriptors*). Uma faixa é definida como sendo um conjunto de linhas de varredura. Cada faixa pode ser composta por diversos mosaicos (*tiles*). Estes mosaicos correspondem às áreas de memória correspondentes ao *bit-map* de diferentes janelas. Um Descritor de Faixa indica, por exemplo, o número de telhas que compõem as linhas de varredura a serem exibidas. De forma que no início de varredura de uma linha é possível saber quais são os endereços de memória que serão usados para buscar os *pixels*.

Este esquema de gerência é bem eficiente, pois a inclusão e retirada de janela é feita alterando simplesmente a tabela de descritores de faixa, sem nenhuma operação de transferêmcia.

A maior dificuldade deste tipo de gerência é que ele necessita utilizar memórias normais, pois é necessário endereçar individualmente cada posição de memória para permitir o início de janela em qualquer *pixel* da tela. O uso de memórias normais traz todos os problemas de concorrência pelo uso do barramento com o processador gráfico, já discutidos na seção 3.10.

Outro problema, mais relacionado com a implementação do processador de exibição do i80786, é a resolução máxima que ele pode tratar, 640*pixels* em 480linhas.

A figura 3.18 ilustra a gerência de janelas pelo i82786. Maiores detalhes sobre este componente pode ser encontrado na referência [Inte 86].

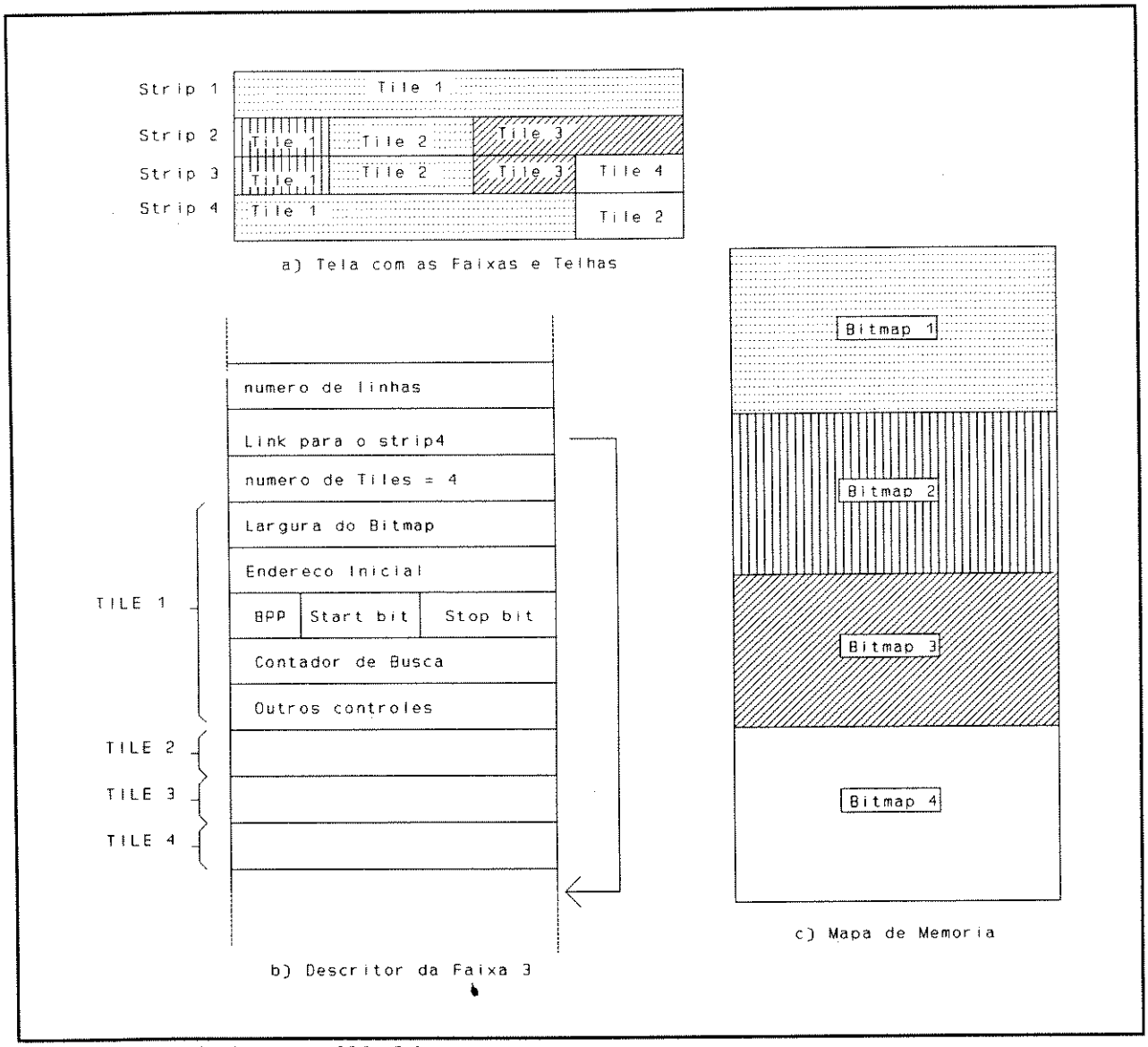


Fig. 3.18 - Windows no i82786

3.11.3 - Gerência de janelas da Família Am95c60

A família de componentes para exibição de *pixels* lançada no mercado pela AMD, chamada de Am95c60 - *Quad pixel* Data Manager - QPDM apresenta uma solução interessante para a gerência de janelas em memórias de vídeo VRAM.

O elemento fundamental para esta gerência é o componente AM8172 - *Video FIFO register*. Este registrador FIFO é colocado no lugar do *Shift Register* do modelo policromático discutido na seção 3.8.2. Ele é ligado à saída serial das memórias VRAM. Essencialmente o que ele faz é operar as transferências da memória de *pixels* VRAM, numa velocidade muito maior que a realmente

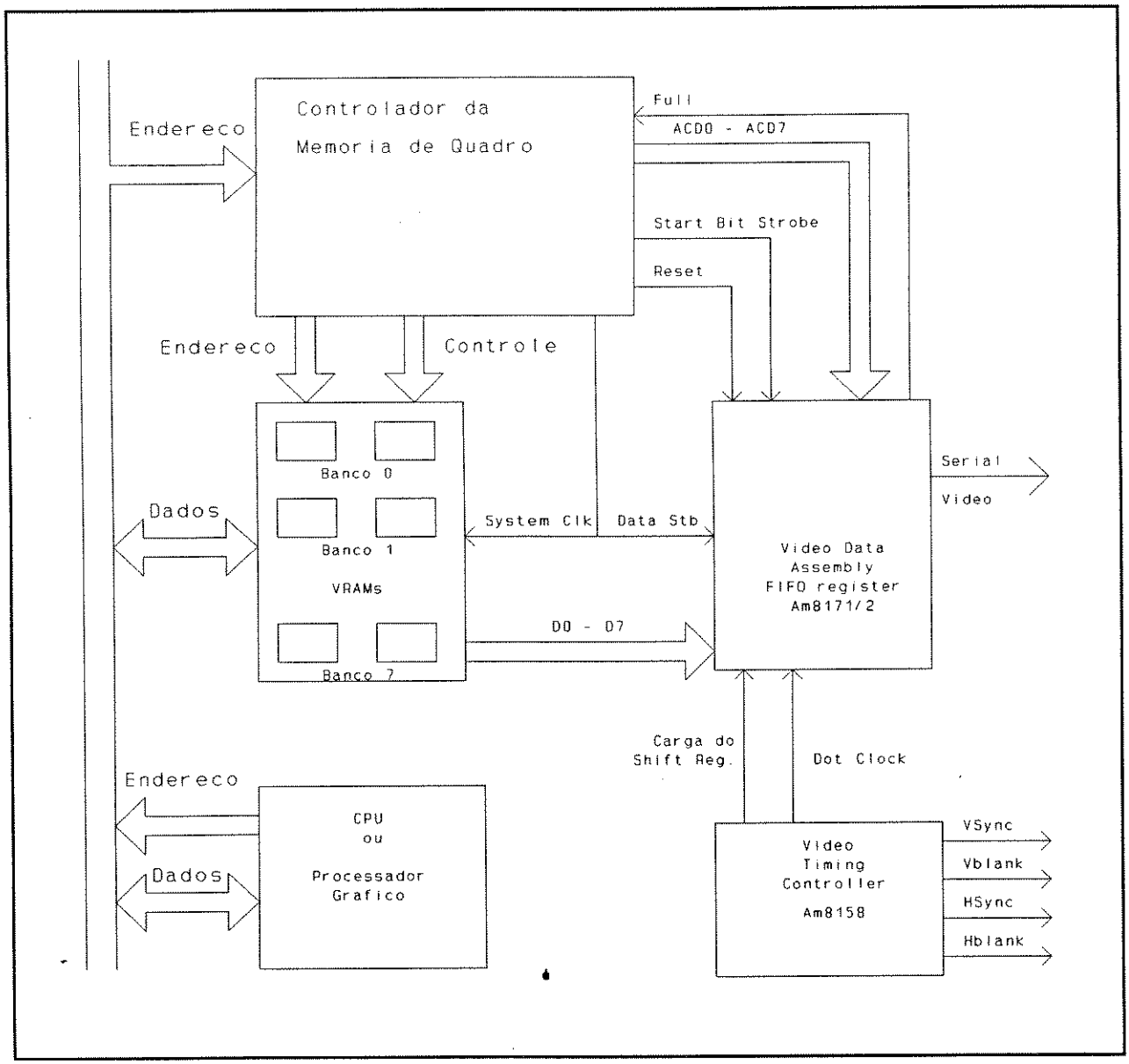


Fig. 3.19 - Exemplo de utilização do QPDM

necessária para a exibição. Desta forma é possível realizar operações, internas ao registrador, de comparação de índice de *pixel* para determinar o início da exibição. Dependendo destas comparações, podem ser necessárias novas transferências da memória. A FIFO deste registrador tem um tamanho de 57 *pixels* válidos para a exibição. Dentro deste registrador existe ainda uma ALU para operações de mascaramento para o tratamento de bordas. Um exemplo de projeto de aplicação do QPDM é mostrado na figura 3.19. Maiores detalhes sobre esta família de componentes pode ser encontrada na referência [PiGa 87].

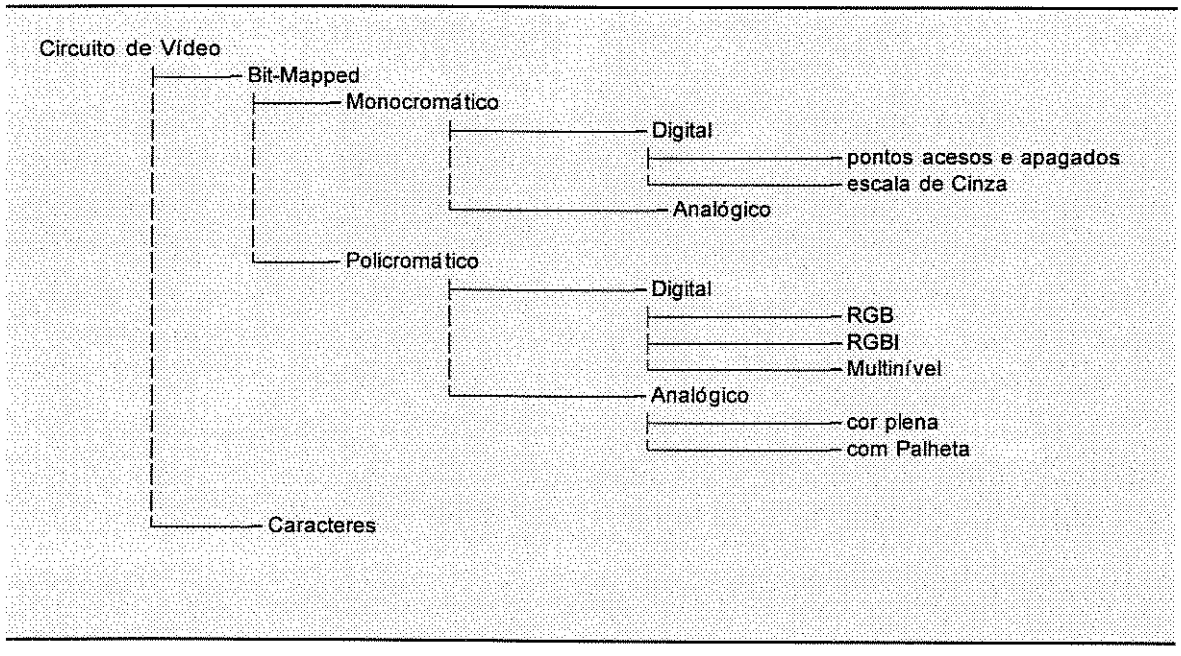
3.12 - Resumo e Comentários.

Este capítulo reuniu um conjunto de conceitos e de definições de termos técnicos pertinentes ao projeto de uma unidade de saída gráfica. Procurou-se, aqui, evidenciar o significado dos termos com o objetivo de familiarizar e padronizar o seu uso no desenvolvimento de um projeto de circuito de exibição gráfica a ser feito no capítulo 5.

Inicialmente foram descritos termos e padrões internacionais de ligação com monitores de vídeo. Depois foram descritas técnicas de controle de vídeo com destaque para o impacto do surgimento de memórias VRAM e para a questão de gerência de janelas por *hardware*.

As organizações de memória de quadro descritas neste capítulo são genéricas e como tal, muito simplificadas. Na prática é comum ter circuitos de controle com a sua configuração programável, de forma que um mesmo circuito possa ser classificado em vários dos esquemas analisados. Esta configuração é feita no estágio do circuito de *shift register*, que no caso não é simplesmente um registrador de deslocamento e sim um circuito com multiplexadores, registradores e outros circuitos elementares que permitem selecionar os planos ou os *bits* a serem enviados para a interface com o monitor. Um resumo dos tipos de circuitos de exibição de imagem é mostrado no quadro abaixo.

Tipos de circuitos de exibição de imagem



Para um estudo complementar sobre a organização de memória de quadros, cita-se, a seguir, três importantes referências:

> [Whit 84] analisa os diversos aspectos de projetos de circuitos de memória de quadros. Desde aspectos relativos ao monitor de vídeo até esquemas de acesso à memória de quadro. Este artigo é contemporâneo ao lançamento das memórias VRAM. A sua análise sobre os projetos com este tipo de memória é portanto muito superficial.
>
> [WEDe 88] considera os principais compromissos de projeto de memórias de quadro em alta resolução e alto desempenho.
>
> [GGSS 89] classifica dez tipos de esquemas de memória de quadro. Nesta referência cada esquema é analisado com profundidade, inclusive as memórias de quadro de caracteres classificadas como sendo memória de quadro orientada a objeto.

Além destas referências, cabe destacar a tese de mestrado de P.C.Berardi [Bera 89] que faz uma interessante descrição de esquemas de memória de quadro e descreve um projeto completo de um circuito de saída de vídeo baseado em VRAM.

Com respeito ao panorama atual da pesquisa, desenvolvimento e do mercado dos circuitos de memória de quadro algumas observações podem ser adicionadas.

A primeira é a tendência de integrar cada vez mais parte do processamento gráfico ao circuito de memória. As tarefas da geração de imagens que mais se aproveitam desta integração são:

- a eliminação de *alias* na composição de imagens. Diversos circuitos incorporam o chamado plano *alpha* introduzido nos trabalhos de Porter e Duff [PoDu 84] e [Duff 85], e

- a remoção de superfícies escondidas por *Z-buffer* [Akel 89].

Além disso, diversos esforços têm sido feitos na divisão dos algoritmos de geração de imagem por diversos processadores elementares integrados no próprio circuito de memória. Exemplos destes esforços são [SDDa 89] e o Pixel-Flow Enhanced Memory [PEMF 92].

Do ponto de vista mais comercial, destaca-se o mercado de memórias VRAM. Observa-se o fornecimento pelos fabricantes de recursos de processamento *bitblt*. Um exemplo ilustrativo é a família de memórias de vídeo da Hitachi [Hita 89] que possuem integrado no próprio chip de memória um dispositivo ALU para este processamento. Além destes recursos, outros são colocados como, por exemplo, a programação do *bit* de início de deslocamento e a inibição de escrita por meio de máscara de acesso. Infelizmente estes recursos não são padronizados entre diversos fabricantes, de forma que não é possível generalizá-los.

Capítulo 4

Arquiteturas para a Geração de Imagem com Realismo

No capítulo 2 foram introduzidas as diversas tarefas que podem ser incorporadas a um sistema de geração de imagens com realismo. No capítulo 3 foram discutidos os diversos esquemas de circuitos de exibição de imagem, ou seja, as alternativas de implementação do circuito responsável pela exibição dos resultados da execução das tarefas mencionadas. Neste capítulo serão apresentadas as alternativas de arquitetura de sistemas de tratamento de imagem com realismo.

Esta apresentação é feita agora, depois das tarefas a serem executadas e dos circuitos de exibição possíveis, pois muitas vezes as arquiteturas de execução das tarefas estão incorporadas na própria estrutura destes circuitos.

4.1 - O Processo de Geração de Imagens com Realismo

O Processo de Geração de Imagens com Realismo parte de uma descrição formal de uma cena, seguida da transformação desta descrição em estímulos visuais para a retina do olho humano, como mostrado na figura 4.1.

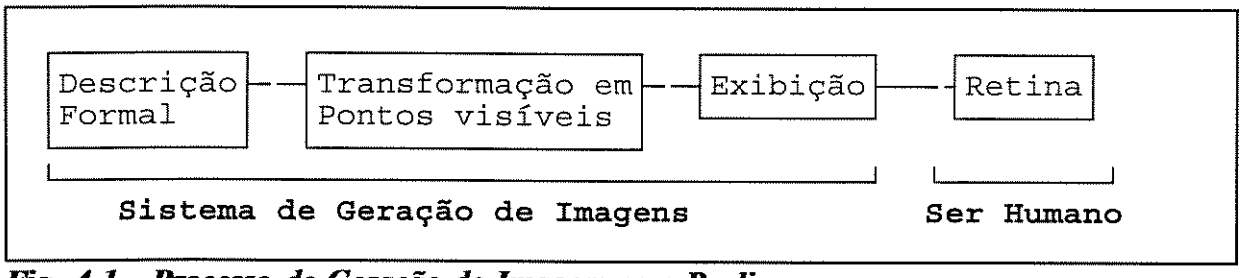


Fig. 4.1 - Processo de Geração de Imagem com Realismo.

Os estímulos visuais são produzidos através de um monitor de vídeo, que recebe de uma interface de saída gráfica sinais elétricos proporcionais aos níveis de estímulos visuais a serem produzidos. Para que os sinais elétricos sejam constantemente enviados ao monitor, os valores destes são mantidos numa memória de quadros interna à interface de saída para o monitor. Um estudo

sobre as principais técnicas de implementação de circuitos de interface com o monitor de vídeo já foi apresentado no capítulo 3 (Circuitos de Exibição de Imagem).

O núcleo principal do processo de geração é o **Mapeamento da Imagem**. Ou seja, a codificação dos dados da descrição formal em valores de cores e de localização no plano de exibição. Estes valores são armazenados em posições da memória de vídeo correspondentes a sua localização no plano da tela.

A descrição formal da imagem é normalmente realizada através de uma estrutura de dados acessível pelo processador de transformação. Esta estrutura de dados está, geralmente, comprometida com o tipo de processamento que será realizado.

O Mapeamento de Imagens deve observar o compromisso entre dois requisitos fundamentais:

- **Qualidade,** e
- **Rapidez no Tratamento.**

A Qualidade exige que o resultado do processamento apresente um padrão de realismo que implica em maior resolução da imagem exibida e maior definição de cores. São necessários ainda modelos de iluminação mais elaborados que considerem o tratamento de rugosidade, difração, polarização e movimento dos objetos. Estes modelos mais elaborados normalmente exigem uma alta capacidade de processamento (processador e memória).

A Rapidez no Tratamento é um requisito que está mais relacionado com a interação com o usuário do sistema ou com a aplicação do sistema de geração de imagens. Segundo este requisito os resultados da geração de imagens devem ser obtidos o mais rapidamente possível. Por exemplo, num sistema de animação, as imagens geradas devem ser tratadas dentro de um período de 30 quadros por segundo, para evitar a percepção de descontinuidade pela vista humana.

Estes dois requisitos são conflitantes e em função da aplicação, um deles é sacrificado. Por exemplo, os sistemas de animação em tempo real, atualmente disponíveis, ainda sacrificam muito a qualidade da imagem para atingir o compromisso de velocidade. Já os sistemas de editoração gráfica podem ser mais lentos mas devem gerar imagens com alto padrão de qualidade.

4.1.1 - Formas de Mapeamento da Imagem.

A implementação do mapeamento dos valores da imagem pode ser feita de duas formas: Mapeamento direto ou Mapeamento reverso.

O **Mapeamento Direto** parte da estrutura de dados que descreve a imagem a ser exibida e gera os valores para os pontos da tela que são ocupados pelos objetos da imagem. Durante muito tempo este foi o mapeamento mais adotado. Ele é formado pela sequência de tarefas que projetam cada objeto (ou superfície) no plano da tela; seguida da amostragem dos objetos nos *pixels*; da varredura em linha e do envio para o monitor de vídeo. De acordo com este mapeamento o sistema de geração de imagens da figura 4.1 pode ser detalhado como mostrado na figura 4.2, onde cada tarefa já foi descrita no capítulo 2 (Geração de Imagens com Realismo). No mapeamento direto a estrutura de dados da imagem é investigada uma única vez. Entretanto neste tipo de mapeamento há uma perda da noção de conjunto da imagem, por exemplo, as interações de sombra e reflexões são mais difíceis de serem tratadas.

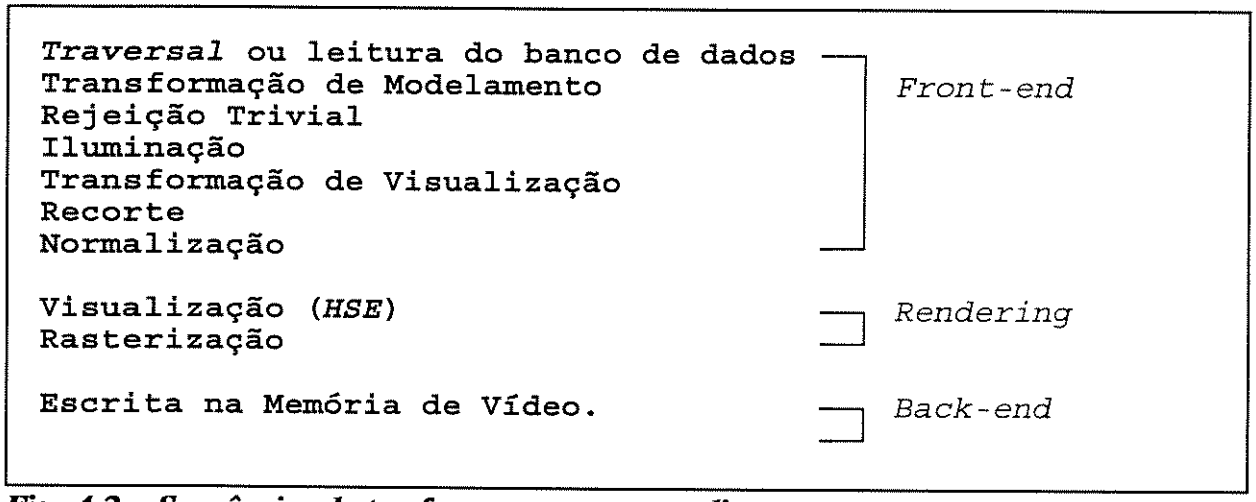


Fig. 4.2 - Sequências de tarefas no mapeamento direto.

O **Mapeamento Reverso** parte dos pontos da tela e para cada ponto é varrida toda a estrutura de dados analisando se o ponto atual é ocupado pelos objetos da imagem. No mapeamento reverso, são utilizados raios que partem de cada *pixel* na tela (*ray casting*). O procedimento passa e repassa diversas vezes sobre a estrutura de dados analisando as interações dos objetos com o raio emitido daquele *pixel*. Neste mapeamento os efeitos são mais próximos da realidade de forma que reflexões, difrações e refrações são tratadas com mais facilidade. Normalmente neste mapeamento os tratamentos de volume, de sombreamento e de iluminação são feitos num único procedimento.

4.1.2 - Comparação entre mapeamento direto e reverso.

Embora seja impossível estabelecer com rigor a duração de cada etapa da figura 4.2, estima-se [Whit 90] que numa máquina sequencial o *front-end* consome tipicamente entre 5 a 10 porcento do tempo total de geração da imagem; o *rendering* consome cerca de 90 porcento do tempo e o *back-end* entre 1 a 5 porcento do tempo.

Comparação entre as formas de mapeamento

MAPEAMENTO DIRETO

Estrutura de dados (*N* objetos todos com *p* pixels)		**mapa de pixel (*T* pixels)**
cubo (com *p* pixels)	------>	aciona os *p* pixels do cubo
esfera (com *p* pixels)	------>	aciona os *p* pixels da esfera
...		
fim da imagem		

O mapeamento direto requer portanto N x p interações de acionamento de pixels na tela.

MAPEAMENTO REVERSO

Estrutura de dados (*N* objetos, todos com *p* pixels)		**mapa de pixel (*T* pixels)**
cubo (com *p* pixels)	*T*	para todos os pixels da tela
esfera (com *p* pixels)	<-------	analise os objetos cobertos
...		e aciona de acordo com a
fim da imagem		cobertura.

O mapeamento reverso requer *T* x *N* análises de cobertura.

A maior parte dos trabalhos que visa aumentar o desempenho da geração de imagem com realismo ataca justamente o *rendering*. Pela sua própria natureza, esta etapa é a mais fácil de ser paralelizada. Entretanto, mesmo num equipamento multiprocessador com as tarefas de *rendering* adequadamente distribuídas, o processamento gráfico é atrasado pela característica sequencial do acesso ao banco de dados e pelas baixas taxas de transferência para a memória de quadro. O maior problema passa a ser: como agilizar as tarefas relativas ao *front-end* e ao *back-end*.

Diversos exemplos de arquiteturas procuram incorporar, parcial ou totalmente, o *back-end* ao *rendering* e com ele paralelizá-lo, por exemplo: Pixel-Machine

[PoHo 89], Silicon Graphics 4D/240GTX [Akel 89], DN10000VS [KiVo 90] e outras descritas em [BBPr 90]. [MoFu 90] descreve algumas alternativas para a solução do problema de desempenho relativo ao *front-end*, principalmente o acesso à descrição da imagem.

Por outro lado, a adoção do mapeamento direto tende a diminuir considerando-se que:

- O número de objetos tratados em cenas tende cada vez mais a crescer (uma imagem com boa qualidade tem, hoje em dia, entre um mil a um milhão de objetos).
- O número de *pixels* na tela dos monitores não tem crescido muito (Tem ficado em torno de 1,5 Mega*pixels*).

4.2 - Aceleração do Processamento Gráfico.

Para atingir taxas mais elevadas de tratamento da informação, os sistemas gráficos precisam vencer as seguintes barreiras:

- Processamento Geométrico em Ponto Flutuante;
- Processamento de *pixel* em Inteiro;
- Banda de Passagem para a Memória de Quadro.

Para enfrentar estes obstáculos são utilizados avanços na tecnologia de circuitos e multiprocessamento, conforme descrito a seguir.

4.2.1 - Tecnologia de circuitos.

Avanços na tecnologia de circuitos permitem aumentar a velocidade do tratamento em *hardware* e a densidade de circuitos, possibilitando que mais tarefas possam ser tratadas pelo *hardware*. O principal problema com o tratamento por *hardware*, é o compromisso que deve haver entre definição de uma máquina especializada e o surgimento de novos algoritmos ou evolução dos existentes.

Hoje em dia as tarefas que têm merecido mais atenção para a solução em *hardware* são:

- *Gouraud Shading*
- *Z-buffering*

- *Antialiasing*
- *Texture Mapping*
- *Alpha Buffering*

A evolução da tecnologia dos circuitos contribui também com o lançamento de novos processadores e principalmente com novos circuitos de memória - VRAM's.

Diversos trabalhos sobre circuitos integrados especializados para computação gráfica têm sido feitos. Estes esforços são estimulados pelos avanços nas técnicas de integração e de projetos de circuitos VLSI(*Very Large Scale Integration*). Estes circuitos são tipicamente projetados para uma operação gráfica específica e são usados como parte de um *pipeline* de processamento. Alguns exemplos de circuitos especializados são descritos na seção 4.6.

4.2.2 - Multiprocessamento

Todos os equipamentos de alto desempenho para o tratamento gráfico tem algum grau de multiprocessamento. Alguns equipamentos utilizam uma arquitetura multiprocessadora de propósito geral e sobre ela é desenvolvida a programação da aplicação; outros equipamentos possuem múltiplos circuitos específicos para este tratamento. O multiprocessamento pode ser distribuído:

- por cenas num sistema de animação;
- por objetos;
- por primitivas de tela;
- por *pixel* ou grupo de *pixels*;
- por componentes de cor, alfa e de profundidade; ou,
- por funções numa linha de processamento (*pipeline*).

De acordo com a sequência definida na figura 4.2 , o *front-end* envolve muitas transferências de entrada e saída e é difícil de ser paralelizado. O *Rendering* é a etapa que consome mais tempo num equipamento sequencial e é a parte mais fácil de ser paralelizada. O *back-end* é paralelizável mas envolve muitas operações de entrada e saída.Um grande número de arquiteturas dirigidas ao suporte de computação gráfica têm sido descritas na literatura. Alguns exemplos de arquiteturas multiprocessadoras para aplicação em gráficos descritas na literatura são comentadas na seção 4.6.

4.3 - Multiprocessamento aplicado a Síntese de Imagens.

Esta seção introduz os conceitos básicos de multiprocessamento aplicado a síntese de imagens, além de apresentar as principais implicações desta aplicação sobre sistemas multi-processadores.

4.3.1 - Conceitos básicos de Multi-processamento.

Interessa ao processamento gráfico duas organizações básicas de multiprocessadores:

- **Organização de processadores em Linha (*pipeline*)**, de acordo com o diagrama mostrado na figura 4.3a;
- **Organização dos processadores em Paralelo**, de acordo com o diagrama mostrado na figura 4.3b.

Em função da aplicação, é possível organizar o sistema de multiprocessamento

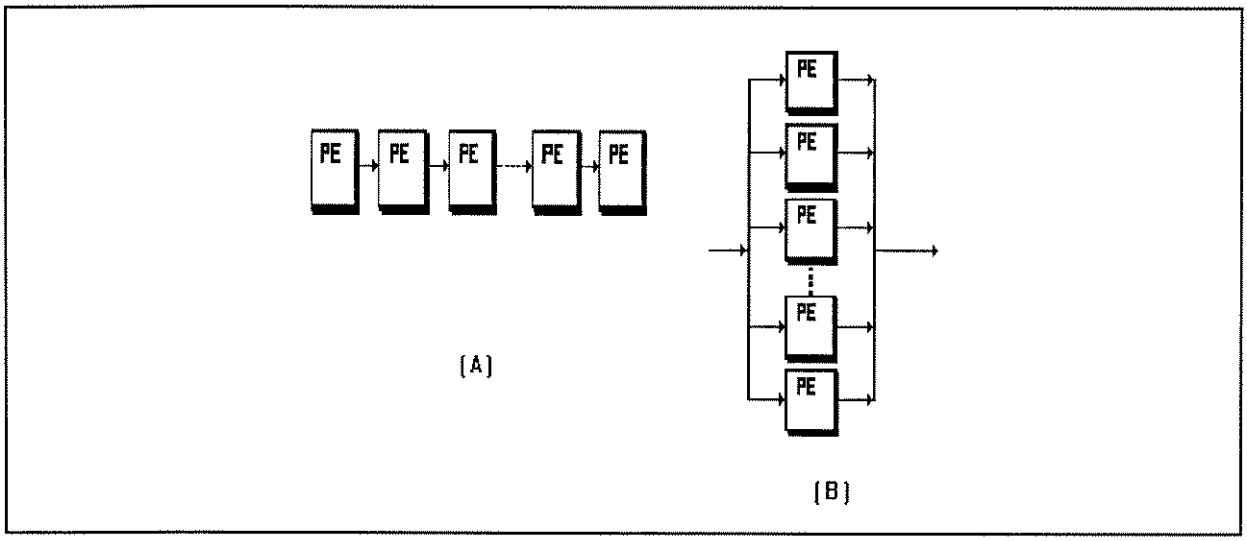


Fig. 4.3 - Organizações básicas de Multiprocessamento. A) Em Linha; B) Em Paralelo.

combinando as configurações acima formando, por exemplo, **Paralelo de Linhas** (de acordo com a figura 4.4a) ou **Linha de Paralelos** (de acordo com a figura 4.4b).

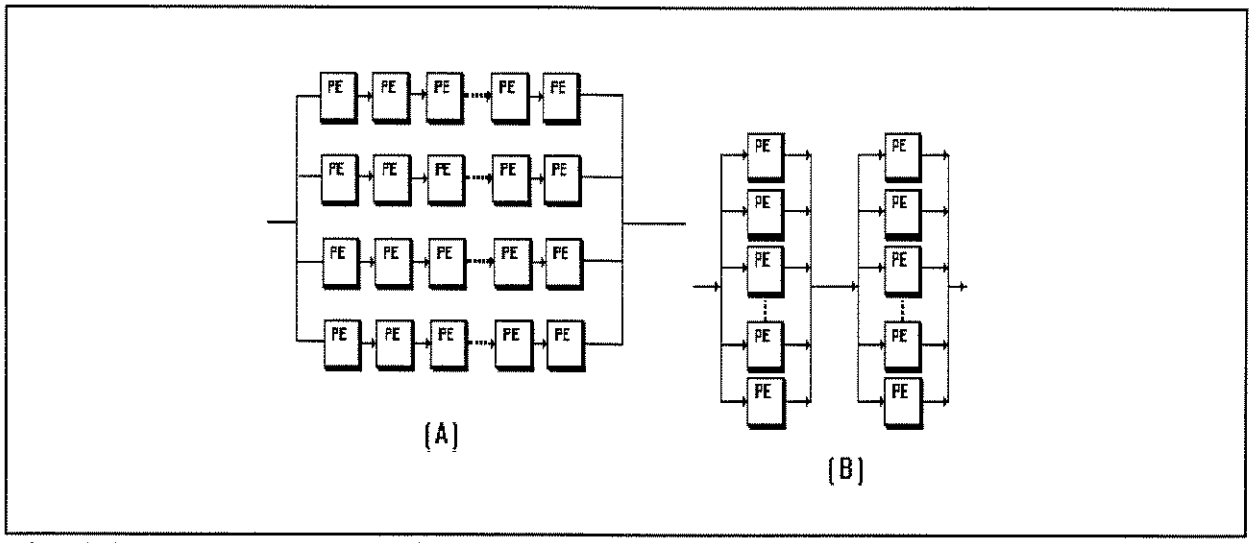


Fig. 4.4 - Organizações possíveis a) - Paralelo de linhas; b) - linha de Paralelos.

4.3.2 - Organização em Linha - *Pipeline*

A organização em *pipeline* é muito usada em arquiteturas especializadas em processamento gráfico. Para entender como a organização em *pipeline* aumenta o desempenho de um sistema de processamento gráfico pode-se compará-la com uma organização monoprocessador.

Por exemplo, se numa organização monoprocessador executando um sistema gráfico com P processos (travessal, recorte, ...) for necessário tratar uma cena com G elementos gráficos e se cada processo demorar a mesma T unidade de tempo; para gerar uma cena serão necessários:

$$P \times G \times T \quad \textit{unidades de tempo}$$

como mostrado na figura 4.5a.

Considerando o mesmo sistema gráfico, se for utilizado um sistema multiprocessador com P processadores colocados em linha (*pipeline*), de forma que cada um destes processadores execute um dos processos gráficos; os processos dos G elementos gráficos poderão ser sobrepostos de forma que o tempo total da geração da cena necessitará de:

$$[G + (P - 1)] \times T \quad \textit{unidades de tempo}$$

como mostrado na figura 4.5b. Nesta figura, o eixo das abscissas representa o tempo e o eixo das ordenadas representa os processos a serem tratados, G_i representa a fatia de tempo para o processamento do elemento gráfico G_i, P_i representa os processos necessários ao tratamento de cada elemento gráfico.

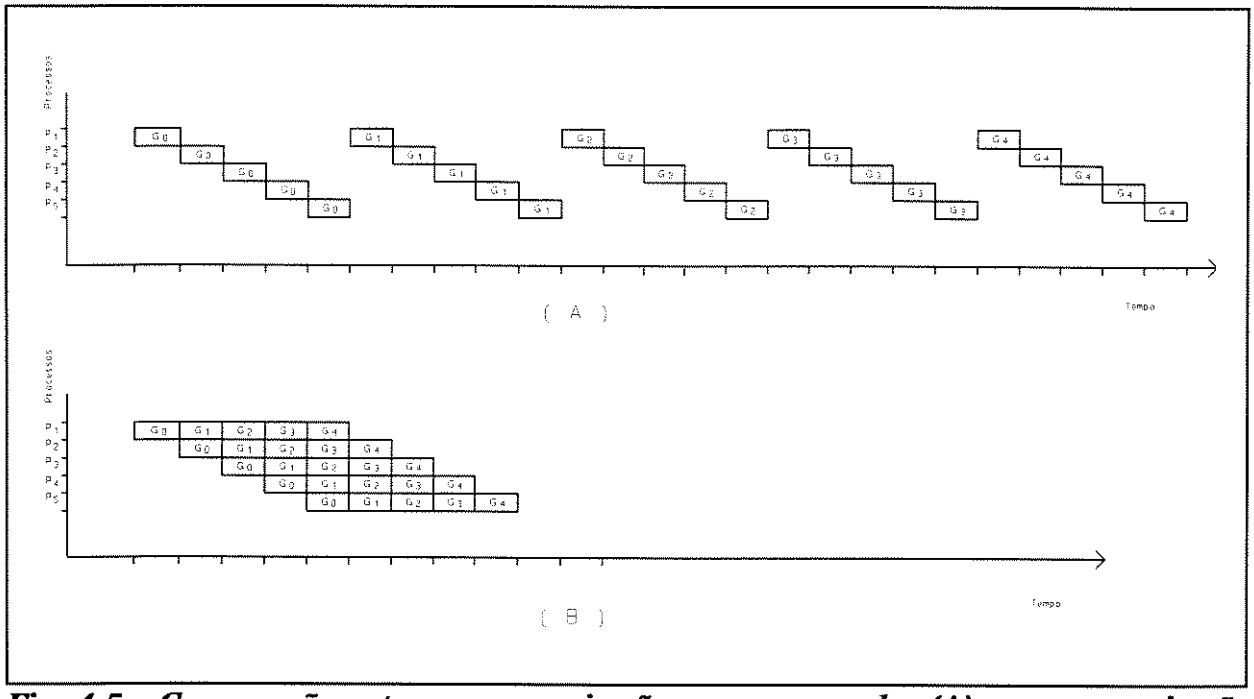


Fig. 4.5 - Comparação entre uma organização monoprocessador (A) e uma organização em pipeline — (B), tratando P=5 processos(processadores) em G=5 elementos gráficos.

Neste exemplo são considerados somente cinco processos.

De uma forma geral pode-se dizer que a organização em *pipeline* apresenta um ganho máximo de desempenho, comparado com uma organização monoprocessador, igual ao número de processadores. O ganho só se aproxima deste valor máximo se o número de elementos a ser tratado for muito grande. Além disso, é necessário que o tempo gasto com cada processamento seja aproximadamente igual para todos, como mostrado na figura 4.5.

A taxa total na qual os dados são tratados é normalmente referenciada na literatura internacional como o ***Throughput*** do sistema. O tempo que um único elemento de dado leva do início ao fim do *pipeline* é chamado de **Latência**. Uma organização em *pipeline* longa, ou seja com muitos processadores apresenta um alto *throughput*, no entanto apresenta também uma alta latência. Esta alta latência pode significar um problema para aplicações gráficas com uma alta interação com o usuário. Pois para uma única ação do usuário o elemento gráfico gerado precisará atravessar toda a linha de processadores.

4.3.3 - Organização em Paralelo

As organizações em paralelo podem ser classificadas de diversas formas, as mais importantes consideram o tipo de processadores elementares e o tipo de

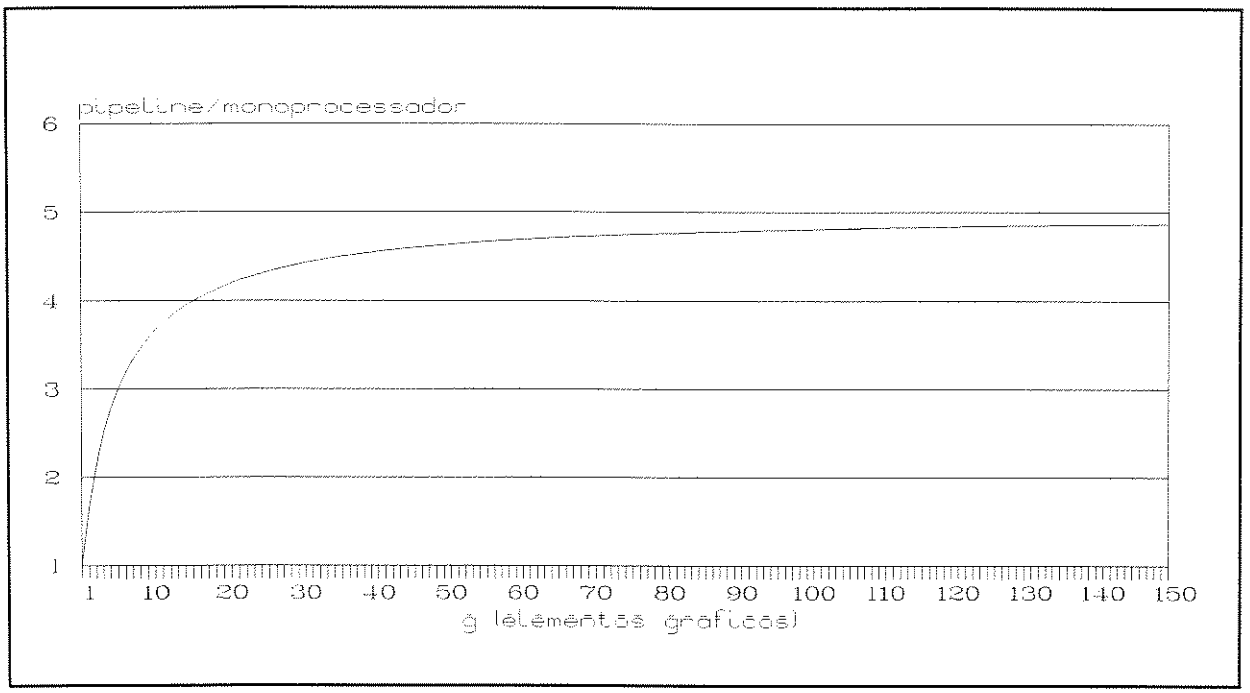


Fig. 4.6 - Ganho da organização pipeline (P=5) x organização monoprocessadora.

controle do processamento. Com respeito ao tipo de processadores elementares utilizados existem organizações dos tipos:

> **Homogênea**: quando os processadores elementares são idênticos; e,
> **Heterogênea**: quando os processadores elementares são diferentes.

Com respeito ao tipo de controle, importam ao processamento gráfico as organizações:

SIMD (**S**ingle **I**nstruction stream, **M**ultiple **D**ata stream)
MIMD (**M**ultiple **I**nstruction stream, **M**ultiple **D**ata stream)

Esta classificação é baseada na taxonomia introduzida por [Flyn 66] e resumida na figura 4.7. A organização SIMD executa a mesma instrução em diversos processadores com diferentes dados. A organização MIMD executa diferentes instruções em diversos processadores com diferentes dados.

SIMD

Na organização SIMD todos os processadores elementares compartilham uma única memória de código. As principais limitações de uma organização SIMD são relacionadas ao tratamento de desvios condicionais e no acesso a dados utilizando apontadores e indexadores.

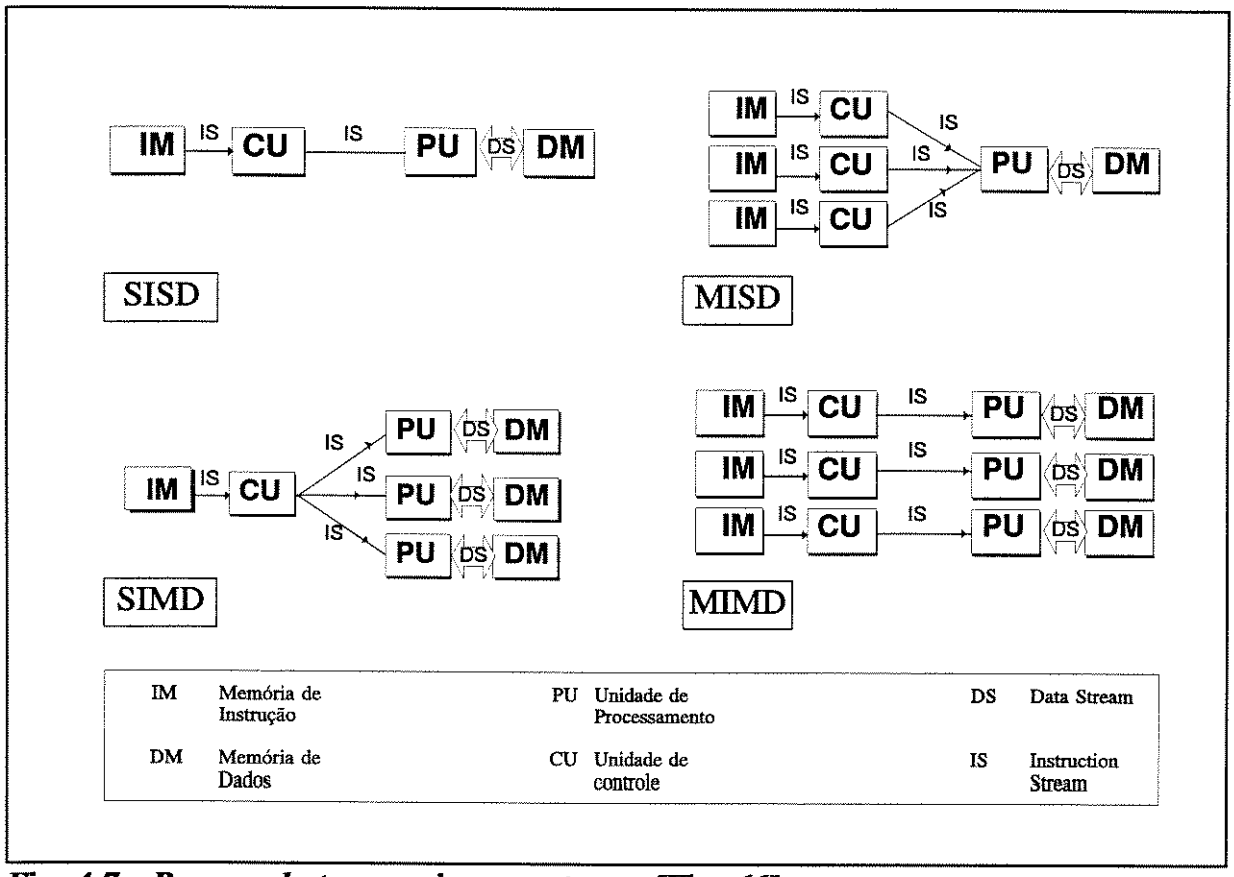


Fig. 4.7 - Resumo da taxonomia proposta por [Flyn 66].

O tratamento de desvios condicionais depende de valores de dados específicos de cada processador elementar. Como estes dados podem ter valores diferentes, cada processador precisaria desviar para pontos diferentes do programa. Como nesta organização a sequência de operações é comum a todos os processadores eles devem executar todas as alternativas de desvio de programa. Para acomodar o tratamento de desvios condicionais, cada processador elementar deve possuir um registrador de habilitação de operações de escrita. O princípio da utilidade deste registrador é que somente os processadores com este registrador no estado de habilitação conseguem realizar micro-operações de escrita. De forma que o processador com este registrador no estado de inibição pode até executar uma instrução que alteraria os valores de seus registros; no entanto como as micro-operações de escrita estão inibidas esta instrução é executada sem afetar nada. Algoritmos com poucos desvios condicionais executam eficientemente em máquinas SIMD. Entretanto algoritmos com diversos desvios condicionais podem ser extremamente ineficientes pois muitos processadores elementares poderão estar inibidos na maior parte do tempo.

O tratamento de dados utilizando-se apontadores e indexadores pode causar problemas similares ao do desvio condicional, uma vez que cada apontador ou

A) Primeiramente o algoritmo do trecho.

```
1   ...
2   se A = 0 {
3           imprime ("FALSO") }
4   senão {
5           imprime ("VERDADEIRO") }
6   continuação do programa
7   ...
```

B) agora como este algoritmo é adaptado a uma máquina SIMD

```
1   ...
2   se A != 0 { habilitador = 0}
3   imprime ("VERDADEIRO")
4   complementa o habilitador
5   imprime ("FALSO")
6   habilitador = 1
7   continuação do programa
8   ...
```

Os dois processadores da máquina exemplo executariam os mesmos sete passos do trecho de programa. Se considerarmos que a variável A na máquina 0 for igual a zero e na máquina 1 esta variável tiver o valor 1, a máquina 0 imprimirá FALSO e a máquina 1 imprimirá VERDADEIRO.

Fig. 4.8 - Trecho de programa que ilustra o uso do registrador de habilitação de escrita para acomodar o desvio condicional em máquinas SIMD.

indexador pode assumir valores diferentes nos diversos processadores elementares.

MIMD

A organização de processamento MIMD é mais cara que uma do tipo SIMD equivalente, uma vez que cada processador possue sua própria memória de código e seu controlador. Os processadores elementares numa máquina MIMD sempre executam o mesmo programa. Entretanto eles não têm o compromisso de operar executando simultaneamente os mesmos passos. Por esta liberdade, os processadores elementares não sofrem nenhum obstáculo ao encontrar desvios condicionais, pois cada processador toma uma decisão independente saltando instruções que não necessitam ser executadas. Com resultado disto, as máquinas MIMD atingem uma maior maior eficiência na maior parte das aplicações. Entretanto, como os processadores podem começar e terminar em diferentes tempos e podem processar dados em diferentes taxas, a sincronização

e o balanceamento de carga é mais difícil, exigindo frequentemente a adição de *buffers* na entrada e saída de cada processador elementar.

Resumo comparativo entre máquinas SIMD x máquinas MIMD.

Resumindo tem-se que:

- Uma arquitetura **SIMD** é mais simples de controlar, por possuir somente uma unidade de controle da sequência de operações. Por isto mesmo é mais barata.
- A arquitetura **SIMD** trabalha melhor em operações repetitivas, sobre um conjunto diversificado de dados.
- A arquitetura **MIMD** trabalha melhor em operações diversificadas, especialmente se as operações têm tempos diversificados.
- **MIMD** pode perder na lógica de controle o que ganha em desempenho.
- **MIMD** é difícil de programar quando uma operação é distribuida, e é mais difícil de depurar.

A seção seguinte faz uma comparação da aplicação das arquiteturas em linha e paralela nas diversas etapas de um sistema de geração de imagens.

4.4 - Arquiteturas com Multiprocessamento Para Sistemas Gráficos.

Processadores em paralelos ou em *pipeline* são os blocos básicos de virtualmente todos os sistemas gráficos de alto desempenho disponíveis. Ambas as técnicas podem ser utilizadas para acelerar tanto o subsistema *front-end* quanto o *back-end* de um sistema gráfico [MoFu 90]. Esta seção examina cada uma destas estratégias.

4.4.1 - Arquiteturas com Multiprocessadores no *Front-end*

O *front-end* de um sistema gráfico tem que tratar duas grandes tarefas: a leitura da base de dados que descreve a imagem; e a transformação das primitivas para o espaço da tela. Como estas tarefas estão intrinsecamente em série, uma arquitetura em *pipeline* pode ter os seus estágios cuidandos de cada uma das tarefas do *front-end*. Entretanto, como numa descrição de imagens podem existir diversas primitivas que podem ser manipuladas simultaneamente, uma arquitetura em paralelo pode ser utilizada com sucesso.

Embora a arquitetura em *pipeline* venha sendo a técnica predominante nos sistemas gráficos, arquiteturas em paralelo oferecem diversas vantagens, dentre elas destacam-se: — a reconfigurabilidade para diversos algoritmos, uma vez que um único processador realiza todos os cálculos do *front-end*, — a modularidade, uma vez que os processadores elementares podem ser idênticos, de forma mais fácil que sistemas em *pipeline*, — e a escalabilidade de processamento, ou seja a anexação de mais um processador em paralelo pode acelerar o processamento do sistema, ao passo que num sistema em *pipeline*, a velocidade do sistema é sempre definida pelo seu estágio mais lento. Por outro lado, os sistemas em paralelo requerem sincronizações e distribuição de carga mais complexas. Estes sistemas também não podem utilizar processadores especializados tão bem quanto nos sistemas em *pipeline*.

4.4.2 - Arquiteturas com Multiprocessadores no *Back-end*

A saída do subsistema *Front-end* é tipicamente um conjunto de primitivas em coordenadas de tela. Já o subsistema de *back-end* (incluindo aí o *rastering*) cria a imagem final fazendo o *scan-converting* das primitivas, determinando qual primitiva é visível em cada *pixel* e tonalizando-o corretamente.

4.4.2.1 - Rastering em Paralelo

O grande problema do multiprocessamento no *rastering* é a necessidade de uma grande banda de passagem entre a estrutura de processamento e a memória de quadro. Ou seja, não adianta ter uma grande capacidade de processamento sem um meio de transferir as informações para a memória de quadro de forma rápida. O princípio básico para o *rastering* em paralelo é o acionamento simultâneo de diversos *pixels*. Para que este acionamento seja realizado em paralelo, devem ser criados diversos acessos independentes a diferentes partes da memória de quadro. Ou seja, a memória de quadros deve ser particionada e processadores são associados a estas partições tratando independentemente o seu conjunto de *pixels*. Dois problemas passam então a ser considerados, primeiro, como particionar os *pixels* da tela, e segundo, quantas partições são necessárias.

Como Particionar a Memória de Quadro.

Duas estratégias básicas são indicadas para isto: o Particionamento Contíguo (4.9a) e o Particionamento Intercalado (4.9b).
No particionamento contíguo cada processador trata independentemente uma região contígua da tela. Se as primitivas a serem exibidas estiverem uniformen-

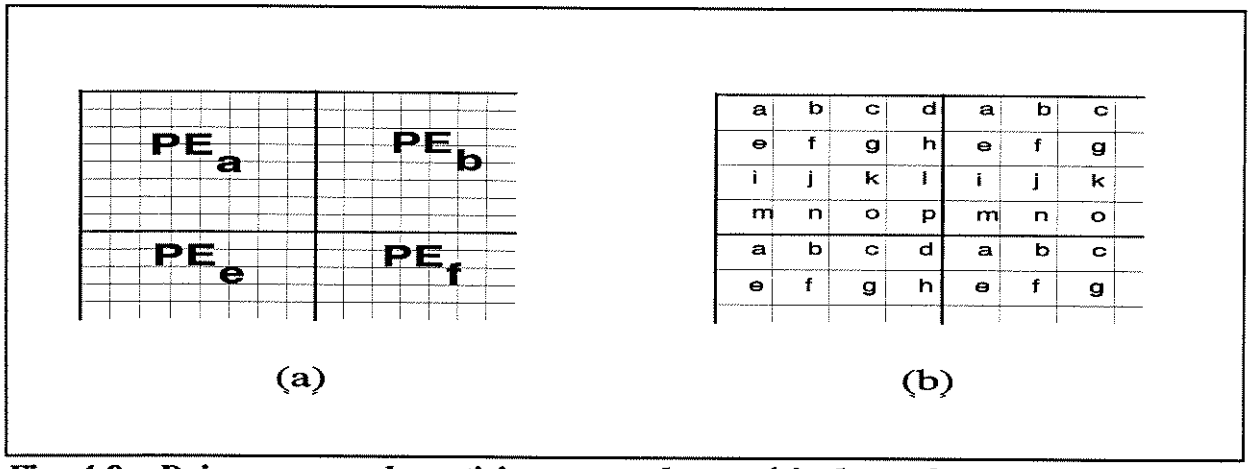


Fig. 4.9 - Dois esquemas de particionamento da memória de quadros.

temente distribuídas nas diversas partições, há um ganho significativo na velocidade de processamento. Entretanto se houver uma concentração de primitivas numa única região da tela, o desempenho do sistema cai para o desempenho de um único processador. Ou seja, este é o conhecido problema de distribuição de carga de sistema multi-processador.

No particionamento intercalado, o problema de distribuição de carga é minimizado, pois cada primitiva pode ser tratada por diversos processadores independentemente, cada um tratando uma fração da primitiva. Neste esquema, o conjunto dos processadores formam o que se denomina de pegada (*footprint*) de processamento, pois em função da localização da primitiva, sempre um conjunto definido de processadores estarão atuando simultaneamente. Pelo seu natural equilíbrio de carga entre os processadores, esta técnica tem sido a preferida. A figura 4.9b mostra um exemplo de particionamento intercalado com uma pegada de 4x4 *pixels*. Ou seja, cada processador é responsável por um *pixel* em cada bloco de 4x4 *pixels*. A figura 4.10 mostra num diagrama de blocos como um sistema gráfico pode ser implementado com particionamento intercalado com pegada 4×4.

A análise que se faz ao particionamento intercalado é que ele tem um desempenho pior no seu caso mais favorável do que a partição contígua, mas é muito mais eficiente no seu pior caso pois o seu desempenho irá depender somente da quantidade de primitivas existentes e não da localização delas.

No particionamento intercalado, com um processador elementar por partição interessa classificar como estes irão atuar. Se todos obedecem simultanemente a uma sequência de comandos, característica de uma organização SIMD, ou todos atuam de forma independente como uma estrutura MIMD. No primeiro caso, o efeito é que todos os processadores atuam sempre sobre a mesma

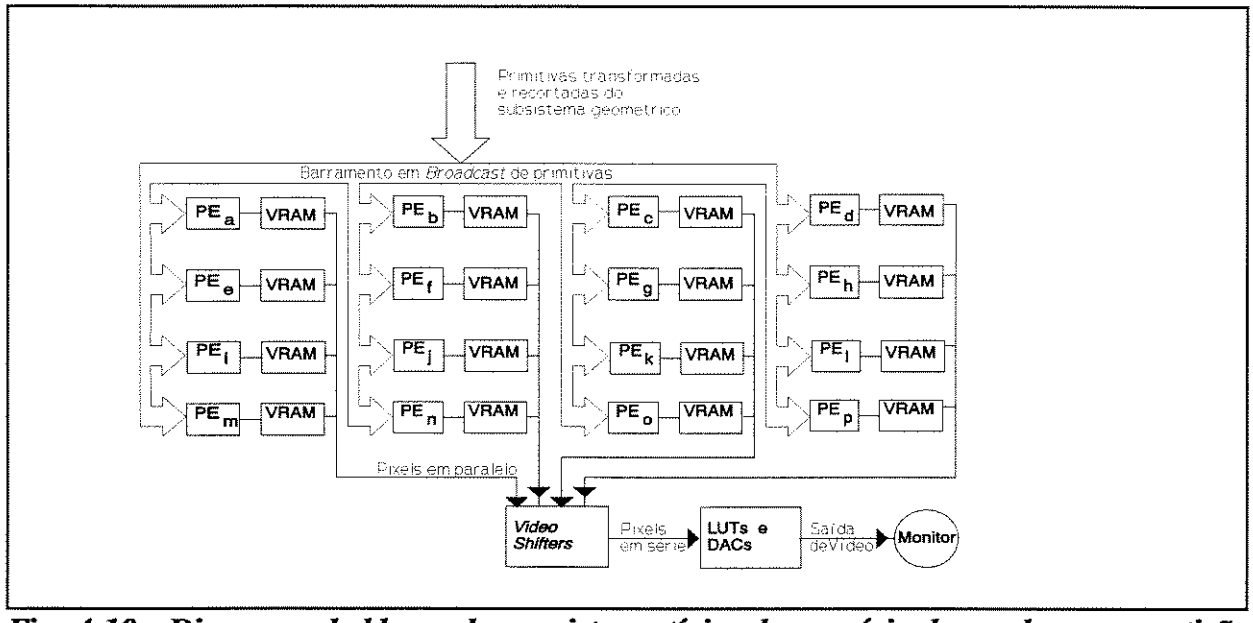


Fig. 4.10 - Diagrama de blocos de um sistema típico de memória de quadro com partição intercalada. Pegada 4x4

pegada. Por isto mesmo, esta forma de organização é chamada de Processador de Pegada (*Footprint Processor*). As principais desvantagens do processador de pegada são: Ele utiliza muito mal os processadores elementares, considere o exemplo de uma operação em que somente um *pixel* na pegada precisar ser modificado, nesta organização, todos os processadores irão ser ativados para realizar operações nulas juntos com o processador associado ao referido *pixel*; — outra desvantagem é que o desempenho estará fortemente dependente do algoritmo a ser executado na estrutura. Por exemplo, algoritmos sem muitos desvios condicionais podem ser implementados de forma muito eficientes, entretanto algoritmos mais complexos com muitos desvios condicionais, podem ser comprometidos. As grandes vantagens desta forma de organização são: as suas dimensões físicas, a sua simplicidade e o seu baixo custo de implementação.

Se os processadores estão organizados em MIMD, cada processador pode estar atuando sobre pegada diferente dos demais. Com a adição de FIFO de primitivas nos processadores elementares, cada processador poderá atuar em primitivas distintas. A necessidade de controle e memória de instrução locais, mais filas FIFO e o *hardware* de sincronismo, aumentam o tamanho e o custo dos equipamentos com esta organização.

Determinação do Número de Partições.

Após apresentar as formas de organização da memória em partições para aumentar a banda de passagem para a memória de quadro, resta definir qual o melhor número de partições. Quanto maior o número de partições maior será esta banda. Entretanto, quanto maior o número de partições mais interligações e mais *chips* de memória serão necessários, ou seja maior o custo do *hardware*. Este compromisso é mais sério a medida que aumenta a densidade do *chip* de memória utilizado. Consideremos, por exemplo, as alternativas para a resolução da tela de 1024 *pixels* de 32 *bits* em 1024 linhas. Primeiramente sem partições, ou seja, entre a estrutura de processamento e a memória de quadro existe somente um barramento. Para esta resolução é necessária uma memória de quadro para 1 Mega *pixels*, ou em termos de *bits*, uma memória de 32 Mega *bits*. A quantidade de *chips* necessários para a implementação é função do seu tipo. Vide na tabela 4.1, na coluna de partição = 1 a quantidade de *chips* de acordo com a dimensão do mesmo.

Se, para este mesmo exemplo de resolução, for definida uma organização com $p = n \times n$ partições serão necessários p barramentos de 32 *bits*, cada um para ligar um processador elementar a uma partição da memória. Independente do número de partições ou da resolução total, cada uma das partições tem um número mínimo — m de *chips* para configurar o barramento de 32 *bits*. O número total de *chips* será dado, no mínimo, por $p \times m$.

A tabela 4.1b encerra a comparação do exemplo acima mostrando o desperdício percentual de posições de memória de acordo com o tipo de *chip* utilizado e a dimensão da partição. Na tabela 4.1b observa-se que quanto maior o número de partições, maior o desperdício de posições de memória para os *chips* mais densos. Por exemplo, para uma partição de 8×8 o uso de uma memória de 1024×4 *bits*, implica num desperdício maior que 90%, ou seja para permitir uma alta banda de passagem é preciso ligar 10 vezes mais *chips* de memória que o necessário.

Este desperdício deve ser considerado com cuidado pois ele poderá ser útil para armazenar dados temporários (por exemplo: páginas de tela, regiões de *windows,..etc.*). Por exemplo se for utilizado um *chip* de 256×4 numa partição 4×4, o desperdício é de 75%, o que pode ser interpretado como sendo a disponibilidade para mais três páginas de tela por partição.

Tab. 4.1 - Comparação entre diversos números de partições para uma memória de quadro com 1024 × 1024 pixels.

Dimensão de *chips*	Número de Partições			
	1	2 × 2	4 × 4	8 × 8
64 × 4	128	128	128	512
256 × 4	32	32	128	512
1024 × 4	8	32	128	512
64 × 1	512	512	512	2048
256 × 1	128	512	512	2048
1024 × 1	32	128	512	2048
Banda de Passagem[1] (Pixel/Seg)	3 846 154	15 384 615	61 538 461	246 153 846

[1] Considerado como tempo de acesso = 260 nano-segundos

(a)

Número de Partições			
1	2 × 2	4 × 4	8 × 8
0	0	0	75
0	0	75	93
0	75	93	98
0	0	0	75
0	75	75	93
0	75	93	98

(b)

4.4.3 - Arquiteturas para Composição de Imagens.

Um tipo de arquitetura importante é a chamada Arquitetura para Composição de Imagens.

O seu princípio básico é distribuir primitivas sobre diversos sistemas completos de *rendering* (*renderer*). Estes sistemas são sincronizados de forma que eles utilizam sempre as mesmas matrizes de transformação e tratam sempre o mesmo quadro. Cada *renderer* trata sua parte da imagem independentemente e a armazena na sua própria memória de quadro.

Os *pixels* são retirados de cada memória de quadro normalmente em série, com a diferença que nesta saída é também serializada a informação de profundidade do *pixel* (z). Esta retirada é sincronizada em todas as memórias de quadro de forma que todas elas fornecem sempre o mesmo *pixel*. Uma árvore de circuitos especiais (Combinadores) combinam as sequências de valores de RGB e z provenientes de cada *renderer* usando uma técnica que é uma simplificação da proposta em [Duff 85]. A figura 4.11 ilustra um sistema de composição com 4 *renderers*. No exemplo cada deles trata um objeto da imagem e no final é exibida a combinação dos 4 objetos.

Arquiteturas para combinação de imagens podem gerar imagens com alta velocidade, se forem utilizados um grande número de *renderers* em paralelo.

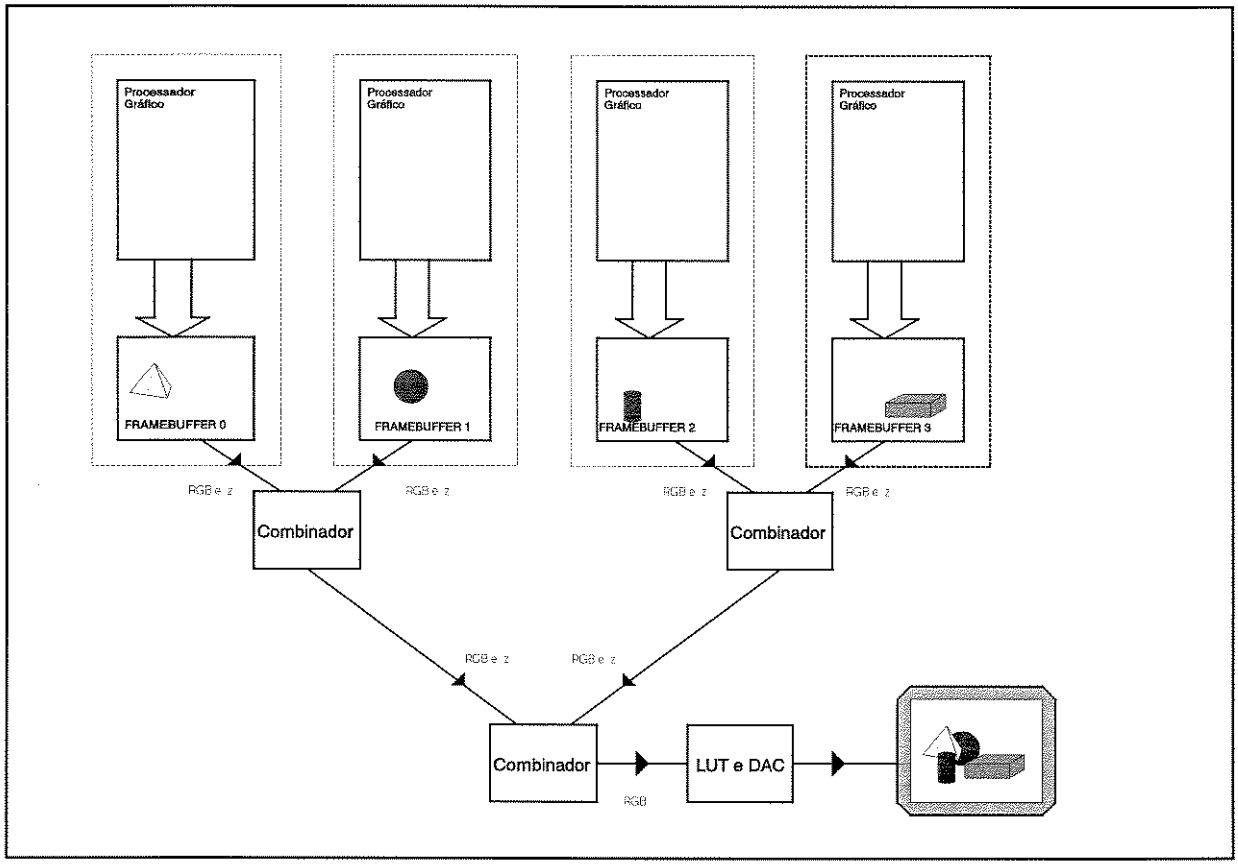


Fig. 4.11 - Um sistema para Composição de Imagem com 4 renderers

Entretanto esta técnica apresenta como desvantangens: a dificuldade de distribuição da base de dados nos diversos processadores; problemas com *aliasing* e a geração de *pixels* errôneos pelos circuitos combinadores; e uma deficiência de flexibilidade, uma vez que a combinação de imagens restrige os algoritmos que podem ser implementados no sistema.

Como exemplos de arquiteturas que utilizam esta técnica existem: — PixelFlow descrito em [MEPo 92] e PROOF descrito em [ScCl 88].

4.5 - A Aplicação Gráfica em Sistemas Multiprocessadores.

A medida que cresce o número de equipamentos multiprocessadores disponíveis no mercado, cresce o interesse de adaptar os algoritmos de geração de imagem para estas máquinas. Nesta adaptação surgem alguns problemas que são próprios do multiprocessamento genérico e outros que são específicos do multiprocessamento da geração de imagem. Em [Whit 90] são destacadas as seguintes questões:

- Utilização de Processador;
- Equilíbrio de Carga de Processamento;
- Comunicação entre Processos;
- Uso da Coerência;
- Decomposição do algoritmo;
- Escalabilidade;
- Compromisso Tamanho da Memória x Tempo de Acesso; e
- Programabilidade.

A seguir todas estas questões são apresentadas em maior detalhe.

O Equilíbrio de Carga de Processamento e a Utilização de Processador.

A Utilização de Processador é um conceito que se refere ao uso do tempo de computação num processador sobre o tempo total do algoritmo. Ele se aplica a cada processador, uma vez que alguns processadores podem estar bastante ocupados na execução do algoritmo enquanto que outros estão totalmente sem uso. O equilíbrio de carga de processamento se refere a idéia de que cada processador deve ser usado efetivamente como os seus vizinhos. Um sistema pode ter a sua carga de processamento equilibrada, mas os processadores podem continuar sub-utilizados se muito tempo é gasto na comunicação entre eles. Em geral, é desejável ter a máxima utilização de cada processador. Ambos, Utilização e Equilíbrio de carga de processamento, dependem da decomposição do algoritmo na rede de processadores.

O Problema de Comunicação entre Processos.

A comunicação entre os processos é um problema básico do processamento paralelo genérico. Este problema aparece, principalmente, quando um processador precisa do resultado de um processo que esta rodando em outro processador. São necessários mecanismos de sincronização, de roteamento e topologias especiais para a interligação entre os diversos processadores. Em síntese de imagem em paralelo algumas tarefas podem requerer pouca comunicação entre os processadores. Como exemplo destas tem-se as tarefas de Transformação e de Recorte. Para estas cada elemento gráfico precisa ser processado numa destas etapas para ser enviado para a seguinte. Assim sendo, uma topologia do tipo *pipeline* é suficiente. Outras tarefas, entretanto, podem requerer uma comunicação muito intensa. Por exemplo, as tarefas de Visualização e de

Iluminação podem requerer muita comunicação para operar todas as interações espaciais existentes entre diversos objetos numa cena.

O Problema de Coerência.

Um recurso muito utilizado na implementação de algoritmos de visualização é o Aproveitamento das Propriedades de Coerência existentes no modelamento de uma cena a ser exibida. [SSSc 74] lista uma série de propriedades de coerência que um algoritmo pode usar na solução do problema de Eliminação de superfície escondida (HSE). As propriedades básicas da coerência, para este caso, são:

> *Scan-line to scan-line coherence* - linhas adjacentes interceptam aproximadamente os mesmos conjuntos de bordas que são assim atualizadas incrementalmente de uma linha para outra.
>
> *Area Coherence* - um *pixel*, uma linha ou uma área e seus vizinhos tendem a ter os cálculos de tonalização e de visualização semelhantes.
>
> *Pixel to pixel coherence* - para interpolação dentro de uma linha (Vertical ou horizontal), um *pixel* deve ter uma pequena diferença no seu valor dos valores de seus vizinhos, a menos de fronteiras de objetos.

Na implementação destes algoritmos em máquinas sequenciais monoprocessadoras, esta utilização é facilitada pois o mesmo processador pode reconhecer a existência de coerência e dela aproveitar para aumentar a eficiência do processamento. Num processamento gráfico em paralelo, o aproveitamento da coerência pode ser diminuido pois, por exemplo, uma área de valores coerentes pode estar sendo tratada por processadores diferentes. Assim sendo, o grau de aproveitamento de coerência decresce a medida que o número de processadores num sistema aumenta. Diversos trabalhos pesquisam se o uso da coerência pode ser mantido na implementação em paralelo dos algoritmos, e se for, em que grau. [GrPa 89b] explora a coerência dos dados juntamente com técnicas de *Caching* para tratar grandes bancos de dados de imagens em sistemas de

processamento distribuido com memória limitada na implementação de algoritmo de *Ray-tracing*.

Decomposição

Em programação paralela dá-se o nome de Decomposição à tarefa de dividir o programa (dados e/ou rotinas funcionais) entre os diversos processadores disponíveis. Em função do que será dividido entre os processadores, existem três tipos de decomposição:

- **Decomposição dos dados**, espacial ou geométrica
- **Decomposição funcional**, temporal ou algoritmica; e
- a combinação dos dois anteriores

Em função do instante da divisão, estas decomposições podem ser feitas:

- **Estaticamente**, quando a decomposição é feita em tempo de compilação. Considere o caso de decomposição funcional. Neste caso o programa é dividido estaticamente em tarefas separadas para serem designadas aos processadores. Uma vez designado o processador, as tarefas não são mais dividas. Em geral neste método o número de tarefas é igual ao de processadores.

- **Parcialmente Estática e Dinâmica**, quando a decomposição é feita parcialmente durante a compilação e parcialmente durante o processamento. Considere, novamente, o caso de decomposição funcional. Neste método um certo número de tarefas é formado durante a compilação e elas são dinamicamente designadas ao processadores por um esquema de *scheduling* durante a execução. Isto permite uma melhor utilização dos processadores e suporta o equilíbrio da carga de uma forma direta. Neste método um número maior de tarefas pode ser designadas do que na forma estática e é, geralmente, bem maior que o número de processadores.

- **Dinâmicamente**, quando a decomposição é feita integralmente durante o processamento. Considere mais uma vez o caso de decomposição funcional. Neste método de decomposição as tarefas são definidas dinamicamente e são designadas aos processadores a medida que eles vão ficando livres. Em geral o número de tarefas é maior que o de processadores, como na forma anterior, mas com a diferença que as tarefas não são conhecidas até o tempo de execução. A flexibilidade é

maior que a forma Estática-Dinâmica. Este método é também chamado de "*Processor Farm*".

Na definição destes três métodos de decomposição foi utilizado o exemplo de decomposição funcional por facilidade de redação, entretanto, os mesmos métodos se aplicam à decomposição de dados.

Na **Decomposição dos Dados**, cópias do mesmo processo são executados em cada processador, cada um deles computando dados diferentes. Em computação gráfica, isto é feito usando Decomposição no Espaço de Imagem, onde cada processador é associado a uma parte separada da imagem, como mostrado na figura 4.12a ou usando Decomposição no Espaço de Objetos, onde cada processador é associado a diferentes conjuntos de objetos, como mostrado na figura 4.12b.

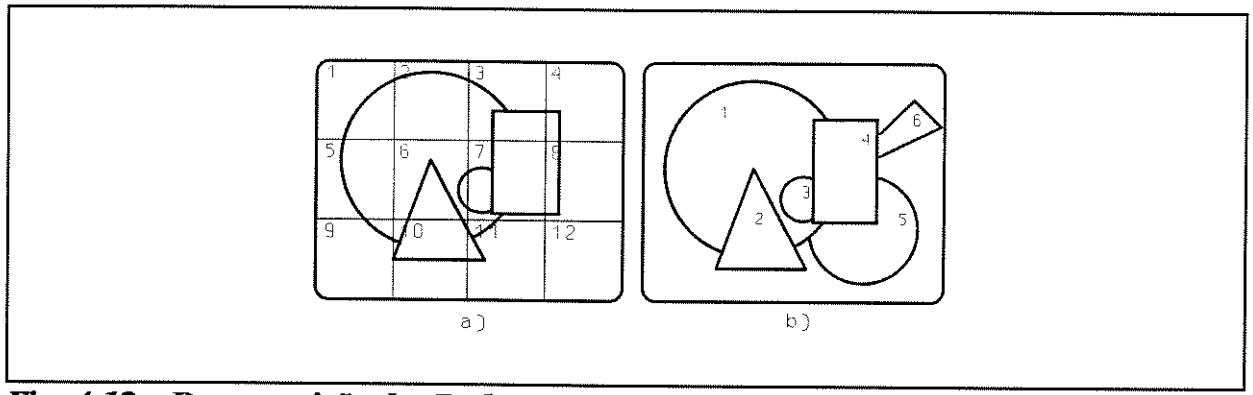


Fig. 4.12 - Decomposição dos Dados: a) no Espaço de Imagem; b) no Espaço de Objetos.

A Decomposição dos Dados no Espaço de Imagem, a mais comumente usada, requer o mínimo de comunicação entre os processos, mas a Coerência é perdida e alguns processadores podem ter mais trabalho a fazer que outros. Por exemplo, numa decomposição estática como na figura 4.12a o processador 6 tem que desenhar 3 objetos, enquanto o processador 12 não faz nada, este é um exemplo do Problema de Equilíbrio de Carga de Processamento (*Load-balance*).

Os métodos de Decomposição Estática-Dinâmica e Dinâmica, onde o Espaço da Imagem é dividido num número de pedaços maior que o número de processadores, reduzem o problema de equilíbrio de carga pois os processadores requerem serviço a medida que forem ficando desocupados. Entretanto eles reduzem mais ainda a utilização da coerência e aumenta a necessidade de comunicação entre os processadores.

A Decomposição de Dados no Espaço de Objetos funciona bem para tarefas que requerem pouca comunicação entre os processadores, como por exemplo as tarefas de Operação de Transformações e de Recorte. Já para a tarefa de

Eliminação de Superfícies Escondidas, ela requer muita comunicação entre os processadores para determinar se um objeto esconde outro. Uma excessão é o algoritmo *Z-Buffer*, que é relativamente fácil de se implementar em máquinas paralelas usando uma decomposição de dados no Espaço de Objetos.

Na **Decomposição Funcional**, cada processador executa um processo diferente, e os dados fluem entre eles. Por exemplo, num sistema gráfico um processador pode tratar as transformações, outro pode executar os recortes, um terceiro pode eliminar as Superfícies Escondidas, e assim por diante. A decomposição funcional é especialmente útil quando os diversos processadores têm diferentes capacidades. Quando as funções são dispostas em linha, como na figura 4.13, a decomposição é chamada *Pipelining*.

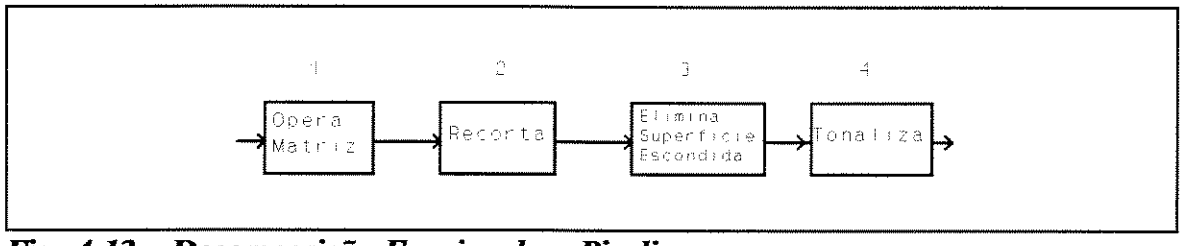


Fig. 4.13 - Decomposição Funcional = Pipeline

A Decomposição Funcional requer mais sincronismo, pois o processador mais ocupado pode significar um "gargalo". Mas quando os requisitos de sincronismo podem ser determinados estaticamente (por exemplo, num *pipeline* de transformação), ou quando a granulação do processamento for grande (por exemplo, num sistema de *server-based window*) a decomposição funcional pode ser muito eficiente.

Escalabilidade

Um aspecto que é lembrado por [Whit 90] é a habilidade dos algoritmos de funcionar num número variável de processadores. Normalmente os algoritmos são projetados para um conjunto de dimensões de multiprocessadores e são otimizados com o seu tamanho estático em mente. Entretanto devido a rápida redução dos custos dos microprocessadores, é possível que redes muito grandes de microprocessadores sejam construídas no futuro. Outros aspectos de Escalabilidade incluem aumento do número de elementos geométricos na cena e mudanças na resolução da tela.

Compromisso Tamanho da Memória x Tempo de Acesso

Em algumas situações, o programador tem de tomar uma decisão sobre usar referências locais ou referenciar memória num ponto remoto do sistema. Se

houver bastante memória local, é melhor armazenar (ou copiar) os dados na memória local. As vezes, devido ao tamanho da estrutura de dados ou da arquitetura da máquina, referências externas são necessárias. Este compromisso resulta em tempo adicional para completar a referência.

Programabilidade

Refere-se a facilidade de programação de um determinado algoritmo numa arquitetura alvo. Mesmo sendo este um aspecto subjetivo, ele requer atenção pois se um algoritmo é extremamente difícil de codificar e testar, isto pode inviabilizar a sua migração para uma arquitetura mais avançada.

4.6 - Exemplos estudados.

A seguir é feito um breve comentário sobre cada um dos exemplos de dispositivos dedicados e de arquiteturas especializadas consideradas na preparação deste trabalho.

[DWSD 88] descreve um conjunto de circuitos que desenha triângulos muito rapidamente. Estes componentes são usados numa arquitetura *pipeline* que também trata a tonalização por Phong, implementa o *anti-aliasing* por super-amostragem e suporta o mapeamento de textura.

[Schn 88] e [ScCl 88] descrevem uma arquitetura baseada num conjunto de processadores de objetos organizados em *pipeline*. Estes processadores fazem o mapeamento dos objetos na cena e a interpolação dos vetores normal e de profundidade de cada objeto. A eliminação das superfícies escondidas está distribuída no *pipeline* de processamento.

[Thay 89] descreve os circuitos VLSI do *pipeline* de processamento gráfico do HP 9000 TurboSRX.

[Clar 82] descreve a arquitetura do sistema IRIS da Silicon Graphics, os diversos usos dos "Engenhos Geométricos" (os *chips* de processamento geométrico) e como eles podem ser integrados ao *pipeline* de processamento.

[GKBh 89] e [KoMa 89] descrevem o processador gráfico integrado no circuito integrado i860 da Intel Co.

[Agat 86] descreve pela primeira vez o circuito integrado - MAGIC. Trata-se de um circuito integrado de propósito geral com ênfase em aplicações em computação gráfica. Nesta referência é feito um esboço de uma arquitetura aplicando este componente. Uma segunda versão deste componente - MAGIC II é descrita em [Finc 87] onde é esboçado um maior número de projetos de aplicação.

[FuPo 81] apresenta pela primeira vez um circuito VLSI para subsistema de exibição de acordo com a arquitetura chamada de PIXEL-PLANE. O trabalho em torno desta arquitetura continua em [FGHS 85] com o sistema PIXEL-PLANE 4 e em [Fuch 89] com a versão do sistema - PIXEL-PLANE 5.

[FuRa 82] descreve uma arquitetura baseada num circuito VLSI especialmente projetado para ela. Esta arquitetura é otimizada para o tratamento da tonalização de polígonos em tempo-real.

[Stae 86] descreve um circuito VLSI controlador de varredura para subsistemas de exibição.

[DWLG 92] descreve o Image Chip projetado para exibir *pixels* com informações de cor, profundidade, e informações para o tratamento *antialiasing* a cada ciclo de relógio. Segundo os autores uma arquitetura baseada neste *chip* é capaz de tratar primitivas triangulares e vetoriais em tempo real.

O MVP é um *chip* com características multiprocessadoras para Multimídia, descrito em [GGAk 92]. O MVP é utilizado para acelerar requisitos pesados de processamento gráfico e de imagem, característicos dos aplicativos para "Multimídia". Ele combina num único *chip*, diversos processadores totalmente programáveis com *streams* de dados conectados via um *crossbar* a uma memória compartilhada.

[PEMF 92] descreve o *chip* chamado *PixelFlow Enhanced Memory*, este realiza um conjunto de operações de *shading, texturing* e de *antialising*. Esta referência contêm também uma boa descrição do problema da transferência para a memória de quadro.

[KoBh 92] apresenta um conjunto de *chips*, ASIC's (*Application-Specific Integrated Circuits*), e processadores programáveis de imagem que segundo os autores indicam as tendências para a aceleração de processamento de imagem.

[GSSu 81] descreve uma arquitetura baseada no uso de *chips* de memória inteligentes que suportam *anti-aliasing*, *bitblt* e outras tarefas que podem ser tratadas na operação de escrita na memória de vídeo.

[Tani 83] descreve uma arquitetura em pirâmide que faz o suporte em *hardware* para uma visão hierárquica de um banco de dados.

[NIMT 84] descreve um sistema que permite o processamento paralelo de *scan-lines* usando vários processadores de *scan-line*.

[GhPo 85] apresenta uma arquitetura chamada de SUPER-BUFFER. Esta arquitetura se aplica a diversas aplicações gráficas. É um sistema de *Scan-line Virtual* construído em torno de um VLSI dedicado, chamado de *Systolic Raster Graphics Engine*. Um sequência deste trabalho pode ser acompanhada em [Ghar 88].

[Kauf 86] apresenta uma memória de quadro baseada em elementos de volumes - VOXEL, chamada de *Voxel Frame-Buffer*. Este tipo de memória pode ser muito bom para a exibição de informações volumétricas, como por exemplo, algumas obtidas por um sistema de tomografia médica. Entretanto requer muita memória para armazenar os Voxels (n^3, onde n é a resolução). O trabalho do prof. Kaufman pode ser acompanhado ainda nas referências [Kauf 87] e [KaBa 88].

[PoHo 89] descreve a arquitetura PIXEL-MACHINE da AT&T. Trata-se de um sistema paralelo de *chips* DSP com um *Frame-buffer* distribuido. Esta referência descreve como as imagens são exibidas e faz comparações com o PIXEL-PLANE de Fuchs, com a *Connection Machine* e com os chamados sistemas de *Systolic Array*.

[Suth 86] descreve uma arquitetura baseada num *Frame-buffer* profundo (com vários *bits* por *pixel*) e uma lista de *display*. Esta referência discute ainda diversos esquemas de conexão entre processadores em arranjos 3x3 ou 4x4.

[ScSt 93] apresenta uma arquitetura em *pipeline* que utiliza um processador de lista que implementa o algoritmo EXACT. Este algoritmo resolve o problema do surgimento de *pixels* errados na interação de superfícies no processamento de eliminação de superfícies escondidas no nível de sub*pixel*. Trata-se de um aperfeiçoamento do algoritmo A-buffer. Este projeto foi desenvolvido na Universidade de Tubingen - Alemanha.

[MEPo 92] descreve a arquitetura PixelFlow, um sistema gráfico experimental projetado para demonstrar as vantagens das arquiteturas com *image-composing* e para servir como uma plataforma de pesquisas para algoritmos e aplicações em computação gráfica em 3D. Os autores estão vinculados à Universidade de Carolina do Norte - USA.

[KWGo 92] apresenta um acelerador de *rendering*, baseado num algoritmo *scanline* modificado. Ele trata diversos *scanlines* em paralelo com alta eficiência e é otimizada para integrar sistemas que suportam altas taxas de dados (como por exemplo sistemas de vídeo). É dada grande ênfase ao baixo custo deste sistema. O projeto foi desenvolvido por uma equipe da Apple Computer Inc. USA.

[DeNe 93] descreve um acelerador gráfico (Leo) de alto desempenho. Ele suporta o tratamento de triângulos sombreados e linhas com *antialising*. O baixo custo da arquitetura é destacado pelos autores. O sistema Leo foi desenvolvido por uma equipe da Sun Microsystems Computer Corporation - USA.

Os princípios de projeto de um acelerador para o visualização de volumes é descrito em [Knit 93]. É descrito um conjunto de circuitos para diversas etapas da visualização de volumes. Projeto desenvolvido na Universidade de Tubingen - Alemanha.

O sistema RealityEngine é descrito em [Akel 93]. Trata-se de um acelerador gráfico dividido em 3, 4 ou 6 placas instaláveis em estações baseadas na tecnologia MIPS RISC. O autor é filiado a Silicon Graphics Computer Systems - USA.

[GrPa 88] e [GrPa 89a] descrevem como tratar *ray-tracing* para uma grande quantidade de dados e ainda ter uma velocidade razoável. É usada uma subdivisão do espaço em *octree*, um arranjo de processadores do tipo piramidal para o gerenciamento das tarefas e uma forma de armazenar os objetos para os teste de interação chamado de LRU (*Least Recently Used*).

[GrPa 89b] introduz o uso da coerência chamada de *Data-coherence* no tratamento de *ray-tracing* de uma grande quantidade de dados por um sistema de processamento distribuído com memória limitada.

[DiSw 84] faz um estudo sobre a implementação de *raytracing* numa máquina vetorial.

[ThPa 87] Mostra como a tarefa de transformação de polígonos e a eliminação de superfícies escondidas pode ser implementada aproveitando das características SIMD e MIMD da arquitetura DISPUTER. Também é feita uma breve descrição desta arquitetura.

4.7 - Resumo e Comentários

Este capítulo finaliza a parte de estudos preliminares à definição do trabalho proposto. Ele parte de uma discussão sobre o processo de geração de imagens com realismo, apresenta as formas de como este processo pode ser tratado em computadores e constata que este tratamento tem sido agilizado graças às tecnologias de circuito integrado e de multiprocessamento.

É feita uma breve introdução aos aspectos de multiprocessamento que mais se aplicam ao processo de geração de imagem. Destes são destacadas as técnicas de *pipeline* e processamento em paralelo. É feita então uma apresentação de como estas técnicas se integram ao processo de geração de imagem, com destaque ao processamento paralelo do *rastering,* e a arquiteturas voltadas a composição de imagens.

O capítulo reproduz ainda os problemas de aplicação gráfica em sistemas multiprocessadores levantados por Whitman [Whit 90].

O capítulo é encerrado comentando brevemente os trabalhos descritos na literatura que foram considerados na preparação deste trabalho. Neste conjunto de trabalhos estão incluídos exemplos de dispositivos VLSI, exemplos de arquiteturas dedicadas e exemplos de aplicações desenvolvidas em equipamentos multiprocessadores.

O estudo realizado neste capítulo foi em parte apresentado como tutorial convidado no XXIV Congresso Nacional de Informática em São Paulo em setembro de 1991 [Oliv 91].

Considerando a proposta de desenvolvimento de um posto de trabalho para aplicação em geração de imagem com realismo, neste ponto pode-se definir os princípios básicos do projeto a ser descrito nos capítulos seguintes.

Primeiramente foram separadas as características de caráter mais genérico deste posto. Foi considerado, que estas características deveriam ficar abrigadas num sistema de propósito geral. As características mais específicas de geração de imagem com realismo deveriam então constituir uma unidade separada. Foi

definido que o presente trabalho deveria se concentrar na definição desta unidade.

Esta unidade tem como objetivo principal a geração de imagens com realismo em alta velocidade. Outros objetivos de projeto que merecem destaque são:

- a **escalabilidade de processamento** — este objetivo considera que a estrutura não poderá estar presa a um número fixo de processadores. Deverá portanto permitir que a medida da disponibilidade, ou mesmo da necessidade de novos algoritmos, novos processadores possam ser incorporados a estrutura; e,

- a **independência do algoritmo de tratamento** — este objetivo procura evitar o desenvolvimento de circuitos processadores dedicados a algoritmos específicos, que podem agilizar em muito determinado tipo de aplicação mas que não beneficia nenhuma outra.

O capítulo a seguir apresenta a descrição do projeto proposto.

Capítulo 5

Projeto de uma Unidade Processadora para Geração de Imagens com Realismo.

Nos capítulos anteriores foram enunciados os principais aspectos envolvidos na Síntese de Imagens Realistas. A proposta deste trabalho focaliza principalmente os aspectos de arquitetura e circuitos dedicados a este tema. Seguramente, tal proposta exige o apoio de uma sistemática de projeto devido à sua própria complexidade. É importante salientar que o processo de escolha da sistemática, bem como a própria sistemática, foram elementos fundamentais para o desenvolvimento deste trabalho.

Para este projeto adotou-se uma sistemática do tipo *top-down*, onde um projeto de um sistema computacional é composto pelas seguintes etapas:

Requisitos do Projeto, quando são discutidas todas as considerações preliminares do projeto, os objetivos e as sugestões de implementação;

Especificação das Características, com base nos requisitos de projeto é feita a especificação das características do sistema a ser implementado. Quando nesta especificação o circuito é subdividido em diversos módulos, cada um deles deve ter as suas características especificadas. Inicialmente esta especificação é feita de forma textual livre, ou seja, com a apresentação de características e com suas justificativas textuais. Esta especificação é então formalizada em uma linguagem de descrição de *hardware* (HDL) [Arms 89],[Perr 91]. A adoção de uma HDL permite não só a especificação formal (sem ambiguidade e redundâncias) do sistema a ser implementado, como também a validação das proposições através de simulações e a síntese automática dos circuitos necessários, em função das ferramentas de desenvolvimento disponíveis;

Implementação, tomando como base as Especificações das Características, o sistema é então implementado. Se a implementação partir da especificação formal pode-se chegar à síntese automática do sistema. Neste caso, a distinção desta etapa com a anterior é reduzida.

Este capítulo apresenta então uma proposta de uma Unidade Processadora para Geração de Imagens com Realismo dentro da sistemática de projeto escolhida e delineada acima. A seção 5.2 descreve todos os itens relativos aos requisitos considerados no desenvolvimento do projeto. A seção 5.3 detalha textualmente a especificação dos módulos relativos a unidade de processamento gráfico (UGR). E por último, a seção 5.4 discute a solução proposta à luz dos requisitos e necessidades da aplicação.

5.1 - Requisitos para o Projeto.

A discussão dos requisitos do *hardware* de uma unidade de processamento gráfico começa pela definição dos requisitos globais do projeto. Para que nesta discussão cada requisito seja referenciado mais facilmente, adota-se a notação de cada requisito global receber um número. A medida que a discussão de um requisito se aprofunda, outros novos sub-requisitos são enunciados. Para identificar o relacionamento hierárquico entre os requisitos, um novo requisito é identificado pelo número do requisito de mais alta hierarquia ao qual está relacionado, separado de um segundo número por uma barra. Dentro deste novo requisito outros sub-requisitos podem ser definidos, e a sua identificação é feita de forma similar. Assim, por exemplo, o código RQ3/2/1 identifica um requisito específico (1) que está relacionado com o requisito global número 3 e com o seu sub-requisito número 2. A discussão dos requisitos feita nesta forma hierárquica pode envolver aspectos específicos do projeto. Estes aspectos são alinhados hierarquicamente de forma a permitir uma opção gerencial da estratégia de desenvolvimento do projeto. A figura 5.3 oferece uma visualização global dos requisitos desejados do projeto.

5.1.1 - Requisitos Globais de uma Unidade de Processamento Gráfico.

A unidade processadora gráfica — UGR (Unidade Gráfica com Realismo) — será projetada para ser conectada a uma unidade de processamento geral — UPG (Unidade de Propósito Geral) — formando assim um posto de trabalho com alto desempenho em computação de imagem. Uma visão global do posto de trabalho proposto é dada na figura 5.1.

Além do suporte àquelas operações próprias da aplicação Computação de Imagens, a UPG deve permitir um eficiente suporte a outras aplicações que envolvem processamento massivo de dados. Sendo assim se faz sentir a necessidade de uma conexão com um processador de propósito geral veloz

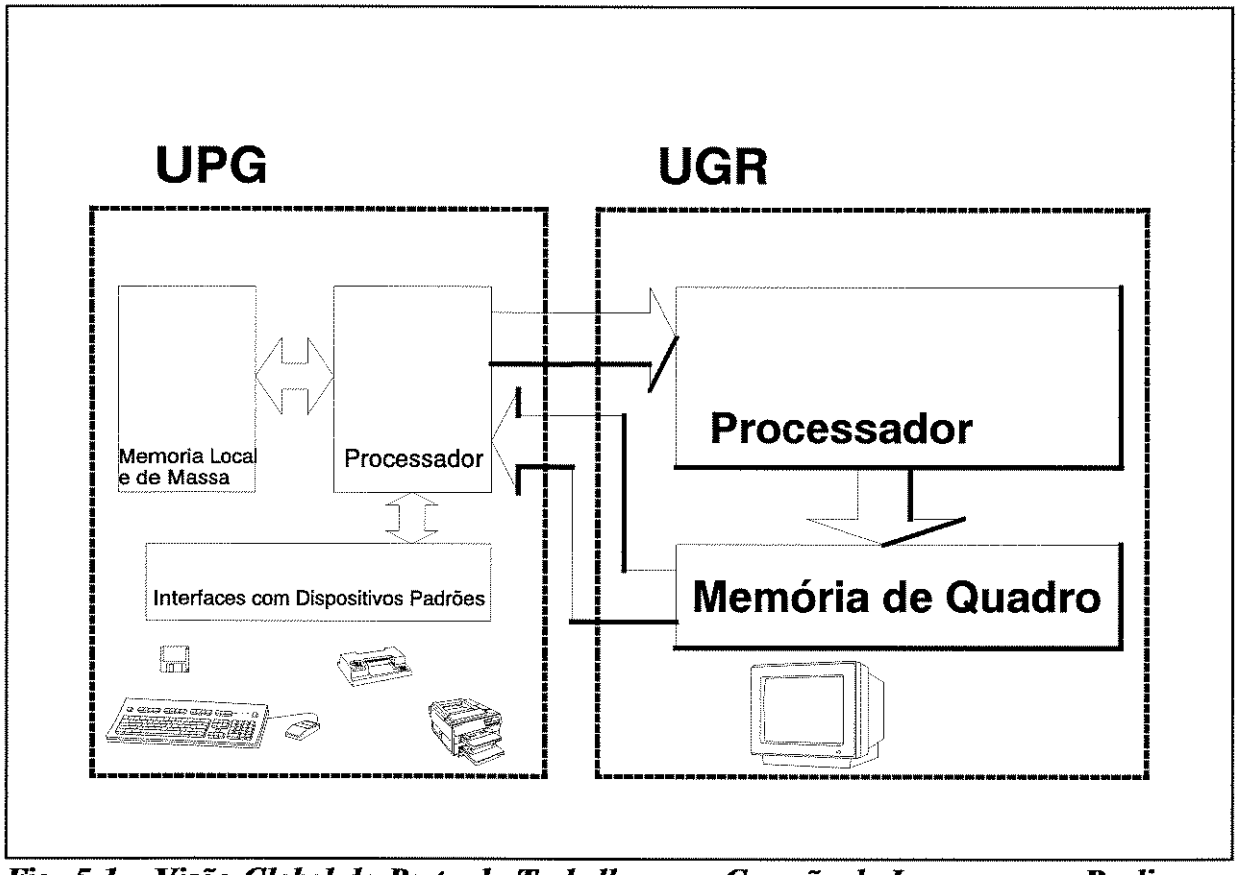


Fig. 5.1 - Visão Global do Posto de Trabalho para Geração de Imagens com Realismo.

(**RQ1 - Alta Velocidade de Processamento para Propósito Geral**).

O projeto da unidade UPG não é objeto deste trabalho, entretanto pode-se definir que esta unidade deverá permitir a ligação dos mais diversos dispositivos de suporte ao processamento de propósito geral, dentre os quais destacam-se: interfaces para teclado, mouse, impressoras, CDROM's e outros dispositivos de entrada/saída; alta capacidade de armazenamento (memória local e memória de massa) (**RQ1/1 - Ligação a Dispositivos Padrões**); interface para conexão a rede local (**RQ1/2 - Conectividade a Rede Local**). Outro requisito que se impõe com respeito a UPG é a disponibilidade de recursos de programação e operação (**RQ1/3 - Ambiente Operacional**).

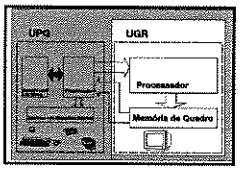
A ligação UPG - UGR.

A ligação UPG - UGR é feita através de um barramento que permite a UPG informar a UGR os dados que descrevem a imagem a ser gerada, bem como transferir o conteúdo da memória de quadro para o processamento do *bit-map* gerado, como por exemplo a gravação em CD's da imagem gerada. Assim sendo, destaca-se aqui o requisito mais importante que este trabalho impõe

sobre a UPG que é a necessidade de que esta seja uma unidade "aberta", no sentido de que permita a conexão de subsistemas de interfaces externas, ou seja disponha de um barramento padrão (**RQ1/4 - Barramento Padrão**).

O formato dos dados que descrevem a imagem não impõe nenhum requisito sobre o projeto *hardware* da unidade, ou seja, este deverá permitir qualquer formato de descrição da imagem.

5.1.2 - Requisitos específicos para o projeto de uma Unidade Gráfica com Realismo — UGR



Este trabalho tem como alvo principal o projeto de uma unidade de processamento gráfico que permita a geração de imagens com realismo e alta velocidade (**RQ2 - Alta Capacidade Gráfica**). Os itens que caracterizam uma boa qualidade de imagem são: — Alta Resolução na imagem (**RQ 2/1**) e Alta Definição de Cores (**RQ2/2**). Além da alta qualidade na imagem é imposta a necessidade de que o tratamento seja feito em Alta Velocidade de Tratamento da Exibição Gráfica (**RQ2/3**).

Por **alta resolução na imagem**, entenda-se uma quantidade entre 1024 e 2048 pontos na linha de varredura e um quadro com uma quantidade entre 1024 e 2048 linhas de varredura na tela. Neste trabalho foi considerado como tela de alta resolução aquela com 1280 (horizontal) x 1024 (vertical) pontos na tela (pixels).

Por **alta definição de cores**, entenda-se uma quantidade de cores presentes simultaneamente na tela acima de 256. A quantidade proposta para este projeto é 16 777 216 cores (2^{24} cores), pois com isto a estação possuirá um circuito de exibição equivalente às chamadas interfaces gráficas *True-color*. Além da quantidade de cores, é importante a possibilidade de opções de cores, ou seja a existência de uma palheta de cores(**RQ2/2/1**), pois desta forma será possível fazer a correção *gamma* das cores.

O requisito **alta velocidade** nas operações gráficas é mais difícil de ser quantificado, por não existir um parâmetro objetivo que defina a velocidade de operações gráficas, permitindo assim a comparação entre diversas opções de engenhos gráficos. O que se observa no mercado é que cada fabricante descreve o desempenho de seu produto na forma mais conveniente para aumentar os seus dados de desempenho. A forma mais comum de especificar o desempenho é descrevê-lo em termos de número de polígonos de tamanho fixo exibidos por segundo. Entretanto cada fabricante descreve o desempenho de seu produto

através de polígonos com formatos diferentes (triângulos, quadriláteros), com ou sem *rendering* (*Gouraud* ou *Phong shading*, *Z-buffer*, *Anti-Aliasing*) e de tamanhos diferentes (lados com oito ou dez pixels ou através da definição de área interna ao polígono em torno de 100 pixels).

Assim sendo, neste trabalho optou-se por definir o requisito de alta velocidade nas operações gráficas em termos de quantidade de polígonos tonalizados de acordo com Gouraud exibidos por segundo. Para tanto optou-se por medir este desempenho utilizando, como em [AkJe 89], polígonos quadriláteros com 10 pixels de lado e tonalizados de acordo com Gouraud (aqui chamados de Quadriláteros Akeley, ou simplesmente QA). Foi fixado como alvo do projeto a taxa de **300 000 QA por segundo**. Esta taxa pode ser interpretada como aquela necessária para exibir 30 quadros por segundo, formados por 10 000 QA's, ou seja, o suficiente para geração de imagens em tempo real para aplicação em animação.

Para atingir esta taxa é preciso considerar cada uma das etapas de síntese de imagens.

Supondo que uma cena a ser gerada é modelada através de uma lista de vértices de polígonos, um sub-sistema de processamento para esta geração pode ser implementado de acordo com o diagrama de blocos da figura 5.2.

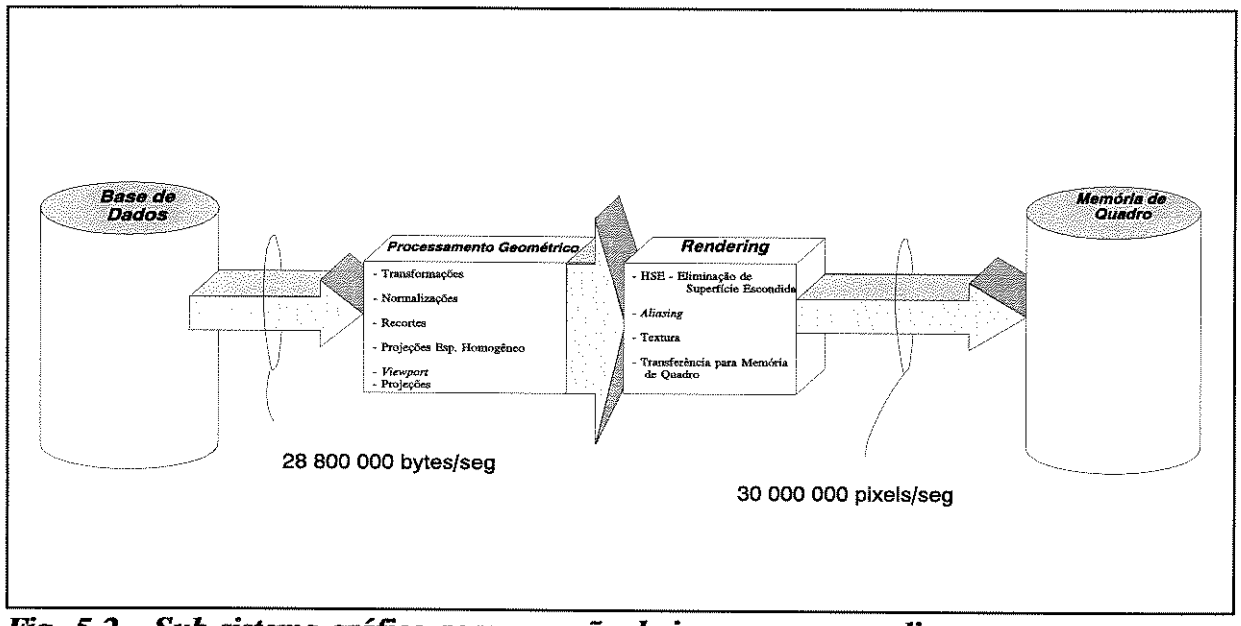


Fig. 5.2 - Sub-sistema gráfico para geração de imagens com realismo.

O processamento geométrico envolve, por exemplo: transformações tridimensionais dos vértices e dos vetores normais, normalizações, recortes triviais, projeção no espaço homogêneo, *viewport* e projeções na tela.

Por outro lado, o processamento do *rendering* envolve as tarefas de Remoção de Superfície escondida, tratamento de *aliasing* e de textura e a transferência para a memória de quadro.

Assim sendo, para o detalhamento quantitativo do requisito de alta velocidade foram considerados os seguintes itens:

- Taxa de transferência para o sub-sistema gráfico;
- Taxa de operações em ponto-flutuante, e
- Taxa de transferência para a memória de quadros.

(RQ2/3/1) - Taxa de transferência para o sub-sistema gráfico. Esta taxa representa a retirada da descrição da base de dados pelo sistema processador. Considerando uma base de dados que descreve a imagem por lista de vértices e que cada QA é descrito por 4 vértices, para atingir a taxa de 300 000 QA por segundo na tela, desprezando inicialmente a degradação no processamento, o sistema processador deverá retirar da base de dados 1 200 000 vértices por segundo. Como para cada vértice são necessários 6 valores em ponto-flutuante com 32 bits (3 para definir sua posição *x, y e z* e 3 para definir o vetor normal ao vértice - *nx, ny e nz* [Gour 71a]), a taxa de transferência para o sub-sistema gráfico deve ser de:

$$1.200.000[transf/seg] \times 6[valor/transf] \times 4[bytes/valor] = 28.800.000[bytes/seg] \quad \textbf{(5.1)}$$

(RQ2/3/2) - Taxa de operação em ponto flutuante. As operações que mais contribuem para a determinação desta taxa são:

- Transformação dos vértices em coordenadas homogêneas,
- Transformação das normais,
- Normalização dos vetores normais,
- Recorte, e
- Projeção.

A **transformação dos vértices tri-dimensionais** envolve operações matriciais em coordenadas homogêneas, ou seja matrizes 4x4. Isto significa que para cada vértice são necessárias até 16 multiplicações e 12 operações de adição o seja:

$$1.200.000[transf/seg] \times (16+12)[flop/transf] = 33.600.000 flops \quad \textbf{(5.2)}$$

Na **transformação dos vetores normais**, para as operações de iluminação é necessário transformar os vetores normais nos vértices . Estas operações podem ser realizadas por multiplicações de matrizes 3x3. Assim sendo são necessárias 9 multiplicações e 6 adições.

$$1.200.000[transf/seg] \times (9+6)[flop/transf] = 18.000.000 flops \quad \textbf{(5.3)}$$

Normalizações dos vetores normais. São necessárias 6 multiplicações, 2 adições e um cálculo de raiz quadrada.

$$1.200.000[transf/seg] \times (6+2+16)[flop/transf] = 28.800.000 flops \quad \textbf{(5.4)}$$

Recorte dos vértices. Aplicando o algoritmo de Sutherland-Hodgman a operação de recorte é essencialmente uma série de comparações com os planos de recorte (6 comparações em coordenadas tri-dimensionais).

$$1.200.000[transf/seg] \times 6[flop/transf] = 7.200.000 flops \quad \textbf{(5.5)}$$

Projeções no espaço homogêneo. Cada vértice é projetado no espaço homogêneo dividindo cada componente (x,y *e* z) pelo seu componente w. Primeiro é determinado o $1/w$ e depois este fator é multiplicado pelos componentes. Considerando que o cálculo do inverso envolve cerca de 8 operações em ponto flutuante, tem-se:

$$1.200.000[transf/seg] \times (8+3)[flop/transf] = 13.200.000 flops \quad \textbf{(5.6)}$$

Viewport e projeção na tela. Cada componente dos vértices pode ser operado individualmente por fatores de escala diferentes, deslocados por fatores de deslocamento diferentes e convertidos para coordenadas da tela. Isto requer 9 operações de multiplicação, ou seja:

$$1.200.000[transf/seg] \times (3+3+3)[flop/transf] = 10.800.000 flops \quad \textbf{(5.7)}$$

Conceitualmente estas tarefas podem ser executadas de forma simultânea, resultando numa taxa total de operações em ponto flutuante dado por:

Transformações de Vértices	33 600 000	
Transformações de Normais	18 000 000	
Normalizações de Vetores	28 800 000	
Recorte	7 200 000	
Projeção	13 200 000	
Viewport	10 800 000	
	111 600 000	flops

ou seja, seguramente, maior que 111 megaFLOPS.

Além destas tarefas acima, as tarefas relativas ao "*rendering*" da imagem devem ainda ser consideradas. Entretanto uma estimativa de desempenho feita *a priori* é muito difícil pois este desempenho depende muito do tipo de algoritmo a ser aplicado. A princípio, é deixado para o programador definir por *software* o tipo de algoritmo a ser utilizado. Entretanto algumas tarefas devem ser apoiadas por *hardware*, em particular as operações com os pixels na tela (*RasterOP ou PixelOP*), o tratamento de superfícies escondidas por *Z-buffer* (**RQ2/3/3**) e a composição ou mistura de cores por plano alfa (**RQ2/3/4**). Com estes recursos a execução destas tarefas é embutida na própria operação de transferência para a memória de quadro.

(RQ2/3/5) - Taxa de transferência para a memória de quadros.

A taxa de 300 000 QA por segundo, proposta na definição de alta velocidade de tratamento gráfico, corresponde a uma taxa de transferência para a memória de quadro de 300 000 × (10 × 10) pixels/segundo, ou seja, o circuito de memória deve ter permitir ciclos completos de acesso da ordem de 33 nanosegundos. Considerando que durante o acesso serão realizadas operações do tipo *rasterop* (em ciclos de *read-modify-write*, t_{rmw}), o período de 33ns para este tipo de operação é muito pequeno. Para manter a taxa de transferência utilizando memórias com tempo de acesso t_{rmw} maiores é considerado o acesso simultâneo a diversos pixels. A forma de permitir este acesso simultâneo é organizar a memória de forma a existir diversos blocos de memória com acessos independentes (**RQ2/3/5/1**).

5.1.3 - Conclusão dos Requisitos.

A figura 5.3 resume os requisitos considerados para definição do projeto de uma unidade processadora para aplicação em geração de imagens com realismo.

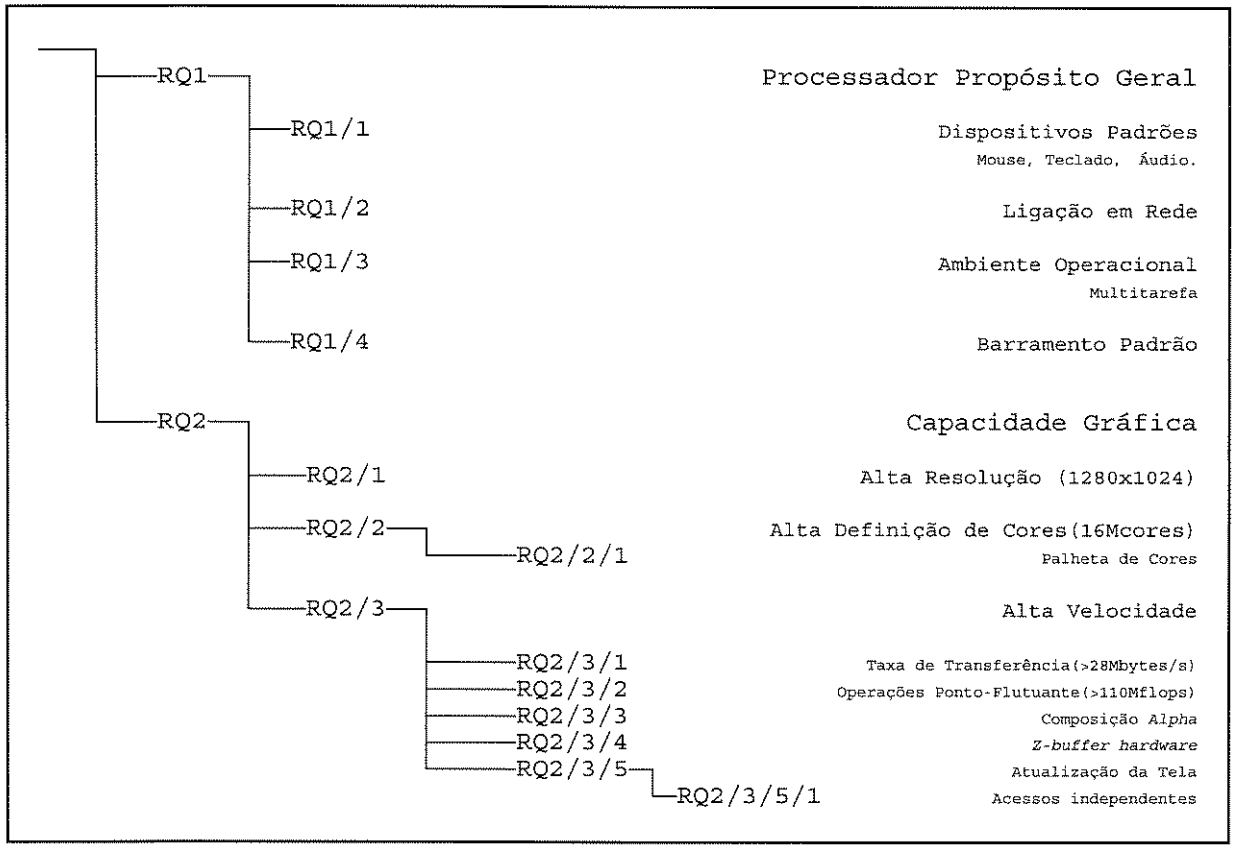


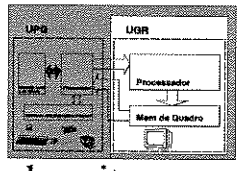
Fig. 5.3 - Resumo dos Requisitos

Foi considerado que um posto de trabalho para esta aplicação é formado por uma unidade processadora de propósito geral (UPG) e uma unidade de processamento gráfico (UGR).

No contexto deste trabalho foi desconsiderado o detalhamento dos requisitos de desenvolvimento da UPG, pois pretendia-se que estes fossem cobertos por equipamentos comerciais. Entretanto, os requisitos genéricos são descritos para que esta unidade possa integrar junto com a UGR um posto de trabalho para a geração de imagens com realismo. Dentre estes destaca-se a necessidade de um barramento padrão aberto. Durante os estudos para estes requisitos foram considerados os barramentos IOChannel do IBM-AT [IBM 84], VME [Sign 81] e o SBUS da Sun MicroSystems [FrBr 89].

Para a UGR, cada requisito foi considerado em detalhe. E é com base nestes requisitos que a seguir é especificada a arquitetura desta unidade.

5.2 - Especificação do *Hardware* da Unidade Gráfica — UGR.



A UGR é responsável pela geração e exibição de uma imagem descrita numa base de dados e fornecida pelo UPG. A geração da imagem, ou seja o processamento geométrico e de *rendering,* é feita num arranjo de processadores de alta velocidade. Este processamento produz os códigos da imagem que são armazenados na memória de quadro. Esta, além de conservar os códigos, executa operações elementares durante os acessos às posições de memória e envia para o monitor de vídeo os valores de cor de cada *pixel* da tela.

Para a definição da arquitetura da UGR foram considerados como princípios básicos do projeto:

- **A divisão do sistema em dois subsistemas — Arranjo de Processadores e Memória de Quadro** , a saber: O arranjo de processadores é responsável pelo processamento geométrico e de *rendering*; e a memória de quadro é o subsistema de exibição responsável pela manutenção da informação a ser exibida e pelos processamentos básicos de acesso às posições de memória (operações de *RasterOp*).

- **Independência da Estrutura dos Dados que descreve a imagem.** O projeto desta arquitetura não deve estar preso a nenhum formato da descrição da imagem. A estrutura de processamento proposta aqui permite a geração de imagem a partir de diversos formatos. É deixado para o *software* instalado neste arranjo o tratamento de cada novo formato.

- **Cada processador do arranjo tem acesso à descrição total da imagem de forma independente e simultânea aos acessos de outros processadores.** A descrição da imagem a ser gerada é enviada simultaneamente para todos os elementos processadores. Desta forma, durante o processamento cada um destes poderá ter acesso livre de concorrência a estrutura de dados que descreve a imagem. Este princípio define então que cada elemento processador deverá ter uma memória local que possa abrigar toda a descrição da imagem.

5.2.1 - Visão Geral da Unidade UGR

Uma visão geral da arquitetura da UGR é apresentada na Figura 5.4. A descrição da imagem a ser gerada é enviada pela UPG através do barramento COMUM para todas as memórias locais (ME*n*) dos processadores elementares (PE*n*). Estes processam estas informações gerando os valores dos pixels a serem enviados para a memória de quadros (MQ). Este envio é feito pela escrita numa interface local (F*n*). Os valores dos pixels são encaminhados para a MQ através da Via de Acessos Múltiplos (VAM). As ME's são memórias de dupla porta que de um lado recebem as descrições da imagem, e de outro permitem acesso pelo processador elementar.

A proposta da arquitetura da UGR tomou como base a estrutura de processamento de vídeo intercalada como descrita na seção 4.4.2 (Arquiteturas com Multiprocessadores no *Back-end*) do Capítulo 4. Aquela estrutura foi modificada de forma a permitir que cada processador possa ter acesso a qualquer pixel na tela e não somente àqueles aos quais esteja diretamente ligado. Esta flexibilidade de alocação processador-pixel permite que, em função da descrição da imagem, diversos processadores possam tratar em conjunto uma única região da tela, ou um mesmo conjunto de objetos. Para que este acesso possa ser feito os processadores foram separados da memória de vídeo. A interligação lógica dos processadores aos pixels passa a ser feita através de uma estrutura de comunicação com a Memória de Quadro que permite reconfiguração dinâmica por *Software*.

A seguir é feita a especificação de cada uma das partes da UGR. Esta especificação é realizada considerando os aspectos mais gerais da arquitetura. Para o presente trabalho, não foi definida a necessidade da especificação detalhada dos processadores PE, pois a proposta da arquitetura prevê a possibilidade de que em seu lugar possam ser anexados módulos processadores de prateleira ou mesmo módulos em desenvolvimento para processamento específico.

Nesta especificação é enfatizada a definição da Memória de Quadro e da Via de Acessos Múltiplos. A definição dos processadores elementares será realizada quando da implementação do sistema. Uma vez definido o processador, a implementação se resume ao desenvolvimento do circuito de interface daqueles processadores com a VAM, ou seja dos módulos F*n*. Quando da definição do processador, será possível também definir o barramento COMUM.

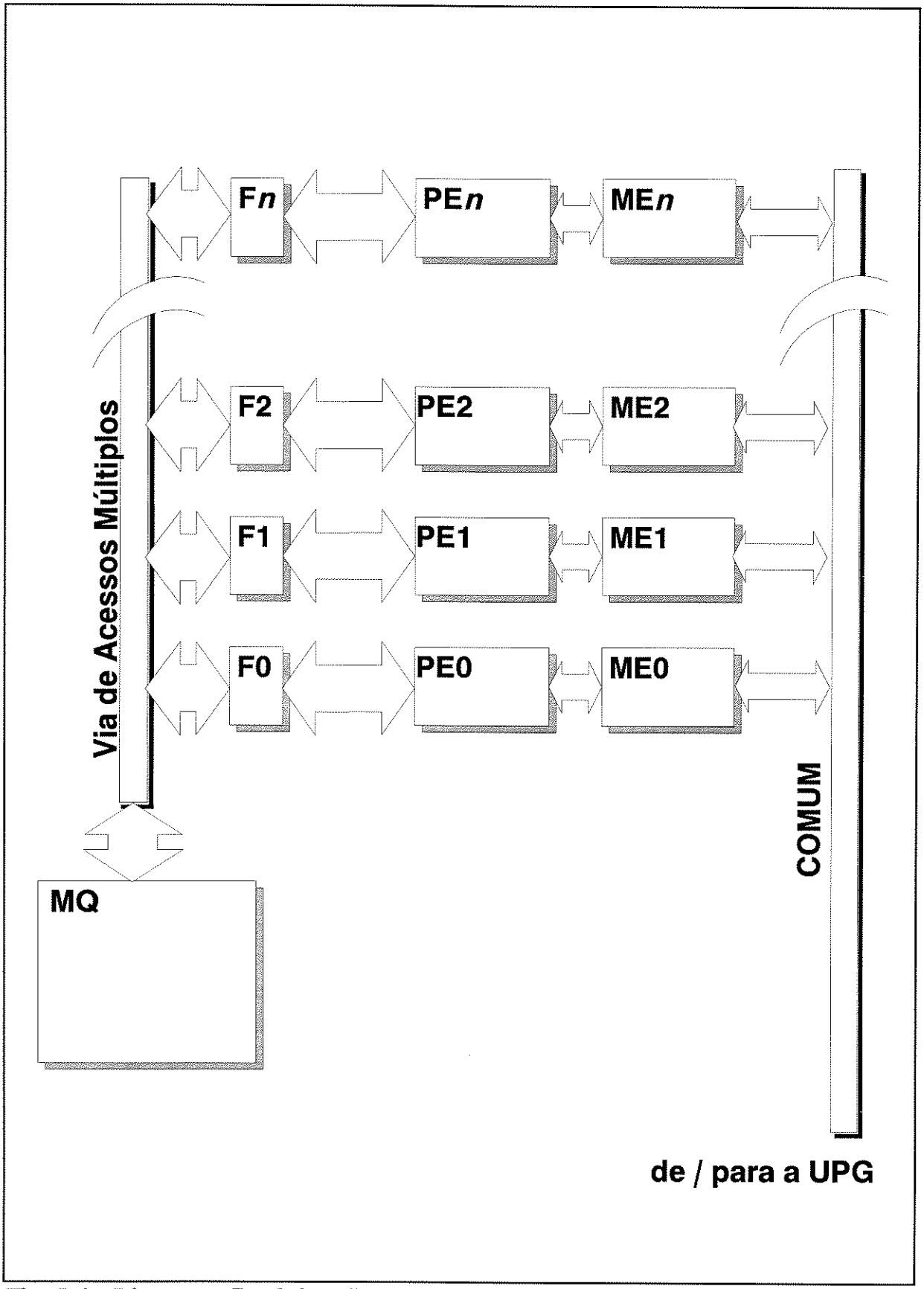
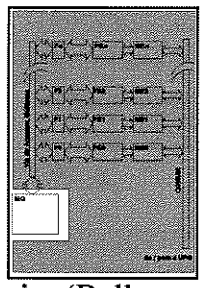


Fig. 5.4 - Diagrama Geral da UGR



5.2.2 - MQ/UGR - Memória de Quadro de Alta Resolução da UGR.

Além dos próprios circuitos de memória de vídeo, a Memória de Quadro da Unidade Gráfica (MQ/UGR) é composta pelos circuitos de interface com o monitor, da lógica de alta frequência (Palheta, deslocadores, osciladores e contadores), do controlador de vídeo, de processamento da escrita na memória de vídeo e da lógica de selecionamento dos elementos.

A organização interna deste módulo é melhor descrita com o auxílio do diagrama de blocos funcionais da figura 5.5.

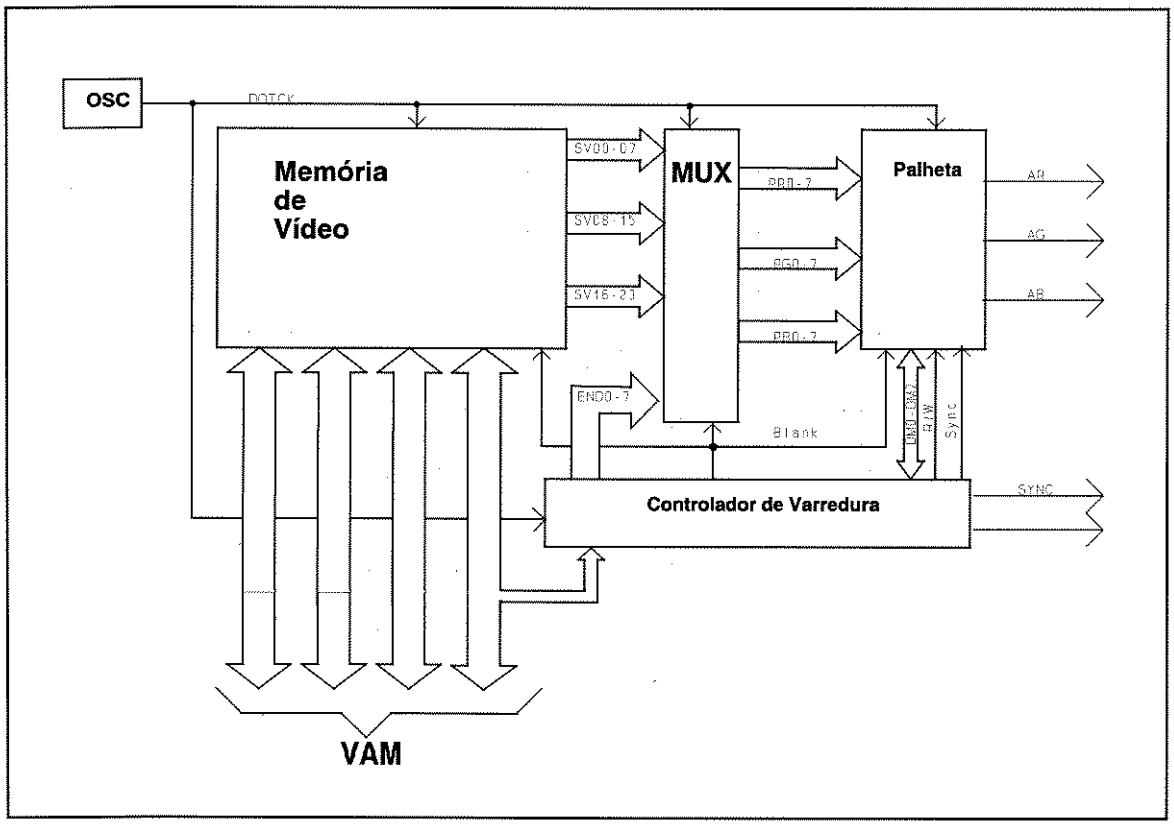


Fig. 5.5 - Diagrama de Blocos Funcionais da Memória de Quadro

Destes blocos destacam-se o da **Palheta** e o da **Memória de Vídeo**. A ligação entre estes dois blocos é feita através do bloco **MUX**. O bloco MUX permite o acesso aos registradores de cor da palheta. O controle do acesso ao registradores de cor e o controle da temporização de todo o circuito da memória de quadro estão concentrados no bloco **Controlador de Varredura**. A base de tempo do circuito de Memória de Quadro é gerada no bloco **Oscilador**.

5.2.2.1 - O circuito Palheta.

Para atender o requisito RQ2/2/1 a MQ/UGR utiliza em sua saída para o monitor um circuito de Palheta. O princípio de funcionamento de um circuito de Palheta foi descrito no capítulo 3 (seção 3.8.3 - Circuito Bit-mapped com Palheta). Resumindo, este circuito gera os sinais de vídeo de cada cor primária, vermelho, verde e azul para o monitor de vídeo. Na MQ/UGR a cada uma destas saídas existe uma LUT (*look-up table*) de 256 posições de oito bits e um conversor DAC de oito bits. As três LUT's podem ser atingidas simultaneamente para carregamento e para leitura de seus conteúdos, desta forma definem-se 256 registradores de cores com 24 bits cada.

As principais características do circuito Palheta são:

- 256 registradores de cores de 24 bits formados por 3 LUT's e um conversor digital/analógico (DAC) de 8 bits para cada cor primária (vermelho, verde e azul);
- entradas independentes nos DACs para soma com os sinais de sincronismo e de apagamento.

Os Registradores de Cores

O valor de um pixel a ser enviado para o monitor (sinais PR0-7, PG0-7 e PB0-7) chega ao circuito de palheta através do circuito de MUX. A cor associada a cada valor possível é programada pelo usuário através de escrita nos registradores de cores. Pode-se ter acesso a estes 256 registradores de 32 bits (dos quais somente 24 bits interessam) de acordo com a seguinte relação:

$$RegCor(n) = Endereço\ da\ palheta + n + 4 \qquad (5.8)$$

Ou seja, é possível atingir os registradores da palheta de cores e, através do endereçamento do número de cor, alterar os componentes das cores vermelho, verde e azul.

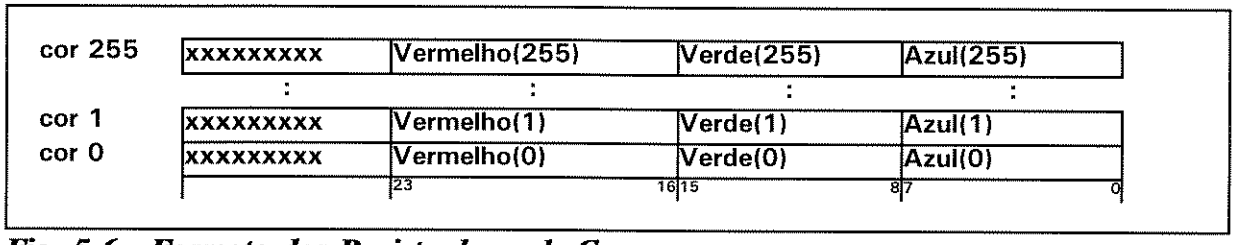


Fig. 5.6 - Formato dos Registradores de Cor.

Sinais para a Ligação com o monitor de Vídeo

Para permitir a ligação com monitor de vídeo na saída do circuito de Palheta existem os seguintes sinais:

Sinais AR e **AB** -Sinais correspondentes às cores primárias vermelha e azul. Trata-se de sinais analógicos que são enviados através de cabo com impedância de 75 ohms para o monitor de vídeo com níveis elétricos variando de um minímo de 0,5 Vpp até um máximo de 1,0 Vpp de acordo como o padrão RS170.

Sinal AG - Sinal correspondente à cor primária verde. Tem as mesmas características elétricas dos sinais acima. Entretanto, este sinal pode ser configurado para ser somado ao sinal de sincronismo composto.

Sinais de entrada no circuito de PALHETA.

Os principais sinais que entram no circuito de PALHETA são:

PR0-PR7, PG0-PG7 e PB0-PB7 - Estes 24 sinais formam o barramento de PIXEL. Este barramento é gerado no circuito de multiplexação de pixels - MUX. Este barramento associa 8 bits para cada LUT de cor primária. Durante o ciclo de escrita nos registradores de cores estes sinais fornecem o endereço do registrador ao qual se quer ter acesso.

DM0-DM7 - barramento de dados interno - O barramento de dados utiliza os intervalos de retraço para ler ou escrever nos registradores de cores.

DOTCK - O sinal de "*dot clock*", ou seja aquele que determina a taxa em que os PIXELs estarão entrando no circuito PALHETA.

BLANK - Sinal de apagamento, a ser somado com cada componente de cor primária.

SYNC - Sinal de sincronismo composto (horizontal e vertical), a ser somado com a componente de cor verde, no modo G-Composto.

R/W - Sinal de controle que indica uma operação de escrita nos registradores de cores do conteúdo em DM00-DM23.

5.2.2.2 - Circuito Multiplexador de Pixel - MUX

O circuito multiplexador de pixel (MUX) encaminha os valores de pixels, retirados da memória de vídeo, para o circuito de palheta no intervalo de tempo em que a tela deve ficar acesa. Quando ela deve ficar apagada, no intevalo

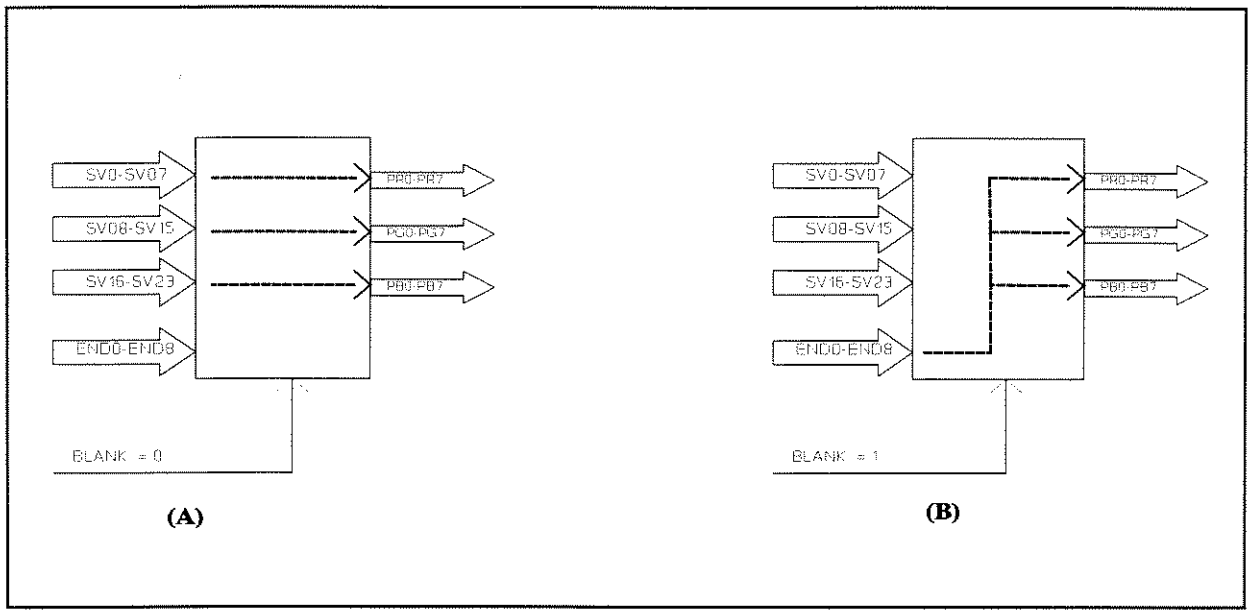


Fig. 5.7 - Multiplexador de Pixels : a) Intervalo tela acesa. b) Durante o retraço.

indicado pelo sinal BLANK, o circuito MUX permite o endereçamento dos registradores de cores.

No intervalo de tela acesa, os sinais SVO00..SVO07 correspondentes aos bits que definem a cor vermelha são ligados aos bits PR0..PR7, ou seja, à LUT R (cor vermelha). Os sinais SV08..SV15 são ligados aos bits PG0..PG7 (LUT G). E os sinais SV16..SV23 são ligados aos bits PB0..PB7 (LUT B).

Durante o intervalo de apagamento da tela, o circuito MUX permite o acesso aos registradores de cor da palheta. Para isto ele deve receber como entrada os bits de endereçamento de memória interno. Durante um acesso aos registradores de cor este circuito gera os sinais PR00..PR07, iguais aos bits PG00..PG07 e aos bits PB00..PB07, em função dos 8 bits menos significativos do barramento de endereço.

5.2.2.3 - A Memória de Vídeo.

Como visto no capítulo 3 (seção 3.6 - O Acesso do Processador Gráfico à Memória de Quadro) uma unidade de memória de vídeo deve atender dois tipos

de acessos concorrentemente. Um tipo é o acesso para *refreshing* da tela e o outro o acesso do processador para a atualização dos valores dos pixels na memória de quadro. Para a especificação da memória de vídeo utilizada na UGR primeiramente é considerado o acesso para *refreshing* para depois considerar o acesso de processamento.

Considerações sobre o Acesso para Refreshing.

Primeiramente é calculada a frequência de acessos para *refreshing*. Em função desta frequência é organizado o banco de memória de forma a compatibilizar o seu período com o tempo de acesso permitido pela memória.

Cálculo da frequência de acessos para *refreshing*.

A frequência de acessos para *refreshing* do monitor de vídeo é determinada pelo sinal DOTCK (*dot clock*). Para a determinação de frequência de DOTCK é considerada a seguinte relação:

$$DOTCK = (Cv+Crh) \times FH \qquad \textbf{(5.9)}$$

onde : Cv = número de pixels visíveis na linha
Crh = número de pixels invisíveis no retraço horizontal .
FH = frequência horizontal

Para 1280 pixels visíveis em uma linha e considerando a frequência horizontal do monitor a ser utilizado igual a 64 KHz, chega-se a seguinte relação:

$$DOTCK = (1280+Crh) \times 64 \times 10^3 Hz \qquad \textbf{(5.10)}$$

O número de pixels que corresponde ao retraço horizontal é determinado a partir do valor máximo de intervalo de retraço especificado pelo fabricante do monitor. O circuito permite que o valor deste intervalo seja definido como parâmetro de configuração da unidade. Como exemplo, para fins desta descrição serão considerados os valores do monitor TEK GMA303 da TEKTRONIC.

$$Crh \times \frac{1}{DOTCK} > 4{,}5mS \qquad \textbf{(5.11)}$$

Combinando com a relação (5.9) , tem-se:

$$Crh > 517 pixels \tag{5.12}$$

Ou seja para 1280 pixels visíveis com FH = 64 KHz o retraço deverá durar mais que o intervalo de 517 pixels, para encobrir o maior período de retraço definido pelo fabricante do monitor. Usando 532 pixels por retraço têm-se:

$$DOTCK = 116 MHz \tag{5.13}$$

A organização da memória de quadro para os acessos para o *refreshing*.

Para atender a demanda de enviar 116 000 000 pixels/seg para o circuito palheta utilizando memórias de vídeo VRAM, com frequência de deslocamento da ordem de 30 Mega pixels/seg, são necessários no mínimo 4 blocos de memória de vídeo. Considerando que as memórias de vídeo possuem, tipicamente, registradores de deslocamento com 256 bits, optou-se por organizar a memória de quadro em 5 blocos de memória em *interleaving*. Desta forma num ciclo de carregamento dos registradores de deslocamento da memória de quadro é carregada uma linha inteira (5 × 256 = 1280 pixels). Esta opção permite que a operação de carregamento dos registradores de deslocamento possa ser feita durante o intervalo de retraço horizontal, evitando assim o efeito de descontinuidade de exibição na tela durante os intervalos de carregamento. Uma vez carregados os registradores de deslocamento dos cinco blocos de memória, a linha de varredura na tela é formada lendo cada bloco de memória em sequência. O primeiro pixel da linha sai do bloco 0, o segundo pixel sai do bloco 1,..., o quinto pixel sai do bloco 4, o sexto pixel sai novamente do bloco 0 e assim por diante até terminar a linha de varredura, ou seja, até descarregar completamente os registradores de deslocamento dos cinco blocos, como mostrado na figura 5.8b. Para completar a resolução da tela é necessário que cada um dos cinco blocos da memória de quadro tenha 1024 linhas.

5.2.2.4 - Considerações sobre os acessos para processamento.

Considerando os ciclos de acesso das memórias de quadro disponível (da ordem de 260ns para T_{rmw}) se a memória de quadro dispusesse de somente uma via de acesso para todos os pixels seria possível acionar 3 846 153 [pixels/seg].

Para atingir uma taxa de transferência para a memória de quadro proposta como requisito RQ2/3/1 (29 360 128 [pixel/seg]) é necessário estabelecer uma via de

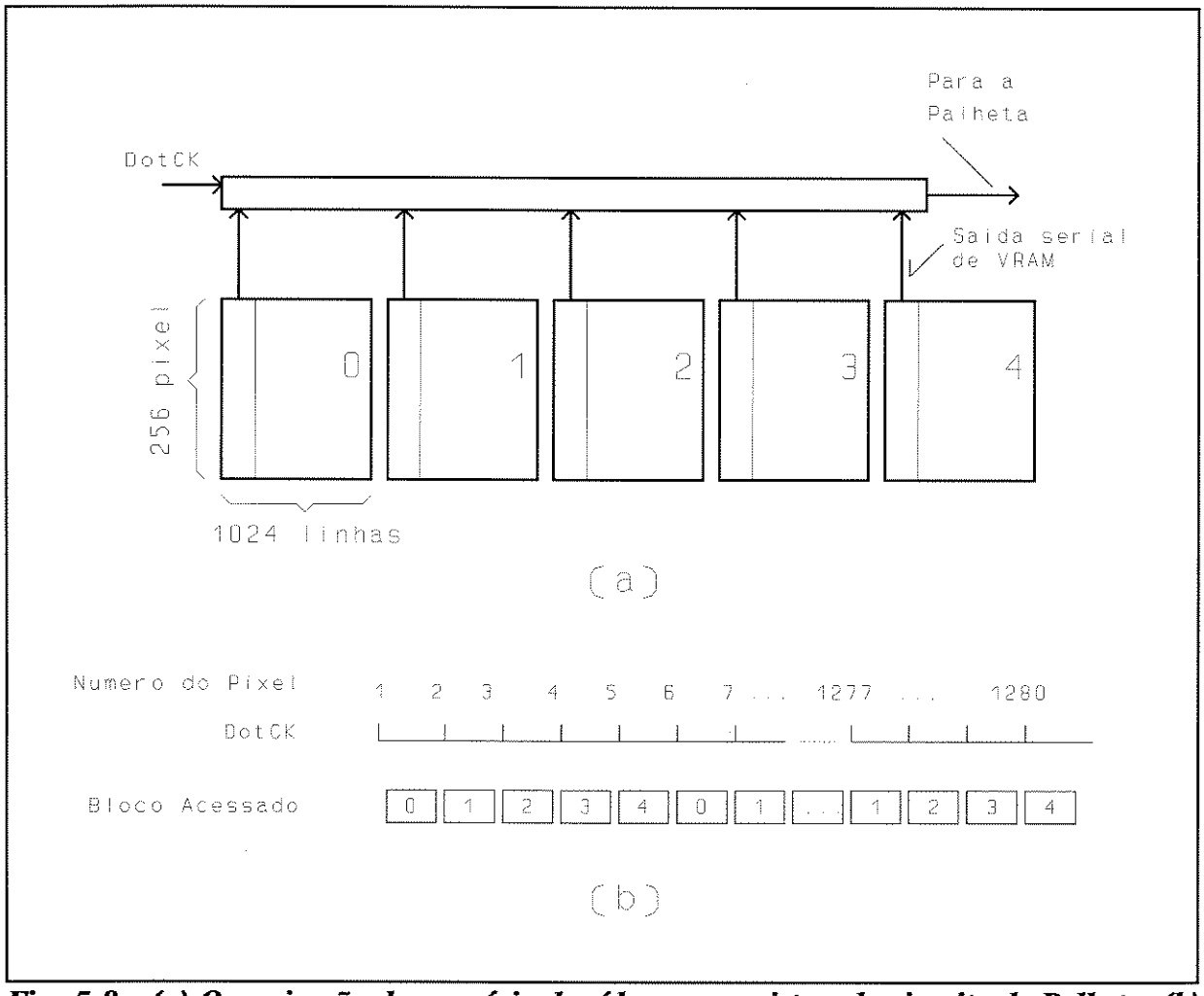


Fig. 5.8 - (a) Organização da memória de vídeo, como vista pelo circuito de Palheta. (b) Relação pixel na tela/bloco de memória.

múltiplo acesso para a memória de quadro. Com isto é possível acionar mais de um pixel por vez. Assim sendo, define-se a necessidade de no mínimo 8 vias de acesso simultâneo (29 360 128 / 3 846 153).

Considerando possíveis degradações do processamento e por facilidade de ligação para a varredura das linhas da tela na resolução de 1280 pixels em 1024 linhas, optou-se por organizar a memória em 20 módulos com acessos independentes.

Assim em cada bloco de memória descrito na subseção anterior existem 4 módulos de memória para processamento ou seja 4 unidades de memória com endereçamento independente para acessos de processamento.

A memória de quadro pode ser vista como sendo um arranjo de 5x4 nós, onde os cinco nós numa linha deste arranjo formam uma linha de varredura da tela

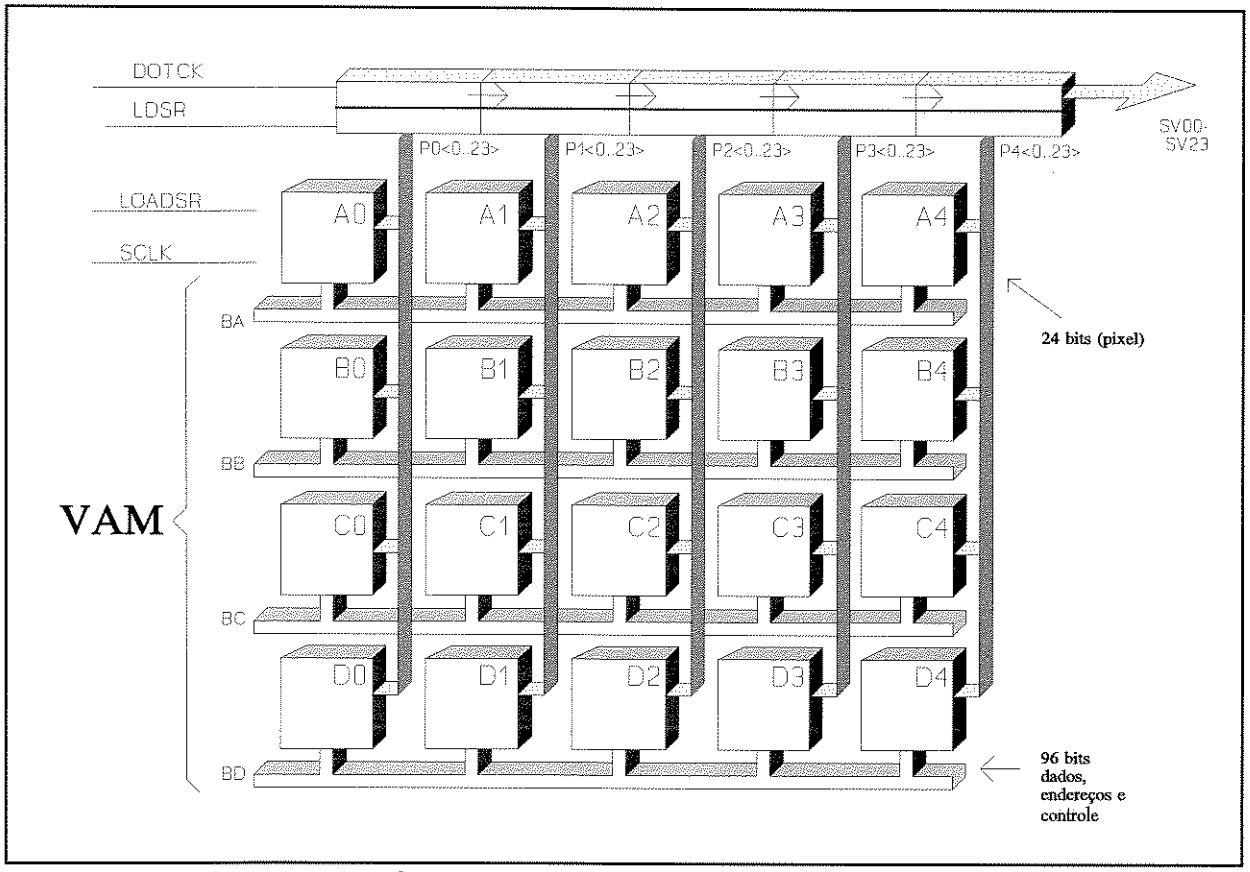


Fig. 5.9 - Arranjo da Memória de Quadro.

e os quatro nós numa coluna deste arranjo formam uma linha vertical na tela. As memórias de vídeo de cada nó alinhado na vertical tem as suas saídas seriais ligadas em comum formando um Barramento Serial. Em função da linha que está sendo varrida, somente uma das saídas seriais deste barramento é ativada. Para proporcionar um intercalamento dos acessos, a ativação das saídas seriais é feita de forma intercalada para os blocos. O controle dos acessos pela porta serial de todos os chips é feito por um único controlador para todos os nós.

Em cada nó existe um processador de acesso para controlar todos os acessos pela porta paralela dos chips da memória de vídeo.

5.2.2.5 - A memória de vídeo interna a cada nó.

Cada nó do arranjo possue 64k pixels. Cada pixel possue 64 bits de dados sendo:

- 32 bits para a definição de cores (8 bits para cada cor primária e 8 bits para a composição de cores);

- 32 bits para a definição do valor de $Z(x_s,y_s)$ para fins de comparação segundo Z-buffer.

O acesso para varredura pelo barramento serial envia em paralelo para o circuito de palheta somente os 24 bits de dados referentes a informação de cor.

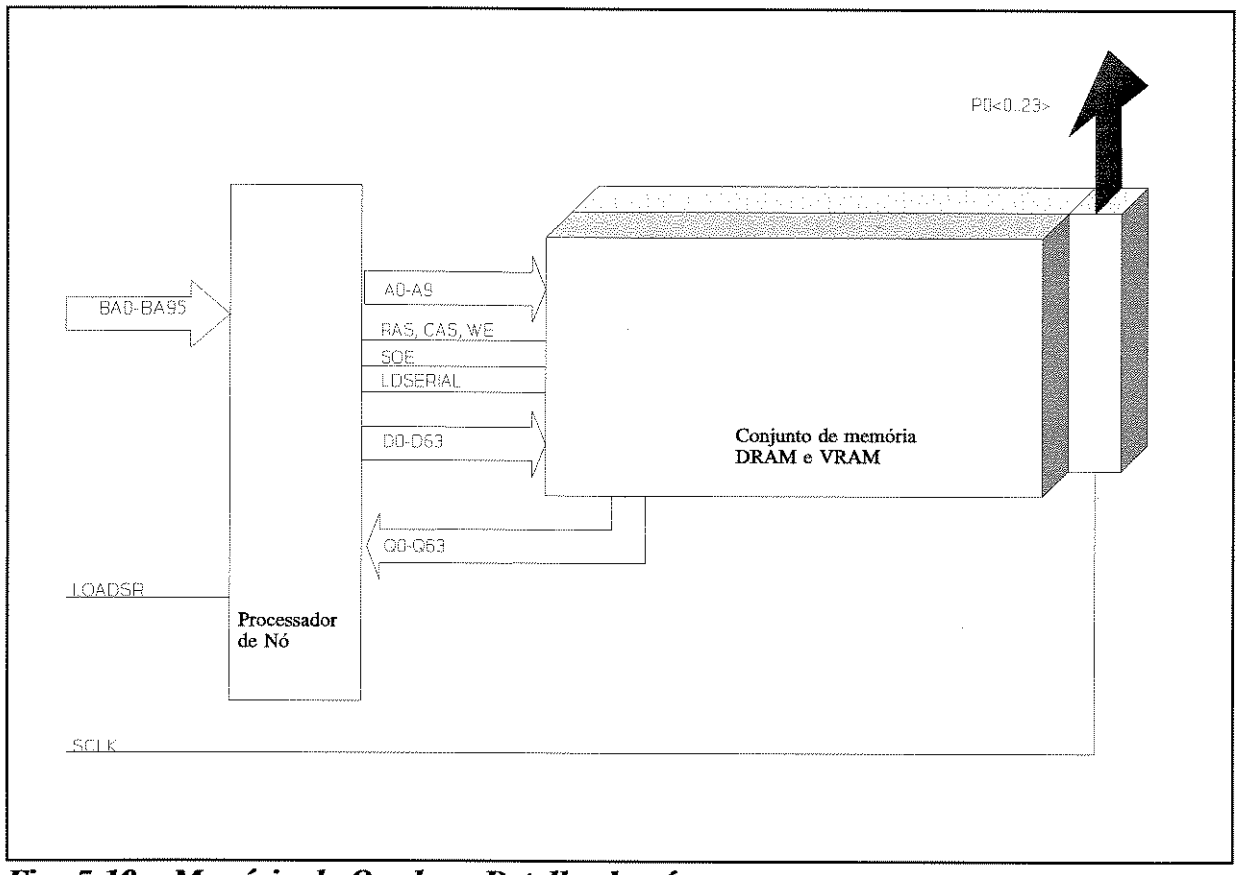


Fig. 5.10 - Memória de Quadro - Detalhe do nó.

O detalhamento das ligações da memória de vídeo interna ao nó de processamento é mostrado na figura 5.10. Nesta figura pode-se ver que o conjunto de chips de memória pode ser dividido em memória do tipo VRAM para o registro da informação de cor e envio para o circuito de palheta (24 bits do pixel), e mais 40 bits do pixel que podem ser implementados com memória do tipo DRAM comum (8 bits de informação alfa e 32 bits do Z-buffer).

Em cada nó do arranjo existem 64 kilo-pixels de memória que são implementados com 6 chips de memória VRAM com a organização 64 kilo-posições com 4 bits e mais 10 chips de memória DRAM comum com a mesma organização [Hita 89].

Do ponto de vista do processador do nó a memória de vídeo tem 64 bits divididos nos campos mostrados na figura 5.11. Os 8 bits mais significativos armazenam a componente vermelha - R (*Red*), a seguir o campo G (*green*)

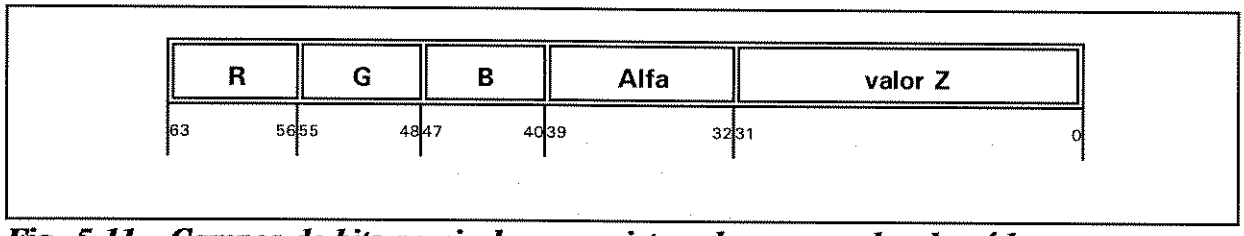


Fig. 5.11 - Campos de bits no pixel, como visto pelo processador de nódo.

armazena a componente verde, o campo B (*Blue*) a azul, o campo Alfa armazena o fator de ocupação para fins de composição e por último os 32 bits menos significativos armazenam o valor da coordenada Z correspondente ao pixel, para fins do tratamento de *Z-buffer*.

Sob o ponto de vista do circuito controlador de varredura somente os bits dos campos R,G e B importam e são enviados em paralelo para o circuito MUX.

5.2.2.6 - O processador interno ao nó.

Este processador é responsável pelo controle dos acessos à memória interna ao nó. Ele gerencia todos os modos de transferência com os *chips* de memória (*page-mode, nibble-mode, read-modify-write ..etc*), controla os ciclos de *refreshing* dos chips de memória e aceita comandos de endereçamento para carga dos registradores seriais das memórias de vídeo. Estes comandos são gerados pelo circuito de controle global de varredura e são comuns a todos os nós do arranjo.

Os ciclos de memória *read-modify-write* são usados para a implementação de operações do tipo Raster-Op, ou seja, aquelas que dependem do conteúdo anterior do pixel que está sendo tratado. As seguintes operações são executadas pelo processador do nó embutidas no ciclo de acesso às memória *read-modify-write*:

XOR Operação lógica OU EXCLUSIVO dos bits de cor - Esta operação é útil à implementação de escritas de informações temporárias como no caso de cursor, *rubber-band* e menus;

ANDNOT Operação lógica E dos bits de cor do conteúdo da posição endereçada com o complemento do valor a ser escrito. Esta operação é útil no apagamento de pontos acionados na tela;

REPLACE Operação de substituição de Pixel, todos os bits do pixel endereçado são substituídos;

REPMIN (***REPlace MINimum***) Operação de substituição condicionada a comparação do valor Z contido na posição endereçada e o valor Z a ser escrito. Se o valor a ser escrito for menor que o valor contido a operação de substituição será realizada. Se o valor a ser escrito for maior, nada será escrito e a posição conservará a seu valor anterior;

RMBLEND (***Replace Minimum with BLENDing***) Operação de substituição como a anterior só que, em caso da condição for satisfeita, a escrita é feita de acordo com a composição de cores sugerida por [PoDu 84], utilizando os bits do campo alfa.

5.2.2.7 - Estimativa do Desempenho Máximo da Memória de Quadro

A memória de quadro da UGR permite o acionamento simultâneo de até 20 pixels. Com isto pode-se atingir as seguintes taxas máximas de acionamento:

Os valores dos parâmetros considerados abaixo são do chip de memória VRAM HM53461-10 da Hitachi Co. segundo [Hita 89].

Valor máximo do ciclo de *read-modify-write* = t_{rmw} = 260 nS
Valor máximo do ciclo normal de acesso = t_{ac} = 100 nS
Valor máximo do ciclo de acesso no modo *page* = t_{PC} = 70 nS

RepMin20 = Taxa de RepMin de um polígono com 20 pixels, ou seja, taxa de acionamento de pixels condicionado a comparação de profundidade, utilizando ciclos de *read-modify-write*.

$$\text{RepMin20} = \frac{1}{t_{rmw}} = 3846153[\textit{polígonos/seg}] \textit{ ou } 76923076[\textit{pixel/seg}] \qquad \textbf{(5.14)}$$

Rep20 = Taxa de Replace de um polígono com 20 pixels, ou seja, taxa de acionamento incondicional de pixel, utilizando ciclos normais de acesso.

$$\text{Rep20} = \frac{1}{t_{ac}} = 10000000[\textit{polígonos/seg}] \textit{ ou } 200000000[\textit{pixel/seg}] \qquad \textbf{(5.15)}$$

FillSc = Taxa de Preenchimento de tela com uma cor. Utiliza acionamento incondicional de pixel, através de ciclos normais no modo *page* de acesso a memória.

$$\begin{gathered} \text{CyFill} = (\frac{1280}{5}) \times (\frac{1024}{4}) \times t_{PC} = 4{,}58752 \textit{miliSeg} \\ \text{FillSc} = \frac{1}{\text{CyFill}} = 217{,}98[\textit{telas/seg}] \textit{ ou} \\ 1280 \times 1024 \times \text{FillSc} = 285714285[\textit{pixel/seg}] \end{gathered} \tag{5.16}$$

AcPix = Taxa de acionamento de pixel isolado aleatório. Utiliza acionamento condicionado a comparação de profundidade.

$$\text{AcPix} = \frac{1}{t_{rmw}} = 3846153[\textit{pixel/seg}]$$

AcHor20 = Taxa de acionamento de linha horizontal e comprimento de 20 pixels. Utiliza acionamento condicionado a comparação de profundidade. Uma linha horizontal de comprimento h pode ser traçada em

$$\text{Roundup}(\frac{h}{5}) \times t_{rmw} \tag{5.18}$$

assim

$$\text{AcHor20} = \frac{1}{(4 \times t_{rmw})} = 961538[\textit{linhas/seg}] \tag{5.19}$$

AcVer20 = Taxa de acionamento de linha vertical e comprimento de 20 pixels. Utiliza acionamento condicionado a comparação de profundidade. Uma linha vertical de comprimento v pode ser traçada em

$$\text{Roundup}(\frac{v}{4}) \times t_{rmw} \tag{5.20}$$

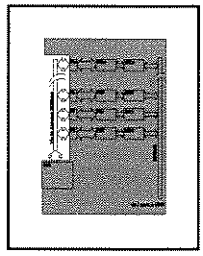
assim

$$\text{AcVer20} = \frac{1}{(4 \times t_{rmw})} = 961538[\textit{linhas/seg}] \tag{5.21}$$

AcLin20 = Taxa de acionamento de linha com inclinação aleatória e comprimento de 20 pixels. Utiliza acionamento condicionado a comparação de profundidade. Uma linha de inclinação aleatória tem no pior caso, a repetição de 20 vezes do acionamento no mesmo nó o que pode significar

$$\text{Aclin20} = \frac{1}{(20 \times t_{rmw})} = 192307[\textit{linhas/seg}] \tag{5.22}$$

Os valores acima estimados representam o limite máximo de desempenho gráfico que o arranjo da memória de quadro pode atingir. A transferência para a memória de quadro, que depende de concorrência entre processadores, é que irá definir o desempenho do sistema gráfico.



5.2.3 - VAM - Via de Acessos Múltiplos.

A ligação entre os processadores e a memória de quadro é feita pela VAM. A topologia proposta para esta via procura reunir a eficiência de interconexão de um *Crossbar* com a simplicidade de implementação de barramento.

O objetivo é conectar p processadores a qualquer um dos m nós da memória de quadros.

A solução mais eficiente, em termos de desempenho, para este tipo de interconexão seria uma estrutura CrossBarr p x m, como na figura 5.12. Numa topologia que empregasse este tipo de interconexão, com $p \leq m$ seria possível dispor de uma ligação direta exclusiva (sem concorrência) entre um processador e um nó de memória, permitindo assim p acessos simultâneos a pixel. Poder-se-ia processar (*rendering* e acionamento de pixel) simultaneamente m regiões da tela, ou processar simultaneamente p objetos distintos. Cada um dos p processadores poderia se conectar a cada um dos m nós de memória de forma dinâmica. Na figura 5.12 tem o exemplo de uma interconexão para $p = m = 20$.

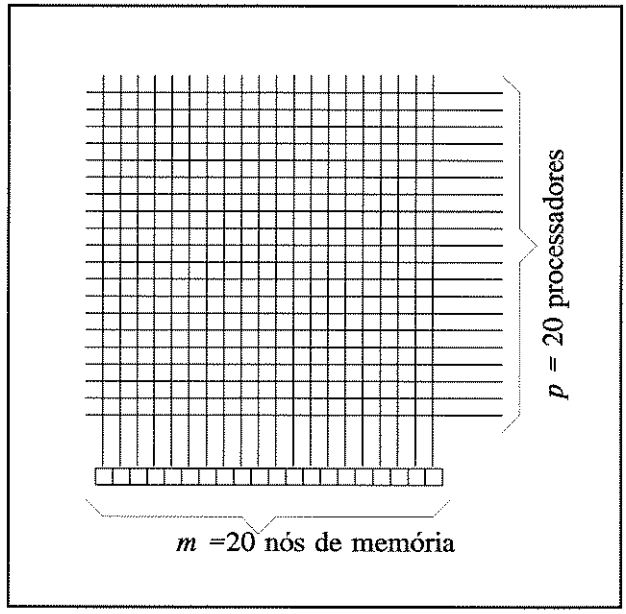


Fig. 5.12- Interconexão Crossbar

A dificuldade do emprego desta forma de interconexão se prende a quantidade de componentes elétricos necessários para a sua implementação. Se for considerado que cada processador elementar deve se comunicar com todos os m nós de memória, e mais, que cada uma destas vias possue d bits de dados, a bits de endereço de pixel e c bits de controle, seriam necessários pelo menos

$$(d + a + c)m \qquad \textbf{(5.23)}$$

gates por módulo de processamento elementar.

Considere o exemplo da interconexão de $p = 20$ processadores a $m = 20$ nós de memória, como na figura 5.12, fazendo:

$$\begin{aligned} d &= 64 \quad \text{(largura do barramento de dados)} \\ a &= 16 \quad \text{(largura do barramento de endereços)} \\ c &= 0 \quad \text{(número de sinais de controle)} \\ d+a+c &= 80 \end{aligned}$$

serão necessários 80 x 20 = 1600 *gates* para cada um dos processadores.

Para diminuir este número de componentes foram analisadas duas alternativas, descritas a seguir:

Alternativa 1 - Multiplexando a informação nas linhas do *crossbar*

Para diminuir o número de componentes pode-se multiplexar a informação (dados e endereços) numa quantidade menor de linhas de comunicação. Ou seja, no lugar dos d bits de dados, mais a bits de endereço e mais c bits de controle, estas informações poderiam estar multiplexadas em f fios de comunicação. No máximo isto degrada a velocidade de transferência de um fator de

$$\frac{(d + a + c)}{f} \tag{5.24}$$

vezes. Entretanto a quantidade de componentes neste caso é

$$\left[\frac{(d + a + c)}{f}\right] m \tag{5.25}$$

gates por módulo processador.

Assim no exemplo da interconexão de 20 processadores a 20 blocos de memória, fazendo $f = 8$ (um *byte*) a degradação da velocidade seria de 10 vezes, mas a quantidade de componentes por módulo, seria reduzida para 200 *gates*. Se for possível manter a multiplexação destas informações dentro de um período de acesso a memória de quadro, não haverá perdas na velocidade de transferência. Para uma memória com tempo de acesso da ordem de 260 nseg a via de comunicação teria que multiplexar num período de 26 nseg, ou seja numa frequência de aproximadamente 40 megahertz.

Alternativa 2 - Reduzindo m, o número de blocos de memória a ser endereçado simultaneamente.

Outra alternativa para diminuir a quantidade de componentes necessários a esta interconexão é reduzir o número de nós possíveis de serem endereçados simultaneamente, ou seja, fazer $m = n < p$. A diminuição do número de canais de acesso a memória, faz com que alguns processadores tenham que compartilhar um mesmo barramento.

Se for considerado que o tempo de acesso do processador à memoria (t_{rw}) é, em geral, menor que o tempo de acesso da memória (t_{ac}), pode-se ter nesta via a comunicação com diversos nós sem degradação na transferência. O número de processadores a compartilhar a mesma via sem degradação da transferência é então

$$r = \frac{t_{ac}}{t_{rw}} \qquad \textbf{(5.26)}$$

Considere o exemplo de um processador com tempo de acesso para escrita da ordem de 50nseg ligado a uma memória de tempo de ciclos de *read-modify-write* de 260 nseg, tem-se 260 / 50 $\approx$ 5 processadores compartilhando simultaneamente a mesma conexão.

Assim sendo, a VAM deve ser composta por

$$b \geq \frac{m}{r}$$

barramentos de comunicação.

Todos os p processadores podem se ligar a cada um dos b barramentos. Uma configuração típica é aquela em que $p = m = r \times b$ processadores se dividirão igualmente nos b barramentos. Cada um dos b barramentos se comunica com somente r nós de memória.

Cada um destes barramento é do tipo síncrono com fatias de tempo dedicadas a somente um nó de memória. Um processador precisando se comunicar com um nó de memória deve então se conectar ao barramento ligado a este nó e esperar a fatia de tempo dedicada àquele nó e quando ele ocorrer, fazer a transferência. A espera máxima seria a mesma daquela que ele teria que esperar se estivesse ligado diretamente a memória.

O valor determinado acima é um limitante inferior para o número de barramentos b. Este número deve ser suficiente para atender o requisito RQ2/3/5/1 (Acessos Simultâneos). É necessário manter uma taxa de transferência para a memória da ordem de 30 000 000 pixel/seg.

Na determinação deste número deve ser considerado o compromisso de maior desempenho, obtido com um maior número de barramentos, contra a menor quantidade de componentes.

Considerando o arranjo de memória proposto na seção 5.2.2.3 e fazendo $p = m = 20$

se for considerado que $t_{ac} = 260$ ns e $t_{wr} = 50$ ns

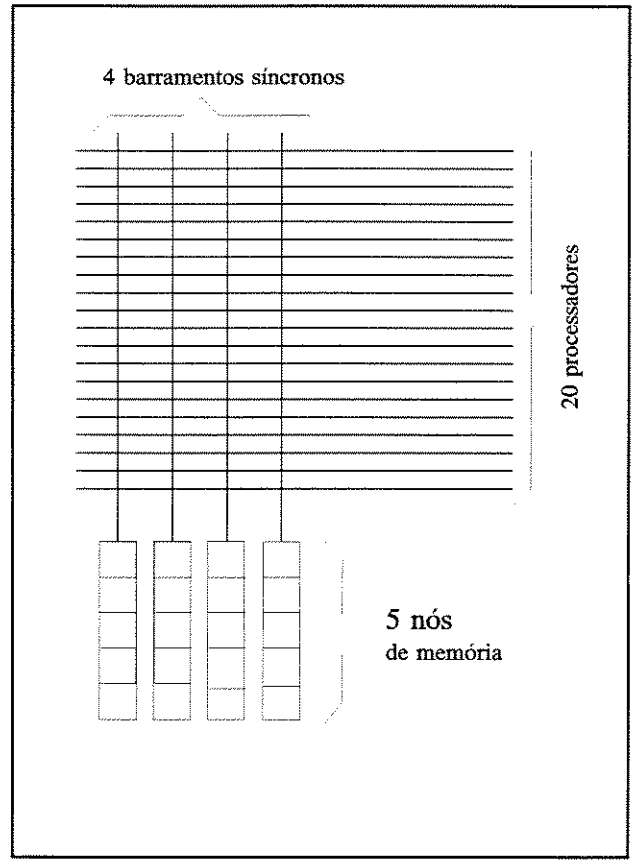


Fig. 5.13 - Interconexão segundo a alternativa 2.

tem-se:

$$r = 260 / 50 = 5{,}2$$

e

$$b \geq 20 / 5{,}2$$
$$b \geq 3{,}8$$

Observando o compromisso entre b e a quantidade de componentes, propos-se a quantidade de 4 barramentos síncronos com fatias de tempo iguais dedicadas a 5 nós de memória, como mostrado na figura 5.13.

A medida que o aplicativo gráfico solicita, até cinco processadores se ligam a uma desta 4 vias e será concluída a transferência.

Numa aplicação que demande acessos sucessivos a memória de quadro a cada 520 ns ter-se-ia 20 transferências que é correspondente a aproximadamente 40 milhões de transferências por segundo.

Considerando o exemplo analisado, resta determinar a quantidade de componentes por módulo processador para a ligação com a VAM. Para os 20 processadores seriam necessárias conexões a 4 barramentos síncronos.

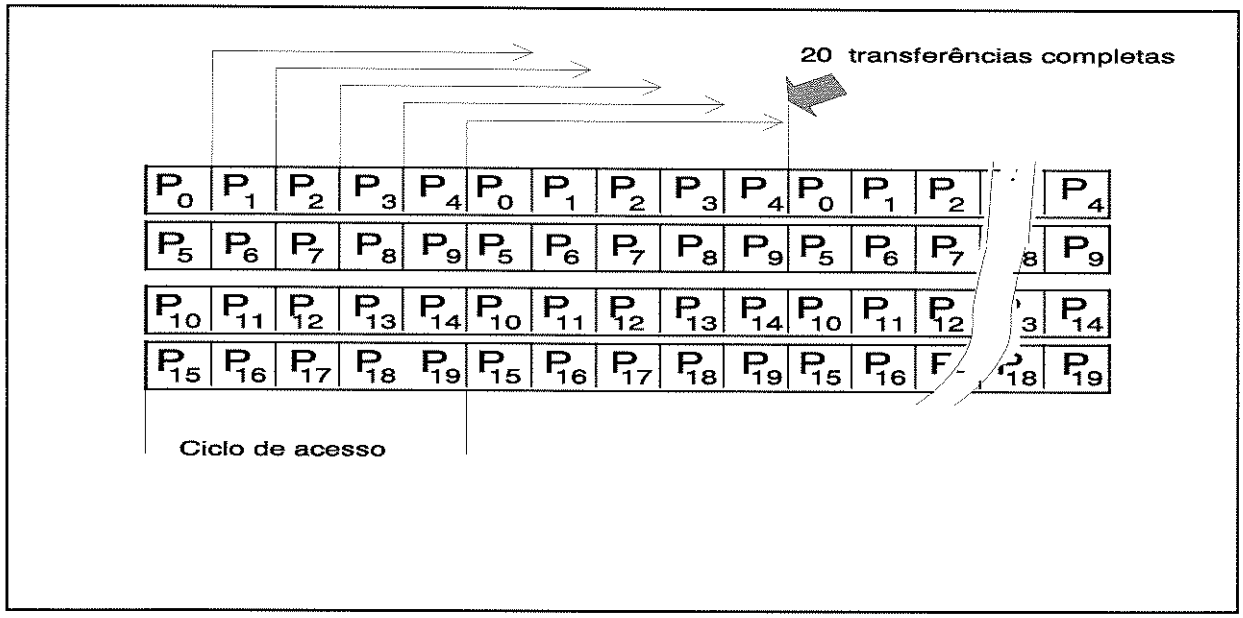


Fig. 5.14 - Temporização da VAM.

Considerando a facilidade de ligação com a organização da memória de quadro, apresentada na seção 5.2.2.6, optou-se pela alternativa 2.

O formato da informação a ser transferida pelos barramentos síncronos da VAM.

O formato desta informação é dividido em 4 campos, a saber:

Campo de dados: contém a informação de cor a ser enviada a tela, o seu formato é o mesmo descrito na figura 5.11, ou seja, com 64 bits — 24 bits de cor (RGB) — 8 de fração do pixel (para mistura, código alfa) — e 32 bits de profundidade;

Campo de endereço: contém a informação da posição do pixel a ser acionado, o seu formato é de 18 bits, permitindo o endereçamento de 256 *kilopixels*;

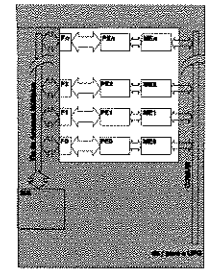
Campo de código: contém a informação do tipo de acesso a ser realizado, ou seja, o tipo de operação a ser realizada durante o ciclo de escrita na memória de quadro; dentre os códigos permitidos neste campo, estão incluídos aqueles que permitem o acesso aos registradores de cor da palheta e aos registradores de configuração da memória de quadro; estão reservados para este campo 5 bits, o que permite até 32 códigos de tipos de acesso;

Campo de controle: contém a sinalização de controle, a saber, sinal de relógio, contador de sincronismo, *reset* e indicador de ciclos de leitura; estão reservados para este campo, 6 bits.

Além dos sinais acima, estão previstos nesta conexão, os sinais de nível de referência (*ground*).

O detalhamento de cada um dos campos e dos seus sinais é feito na referência [Oliv 95].

5.2.4 - O Arranjo de Processadores.



O processamento geométrico e de *rendering* é feito num conjunto de módulos processadores de propósito geral e alto desempenho. Para a definição da arquitetura de processamento procurou-se deixar o processamento numérico gráfico restrito a módulos que pudessem ser atualizados (*upgraded*) a medida que novos processadores mais velozes estejam disponíveis. A opção por processador de propósito geral se justifica pela facilidade de desenvolvimento de programas aplicativos. Todos os módulos de processamento que compõem este conjunto são absolutamente iguais entre si. Pretende-se com isto estabelecer uma escalabilidade que permita ampliar a capacidade de processamento pela simples adição de novos módulos. Considera-se inicialmente que uma configuração limite é aquela com um processador por nós da memória de quadro. Assim, cada um dos módulos processadores deste conjunto pode ser conectado a um dos nós de memória de vídeo através da VAM, podendo tratar (processar e acionar) independentemente todos os pixels associados a este nó. Considerando a proposta de um arranjo de memória de vídeo com vinte nós, o limite proposto para este conjunto é de vinte módulos processadores.

Os módulos processadores estão ligados a um barramento paralelo comum (barramento COMUM na figura 5.4) que é usado principalmente para o envio de dados de descrição de cena a ser exibida. Este barramento permite a transferência de informação em *broadcasting*, ou seja, uma informação pode ser enviada simultaneamente para todos processadores. A informação enviada desta forma é retida numa memória local interna ao módulo processador. O paralelismo do processamento é baseado na repetição da estrutura de dados que descreve a imagem em todos os módulos processadores. Cada processador poderá tratar de forma absolutamente independente a mesma estrutura de dados. Ou seja, o processo de Busca da Estrutura de Dados é também paralelizado. Cada processador recebendo todas as informações relativas à imagem, testa se

algum dos seus pixels pode ser alterado. Se for este o caso o processador inicia o seu tratamento. Desta forma diversos pixels, relativos a um mesmo elemento de imagem, podem estar sendo tratados simultaneamente em diferentes nós.

Por exemplo, a descrição de um objeto pode ser enviada ao mesmo tempo para todos os nós. Ao receber estas informações cada processador executa as operações necessárias a escrita dos pixels correspondentes ao objeto no nó de memória de vídeo a que está conectado. Desta forma, na configuração considerada acima, pode-se ter até vinte pixels sendo endereçados simultaneamente.

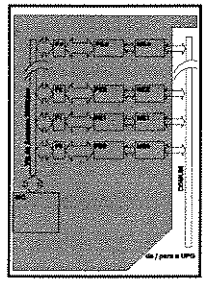
5.2.4.1 - A ligação com o barramento COMUM.

A ligação com o barramento COMUM é feita principalmente através de uma memória local (ME*n*) de dupla porta. Esta memória é alocada no mapa de endereçamento do processador PE*n* que pode endereçar de forma aleatória todo o seu conteúdo. Por outra porta, esta memória pode ser operada pelo barramento COMUM, também de forma aleatória. Além da memória local, faz parte do mapa de memória do processador endereços que permitem ao processador ter acesso ao barramento COMUM para envio de mensagens a outros processadores, ou para a UGR.

5.2.4.2 - A interface de acesso à VAM

O acesso do processador PE*n* à Via de Acessos Múltiplos é feita através da Interface local F*n*. Esta interface tem como principal tarefa fazer com que o protocolo de comunicação com a VAM fique transparente ao processador. A ideia é que o processador PE*n* simplesmente enderece uma posição de memória, a interface F*n* trata os bits de endereço deste acesso de acordo com o protocolo da VAM; selecionando por exemplo, qual barramento será utilizado, qual a temporização,...etc. Para adequar a temporização do processador aos acessos a VAM esta interface possui uma fila de acessos.

5.2.5 - O barramento COMUM.



O barramento COMUM é um barramento paralelo para multiprocessamento, qualquer módulo ligado a este barramento pode se tornar seu mestre, ou seja, controlar um ciclo de transferência no mesmo.

Este barramento permite além dos ciclos de escrita e leitura simples, a transferências em *broadcasting* (de um mestre para todos os módulos do barramento.

A principal atividade no barramento COMUM é o envio pela UGR para todos os módulos processadores da descrição de uma imagem a ser processada e exibida.

5.3 - Resumo e Comentários.

5.3.1 - Resumo

Este capítulo partiu da definição de um posto de trabalho para a geração de imagens com realismo. Foi definido que um posto de trabalho para esta aplicação pode ser dividido em duas unidades, uma de propósito geral (UPG) e outra com características específicas para atender as demandas de processamento gráfico (UGR). A partir disto foram formalizados os requisitos de projeto deste posto.

Na formalização dos requisitos evitou-se aprofundar na UPG, pois considerou-se desde o início que ela poderia ser um equipamento comercial. Procurou-se então concentrar esforços na definição da UGR.

Com os requisitos de projeto da UGR definidos, na seção 5.2 foi feito um resumo da especificação das características desta unidade, com mais destaque para a especificação da memória de quadro (MQ/UGR) feita na seção 5.2.2 e da via de acessos múltiplos (VAM) feita na seção 5.2.3. A especificação dos módulos processadores e do barramento COMUM é feita de forma bem resumida, para enfatizar a característica de abertura da arquitetura proposta.

5.3.2 - Requisitos x Solução

A seguir é feita uma análise de como as propostas apresentadas cobrem os requisitos definidos para o projeto da UGR.

O RQ2/3/5/1 - Acessos independentes - definiu a necessidade de criar uma via com múltiplos acessos para a memória de quadro da UGR. Os estudos de alternativas e a especificação da VAM foram feitos na seção 5.2.3. Com esta via foi possível cobrir o requisito de alta velocidade na atualização da tela (RQ2/3/5).

A definição de um campo de informação de profundidade e de um campo de fração de ocupação (código alfa) junto a cada pixel, como descrito na seção 5.2.2.5, visam cobrir os requisitos RQ2/3/3 - Composição *Alpha* e RQ2/3/4 - *Z-buffer hardware*.

A possibilidade de poder conectar até vinte módulos processadores para formar a UGR aliada às taxas de processamento em ponto flutuante obtidas por processadores comerciais (da ordem de 4 milhões de multiplicações por segundo em circuitos do tipo DSP), torna factível um arranjo de processadores que ultrapasse o requisito de 111Mflops — RQ2/3/2 - Operações Ponto-Flutuante($>$110Mflops).

A repetição de toda a base de dados nas memórias locais de cada módulo processador elementar permite que a descrição da imagem a ser exibida seja varrida em paralelo por até vinte processadores, isto aliado a uma eficiente distribuição de tarefas pelos processadores garante a cobertura do requisito RQ2/3/1 - Taxa de Transferência($>$28Mbytes/s)

Os últimos parágrafos comprovam a cobertura do requisito de alta velocidade (RQ2/3).

O circuito de Palheta descrito na seção 5.2.2.1 visa atender o requisito RQ2/2/1 - Palheta de Cores.

A definição de 24 bits para o registro da informação de cor em cada pixel, atendem ao requisito de alta definição de cores (16Mcores RQ2/2).

A estrutura de memória de quadro descrita na seção 5.2.2 foi centrada no atendimento do requisito RQ2/1 - Alta Resolução (1280x1024). A principal crítica que pode ser feita a esta estrutura é justamente a sua rigidez na cobertura deste requisito, que não permite a configuração de outras relações de aspecto que a pegada (5x4) desta resolução define.

Cobrindo os requisitos RQ2/1, RQ2/2 e RQ2/3, como comprovado acima, a implementação da estrutura aqui descrita, resultará numa bancada com alta capacidade gráfica (RQ2).

5.3.3 - A UGR como uma Bancada de Experimentos

Concluindo este capítulo, cabe comentar algo sobre a proposta da UGR.

Em resumo, a especificação desta unidade define uma arquitetura de memória de quadro, ligada a uma via de comunicação, chamada VAM. Esta via está aberta à ligação de até vinte elementos processadores. Estes elementos processadores não estão definidos *a priori* neste trabalho. Pretende-se com isto deixar para a aplicação definir o tipo de processador a ser anexado, formando assim uma UGR completa.

Para evidenciar esta característica de unidade aberta foi adotado o termo **BANCADA** para definí-la. O qualificativo "**de EXPERIMENTO**" reforça o seu aspecto acadêmico.

A definição desta bancada é que concentrou a maior parte dos esforços de desenvolvimento do trabalho aqui apresentado. A referência [Oliv 95] descreve os resultados da descrição formal desta bancada utilizando-se os recursos de Engenharia Concorrente disponíveis na FEE-Unicamp.

O capítulo seguinte ilustra, através de 2 exemplos, como processadores elementares podem ser integrados à bancada em função da aplicação.

Capítulo 6

Aplicação da Bancada.

O presente capítulo tem como objetivo exemplificar o uso da UGR sob diferentes pontos de vista. Na seção 6.1, é discutida utilização da UGR na aplicação Visualização de Volumes. Na seção 6.2 são feitas considerações sobre como as aplicações de geração de imagem com realismo podem se valer da bancada UGR. O capítulo continua na seção 6.3, com comentários sobre a utilização da UGR segundo os aspectos levantados por [Whit 90]. A seção 6.4 finaliza resumindo o capítulo.

6.1 - Aplicação Visualização de Volumes.

A visualização de volumes se aplica, por exemplo, no tratamento de imagens médicas. Neste caso, a partir de uma base de dados obtida por meios magnéticos (ressonância magnética) ou ultra-sônico, uma imagem deverá ser exibida num monitor de vídeo ao seu analista (no exemplo, um médico). Para uma melhor análise, esta base de dados deverá poder ser "varrida" sob comandos da forma mais natural possível, como por exemplo, reposicionando o ponto de vista do analista, alterando o grau de transparência de determinados volumes permitindo a visualização de volumes escondidos. Desta forma é possível ao analista "mergulhar" literalmente na estrutura a ser analisada. Outros exemplos de aplicação da visualização de volumes podem ser encontradas no sensoriamento remoto (visualização de dados de satélites meteorológicos, ..., etc.), análises não destrutivas de estruturas na construção civil, na mineralogia, na biologia....etc.

O tratamento da presente aplicação, utilizando a UGR, poderá ser feito por um arranjo de processadores na estrutura de duas formas distintas, uma utilizando aceleradores em *hardware* específicos e outra utilizando processadores de propósito geral (de prateleira).

6.1.1 - A Integração de Processadores Específicos.

Para exemplificar a integração de um processador de uso específico à UGR, foi escolhido o VERVE *Voxel Engine for Real-time Visualization and Examination, Volume Traversal*, publicado no Eurographic, volume 12, número 3, 1993. O autor, Günter Knittel, da Universidade de Tübingen, propõe um *hardware*

acelerador de *rendering* de volumes, visando realidade virtual, onde são tratados também os problemas de projeção em perspectiva, iluminação com raios não paralelos oriunda de uma fonte móvel qualquer e tratamento de profundidade.

Visando dar uma idéia da complexidade do processamento executado pelo VERVE é apresentado, a seguir, um resumo dos diversos algoritmos usados por ele nas etapas de *Volume Traversal, Surface Classification*, *Shading* e *Compositing*.

6.1.1.1 - Volume Traversal

A informação é exibida utilizando-se o método *ray-casting*. A partir do olho do observador, um raio de visada é seguido através do volume para cada *pixel* (ou *sub-pixel*) no plano da tela. A informação é reconstruída e amostrada (ou re-amostrada) a cada localização regularmente espaçada ao longo do raio usando interpolação tri-linear. Por isto, um elemento de volume é subdividido novamente de forma que o endereço de um ponto tenha a forma $\{X_I.x_f; Y_I.y_f; Z_I.z_f\}$, onde I representa a parte inteira e F a parte fracionária do endereço. O endereço de uma amostra original é descrita por:

$$x_F = y_F = z_F = 0 \quad \text{e} \quad 0 \leq X_I;\ Y_I;\ Z_I \leq 511 \qquad \textbf{(6.1)}$$

A parte fracionária tem 8 *bits* para simplificar o *hardware*. Os parâmetros exigidos para cada ponto são a densidade, o gradiente de densidade e a magnitude do gradiente.

Para fins de velocidade e conforto do usuário, a máquina deve ser capaz de processar autonomamente um raio completo e de detectar a entrada e a saída de cada raio de cada volume. Desde que os limites do volume são programados pelo usuário, planos de recorte arbitrários (paralelos aos planos (x,y), (x,z) (y,z)) podem ser instalados. Além disto, planos de recorte frontais de qualquer formato podem ser preparados pela programação individual da primeira localização de re-amostragem de cada raio.

6.1.1.2 - Classificação de Superfícies (*Surface Classification*)

A visibilidade de um ponto é definida pela sua opacidade Ω. Estruturas internas ao volume são distinguidas pela sua densidade específica D e a magnitude de gradiente G. A tarefa de classificação é então o mapeamento apropriado de D e G para Ω. Isto é conseguido usando D e G como um ponteiro para uma tabela de opacidade pré-computada como proposto por Levoy [Levo 88]. A extração do objeto desejado de um conjunto de dados é principalmente obtida pela

adaptação interativa da forma da função de opacidade.

6.1.1.3 - Tonalização (*Shading*)

Para cada localização de re-amostragem, um modelo de iluminação de Phong modificado é calculado.

$$I_\lambda = f_{dc\lambda}(\Delta)\left[I_{A\lambda}k_a C_\lambda + I_{L\lambda}\left(k_d C_\lambda\left(\overline{G_N}\,\overline{L_N}\right) + k_s\left(\overline{G_N}\,\overline{H_N}\right)^n\right)\right] \qquad \textbf{(6.2)}$$

É utilizado um modelo com três componentes de cores onde λ pode ser tanto vermelha (R), verde (G) ou azul (B). A intensidade de luz I_λ de um ponto tem uma parte ambiental, uma parte de reflexão difusa e uma parte de reflexão especular. $f_{dc\lambda}(\Delta)$ é um fator de tratamento de profundidade (*depth cueing*), que é função da distância Δ entre o observador e o ponto de re-amostragem. Este fator é usado para simular uma atenuação de intensidade, bem como um deslocamento de cores de pontos distantes. A luz ambiental $I_{A\lambda}$ e o coeficiente de reflexão ambiental k_a são considerados como constantes através do volume. A cor C_λ de um ponto é retirada de uma tabela de cores (*color look-up table*) que é endereçada pela densidade interpolada D. $I_{L\lambda}$ é a intensidade da fonte de luz. $\overline{G_N}$ é o gradiente de densidade normalizado que serve como a normal da superfície. $\overline{L_N}$ é o vetor normalizado que aponta para a fonte de luz. $\overline{G_N}\,\overline{L_N}$ é referido como $\cos\alpha$. $\overline{H_N}$ é o vetor normalizado de iluminação mais forte sendo chamado de $\cos\beta$. Tanto o coeficiente de reflexão difusa k_d quanto o coeficiente de reflexão especular k_s são constantes. O expoente da reflexão especular n varia de 1 até várias centenas. A cor refletida especular é considerada como sendo a da fonte de luz.

Assim (6.2) pode ser re-escrita como :

$$I_\lambda = C_\lambda(I_{1\lambda} + I_{2\lambda}\cos\alpha) + I_{3\lambda}\cos^n\beta \qquad \textit{onde:} \qquad \textbf{(6.3)}$$

$$I_{1\lambda} = f_{dc\lambda}(\Delta)I_{A\lambda}k_a \;; \qquad I_{2\lambda} = f_{dc\lambda}(\Delta)I_{L\lambda}k_d \;; \qquad I_{3\lambda} = f_{dc\lambda}(\Delta)I_{L\lambda}k_s \;.$$

$I_{1\lambda}$, $I_{2\lambda}$ e $I_{3\lambda}$ são pré-calculados e retirados de um tabela que é endereçada por Δ. Uma vez que $\cos\beta$ é computado, ele é usado como índice para uma tabela de $\cos^n\beta$. Considerando que $\cos\alpha$ é também calculado, uma operação de tonalização de Phong pode ser executada por um *pipeline* de multiplicação e acúmulo. Assim, o problema de tonalização por Phong em tempo real é de fato o problema de vetores normalizados em tempo real, e desde que são suportadas

a projeção de perspectiva e luz não paralela, nenhum dos vetores é constante.

6.1.1.4 - Composição (*Compositing*)

Um ponto com uma dada opacidade Ω age de duas maneiras: ele reflete uma fração da luz proveniente da fonte de luz; e atenua a luz que vem de trás dele. A cor final do (*sub-*)*pixel* P_λ que é retornada após N amostras é dada pela fórmula

$$P_\lambda = \sum_{n=0}^{N} \left[I_{\lambda n}\, \Omega_n \prod_{m=0}^{n-1} (1 - \Omega_m) \right] \qquad \textbf{(6.4)}$$

onde $I_{\lambda 0}$ refere a luz refletida do primeiro ponto no raio. $I_{\lambda N}$ é a cor de fundo e $\Omega_N = 1$. O processamento do raio pode ser terminado num ponto k se a transparência $(1-\Omega_0)(1-\Omega_1)(1-\Omega_2)\cdots(1-\Omega_{k-1})$ de um ponto na sua frente estiver num determinado limiar ϵ.

6.1.1.5 - A Integração do Verve à UGR.

Além do *hardware* do VERVE englobar os algoritmos acima, ele também resolve a interpolação tri-linear e possui um normalizador de vetores.

Assim todo o processamento necessário para o tratamento individual de cada raio é feito pelo próprio *hardware* do VERVE, deixando para a UGR a visualização da imagem e recursos para a troca de informações entre os diversos processadores VERVE's instalados na UGR.

Segundo Knittel, para obter-se velocidade suficiente com estes processadores, para o tratamento interativo, bastaria um único processador, mas para operações em tempo real seriam necessários de 4 a 16 processadores. A VAM permite a interconexão de até vinte processadores, cabendo a cada um deles, o cálculo de propagação de um conjunto de raios.

O *hardware* específico para a instalação de VERVE's na UGR, reduz-se então à interface para compatibilizar a saída do VERVE com os barramentos da VAM. Seria necessário adicionar ao circuito de processamento uma interface de acesso à VAM. Esta seria ligada à saída da memória de quadro da proposta original. A VAM permite que cada processador possa ter acesso a qualquer *pixel* da tela independentemente de sua localização na memória de quadro. A distribuição dos raios por processador se daria em função dos pixels na tela. Como cada *slot* de tempo num dos barramentos da VAM está dedicado a um

conjunto de *pixels*, a distribuição é feita simplesmente pela conexão dos processadores a um *slot* de tempo de um dos barramentos.

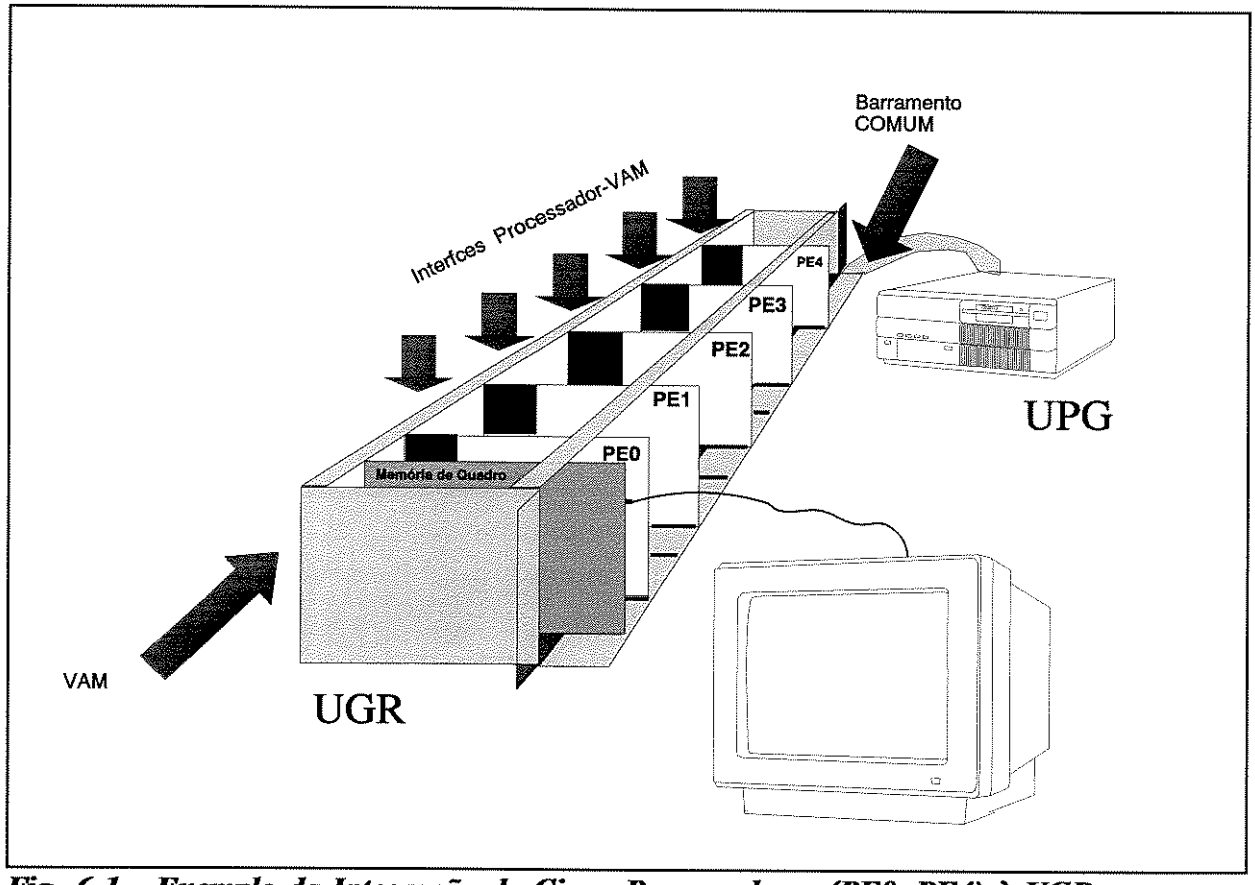


Fig. 6.1 - Exemplo da Integração de Cinco Processadores (PE0...PE4) à UGR.

A figura 6.1, apresenta de forma pictórica a integração de cinco processadores VERVE à UGR.

O *software* e estrutura de dados não são particulares, ou acoplados ao VERVE, e sim, fortemente dependentes da aplicação. Cabe aqui realçar, a necessidade de uma interface de *software* para a comunicação VERVE-UGR.

6.1.2 - A Integração de Processadores de Propósito Geral.

Outra alternativa de processamento considera uma UGR formada por módulos processadores construídos com dispositivos comerciais do tipo, i860 [KoMa 89], DSP como por exemplo o DN56000 da Motorola, ou componentes SPARC da Sun MicroSystem. Estes módulos além do processador dispõem de memória local com dimensão para receber toda a descrição de uma imagem, interface com o barramento COMUM, e interface com a VAM. Esta alternativa permite que todo o aplicativo seja desenvolvido utilizando-se de todos os recursos de

programação disponível para o processador, desde o *front-end, traversal* e tratamento geométrico até *rendering*.

Para o desenvolvimento do aplicativo deve-se levar em conta que:

1) - para cada processador a imagem está totalmente descrita em sua memória local. Com isto é evitada a concorrência durante o *traversal* da descrição da imagem. A replicação da descrição da cena nas memórias de cada processador proporciona um processamento de acoplamento desprezível.

2) - cada processador está conectado logicamente a uma partição da memória de quadro, e deve tratar isoladamente os *pixels* desta partição.

3) - cada processador poderá ter acesso a outra partição além da sua própria, bastando para isto endereçar convenientemente. O acesso a outra partição implica em concorrência no barramento.

4) - as partições da memória de quadro estão dispostas de forma intercalada, formando uma pegada de 5 *pixels* por 4 linhas. Os 5 *pixels* na linha da pegada vêem de um mesmo barramento da VAM.

Para resolver as diversas etapas do processamento para obtenção da imagem final na tela é importante que se defina uma estratégia global para o problema. Considera-se que método *ray-casting* é uma boa estratégia, pois o particionamento do conjunto de raios que chegam ao ponto de vista do observador permite uma distribuição de trabalhos autônomos e sem sobreposição de processamento.

6.2 - A Geração da Imagem com Realismo

Faz-se em seguida uma breve descrição de como os algoritmos de algumas etapas da geração de imagem podem ser tratados utilizando-se das características da UGR.

6.2.1 - Estrutura de Dados.

É recomendado que se desenvolva uma biblioteca de objetos tri-dimensionais descritos por uma técnica de fronteiras. Na construção dos dados descritivos de cada objeto é importante que se definam parâmetros tais como vetor-normal a superfície, coeficiente de reflexão, refração, radiação de cada elemento da

superfície, a descrição temporal no espaço tridimensional do ponto de vista e outros mais que se configurem necessários.

A existência da biblioteca de objetos possibilita um pré-processamento de todos os dados inerentes ao objeto, diminuindo assim a complexidade do processamento em cada processador e aumentando a eficiência do sistema.

6.2.2 - Tratamento Geométrico.

O tratamento geométrico é feito em paralelo pelos diversos processadores elementares.

6.2.3 - *Z buffer.*

O registro de cada *pixel* na memória de quadro inclui um campo de definição de profundidade. Isto é utilizado para a comparação da posição da informação já existente com uma nova informação a ser registrada. De forma que a operação de escrita na posição de memória de quadro correspondente, é feita num ciclo *read-modify-write*. Durante este ciclo o valor *z* já registrado é comparado com o valor *z* do novo *pixel*; se este novo valor for menor, ou seja mais próximo da tela que o anterior, ele é escrito na posição de memória correspondente; caso contrário, ela conserva os valores anteriores de cor e profundidade do *pixel*. A sequência de operações é definida no controlador de acesso à partição da memória de quadro.

6.2.4 - Composição e *Aliasing*.

Para cada *pixel* é definido um campo de informação de composição, de forma que uma vez definidos os valores RGB para um *pixel*, é também registrado na memória de quadro o percentual de ocupação destes valores no *pixel* (valor Alfa). A cada nova escrita na memória um novo valor alfa é definido para o *pixel*. Isto é feito da seguinte forma, durante a operação de escrita (*read-modify-write*) o campo alfa é comparado com o valor anteriormente registrado na posição de memória correspondente ao *pixel*. Em função destes dois valores , as cores (que se escreve e a anterior) são misturadas (*blend*), definindo assim o novo valor de cor para o *pixel*. Esta mistura de cores é util principalmente nas bordas de primitivas ajudando na eliminação dos efeitos de *aliasing*. A seqüência de operações para esta composição é definida no controlador de

acesso à partição da memória de quadro.

6.3 - Análise da UGR segundo os aspectos levantados por Whitman [Whit 90].

Para encerrar, a seguir é feita uma série de considerações sobre como a UGR trata os aspectos de desenvolvimento de um aplicativo gráfico sobre uma topologia multiprocessadora. Estes aspectos foram levantados em [Whit 90] e reproduzidos na seção 4.5 deste trabalho.

6.3.1 - O Aproveitamento de Coerência

A coêrencia poderá ser detectada na descrição da imagem, neste caso o próprio processador poderá considerar a existência de coerência e tratá-la localmente. Lembrar que inerentemente cada processador trata diretamente todos os *pixels* com mesmo índice MOD 5 na linha de mesmo índice em MOD 4. Como exemplo, o processador que trata do *pixel* de coordenada (x,y) 0,0, trata os *pixels* 5,0, 10,0, ..., 0,4, 10,4, ..., etc. Entretanto a VAM permite que um mesmo processador seja responsável, através de uma alocação lógica (por *software*), de uma região da tela, ou de toda uma linha horizontal, vertical, ou diagonal. A possibilidade de ter acesso a outros *pixels* além daqueles que inerentemente estariam alocados a um processador, é uma característica importante na definição desta bancada.

6.3.2 - A Utilização dos Processadores e o Equilíbrio da Carga de Processamento.

A alocação dos processadores em partições intercaladas, contribui fortemente para um equilíbrio de carga de processamento [MoFu 90]. Por isto, esta é a alocação natural proposta para os processadores nesta bancada. Numa configuração completa, vinte processadores tratariam de forma autonôma vinte partições intercaladas da memória de quadros.

6.3.3 - A Comunicação entre Processos (Processadores).

O maior desempenho da topologia proposta é conseguido numa estrutura de processamento fracamente acoplada, com trabalhos autonômos em cada processador.

Entretanto a proposta prevê a possibilidade de comunicação entre os processado-

res de duas formas: uma através do barramento COMUM, e outra através da VAM.

Através do barramento COMUM, cada processador pode escrever mensagens na memória local de um outro processador, ou em *broadcasting* para todos os processadores, utilizando este barramento.

Pela VAM, a forma de ligação dos processadores às partições de memória de quadro permite que os dados gerados por um processador possam ser recolhidos por outro processador.

6.3.4 - A Decomposição do algoritmo.

Os algoritmos a serem desenvolvidos são otimizados se for considerado o fraco acoplamento entre os processadores. Por exemplo, as técnicas de *ray-casting* podem ser adotadas distribuindo os raios entre os diversos processadores.

6.3.5 - A Escalabilidade do Processamento.

Dentro de um ambiente acadêmico, com as suas dificuldade de aquisição de recursos é importante definir uma estrutura de processamento que possa ser escalável, no sentido de que se possa começar com poucos processadores, e na medida da disponibilidade de recursos, novos processadores são incorporados ao sistema.

6.3.6 - A Programabilidade.

A facilidade de programação de aplicativos impôs à proposta da bancada a utilização de processadores elementares baseados em processadores comerciais. Portanto evitou-se o desenvolvimento de componentes de processamento específicos. Na disponibilidade destes, eles poderão ser integrados a estrutura de processamento sem maiores dificuldades.

6.4 - Resumo e Comentários.

Este capítulo apresentou como é possível integrar duas aplicações distintas na bancada da UGR. Uma aplicação foi a Visualização de Volumes e outra a Geração de Imagens com Realismo. Com isto pretende-se caracterizar a UGR

como uma bancada para aplicações em Computação de Imagem.

Dentro da aplicação Visualização de Volumes, foram analisadas duas alternativas de integração. Na primeira foi considerado um processador dedicado a esta aplicação, e na segunda alternativa foi considerada a integração de dispositivos comerciais.

A utilização de processadores de uso específico integrados à UGR para a solução do problema de Visualização de Volumes foi exemplificada na seção 6.1, citando-se o processador VERVE [Knit 93]. É importante enfatizar que para qualquer processador de uso específico em computação de imagem, a sua integração à UGR se faz da mesma forma que o VERVE, ou seja, faz-se necessário desenvolver um circuito de interface com a VAM.

O capítulo faz ainda uma análise de como os problemas de desenvolvimento de aplicativo gráfico são considerados na UGR.

Capítulo 7

Conclusões e Comentários Finais

7.1 - Conclusões.

Experimentar uma idéia nova contando com maior velocidade na montagem do experimento e dispondo de equipamento adequados e eficientes, nem sempre é viável ao pesquisador acadêmico, independente de sua área de atuação. Às vezes, estas duas condições só podem ser atendidas, através de um aumento exorbitante nos custos do experimento que, frequentemente, o torna proibitivo.

Por outro lado quando um pesquisador dispõe de equipamentos flexíveis, ou seja de equipamento facilmente adaptável a experimentos diversos, seus custos de pesquisa se tornam baixos e sua produtividade aumenta.

Este último enfoque influenciou fortemente a escolha pelo projeto de uma bancada de exibição gráfica em confronto à opção de projeto de uma estrutura rígida de multiprocessamento gráfico.

Quando se examina a especificação da bancada definida destacam-se dois itens, a VAM (Via de Acesso Múltiplo) e a estrutura da memória de quadro.

A VAM permite uma ligação otimizada entre os processadores elementares e a memória de quadro. Ela permite que diversos *pixels* possam ser acionados simultaneamente, agilizando a transferência com a memória de quadro. A sua simplicidade a torna bastante flexível para aceitar a ligação de módulos prontos (comerciais), bastando para isto a implementação de uma interface específica. Outro aspecto importante, principalmente dentro de um ambiente acadêmico é a escalabilidade da estrutura, que permite que os módulos processadores possam ser anexados, à medida que estejam disponíveis. Todos os *pixels* poderão ser endereçados, se necessário, por um único módulo processador (como solução inicial, considerando os custos de desenvolvimento).

A estrutura da memória de quadro da unidade UGR atinge uma taxa de transferência de primitivas gráficas da ordem de 300 000 polígonos (QA) por segundo. Esta taxa é viabilizada porque a memória de quadro é formada por um arranjo de nós inteligentes de memória de vídeo, onde cada um possui um

controlador e memória para monitorar 1/20 da resolução total da tela. Os nós são agrupados em cinco blocos de quatro, sendo a informação para a tela retirada destes blocos de forma intercalada.

A versatilidade da bancada foi mostrada através das análises e exemplos do capítulo 6. Ali, apesar de não ser considerado o tipo de processador montado sobre a bancada, foi possível descrever toda a sistemática de processamento e distribuição de carga entre processadores, a estrutura de dados do problema enfocado e sugerir algoritmos a serem implementados nos processadores. Isto demonstra uma grande independência entre a bancada e o tipo de processador utilizado. Por outro lado, a estrutura de memória de quadros é dedicada à exibição gráfica para a qual é otimizada.

Assim o objetivo mais amplo de se definir um equipamento de auxílio ao pesquisador da área de exibição de imagens de cenas tridimensionais, tanto no experimento de algoritmos quanto na avaliação da eficiência de processadores, foi alcançado.

7.2 - Comentários Finais

Este trabalho iniciou com a realização de estudos sobre a aplicação — Geração de Imagens com Realismo. Para este estudo foram realizados trabalhos de programação e de análise de sistemas de programas de síntese de imagens.

Dentre outras experiências necessárias à definição deste trabalho, destacam-se aquelas relativas ao desenvolvimento de projetos de sub-sistemas computacionais gráficos realizados junto ao CPqD-Telebrás (1986-1987) e na Universidade de Bristol (1989). Na Universidade de Bristol foi possível participar do projeto de um terminal de vídeo gráfico de alta resolução utilizando o circuito de controle de vídeo G300 em lançamento (na época) pela firma INMOS [Oliv 89]. No CPqD-Telebrás, dentro do projeto do Processador Preferencial, além dos estudos relativos ao projeto de um circuito de saída gráfico para o sistema PP (placa PPGRF), merece destaque a experiência adquirida em participar do projeto de um sistema completo de computador de alto desempenho [OBZi 89]. Os estudos realizados sobre circuitos de exibição de vídeo para os projetos da placa PPGRF e em Bristol resultaram no texto do capítulo 3.

O capítulo 4 resultou dos estudos realizados sobre as diversas arquiteturas de processamento e geração de imagens. Aquele capítulo resume o tutorial apresentado no XXIV Congresso Nacional de Informática em São Paulo no mês

de setembro de 1991 sobre Arquiteturas de Alto Desempenho para a Geração de Imagens com Realismo [Oliv 91].

Assim, com base nestas experiências e nos estudos realizados, este trabalho resultou na proposição de uma bancada de alto desempenho para a geração de imagens com realismo. A realização da proposta deu-se através do seu projeto descrito no capítulo 5, onde estão detalhados os requisitos e a solução obtida, bem como da realização de simulações da memória de quadro, da VAM e de uma interface para acesso a VAM para um processador genérico. Para estas simulações foi utilizada como linguagem de descrição o VHDL, conforme relatório [Oliv 95].

A principal contribuição do presente trabalho foi o projeto de uma arquitetura para geração de imagem com realismo com características de alto desempenho e escalabilidade de processamento. Neste contexto foi projetada uma bancada inédita com facilidade de conexão a um arranjo de processadores (de prateleira ou específico) voltada à área de Computação de Imagem. Este projeto está validado através de [Oliv 95].

7.2.1 - Próximos Trabalhos.

A experiência adquirida em projeto de sistema digital no desenvolvimento do presente projeto sugere que futuros trabalhos de desenvolvimento sobre esta arquitetura serão grandemente facilitados.

Entre outros prevê-se :

- Implementação de um arranjo de processadores conectável a VAM. Antevê-se como opções de processamento as tecnologias DSP ou Micro-Sparc.

- aprimoramento do processamento de composição da Memória de Quadro.

Capítulo 8

Bibliografia

8.1 - Bibliografia Selecionada

[ABMa 88] B.Apgar, B.Bersack & A.Mammen - "A Display System for Stellar Graphics Supercomputer Model GS1000" - *Computer Graphics* - vol.22 nr.4 - aug 1988 - pp.:255-262.

[AEZa 87] D.J.Allerton, J.D.Evemy & E.J.Zaluska - "Real Time Scan Line In-Fill" - *Proc. Eurographic 87* - 1987 - pp.:209.

[Agat 86] M.Agate *et alii* - "A Multiple Application Graphics Integrated Circuit - MAGIC" - *Proceeding Eurographic 86* - 1986 - pp.:63-77.

[Akel 89] Kurt Akeley - "The Silicon Graphics 4D/240GTX Superworkstation" - *IEEE Computer Graphics & Applications* - vol.9 nr.4 - july 1989 - pp.:71-89.

[Akel 93] Kurt Akeley - "RealityEngine Graphics" - *Computer Graphics Proceedings, Annual Conference Series* - 1993 - pp.:109-116.

[AkJe 88] Kurt Akeley & T. Jermoluk - "High Performance Polygon Rendering" - *Computer Graphics* - vol.22 nr.4 - aug. 1988 - pp.:239-246.

[Aman 87] J. Amanatides - "Realism in Computer Graphics: A Survey" - *IEEE Computer Graphics & Applications* - vol.7 nr.1 - january 1987 - pp.:44-56.

[AmWo 87] J. Amanatides & A. Woo - "A Fast Voxel Transversal Algorithm for Ray-Tracing" - *Proceeding Eurographic 87* - 1987 - pp.:3.

[Apff 89] J.M.Apffel *et alii* - "An Architecture for Region Boundary Extraction in Raster Scan Images Suitable for VLSI Implementation" - *Machine Vision and Applications* - 2 - Springer-Verlag ~ - New York - 1989 - pp.:193-214.

[Appe 68] A.Appel - "Some Techniques for Shading Machine Rendering of Solids" - *AFIPS 1968 Spring Joint Computer Conf. 32* - 1968 - pp.:37-45.

[ArKi 87] J. Arvo & D. Kick - "Fast Ray Tracing by Ray Classification" - *Computer Graphics* - vol.21 nr.4 - july 1987 - pp.:55.

[Arms 89] James R. Armstrong - *Chip-Level Modeling with VHDL* - Prentice Hall ~ - Englewood Cliffs, NJ - 1989.

[AWGr 78] P.Atherton, K.Weiler & D. Greenberg - "Polygon Shadow Generation" - *Computer Graphics* - vol.12 nr.3 - august 1978 - pp.:275-281.

[BaBr 82] D.H. Ballard and C.M. Brown - *Computer Vision* - Prentice-Hall, Inc. ~ - Englewood Cliffs, New Jersey - 1982.

[BaCh 90] E. Bahar & S. Chakrabarti - "Full-Wave Theory Applied to Computer-Aided Graphics for 3D Objects" - *IEEE Computer Graphics & Applications* - vol.7 nr.7 - july 1987 - pp.:46.

[Bark 90] A.C.Barkans - "High Speed High Quality Antialiased Vector Generation" - *Computer Graphics* - vol.24 nr.4 - Aug.1990 - pp.:319-326.

[BaWi 90] D. Baum & J.M. Winget - "Real Time Radiosity Through Parallel Processing and Hardware Acceleration" - *Computer Graphics* - vol.24 nr.2 - march 1990 - pp.:67-75.

[BBDM 85] C.Bouville, R. Brusq, J.L.Dubois, I.Marchal - "Generating High Quality Picture by Ray Tracing" - *Computer Graphics Forum* - 4 - 1985 - pp.:87.

[BBPr 90] D.Badouel, K. Bouatouch & T. Priol - "Ray Tracing on Distributed Memory Parallel Computers: Strategies for distributing computation and data" - *Siggraphs 90 Course Notes, course 28* ~ - Dallas - aug. 1990 - pp.:185-198.

[BBPr 94] Didier Badouel, Kadi Bouatouch and Thierry Priol - "Distributed Data and Control for Ray Tracing in Parallel" - *IEEE Computer Graphics & Applications* - vol.14 no. 4 - july 1994 - pp.:69-77.

[Bera 89] P.C.Berardi - "Tese de Mestrado" - *Projeto e Implementação de Hardware e Software para uma Placa Gráfica de Média/Alta Resolução com Capacidade de Processamento Local* - CPG-FEE-UNICAMP ~ - Campinas S.P. - 1989.

[Berg 86] P.Bergeron - "A General Version of Crow's Shadow-volumes" - *IEEE Computer Graphics & Applications* - vol.6 nr.9 - september 1986 - pp.:17-28.

[BFPa 86] K.S.Booth, D.R.Forsey & A.W.Paeth - "Hardware Assistance for Z-Buffer Visible Surface Algoritms" - *IEEE Computer Graphics & Applications* - vol.6 nr.11 - nov.1986 - pp.:31-39.

[Blak 87] E.H.Blake - "A Metric For Computing Adaptative Detail In Animation Scenes Using Object-Oriented Programming" - *Proceeding Eurographic 87* - 1987 - pp.:295.

[Blin 92] James F. Blinn - "The World of Digital Video" - *IEEE Computer Graphics & Applications* - september 1992 - pp.:107-112.

[Blin 78] Blinn, James F. and Martin E. Newell - "Simulation of Wrinkled Surfaces" - *Proceeding of SIGGRAPH'78 in Computer Graphics* - vol.12 no. 3 - 1978 - pp.:286-292.

[BlNe 76] Blinn, James F. and Martin E. Newell - "Texture and Reflection in Computer Generated Images" - *Communications of the ACM* - vol.19 no. 10 - 1976 - pp.:542-547.

[BoKe 70] Bouknight, W.J. and K.C.Kelley - "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources" - *Proceedings of AFIPSSpring Joint Computer Conference* - Vol.36 - AFIPS Press ~ - Reston, Va. - 1970 - pp.:1-10.

[Bord 89] B.S.Borden - "Graphics Processing on a Graphics Supercomputer" - *IEEE Computer Graphics & Applications* - vol.9 nr.4 - july 1989 - pp.:56-62.

[Brag 85] C.A.M.Braga - "Trabalho de Final de Curso" - *Projeto de um Terminal Colorido* - DEE-FEC-UNICAMP ~ - Campinas SP - 1985.

[BrKl 85] W. F. Bronsvoort & F. Klok - "Ray Tracing Generalized Cylinders" - *ACM Transaction on Graphics* - vol.4 nr.4 - october 1985 - pp.:291.

[Brok 85] Brooktree Corporation - "Application Note AN-3: Comparison of NTSC, PAL and SECAM Video Levels" - 1985.

[BuDe 89] J.Bu & E.F.Deprettere - "A VLSI System Architecture for High-Speed Radiative Transfer 3D Image Synthesis" - *Visual Computer* - 5 - Springer-Verlag - 1989 - pp.:121-133.

[BWJa 84] W. F. Bronsvoort, J. J. van Wijk & Frederik W. Jansen - "Two Methods for Improving the Efficiency of Ray Casting in Solid Modelling" - *Computer Aided Design* - vol.16 nr.1 - jan. 1984 - pp.:51.

[Carp 84] L.Carpenter - "The A-buffer, an Antialiased Hidden Surface Method" - *Computer Graphics* - vol.18,nr.3 - july 1984 - pp.:103-108.

[Catm 74] E.Catmull - "Doctoral Dissertation" - *A subdivision Algorithm for Computer Display of Curved Surfaces* - Univ. of Utah~ - Salt Lake City - 1974.

[Catm 75] E.Catmull - "Computer Display of Curved Surface" - *Proc. IEEE Conf. on Computer Graphics , Pattern Recognition and Data Structures*~ - Piscataway, N.J. - 1975 - pp.:11-17.

[CCHs 92] Ingrid Carlbom, Indranil Chakravarty & Willim M. Hsu - "SIGGRAPH'91 Workshop Report - Integrating Computer, Computer Vision & Image Processing in Scientific Applications" - *Computer Graphics* - Vol.26 nr.1 - January 1992 - pp.:8-17.

[ChRo 86] F.Charot, F.Rousee - "CSI - A processor for Image Synthesis" - *Proc.Eurographic 86* - 1986 - pp.:79-91.

[Clar 82] J. H. Clark - "The Geometry Engine: A VLSI Geometry System for Graphics" - *Computer Graphics* - vol 16 nr.3 - July 1982 - pp.:127-133.

[CoHa 94] Michael Cox and Paul Hanrahan - "A Distributed Snooping Algorithm for Pixel Merging" - *IEEE - PARALLEL & DISTRIBUTED TECHNOLOGY - Systems & Applications* - vol.2 nr.2 - 1994 - pp.:30-36.

[Croc 87] G.A.Crocker - "Screen-Area Coherence for Interactive Scanline Display Algorithms" - *IEEE Computer Graphics & Applications* - vol.7 nr.9 - september 1987 - pp.:10-17.

[CrOr 94] Thomas W. Crockett and Tobias Orloff - "Parallel Polygon Rendering for Message-Passing Architectures" - *IEEE - PARALLEL & DISTRIBUTED TECHNOLOGY - Systems & Applications* - vol.2 nr.2 - 1994 - pp.:17-28.

[Crow 77] F.C.Crow - "Shadow Algoritms for Computer Graphics" - *Computer Graphics* - vol.11,nr.2 1977 - 242-248.

[Crow 81] Crow, Franklin C. - "A Comparation of Antialiasing Techniques" - *IEEE Computer Graphics and Applications* - Vol.1 no.1 - jan. 1981 - pp.:40-48.

[CSLe 86] E.S.Cohen, E.T. Smith & A.I. Lee - "Constraint-Based Tiled Windows" - *IEEE Computer Graphics & Application* - vol.6 nr.5 - may 1986 - pp.:35-45.

[Curt 90] R.P.Curtis - "Developing the GX Graphics Accelerator Architecture" - *IEEE MICRO* - vol.10 nr.1 - february 1990 - pp.:44-54.

[CyBe 78] M.Cyrus & J.Beck - "Generalized Two and three Dimensional Clipping" - *Computer & Graphics* - vol.3 - 1978 - pp.:23-28.

[Deme 85] S.Demetrescu - "High Speed Image Rasterization Using Scan Line Acess Memories" - *Proc. 1985 Chapel Hill Conference on VLSI* - Computer Science Press - 1985 - pp.:pag. 221-243.

[DeNe 93] Michael F. Deering & Scott R. Nelson - "Leo: A System for Cost Effective 3D Shaded Graphics" - *Computer Graphics Proceedings, Annual Conference Series* - 1993 - pp.:101-108.

[DiSw 84] M.Dippé, J. Swensen - "An Adaptive Subdivision Algorithm & Parallel Architecture for Realistic Image Synthesis" - *Computer Graphics* - vol.18 nr.3 - july 1984 - pp.:149.

[Duff 85] T. Duff - "Compositing 3-D Rendered Images" - *Computer Graphics* - vol.19 nr.3 - 1985 - pp.:41.

[Dunc 90] R.Duncan - "A Survey of Parallel Computer Architecture" - *IEEE Computer* - vol.23 nr.2 - feb.1990 - pp.:5-16.

[DuWa 85] K.A.Duke & W.A.Wall - "A Professional Graphics Controller" - *IBM Systems Journal* - vol.24 nr.1 - 1985 - pp.:14-25.

[DWLG 92] Graham J. Dunett, Martin White, Paul F. Lister, Richard L. Grimsdale and France Glemot - "The Image Chip for High Performance 3D Rendering" - *IEEE Computer Graphics & Applications* - november 1992 - pp.:41-52.

[DWSD 88] Deering, M., S.Winner, B.Schediwy, C.Duffy and N. Hunt - "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics" - *SIGGRAPH'88* - 1988 - pp.:21-30.

[DyWh 87] S. Dyer & S. Whitman - "Vectorized Scan-line Z-buffer Rendering Algorithm" - *IEEE Computer Graphics & Applications* - vol.7 nr.7 - july 1987 - pp.:34-45.

[EIA 57] Eletronic Industries Association - "EIA RS170 - Eletrical Performance Standards Monochrome Television Studio Facilities" - november 1957.

[EIA 69] Eletronic Industries Association - "EIA RS343-A - Eletrical Performance Standards for High Resolution Monochrome Closed Circuit Television Camera" - september 1969.

[Ells 94] David Ellsworth - "A New Algorithm for Interactive Graphics on Multicomputers" - *IEEE Computer Graphics & Applications* - vol.14 no. 4 - july 1994 - pp.:33-40.

[Elvi 92] T.Todd Elvins - "A Survey of Algorithms for Volume Visualization" - *Computer Graphics* - Vol.26 nr.3 - August 1992 - pp.:194-201.

[FDFH 90] J.D.Foley, A.van Dam, S.K.Feiner and J.F.Hughes - *Computer Graphics - Principles and Practice* - Addison-Wesley - 1990.

[FFCa 82] Fournier, Alain, Don Fussel and Loren C. Carpenter - "Computer Rendering of Stochastic Models" - *Communications of the ACM* - Vol.25 no.6 - june 1982 - pp.:371-384.

[FGHS 85] Fuchs, Henry, Jack Goldfeather, Jeff P.Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks Jr., John G. Eyles and John Poulton - "Fast Spheres, Shadows, Textures, Transparencies and Image Enhancements in Pixel-Planes" - *Proceeding of SIGGRAPH'85 in Computer Graphics* - vol.19 no.3 - 1985 - pp.:111-120.

[FGRe 85] G.Frieder, D.Gordon & R.A.Reynolds - "Back-to-front Display of Voxel-Based Objects" - *IEEE Computer Graphics & Applications* - vol.5 nr.1 - january 1985 - pp.:52-60.

[Figu 95] Renato Jansen de Oliveira Figueiredo - "Tese de Mestrado" - *Síntese Comportamental de Circuitos Digitais Utilizando SDL* - DT-FEE-UNICAMP ~ - Campinas-SP - 1995.

[Flyn 66] Flynn, M.J. - "Very High-Speed Computers" - *Proceeding of IEEE* - vol.54 - December 1966 - pp.:1901-1909.

[FoKi 87] J.D.Foley & W.C.Kim - "Image Composition via Lookup Table Manipulation" - *IEEE Computer Graphics & Applications* - vol.7 nr.11 - november 1987 - pp.:26-35.

[FoVD 82] J.Foley & A.Van Dam - *Fundamental of Interactive Computer Graphics* - Addison-Wesley - 1982.

[FPIw 84] A. Fujimoto, C. G. Perrot & K. Iwata - "A 3-D Graphics Display System with Depth Buffer and Pipeline Processor" - *IEEE Computer Graphics & Applications* - june 1984 - pp.:11-23.

[FrBr 89] Edward H.Frank & W.Mitch Bradley - "The Sbus Specification rev.A" - Sun MicroSystem - 1989.

[FrKa 90] W.R.Franklin & M.S. Kankanhalli - "Parallel Object-Space Hidden Surface Removal" - *Computer Graphics* - vol.24 nr.4 - aug 1990 - pp.:87-93.

[FTDa 87] E.Fiume, D.Tsichritzis & L.Dami - "A Temporal Scripting Language for Object-Oriented Animation. " - *Proceeding Eurographic 87* - 1987 - pp.:283.

[Fuch 89] H. Fuchs, *et alii* - "A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories" - *Computer Graphics* - vol.23 nr.3 - july 1989 - pp.:79-88.

[FuPo 81] H.Fuchs & J.Poulton - "Pixel Planes: A VLSI-Oriented Design for a Raster Graphics Engine" - *VLSI Design* - vol.2 nr.3 - 1981 - pp.:20-28.

[FuRa 82] D.Fussel & B.D.Rathi - "A VLSI-Oriented Architecture for Real-Time Raster Display of Shaded Polygons" - *Proc. of Graphics Interface* - 1982 - pp.:373-380.

[Gard 84] Gardner, Geoffrey Y. - "Simulation of Natural Scenes Using Textured Quadric Surfaces" - *Proceedings of SIGGRAPH'84 in Computer Graphics* - vol.18 no.3 - July 1984 - pp.:11-20.

[GBWi 90] B.J.Garlick, D.R.Baum & J. M. Winget - "Interactive Viewing of Large Geometric Databases Using Multiprocessor Graphics Workstations" - *Siggraphs 90 Course Notes* ~ - Dallas - aug. 1990.

[GFBa 87] A.Goris, B. Fredrickson & H. Baeverstad - "A Configurable Pixel Cache for Fast Image Generation" - *IEEE Computer Graphics & Applications* - vol.7 nr.3 - march 1987 - pp.:24-32.

[GGAk 92] Karl Guttag, Robert J. Gove, and Jerry R. Van Aken - "A Single-Chip Multiprocessor For Multimedia: The MVP" - *IEEE Computer Graphics & Applications* - november 1992 - pp.:53-64.

[GGSS 89] N.Gharachorloo, S.Gupta, R.F.Sproull & I.E. Sutherland - "A characterization of Ten Rasterization Techniques" - *Computer Graphics* - vol.23 nr.3 - july 1989 - pp.:355-368.

[Ghar 88] N.Gharachorloo *et alii* - "Subnanosecond Pixel Rendeering with Million Transistor Chips" - *Proc. Siggraph'88 - Computer Graphics* - vol.22 nr.4 - aug 1988 - pp.:41-49.

[GhPo 85] N.Gharachorloo, C. Pottle - "SUPER BUFFER: A Systolic VLSI Graphics Engine for Real Time Raster Image Generation" - *Proc. 1985 Chapel Hill Conference on VLSI* - Computer Science Press - 1985 - pp.:pag. 285-305.

[GKBh 89] J.Grimes, Les Kohn e R. Bharadhwaj - "The Intel i860 64-bits Processor: A General Purpose CPU with 3D graphics Capabilities" - *IEEE Computer Graphics & Applications* - vol.9 nr.4 - july 1989 - pp.:85-94.

[Glas 84] A. S. Glassner - "Space Subdivision for Fast Ray Tracing" - *IEEE Computer Graphics & Applications* - vol.4 nr.10 - october 1984 - pp.:15-22.

[GoSa 87] J. Goldsmith & J. Salmon - "Automatic Creation of Object Hierarchies for Ray Tracing" - *IEEE Computer Graphics & Applications* - vol.7 nr.5 - may 1987 - pp.:14-20.

[Gour 71b] H. Gouraud - "Doctoral Dissertation" - *Computer Display of Curved Surfaces* - Univ. Utah ~ - Salt Lake City - 1971.

[Gour 71a] H. Gouraud - "Continuous Shading of Curved Surfaces" - *IEEE Transactions on Computers* - june 1971 - pp.:623-629.

[GoWi 77] R.C. Gonzalez and P. Wintz - *Image Processing* - Addison-Wesley Publishing Company ~ - Reading Massachusetts - 1977.

[GrPa 88] S.A.Green & D.J. Paddon - "An Extension of the Processor Farm Using a Tree Architecture" - *Computer Science Department CS-88-13* - University of Bristol - April 1988.

[GrPa 89a] S.A.Green & D.J. Paddon - "A highly Flexible Multiprocessor Solution for Ray Tracing" - *Computer Science Department TR-89-02* - University of Bristol - march 1989.

[GrPa 89b] S.A.Green & D.J. Paddon - "Exploiting Coherence for Multiprocessor Ray Tracing" - *Computer Science Department TR-89-03* - University of Bristol - April 1989.

[GSSu 81] S.Gupta, R.F.Sproull & I.E.Sutherland - "A VLSI Architecture for Updating Raster Scan Display" - *Computer Graphics* - vol.15 nr.3 - july 1981 - pp.:71-78.

[GTGr 84] Goral, Cindy M., Kenneth E. Torrance and Donald P.Greenberg - "Modeling the Interation of Light between Diffuse Surfaces" - *Proceeding of SIGGRAPH'84 in Computer Graphics* - Vol. 18 no.3 - July 1984 - pp.:213-222.

[HaAk 90] P. Haeberli & Kurt Akeley - "The Accumulator Buffer: Hardware Support for High-Quality Rendering" - *Computer Graphics* - vol.24 nr.4 - Aug.1990 - pp.:309-318.

[HaFo 93] Chandlee B. Harrell & Farhad Fouladi - "Graphics Rendering Architecture for a High Performance Desktop Workstation" - *Computer Graphics Proceedings, Annual Conference Series* - 1993 - pp.:93-100.

[HaGr 86] E.A.Haines & D.P.Greenberg - "The light Buffer: A Shadow-Testing Accelerator" - *IEEE Computer Graphics & Applications* - vol.6 nr.9 - september 1986 - pp.:6-16.

[HaGr 83] R.A.Hall & D.P.Greenberg - "A Testbed for Realistic Image Synthesis" - *IEEE Computer Graphics & Applications* - vol.3 nr.8 - november 1983 - pp.:10-20.

[HaSc 87] P.J.W.T. Hagen & H.J.Schouten - "Parallel Graphical Output From Dialog Cells" - *Proceeding Eurographic 87* - 1987 - pp.:101.

[HaSp 87] M. Hardwick & D. L. Spooner - "Comparation of Some Data Models for Engineering Objects" - *IEEE Computer Graphics & Applications* - vol.7 nr.3 - march 1987 - pp.:56-66.

[HCWh 94] Chuck Hansen, Tom Crockett and Scott Whitman - "Guest Editors' Introduction: Parallel Rendering" - *IEEE - PARALLEL & DISTRIBUTED TECHNOLOGY - Systems & Applications* - vol.2 nr.2 - 1994 - pp.:7.

[HeBa 86] D. Hearn & M.P. Baker - *Computer Graphics* - Prentice-Hall, INC. - 1986.

[Heck 86] P.S.Heckbert - "Survey of Texture Mapping" - *IEEE Computer Graphics & Applications* - vol.6 nr.11 - nov.1986 - pp.:56-67.

[HeHa 84] P. S. Heckbert & P. Hanrahan - "Beam Tracing Polygonal Objects" - *Computer Graphics* - vol.18 nr.3 - july 1984 - pp.:119.

[HHPT 92] Xiao D. He, Patrick O. Heynen, Richard L. Phillips, Kenneth E. Torrance, David H. Salesin & Donald P. Greenberg - "A Fast and Accurate Light Reflection Model" - *Computer Graphics* - vol.26 nr.2 - july 1992 - pp.:253-254.

[Hill 90] F.S.Hill Jr. - *Computer Graphics* - Mcmillan Publishing Company - 1990.

[Hita 89] Hitachi Europe Ltd - *Hitachi IC Memory -Data Book* - 1989.

[Horn 86] B.K.P. Horn - *Robot Vision* - The MIT Press~ - Cambridge, Massachusetts - 1986.

[HuRo 91] John P.Huber & Mark W.Rosneck - *Successful ASIC Design, the First Time Through* - Van Nostrand Reinhold~ - New York - 1991.

[HwBr 84] Kai Hwang & Fayé A. Briggs - *Computer Architecture and Parallel Processing* - McGraw-Hill Publising Company - 1984.

[IBM 84] International Business Machines Corporation - "IBM PC-AT Technical Reference" - International Business Machines Corporation - 1984.

[Inte 86] Intel Coorporation - *82786 User's Manual* - 1986.

[Jans 85] F.W.Jansen - "A CSG List Priority Hidden Surface Algorithm" - *Proceeding Eurographics'85* - Elsevier Science Publishers B.V. (North-Holland) - 1985 - pp.:51.

[JaSu 86] F.W.Jansen & R.J.Sutherland - "Display of Solid Models with a Multi-Processor System" - *Proceeding Eurographic 87* - 1986 - pp.:377.

[Jeff 85] T. Jefferry - "THE uPD7281 PROCESSOR" - *BYTE* - november 1985 - pp.:237.

[JGMH 88] Kenneth I. Joy, Charles W. Grant, Nelson L. Max & Lansing Hatfield - "Tutorial - Computer Graphics: Image Synthesis" - *SIGGRAPH'88* - 1988.

[JoLu 94] Jonas Gomes e Luiz Velho - *Computação Gráfica: Imagem* - Instituto de Matemática Pura e Aplicada - IMPA / Soc. Brasileira de Matemática - SBM - Série Computação e Matemática. - 1994.

[KaBa 88] A.Kaufman & R. Bakalash - "Memory and Processing Architecture for 3D Voxel-Based Imagery" - *IEEE Computer Graphics & Applications* - vol.8 nr.6 - november 1988 - pp.:10-23.

[KaGr 78] D.S.Kay & D.P.Greenberg - "Transparency for Computer Synthesized Images" - *Computer Graphics (Siggraphs'79 proc.)* - vol.13 nr.2 - august 1979 - pp.:158-164.

[Kaji 83] J. T.Kajika - "New Techniques for Ray Tracing Procedurally Defined Objects" - *ACM Transaction on Graphics* - vol.2 nr.3 - july 1983 - pp.:161.

[KaKa 86] T. L. Kay & J. T. Kajika - "Raytracing Complex Scenes" - *ACM* - vol.20 nr.4 - 1986 - pp.:269.

[Kauf 87] A.Kaufman - "An Algorithm for 3.D. Scan Convertion Of Polygons" - *Proceeding Eurographic 87* - 1987 - pp.:197.

[Kauf 86] A.Kaufman - "Memory Organization for a CUBIC FRAME BUFFER" - *Proceeding Eurographic 86* - 1986 - pp.:93-100.

[KiVo 90] D.Kirk & D.Voorhies - "The Rendering Architecture of the DN10000VS" - *Computer Graphics* - vol.24 nr.4 - Aug.1990 - pp.:299-307.

[Knit 93] Günter Knittel - "Verve - Voxel Engine for Real-time Visualization and Examination" - *Eurographics'93* - vol.12 nr.3 - 1993 - pp.:37-48.

[KoBh 92] Konstantinos Konstantinides & Vasudev Bhaskaran - "Monolithic Architecture for Image Processing and Compression" - *IEEE Computer Graphics & Applications* - november 1992 - pp.:75-87.

[KoMa 89] L. Kohn & N. Margulis - "Introducing the Intel i860 64-bit Microprocessor" - *IEEE MICRO* - vol.9 nr.4 - august 1989 - pp.:15-30.

[KWGo 92] Michel Kelley, Stephanie Winner & Kirk Gould - "A Scalable Hardware Render Accelerator using a Modified Scanline Algorithm" - *Computer Graphics* - Vol.26 nr.2 - July 1992 - pp.:241-248.

[Levo 88] Marc Levoy - "Display of Surfaces from Volume Data" - *IEEE Computer Graphics & Applications* - vol.8 no. 5 - may 1988 - pp.:29-37.

[LiBa 83] Y.D.Liang & B.Barsky - "An Analysis and Algorithm for Polygon Clipping" - *CACM* - vol.26 - 1983 - pp.:868-877.

[LiBa 84] Y.D.Liang & B.Barsky - "New Concept and Method for Line Clipping" - *ACM Transaction on Graphics* - 3(1) - jan.84 - pp.:1-22.

[Loga 90] Luis A. Logatti - "Tese de Mestrado" - *Linhas e Superfícies Escondidas: Taxonomia e Proposta de Implementação VLSI de um Algoritmo Z-Buffer* - DCA/FEE/Unicamp - 1990.

[MaCo 94] Paul Mackerras and Brian Corrie - "Exploiting Data Coherence to Improve Parallel Volume Rendering" - *IEEE - PARALLEL & DISTRIBUTED TECHNOLOGY - Systems & Applications* - vol.2 nr.2 - 1994 - pp.:8-16.

[Mamm 89] A.Mammen - "Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique" - *IEEE Computer Graphics & Applications* - vol.9 nr.4 - july 1989 - pp.:43-55.

[Mamm 91] Alaide Pellegrini Mammana - "Mostradores para Imagens de Alta Resolução" - *Estudo sobre TV de Alta Definição - X* - Presidência da República - 1991.

[Mand 82] B.Mandelbrot - *The Fractal Geometry of Nature* - W.H.Freeman ~ - San Francisco - 1982.

[Maso 87] W. A. Mason - "In depth - Workstation Technology / Distributed Processing: The State of Art" - *BYTE* - november 1987 - pp.:291.

[MaSt 89] G.L.Marchant & M.Stephenson - "A set of Benchmarks for Evaluating Engineering Workstations" - *IEEE Computer Graphics & Applications* - vol.9 nr.3 - may 1989 - pp.:29-33.

[MaTh 87] N.Magnenat-Thalmann & D.Thalmann - "An Indexed Bibliography on Image Synthesis" - *IEEE Computer Graphics & Applications* - vol.7 nr.8 - august 1987 - pp.:27-38.

[Max 90] N.L.Max - "Antialiasing Scan-Line Data" - *IEEE Computer Graphics & Applications* - vol.10 nr.1 - january 1990 - pp.:18-30.

[MCEF 94] Steven Molnar, Michel Cox, David Ellsworth and Henry Fuchs - "A Sorting Classification of Parallel Rendering" - *IEEE Computer Graphics & Applications* - vol.14 no. 4 - july 1994 - pp.:23-32.

[Ment 89] Mentor Graphics Co. - *An Introduction to Digital Simulation* - Mentor Graphics Co. - 1989.

[MEPo 92] Steve Molnar, John Eyles & John Poulton - "PixelFlow: High-Speed Rendering Using Image Composition" - *Computer Graphics* - vol.26 nr.2 - july 1992 - pp.:231-240.

[MoFu 90] Steven Molnar and Henry Fuchs - "Advance Raster Graphics Architecture" - *Capítulo 18 de [FDFH 90]* - Addison-Wesley - 1990.

[Mora 81] H. Moravec - "3D Graphics and the Wave Theory" - *Computer Graphics* - vol.15 nr.3 - august 1981 - pp.:289.

[MoRe 87] C. Montani & M. Re - "Vector & Raster Hidden-Surface Removal Using Parallel Connected Stripes" - *IEEE Computer Graphics & Applications* - vol.7 nr.7 - july 1987 - pp.:14-23.

[MPHK 94] Kwan-Liu Ma, James S. Painter, Charles D. Hansen and Michael F.Krogh - "Parallel Volume Rendering Using Binary-Swap Compositing" - *IEEE Computer Graphics & Applications* - vol.14 no. 4 - july 1994 - pp.:59-68.

[Mulm 89] K. Mulmuley - "An Efficient Algorithm for Hidden Surface Removal" - *Computer Graphics* - vol.23 nr.3 - july 1989 - pp.:379-388.

[Myer 75] A.J.Myers - "An Efficient Visible Surface Program" - *Report to the NFS* - Ohio State University Computer Graphics Research Group - july 1975.

[Naka 89] E.Nakamae *et alii* - "Compositing 3D Images with Antialiasing and Various Shading Effects" - *IEEE Computer Graphics & Applications* - vol.9 nr.2 - march 1989 - pp.:21-29.

[NeSp 79] W.M. Newman & R.F. Sproull - *Principles of Interactive Computer Graphics* - MCGRAW--HILL Book Company - 1979.

[NeSp 79] Newman, W.M. nd R.F.Sproull - *Principles of Interactive Computer Graphics* - McGraw-Hill Publishers ~ - New York,N.Y. - 1979.

[Newm 94] Ulrich Newmann - "Communication Costs for Parallel Volume-Rendering Algorithms" - *IEEE Computer Graphics & Applications* - vol.14 no. 4 - july 1994 - pp.:49-58.

[Nico 88] J.D. Nicoud - "VIDEO RAMs: Structure and Application" - *IEEE-MICRO* - february 1988 - pp.:8-27.

[NIMT 84] H. Niimi, Y. Imai, M. Murakami, S. Tomita & H. Hagiwara - "A Parallel Processor for Tree-Dimensional Color Graphics" - *Computer Graphics* - vol 18 nr.3 - july 84 - pp.:67-76.

[NNSa 72] M.E.Newell, R.G.Newell & T.L.Sancha - "A new approach to the Shaded Picture Problem" - *Proc. ACM National Conference* - 1972 - pp.:443-450.

[OBZi 89] J.R.Oliveira, C.A.M.Braga & J.H.Zilberberg - "Estações de Trabalho Baseadas no Sistema PP" - *Proc. SIBGRAPI 89~* - A.Lindoia - 1989 - pp.:543-548.

[OeGr 90] R.R.Oehler & R.D.Groves - "IBM RISC System/6000 Processor Architecture" - *IBM Journal of Research and Development* - vol 34 nr.1 - jan.1990 - pp.:23-36.

[Oliv 83] J.R. de Oliveira - "Topicos 83 - Projeto de um Terminal de Vídeo" - *Notas de Curso Tópicos em Engenharia de Computação EA098* - DEE-FEC-UNICAMP - 1983.

[Oliv 89] J.R. de Oliveira - "Relatorio de Viagem a Bristol" - *Relatório Técnico DCA 09/91* - DCA-FEE-UNICAMP - 1989.

[Oliv 91] J.R. de Oliveira - "Arquiteturas de Alto Desempenho para Computação de Imagem" - *Notas de Tutorial* - DCA-FEE-UNICAMP - 1991.

[Oliv 95] J.R. de Oliveira - "Descrição da MQ/UGR e VAM em VHDL e simulação" - *Relatorio Técnico DCA 02/95* - DCA-FEE-UNICAMP - 1995.

[Pack 89] J.Packer - "Exploiting Concurrency: A Ray Tracing Example" - *Inmos Technical note 7* - 1989.

[Peac 85] Peachey, Darwyn R. - "Solid Texturing of Complex Surfaces" - *Proceeding of SIGGRAPH'85 in Computer Graphics* - vol.19 no.3 - 1985 - pp.:279-286.

[PEMF 92] John Poulton, John Eyles, Steven Molnar and Henry Fuchs - "Breaking the Frame-Buffer Bottleneck with Logic-Enhanced Memories" - *IEEE Computer Graphics & Applications* - november 1992 - pp.:65-74.

[Perl 85] Perlin, Ken - "An Image Synthesizer" - *Proceeding of SIGGRAPH'85 in Computer Graphics* - vol.19 no.3 - july 1985 - pp.:287-296.

[Perr 91] Douglas L. Perry - *VHDL* - McGraw-Hill International Editions - 1991.

[Phon 75] B.T. Phong - "Illumination for Computer Generated Pictures" - *Communication of the ACM* - vol.18 nr.6 - june 1975 - pp.:311-317.

[PiGa 87] S. Pieper & O. Garbe - "200-MHz Video FIFO Buffer Juggles Multiple Windows" - *Eletronic Design* - February 5, 1987.

[Plae 87] M. Plaehn - "In depth - Workstation Technology / PHIGS: Programmers Hierarchical Interactive Graphics Standard" - *BYTE* - november 1987 - pp.:275-286.

[PlBa 85] D. J. Plunkett & M. J.Bailey - "The Vectorization of a Ray-Tracing Algorithm for Improved Execution Speed" - *IEEE Computer Graphics & Applications* - august 1985 - pp.:52-60.

[PoDu 84] T.Porter & T. Duff - "Compositing Digital Images" - *Computer Graphics* - vol.18 nr.3 - 1984 - pp.:253-259.

[PoHo 89] M. Potmesil, E.M. Hoffert - "The Pixel Machine A Parallel Image Computer" - *Computer Graphics* - vol.23 nr.3 - july 1989 - pp.:69-78.

[PuKa 87] R. Pulleyblank & J. Kapenga - "The Feasibility of a VLSI Chip for Ray Tracing Bicubic Patches" - *IEEE Computer Graphic & Applications* - march 1987 - pp.:33-44.

[PZLi 87] Q. Peng, Y. Zhu & Y. Liang - "A fast Ray Tracing Algorithm Using Space Indexing Techniques" - *Proceeding Eurographics'87* - Elsevier Science Publishers B.V. (North-Holland) - 1987 - pp.:11-23.

[RhWi 89] D. Rhoden, C. Wilcox - "Hardware Acceleration for Window Systems" - *Computer Graphics* - vol.23 nr.3 - july 1989 - pp.:61-68.

[RiAr 94] Philippe Ris and Didier Arqués - "Parallel ray tracing based upon a multilevel topological knowledge acquisition of the scene" - *Eurographics '94* - 1994 - pp.:C-221/C-232.

[Robi 87] P. Robinson - "In depth - Workstation Technology / A World of Workstations" - *BYTE* - november 1987 - pp.:251-264.

[Roge 85] D. Roger - *Procedural Elements for Computer Graphics* - McGraw-Hill - 1985.

[RoKa 76] A. Rosenfeld and A.C. Kak - *Digital Picture Processing* - Academic Press~ - New York, New York - 1976.

[RoRe 86] J.R. Rossignac & A. A.G. Requicha - "Depth-Buffering Display Techniques for Constructive Solid Geometry" - *IEEE Computer Graphics & Applications* - vol.6 nr.9 - september 1986 - pp.:29-39.

[Roth 82] S. D. Roth - "Ray Casting for Modeling Solids" - *Computer Graphics and Image Processing* - nr.18 - 1982 - pp.:109-144.

[ScCl 88] B.O.Schneider & U.Claussen - "PROOF: An Architecture for Rendering in Object Space" - Universitat Tubigen - 1988.

[ScGe 86] R. W. Scheifler & J. Gettys - "The X Window System" - *ACM Transactions on Graphics* - vol.5 nr.2 - April 1986 - pp.:79-109.

[Schn 88] B.O. Schneider - "A Processor for an Object-Oriented Rendering System" - *Computer Graphics Forum* - 7 - North-Holland - 1988.

[Schu 69] R.A.Schumacker *et alii* - "Study for Applying Computer Generated Images to Visual Simulation" - *Tech.Rep. AFHRI TR-69-14* - US Air Force Human Resources Lab - 1969.

[ScSt 93] Andreas Schilling Wolfgang Straβer - "EXACT: Algorithm and Hardware Architecture for an Improved A-Buffer" - *Computer Graphics Proceedings, Annual Conference Series* - 1993 - pp.:85-91.

[SDDa 89] R.Storer, A.W.G.Duller & E.L.Dagless - "Image Generation with an Associative Processor Array" - *publicado em: New Trends in Computer Graphics ed.:N.Magnenat-Thalmann & D.Thalmann* - Springer-Verlag~ - Geneva - 1988 - pp.:150-159.

[Sieg 92] Howard Jay Siegel - *Interconnection Networks for Large-Scale Parallel Processing - Theory and Case Study* - Mcgraw-Hill Internation Editions - 1992.

[Sign 81] Signetics/Philips - "VME bus Specification Manual" - Signetics/Philips - 1981.

[SKWF 92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran & Paul Haeberli - "Fast Shadows and Lighting Effects Using Texture Mapping" - *Computer Graphics* - vol.26 nr.2 - july 1992 - pp.:249-252.

[SSSc 74] I. Sutherland, R.F. Sproull & R.A. Schumacker - "A Characterization of Ten Hidden-Surface Algorithms" - *Computing Survey* - vol.6 nr.1 - march 1974 - pp.:1-55.

[SSUl 85] R. F. Sproull, W.R.Sutherland & M.K. Ullner - *DEVICE-INDEPENDENT GRAPHICS With Examples from IBM Personal Computers* - McGRAW-HILL Book Company - 1985.

[Stae 86] J.Staerk - "The Integrated Display Controller (IDC) for VLSI-Workstation" - *Proceeding Eurographic 86* - 1986 - pp.:279-291.

[Ster 87] H. L. Stern - "In depth - Workstation Technology / Comparation of Window Systems" - *Byte* - november 1987 - pp.:261-274.

[STNa 87] M. Shinya, T.Takahashi & S. Naito - "Principles and Applications of Pencil Tracing" - *Computer Graphics* - vol.21 nr.4 - july 1987 - pp.:45-53.

[Ston 93] Harold S. Stone - *High-Perfomance Computer Architecture* - Addison-Wesley Publishing Company - 1993.

[SuHo 74] I.E.Sutherland & G.W.Hodgman - "Reentrant Polygon Clipping" - *CACM* - vol.17 - 1974 - pp.:32-42.

[Suth 86] R.J.Sutherland - "A Multiprocessor Architecture for High Quality Interative Display" - *Proceeding Eurographic 86* - 1986 - pp.:265-277.

[Tani 83] S. L. Tanimoto - "A Pyramidal Approach to Parallel Processing" - *Computer Graphics* - vol 17 nr.3 - july 1983.

[Texa 86] Texas Instruments - *TMS34010 User's Guide* - 1986.

[Texa 83] Texas Instruments - *TMS4161, 65536 Bits Multiport Video RAM* - 1983 - pp.:1-22.

[Texa 84] Texas Instruments - *High Performance Memory Acess with the TMS4161* - 1984 - pp.:1-4.

[Thay 89] L. J. Thayer - "Custom VLSI in the 3D Graphics Pipeline" - *Hewlett-Packard Journal* - December 1989 - pp.:74-77.

[ThPa 87] T.Theoharis & I. Page - "Parallel Polygon Rendering With Precomputed Surface Patches" - *Proceeding Eurographic 87* - 1987 - pp.:85.

[Tran 87] Trancept Systems Inc. - *Introducing the TAAC-1 Application Accelerator from Trancept Systems* - 1987.

[VanW 84] J.J.Van Wijk - "Ray Tracing Objects Defined by Sweeping Planar Cubic Splines" - *ACM Transaction on Graphics* - vol.3 nr.3 - july 1984 - pp.:223-237.

[VKLa 88] D. Voorhies, D. Kirk & O. Lathrop - "Virtual Graphics" - *Computer Graphics* - vol.22 nr.4 - aug.88 - pp.:199-205.

[Voor 89] D.Voorhies - "Reduced-Complexity Graphics" - *IEEE Computer Graphics & Applications* - vol.9 nr.4 - july 1989 - pp.:63-70.

[Warr 83] C.Warren - "Focus on Graphics Terminals: VLSI Raises Performances" - *Eletronics Design* - vol 31 nr.2 - jan 20 1983 - pp.:183-192.

[Watk 70] G.S.Watkins - "Ph.D Thesis" - *A Real-Time Hidden Surface Algorithm* - Computer Science Department, Univ. of Utah ~ - Salt Lake City, Utah - 1970.

[WEDe 88] M.C.Whitton, N.England & C. DeMonico - "Manage Design Trade-offs in High-End Graphics Board" - *Eletronic Design* - mar. 1988 - pp.:77-84.

[Weil 85] K. Weiler - "Edge-Based Data Structure for Solid Modeling in Curved-Surface Modeling Environments" - *IEEE Computer Graphics & Applications* - vol.5 nr.1 - january 1985 - pp.:21-40.

[Whit 84] M. C. Whitton - "Memory Design for Raster Graphics Displays" - *IEEE Computer Graphics & Applications* - march 1984 - pp.:48-65.

[Whit 90] S.Whitman - "Computer Graphics Rendering on a Parallel Processor" - *Siggraphs 90 Course Notes, course 28* - Dallas aug. 1990.

[Whit 94] Scott Whitman - "[Dynamic Load Balancing form Parallel Polygon Rendering]" - *IEEE Computer Graphics & Applications* - vol.14 no. 4 - july 1994 - pp.:41-48.

[WhPa 88] S. Whitman & R. Parent - "A Survey of Parallel Hidden Surface Removal Algorithms" - *Pixim 88 Conference Proceeding* ~ - Paris, France - Oct 1988.

[Will 78] L.Williams - "A Comprehensive Shading Model" - *Computer Graphics* - vol.12 nr.3 - august 1978 - pp.:270-274.

[WMSr 92] Yulum Wang, Amante Mangaser & Partha Srinivasan - "A Processor Architecture for 3D Graphics" - *IEEE Computer Graphics & Applications* - september 1992 - pp.:96-105.

[YaTo 84] F. Yamaguchi & T. Tokieda - "A Unified Algorithm for Boolean Shape Operation" - *IEEE Computer Graphics & Applications* - vol.4 nr.6 - june 1984 - pp.:24-37.

[YCKa 92] Roni Yagel, Daniel Cohen & Arie Kaufman - "Discrete Ray Tracing" - *IEEE Computer Graphics & Applications* - september 1992 - pp.:19-28.

[YKFu 84] Yamaguchi, T.L.Kunn & K.Fujimura - "Octree-Related Data Structures and Algorithms" - *IEEE Computer Graphics & Applications* - vol.4 nr.1 - 1984 - pp.:53-59.

[Yous 86] S. Youssef - "A New Algorithm for Object Oriented Ray Tracing" - *Computer Vision, Graphics & Image Processing* - 34 - 1986 - pp.:125-137.

8.2 - Bibliografia Temática

8.2.1 - Algoritmos.

[Appe 68], [AWGr 78], [BaCh 90], [Bark 90], [Berg 86], [BoKe 70], [Carp 84], [Catm 74], [Catm 75], [CoHa 94], [Croc 87], [Crow 77a], [Crow 77b], [CyBe 78], [Deme 85], [DiSw 84], [Duff 85], [DyWh 87], [FrKa 90], [Gour 71a], [Gour 71b], [GTGr 84], [HHPT 92], [Jans 85], [KaGr 78], [Kauf 87], [KWGo 92], [LiBa 83], [LiBa 84], [MaCo 94], [Mamm 89], [Max 90], [Mulm 89], [Myer 75], [NNSa 72], [Peac 85], [Perl 85], [Phon 75], [Watk 70], [YaTo 84]

8.2.2 - Animação

[Blak 87], [FTDa 87]

8.2.3 - Arquiteturas Especializadas

[Apff 89], [BuDe 89], [CrOr 94], [DiSw 84], [DWSD 88], [FGHS 85], [FuPo 81], [FuRa 82], [GhPo 85], [GSSu 81], [HaAk 90], [KaBa 88], [Kauf 86], [Knit 93], [KWGo 92], [MEPo 92], [NIMT 84], [PoHo 89], [ScCl 88], [ScSt 93], [Suth 86], [WMSr 92]

8.2.4 - *Anti-Aliasing*

[Crow 77b], [Crow 81], [Mamm 89], [Max 90]

8.2.5 - *Chip*

[Agat 86], [ChRo 86], [Clar 82], [Deme 85], [DWLG 92], [Fuch 89], [GGAk 92], [Ghar 88], [GhPo 85], [GKBh 89], [Jeff 85], [KoMa 89], [PEMF 92], [PiGa 87], [PuKa 87], [Schn 88], [Stae 86], [Thay 89], [Warr 83], [Whit 84]

8.2.6 - Clipping

[CyBe 78], [LiB 83], [LiBa 84], [SuHo 74]

8.2.7 - Compositing

[Duff 85], [FoKi 87], [Knit 93], [MEPo 92], [Naka 89], [PoDu 84]

8.2.8 - Coherence

[Croc 87], [GrPa 88], [GrPa 89a], [GrPa 89b], [MaCo 94], [Pack 89]

8.2.9 - CGS

[Jans 85], [RoRe 86]

8.2.10 - Circuitos de Memória de Quadro

[Bera 89], [Brag 85], [Curt 90], [DuWa 85], [Nico 88], [Oliv 83], [Warr 83], [WEDe 88]
[Whit 84], [Oliv 89]

8.2.11 - Exemplos de *Hardware* Comerciais.

[Akel 89], [Akel 93], [AkJe 88], [GKBh 89]

8.2.12 - Janelas

[PiGa 87], [RhWi 89], [ScGe 86], [Ster 87]

8.2.13 - Manuais de Componentes.

[Brok 85], [Hita 89], [Inte 86], [Texa 83], [Texa 84], [Texa 86]

8.2.14 - Metodologia de Projetos.

[Arms 89], [Figu 95], [HuRo 91], [Ment 89], [Perr 91]

8.2.15 - *Modelling*

[GBWi 90], [GTGr 84], [HaSp 87], [HHPT 92], [Mand 82], [Max 90], [Mora 81], [Roth 82], [SKWF 92],
[VanW 84], [Weil 85], [Whit 80], [Willi 78], [YKFu 84]

8.2.16 - Multi-mídia

[GGAk 92]

8.2.17 - Parallel Rendering

[MCEF 94], [Ells 94], [Whit 94], [Neum 94], [MPHK 94], [BBPr 94], [CoHa 94], [HCWh 94], [CrOr 94], [MaCo 94]

8.2.18 - Padrões

[Blin 92], [Brok 85], [EIA 57], [EIA 69], [Mamm 91], [Plae 87]

8.2.19 - Processamento de Alto Desempenho

[Bord 89], [CrOr 94], [Dunc 90], [Flyn 66], [Fuch 89], [GBWi 90], [GrPa 88], [GrPa 89a], [GrPa 89b], [HwBr 84], [JaSu 86], [SDDa 89], [Sieg 92], [Ston 93], [Tani 83]

8.2.20 - Processamento Distribuido

[Maso 87]

8.2.21 - Processamento de Imagens

[KoBh 92], [RoKa 76], [GoWi 77]

8.2.22 - *Radiosity*

[BaWi 90], [GTGr 84]

8.2.23 - *Ray-Tracing.*

[AnWo 87], [ArKi 87], [BBDM 85], [BBPr 90], [BrKl 85], [BrKl 85], [Glas 84], [GoSa 87], [GrPa 88], [GrPa 89a], [GrPa 89b], [HeHa 84], [Kaji 83], [KaKa 86], [PlBa 85], [PuKa 87], [PZLi 87], [RiAr 94], [VanW 84], [YCKa 92], [Yous 86]

8.2.24 - *Rasterization*

[Deme 85], [DeNe 93], [FGRe 85], [FPIw 84], [Fuch 89], [FuPo 81], [GFBa 87], [GGSS 89], [GSSu 81], [HaSc 87], [PEMF 92]

8.2.25 - *Rendering*

[CrOr 94], [DWLG 92], [DyWh 87], [FrKa 90], [Ghar 88], [Gour 71], [Gour 71], [GrPa 88], [GrPa 89a], [GrPa 89b], [HaAk 90], [HaFo 93], [HaGr 86], [KiVo 90], [KWGo 92], [Levo 88], [MEPo 92], [MoRe 87], [ScCl 88], [Schn 88], [ThPa 87]

8.2.26 - *Scanline*

[AEZa 87], [Croc 87], [Deme 85], [KWGo 92], [Watk 70]

8.2.27 - *Surveys*, Tutoriais e Livros.

[Aman 87], [Arms 89], [CCHs 92], [Crow 81], [Dunc 90], [Elvi 92], [FDFH 90], [FoVD 82], [GGSS 89], [HaSp 87], [HCWh 94], [HeBa 86], [Heck 86], [Hill 90], [JoGr 88], [MaTh 87], [MoFu 90], [NeSp 79], [NeSp 79], [Oliv 91], [Perr 91], [Roge 85], [RoRe 86], [Schu 69], [SSSc 74], [SSUl 85], [Whit 90], [BaBr 82], [GoWi 77], [Horn 86], [RoKa 76].

8.2.28 - *Texture*

[Blin 78], [BlNe 76], [FFCa 82], [Gard 84], [Heck 86], [Peac 85], [Perl 85]

8.2.29 - VHDL

[Arms 89], [Figu 95], [Oliv 95], [Perr 91]

8.2.30 - Visão Computacional

[BaBr 82], [Horn 86]

8.2.31 - *Volume Visualization*

[Elvi 92], [KaBa 88], [Knit 93]

8.2.32 - *Work-Station*

[ABMa 88], [Akel 89], [Akel 93], [DeNe 93], [KiVo 90], [Maso 87], [MaSt 89], [OBZi 89], [OeGr 90], [Robi 87], [Tran 87]

8.2.33 - Z-Buffer.

[BFPa 86], [Catm 74], [Catm 75], [FPIw 84], [Loga 90]

8.3 - Referências Bibliográficas

8.3.1 - Índice de Referências do Capítulo 1.

[BaBr 82] 3
[BBPr 94] 4
[Ells 94] 4
[FDFH 90] 2
[Horn 86] 3
[Jolu 94] 2
[MCEF 94] 4
[MPHK 94] 4
[Neum 94] 4
[Whit 94] 4

8.3.2 - Índice de Referências do Capítulo 2.

[AkJe 88] 52
[Aman 87] 51
[AWGr 78] 39
[BaCh 90] 53
[BaWi 90] 53
[Berg 86] 40
[Blin 78] 51
[BlNe 76] 51
[BoKe 70] 42
[BuDe 89] 53
[Catm 74] 33, 50
[Crow 77a] 40
[Crow 77b] 50
[CyBe 78] 25
[FDFH 90] 9, 50
[FFCa 82] 51
[Gard 84] 51
[Gour 71a] 43
[Gour 71b] 43
[GTGr 84] 52
[HeBa 86] 9, 10, 12, 13
[Heck 86] 51
[Hill 90] 9
[JGMH 88] 32
[KaGr 78] 38
[LiBa 83] 26
[LiBa 84] 25
[Mora 81] 52
[Mulm 89] 53
[Myer 75] 36
[NeSp 79] 25
[NNSa 72] 34, 35
[Peac 85] 51
[Perl 85] 51
[Phon 75] 47
[Roge 85] 9, 25, 27
[Schu 69] 34
[SSSc 74] 31, 38
[SuHo 74] 26
[Watk 70] 36
[Whit 80] 38
[Will 78] 40

8.3.3 - Índice de Referências do Capítulo 3.

[ABMa 88] 68
[Akel 89] 92
[Bera 89] 92
[Brag 85] 82
[Brok 85] 59
[CSLe 86] 88
[Duff 85] 92
[EIA 57] 61
[EIA 69] 61
[GGSS 89] 92
[Hita 89] 93
[Inte 86] 88
[Mamm 91] 56
[Oliv 83] 82
[PEMF 92] 92
[PiGa 87] 90
[PoDu 84] 92
[RhWi 89] 88
[ScGe 86] 86
[SDDa 89] 92
[Ster 87] 86
[WEDe 88] 92
[Whit 84] 92

8.3.4 - Índice de Referências do Capítulo 4.

8.3.5 - Índice de Referências do Capítulo 5.

8.3.6 - Índice de Referências do Capítulo 6.

8.3.7 - Índice de Referências do Capítulo 7.

Anexo A

RT DCA 02/95

Descrição da MQ/UGR e VAM em VHDL e Simulação

Conteúdo

1. - Introdução

Este relatório faz uma descrição formal do projeto da plataforma UGR desenvolvido como parte do trabalho de doutoramento do autor. Ele também apresenta a metodologia de desenvolvimento de projeto adotada.

Pretende-se com este trabalho fazer um registro formal da arquitetura da UGR. era descrever em VHDL (VHSIC *Hardware Description Language*) a interface de acesso à VAM pelas unidades de processamento elementar. Uma vez obtida esta descrição seria possível a implementação através dos recursos de síntese disponíveis na FEE.

Para esta descrição foi utilizado o sistema SA (*System Architect*) componente da série de ferramentas para projeto de sistemas da Mentor Graphics. Este sistema está instalado no LCAEe (Laboratório de Computação Aplicada à Eng. Elétrica) da FEE.

Na seção 2 é feita uma introdução a metodologia de projeto adotada. A seção 3 descreve as principais características da série SDS da Mentor Graphics. E a partir da seção 4 é feita a descrição do sistema UGR.

2. - Metodologia de Projeto.

2.1 - O Ciclo Convencional de Projeto

O projeto de sistemas eletrônicos tem convencionalmente seguido um ciclo de desenvolvimento similar àquele mostrado na figura 1. O termo "sistema" é usado aqui num sentido genérico. Um sistema refere a um conjunto, possivelmente complexo, de componentes eletrônicos e de *software*. Idealmente, o desenvolvimento começa com uma especificação que é ao mesmo tempo completa e consistente. A especificação define o comportamento requirido, a interface com outros sistemas, os parâmetros de desempenho e outras características de sistema.

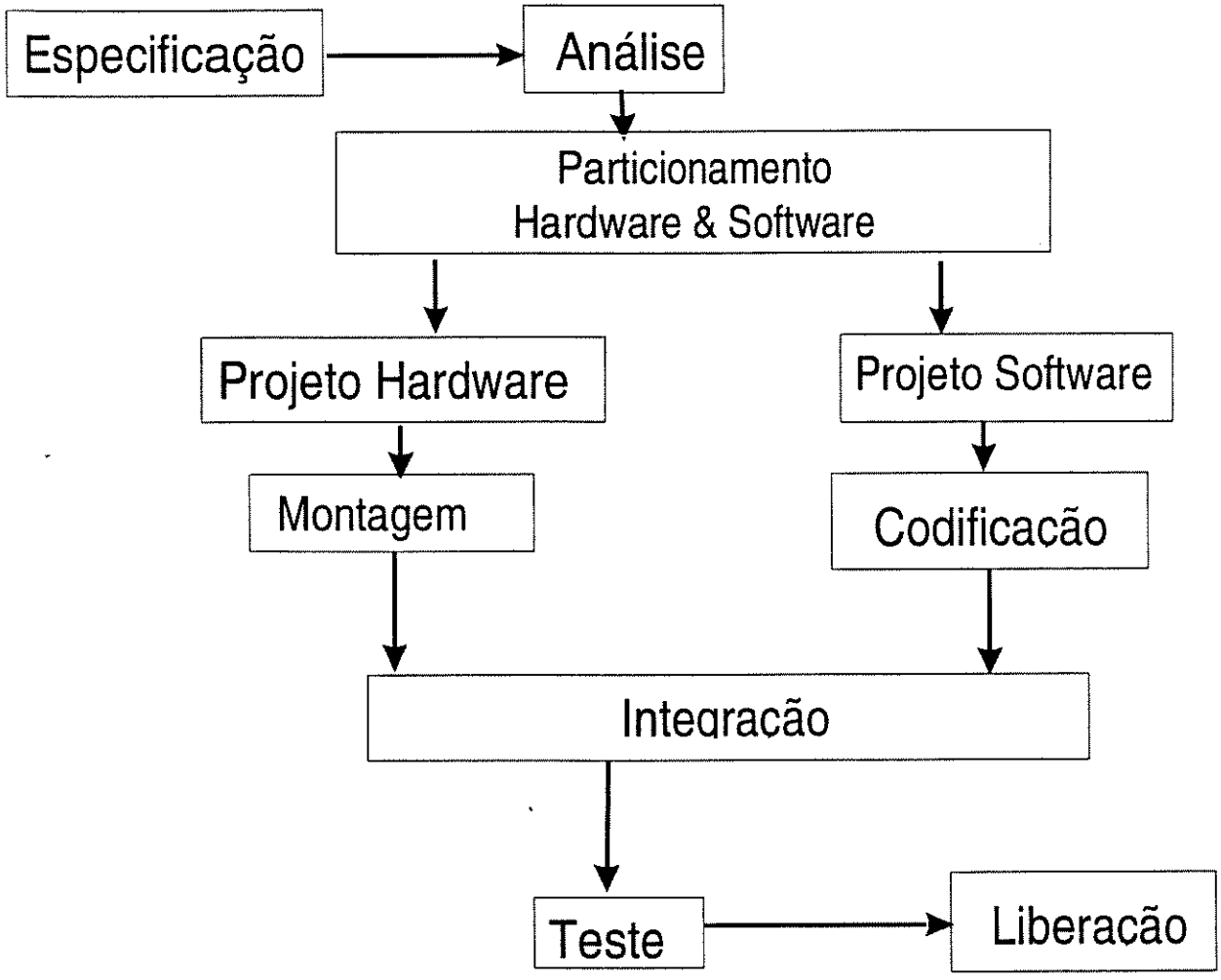


Fig. 1 - Ciclo Convencional de Desenvolvimento

O segundo estágio no processo é a análise da especificação para obter um modelo de alto nível do sistema. Este modelo define os principais componentes do sistema, os estados, que funções cada componente irá realizar e define a interação entre os componentes. Para determinados sistemas é importante avaliar o seu modelo para assegurar que o sistema irá atingir o desempenho e outros critérios requeridos.

Como até recentemente, ferramentas adequadas de projeto em alto-nível não estavam disponíveis, as decisões de implementação costumavam ser tomadas no estágio de análise. Estas decisões restrigiam alternativas nos estágios posteriores de desenvolvimento, tornando dificeis quaisquer mudanças.

Uma vez o modelo de alto-nível tenha sido completado, o sistema pode ser particionado em funções que podem ser implementadas por *hardware*, e funções que podem ser implementadas via *software*. Os componentes em *software* e em *hardware* podem então ser projetados e construídos. Os estágios finais do ciclo convencional de projeto requer que os componentes em *hardware* e em *software* sejam integrados, testados e por último o sistema é liberado para o mercado.

O ciclo de projeto descrito até aqui é uma versão idealizada do ciclo de projeto no mundo real. Na verdade, as especificações são em geral incompletas e contraditória. Em muitos casos, os requisitos do consumidor podem não ser ainda conhecido.

Neste ciclo de projeto convencional, erros introduzidos nos estágios anteriores propagam-se através do ciclo e podem ser detectados somente durante os testes de integração, ou mesmo após a liberação para o mercado. Quanto mais tarde for detectado o erro, mais cara será a sua correção. E mais, o ciclo descrito não estabelece nenhuma forma de ajuda na decisão de como particionar um sistema em componentes *hardware* e *software*. Em geral, esta divisão recai na experiência do projetista que pode não ter recursos para comparar estratégias alternativas.

2.2 - O Ciclo de Desenvolvimento Evolutivo

O ciclo de desenvolvimento evolutivo mostrado na figura 2 é uma resposta as deficiência do ciclo convencional de desenvolvimento. O ciclo de projeto convencional é modificado pela adição de um novo estágio de prototipagem e a redução do tempo gasto no estágio de análise comparado com o estágio de análise no ciclo convencional.

Durante o estágio de análise no ciclo evolutivo, a especificação é examinada rapidamente e um modelo simples e não necessariamente completo do sistema é produzido. Um protótipo do sistema é então desenvolvido para demonstrar a viabilidade do sistema. O protótipo fornece vantagens como por exemplo:

- a habilidade para verificar que o sistema irá atender todos os requisitos conhecidos;
- a habilidade para descobrir novos requisitos;
- a habilidade para examinar diferentes alternativas sem as despesas de ter que desenvolver todo o sistema.;
- a habilidade para decidir qual parte do sistema pode ser implementado via *hardware* ou via *software*.

Duas formas diferentes de prototipagem podem ser seguidas. Um protótipo "*throwaway*" pode ser criado, de forma a servir somente como um veículo de teste e que não será usado no sistema final. Outra forma é construir um protótipo "evolutivo" ("*evolutionary*"), que pode evoluir e formar parte do produto final.

Protótipos *throwaway* são o tipo mais comum de protótipo. Ele se aplica principalmente em grandes projetos de engenharia, por exemplo, de uma aeronave. Os protótipos evolutivos são particularmente adequados para sistemas que envolvem uma significante quantidade de *software*, ou componentes cujo projeto possa ser facilmente modificado. Este tipo de protótipo é construído de forma que ele pode evoluir até o final do ciclo de projeto do sistema, agregando qualidade e confiabilidade ao mesmo tempo.

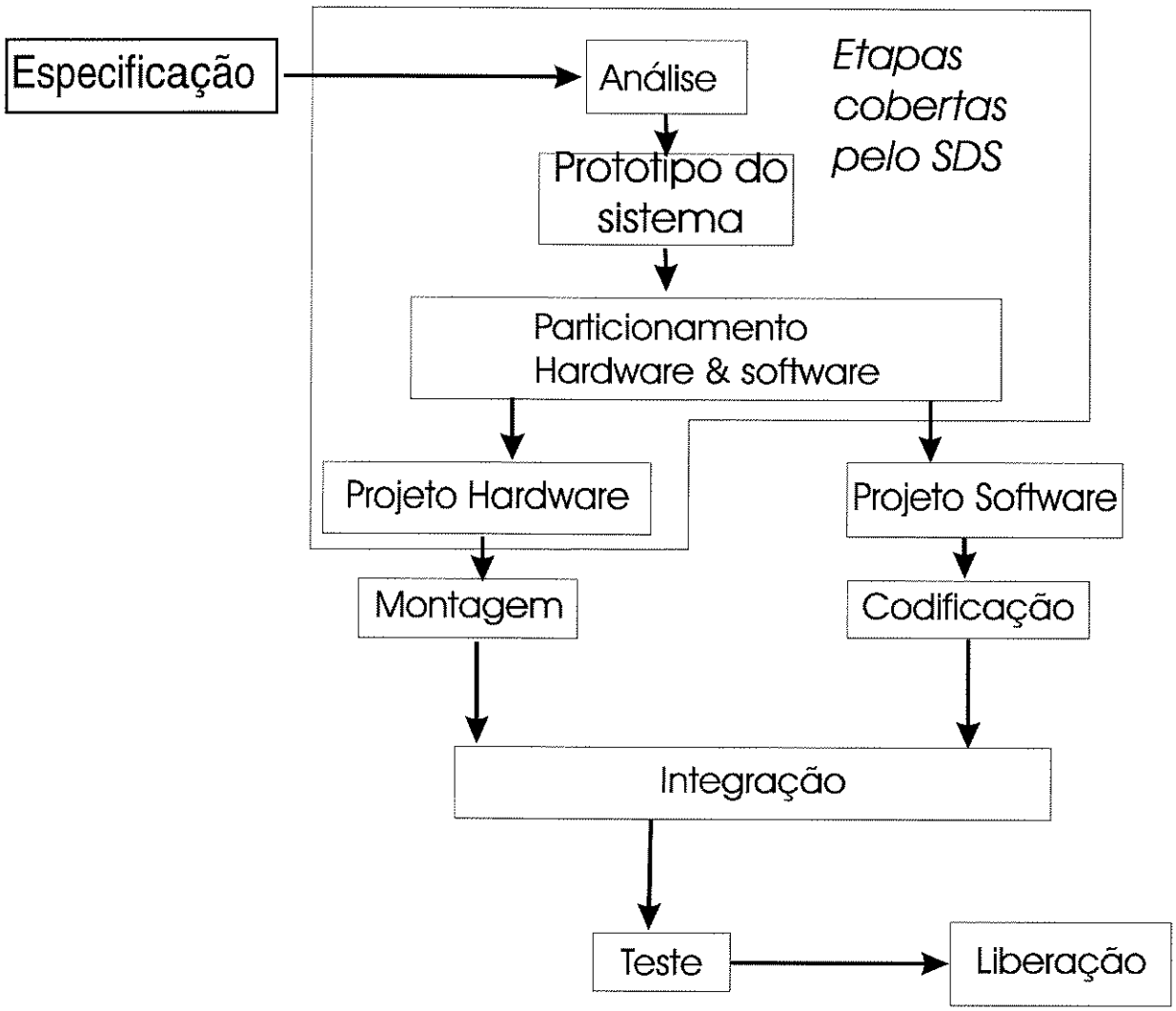


Fig. 2 - Ciclo de desenvolvimento evolutivo.

3. - System Design Series

A série de produtos da Mentor Graphics voltados para o projeto de sistemas, a chamada SDS (*System Design Series*), cobre parte do ciclo de desenvolvimento evolutivo. A parte coberta pelo SDS é mostrada na figura 2. A forma como esta cobertura é feita é mostrada na figura 3 e descrita a seguir.

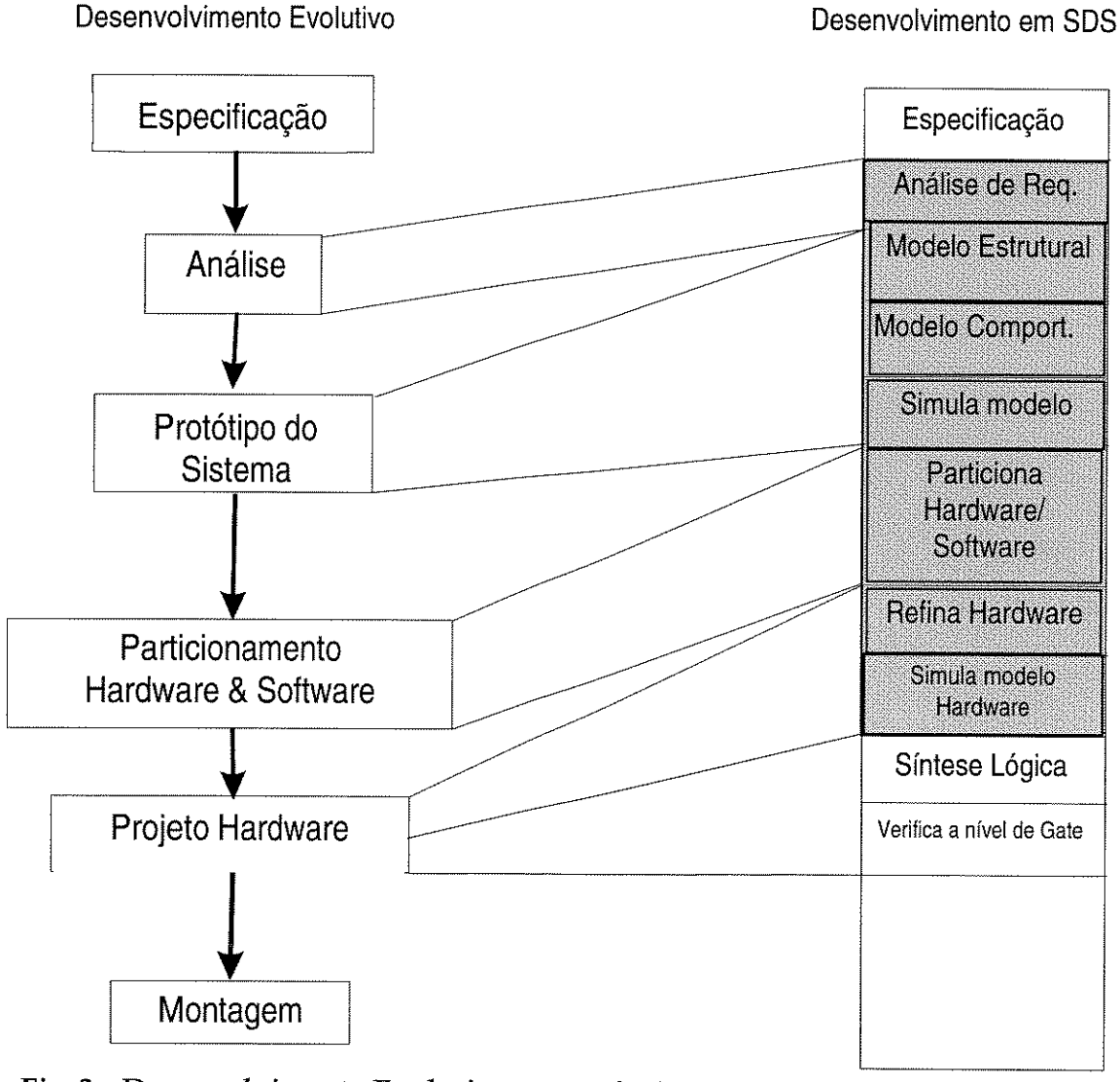


Fig. 3 - Desenvolvimento Evolutivo com o SDS

No ciclo de desenvolvimento com SDS a etapa de análise é substituída por duas etapas, "Análise dos Requisitos" e "Definição da Estrutura do Modelo". Durante a primeira etapa a especificação é examinada para identificar todos os atributos que o sistema completo deverá possuir. Na segunda, a estrutura do modelo é criada de

forma a definir as relações entre os componentes usados no modelo do sistema, mas sem determinar o comportamento dos mesmos. Isto é deixado para a etapa seguinte, chamada de "Definição do Comportamento do Modelo" quando o comportamento de cada componente é descrito, seja como Máquina de Estado seja utilizando VHDL. Quando se constroi o modelo, a maior preocupação é com a arquitetura de alto-nível e o estabelecimento do seu comportamento correto, não com os detalhes de implementação em baixo-nível.

Uma vez completadas estas três etapas, o modelo pode ser simulado. A simulação permite ao projetista verificar se o modelo opera como o requirido, descartando os requisitos contraditórios e incompletos. Na simulação se o modelo comporta-se satisfatoriamente, então o ciclo de desenvolvimento pode prosseguir. Se não, então o modelo pode ser corrigido e então re-simulado.

Até aqui o modelo do sistema não foi dividido em componentes *hardware* e componentes *software*. Adiando esta decisão dá ao projetista o máximo de informações sobre qualquer compromisso envolvido em especificar os componentes em *hardware* ou em *software*, permitindo explorar as alternativas de projeto.

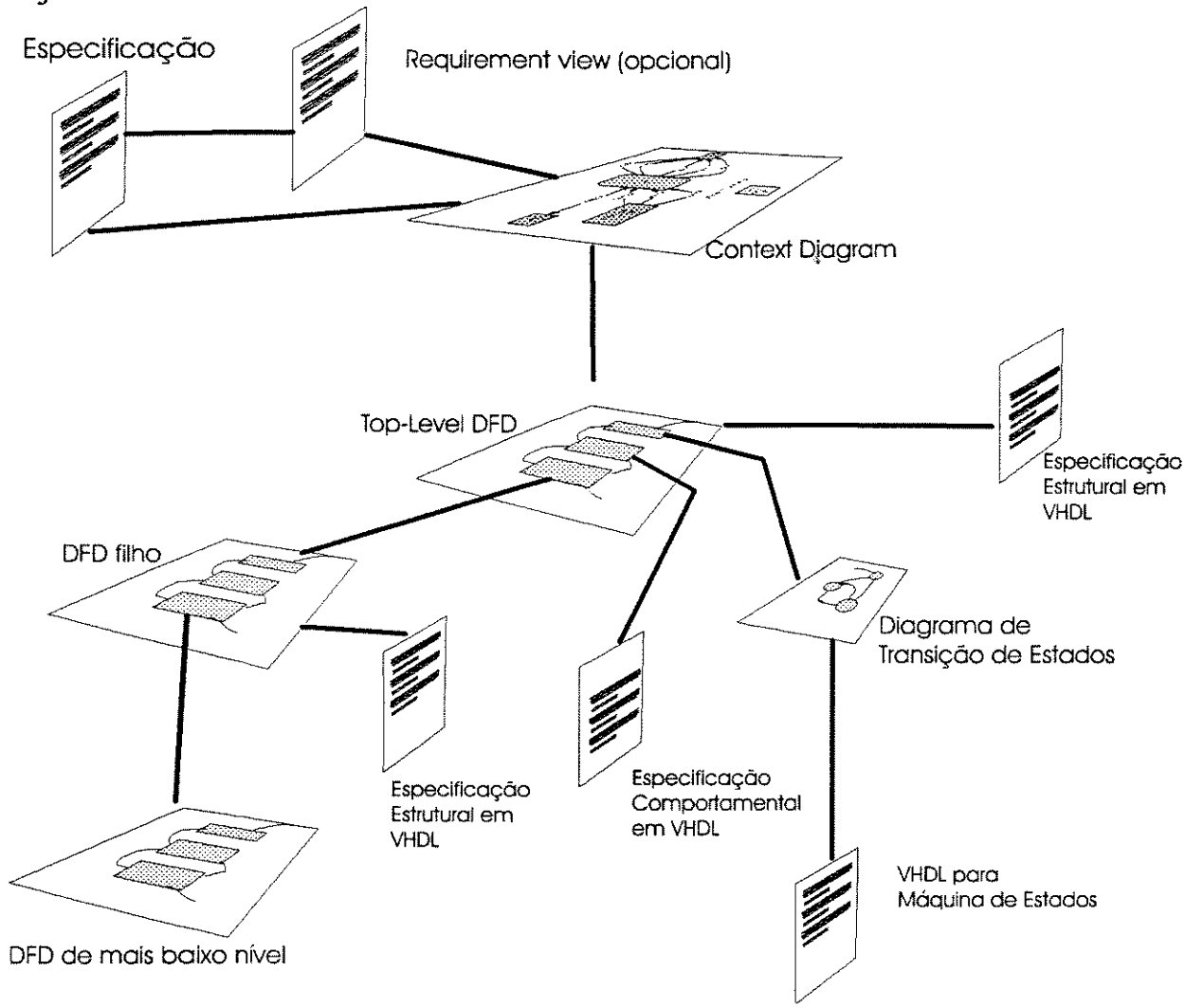
Aquelas partes do sistema que deverão ser implementadas em *software* podem ser exportadas para ferramentas de projeto de *software* usando interfaces CDIF e refinadas para fornecer o mais detalhado comportamento. As partes que deverão ser implementadas em *hardware* podem ser refinadas em SDS de forma que elas modelem exatamente a eventual implementação. Os componentes em *hardware* podem mais tarde serem sintetizados. Uma vez que o modelo esteja refinado, ele pode novamente ser simulado.

3.1 - Estrutura do Modelo do Sistema

O ponto inicial para um modelo de sistema é a especificação. Para ajudar capturar os requisitos importantes presentes na especificação, um *Requirements View* pode ser criado pela análise da especificação. Verificações automáticas auxiliam o projetista a se assegurar que um requisito é observado por uma ou mais partes do modelo do sistema. Entretanto, o projetista pode dispensar esta etapa e iniciar interpretando diretamente os requisitos.

A estrutura do modelo do sistema é definida por diagramas de fluxos de dados (*Data Flow Diagrams* - DFD) utilizando-se das informações obtidas da especificação ou dos requisitos. Os diagramas DFD formam uma hierarquia, com cada DFD tendo

zero ou mais DFD filhos. Cada DFD de menor nível acrescenta progressivamente mais detalhes como mostrado na figura 4. O DFD utiliza uma notação gráfica para definir as relações estruturais entre as partes do sistema definidas em detalhes pelos objetos de menor nível.



ig. 4 - Estrutura do Modelo de Sistema

O DFD de mais alto nível da hierarquia é denominado *Top Level Data Flow Diagram*. Ele tem uma forma alternativa, na forma de "caixa preta", denominada *Context Diagram*, ou simplesmente CD, que descreve a relação entre o sistema e o ambiente externo. O *Context Diagram* utiliza uma notação gráfica similar a do DFD. O comportamento do Modelo do sistema é definido diretamente, em VHDL, ou pela criação de máquinas de estado. Gabaritos de especificação em VHDL são geradas automáticamente mas devem ser editadas para descrever plenamente o comportamento do sistema.

Máquinas de estado fornecem um meio eficiente de descrever o comportamento dos componentes reativos de um sistema. Um componente reativo tem que responder uma mudança na entrada com um conjunto de valores e de ações. SDS suporta duas formas de descrição de máquinas de estado, uma em forma de diagrama gráfico — *State Transition Diagram* (STD) e outra em forma de matriz — *State Transition Matrix* (STM). As duas formas tem capacidade de modelamento identicas, a escolha do formato é deixada para o projetista. Descrições em VHDL podem ser geradas diretamente das descrições em máquinas de estado, assim como um painel de controle — *Control Panel*. O *Control Panel* serve como uma interface definida pelo usuário durante a simulação ou como uma interface protótipo para o sistema.

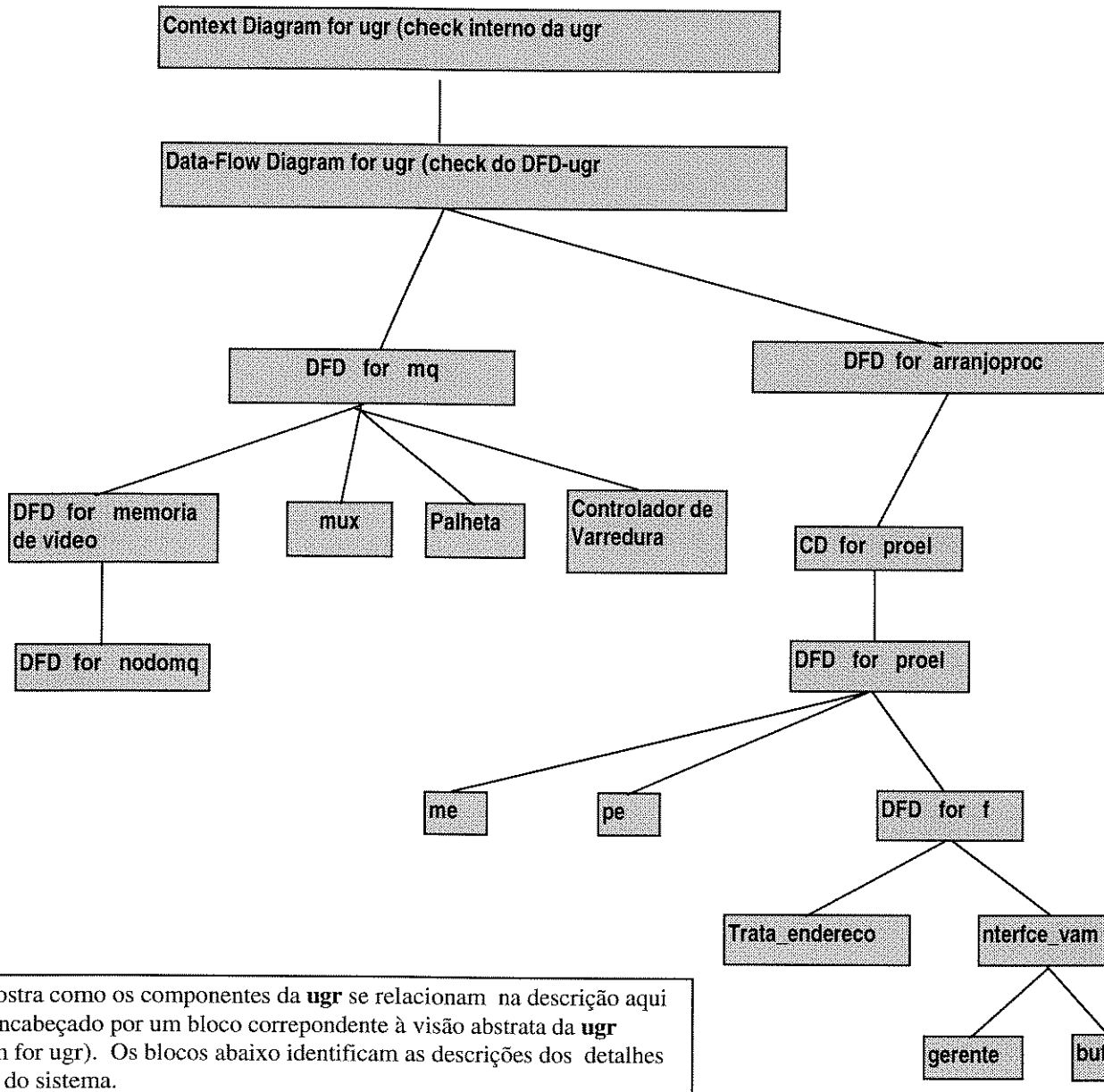
Descrições em VHDL podem ser geradas para cada DFD. A notação gráfica utilizada, possibilita fazer uma associação direta entre o diagrama e a descrição estrutural em VHDL.

É importante destacar que a medida que novos níveis de abstração vão sendo explorados, ou seja, os componentes de um determinado nível são descritos, o sistema permite uma verificação (*check*) da consistência das descrições. Isto é feito considerando por exemplo os sinais internos da descrição (*internal check*), os sinais que são descritos em níveis mais abstratos (*parent check*) e os tipos de dados ou sinais (*Type Definition Package Check*) utilizados na descrição que está sendo verificada.

Outro destaque é o recurso de poder editar sinais comuns a níveis hierárquicos diferentes. Por exemplo, considere um sinal de interface presente no Top-Level DFD e numa sequência aninhada de DFD's de menor nível. Se este sinal, por razões mnemônicas precisar ser mudado de nome no nível mais inferior, o sistema realiza todas as mudanças de nome nos demais níveis.

4. - Descrição da UGR

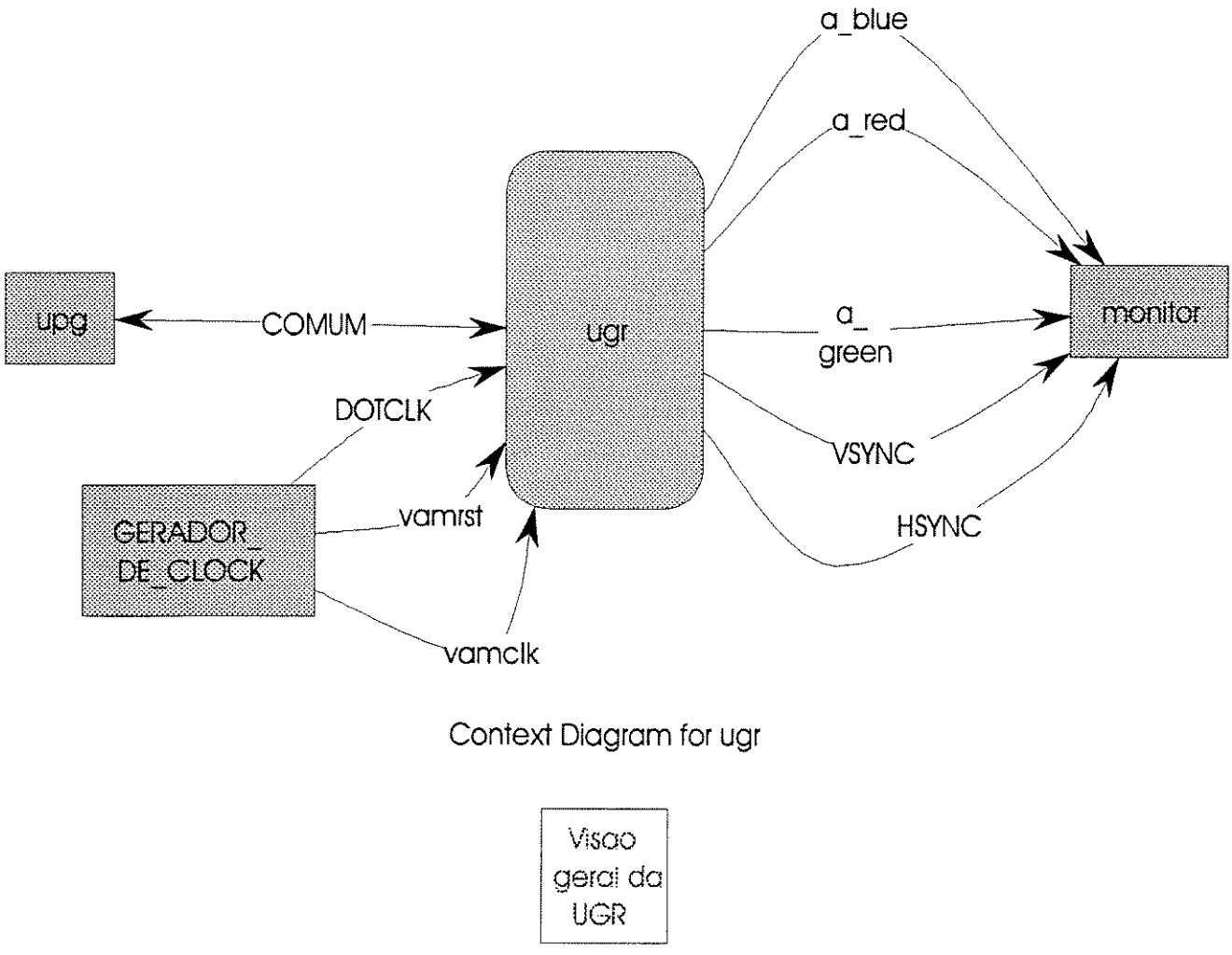
A partir desta seção é feita a descrição da plataforma UGR de acordo com o SDS da Mentor Graphics. A figura 5 ilustra a hierarquia do modelo do sistema UGR. Nas seções seguintes, cada um dos componentes do sistema é descrito. Alguns componentes por simplificação do texto não serão descritos profundamente.



Este diagrama mostra como os componentes da **ugr** se relacionam na descrição aqui realizada. Ele é encabeçado por um bloco correpondente à visão abstrata da **ugr** (Context Diagram for ugr). Os blocos abaixo identificam as descrições dos detalhes das sub-unidades do sistema.

Observe que alguns componentes (como por exemplo: **mux**, **palheta** e **me**) não estão descritos em detalhe. Para o desenvolvimento deste trabalho, bastou descrevê-los na forma comportamental. Já o componente **proel**, foi dividido em suas sub-unidades e estudada com maior ênfase a sua interface com a VAM. Outra unidade considerada com maior detalhe foi o componente **nodomq**, que interage diretamente com a VAM.

Fig. 5 - Modelo Hierárquico da UGR



ig. 6 - Visão geral da UGR

4.1 - Context Diagram for UGR

A figura 6 mostra o *Context Diagram* da UGR. Neste diagrama estão representados o componente UGR como uma "caixa preta" e os elementos externos que podem ser ligados a UGR. A UGR se liga pelo barramento COMUM a upg. A UGR se liga ainda ao monitor de vídeo através dos sinais a_red (sinal analógico da informação vermelha) a_green (sinal analógico da informação verde), a_blue (sinal analógico da informação azul) e os sinais de sincronismo (Vsync - sincronismo vertical e Hsync -

sincronismo horizontal). Por facilidade de simulação foi colocado ainda como sinal gerado externamente a UGR os sinais de controle DOTCLK, vamclk e vamrst. Entretanto, estes sinais deverão ser incorporados no circuito da UGR.

4.2 - Top-level Data Flow Diagram for UGR

A "caixa preta" da UGR pode ser explorada pela visão do diagrama *Top-level DFD* mostrado na figura 7. Nesta figura pode-se verificar que a UGR é composta de dois grandes componentes, o arranjo de processadores — arranjoproc e a memória de quadro — mq. Observe que os elementos externos do CD da UGR são substituidos por uma representação de conexão. Os sinais que faziam a ligação da UGR com os elementos externos agora estão ligados a componentes específicos da UGR. Este diagrama introduz os sinais que compoem a VAM, a saber, os barramentos BA, BB, BC e BD que interligam os dois grandes componentes da UGR.

Numa primeira aproximação poderia ter sido feita uma descrição comportamental de cada um dos componentes da UGR, com a abstração sugerida por este DFD. Entretanto, optou-se por explorar separadamente cada um deles. A mq é descrita na seção 5 e o arranjoproc na seção 6.

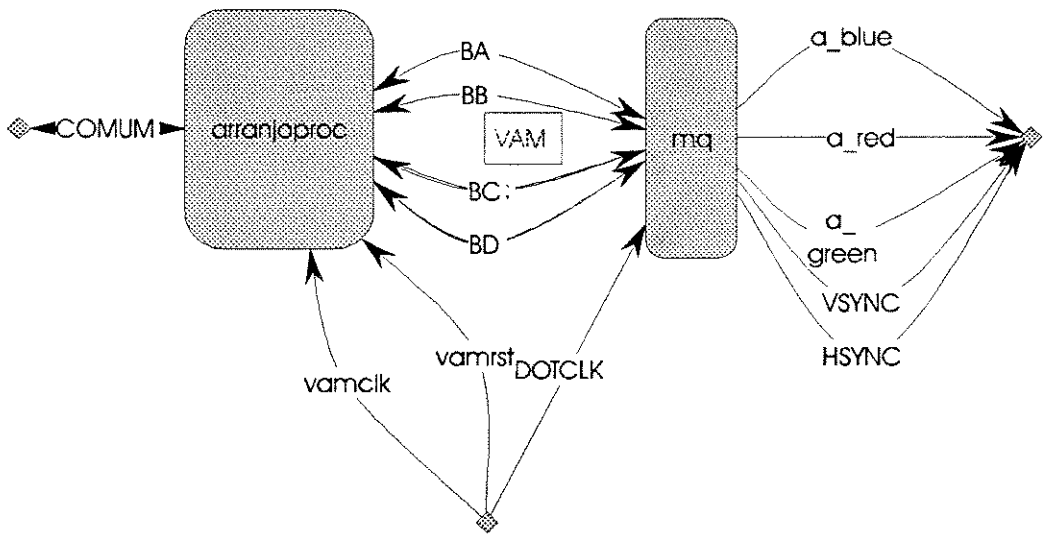


Fig. 7 - Top-Level Data Flow Diagram da UGR

Para exemplificar, a verificação da consistência de uma descrição de nível hierárquico inferior a figura 8 mostra o *check report* da DFD da UGR, o nível *parent* é o CD da UGR.

```
Check Report
============

View Selection
--------------
Component        : $DESIGNS/ugr
View Name        : data_flow
View Type        : Data Flow Diagram
Check Extent     : single

Relevant Checks Enabled
-----------------------
Internal         : Errors & Warnings
Parent           : Errors & Warnings
TDP              : Errors & Warnings
Formal Instance : No Check

Check Data Flow Diagram "$DESIGNS/ugr/data_flow":
Passed check: 0 Errors, 0 Warnings.

Check Summary
-------------
Total number of views checked                : 1
Number of views that failed checks           : 0
Number of views that partially passed checks : 0
Total items reported                         : 0 Errors, 0
Warnings
```

ig. 8 - Exemplo de check report

5. - Descrição da MQ

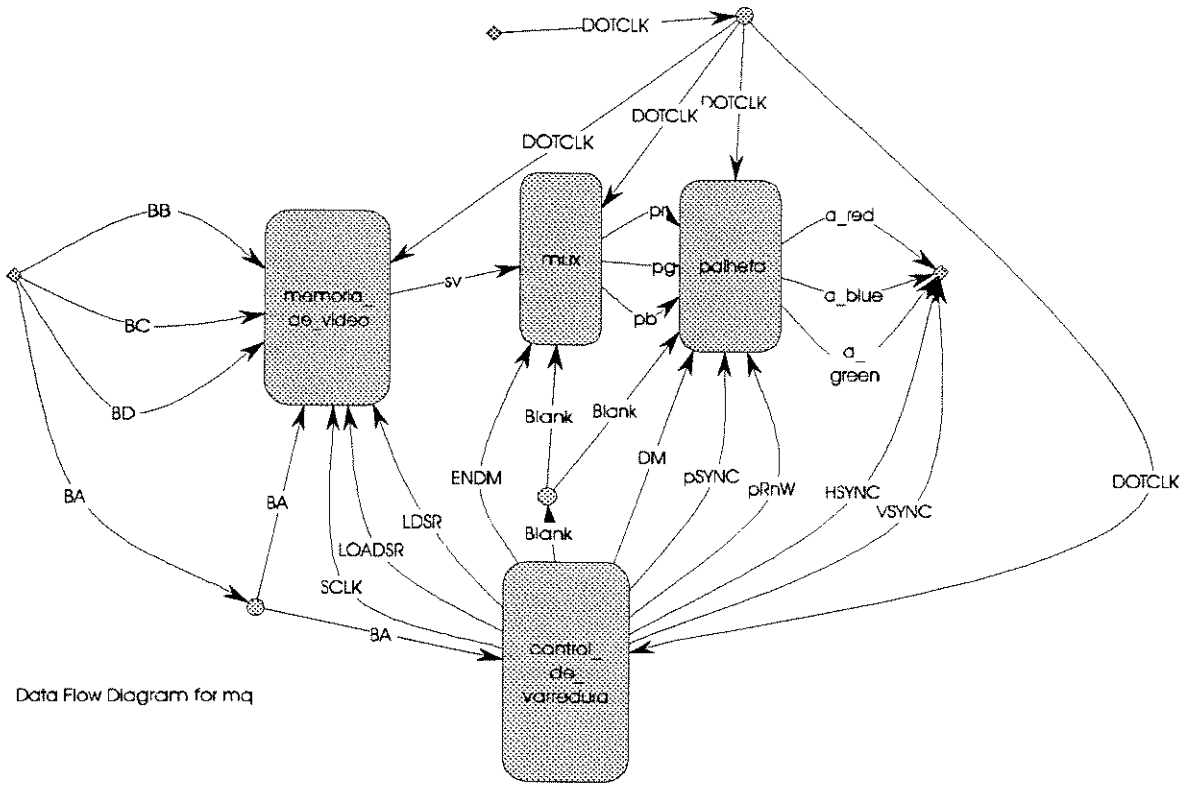


Fig. 9 - Visão DFD da MQ da UGR

Partindo do *Top-Level DFD* da UGR esta seção explora o componente mq, que corresponde ao circuito da memória de quadro. A figura 9 mostra os componentes da memória de quadro da UGR, bem como todos os sinais internos.

5.1 - O Mux, a Palheta e o Controle de Varredura

Estes três componentes da memória de quadro não serão descritos neste documento. A razão para isto é que se tratam de circuitos com a descrição comportamental bem conhecida e se forem integradas ao nosso modelo do sistema da UGR, descrições comportamentais simplificadas de cada um, isto em nada comprometerá a análise do sistema como um todo.

5.2 - Descrição da Memória de Vídeo

É importante explorar com mais detalhes o componente memoria_de_video, pois é ele que receberá as informações da VAM. Esta exploração procura definir a melhor alternativa de projeto para esta via.

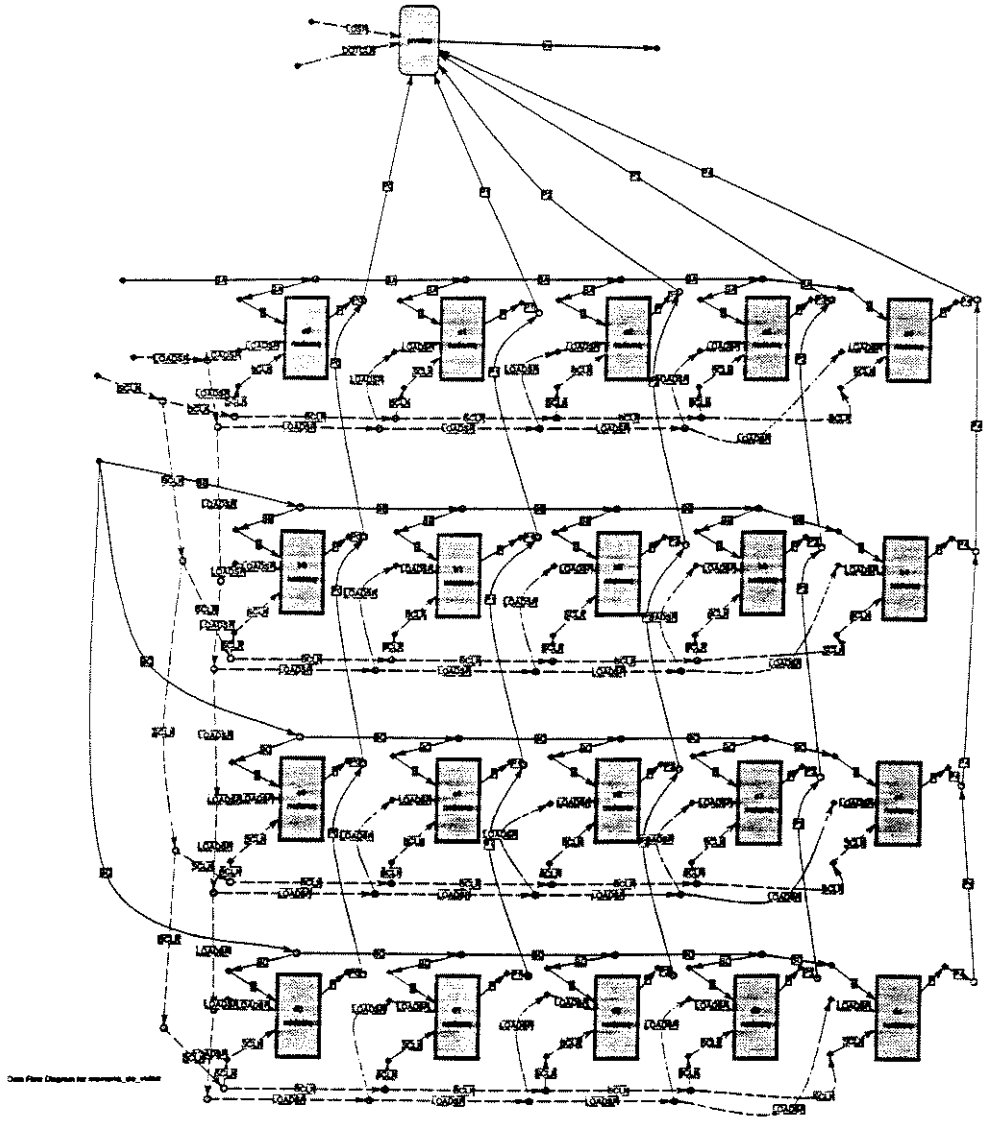


Fig. 10 - DFD do componente memoria_de_vídeo

A figura 10 mostra o diagrama DFD do componente memoria_de_video. A memória de vídeo da MQ é formada por 20 nós de armazenamento de informações de imagem. Estes 20 nós estão a princípio intercalados formando uma pegada de 4 linhas e 5 colunas. A saída serial de cada um destes nós formam um barramento de *pixels* (sinais P_n - $0 \leq n \leq 4$) que são enviados para o componente srveloz (registrador de deslocamento veloz) que trabalhando na velocidade do DOTCLK envia de forma intercalada os *pixels* para a conexão externa.

Cada nó da memória de quadro é modelado aqui como um componente externo, este componente é instanciado 20 vezes no modelo da memória de vídeo. Como se trata de um componente externo, este é descrito separadamente na seção 6.

6. - Descrição do Nó de Memória de Vídeo

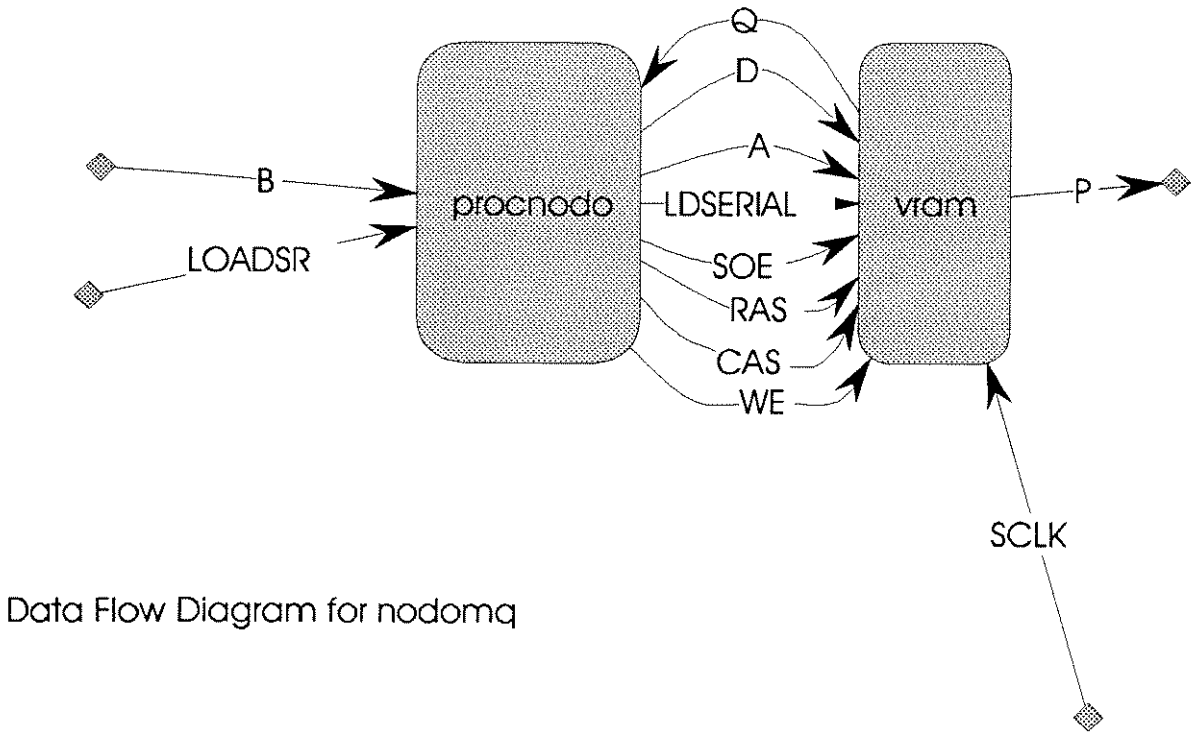


Fig. 11 - Descrição Top-Level do nó de memória de vídeo

A figura 11 apresenta a *Top-Level DFD* do componente nodomq utilizado 20 vezes no circuito de memória de vídeo. Pode-se descrevê-lo como sendo composto por dois elementos básicos: - Um controlador de acesso à memória de vídeo, aqui chamado de procnodo e o bloco de memória vram propriamente dito. O controlador de acesso (procnodo) se interliga com o barramento b de onde retira os dados e endereços durante os ciclos de acesso a memória vram. Durante os ciclos de carga dos *shift-registers* da vram ele recebe o sinal LOADSR.

O bloco vram é uma unidade de memória de vídeo que em função do sinal SCLK envia os *pixels* para o barramento de *pixels* p.

7. - Descrição do Arranjo de Processadores

A figura 12 mostra o arranjo de processadores da UGR. Este arranjo é formado por 20 unidades de processamento, aqui chamadas de proel. Para permitir uma visualização melhor a figura apresentada não mostra todas as ligações. E para compatibilizar com as descrições anteriores (de maior nível hierárquico) foi incluido nesta descrição um componente que faz a separação dos sinais de dados, endereço e de controle. Este componente chamado de trata_comum deve desaparecer num refinamento do projeto.

De uma forma geral pode-se dizer que cada proel deve estar ligado aos quatro barramentos da VAM e ao barramento comum. Além disso, todos os proel's devem estar ligados aos sinais de controle vamclk e vamrst.

A descrição do componente proel é feita na seção 8.

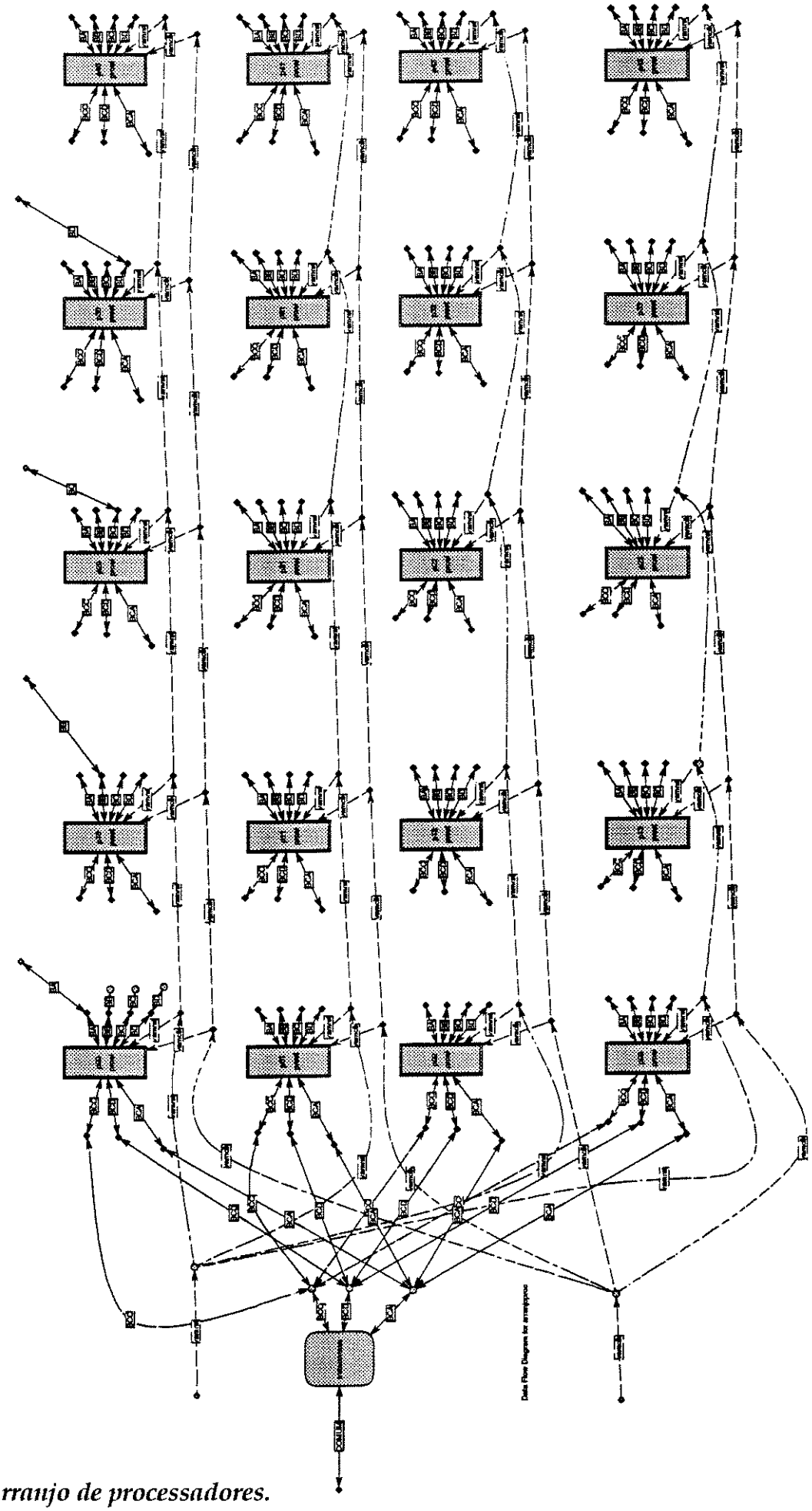


Fig. 12 - DFD do arranjo de processadores.

8. - Descrição da Unidade Elementar de Processamento - PROEL

A unidade de processamento proel é apresentada nesta seção. Uma visão geral da sua interação com elementos externos é mostrada na figura 13, onde é definido o Context Diagram do proel. Naquela figura pode-se destacar que um proel se liga com os barramentos de dados, de endereço e de controle do barramento COMUM, Além disso, ele se conecta por meio de conexões bidirecionais aos quatro barramentos da VAM. A base de tempo das transferências nos barramentos da VAM, é dada pelo sinal de controle vamclk. Para permitir iniciar o funcionamento dos barramentos VAM num estado conhecido, o componente proel recebe o sinal vamrst (reset).

A figura 14 apresenta a Top-level DFD do componente proel. Fundamentalmente, um componente proel é uma unidade processadora completa com memória local -ME, com processador local - PE e uma interface local com a VAM.

8.1 - A Memória Local - ME e o Processador Elementar - PE

É importante destacar neste ponto que a plataforma UGR foi concebida para permitir a ligação de qualquer unidade processadora. Não cabe, portanto, dedicarmos muita atenção aos componentes me e pe do proel. Ele poderão ser descritos de uma forma genérica de forma que o enfase no atual estágio do projeto se dê na definição e exploração das características da interface f com a VAM.

De uma forma genérica o componente me é uma memória compartilhada entre o barramento COMUM e o processador pe do proel. Ela deverá ter uma alta capacidade de armazenamento para permitir que toda a descrição de uma imagem possa ficar ai retida.

O processador pe é descrito de uma forma genérica. Ele possui um barramento de endereços (ender_local), um barramento de dados (dados_local) e ele gera dois sinais de controle que identificam os ciclos de leitura (leitura_local) e os ciclos de escrita (escrita_local). Os seus ciclos de escrita e de leitura são

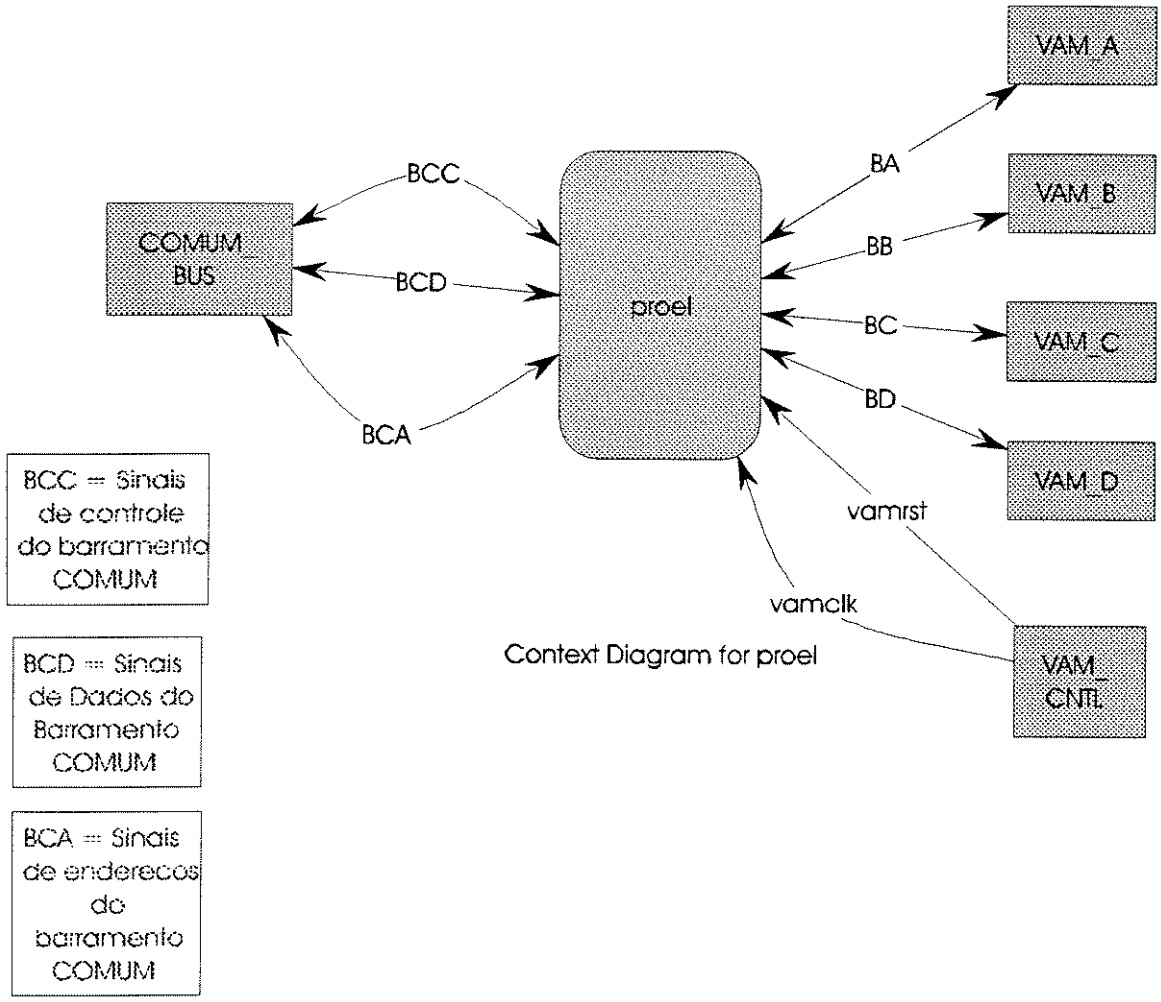


Fig. 13 - Visão Geral do proel - Unidade de Processamento Elementar.

concluídos pela recepção do sinal espera_local. Todos os elementos endereçados pelo processador pe são alocados no seu mapa de memória.

A interface com a VAM, aqui chamada de interface f, é descrita de duas formas alternativas. Na primeira forma (seção 8.2) ela é descrita de uma forma abstrata considerando os seus aspectos funcionais a nível de registradores (Register Transfer Level). Na segunda forma (seção 8.3) ela é descrita procurando enfatizar alguns detalhes de implementação. A colocação destas duas formas neste documenta visa enfatizar a flexibilidade de alteração das descrições dos modelos nesta fase do desenvolvimento evolutivo. Como pode ser constatado, basta agora procurar descrever cada parte do sistema com o nível de detalhes da segunda forma que teremos todo o projeto concluído.

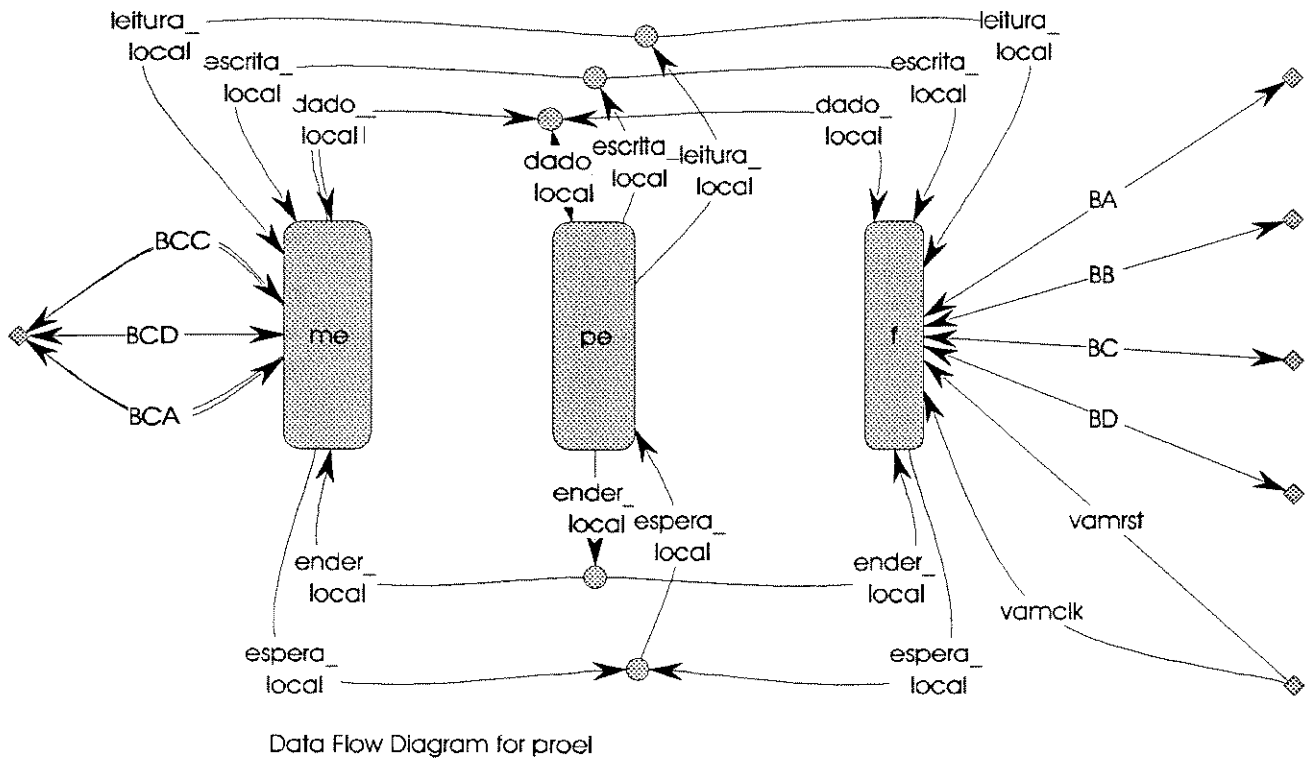


Fig. 14 - Top-Level DFD do PROEL

8.2 - Descrição da Interface com a VAM - F (1ª alternativa - Diagrama de estados)

A figura 15 descreve em forma de diagrama de máquina de estados o comportamento da interface f. Esta descrição resume o comportamento de toda a VAM. Ou seja, tipicamente ela funciona de forma síncrona com o relógio vam_clk possuindo 5 estados possíveis, vam_0, vam_1, vam_2, vam_3 e vam_4. De uma forma geral, cada estado indica a disponibilidade de ocorrer transferências com o nó de memória de quadro de mesmo índice. A mudança de estado é forçado pela simples transição do sinal vamclk. A única mudança assíncrona é a causada pelo sinal vam_rst (reset), quando este sinal ocorrer, independente do estado atual será forçada uma transição para o estado vam_0 (o estado inicial). Na interface f os bits do barramento de endereço são utilizados para identificar o barramento da VAM a ser utilizado, e ainda neste barramento, qual fatia de tempo deverá ser utilizada para a transferência atual. Assim dentro de cada estado, devem ser testados parte dos bits de endereçamento que identificam o barramento e outra parte dos bits que identificam se a

transferência deve ocorrer naquele instante. As simulações realizadas utilizando este modelo estimularam o desenvolvimento do refinamento da descrição. Este refinamento é descrito na seção 8.3.

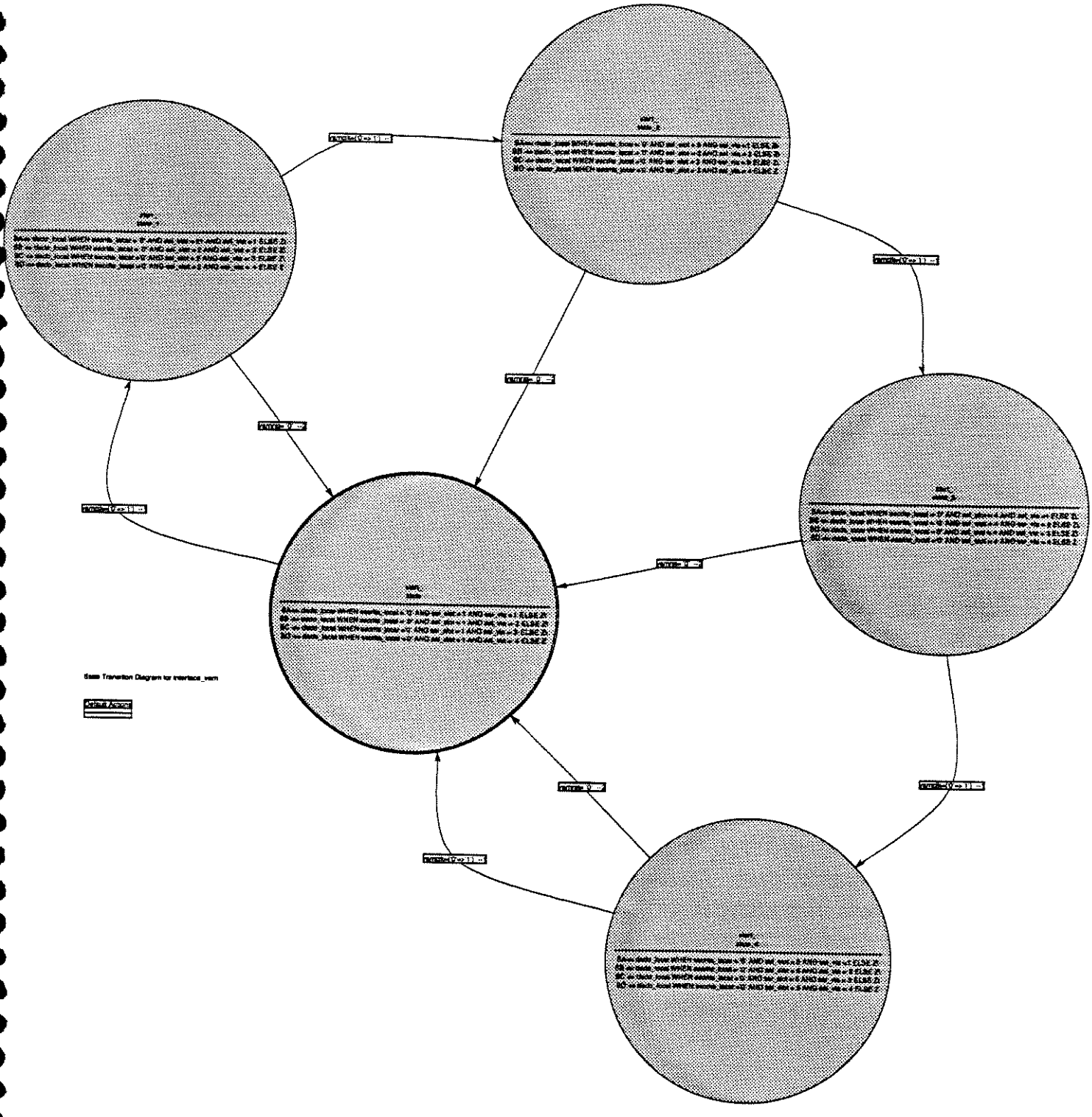


Fig. 15 - Descrição da interface F em diagrama de estados

8.3 - Descrição da Interface com a VAM - F (2ª alternativa - DFD)

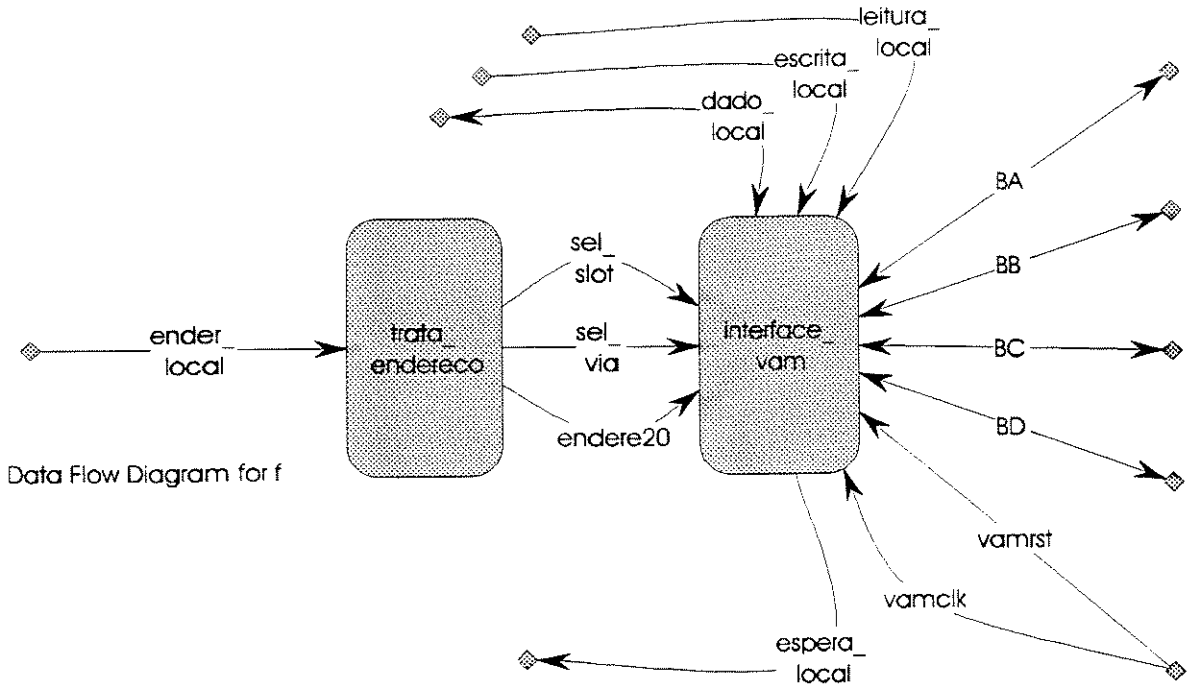


Fig. 16 - A interface F descrita em DFD

A figura 16 descreve em DFD a interface f. Basicamente, pretendeu-se com esta descrição detalhar o circuito de tratamento de endereços do circuito de ligação com a VAM.

8.4 - O Tratamento de Endereço

O circuito de tratamento de endereço separa do barramento de endereço interno ao proel os bits que iram definir o barramento da VAM a ser utilizado, bem como a fatia de tempo da transferência. Trata-se portanto de um circuito decodificador de endereço que gera dois códigos distintos, a saber: sel_slot que identifica a fatia de tempo (melhor dizendo, o estado da VAM), e sel_via que identifica o barramento a ser utilizado. Este circuito deixa passar os demais bits do barramento de endereço de forma que eles irão ser enviados para a memória de vídeo juntamente com a informação de dados.

8.5 - Descrição da Ligação com a VAM

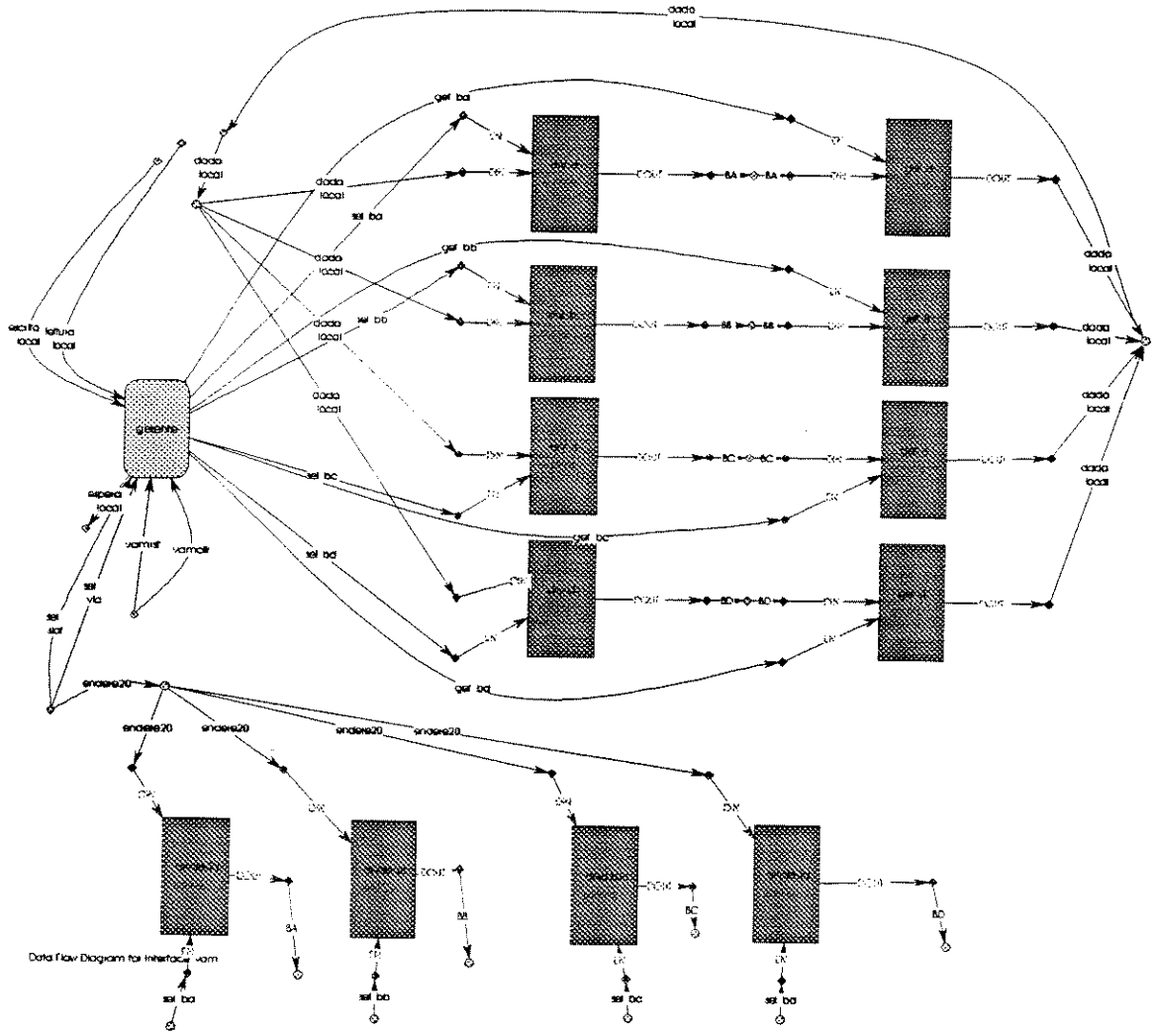


Fig. 17 - Buffers de ligação com a VAM e o circuito de gerenciamento do acesso.

A figura 17 detalhe o circuito de ligação com a vam. trata-se de um conjunto de circuitos acionadores de linha (buffers) com saída tri-state controla pelo circuito de gerenciamento de acesso, circuito gerente.

8.6 - Descrição do Gerenciador da Interface com a VAM - gerente

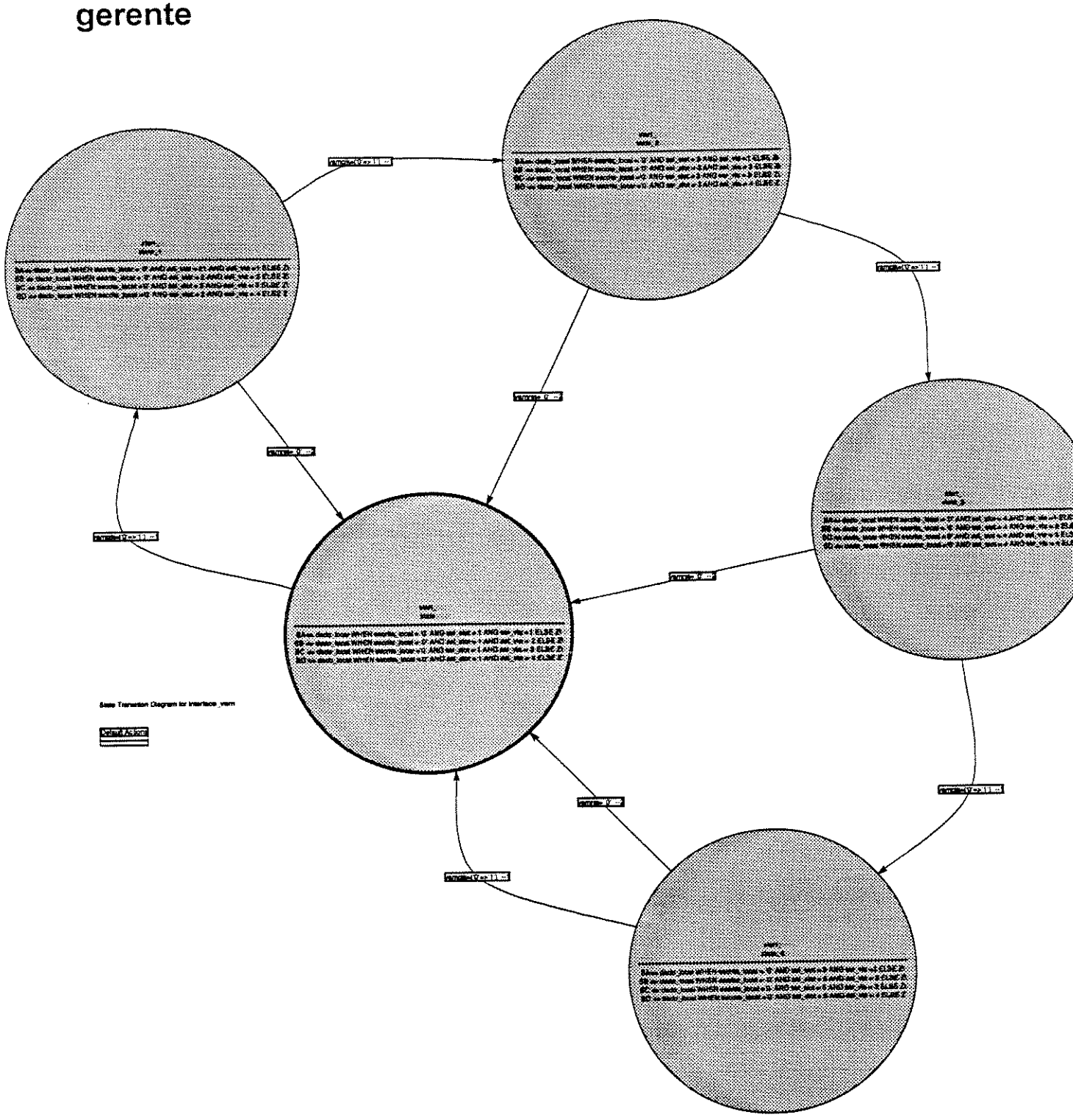


Fig. 18 - O circuito de gerenciamento do acesso a VAM

A descrição do circuito gerente é muito parecida com a máquina de estados que descreve o comportamento da interface f. A principal diferença é que agora são testados códigos específicos para cada slot e barramento.

8.7 - Descrição dos circuitos buffer

```
library mgc_portable;
use mgc_portable.qsim_logic.all;

--
-- Written by LL_to_VHDL at Thu Dec  7 19:47:10 1995
-- Parameterized Generator Specification to VHDL Code
--
--
-- LogicLib generator called: BUFFER
-- Passed Parameters are:
--   tinst name = drvadda
--   parameters are:
--      output_type = TRISTATE
--      W = 20
--      invert_input = NO
--      invert_output = NO
--

-- drvadda Entity Description
entity drvadda is
   port(
      DIN: in qsim_state_vector(19 downto 0);
      DOUT: out qsim_state_resolved_X_vector(19 downto 0) bus;
      EN: in qsim_state
   );
end drvadda;

-- drvadda Architecture Description
architecture rtl of drvadda is
begin
   -- Implementing a TRISTATE buffer.
      TRISTATE: block (EN = '1')
         begin
            DOUT <= guarded qsim_state_resolved_x_vector(
DIN);
         end block TRISTATE;
end rtl;
```

ig. 19 - Descrição em VHDL dos circuitos buffer de acesso à VAM.

A figura 19 mostra a descrição em VHDL do circuito de buffer utilzado na interface f.