UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
POS GRADUAÇAO EM CIENCIA DA COMPUTAÇAO

4954-9

UM PROJETO DE ARQUITETURA PARALELA PARA A GERAÇAO DE IMAGENS REALISTICAS DE OBJETOS TRIDIMENSIONAIS A TEMPO REAL

por

LUIS HENRIQUE DA SILVEIRA LOSS

Dissertação submetida como requisito parcial para
a obtenção de grau de mestre em
Ciência da Computação

Prof. Philippe O. A. Navaux
Orientador

Prof. Anatólio Laschuk
Co-orientador

Porto Alegre, junho de 1988.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Pós-Graduação em Ciência da Computação

Um projeto de arquitetura paralela para
a geração a tempo real de imagens realísticas
de objetos tridimensionais

Dissertação apresentada aos Srs.

Flávio Rech Wagner

Prof. Flávio Rech Wagner

Anatólio Laschuk

Prof. Anatólio Laschuk

Prof. Dante A. C. Barone

Visto e permitida a impressão
Porto Alegre, 26./10./89.

Philippe Navaux

Prof. Philippe O. A. Navaux
Orientador

Ingrid Jansch

Profa. Ingrid E. S. Jansch Pôrto
Coordenadora do Curso de Pós-Graduação
em Ciência da Computação

CATALOGAÇÃO NA FONTE

Loss, Luis Henrique da Silveira.
Projeto de arquitetura paralela para a geração de imagens realísticas de objetos tridimensionais. Porto Alegre, PGCC da UFRGS, 1988.
206p.

Diss. (mestr. ci. comp.) UFRGS-PGCC, Porto Alegre, BR-RS, 1988.

Dissertação: Arquiteturas Paralelas
Computação Gráfica: DDA

AGRADECIMENTOS

Agradeço à IKRO S.A. pelo apoio e incentivo, bem como pela confiança e fé depositadas, ao participar como entidade financiadora da bolsa.

Agradeço ao prof. Phillipe Navaux, meu orientador, pela disposição e sempre pronta atenção no atendimento de minhas atividades e na revisão do trabalho; ao prof. Anatólio Laschuk, pelas dicas e importantes advertências sobre a constituição do trabalho.

Agradeço ao desenhista George Augusto pela grande criatividade e paciência na confecção dos desenhos.

Agradeço ao CPGCC, pelas condições de trabalho oferecidas; aos colegas da IKRO, pelo incentivo e por compreenderem o meu afastamento; e em especial ao Flavio Graziuzo, engenheiro e coordenador do Laboratório de Desenvolvimento da IKRO, pela confiança, apoio e pelo espírito de incentivo à pesquisa.

Agradeço também a meus pais, familiares, amigos, professores, colegas, enfim a todos que de alguma forma contribuiram positivamente para a conclusão do trabalho.

A Edna, com todo
o amor e carinho.

SUMARIO

LISTA DE FIGURAS

RESUMO

Este trabalho aborda o emprego de arquiteturas paralelas para a geração realística de imagens gráficas de objetos tridimensionais, voltada para aplicações de modelagem de sólidos e animação, a tempo real.

Inicialmente são apresentados os aspectos teóricos relevantes à geração de imagens de sólidos com realismo. Após isso, alguns artigos sobre projetos de arquiteturas especiais para geração de imagens tridimensionais a tempo real são exemplificados e discutidos. Por fim, é apresentado o desenvolvimento do GIRTD, um projeto baseado no emprego de arquiteturas especiais para a geração de imagens de sólidos a tempo real. É dada uma ênfase especial ao projeto do DDA, responsável pelo cálculo dos pixels para preenchimento rápido da imagem, com sombreamento contínuo, remoção de partes ocultas e antialiasing.

ABSTRACT

This work discusses the use of parallel architecture for realistic 3-D graphic image rendering, for real-time modelling and animation systems.

Initially it presents the theory related to realistic solid image generation. Then some architecture projects for real time 3-D image generation are presented and discussed. Finally the development of a real-time 3-D system for image generation, the GIRTD, is presented. A more intensive attention is given to the DDA project, which performs the pixel calculus for fast image filling with continous shading, hidden surface removal, and antialiasing.

1 - INTRODUÇÃO

1.1 - Apresentação

A computação gráfica tem tido um importante papel na nossa sociedade. Desenvolvida com o intuito de facilitar e dinamizar a comunicação entre homem e máquina, ela está presente em quase todas as áreas que fazem uso das facilidades oferecidas pelos computadores.

Por computação gráfica (C.G.) entende-se qualquer criação e manipulação de imagens gráficas por intermédio do computador. Este é um recurso cuja utilização tem se tornado fundamental junto à maioria das aplicações dos computadores. Muitos esforços estão sendo aplicados em pesquisas e desenvolvimentos tecnológicos nesta área com a finalidade de viabilizar este poderoso e eficiente recurso, que é a C.G.. Por esta razão esta é uma das áreas da ciência da computação que mais tem evoluído desde a década de 60.

Surgida a partir da idéia de que textos e tabelas nem sempre são suficientes para transmitir informações de maneira clara e com interpretação garantida, a C.G. está sendo amplamente utilizada em qualquer sistema que interaja com o usuário, fazendo o uso de imagens gráficas para a comunicação com este. Assim o usuário está livre da interpretação de textos ou tabelas, sempre sujeitas a concepções errôneas, podendo receber informações muito mais eficientemente.

Inicialmente a necessidade da C.G. era evidente mas seu uso inviável, devido a carência de recursos tais como estações gráficas e processadores capazes de computar as informações gráficas de maneira satisfatória, a menos que sua necessidade justificasse o grande empenho e custo necessários para atingir estes propósitos, como ocorria em sistemas de segurança.

Atualmente a C.G. tem aplicações que vão desde as de mais baixa escala como video-games, equipamentos baratos e popularizados em todo o mundo, até em sistemas altamente inteligentes de processamento de imagens ou de reconhecimento de padrões de imagens de satélite, ou na reconstituição de imagens tridimensionais de partes do corpo humano para análises clínicas, a partir de dados obtidos por medicina nuclear.

Um emprego muito significativo para a C.G. está nos sistemas de CAD (computer aided design). Estes são sistemas de projeto industrial assistido por computador, amplamente utilizados nas indústrias modernas. Os sistemas permitem, por exemplo, que um usuário possa projetar peças mecânicas através da manipulação direta de imagens em perspectiva, com grande facilidade e sem a necessidade de muito treinamento, além de poder dispor de certas facilidades como a simulação de esforços e desgastes destas peças mecânicas e da documentação do projeto, também executados pelo computador.

Essa popularização da C.G. deve-se , em parte, aos grandes avanços tecnológicos em micro-eletrônica, pois produziram circuitos integrados mais baratos e que incorporaram muito mais recursos em um mesmo encapsulamento resultando em processadores poderosos e rápidos. A

utilização destes circuitos de alta integração, acompanhada por um desenvolvimento de software especial, viabilizou o uso abundadnte da C.G.. Paralelamente, outros desenvolvimentos ocorreram também nas tecnologias dos dispositivos de saida gráfica utilizados em C.G., tais como monitores de video, impressoras gráficas, plotters e outros, que também foram fundamentais para a evolução da C.G..

1.2 - Origens da Computação Gráfica [NEW 79]

Em 1950 pela primeira vez um monitor de video foi empregado para a representação de imagens gráficas simples. Era um monitor do tipo CRT, utilizado nas televisões, conectado em um computador do MIT.

Ao longo desta década a C.G. teve poucos progressos porque os computadores da época eram inviáveis para aplicações interativas. Somente no final da década, com o desenvolvimento de novas máquinas, a computação interativa se tornou possivel e o interesse por C.G. começou a crescer rapidamente.

A história da C.G. teve o seu principal marco em 1962, com a publicação da tese de doutorado de Ivan Sutherland. Esta tese, entitulada "A Man-Machine Graphical Communication System", provou, para muitos, que a computação gráfica interativa era um campo de pesquisa viável, útil e interessante. A partir de então vários projetos e pesquisas em computação gráfica foram desenvolvidos em diversas centros de pesquisa como o MIT, a General Motors, a Bell Telephone Laboratories e a Lockheed Aircraft.

A década de 60 representa a era da pesquisa em C. G. enquanto que a década de 70 representa a década em que estas pesquisas começaram a ter frutos. Sistemas gráficos agora são aplicados nas mais diversas áreas e a sua popularidade certamente continuará crescendo nos próximos anos.

1. 3 - Principais aplicações da Computação Gráfica

As áreas onde há utilização intensa da C. G. e que merecem ser destacadas são a indústria, engenharia, medicina, química, arquitetura, matemática e artes.

A indústria é talvez uma das áreas que mais está tirando proveito dos recursos da C. G., através dos modernos sistemas de CAD-CAM (Computer Aided Design e Computer Aided Manufacture). Estes sistemas distribuem recursos de um computador pelos diversos setores de uma indústria de maneira a que todas as etapas do desenvolvimento e da produção de um determinado produto fiquem centralizadas num único banco de dados do computador central. Assim, diversas etapas do desenvolvimento de um produto podem ser realizadas nos setores responsáveis por estas, fazendo uso destes recursos, sem a necessidade da transferência de documentos físicos de um setor para outro pois estes estão disponíveis no computador. O computador formaliza a entrada de documentos de engenharia e de processos, efetua a comunicação entre etapas e a emissão de listagens de documentos, controla cronogramas, auxilia o projeto e monitora e controla a fabricação.

O CAD-CAM está intimamente ligado à C.G. no sentido em que a maioria das informações deve ser representada graficamente, tanto durante o projeto de um produto quanto na emissão de relatórios e na geração de fitas para fabricação por controle numérico. O projetista pode fazer uso de pacotes de geração de imagens 2D ou 3D para a confecção do modelo interno ao computador e de outros pacotes para testes simulados de comportamento do produto na sua aplicação em campo, todos estes pertencentes ao conjunto que é o sistema CAD-CAM. Inúmeras outras tarefas dentro do sistema também dependem da C.G..

A medicina faz uso da C.G. na vizualização e análise clínica de imagens 2-D ou 3-D de partes do organismo dos pacientes, obtidas por radiografia, tomografia, ecografia ou ressonância magnética.

Na engenharia a C.G. participa na pesquisa de materiais e estruturas. Aqui o uso de análise de elementos finitos se dá paralelamente à C.G. na simulação de esforços físicos e desgastes.

Na química a C.G. tem facilitado a visualização de estruturas moleculares tridimensionais, o que é um recurso importante na pesquisa de novos compostos.

A matemática tem se aproveitado da C.G. na vizualização gráfica de comportamentos matemáticos, como funções tridimensionais ou expansões matemáticas.

Para a arquitetura existem sistemas de exibição em perspectiva com realismo de ambientes ou construções que permitem ao arquiteto uma boa avaliação de seus projetos.

As artes gráficas encontram na C.G. um novo e ilimitado recurso de exploração artistica. Aqui a C.G. pode ser utilizada como uma poderosa ferramenta na geração de cenas animadas espaciais para televisão.

Uma especial aplicação da C.G. está no treinamento de pilotos de avião, através de simuladores de vôo onde todo o ambiente de rotas aéreas e de aeroportos pode ser simulado através de animação em tempo real. Assim o piloto pode aprender a fazer decolagens, aterrisagens e manobras, sem riscos de vida e gastos de combustivel, além de poder se familiarizar com diversos aeroportos de todo o mundo.

1.4 - Area de especial interesse: Geração de imagens de objetos sólidos

Uma boa parte destas aplicações depende de técnicas de exibição de imagens tridimensionais sobre uma tela plana de computador, como em simulação de vôo e modelagem de sólidos. Diversos problemas são inerentes ao emprego destas técnicas, como a projeção da imagem, a remoção de partes ocultas e a simulação de realismo. Uma técnica especial, que vem sendo utilizada intensamente, é o emprego de arquiteturas paralelas de processamento para a geração, em alta velocidade, de imagens gráficas de objetos tridimensionais. O emprego desta técnica viabiliza aquelas aplicações em que a imagem tem que ser gerada instantaneamente, como em sistemas interativos rápidos, e aplicações a tempo real, como simuladores de vôo e animação.

Este trabalho se propõe a abordar estes aspectos. Para isso o capitulo II é voltado à apresentação dos

conceitos teóricos e práticos das técnicas comumente empregadas para a geração de imagens de objetos tridimensionais. O capitulo III dedica-se à investigação da aplicação de arquiteturas paralelas de processamento para a geração de imagens de sólidos em alta velocidade, através de exemplificação com artigos sobre projetos desta classe. No capitulo IV é apresentado um projeto de arquitetura paralela para a geração, a tempo real, de imagens de objetos sólidos com realismo. Por último, no capitulo V, são tecidas algumas conclusões sobre este projeto.

2 - CONCEITOS TEORICOS EM GERAÇAO DE IMAGENS REALISTICAS TRIDIMENSIONAIS

Considerando que se quer explorar este recurso, que é a exibição de imagens de sólidos no espaço em uma tela plana, como a de um monitor gráfico, este capitulo dedicar-se-á a apresentar sucintamente a teoria envolvida neste processo bem como as técnicas necessárias para tal.

Para a exibição de imagens de sólidos as seguintes questões são naturalmente levantadas:

- Como representar os dados espaciais dentro de um computador?
- Como representar a terceira dimensão, ou profundidade, em uma imagem plana?
- Como eliminar da imagem partes de objetos que não são visiveis por estarem obscurecidas por outras?

Ao se considerar também o realismo deve-se ter a preocupação de modular, na imagem, a luminosidade e a coloração das superficies dos objetos de maneira a se ter uma idéia do sombreamento causado por fontes de luz localizadas no espaço. Aí surge outra questão: como modelar estes efeitos realisticos.

Este capitulo procura esclarecer as questões acima mencionadas, não se atendo aos aspectos de computação gráfica que não estejam relacionados com o objetivo deste trabalho. Maiores informações podem ser obtidas em livros de computação gráfica, tais como *[NEW 79]* e *[HEA 86]*.

2.1 - Técnicas de representação de dados em 3-D

Para a representação de dados em 3-D inevitavelmente faz-se uso de um sistema de referência espacial, implicita ou explicitamente. Dentre todos os sistemas de referência espaciais o mais usual é o sistema de eixos cartesianos X-Y-Z, talvez por seus componentes serem de mesma dimensão escalar, ao invés de um sistema polar, que emprega ângulos e distâncias.

Neste sistema um objeto é descrito por um conjunto de referências ao espaço 3-D, através das coordenadas de posições no espaço, estabelecendo a geometria do objeto. Uma representação real necessitaria do conhecimento de todos os pontos dos objetos, o que é impraticável já que necessitaria de uma quantidade infinita de referências. Na prática o que se faz é uma aproximação da geometria dos objetos por um número finito e mínimo de referências de maneira que a conseqüente imprecisão seja imperceptível pelos olhos humanos, ou pelo menos insignificante para uma correta concepção dentro da aplicação a que se destina.

As técnicas de representação de dados 3-D em computadores podem ser chamadas também de técnicas de aproximação, pois cada uma utiliza uma maneira diferente de aproximar algo por uma descrição finita. As principais técnicas são:

- A B-rep (Boundary representation), que é a representação de sólidos por suas superficies. O modelo facetado é um caso particular do B-rep, onde as superficies são representadas por polígonos. Por isso pode também ser chamada de técnica B-rep poliédrica.
- A CSG (Construtive Solid Geometry), em que os objetos são constituídos por sólidos primitivos, como cubos, esferas e

cilindros, combinando-se por operações do tipo união, intersecção e diferença.

- Por último, a técnica Oc-tree, definida recursivamente pela divisão do espaço em octantes e estes, por sua vez, também divididos enquanto forem cortados pela superfície limite dos objetos, até um limite prático.

Cada técnica tem sua razão de ser pelas vantagens que oferece para determinadas aplicações. Veremos a seguir uma por uma estas três principais técnicas, suas vantagens e desvantagens e as aplicações que lhe são apropriadas.

Outras técnicas são utilizadas menos freqüentemente, tais como representação por equações paramétricas, rotação de perfis, fractals, etc *[HEA 86]*.

2.1.1 - B-rep poliédrico

Nesta técnica os objetos são representados pela superfície limite entre o interior e o exterior destes. Esta superfície é aproximada por polígonos justapostos pelas suas arestas, formando um poliedro fechado e sem furos. Os únicos pontos representativos dos objetos passam a ser os vértices dos polígonos. A topologia dos poliedros é garantida pela estrutura de dados hierárquica que os contém, como sugere a figura 2.1. Os objetos são definidos por um conjunto de polígonos através de um conjunto de apontadores para polígonos. Estes por sua vez são definidos por um conjunto ordenado de vértices, ou apontadores para vértices. Somente os vértices possuem informações de coordenadas espaciais X, Y e Z.

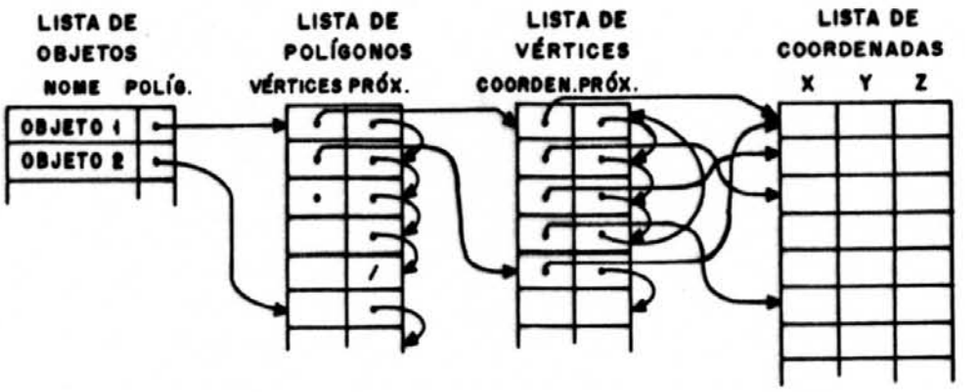


fig 2.1 - Estrutura de dados para representação B-rep

A estrutura de dados garante a formação dos poligonos por vértices e a formação de poliedros por poligonos. As arestas são linhas retas, portanto para especificá-las só são necessários seus pontos extremos, não se necessitando os pontos intermediários. Os poligonos, analogamente, são planos e portanto só precisam de suas arestas para serem representados, e assim por diante.

Qualquer transformação linear, de vista ou de perspectiva, somente necessita ser aplicada nas coordenadas dos vértices, sem a necessidade de mudar-se a estrutura de dados.

Esta estrutura apresentada na figura 2.1 pode sofrer variações, conforme a aplicação. Pode haver a necessidade de uma lista explicita de arestas, por exemplo.

A técnica B-rep é apropriada para sistemas em que quer-se conhecer grandezas volumétricas dos sólidos, tais como volume, centro de massa ou momento de inércia, e grandezas superficiais, como a área de um sólido.

Os sistemas que fazem uso da representação B-rep geram suas imagens usando algoritmos simples do tipo Z-

buffer, onde as imagens são construidas poligono por poligono, em qualquer ordem, com a remoção das parte ocultas feita automaticamente pelo algoritmo.

Esta é a técnica mais difundida devido a sua facilidade de operação e relativa baixa utilização de memória. E a técnica comumente empregada nos sistemas de CAD, em modelagem de sólidos, e em várias outras aplicações onde é permitido que sólidos sejam representados apenas pela sua superficie.

2.1.2 - CSG - Sólidos por Geometria Construtiva

Na técnica de representação CSG os objetos são definidos por um conjunto de sólidos primitivos, tais como cubos, cilindros, cones, esferas ou paralelepipedos, operados entre si por funções lógicas como união, intersecção e diferença. Esta técnica é bastante utilizada em modelagem de peças mecânicas, na indústria, pois a grande maioria dessa classe de sólidos pode ser definida por partes ou uniões destes sólidos primitivos. Um exemplo é a modelagem da cruzeta mostrada na figura 2.2. A seqüência de operações lógicas para a criação deste objeto é representada sob a forma de uma árvore de operações.

Nas folhas temos a descrição dos sólidos primitivos e nas sub-raizes temos as operações lógicas entre estes sólidos primitivos ou sólidos intermediários gerados.

Nas folhas, os sólidos primitivos necessitam de parâmetros que descrevam as suas dimensões bem como sua posição e orientação.

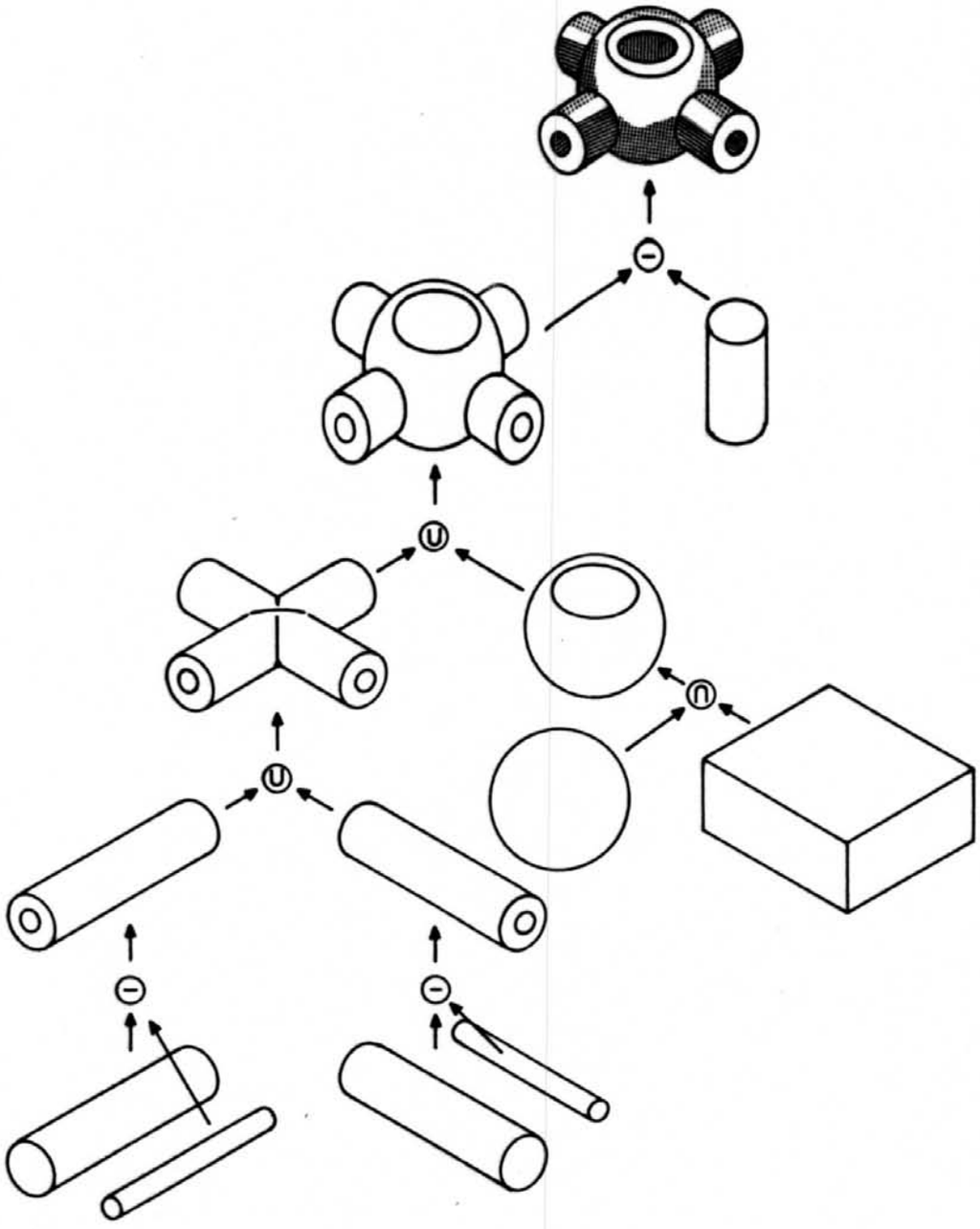


fig 2.2 - Cruzeta modelada por CSG

Um sistema de modelagem interativo usando CSG é de fácil manipulação e requer pouco treino, pois é mais fácil imaginar-se objetos gerados por primitivas. Além disso o ferramental para fabricação destas peças é mais fácil de ser especificado, uma vez que os primitivos individuais podem

ser facilmente moldados por ferramentas comuns, tais como frezas ou brocas.

A modelagem por CSG é simples mas a conversão de volta para a representação por superficies é trabalhosa. Uma outra limitação do CSG é a variedade de primitivas disponives. Alguns sistemas de modelagem CSG permitem a criação de novos sólidos primitivos, por parte do usuário, a partir de sólidos já obtidos por CSG ou primitivos mais complexos, definidos por expressões matemáticas, por exemplo. Assim o usuário pode criar uma biblioteca de sólidos para seu uso próprio.

2.1.3 - Oc-trees - árvores de octantes

Oc-tree é uma técnica moderna capaz de representar um objeto sólido em toda sua extensão de massa. Por isso é atrativa, por exemplo, para a área médica de análises clínicas por computador, onde a representação tridimensional de partes internas do organismo humano, obtidas por tomografia, auxilia a análise clínica. E uma técnica interessante por ocupar uma relativamente pequena quantidade de memória, se comparada com outras técnicas de representação pela massa dos sólidos.

Os objetos são capturados por um cubo, ou janela tridimensional de visualização, de maneira que este cubo os contenha totalmente ou suas partes de interesse. O cubo é então subdividido em 8 cubos menores, os 8 octantes, e estes sub-cubos, sucessivamente, são divididos em 8 até que estejam completamente dentro ou fora dos objetos e não tenha sido atingida a resolução máxima.

Assim, enquanto houver um sub-octante que seja cortado pela superficie limite dos objetos este será recursivamente subdividido até a obtenção de células homogêneas ou que se tenha sido atingida a resolução limite.

A representação de objetos por Oc-tree é armazenada sob a forma de uma árvore em que cada nó se subdivide em 8 sub-árvores, representando os octantes. As folhas representam sub-octantes que estão dentro ou fora dos objetos, identificados por 1 ou 0 respectivamente, ou são aquelas que atingiram a resolução máxima. Uma árvore Oc-tree é exemplificada na figura 2.3 com 3 niveis de subdivisão.

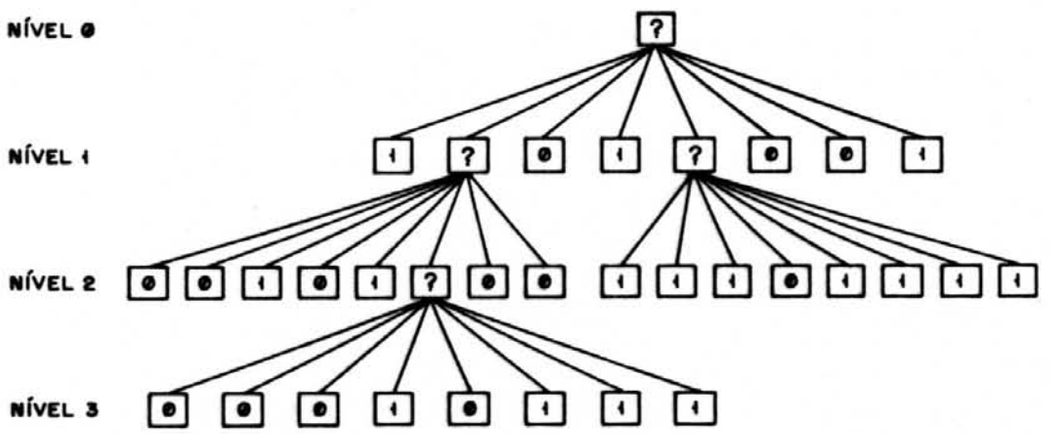


fig 2.3 - Arvore de Representação Oc-tree

Entre as vantagens da representação Oc-tree podemos citar:

- possibilidade de representação de corpos ocos
- facilidade de obter-se o perfil de cortes planos
- translação e rotação em múltiplos de 90· simples
- determinação rápida de propriedades volumétricas
- facilidades com operações booleanas

Os problemas que o uso de Oc-trees podem trazer são:

- a dificuldade da geração de imagens com suavização das superficies
- reobtenção da representação por superficies
- utilização de bastante memória

2.2 - Transformações de vista e de perspectiva

Transformações geométricas desempenham um importante papel na geração de imagens de cenas tridimensionais. Elas são usadas em modelagem para expressar a localização dos objetos em relação a outros. Na geração de uma vista de uma cena estas são usadas para obter o efeito de diferentes posições de direções do ponto de vista. Por fim, as transformações em perspectiva também são usadas para projetar uma cena em 3-D em uma tela 2-D. Há casos em que o ponto de vista muda rapidamente ou objetos movem-se uns em relação aos outros, como em animação. Aí estas transformações são efetuadas rapidamente, uma após a outra.

Embora essas transformações sirvam também para o movimento relativo de partes individuais da imagem, nos preocuparemos aqui apenas com o aspecto da geração da imagem plana dos objetos.

A projeção de imagens 3-D para a obtenção de uma cena 2-D faz uso de um modelo matemático baseado em uma câmera fotográfica que captura as imagens espaciais e as projeta em um plano, a película fotográfica. Este modelo leva em conta a infinidade de orientações e posições possíveis que esta câmera pode assumir, significando diferentes tomadas de cenas.

A primeira fase deste modelo matemático é a mudança de sistema de eixos. Os objetos entregues ao sistema são referenciados por um sistema de eixos universal, o SRU. A mudança de eixos calcula as coordenadas dos objetos segundo um sistema de eixos fixo à câmera. Para isso uma transformação de translação e outra de rotação devem ser aplicadas sobre as coordenadas dos objetos para que estes passem a ser referenciados pelos novos eixos coordenados fixos à câmera. Por fim uma operação de projeção é feita de maneira a dar uma sensação de profundidade à imagem, onde os objetos mais distantes da câmera fotográfica, ou câmera sintética, aparecerão menores.

Na figura 2.4 vemos esta analogia da câmera fotográfica.

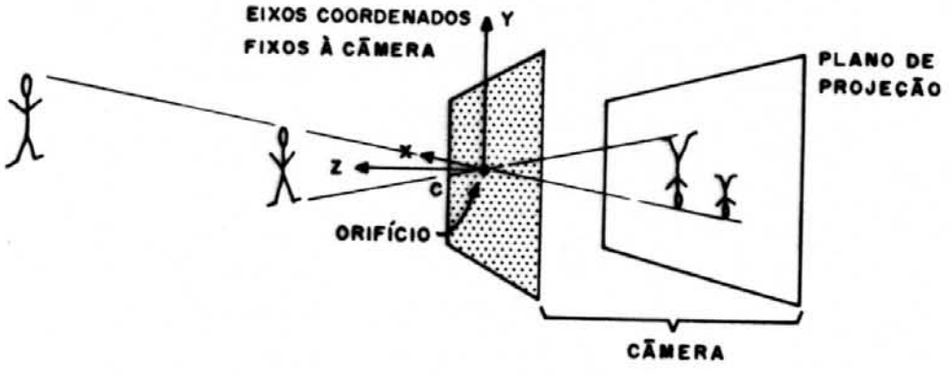


fig 2.4 - O modelo da câmera fotográfica

A imagem passa por um furo estreito, o centro focal, e se projeta invertida na película. O sistema de eixos da câmera tem sua origem coincidindo com este centro focal. Os seus eixos X, Y e Z são fixos a ela. O nosso modelo matemático tem a película fotográfica, ou plano de projeção, à frente

do centro de projeção, ao contrário da câmera real, assim evita-se a inversão da imagem. O espaço entre o centro focal e o plano de projeção é chamado distância focal.

Para a determinação da imagem projetada no plano de projeção a descrição dos objetos deve passar por três processos, a translação, a rotação e a projeção.

2.2.1 - A translação e a rotação

A translação depende das coordenadas da câmera representadas pelo seu céntro focal, C: (X_c, Y_c, Z_c). Todos os pontos representativos dos objetos no espaço tem suas coordenadas subtraídas pelas coordenadas do centro focal C, fazendo-se com que os objetos passem a ser descritos como se a origem do seu sistema de eixos fosse deslocada para o ponto C. No caso de uma descrição B-rep dos objetos, por exemplo, somente os vértices das faces dos objetos necessitam ser transformados.

A rotação, por sua vez, é efetuada por um produto matricial entre o vetor de coordenadas x-y-z e a matriz rotação R. E uma matriz 3x3 onde cada termo $R_{i,j}$ expressa o produto escalar entre o versor i do sistema de eixos do universo e o versor j do sistema de eixos da câmera, onde i ou j $\in \{1, 2, 3\}$, referindo-se, respectivamente aos versores (vetores unitários) dos eixos X, Y e Z. Isso significa, por exemplo, que se o eixo X do sistema de referência do universo é paralelo ao eixo Y do sistema de eixos de referência da câmera, segundo a mesma orientação, então $R_{1\,2} = 1$. Se estes forem versores opostos $R_{1\,2} = -1$ e se forem ortogonais $R_{1\,2} = 0$ pois o produto escalar é zero.

Uma caracteristica desta matriz, que está relacionada com o fato de não causar distorção nos objetos apenas rotação pura, é que a soma dos quadrados dos três termos de qualquer linha ou coluna é sempre 1.

Os termos $R_{i\,j}$ da matriz normalmente são expressos por funções seno e/ou cosseno dos ângulos formados entre a orientação da câmera e o universo.

A translação e rotação é expressa por:

$$P_r = (P_u - C) \times R$$

onde $P_r = [X_r, Y_r, Z_r]$ o ponto descrito segundo os eixos da câmera;

$P_u = [X_u, Y_u, Z_u]$ o ponto descrito segundo os eixos do universo;

$C = [X_c, Y_c, Z_c]$ a posição da câmera segundo os eixos do universo;

$$R = \begin{bmatrix} R_{1\,1} & R_{1\,2} & R_{1\,3} \\ R_{2\,1} & R_{2\,2} & R_{2\,3} \\ R_{3\,1} & R_{3\,2} & R_{3\,3} \end{bmatrix}$$

expressa a orientação da câmera em relação aos eixos de referência do universo,

ou seja:

$$\begin{bmatrix} X_r & Y_r & Z_r \end{bmatrix} = \begin{bmatrix} X_u - X_c & Y_u - Y_c & Z_u - Z_c \end{bmatrix} \times \begin{bmatrix} R_{1\,1} & R_{1\,2} & R_{1\,3} \\ R_{2\,1} & R_{2\,2} & R_{2\,3} \\ R_{3\,1} & R_{3\,2} & R_{3\,3} \end{bmatrix}$$

Se for escolhido o eixo Z como a direção de vista e o plano X-Y paralelo à película fotográfica, neste plano teremos a projeção paralela da imagem simplesmente desconsiderando o eixo Z. A figura 2.5 mostra esse tipo de projeção.

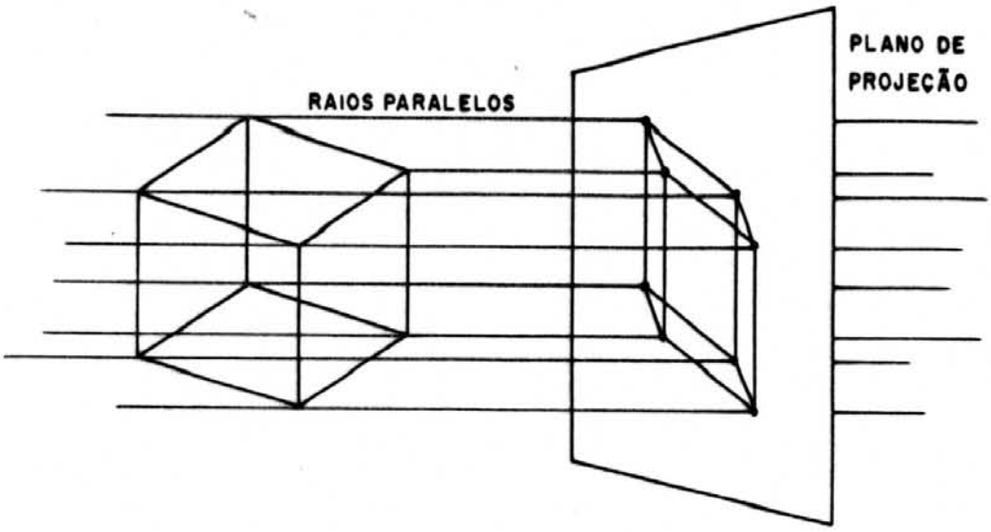


fig 2.5 - Projeção paralela

A imagem assim obtida não considera a profundidade dos objetos. Um objeto afastado apresenta a mesma dimensão de um próximo pois estes são projetados paralelamente, como a sombra de uma fonte de luz distante.

Esse tipo de imagem é ainda usado em sistemas onde a perspectiva não é fundamental, por exemplo na visualização de funções matemáticas de duas variáveis ou em alguns sistemas simples de CAD. Há também aplicações que usam projeção paralela para preservar as dimensões dos objetos, permitindo assim uma avaliação dimensional.

2.2.2 - Projeção em perspectiva

Para a obtenção de uma imagem projetada como na figura 2.4, com a percepção do afastamento com a profundidade, utiliza-se mais uma transformação que é a projeção em perspectiva. Essa transformação computa o efeito da câmera fotográfica, onde os raios de luz que passam pelo orifício são projetados na película, não com raios paralelos, como na projeção paralela, mas com raios que

passam todos por um mesmo ponto, o centro de convergência. A configuração desses raios passando por um ponto comum tem um aspecto de um cone, daí porque é também chamada projeção cônica.

O significado da transformação em perspectiva é visto na figura 2.6.

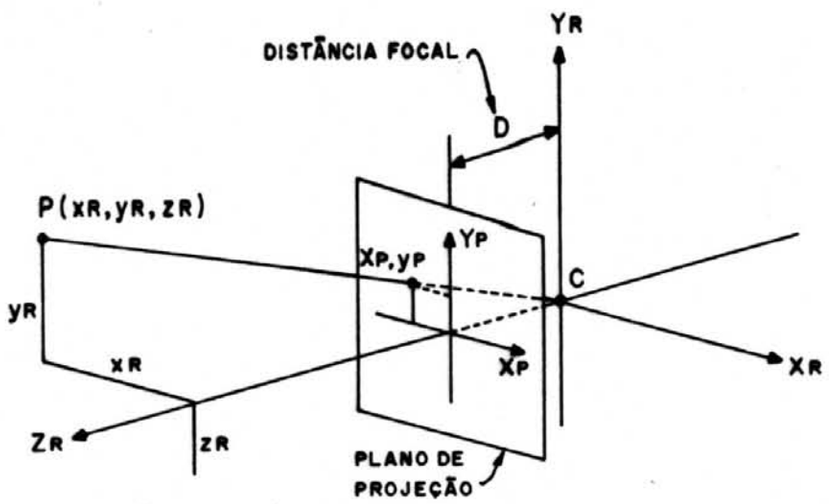


fig 2.6 - Projeção em perspectiva

A projeçao x_p , y_p do ponto P projetado no plano é dada por:

$$x_p = x_r \frac{D}{z_r} \qquad y_p = y_r \frac{D}{z_r}$$

As coordenadas X e Y são reduzidas segundo uma proporção inversa ao afastamento Z, fazendo com que objetos distantes fiquem proporcionalmente menores, como ocorre em uma câmera fotográfica, ou na nossa visão. x_p e x_r são as coordenadas da projeção no plano. Se quisermos que elas sejam independentes da dimensão do plano de projeção divide-se x_p e y_p por S, que é a altura e largura do plano, assim x_s e y_s variam de -1 a 1.

Os objetos agora representados por (x_p, y_p, z_r) possuem uma distorção causada pela não linearidade da divisão por 1/z. Segmentos que antes eram retos agora são curvos. Planos também se tornam curvados. A descrição dos objetos, pelos vértices, agora não carrega implicitamente a certeza de que os pontos intermediários das arestas podem ser obtidos por interpolação entre os vértices. Para que a interpolação da profundidade nos pontos intermediários seja correta, o que é importante para a remoção de partes ocultas, é essencial que linhas retas antes da projeção também sejam linhas retas após a projeção assim como planos continuem planos. Para que isso ocorra uma correção deve ser feita sobre z_r, conforme:

$$z_p = \frac{Z_{max}}{Z_{max} - Z_{min}} - \frac{1}{z_r} \times \frac{Z_{max} \cdot Z_{min}}{Z_{max} - Z_{min}}$$

Com esta expressão z_p varia entre 0 e 1 enquanto z_r varia entre Z_{min} e Z_{max}, respectivamente, de uma maneira não linear capaz de compensar a não linearidade das expressões para x_p e y_p. Z_{min} e Z_{max} são a menor e a maior profundidade encontrada no conjunto de vértices projetados.

2.3 - Técnicas de visualização

Nos primórdios da computação gráfica toda a geração de imagens gráficas por computador era baseada na exibição de segmentos de retas, devido às caracteristicas dos monitores disponiveis então. Diversas técnicas foram desenvolvidas com o intuito de gerar imagens planas de objetos tridimensionais apenas com o uso de segmentos de

retas. Atualmente dispõe-se de equipamentos capazes de gerar imagens bem mais reais, com a reprodução completa de suas superficies. Entretanto, a exibição por segmentos, ou vetores, ainda é bastante utilizada hoje em sistemas mais simples, enquanto que a exibição pelas superficies dos objetos ainda está restrita a aplicações mais profissionais.

Estas duas abordagens, a exibição por vetores, ou wire-frame, e a exibição por superficies, trouxeram grandes contribuições para a computação gráfica, através de técnicas que são largamente utilizadas hoje. Estas serão sucintamente apresentadas aqui.

2.3.1 - Exibição por Wire-Frame

A representação de imagens por wire-frame caracteriza-se por exibir objetos espaciais apenas por segmentos de retas, os quais explicitam alguns elementos geométricos destes, tais como perfis, arestas ou bordas, evidenciando-nos a anatomia desses objetos.

A simplicidade de uso é um fator atrativo para o seu emprego, além de ser bastante satisfatória numa vasta gama de aplicações. As transformações de vista aplicadas aos vértices das faces dos objetos são implicitamente aplicadas também em suas arestas, em todos os seus pontos, pois essas transformações mantêm a caracteristica retilinea das arestas.

A aplicação mais comum está na representação de sólidos por B-rep, os quais podem ser desde superficies matemáticas até modelos de projeto industrial, como exemplificado na figura 2.7.

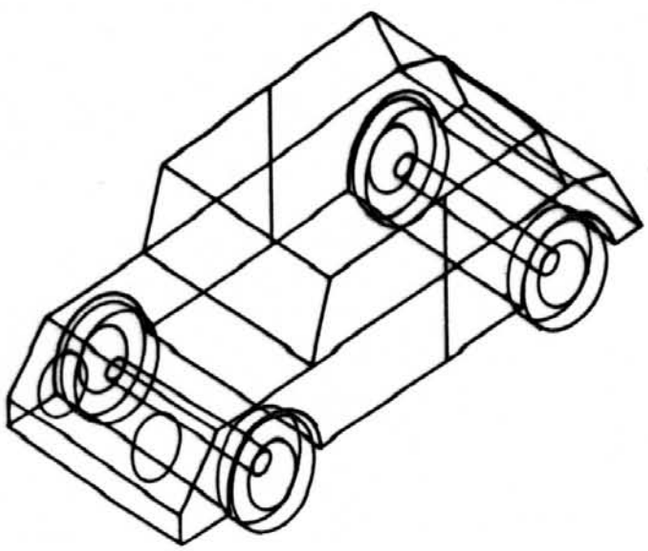


fig 2.7 - Representação de um objeto sólido por wire-frame

Podemos ver que esta forma de representação dá uma boa idéia da realidade dos sólidos simples. Além disso podemos ver detalhes de ambos os lados dos objetos, o frontal e o de trás, como se o objeto fosse construído apenas por arames retos interligados, daí o nome de wire-frame. Esta representação possibilita uma concepção completa do objeto por apenas uma vista. Entretanto, para sólidos com uma maior quantidade de detalhes, a imagem pode ficar confusa devido à quantidade excessiva de vetores sobrepostos.

Um outro problema que pode ocorrer, principalmente quando se usa projeção paralela, é o efeito de ambigüidade, causada pelo fato das faces serem transparentes. Na figura 2.8 a) vemos um cubo, projetado paralelamente, em que não conseguimos distinguir qual das duas vistas, apresentadas em b) e c), este cubo realmente representa, portanto sua representação em a) é ambígua. Em d) vemos que a representação do mesmo cubo por projeção em perspectiva reduz um pouco essa ambigüidade.

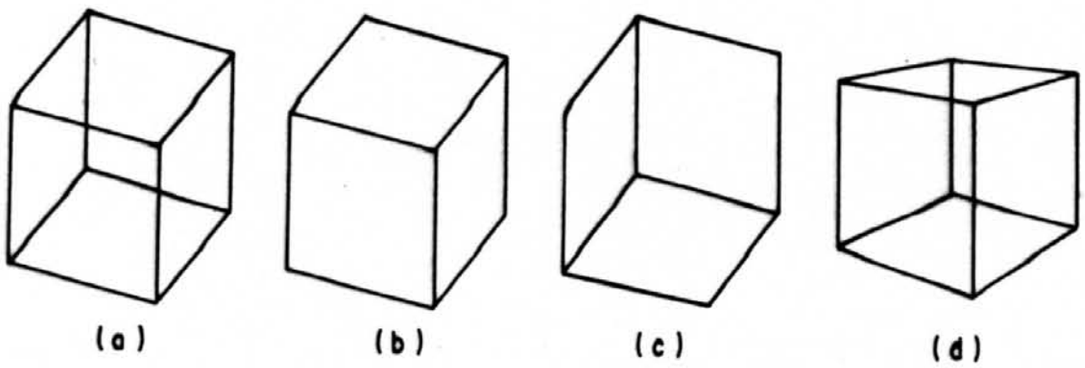


fig 2.8 - Ambiguidade do wire-frame

Diversas soluções existem para esses problemas, tais como a intensificação das arestas mais próximas, o uso de rotação da imagem, vistas estereoscópicas, etc *[NEW 79]*. A solução usual é a remoção das arestas ocultas. Uma imagem com essas caracteristicas é mostrada na figura 2.9.

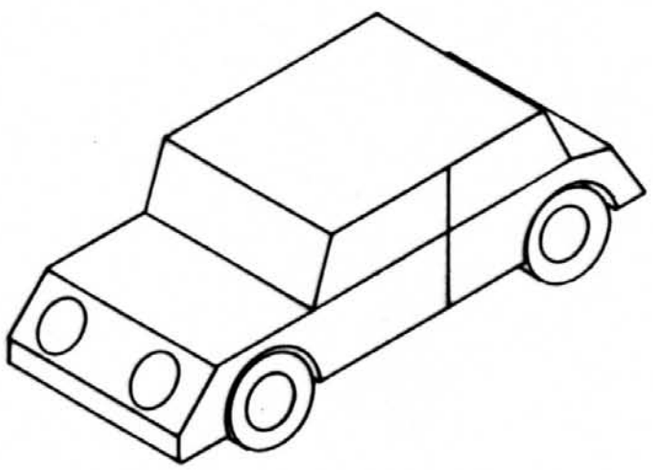


fig 2.9 - Remoção de arestas ocultas

A imagem fica bem mais inteligível e mais de acordo com a nossa percepção visual. Entretanto a remoção das arestas ocultas envolve um esforço considerável, na determinação de quais arestas são removidas e de como são removidas, inte-

gralmente ou parcialmente. Uma determinada aresta pode estar totalmente oculta, por pertencer a uma face não voltada para o ponto de observação, ou pode ser parcialmente visivel devido ao ocultamento de uma parte desta por uma outra face. A remoção de faces e arestas será tratada mais adiante.

Um problema intrinseco à exibição wire-frame é o recorte das linhas sobre a tela. Uma linha que atinge o exterior da tela pode causar problemas na exibição, dependendo do sistema utilizado. Em sistemas baseados em matriz de pixels, uma ultrapassagem dos limites da tela pode causar uma rotação da tela sobre si mesma, dependendo da forma de geração de endereços de pixels. Este fenômeno é retratado na figura 2.10.

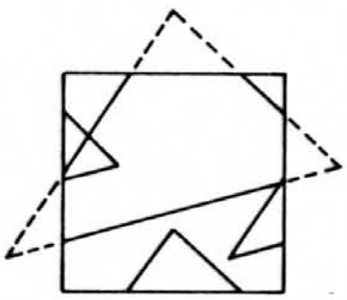


fig 2.10 - Rotação da tela sobre si mesma

Para se evitar este problema pode-se simplesmente testar, para cada pixel gerado, se estes caem dentro da tela. Entretanto, para linhas com grande extensão para fora da tela, o cálculo de seus pixels pode significar uma grande perda de tempo. E necessário, portanto, que as linhas passem por um processo de transformação para a extração de sua porção visivel. Este processo é chamado de "recorte" (line-clipping). A figura 2.11 apresenta algumas possiveis localizações de uma linha a-b sobre a tela. A função do

recorte é identificar os pontos a' e b' para formar a nova reta interna à tela.

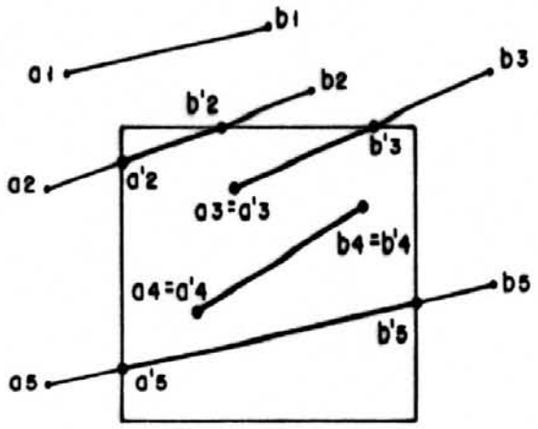


fig 2.11 - Diversas localizações de segmentos para recorte

Os pontos a' e b' podem ser identificados pela intersecção da reta formada por a-b com as quatro retas das bordas da tela. Os quatro pontos de intersecção com as bordas e mais a e b são os seis candidatos a assumirem a' e b'. Os pontos selecionados são aqueles que estiverem dentro dos intervalos formados pelos limites da tela (as intersecções com as retas das bordas da tela podem cair fora da tela) e os limites representados pelos pontos a e b.

Este processo é eficaz mas dispendioso pela grande utilização de operações em ponto flutuante para a determinação das intersecções e as comparações para escolhas dos candidatos. Algumas técnicas foram desenvolvidas para um recorte mais rápido e eficiente. Um exemplo é o algoritmo de Cohen-Sutherland *[NEW 79]* que identifica eficientemente as linhas que podem ser trivialmente aceitas ou rejeitadas, pela checagem das regiões em que estão os pontos extremos da linha. Estas regiões são mostradas na figura 2.12.

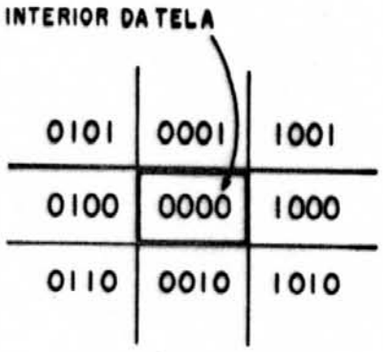


fig 2.12 - Regiões de checagem do algoritmo de Cohen

Para cada região é atribuido um código de 4 bits com o seguinte significado:

bit 0 - 1 se está acima da tela
bit 1 - 1 se está abaixo da tela
bit 2 - 1 se está à esquerda da tela
bit 3 - 1 se está à direita da tela

Um ponto está na tela se seu código é 0000. Se os dois pontos extremos têm código 0000 a linha está dentro da tela e não precisa ser recortada. Uma linha é rejeitada, por não interceptar a tela, se ambos os seus pontos estão acima, abaixo, à esquerda ou à direita da tela. Isto é identificado pelo resultado não nulo da operação lógica E entre os códigos dos pontos extremos da linha. Se o resultado for não nulo esta linha pode ainda interceptar a tela, ou não. Neste caso o recorte convencional deve ser aplicado.

Uma outra técnica foi desenvolvida para aprimorar esta última, chamada de divisão de ponto médio (mid-point subdivision *[NEW 79]*). As linhas em que o E lógico dos códigos for não nula são subdivididas ao meio, pelo ponto médio $x_m = (x_1 + x_2)/2$ e $y_m = (y_1 + y_2)/2$. Os dois segmentos resultantes são então tratatos pelo algoritmo dos códigos das regiões, individualmente. Enquanto os dois segmentos não

cairem em um dos casos triviais a subdivisão continua com os segmentos.

Estes algoritmos apresentados são facilmente implementáveis em linguagem de baixo nivel, dispensando operações de ponto flutuante. A obtenção do ponto médio só requer duas somas e duas rotações e a determinação do código de região depende de quatro comparações de números inteiros.

2.3.2 - Exibição pelas faces

Uma maneira mais desejável de se obter imagens de sólidos é pela representação de sua própria superficie como um todo, projetada no plano de imagem, em lugar de exibir apenas segmentos de retas. Isto é feito pelo preenchimento do interior dos poligonos da superficie dos objetos com a cor deste.

Para evitar confusões na distinção das faces do sólido, o emprego de um maior realismo na apresentação da imagem começa a ser necessário. Os poligonos não são todos preenchidos com a mesma cor, variando-a suavemente entre um poligono e seu vizinho, dando um aspecto de sombreamento acompanhando as orientações das faces. No caso em que as faces poligonais representam uma aproximação de superficies continuas emprega-se um sombreamento continuo dentro de cada poligono de maneira a não se perceber a transição de intensidade de cor entre um poligono e outro adjacente. O aspecto final é de uma superficie continua, sem a percepção das faces planas que a compõe. Estas técnicas são vistas mais adiante.

2.3.3 - Remoção de partes ocultas

Na geração de imagens pela exibição das faces, cada poligono é tratado individualmente até o seu preenchimento na memória de video. Para a correta exibição da imagem nos deparamos com uma dificuldade: a seleção das partes visiveis da imagem, ou seja a escolha das faces, ou partes dessas, que realmente contribuem com a cena. O restante são as partes dos objetos que são ocultas por outras, e que não são visiveis do ponto de vista escolhido e portanto não devem ser exibidas.

Existem numerosos algoritmos desenvolvidos para remover eficientemente partes escondidas das cenas, para diferentes tipos de aplicação. A escolha de um algoritmo para uma aplicação particular depende de fatores como a complexidade da cena, tipos de objetos, equipamento disponivel e se as cenas são estáticas ou animadas *[HEA 86]*.

Normalmente estes algoritmos são classificados conforme a sua atuação, se diretamente sobre a definição do objeto ou sobre a imagem projetada, respectivamente espaço-objeto e espaço-imagem *[SUT 74]*. Os métodos espaço-objeto comparam objetos e partes destes com outros para determinar quais superficies devem ser tratadas como invisiveis. Nos algoritmos espaço-imagem a visibilidade é decidida ponto por ponto em cada posição de pixel no plano de projeção.

Os algoritmos espaço-objeto são convenientes para a remoção de linhas ocultas para a reprodução de imagens tridimensionais em dispositivos de traçado de linhas, como plotters e displays de varredura de vetores. A remoção de partes ocultas na imagem é usual em sistemas de pixels mapeados em memória.

Alguns algoritmos fazem uso de ordenação para obter maior desempenho. A ordenação é um método espaço-objeto para facilitar as comparações de profundidade pela colocação em ordem, de linhas, superficies e objetos da cena, de acordo com a sua distância ao plano de visualização. O artigo de Ivan Sutherland *[SUT 74]* caracteriza e compara dez algoritmos para a remoção de partes ocultas, sob o ponto de vista da ordenação.

Certos algoritmos empregam técnicas de coerência, para tirar proveito de certas regularidades existentes no espaço-imagem. Por exemplo, pode-se esperar que uma linha individual de varredura contenha intensidades de cor de pixels constantes ao longo de certos intervalos. Similarmente o padrão de linha quase sempre muda muito pouco de uma linha para outra e entre as cenas de animação so há mudanças nas pequenas regiões de movimento dos objetos.

2.3.3.1 - Remoção das faces opostas

As faces não voltadas ao observador são seguramente obscurecidas por aquelas voltadas ao observador. Estas podem ser identificadas pela análise do seu vetor normal, transformado para as coordenadas de projeção. Se sua componente Z for positiva, ou seja, tem a mesma direção da vista, é uma face oposta ao observador e será ignorada na construção da imagem. As faces opostas representam, teoricamente, metade das faces que compõem os sólidos, portanto a remoção destas poupa metade do trabalho de teste das faces ocultas.

2.3.3.2 - Método da profundidade "Deep-buffer"

Um método espaço-imagem, comumente utilizado para a eliminação de faces ocultas, é o deep-buffer, também denominado Z-buffer. Basicamente este algoritmo testa a visibilidade das superfícies ponto-a-ponto. Para cada posição (x, y) no plano de vista, a face com a menor coordenada z será visível neste pixel. Uma área de memória adicional é necessária para armazenar a profundidade z de cada pixel.

Inicialmente todos os pixels recebem a cor de fundo e o Z-buffer a profundidade máxima. Cada face é então processada, uma linha de varredura por vez, calculando a profundidade z da face ao longo da linha. Em cada pixel o z calculado é comparado com o valor previamente armazenado no Deep-bufer nesta posição. Se o z calculado for menor que o valor encontrado no Z-buffer o novo z é aí armazenado e a intensidade de cor calculada é transferida para esta posição (x, y). Ao final do processamento de todas as faces o Z-buffer conterá a profundidade da imagem visível e a memória de vídeo conterá as intensidades de cor das faces visíveis.

O valor da profundidade pode ser obtido colocando-se em evidência o termo z da equação do plano da face:

$$A.x + B.y + C.z + D = 0$$

$$z = \frac{-A.x - B.y - D}{C}$$

Sua obtenção para cada pixel pode ser feita incrementalmente, por ser uma expressão linear em x e y. Ao longo de x, z

varia em passos iguais a -A/C, e ao longo de y, em passos iguais a -B/C.

2.3.3.3 - Método da ordenação por profundidade

Este é um método de remoção de faces ocultas que atua tanto no espaço-objeto como no espaço-imagem. As faces são ordenadas em ordem decrescente de profundidade e em seguida são preenchidas na memória de video, a partir da de maior profundidade. A ordenação ocorre no espaço-objeto enquanto que o preenchimento se dá no espaco-imagem.

Este método de remoção de faces ocultas é algumas vezes referido por "algoritmo do pintor". Na criação de uma tela a óleo, o pintor inicialmente pinta as cores de fundo, só depois então os objetos são pintados, desde os mais distantes até o mais próximo, os quais vão sobrepondo os anteriores.

O algoritmo inicialmente ordena as faces segundo a profundidade máxima de cada face (Z_{max}), escolhida entre os vértices das faces. A face de maior profundidade é então comparada com as outras, para a verificação de sobreposição em profundidade.

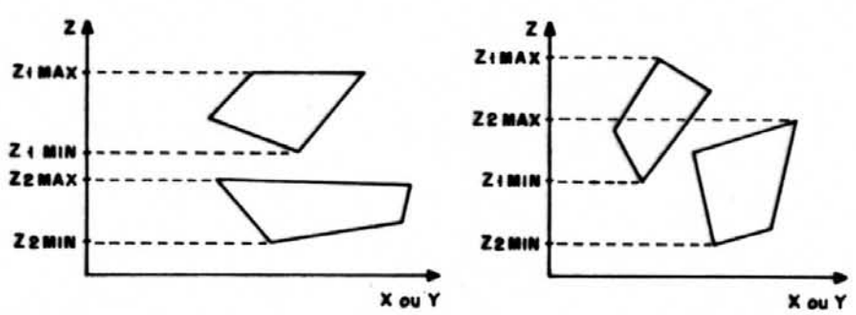


Fig 2.13 - Teste de sobreposição em profundidade z

Esta sobreposição é caracterizada pela intersecção não nula dos intervalos Z_{min}-Z_{max} das faces sendo comparadas, conforme a figura 2.13. Em a) não ocorre sobreposição e em b) ocorre.

Se não há sobreposição então a face mais profunda pode ser exibida na memória de video. Se há sobreposição isto indica que há uma possibilidade da face de maior profundidade ocultar a face sendo comparada, o que implicaria na necessidade de trocá-las de ordem de exibição para que a face oculta seja exibida primeiro.

Se uma sobreposição em profundidade é encontrada, decide-se se estas faces devem ser trocadas de ordem pela seqüência de testes apresentada a seguir. Se algum destes testes é verdadeiro indicará que a ordem de exibição destas duas faces é irrelevante, dispensando a troca de ordem. Os testes são executados em ordem de maior dificuldade:

1 - Os intervalos X_{min}-X_{max} dos polígonos não se interceptam (figura 2.14 a)
2 - Os intervalos Y_{min}-Y_{max} dos polígonos não se interceptam (figura 2.14 a)
3 - A face de maior Z está do lado de fora da superfície comparada, relativa ao plano de vista (figura 2.14 b)
4 - A face sendo comparada está do lado de dentro da superfície de maior Z, relativa ao plano de vista (figura 2.14 c)
5 - A projeção das duas superfícies no plano de vista não se interceptam

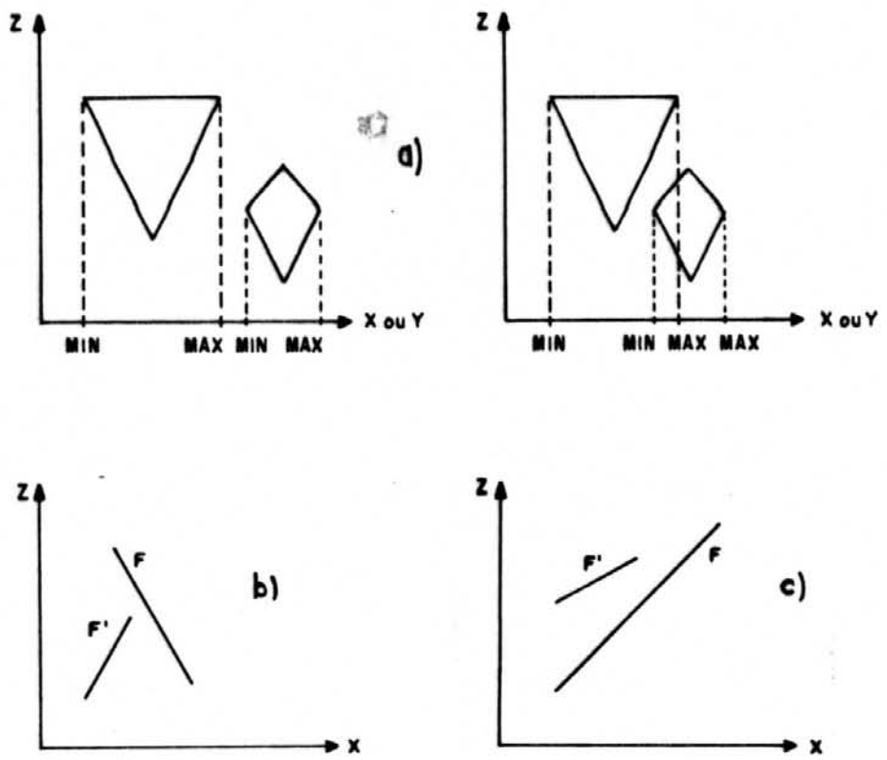


Fig 2.14 - Sequência de testes de sobreposição

O teste 3 é feito substituindo-se as coordenadas dos vértices da face de maior profundidade Z_{max} pela equação do plano da face sendo comparado. Se o resultado for positivo para cada vértice a condição é satisfeita.

Similarmente, o teste 4 é feito pela substituição dos vértices da face sendo comparada na equação da face de maior profundidade. A condição é satisfeita se todos os resultados forem negativos.

Se todos os testes até o 4 falharem então é ativado o teste 5. Este é feito na imagem projetada, onde se testa a presença de vértices de uma face no interior da outra. Caso houver, decide-se a ordem de exibição pela comparação da profundidade nestes vértices.

Esta ordenação final muitas vezes é um processo insolúvel. Nos casos de faces interpenetrantes, ou no caso dos três polígonos em anel como na figura 2.15, é impossível obter uma ordem correta de obscurecimento entre as faces. Nestes casos deve-se dividir as faces em partes e tratá-las independentemente.

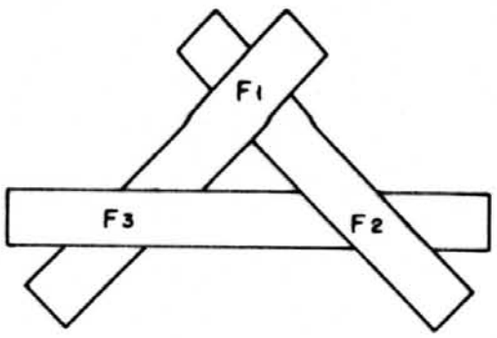


Fig 2.15 - Caso insolúvel de ordenação de faces

2.3.3.4 - Método da coerência de linha de varredura

Este método espaço-imagem de remoção de superfícies ocultas trabalha sobre cada linha de varredura horizontal, uma após outra. A medida que cada linha é processada, todas as superfícies dos polígonos que interceptam esta linha são examinados para determinar quais são visíveis, pela comparação de profundidade.

Para facilitar a procura de faces interceptando uma dada linha de varredura horizontal, utiliza-se uma lista de arestas ativas. Esta lista contém somente as arestas que interceptam a linha de varredura corrente ordenadas em ordem crescente de x. Um flag é alocado para cada face interceptada, indicando que uma posição na linha de varredura está dentro ou fora da face.

Após a preparação da lista para cada linha de varredura, os pixels são calculados um após o outro na direção x. A medida que o ponto atravessa as arestas da lista os flags dos poligonos são trocados indicando qual face é para ser exibida.

A eficiência do algoritmo se deve à coerência existente entre uma linha de varredura e outra, que minimiza as alterações da estrutura de dados durante a exibição. Poucas vezes as alterações são muito drásticas e na maior parte das vezes a lista é inalterada, alterando somente as posições de início e fim das intersecções entre as faces e a linha de varredura. Na própria linha de varredura, por sua vez, existe também um certo grau de coerência. Cálculos de profundidade só necessitam ser feitos nos pontos correspondentes às arestas das faces e entre estes a cor dos pixels é sempre a mesma, não necessitando cálculos de pixel a pixel.

2.3.4 - Técnicas para obtenção de realismo

Imagens realísticas de objetos sólidos podem ser obtidas pela geração de projeções em perspectiva com remoção de partes ocultas e então aplicação de um sombreamento. Um modelo de sombreamento é empregado para calcular as intensidades de luz que nós veríamos se víssemos o objeto no mundo real. Estes cálculos de intensidade luminosa são baseados nas propriedades ópticas das superfícies, suas posições relativas e sua orientação em relação à fonte de luz.

2.3.4.1 - Modelo de iluminação

Fontes de luz:

Quando olhamos para um objeto vemos a intensidade de luz refletida da superficie deste. Esta luz é resultado de diversas fontes de luz que circundam o objeto. Se o objeto é transparente a luz proveniente de trás do objeto também contribui.

As fontes de luz podem ser diretas, como a luz de uma lâmpada ou do sol, ou indiretas, provenientes de outros objetos iluminados. As reflexões múltiplas de luz dos objetos se combinam para produzir uma iluminação que tende a ser uniforme em todas as direções, chamada luz ambiente.

As fontes de luz podem ser classificadas como pontuais, quando sua dimensão é desprezivel comparada com os objetos, ou distribuidas, quando produz luz que incide, sobre cada ponto do objeto, sob um certo intervalo de ângulos diferentes. Uma fonte de luz pontual é muito mais fácil de se modelar do que uma fonte de luz distribuida. Esta, porém, pode ser aproximada por um conjunto grande de fontes pontuais justapostas.

Um modelo de sombreamento para o cálculo da intensidade de luz refletida por uma superficie pode ser estabelecido pela consideração da luz ambiente e das fontes de luz pontuais. Ambas as fontes produzem sobre a superficie uma luz refletida espalhada em todas as direções, devido à reflexão difusa das superficies, conseqüência das rugosidades naturais. Além da reflexão difusa, as fontes pontuais podem causar brilhos nas superficies, ou pontos luminosos, devido à reflexão especular. Isto é mais pronunciado em superficies lisas ou espelhadas.

Reflexão difusa:

A reflexão difusa é a luz emitida pelos objetos, uniformemente em todas as direções. Pode ser causada tanto por fontes de luz difusa como pontuais. Para efeitos práticos podemos considerar a iluminação difusa Id constante em todas as direções. A reflexão é também constante em todas as direções, para fora da face, e pode ser expressa por:

$$I = k_d \cdot I_d$$

onde k_d é o coeficiente de reflexão difusa, que pode assumir valores entre 0 e 1. Superficies brancas possuem k_d próximo de 1 enquanto que superficies negras possuem k_d próximo de 0.

O comportamento da reflexão difusa por uma fonte pontual é baseado na lei de Lambert, que estabelece que a luz difusa refletida depende do ângulo de incidência sobre a superficie, na proporção do cosseno deste ângulo. Isto pode ser explicado pelo fato de que quanto mais inclinada a superficie em relação à fonte de luz menos raios de luz esta superficie capta, pois sua projeção na direção da luz representa uma área menor.

Considerando **N** o vetor normal à superficie, **L** o vetor unitário que expressa a direção da luz e θ o ângulo de incidência formado por estes dois vetores, podemos dizer que $\cos\theta$ pode ser obtido pelo produto escalar entre estes vetores. Podemos considerar também, para efeitos práticos, que a luz decai na proporção inversa da distância da fonte. Assim o efeito da reflexão difusa pode ser descrito por:

$$I = \frac{k_d \cdot I_p}{d} (N.L)$$

I_p é a intensidade de luz emitida pela fonte pontual e d é a distância da fonte ao objeto.

Para expressar as intensidades de luz em termos de cores pode-se aplicar o modelo independentemente sobre cada uma das três cores básicas utilizadas em tubos de imagem colorida, R, G e B (Red, Green e Blue). I_p, I_d e k_d terão cada um três termos, um para cada componente de cor. Assim I_p e I_d passam a ser as cores das fontes de luz e k_d a cor do objeto.

Reflexão especular:

Para certos ângulos de vista uma superfície reflete a luz incidente, independentemente da cor do objeto, criando na sua superfície uma região estreita com a cor da luz da fonte. Este fenômeno é chamado reflexão especular, semelhante ao reflexo de um espelho. No espelho, que é um refletor ideal, o ângulo de reflexão é o mesmo do de incidência. Em objetos mais foscos o ângulo de reflexão é distribuído sobre um pequeno intervalo angular ao redor do ângulo de reflexão ideal do espelho. Este espalhamento depende do grau de rugosidade das superfícies.

Um método para modelar este efeito, desenvolvido por Phong Bui Thong (chamado modelo de Phong) *[HEA 86]*, estabelece que a intensidade de luz refletida é proporcional a $\cos^n \phi$, onde ϕ é o ângulo entre a direção de reflexão ideal e a reflexão próxima a esta direção. Uma superfície espelhada é modelada com n grande (200 ou mais) e uma

superficie fosca possui um n pequeno (até no minimo 1). Quanto maior o n menor é o espalhamento da reflexão especular.

A reflexão especular também depende do ângulo de incidência θ, que em geral aumenta com o ângulo. Como é um fator de comportamento irregular será expresso por uma função $W(\theta)$. Cosiderando **R** o vetor unitário da direção de reflexão ideal, e **V** expressando a direção de reflexão próxima a **R**, conforme a figura 2.16, $\cos\phi$ pode ser dado pelo produto escalar **R.V** . Portanto, o modelo completo de reflexão pode ser resumido por:

$$I = k_d . I_d + \frac{I_p}{d} . [k_d . (N . L) + W(\theta) . (V . R)^n]$$

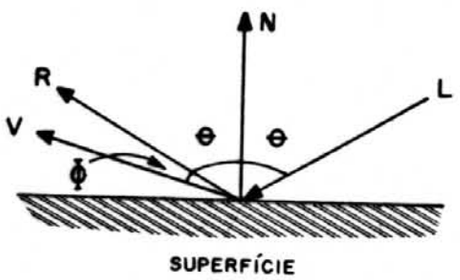


Fig 2.16 - Vetores de incidência e reflexão

Transparência:

Quando objetos transparentes são modelados, a equação de intensidades deve ser modificada para incluir as contribuições das fontes de luz de trás dos objetos.

A transparência também ocorre ocorre de maneira difusa ou especular, podendo utilizar-se dos modelos

definidos para a reflexão. Uma transparência especular é semelhante à do vidro e uma transparência difusa é como ocorre com uma folha de papel vegetal.

A transparência especular idealmente deve considerar efeitos do coeficiente de refração dos objetos. Esta consideração é normalmende bastante dispendiosa e só tem sido aplicada em sistemas ray-tracing.

texturas:

Muitos objetos não têm um sombreamento continuo, como o idealizado até agora. Superficies como muros de cimento salpicado, gramados, casca de frutas e madeira possuem irregularidades de cor ou rugosidades que devem ser consideradas na modelagem para uma concepção mais real destes objetos.

Algumas técnicas têm sido empregadas para isso, que são: variação randômica do vetor normal, variação randômica do coeficiente da cor do objeto e mapeamento de texturas. A variação randômica segue uma distribuição estatistica baseada na análise no dominio freqüência do comportamento das superficies dos objetos reais. O mapeamento de texturas se utiliza de um padrão plano que interage sobre a cor dos objetos pela repetição sucessiva deste na reprodução das suas superficies, acompanhando a sua curvatura.

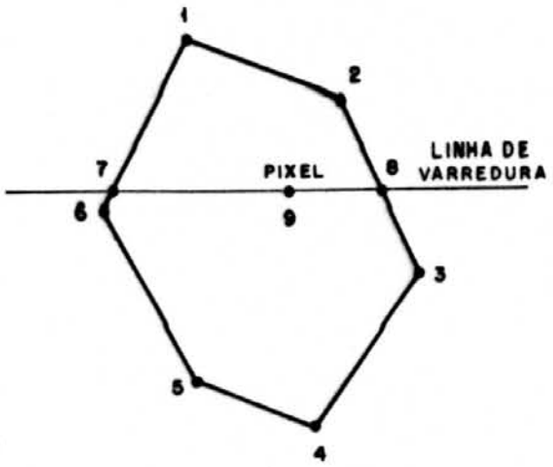
Sombras:

Os métodos espaço-objeto de remoção de faces ocultas podem ser utilizados para localizar as áreas onde as

fontes de luz produzem sombras. Aplicando-se o método como se a fonte de luz fosse o observador, determinam-se as porções não visíveis da imagem por parte da fonte de luz. Estas porções deverão ser iluminadas somente por luz difusa, na projeção convencional da imagem. Mais do que uma fonte de luz pode ser considerada, inclusive para simular fontes distribuídas de luz. Muito dificilmente pode-se determinar as sombras por técnicas espaço-imagem *[BRO 84] [FUC 85]*.

2.3.4.2 - Preenchimento sombreado de faces poligonais

Para o preenchimento sombreado de superfícies definidas por facetas poligonais, Gouraud *[GOU 71]* sugere uma técnica em que os cálculos da modelagem de iluminação são aplicados somente nos vértices dos poliedros para posterior interpolação linear ao longo do restante dos pontos da superfície, a partir das informações contidas nos vértices. Este método é representado na figura 2.17. Cada pixel da linha de varredura recebe um valor interpolado entre aqueles encontrados nos cruzamentos da linha com as arestas, conforme a proporção entre as distâncias do pixel às arestas. Os valores dos pixels nas intersecções da linha de varredura com as arestas, por sua vez, são interpolados a partir das informações contidas nos vértices destas arestas, na proporção das distâncias entre as intersecções e as arestas. Para o cálculo do sombreamento nas arestas é necessário conhecer-se o vetor normal sobre estas. Para isso a técnica de Gouraud utiliza-se da média aritmética entre os vetores normais das faces que contêm o vértice.



$$I_7 = \frac{(Y_7 - Y_6) I_6 + (Y_1 - Y_7) I_1}{Y_1 - Y_6}$$

$$I_8 = \frac{(Y_8 - Y_3) I_3 + (Y_2 - Y_8) I_2}{Y_2 - Y_3}$$

$$I_9 = \frac{(X_9 - X_8) I_8 + (X_6 - X_9) I_7}{X_6 - X_8}$$

Fig 2.17 - Interpolação de Gouraud

Com esta técnica eliminam-se as descontinuidades causadas pelo sombreado de cores constantes em cada face. Entretanto, outras deficiências podem ocorrer. Um reflexo especular estreito pode oscilar entre as faces, com o movimento das figuras, por ser uma descontinuidade de luz de tamanho menor que as faces. A interpolação linear causa também uma ilusão de ótica em que as bordas das faces aparentam ter uma mancha escura ou clara ao longo destas. Esse efeito é chamado bandas de Mach, que ocorrem devido à descontinuidade da segunda derivada da intensidade luminosa, que é perceptivel ao olho humano após um pequeno tempo de acomodação da retina.

Estes problemas podem ser amenizados pela divisão da superficie em poligonos menores ou utilização de outro método, como o método de Phong.

O sombreamento de Phong é uma aprimoração da técnica de Gouraud, em que o vetor normal é interpolado ao longo da superficie, ao invés das informações de cor. O

cálculo das intensidade de cor é feito depois, para cada pixel interpolado. E um método com maior demanda de cálculos mas resolve os problemas associados a técnica de Gouraud.

2.3.4.3 - Sistemas Ray-Tracing

Uma maneira efetiva de determinar as intensidades de luz nas superficies dos objetos é traçar o raio de luz que atinge o observador no sentido contrário a este, desde a posição de visualização, até as fontes de luz, como o ilustrado na figura 2.18.

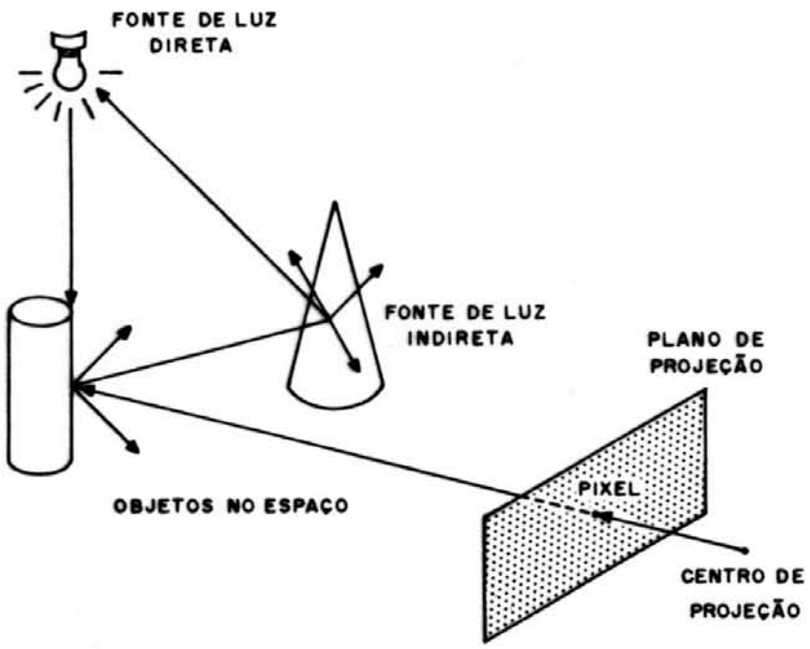


Fig 2.18 - Ray-tracing

Partindo do centro de projeção, o raio passando sobre cada pixel no plano de vista é traçado de volta à superficie na cena tridimensional. O raio é então refletido por esta superficie para determinar se esta vem de outra superficie ou de uma fonte pontual. Este traçado no sentido contrário continua até que todos os raios encontrem as

fontes pontuais ou caiam fora da cena. Cada pixel deve ser tratado desta maneira e o número de raios traçados de volta é o número de pixels da imagem. Após determinar-se toda a árvore de caminhos dos quais o raio teria se originado, calcula-se o efeito da contribuição da cada fonte de luz em cada caminho.

Quando objetos transparentes são encontrados, a contribuição da luz que atravassa o objeto e a sua refração são levados em conta.

As técnicas Ray-tracing requerem uma computação considerável, mas são as que produzem os resultados mais realisticos. Fenômenos reais são automaticamente levados em conta, como transparência, refração, reflexão, reflexões múltiplas, sombras, etc. *[HAL 83], [HEA 86].*

2.3.5 - Antialiasing

Objetos representados por mapeamento de pixels estão sujeitos a uma distorção causada pela digitalização da imagem. As coordenadas dos objetos são arredondadas para posições discretas de pixels. Como conseqüência as bordas dos objetos aparentam ser serrilhadas. Este efeito traz conseqências desagradáveis, especialmente em animação de imagens. Com o movimento de partes da cena nas suas bordas aparece um movimento parasita serrilhado, causado pelo batimento deste com a resolução da tela.

Uma maior resolução pode ser empregada para melhorar a aparência dos objetos, mas isto aumenta o tamanho da memória requerido e não elimina totalmente o problema.

Uma técnica de pós-filtragem (filtragem após a exibição da imagem) pode ser empregada para o antialiasing *(FUJ 83)*. Cada pixel computa sobre seu valor uma média entre seus vizinhos, acarretando um leve desfocamento da imagem. Este processo é eficaz mas tem desvantagens tais como a redução da resolução aparente da imagem e o fato de não poder recuperar informações perdidas na digitalização da imagem e portanto não é um método usual.

Um método de desenvolver uma rotina antialiasing (anti-serrilhado) é baseado na idéia de que os pixels não são entidades geométricas infinitesimais, mas possuem uma dimensão finita igual à distância entre os pixels vizinhos. Ao invés do pixel assumir a cor que recai sobre o seu centro infinitesimal, este assume um valor correspondente a toda a sua área, que pode ser considerada quadrada ou retangular. Assim se a área do pixel é ocupada por mais de uma borda de face ou objeto, este recebe uma intensidade de cor correspondente a média entre as cores presentes dentro do pixel, na proporção que estas ocorrem.

A aperência final da imagem na tela, vista da distância usual, é como se o sistema tivesse uma resolução maior, uma vez que se considera dimensões inferiores à do pixel para a determinação de sua cor. Na figura 2.19 a) vemos um segmento de reta com largura de 1 pixel, traçado do pixel (1,2) ao pixel (8,6); em b) a aproximação deste segundo a resolução da tela; e em c) o mesmo segmento com antialiasing. A intensidade de cor atribuída a cada pixel representa a proporção em que o segmento atinge este.

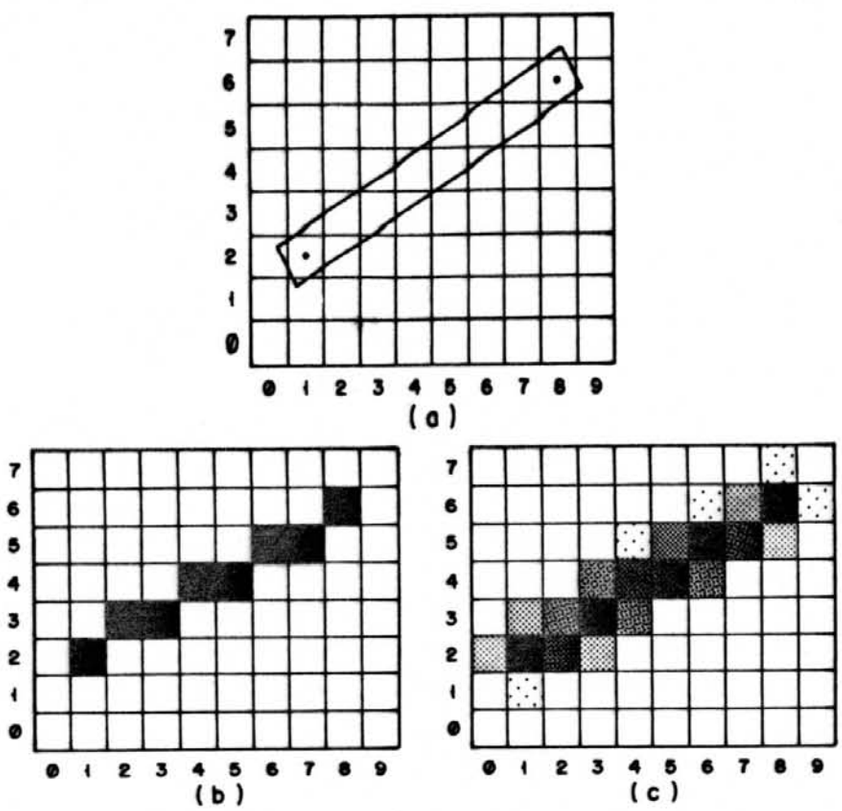


Fig 2.19 - Antialiasing aplicado sobre um segmento de reta

Uma técnica para a determinação da cor dos pixels com antialiasing é a de subividir este por uma matriz de subpixels, 4x4 por exemplo. Executa-se a geração da imagem com uma ampliação de 4x, para este caso, e atribui-se a cada pixel a média de seus 16 subpixels.

Outras técnicas fazem uso de aproximações de informações disponíveis em seus sistemas. O preenchimento de polígonos com antialiasing, por exemplo, pode determinar a cor dos pixels nas bordas dos polígonos pela distância entre o centro do pixel e a borda do polígono *[FUJ 83]*. O pixel recebe uma cor intermediária entre a cor do polígono e a cor de fundo, de acordo com uma dosagem representada por "n",

que situa-se entre 0 e 1. "n" indicará a porcentagem de dosagem entre as cores de fundo e do pixel calculado:

$$pixel = n.pixel + (1-n).fundo$$

A obtenção de "n" pode ser dada por diversas aproximações, que não prejudicam o antialiasing de maneira significativa. Podemos estabelecer "n" como uma função da distância, em unidade de pixels, entre o centro do pixel e o polígono, n=f(d), de maneira que "d" variando entre -0,5 a 0,5 (do interior do polígono ao exterior) faça "n" variar de 1 a 0 linearmente. "d", por sua vez, pode ser aproximado pela menor projeção da distância sobre os eixos X e Y, acarretando um erro imperceptível ao antialiasing *[FUJ 84]*. Estas aproximações da área ocupada pelo polígono no pixel pela distância ao centro deste podem ser úteis quando se dispõe desta informação, uma vez que a determinação da área exata ocupada é normalmente difícil.

Para o cálculo do antialiasing, não há estudos aprofundados sobre a validade de se considerar os pixels como quadrados ou retangulares e nem de se atribuir o mesmo peso a todos os pontos internos ao pixel, na integração de sua área. Poderia, talvez, obter-se uma implementação mais ideal, por exemplo, se considerássemos os pixels circulares e com peso segundo uma função de Gauss centrada no centro do pixel. Isto pode ser implementado na função f(d), apresentada acima. Ao invés de variar linearmente entre 0 e 1, f(d) poderia ser a função erf(d), que é a integral da função de Gauss.

3. ARQUITETURAS PARALELAS PARA A GERAÇÃO GRAFICA 3-D

Este capitulo destina-se a abordar a exploração de paralelismo e o uso de arquiteturas especiais na geração de imagens tridimensionais com realismo, com a exemplificação através de artigos apresentando propostas de projetos em arquiteturas paralelas para tal. No final do capitulo serão discutidos dois projetos de arquiteturas especiais para a obtenção de imagens gráficas de sólidos com realismo, que serviram de incentivo e de impulso criativo para o projeto do GIRTD, meta final deste trabalho.

3.1 - Introdução

A tarefa de geração de imagens com realismo é normalmente implementada em software. A justificativa para a procura de arquiteturas paralelas deve-se à velocidade e eficiência exigidas nesta tarefa, em aplicações a tempo real, as quais não podem ser atingidas simplesmente por software. Enquanto não houver processadores capazes de executar instruções a uma velocidade acima de Giga-instruções por segundo, o que é uma expectativa futura com o desenvolvimento das tecnologias de supercondutores ou Ga-As (Gálio-arsênio), alternativas são exploradas para atingir esses altos desempenhos requeridos, dentro do atual estado

da arte da tecnologia de semicondutores. As técnicas em arquiteturas paralelas são o principal foco de atenção dentro dessa busca de alternativas. Considerando-se a atual possibilidade de integração em larga escala de multiprocessadores, o uso de arquiteturas paralelas não implica em equipamentos gigantescos tornado-se também uma solução atrativa para equipamentos pequenos como estações gráficas. Estas podem hoje incorporar inúmeras facilidades decorrentes do uso de arquiteturas paralelas de alto desempenho localmente.

Na área da obtenção de imagens realisticas, a utilização do paralelismo entre processadores tem sido explorada, para as aplicações a tempo real. Isto requer um processamento intenso de grande quantidade de dados.

3.2 - Exploração de paralelismo em algoritmos

A filosofia básica na implementação de arquiteturas paralelas como meio de aceleração de algoritmos é a exploração de paralelismos implicitos desses algoritmos. Isso pode dar-se de três formas:

1 - Algoritmos que manipulam uma série de dados de mesma natureza que passam cada um por uma seqüência de processamentos diferentes. As arquiteturas que possuem paralelismo compativel com esse tipo de algoritmo são as organizações de processadores sob forma de pipeline para tarefas diferentes, ou sistólicos para tarefas iguais. As diversas tarefas que manipulam esses dados são implementadas por unidades de processamento individuais, interligadas sob a forma de cadeia, de maneira que no término da execução de uma tarefa em uma unidade da cadeia, esta entrega seu dado

resultante à unidade seguinte na cadeia. A partir desses instante esta unidade de processamento está apta para receber um novo dado da unidade anterior, permanecendo quase sempre ocupada processando algum dado. Assim todas as unidades da cadeia trabalham simultaneamente com dados e tarefas diferentes, como a produção em série em uma fábrica em que cada operário se responsabiliza pela inclusão de uma peça de um produto. Podemos chamar esse paralelismo de horizontal, imaginando o fluxo de dados indo da esquerda para direita, onde cada dado tem que passar por todos os processos.

2 - Algoritmos caracterizados pela presença de uma tarefa que deve tratar uma grande quantidade de dados temporalmente independentes entre si. Essa tarefa pode ser repetida em diversos processadores iguais para permitir que mais dados possam ser trabalhados ao mesmo tempo. Uma estrutura de processadores desse tipo é chamada de "array-processor", ou processadores matriciais. Imaginando novamente o fluxo de dados da esquerda para a direita, este paralelismo pode ser chamado de paralelismo vertical pois os dados se distribuem entre os processadores para tratamento concorrente.

3 - Uma terceira forma pode ser incluida na classificação para distinguir aquelas implementações que não são decorrentes da exploração de paralelismo em um dado algoritmo conhecido mas da modelagem de novos algoritmos já na forma de processos concorrentes, para resolver uma dada tarefa, que tiram grande proveito do uso de processamento paralelo. Podem resultar em configurações como em 1 ou 2, ou em outras formas menos comuns como em processadores associativos, "data-flow", etc.

Essa classificação não distingue tipos de arquiteturas e sim a maneira pela qual exploramos o paralelismo em algoritmos. Existem algoritmos que podem ser implementados tanto na forma de pipeline como em "array-

processors"; por exemplo aqueles que tratam uma cadeia de tarefas diferentes e que também manipulam uma grande quantidade de dados. Podemos ter soluções hibridas.

Pode-se afirmar que o ganho com o uso do paralelismo está relacionado com o número de unidades processadoras concorrentes usadas. Embora possa parecer que o tipo de arquitetura é irrelevante, apenas importando o número de processadores utilizados, cada arquitetura possui caracteristicas que podem ou não favorecer determinadas aplicações, além das dificuldades de implementação intrinsecas a cada uma.

Uma vantagem da arquitetura pipeline pode ser avaliada pela analogia com a fábrica. Se as mesmas pessoas da linha de produção fossem instruidas para que cada uma produzisse uma unidade completa do produto que antes era feito em série, a quantidade de mão de obra seria a mesma só que cada pessoa teria que ser especializada em todas as tarefas das quais o produto depende, o que é caro. Em um processamento em pipeline cada unidade pode ser mais simples e dedicada a sua tarefa. Assim o aumento de custo computacional, com o uso de pipeline, é menor que o aumento de performance obtido.

Em arquiteturas pipeline, por exemplo, os diferentes processadores encarregados das diferentes tarefas devem possuir entre si um equilibrio de forma a executar essas tarefas em tempos semelhantes para não haver um congestionamento em um determinado estágio do pipeline. Dificilmente pode-se dividir um algoritmo em tarefas com mesmo grau de complexidade, e além disso há tarefas cujo tempo de execução depende muito das caracteristicas dos dados sendo processados. Em determinados casos pode-se utilizar paralelismo local numa determinada unidade de

processamento dentro da cadeia do pipeline, para que essa tarefa possa equilibrar-se com as outras.

Em "array-processors", por sua vez, encontramos algumas dificuldades tais como a distribuição dos dados entre esses, que pode tomar um tempo extra, e o problema das interligações entre processadores. Se simplesmente for feita uma distribuição de dados para execução em vários pontos simultaneamente, com o mesmo algoritmo original distribuido entre estes pontos, pode não haver necessidade de interligações embora haja aplicações em que a troca de informações entre processadores é fundamental.

3.3 - Paralelismo na geração de imagens gráficas

Inúmeras operações potencialmente paralelizáveis podem ser encontradas nas tarefas de geração gráfica de sólidos por computador. O cálculo de pixels, por exemplo, é um procedimento repetitivo, que deve ser feito de maneira igual para cada pixel. Esta tarefa pode ser executada por vários processadores iguais, operando em paralelo, cada um dedicado a uma partição da memória de vídeo. Por outro lado, as tarefas comuns na exibição de imagens tridimensionais, tais como transformação de vista, projeção, recorte, etc, podem ser executadas em paralelo segundo uma estrutura pipeline de processadores. Estas tarefas, por sua vez, também podem ser executadas por processamento paralelo local, para prover um balanceamento no pipeline. Podemos ter soluções mistas, portanto.

Veremos na próxima sessão, através de exemplos fornecidos por artigos, algumas propostas de arquiteturas

especiais para a geração gráfica tridimensional. Na seção seguinte serão comentados dois artigos, considerados como casos de estudo.

3.4 - Exemplos de arquiteturas especiais

A estação gráfica comercial PMX900, descrita em [TUN 87], é um exemplo de arquitetura mista. Nela, um pipeline de 9 estágios, de 2 processadores cada, executa as tarefas convencionais de processamento gráfico, como rotação, recorte, fator de escala, etc. No final do pipeline encontra-se um array de 64 nodos processadores, arranjados em uma matriz 8x8 (figura 3.1). Os processadores são do mesmo tipo que os encontrados no pipeline, o Digital Signal Processor da AT&T - DSP32, cada um com poder de 10Mflops. Cada nodo contém uma porção da memória de vídeo e de Z-buffer para a remoção de pixels ocultos. Toda a memória de vídeo - 1024x1024 - é dividida em 64 porções de 128x128 cada, atribuídas a cada um dos nodos. Com 10Mflops em cada nodo tem-se 640Mflops de desempenho para o cálculo dos pixels.

É um equipamento relativamente volumoso para uma estação gráfica, mas versátil, pois cada um dos seus 82 processadores possui RAM para carga de seus dados e rotinas, podendo o sistema ser configurado para aplicações desde processamento de imagens até a geração gráfica por ray-tracing.

O desempenho obtido é bastante grande, considerando-se que é uma estação gráfica de propósitos gerais. Executando-se algoritmos de sombreamento de Gouraud pode-se calcular até 16Mpixels/s, ou 250Kpixel/s por nodo.

Além disso executa um algoritmo de ray-tracing por volta de 300 vezes mais rápido do que estações de engenharia equipadas com acelerador de ponto flutuante.

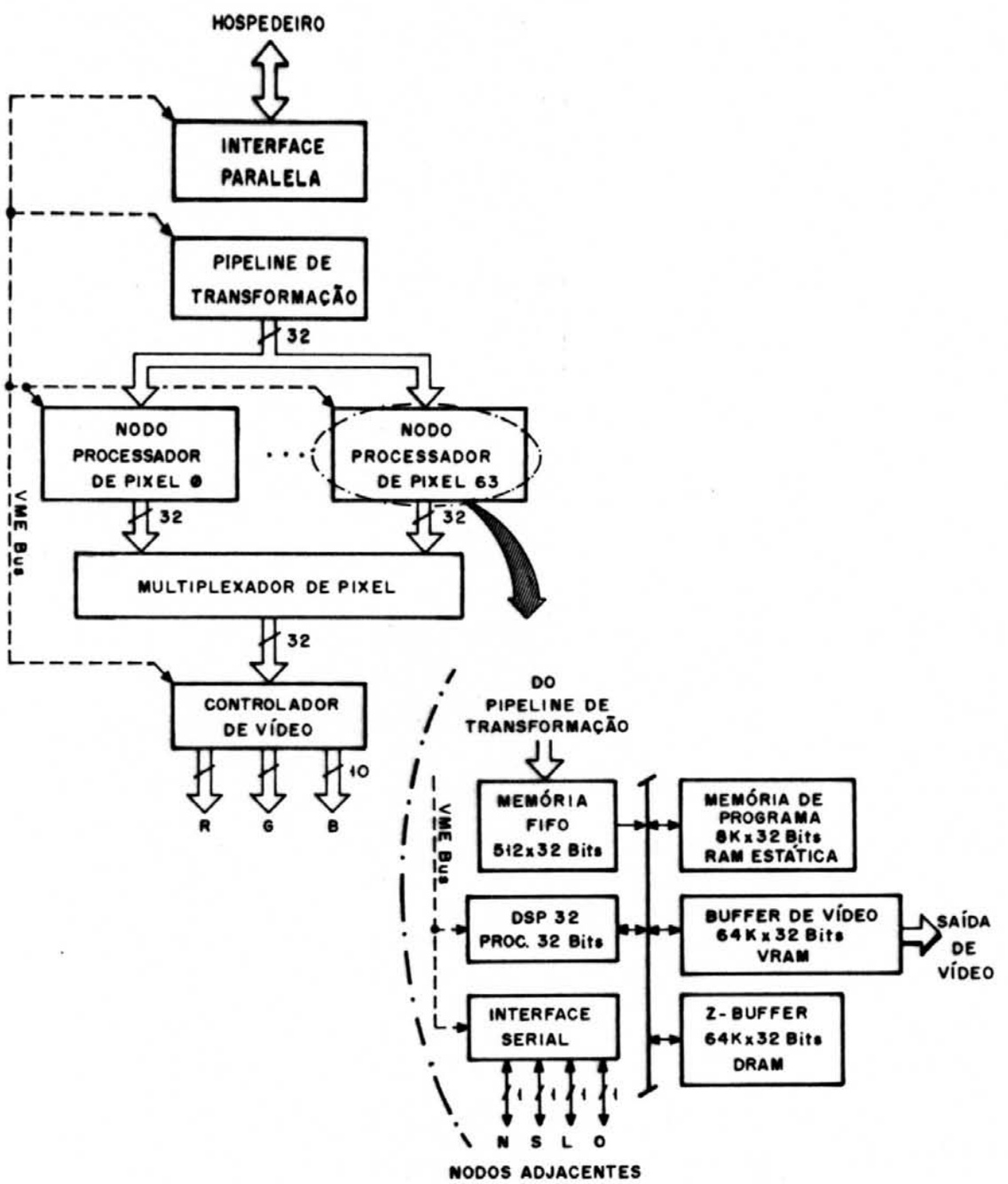


Fig. 3.1 - Sistema PMX900

Um outro exemplo de array-processor para a geração de imagens é proposto em [SAT 85]. Apesar de ser voltado para um algoritmo especifico, geração de imagens por CSG e Z-buffer, este se utiliza de processadores de uso geral em seus nodos, especificamente o processador 80186 acompanhado de seu coprocessador de ponto flutuante, 8087.

A arquitetura também é arranjada em uma matriz 8x8, totalizando 64 processadores, e a imagem é dividida em sub-imagens para tratamento individual em cada nodo, como pode ser visto na figura 3.2.

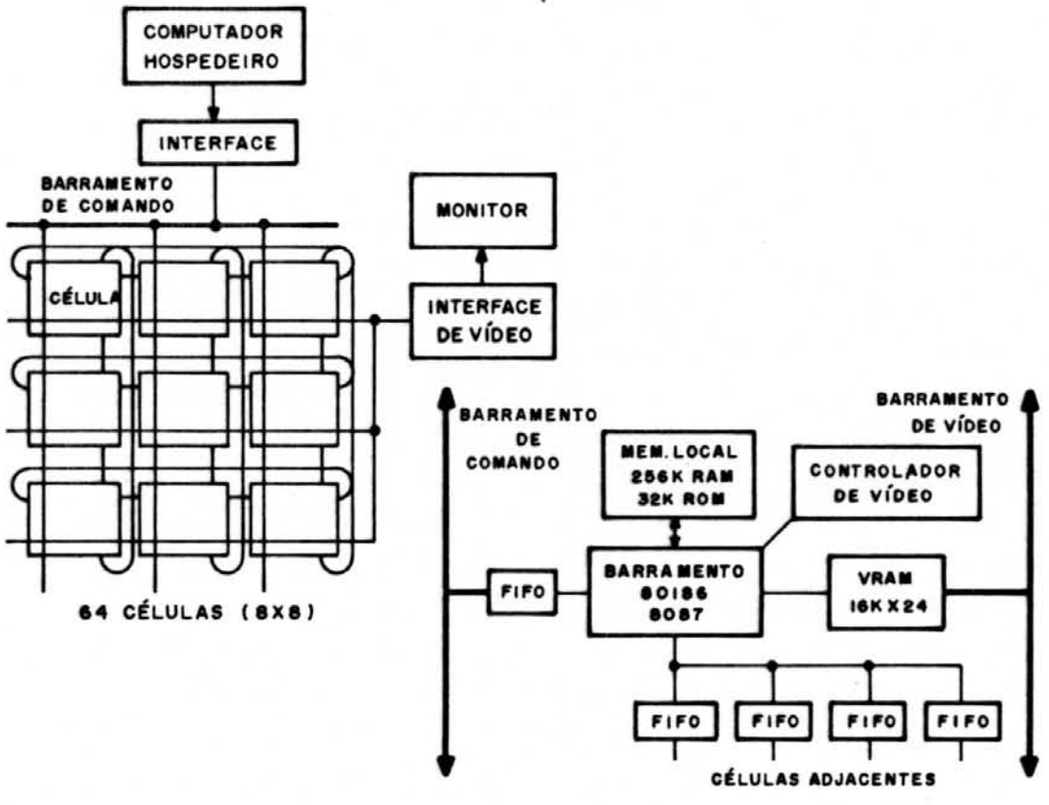


Fig. 3.2 - Configuração do "array" e suas interconexões

Neste projeto emprega-se a comunicação inter-processadores para permitir acesso a dados de outros processadores. A comunicação inter-processadores reduz os requerimentos de memória local, pois cada processador pode

acessar dados de outros, embora seja um procedimento de menor eficiência.

O desempenho é inferior ao projeto anterior pois cada nodo possui em torno de 200 vezes menos capacidade de ponto flutuante. Por outro lado possui um desempenho teórico pelo menos 64 vezes maior do que o de uma estação gráfica que faz uso de apenas um par 80186/8087, que é uma configuração bastante comum comercialmente.

Em [NII 84] encontra-se uma interessante arquitetura de processamento paralelo misto para geração de imagens tridimensionais pela exploração de paralelismo em algoritmo de remoção de partes ocultas "Scan-line coherence".

Trata-se de uma disposição hierárquica de processamento paralelo em dois níveis. A tarefa de geração e manutenção das tabelas de segmentos ativos (descrita na seção 2.3.3.4), utilizadas neste algoritmo, fica ao encargo de oito processadores SLPs (Scan-Line Processors), dividindo entre si áreas iguais da memória de imagem. Estes por sua vez gerenciam, cada um, oito processadores escravos, os PXPs (Pixel Processors), dedicados ao preenchimento da memória de vídeo no território que lhes foi atribuído. Assim, 64 processadores PXP, dispostos como um array-processor, são controlados por 8 processadores SLP, trabalhando em paralelo com os PXPs como um pipeline. Seu circuito é mostrado na figura 3.3.

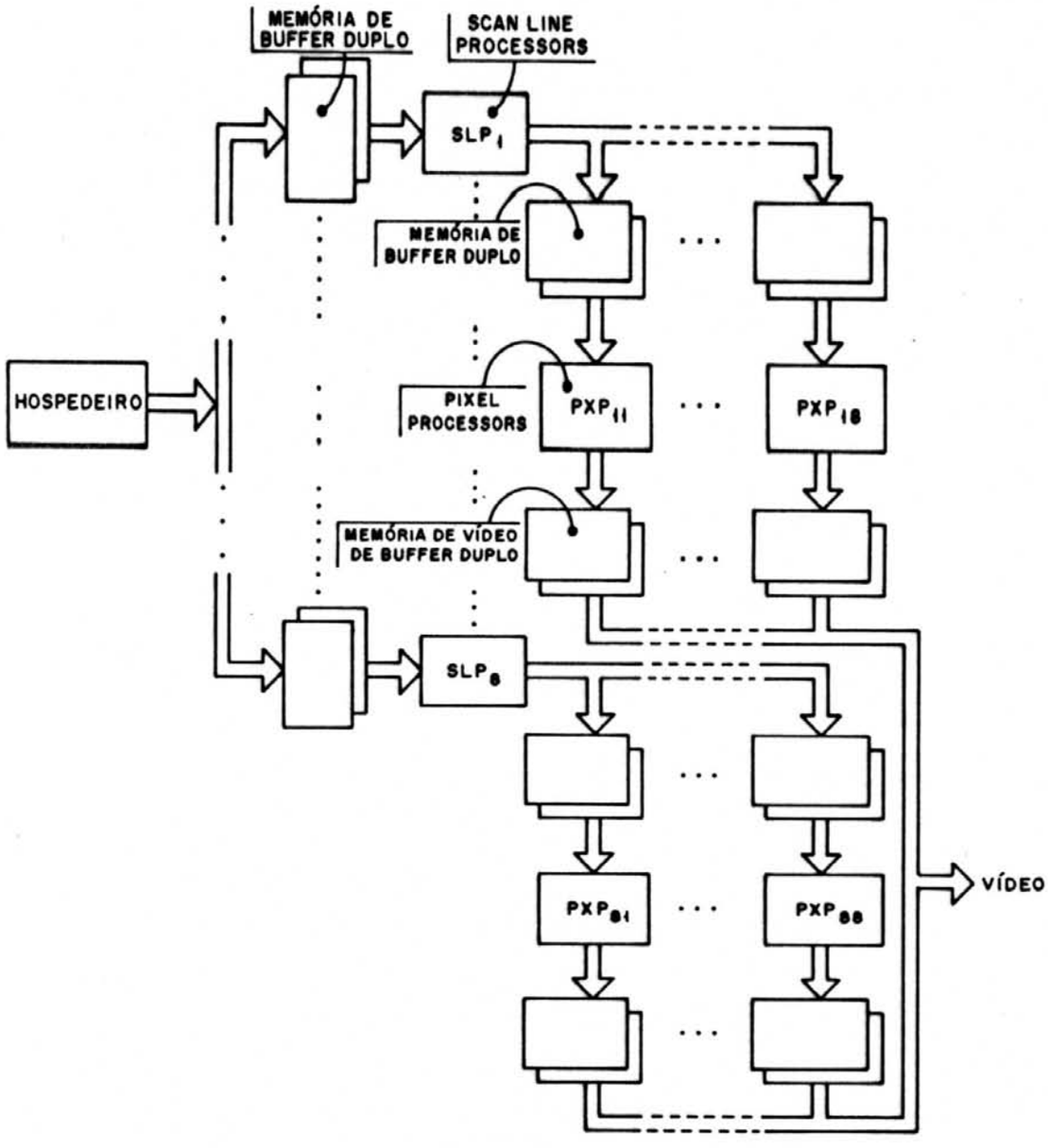


Fig. 3.3 - Estrutura do multiprocessador hierárquico

Uma arquitetura pipeline voltada para o controle de imagens 3-D pode ser vista em [BUR 87]. Esta possui distintas tarefas para geração das imagens, distribuídas entre os estágios do pipeline (a transformação de vista, a modelagem de iluminação, a projeção e o preenchimento de

memória de vídeo). Seus estágios são separados por memórias de porta dupla, o que é um procedimento comum para prover um fluxo balanceado de dados entre os estágios do pipeline. A atualização da memória de vídeo faz uso de uma memória cache intercalada antes da memória de vídeo. Enquanto a memória de vídeo possui um ciclo de acesso de 360ns, a memória cache, contendo apenas 16 pixels, pode ser acessada a cada 60ns. A memória de vídeo possui uma extensão para o armazenamento do Z-buffer, para remoção de pixels ocultos. Esta estrutura pipeline é mostrada na figura 3.4.

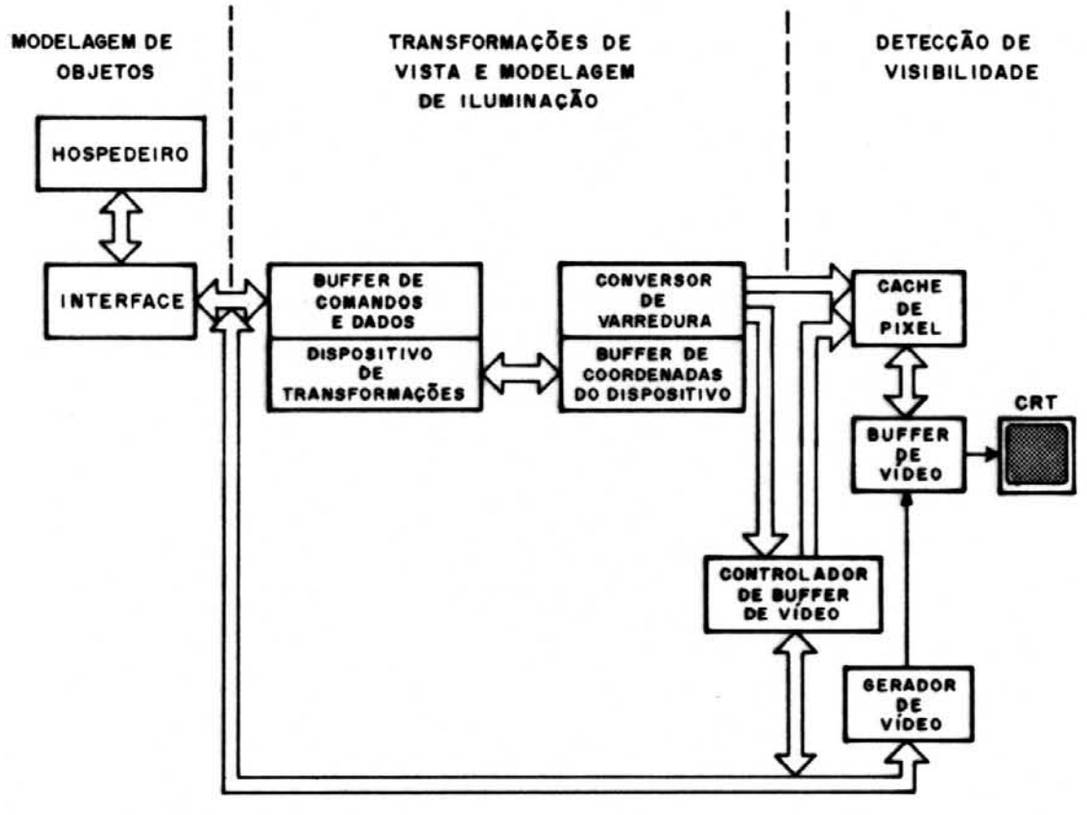


Fig. 3.4 - Disposição dos estágios do pipeline gráfico

A memória de vídeo é normalmente o principal gargalo para o processo de exibição gráfica em tempo real. A quantidade de pixels é grande e o tempo de acesso de cada

pixel é tal que para o acesso de todos os pixels gasta-se em torno de 0,4s. Em animação deseja-se pelo menos 30 cenas por segundo, ou 33ms por cena, o que requer um acesso doze vezes mais rápido. Na falta de memórias com tempos de acesso mais rápidos utiliza-se o particionamento da memória de vídeo, conforme já exemplificado pelos artigos mencionados.

O artigo [SPR 83] descreve uma construção de memória de vídeo particionada de maneira que num mesmo ciclo de memória são acessados 64 pixels, organizados em uma matriz quadrada de 8x8 pixels, localizada em qualquer posição da imagem. Desta maneira se o ciclo de memória é de 500ns, em média cada pixel será acessado em 8ns. O sistema tem uma grande eficiência em geração de textos e transferências de blocos de memória de vídeo. Com o paralelismo no cálculo dos 64 pixels tem-se um bom desempenho para animação e/ou realismo.

Uma outra abordagem do problema do acesso à memória de vídeo é vista no artigo [DEM 85]. Neste foi questionado o gargalo do limite de largura da banda do meio físico entre o processador gráfico e a memória de vídeo. Como solução o autor propõe simplesmente a união destas duas entidades, o processador gráfico e a memória de vídeo, em um único chip entitulado SLAM, ou "Scan-Line Access Memory". Baseando-se na idéia das VRAMs (Vídeo-RAMs), onde internamente uma linha inteira de pixels é acessada num deslocamento série para geração do sinal de vídeo, esta mesma linha pode ser alterada por inteiro em um único ciclo por uma ULA simples (figura 3.5), de tantos bits quanto a largura da linha (usualmente 256). A SLAM recebe comandos do tipo "preencher os pixels desde a posição P1 até a posição P2 segundo o padrão P", simplesmente recebendo esses parâmetros, P1, P2 e P, e o comando para que até 256 pixels sejam preenchidos em um único ciclo. E um procedimento

bastante desejável para o preenchimento de polígonos na geração de imagens para animação podendo-se facilmente atingir uma taxa de 60 imagens complexas por segundo. Como o preenchimento de uma linha se dá de maneira igual para todos os pixels da linha, de acordo com o padrão especificado, as SLAMs não se prestam para a geração de imagens com realismo.

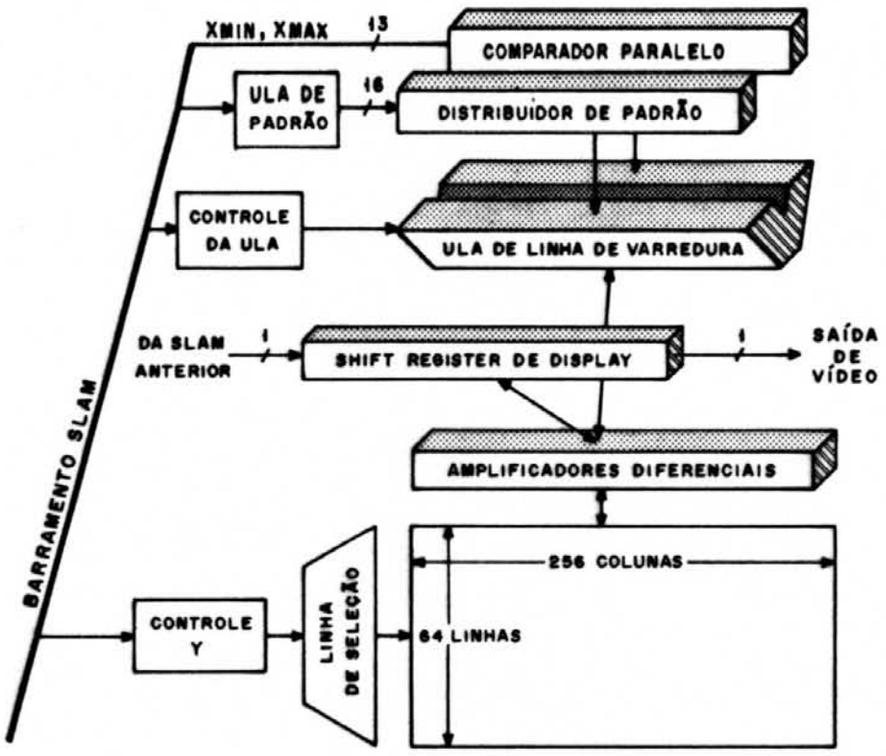


Fig. 3.5 - Estrutura interna do chip SLAM

3.5 - Estudo de casos

Nesta seção, com a finalidade de ilustrar a exploração do paralelismo em processamento gráfico, serão comentados dois artigos de projetos de equipamentos gráficos voltados ao realismo e tempo real, baseados em arquiteturas especiais de processadores. O primeiro é uma construção em forma pipeline e o segundo um array-processor especial. Estes dois artigos são de especial importância pois serviram de inspiração para o projeto do GIRTD, no próximo capítulo.

3.5.1 - Processador pipeline com Deep-Buffer - [FUJ 84]

Este artigo descreve as técnicas empregadas para a geração de imagens realísticas de alta velocidade, utilizadas em uma estação gráfica especial.

Suas principais características são:

- geração de imagens a partir de objetos poliédricos
- simulação de iluminação real das superfícies
- sombreamento contínuo de Gouraud
- remoção de partes ocultas por Z-buffer
- distribuição da carga de processamento entre processadores arranjados em uma configuração pipeline
- emprego de DSPs nos estágios do pipeline
- preenchimento da imagem por circuito incremental DDA (Differential Digital Analizer) com sombreamento e anti-aliasing

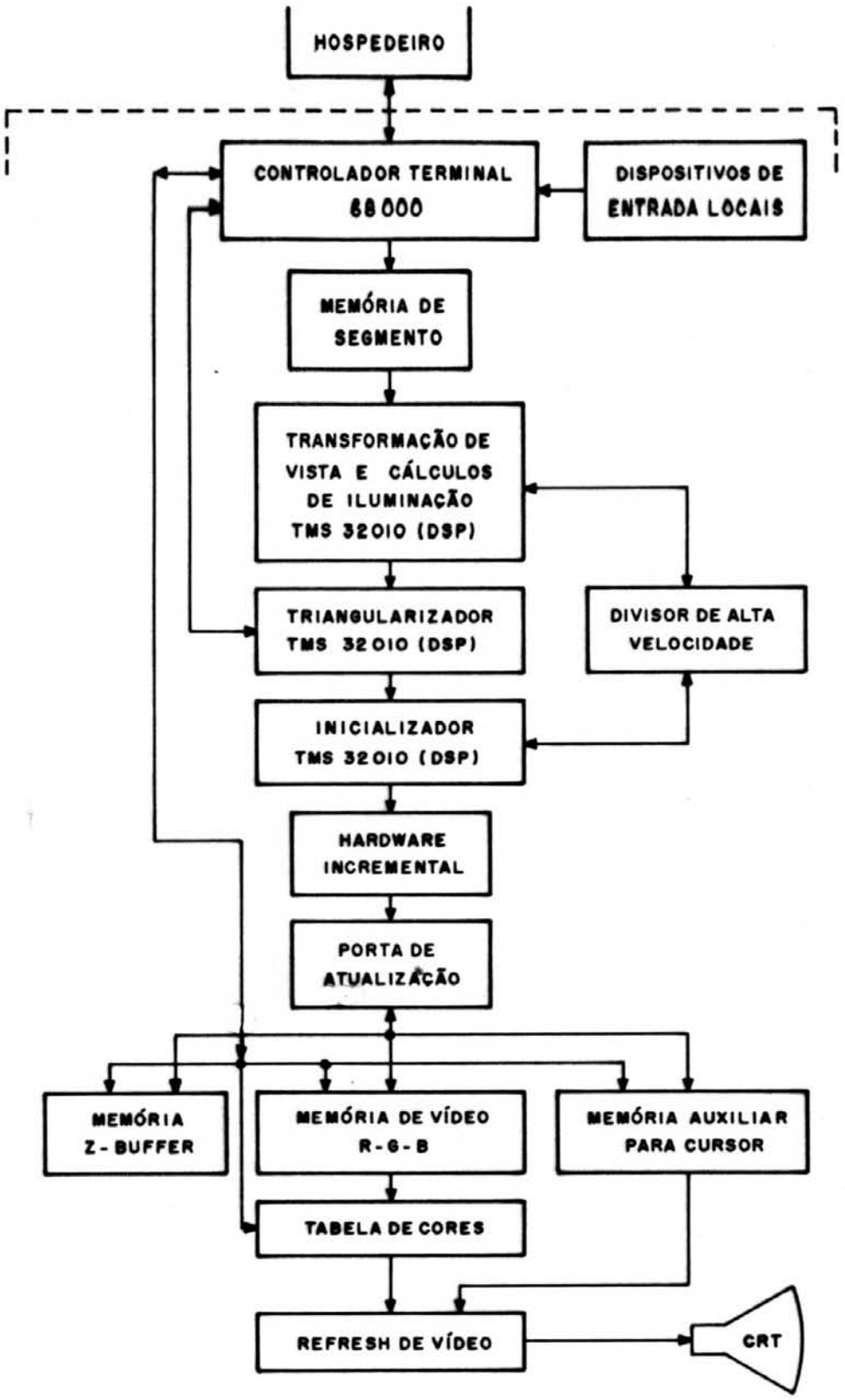


Fig. 3.6 - Organização do pipeline gráfico

A organização dos módulos do terminal é vista na figura 3.6.

A imagem é definida pelo programa aplicativo em execução no computador hospedeiro e transmitida serialmente para o terminal. O processador 68000 que recebe as informações em série do hospedeiro também gerencia a operação dos módulos restantes do equipamento. Este tem acesso à memória de segmento, à memória de vídeo, à memória de profundidade Z e à tabela de cores. A geração da imagem não é feita neste processador e sim pelo pipeline, nos seguintes estágios:

- memória de segmento
- transformação de vista e cálculo de iluminação
- triangularizador
- hardware incremental
- porta de atualização
- hardware de refresh de vídeo

Memória de segmento:

A memória de segmento é uma memória de dupla porta que contém a descrição dos objetos, incluindo polígonos, vetores, retângulos e círculos. Cada vértice é representado por suas coordenadas x, y e z, a direção do vetor normal, a cor e a transparência.

Transformação de vista e cálculo de iluminação:

Esta operação transforma as coordenadas dos vértices e os vetores normais através da aplicação de rotação, translação e fator de escala. Logo após é aplicada a modelagem de iluminação e em seguida é feita a transformação em perspectiva.

Triangularizador:

O problema de lidar-se com poligonos de tamanhos e formas diferentes é eliminado com a transformação destes em triângulos. A transformação faz uso dos mesmos vértices dos poligonos, apenas altera a topologia da descrição para que os estágios seguintes apenas lidem com os triângulos que compõem estes poligonos.

Inicializador:

Este estágio é responsável por preparar parâmetros para o preenchimento de triângulos por parte do hardware incremental. Aqui calculam-se as inclinações das arestas do triângulo no sentido de inicializar a geração das bordas deste no DDA. Aqui são obtidos também os incrementos em X e Y dos valores R, G, B e Z para a determinação incremental dos atributos de cor e profundidade dos pixels do triângulo.

Hardware incremental:

Um hardware em ECL é capaz de determinar incrementalmente as variáveis aplicáveis à continuidade do sombreamento, dentro dos ciclos de escrita da memória de video.

Hardware de refresh de video:

Este hardware lê continuamente os valores R, G e B da memória de video, utilizando uma tabela de cores para converter os valores de intensidade lidos para os niveis requeridos pela caracteristica não linear dos tubos de CRT.

Os estágios do pipeline fazem uso do DSP de 16 bits da TEXAS, TMS32010, com ciclos de 200ns. Este incorpora internamente um multiplicador paralelo de 16x16 bits que é vastamente empregado nos algoritmos. Todos os dados são manipulados em números de ponto fixo. A maioria das operações são somas e multiplicações. Operações de divisão só são necessárias no processo de projeção e no inicializador. Estes processos compartilham um único divisor rápido implementado com uma tabela de inversos 1/x em ROM.

O triangularizador é projetado para ser bastante eficiente no processamento dos poligonos mais comuns. Poligonos com mais de cinco lados são raros e na prática a maioria são convexos. O número de vértices é avaliado, se for 3 passa direto como triângulo, se for maior que 3 deve-se então avaliar o número de vértices côncavos. Para um ou nenhum vértice côncavo a solução é imediata, caso contrário a solução não é mais trivial. Neste caso o poligono é devolvido ao 68000 onde um algoritmo mais geral de triangularização é aplicado. As soluções não triviais são determinadas no 68000 por ser este um processador mais fácil de programar e ter arquitetura mais regular e flexivel do que o TMS32010, segundo justifica o autor.

O inicializador prepara parâmetros para o hardware incremental executar a interpolação da cor e da pofundidade no preenchimento dos triângulos, que são as derivadas, ou incrementos, nas direções X e Y para R, G, B, Z e a transparência T, e declividade das bordas dos triângulos. Os triângulos emitidos pelo triangularizador são descritos pelas coordenadas de seus vértices, x, y e z, e pelas informações de cor e transparência. A partir destes o inicializador calcula as derivadas em relação a x:

$$\frac{dU}{dx} = \frac{(U2-U1)(y3-y1)-(U3-U1)(y2-y1)}{(x2-x1)(y3-y1)-(x3-x1)(y2-y1)}$$

e em relação a y:

$$\frac{dU}{dy} = \frac{-(U2-U1)(x3-x1)+(u3-u1)(x2-x1)}{(x2-x1)(y3-y1)-(x3-x1)(y2-y1)}$$

onde R, G, B, Z e T são representados por U.

O denominador das expressões só contem termos em x e y e é, de fato, o produto vetorial que já é calculado na triangularização. O inverso desse valor é armazenado para minimizar o número de divisões. O cálculo da perspectiva inclui divisão de x e y pela profundidade z, fazendo uso do divisor.

O hardware incremental DDA executa um algoritmo em laço duplo. O laço externo varre a imagem na direção de y. Para cada y este determina o inicio e o fim da varredura horizontal necessária para o preenchimento do triângulo. Nos extremos da linha é efetuado o anti-aliasing. Os pixels próximos da aresta dos triângulos recebem uma informação de cor intermediária com a cor de fundo ou contraste, segundo uma dosagem que depende do afastamento do pixel da borda real do triângulo. O laço interno executa a varredura na direção x, calculando incrementalmente os atributos R, G, B, Z e T para cada pixel.

A porta de atualização é um registrador que armazena os dados calculados pelo hardware incremental, enquanto este calcula um novo pixel dentro do próximo ciclo de memória. Para cada pixel o DDA avalia se (x,y) está dentro dos limites estabelecidos pela janela, ignorando-o caso esteja fora. Caso esteja dentro, a profundidade z é comparada com o valor armazenado no z-buffer para a determinação do obscurecimento deste pixel. Se não for obscurecido, R, G e B são transferidos para a memória de vídeo bem como z para a o Z-buffer. A transparência, se houver, deve ser considerada antes da transferência destes para a memória de vídeo, fazendo-se uma média ponderada de R, G e B com os valores lidos na memória de vídeo.

O sistema prevê uma extensão da memória de vídeo para um cursor 3-D não destrutível, que se aproveita da constância da informação de profundidade z na memória de vídeo. Assim consegue-se mais um recurso para que o usuário possa avaliar melhor a profundidade da imagem.

Na época de publicação do artigo, 1984, não havia sido completamente implementado um protótipo mas os autores estimaram o desempenho do sistema pela capacidade média dos DSPs do pipeline para processar polígonos, não considerando o tempo de acesso às memórias. Suas estimativas indicam a possibilidade da geração de 10.000 polígonos por segundo, adequado para animação de imagens com algumas centenas de polígonos ou para gerar imagens detalhadas de vários milhares de polígonos em poucos segundos. Se considerarmos que a taxa de geração de pixels está limitada pelo ciclo de acesso à memória de vídeo podemos dizer que o tempo necessário para a alteração de uma imagem completa de 1024x1024 é no mínimo 0,4s, limitando a animação para no máximo 2,5 quadros por segundo.

3.5.2 - Pixel-Planes [FUC 85]

Esta é uma interessante arquitetura de processamento paralelo para geração de imagens realísticas de sólidos 3▲D. É um sistema especial de memória de vídeo em que cada pixel incorpora um processador série de 1 bit capaz de calcular a expressão linear A.x + B.y + C para cada pixel (x,y) simultaneamente.

Trata-se de uma arquitetura SIMD de processadores de 1 bit que recebem dados de uma estrutura especial de ULAs dispostas em árvore para propagar serialmente a expressão linear até cada pixel. Uma variedade de algoritmos foi desenvolvida para explorar essa capacidade de avaliar rapidamente a expressão linear. Com isso pode-se determinar:

- os limites das bordas dos polígonos
- o sombreamento contínuo de Gouraud
- a profundidade Z para a remoção de pixels ocultos
- os limites de volumes para a geração de sombras
- o anti-aliasing
- a transparência
- mapas de texturas

Para os experimentos os autores utilizaram um chip protótipo de 8x8 pixels, cujo circuito é apresentado na figura 3.7.

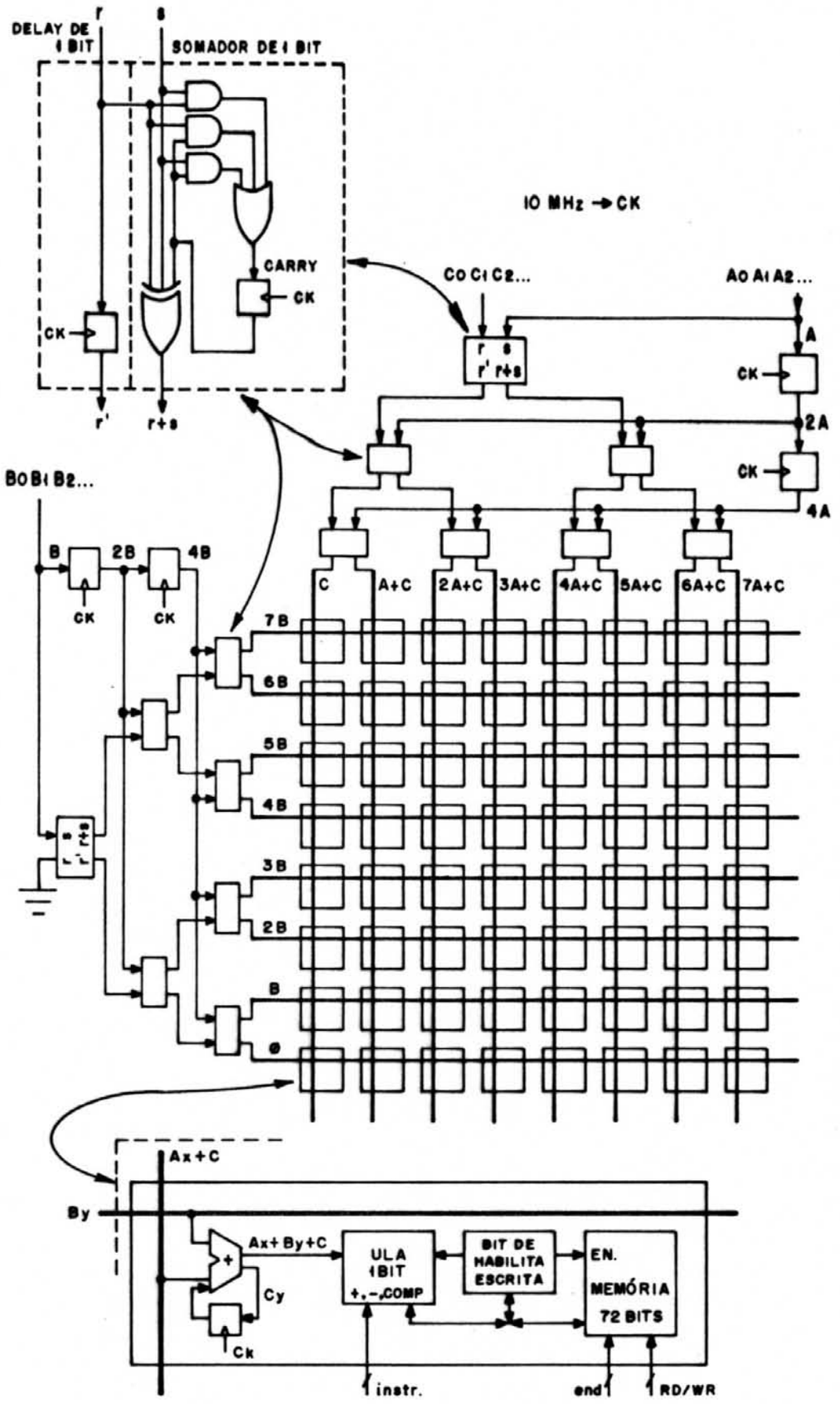


Fig 3.7. - Circuito do Pixel-Planes

Embora seja um processador paralelo, com um processador para cada pixel, a grande velocidade deve-se mais à árvore binária de somadores de 1 bit que computa a expressão linear para cada pixel (x,y) em todo o sistema. Fornecendo-se serialmente os coeficientes A, B e C, as duas árvores multiplicadoras mais o somador de 1 bit em cada pixel computam a expressão $F(x,y) = A.x + B.y + C$.

A árvore superior (vide fig. 3.7) gera a expressão $A.x + C$ e a árvore da esquerda a expressão $B.y$, que em cada pixel (x,y) serão somadas.

Cada processador de pixel contém um somador de 1 bit, uma ULA de 1 bit, um "flag" ou registro de habilitação de escrita na RAM, e uma RAM de 72 bits. As operações de transferência com a memória e a função da ULA são controladas da mesma maneira em todos os pixels por um barramento de instruções e de endereço de RAM comuns. Os 72 bits da memória RAM podem ser arranjados para comportar as informações de R, G e B, a profundidade Z para remoção de pixels ocultos, resultados intermediários e os bits de controle e habilitação.

Cada nodo da árvore contém dois sinais de entrada, "r" e "s", e dois de saída, "r'" e "r+s". A saída "r'" é o atraso de 1 clock da entrada "r" enquanto que a saída "r+s" é a soma parcial de um bit entre "r", "s" e o carry da soma do clock anterior, armazenado em um flip-flop. Assim executa-se a soma de dois valores serialmente, a partir do bit menos significativo, sincronamente ao clock.

Os coeficientes A, B e C entram serialmente no topo das árvores a partir de seus bits 0. A e B passam por flip-

flops de atraso para a obtenção dos termos 1xA, 2xA e 4xA, na árvore superior, e 1xB, 2xB e 4xB, na árvore lateral. A partir desses, somados entre si nas 8 combinações possíveis, a árvore obtém os termos A, 2A, 3A, 4A, 5A, 6A e 7A na árvore superior, e B, 2B, 3B, 4B, 5B, 6B e 7B na árvore lateral. Nas folhas das árvores temos respectivamente A.x (X=0..7) e B.y (y=0..7).

As saídas das árvores se cruzam nos processadores de pixel, onde um somador de um bit (com acumulador de carry) em cada pixel obtém o seu resultado para A.x + B.y + C . Este resultado é então comparado e/ou combinado com informações já armazenadas na memória de cada pixel.

Prenchimento de polígonos convexos com sombreamento contínuo

Inicialmente deve-se determinar quais são os pixels internos ao polígono. Para isso procede-se da seguinte forma:

1 - Inicializa-se todos os pixels com seu registrador de habilitação de escrita na RAM (RHE) com 1 (escrita habilitada).

2 - Para cada aresta do polígono é distribuída para todos os pixels a expressão linear F(x,y) = Ax + By + C, representando a equação da aresta, de maneira que no lado da aresta que corresponde ao interior do polígono F(x,y) seja positivo e do outro lado negativo. Em cada pixel o sinal de F(x,y) é combinado com o RHE de maneira que este seja desativado quando F(x,y) for negativo, indicando pixel externo ao polígono.

No final de todas as arestas tem-se o interior do polígono representado pelos bits RHE=1 (válido somente para polígonos convexos).

Em seguida executa-se o teste de visibilidade, para a remoção de pixels ocultos, distribuindo-se ao longo dos pixels a profundidade Z do polígono. Esta será comparada, em cada pixel, com o valor Z já presente na memória, determinando se este pixel deste polígono é visível ou se está obscurecido por outro polígono já transferido para a memória. Se o Z na memória for menor que o Z calculado, isto indica que o pixel presente na memória obscurece o calculado. Neste caso o pixel não deve ser escrito, o que é evitado desligando-se o seu RHE. Caso o pixel não seja obscurecido o novo e menor Z é transferido para a memória e o RHE é inalterado.

Após isto temos a configuração dos bits RHE representando o polígono recortado pelos polígonos já transferidos para a memória. Só agora é possível utilizar o espalhamento de F(x, y) para as componentes R, G e B, da cor, para um sombreamento contínuo de Gouraud.

Geração de sombras

Depois que a imagem visível está construída, pode-se utilizar os recursos dos Pixels-Planes para a geração de sombras por fontes luminosas puntuais. A geração de sombras aproveita-se da constância da profundidade Z em cada pixel, embora todos os polígonos tenham que ser propagados novamente para a determinação dos volumes de sombra, para cada fonte de luz.

Procedimento:

1 - Para cada pixel aloca-se um "flag" de indicação de sombra, chamado RS, inicializado em 0.

2 - Para cada poligono se ativa todos os RHE. Para cada aresta do poligono propaga-se a equação do plano formado com esta aresta e o ponto onde encontra-se a fonte de luz, de forma a que cada pixel receba a profundidade Z desse plano. Este plano P define dois semi-espaços, aquele que contém o poligono e aquele que não o contém.

Se o observador e o poligono corrente estão no mesmo semi-espaço relativo ao plano P então são desabilitados os pixels que estão abaixo do plano P, e se o observador não estiver no mesmo semi-espaço que o poligono, os pixels acima deste plano são ativados. Para isso inicialmente é determinada a equação do plano P, na forma Z=F(x,y); escolhe-se um vértice (xi,yi) que não esteja contido no plano P e avalia-se o sinal de F(xi,yi), que, se for positivo, indicará que o observador e o poligono estão no mesmo semi-plano, caso contrário estarão em semi-planos diferentes.

Em seguida os coeficientes A, B e C da equação do plano são processados para a determinação de F(x,y) de todos os pixels. Se F(xi,yi) é positivo, todos os pixels cujo Zmin é menor que F(x,y) são desabilitados; e se F(xi,yi) for negativo, todos os pixels cujo Zmin é maior que F(x,y) são desabilitados, por RHE=1.

Para cada poligono o conjunto de pixels visiveis que recebem sombra são determinados e seu RHE é colocado em 1. A operação lógica "OU" entre o RS e o RHE é armazenada em RS.

Após processar-se todos os polígonos, aqueles pixels que estão sob sombra contêm RS=1, e sua cor é subtraída de um fator apropriado.

O autor enumera os seguintes algoritmos que podem ser executados no Pixel-planes:

- Recorte da imagem por um plano invisível
- Aproximação para o desenho de esferas sombreadas
- mapeamento de texturas
- transparência
- anti-aliasing

Os autores esperam, no futuro, integrar a arquitetura de pixels-planes em um display de tela plana baseada em silício, de maneira que o Display por si próprio possa executar as computações de imagem.

As estimativas de desempenho feitas pelo autor consideram um clock de 10MHz e uma tela de 512x512 composta de múltiplos pixels-planes de 64 pixels cada. Os resultados são os seguintes:

- 30.000 triângulos/seg. com remoção de partes ocultas e sombreamento contínuo;
- 21.000 triângulos/seg. com a inclusão do cálculo de sombras.
- 25.000 esferas sombreadas/seg.

Uma extensão dos Pixel-planes para interpolação quadrática, intitulada Pixel-powers, é apresentada no artigo [GOL 86]. O aprimoramento reside na árvore de geração que é capaz de avaliar a expressão $AX^2 + BXY + CY^2 + DX + EY + F$ e

espalhá-la entre todos os pixels simultaneamente. Os recursos são os mesmos que os dos pixel-planes, exceto que os objetos podem ser descritos por superficies quadráticas.

4 - PROJETO DO G I R T D

GIRTD - Gerador de Imagens Realisticas Tridimensionais

Este capitulo tem por objetivo apresentar o desenvolvimento de um projeto em processamento paralelo para a geração de imagens realisticas em sistemas gráficos baseados na exibição por tubos de varredura, intitulado GIRTD.

Na primeira parte deste capitulo serão apresentados os objetivos deste projeto, imposições iniciais e aplicações a que se destinam. Isto servirá de partida para as segunda e terceira partes deste capitulo que são o projeto propriamente dito, até um nivel de detalhamento dos algoritmos envolvidos bem como a descrição do hardware a nivel de blocos funcionais com algumas referências a alguns componentes eletrônicos comerciais para uma base realistica do desempenho do equipamento. Serão discutidas as escolhas de soluções bem como justificadas estas decisões.

Por fim, na quarta e última parte, serão levandas possibilidades e propostas para futuras melhorias no equipamento.

4.1 - Introdução

4.1.1 - Imposição de metas e previsão do porte

Como parâmetros iniciais para a determinação do projeto temos algumas aplicações para as quais o GIRTD poderá contribuir, que podem ser simuladores de vôo, sistemas para produção de imagens sintéticas animadas para televisão, ou sistemas para modelagem gráfica e interativa de sólidos com imagens realísticas, com a possibilidade de um desempenho de tempo real.

Com essas imposições preliminares, algumas conclusões relativas ao porte do projeto podem ser tiradas:

- um grande volume de memória de vídeo será necessário para abarcar tanto a alta resolução exigida, usualmente 1 Mpixel (1024x1024) ou mais, para estas aplicações, quanto a razoável quantidade de planos de memória, ou bits por pixel, necessários para o grande número de cores usadas pelo realismo;
- devido às exigências de tempo real dever-se-á adotar estruturas especiais de acesso à memória capazes de vencer o gargalo de tempo de acesso a estas;
- como unidades de processamento para o cálculo dos pixels é descartado o uso de processadores comerciais de uso geral.

A geração de pixels faz uso de algoritmos não muito complexos (tal como a remoção de faces ocultas por Z-buffer) mas que, para atender a demanda de pixels exigida pela geração a tempo real, têm que ser executados a uma velocidade que só pode ser atingida por processamento paralelo. Uma geração de imagens a uma velocidade de 20 Mpixels/s (50ns/pixel) é considerada razoável para a

animação a tempo real em uma tela de 1 Mpixel e portanto devemos ter algo dessa ordem como objetivo.

Cabe observar aqui que entendemos por imagens realisticas aquelas que pelo menos apresentam remoção de faces ocultas e sombreamento continuo ao longo da curvatura das superficies e eventualmente um anti-aliasing, embora existam outras qualidades desejadas na geração de imagens com realismo tais como a refração, sombras, texturas, etc, que trarão uma imagem bem mais fiel à realidade mas exigem um processamento numérico muito intenso, não sendo o objetivo desse trabalho.

Para se obter um sombreamento das imagens sem quebras na coloração precisaremos ter à disposição uma tal gama de cores de maneira que, numa escala crescente de variação de intensidades, cores vizinhas praticamente se confundam. O uso de 8 bits (256 niveis) para a representação de cada uma das componentes de cor (R, G e B) tem demonstrado na prática ser capaz de atender essa exigência. Isso é o que adotaremos neste projeto, ou seja, um total de 24 bits (3x8) ou 16 milhões de cores possiveis (2^{24}).

4.1.2 - Visão global do equipamento: -ambiente de trabalho-

Este equipamento deverá servir a um computador hospedeiro. Neste computador rodará a interação com o usuário bem como algum utilitário especial que necessite realimentar o usuário com informações gráficas de imagens dos objetos envolvidos neste aplicativo. Exemplos de aplicativo deste tipo são sistemas de modelagem de animação de cenas sintéticas para televisão e sistemas de projeto de estradas de rodagem sobre terrenos montanhosos. Este hospedeiro emitirá informações ao GIRTD para que este possa

exibir estas imagens em um monitor RGB de alta resolução. Estes dados são emitidos pelo computador numa forma de descrição dos objetos sólidos ou planos. O GIRTD também obtém do computador hospedeiro informações de cor e do comportamento da reflexão e da transparência das superfícies destes objetos e deverá ser capaz de gerar rapidamente imagens baseadas nestes parâmetros fornecidos.

A escolha da maneira pela qual os objetos serão descritos no ambiente do GIRTD é de primordial importância na definição de sua arquitetura. A técnica que aqui usaremos para descrever os objetos sólidos emitidos pelo hospedeiro é a B-rep poliédrico (representação por fronteiras poligonais) por ser uma técnica bastante difundida, por ser de fácil manipulação e por não exigir muito processamento numérico. Nesta técnica os objetos tridimensionais são definidos pelas superfícies que os contornam (sua fronteira) aproximadas por polígonos. Esta aproximação implica no uso de uma estrutura de dados bem mais simples para a representação dos objetos além de requerer menor ocupação de memória e poder de processamento para manipular esta descrição, em comparação com outras técnicas. Os objetos são então representados como se fossem poliedros definidos pelas suas faces, que por sua vez são definidas pelas coordenadas espaciais de seus vértices. A responsabilidade de gerar essa representação, ou de converter de uma outra representação para esta B-rep, fica atribuída ao computador hospedeiro.

4.1.3 - O pipeline gráfico

O GIRTD gera as imagens manipulando apenas os polígonos que descrevem os sólidos. Estes polígonos deverão passar por uma seqüência de processamentos ilustrados na figura 4.1.

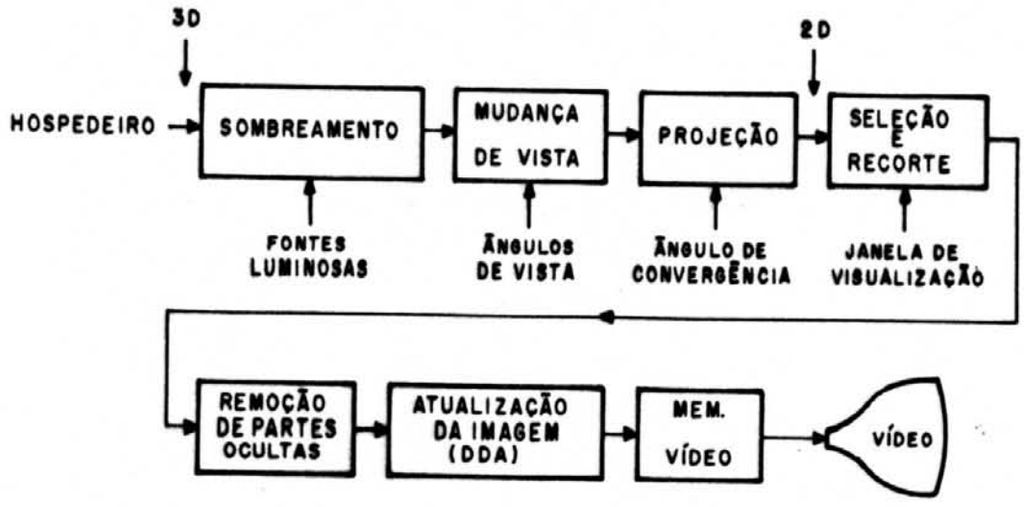


Fig. 4.1 Pipeline típico para a geração de imagem baseada em representação B-rep

O primeiro estágio deste pipeline é responsável pelo sombreamento da imagem. Aí são aplicados modelos matemáticos parametrizados para a síntese de iluminação, sombreamento e transparência dos objetos. Esses cálculos usualmente são aplicados somente nos vértices dos polígonos para posterior aproximação linear do restante da área dos polígonos, feita no estágio DDA.

O segundo bloco executa a mudança de vista da imagem. Essa operação depende de qual ponto de vista o usuário deseja que a imagem seja exibida na tela gráfica. As coordenadas dos vértices dos poliedros são trocadas por outras relativas a um novo sistema de eixos coordenados associado aos ângulos de visualização da imagem.

Após vem a projeção da imagem. As coordenadas dos vértices são agora alteradas novamente para dar uma noção de profundidade à imagem. Os objetos mais ao fundo (longe do observador) ficarão proporcionalmente menores.

No estágio de seleção e recorte os polígonos que, após a mudança de vista, caírem fora da área de visão serão descartados enquanto que outros que estiverem parcialmente

visiveis na tela terão que ser recortados, ou seja, transformados em outros poligonos que representam a porção visivel destes.

Por fim, antes da atualização da memória imagem (ou memória de video), deve-se remover totalmente ou parcialmente os poligonos sobrepostos por outros para uma correta exibição da imagem. Este poligonos devem ser recortados ou eliminados, conforme a sua situação. Para isso poderia-se executar o recorte, como feito acima para a área de visão, para cada par de poligonos, só que desta maneira esta tarefa seria um grande gargalo devido à grande quantidade de processamento requerida, prejudicando o balanceamento do fluxo de dados no pipeline além de comprometer as exigências de velocidade para a geração de imagens a tempo real. Quando se exige alta velocidade neste processo costuma-se empregar algoritmos de remoção de faces ocultas baseados em memória de video. Estes se prestam muito bem para aplicações de geração de imagens, representadas por B-rep, a tempo real. Os principais algoritmos desse grupo são o "do pintor", "Z-buffer" e "scan line coherence" que já foram discutidos no capitulo II.

O último estágio do pipeline gráfico é o responsável pela atualização da memória de imagem. Se estivermos usando alguma das técnicas mencionadas acima este estágio também se responsabilizará pela remoção de faces ocultas. Neste processo de atualização da memória também podem ser executadas algumas tarefas desejáveis para o realismo: o sombreamento e o anti-aliasing.

O sombreamento, no caso de B-rep, pode ser uma interpolação linear das informações de cor e luminosidade calculadas nos vértices dos poligonos. Isso implica em que cada pixel poderá ter uma cor levemente diferente da do seu

vizinho, sendo necessário portanto processamento individual para cada pixel. Como a interpolação é linear, podemos usar métodos incrementais para a obtenção dos atributos dos pixels. Estes métodos têm a grande vantagem de necessitar apenas de operações de soma e subtração para a obtenção da cor e luminosidade de um pixel vizinho. Esse processamento é feito por um circuito comumente chamando de DDA (Digital Differential Analyzer).

Um DDA é capaz de gerar somente um pixel de cada vez, embora com grande velocidade, pois opera apenas com operações de soma e subtração de números inteiros. Se for capaz de gerar pixels a uma taxa de 20 Mpixels por segundo, se prestará para as aplicações de modelagem a tempo real e animação.

O objetivo deste trabalho será o de propor um equipamento gráfico para atender as caracteristicas aqui colocadas, fazendo uso das técnicas mencionadas:

- estrutura em forma de pipeline gráfico;
- uso de DDA
- remoção por algoritmo baseado em memória de video

Por ser o DDA o estágio mais critico do sistema, onde se encontram os gargalos de tempo de acesso de memória e processamento rápido, o foco principal deste projeto será a especificação e detalhamento de um DDA capaz de executar as tarefas acima mencionadas. Os estágios anteriores do pipeline (projeção, recorte, etc) são todos algoritmos conhecidos e não requerem muitos cuidados para que possam ser implementados em processadores de uso geral sem um comprometimento do equilíbrio do fluxo de dados do pipeline. Estes serão descritos na terceira parte deste capitulo.

4.2 - O projeto

4.2.1 - Algoritmo para remoção de faces ocultas

Os algoritmos que se enquadram dentro das exigências de velocidade impostas aqui são aqueles baseados no acesso à memória de vídeo: o "do pintor", o "Z-buffer" e o "scan-line coherence".

O último executa a remoção de partes ocultas a nível de linha de varredura. O artigo [NII 84], apresentado no capítulo III, sugere uma estrutura MIMD com esse algoritmo distribuído entre os PEs (elementos de processamento) para a execução em paralelo de porções distintas da imagem. O algoritmo "scan-line-coherence" em si é dificilmente particionável em sub-tarefas para execução em paralelo. O paralelismo sugerido no referido artigo decorre do particionamento da área de trabalho de imagem em partes independentes e não pelo particionamento do algoritmo em si. Isso exige também que cada unidade de processamento tenha uma réplica da "tabela de segmentos ativos" usada pelo algoritmo.

O algoritmo "do pintor" executa a remoção de faces ocultas através da sobreposição destas na própria memória de vídeo, à medida que estas vão sendo aí preenchidas. O preenchimento de polígonos se dá numa ordem partindo das faces mais distantes do observador até as faces mais próximas, que, como são as que provavelmente cobrirão as outras, devem ser preenchidas por último.

O algoritmo do pintor é eficiente na remoção de partes ocultas mas exige que seja feita uma pré-ordenação em

todas as faces que participam da imagem. Todos as combinações de pares de faces devem ser comparadas entre si a fim de se determinar a ordem de preenchimento em função de quem está na frente ou atrás. Esta comparação é, às vezes, difícil de ser avaliada com segurança por um algoritmo, podendo-se até encontrar casos insolúveis, como faces inter-penetrantes.

O algoritmo Z-buffer é semelhante ao anterior, acrescido de uma extensão à memória de vídeo, chamada de "Z-buffer". Este armazena, para cada pixel, o valor de sua profundidade Z, para auxiliar na detecção de pixels ocultos e que não devem ser atualizados na memória. Assim, a remoção é feita sem a necessidade de pré-ordenação e, como é feita independentemente para cada pixel dos polígonos, os problemas encontrados no algoritmo do pintor, tais como as faces inter-penetrantes, são aqui resolvidos.

O uso deste algoritmo simplifica bastante os recursos de hardware e software requeridos, se comparado com o "scan line coherence" por exemplo, mas exige um acréscimo de memória eletrônica para armazenamento da profundidade z. Devido ao desenvolvimento das tecnologias empregadas em memórias dinâmicas dispomos agora de memórias bem maiores, mais baratas e mais rápidas, reduzindo drasticamente o custo por bit, o que vem contribuindo para a grande aceitação do algoritmo Z-buffer em equipamentos gráficos profissionais voltados à geração de imagens tridimensionais com remoção de faces ocultas.

Diversos artigos propõem arquiteturas baseadas na técnica Z-buffer ([FUJ 84] [BUR 87] [TUN 87]) e todos utilizam um único DDA para o preenchimento da imagem com realismo. Além disso já existem diversas estações gráficas comerciais para a geração de imagens realísticas por B-rep

poligonal fazendo uso do Z-buffer (exemplos: [IKE 84] e [MOK 87]), já que é grande a gama de aplicações para equipamentos com esses recursos.

4.2.2 - O algoritmo DDA

A função básica do DDA do GIRTD é desenhar polígonos na memória de imagem, pixel a pixel, a uma taxa mínima de 20Mpixels/s, com as características de coloração necessárias para o sombreamento contínuo, fazendo uso de Z-buffer para a remoção de faces ocultas. Para isso o DDA deverá dispor de parâmetros que desencadearão esse preenchimento da memória de maneira mais rápida possível. Esses parâmetros deverão ser bastante detalhados para que o DDA não perca tempo com operações afora o preenchimento da imagem. Para tal será incluído um estágio antes do DDA responsável pelo preparo destes parâmetros - o preparador do DDA, que não está representado na figura 4.1.

O preparador do DDA recebe os polígonos descritos pelas coordenadas dos seus vértices (já nas coordenadas de tela), os quais também trazem as informações locais de coloração e luminosidade dos próprios vértices. Este por sua vez entrega ao DDA informações tais como o ponto de partida para o preenchimento do polígono, as derivadas em relação a X e Y das cores, para o processamento incremental, e alguns parâmetros para que o DDA reconheça os limites do polígono em questão. Este estágio preparador do DDA será descrito na terceira parte deste capítulo.

No DDA o preenchimento se dará controlado por um algoritmo que percorra os pixels da imagem, determinando a

sua cor e se devem ou não ser atualizados. E desejável que o algoritmo do DDA seja capaz de:

- identificar os limites de preenchimento dos polígonos;
- andar em passos unitários de pixel evitando-se assim a necessidade de operações de multiplicação para a determinação dos atributos dos pixels afastados por mais de uma unidade;
- executar o anti-aliasing nas bordas dos polígonos;
- usar aritmética inteira (ponto fixo);
- identificar, através do Z-buffer, pixels sobrepostos.

Por definição, as informações de cor, R, G e B, e a profundidade Z serão aproximadas linearmente ao longo da área do polígono. O comportamento de R, G, B e Z pode então ser expresso, simbolizado por P, pela equação linear

$$P = a.x + b.y + c \qquad \text{ou} \qquad P = P_x'.x + P_y'.y + P_0$$

onde:

$$a = P_x' = dP/dX \qquad b = P_y' = dP/dy \qquad c = P_0 = P(x=0, y=0)$$

x, y = endereço dos pixels na tela e $P \in \{ R, G, B, Z \}$

O DDA então recebe do preparador, como parâmetros de entrada, os valores iniciais P_i e suas derivadas P_x' e P_y':

$$\begin{array}{lll} R_x' & R_y' & R_i \\ G_x' & G_y' & G_i \\ B_x' & B_y' & B_i \\ Z_x' & Z_y' & Z_i \end{array}$$

onde Pi é o valor da iteração inicial, conforme o endereço de partida do pixel xi, yi , ou seja,

$$P_i = P(x=x_i, y=y_i) = P_x'.x_i + P_y'.y_i + P_0$$

O algoritmo DDA parte do pixel inicial xi, yi com sua profundidade inicial Zi e cores Ri, Gi e Bi. Para cada pixel visitado o algoritmo decide se este deve ou não deve ser atualizado na memória, dependendo da comparação de Z com Zb (Z buffer). A visita ao pixel seguinte é sempre feita por intermédio de saltos unitários nas direções x ou y:

$$x = x \pm 1 \qquad \text{ou} \qquad y = y \pm 1$$

Em virtude dos saltos serem unitários os valores P são atualizados por somas ou subtrações com as respectivas derivadas em relação a x ou y:

$$P = P \pm P_x' \qquad \text{ou} \qquad P = P \pm P_y'$$

O problema agora é como percorrer os pixels dentro do polígono somente por saltos unitários e como identificar as fronteiras dos polígonos. Vamos supor inicialmente que o algoritmo do DDA tenha condições de saber a cada instante se o pixel é interno ou externo ao polígono.

Temos que ter um algoritmo que seja capaz de percorrer todos os pixels internos do polígono sem perder tempo visitando pixels já visitados ou tendo que ler marcas do tipo "já fui visitado!" nesses pixels.

Uma primeira idéia para esse algoritmo seria a de percorrer em zigue-zague os pixels internos ao polígono, ou seja, encerrando-se uma linha horizontal parte-se para a imediatamente inferior, percorrendo-a em sentido contrário, e assim por diante. A realização do algoritmo desta maneira apresenta uma grande dificuldade. Ao descer para a linha inferior, somando 1 ao y, o pixel aí encontrado pode estar bastante fora do polígono ou pode estar com vizinhos para

serem visitados de ambos os lados deixando a dúvida "que sentido percorrer os pixels agora, para direita ou para esquerda?". Isto sugere que é inevitável que se use uma pilha para armazenar um caminho a percorrer, deixado para mais tarde; ou que se permita que se possa passar mais de uma vez pelo mesmo pixel, o que é uma perda de tempo. Esta pilha teria que armazenar, além do x,y do pixel postergado, os atributos R, G, B e Z deste pixel, evitando-se assim a necessidade de operações de multiplicação para a reobtenção destes na busca de um pixel da pilha.

Essa pilha pode complicar a arquitetura do DDA e consumir um tempo significativo para o seu acesso, além de desestruturar a idéia inicial de andamento incremental. Se ela for inevitável devemos ao menos reduzi-la ao máximo.

E interessante comentar que normalmente é usada recursividade nos algoritmos para o preenchimento de quaisquer figuras em uma tela gráfica, o que, de uma certa forma, vem reforçar a idéia de que esse tipo de tarefa necessita de pilha. Aqui devemos evitar recursividade pois consome muito tempo além de fazer visitas repetidas a uns mesmos pixels.

O algoritmo que aqui propomos, mostrado na figura 4.2, se aproxima bastante do ideal sugerido.

Ele não usa propriamente uma pilha mas um único registrador auxiliar para armazenar caminhos postergados e, se o implementarmos com uma série de ULAs (Unidades Lógicas e Aritméticas) simples, para a determinação incremental de R, G, B, X, Y e Z, operando em paralelo, podemos atingir a taxa de pixels desejada (<50ns/pixel).

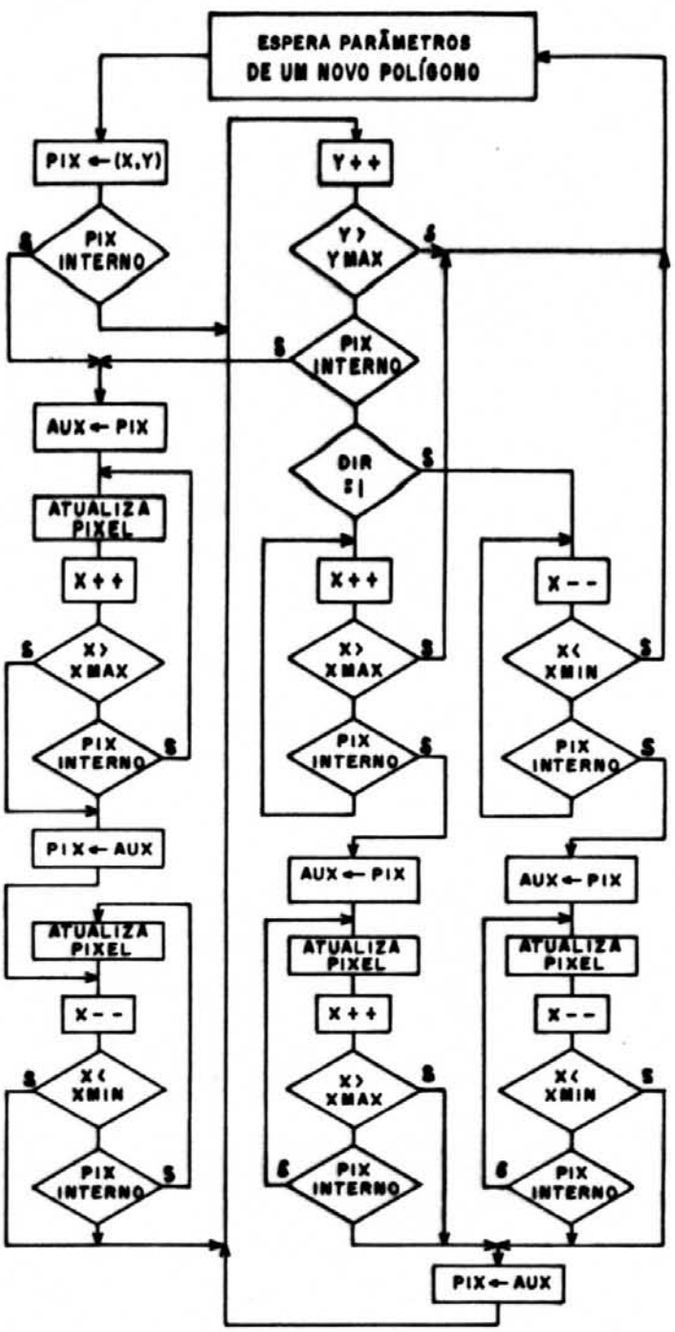


Fig. 4.2 - Algoritmo DDA

O algoritmo supõe a disponibilidade da informação de pixel interno ou externo ao poligono e se utiliza desta para desviar o caminho ao longo dos pixels.

O inicio do preenchimento se dá no pixel mais superior, à esquerda ou direita, do polígono. O movimento de "varredura" por parte do algoritmo é prioritariamente na horizontal, como se o algoritmo fosse um laço duplo com o laço interno varrendo em x e o externo em y. Ao acabar de preencher completamente uma linha horizontal interna ao polígono, o DDA passa a preencher a linha imediatamente inferior, e assim por diante até o fim do polígono. Esse movimento ora se comporta como um zigue-zague, ora como uma varredura de uma direção só, dependendo do formato do polígono.

O teste <DIR=1?> é necessário quando, após um incremento em y, o pixel estiver fora do polígono. Este teste indicará ao algoritmo que direção tomar (esquerda ou direita) até reencontrar a região interna do polígono. Maiores detalhes deste teste serão vistos mais adiante.

Este algoritmo, como era de se esperar, não é capaz de visitar todos os pixels do polígono apenas por saltos unitários. Uma quebra de caminhamento ocorre toda a vez que a varredura atinge um pixel exterior, e o algoritmo passa a percorrer os pixels a partir de uma posição armazenada anteriormente no registrador AUX, que desempenha a função de uma pilha de um nível.

O algoritmo avalia também se a varredura não ultrapassou X_{min}, X_{max} e Y_{max}. A passagem por Y_{max} define o fim do polígono, detectado no laço externo. X_{min} e X_{max}, embora sua função básica não seja essa, estabelecem um pseudo-recorte a nível de preenchimento da tela. Estes dois atuarão principalmente no processo de antialiasing, que será descrito mais adiante. A atenção a esses limites, junto com o ponto de partida x_i, y_i, define uma janela retangular para uma limitação do preenchimento.

Como podemos ver, o algoritmo não passa por pixels já visitados, mas perde alguma fração de tempo visitando pixels externos e tangentes ao polígono para a identificação de suas fronteiras. Algum tempo é perdido também quando, ao incrementar y e cair fora do polígono, tem que caminhar, segundo a direção dada por "DIR", por esses pixels externos até alcançar novamente o interior do polígono. Isso acontece raramente, quando as arestas inferiores do polígono são muito inclinadas.

Cada deslocamento de pixel é simbolizado, no algoritmo da figura 4.2, por X++, X-- ou Y++ (Y-- não ocorre pois o preenchimento é de cima para baixo), como incrementos da linguagem "C". Cada um desses símbolos representa as seguinte operações paralelas:

```
X++  ≡  ( X=X+1   Z=Z+Zx'   R=R+Rx'   G=G+Gx'   B=B+Bx' )

X--  ≡  ( X=X-1   Z=Z-Zx'   R=R-Rx'   G=G-Gx'   B=B-Bx' )

Y++  ≡  ( Y=Y+1   Z=Z+Zy'   R=R+Ry'   G=G+Gy'   B=B+By' )
```

Estas 5 operações são independentes e podem ser executadas em paralelo, bem como as comparações entre X e Xmin e Xmax e entre Y e Ymax, que fornecerão condições de teste para o algoritmo.

A figura 4.3 sugere uma estrutura de hardware contendo seis elementos de processamento - EPs - cada um contendo uma ULA e registradores. Em cada um dos EPs responsáveis pela determinação de R, G, B e Z encontra-se uma ULA, para as somas e subtrações das derivadas, e quatro registradores, um para armazenar o resultado calculado e de saída, dois contendo as duas derivadas, e um auxiliar para armazenamento de pixel postergado. Os EPs X e Y são

diferentes pois suas ULAs só calculam incremento e decremento, em paralelo com uma comparação de magnitude. Também possuem um registrador para o resultado e saida, e outro auxiliar para o endereço de pixel postergado, enquanto que os outros dois registradores são os limites máximo e minimo de excursão.

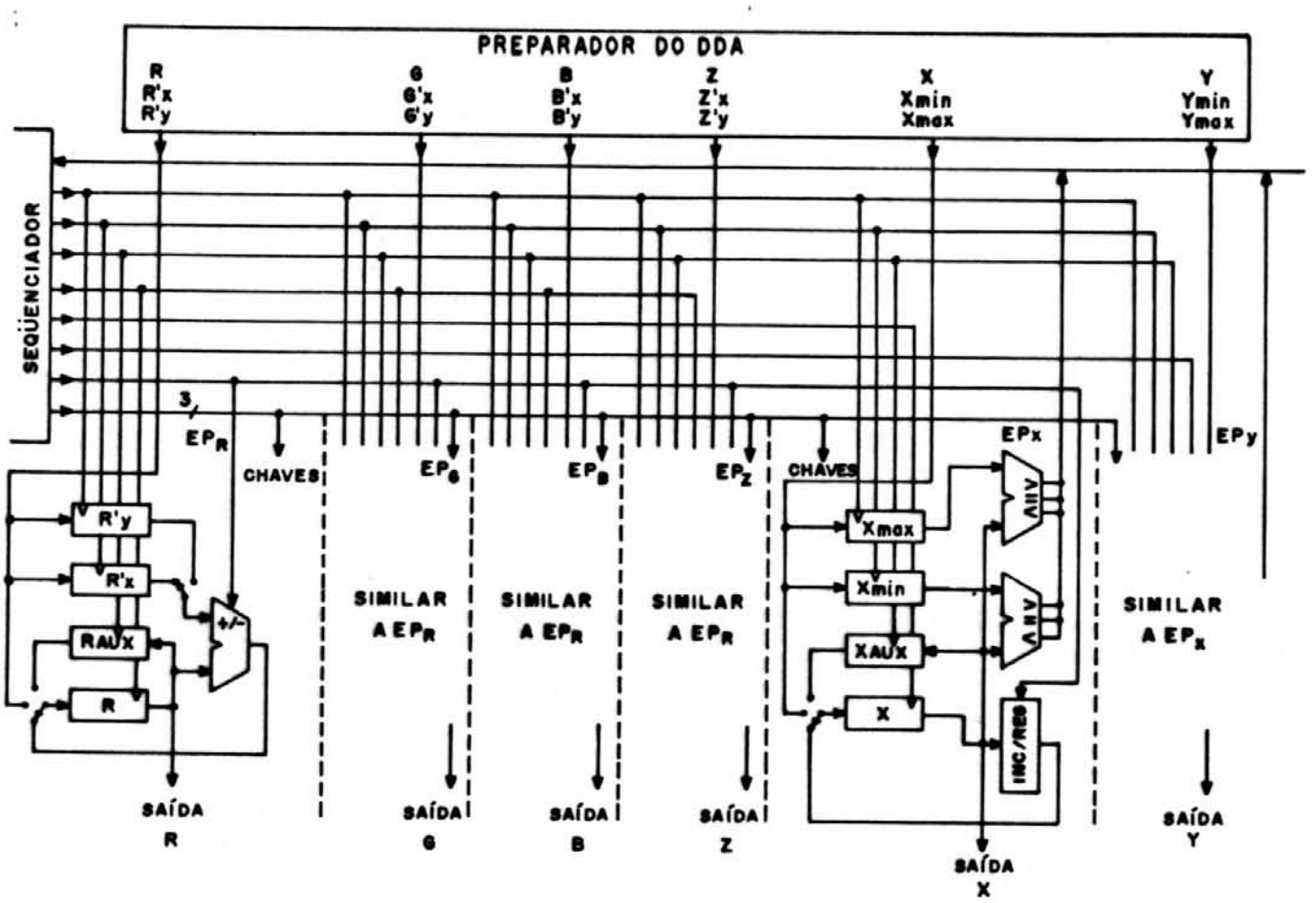


Fig. 4.3 - Bloco operacional do gerador de pixels

O seqüenciador possui o algoritmo DDA implementado em micro-código. O controle dos EPs é dado por algumas linhas de controle emitidas pelo seqüenciador. Uma delas

avisa as ULAs R, G, B e Z se a operação é de soma ou subtração, e as ULAs X e Y se é incremento ou decremento e se a comparação de magnitude é com o registrador max ou o min. Um outro sinal indica se a operação é na direção Y ou X. Este faz com que sejam selecionadas as devidas derivadas em R, G, B e Z, ao mesmo tempo que habilita ou o EP X ou o EP Y para trabalho.

O sequenciador também controla a carga inicial dos registradores dos EPs e as transferências entre os registradores de saida e o auxiliar.

Este algoritmo DDA supõe que cada linha horizontal, interna ao polígono, não possui interrupção, podendo ser causada por uma parte côncava deste. Pode não funcionar corretamente com polígonos côncavos restringindo seu uso para polígonos convexos. Como todo polígono côncavo pode ser convertido em um conjunto de polígonos convexos ou triângulos, esse problema pode ser contornado com a inserção de mais um estágio no pipeline gráfico responsável por esta conversão.

4.2.3 - Detecção das fronteiras dos polígonos

O algoritmo DDA supõe a existência de um mecanismo capaz de detectar se o pixel visitado pertence ou não ao polígono. Esse mecanismo deve operar em paralelo com a geração de pixels, não tomando tempo desta e fornecendo imediatamente, para cada pixel visitado, a informação de pixel interno ou externo.

Um meio de compatibilizar esta operação com a geração de pixel é de também usar métodos incrementais para isto, o que pode ser feito propagando-se incrementalmente,

ao longo da geração de pixels, os valores de distância entre o pixel sendo visitado e as retas formantes do polígono. Enquanto todas essas distâncias tiverem o mesmo sinal (todas positivas ou todas negativas) o pixel visitado estará dentro do polígono, caso contrário, fora (válido somente para polígonos côncavos).

A distância de um ponto a uma reta é dada pelo módulo de:

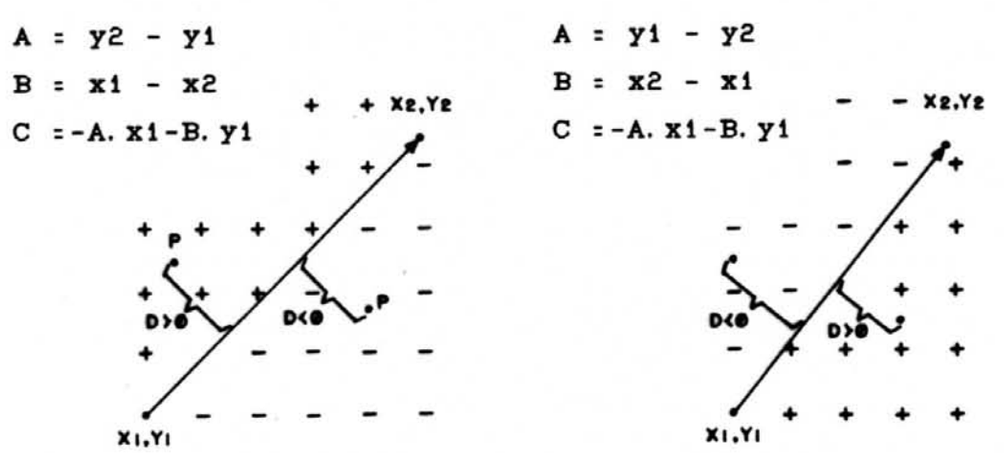
$$D = \frac{A.x + B.y + C}{\sqrt{(A^2 + B^2)}}$$

onde

x, y = ponto

A, B, C = definem a orientação e posição da reta

O sinal de D indicará de qual lado da reta o ponto está localizado. A correspondência entre o sinal e qual lado da reta depende da orientação da reta e de como foram obtidos A, B e C. A figura 4.4 mostra os dois significados possíveis para o sinal de D.



a) b)

Fig. 4.4 - Duas interpretações para o sinal de D

Para a correta interpretação do sinal das distâncias do pixel às retas definidas pelas arestas do polígono deve-se ter estas arestas todas com a mesma orientação circular e obter A, B e C de cada aresta respeitando essa orientação através da escolha de um dos significados da figura 4.4.

Por exemplo, usando-se o procedimento da figura 4.4 a) e representando-se as arestas em sentido anti-horário temos a indicação de ponto interno pela coincidência de todos os sinais positivos, como mostra a figura 4.5.

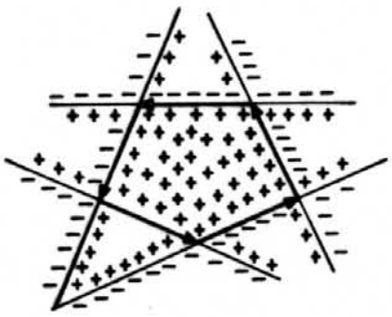


fig. 4.5 - sinais das distâncias às arestas.

O cálculo incremental das distâncias pode ser feito da mesma forma que para R, G, B e Z. Para cada distância D das arestas necessitamos de um EP, com uma ULA e quatro registradores, para execução em paralelo. Com isso o número de arestas permitido para cada polígono estará limitado pelo número de EPs disponíveis para o cálculo das distâncias. Para resolver esse inconveniente optou-se por limitar os polígonos a triângulos e por se intercalar um estágio no pipeline responsável pela decomposição dos polígonos em triângulos. Essa decomposição vem também resolver o problema dos polígonos convexos, mencionado há pouco. A partir desse estágio, chamado triangularizador,

toda a manipulação será sobre triângulos. Este estágio será descrito na terceira parte deste capitulo.

Agora o algoritmo é capaz de identificar, a cada pixel visitado, se este está dentro ou fora do polígono, agora restrito a triângulos, à custa de três EPs adicionais. A figura 4.6 sugere o circuito a se adicionar aos outros EPs já apresentados, e a detecção de pixel interno pelos sinais positivos das distâncias.

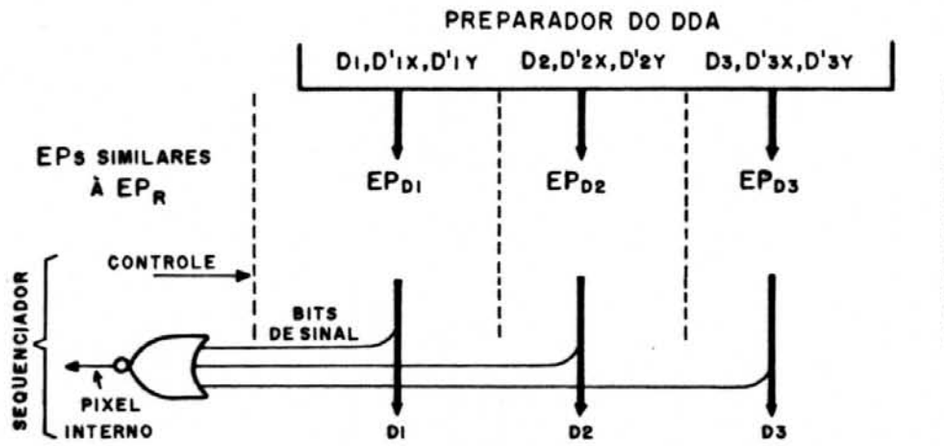


Fig. 4.6 - Circuito de detecção de fronteiras

Com a inclusão desses três EPs, de construção idêntica a dos outros EPs (talvez diferindo somente no número de bits necessários em seus registradores) o DDA necessita receber mais 9 parâmetros para o disparo do preenchimento de um triângulo, 3 por EP. Estes são as distâncias iniciais $D1_i$, $D2_i$ e $D3_i$, e suas derivadas em x e y, $D1_x'$, $D1_y'$, $D2_x'$, $D2_y'$, $D3_x'$ e $D3_y'$. Tanto esses parâmetros quanto os outros dos outros EPs são calculados pelo preparador do DDA e fornecidos ao DDA para cada triângulo a ser exibido. O meio de cálculo desse parâmetros também será descrito na terceira parte desse capitulo.

4.2.4 - Detecção do "DIR"

Como foi visto, o algoritmo DDA aqui proposto necessita de uma orientação, quando o pixel cai fora do triângulo por um salto positivo em Y, para decidir que direção tomar até encontrar novamente o interior do triângulo, pela direita ou pela esquerda. E necessário desenvolver um mecanismo para isso. A figura 4.7 mostra os dois casos possíveis.

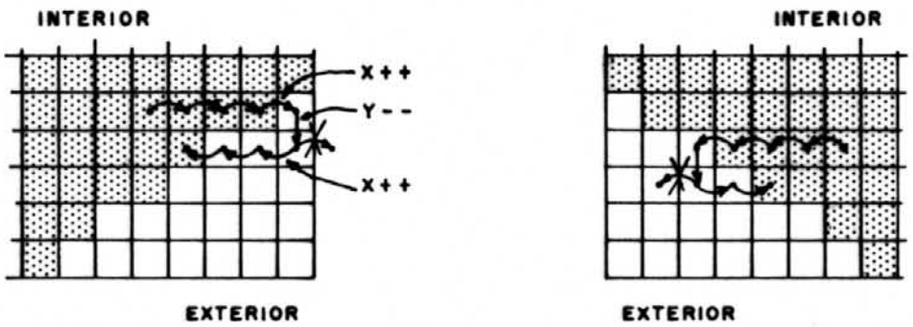


Fig. 4.7 - Retorno ao triângulo pela direita e pela esquerda

A solução baseia-se no fato de que, quando um pixel cai fora do polígono, somente uma das distâncias D_i se torna negativa, que é justamente a distância associada à aresta que o pixel cruzou para sair fora do triângulo.

A direção a tomar é aquela em que essa distância D_i cresce, a ponto de se tornar positiva novamente. Para isto basta se avaliar o sinal da derivada em relação a X dessa distância. Se D_{iX}' for positiva a direção crescente de D_i é para a direita (sentido crescente de X), caso contrário, para a esquerda.

A determinação de DIR é feita identificando-se, inicialmente, qual das direções D_i é negativa e então tomando-se o sinal da derivada D_{iX}' dessa distância como DIR. Isso pode ser feito por um circuito simples como o mostrado na figura 4.8.

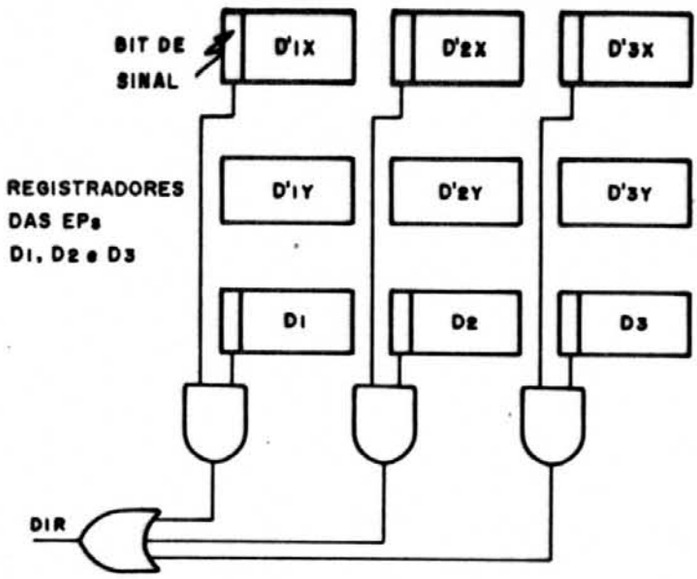


Fig. 4.8 - Circuito de detecção DIR

Se por um acaso duas distâncias forem simultaneamente negativas, isto indica que o pixel cruzou por um vértice, ou seja, duas arestas ao mesmo tempo. Certamente estas duas arestas terão as suas derivadas em relação a X com o mesmo sinal. Seus sinais só poderão ser diferentes se o pixel já tiver ultrapassado Ymax, o que já é identificado como fim de preenchimento, na comparação de Y com Ymax.

4.2.5 - Antialiasing

O que foi até aqui esboçado computa pixels para preenchimento de polígonos com sombreamento linear de Gouraud [GOU 71]. Outra facilidade também bastante desejada na geração de imagens realísticas, especialmente quando se trata de animação, é o antialiasing (anti-serrilhado), ou seja, a eliminação do efeito serrilhado nas bordas inclinadas. Este efeito é causado pela resolução limitada da imagem. Bordas inclinadas de figuras são aproximadas por uma composição de segmentos sucessivos horizontais e verticais com dimensões múltiplas inteiras da dimensão do pixel, que é a menor unidade representável na imagem.

Em aplicações de animação o serrilhado pode causar efeitos bastante visíveis e desagradáveis para a imagem. No movimento de uma figura ao longo da área da imagem ocorre um efeito de batimento entre o serrilhado e a resolução da imagem que se manifesta como um aparente movimento parasita do serrilhado ao longo das bordas das figuras.

No domínio freqüência esse efeito é encarado como a presença de componentes de freqüência muito acima daquelas necessárias para a representação da imagem. Por isso algumas técnicas antialiasing atuam como uma pós filtragem "passa-baixa" na imagem, resolvendo parcialmente o problema [FUJ 83].

Uma solução mais aceita é uma pré-filtragem "passa-baixa" que consiste em atualizar os pixels, nas bordas das figuras, com características de cor intermediárias entre a do próprio pixel e a cor do fundo encontrada neste pixel antes da atualização, conforme o afastamento deste com a

borda do poligono. Essa cor intermediária é obtida pela média ponderada:

$$Pix = PixCalc \cdot n + Fundo \cdot (1-n)$$

onde Fundo é a cor lida na memória de vídeo, PixCalc é a cor calculada pelo gerador de pixels e n é um fator que varia de 0 a 1 e determina a dosagem entre essas duas cores. Em termos de componentes de cor da imagem esse cálculo passa a ser:

$$Rb = R.n + Rb.(1-n)$$
$$Gb = G.n + Gb.(1-n)$$
$$Bb = B.n + Bb.(1-n)$$

onde (R, G, B) é o pixel calculado e (Rb, Gb, Bb) são as cores lidas da memória de vídeo (buffer) na posição deste pixel.

Nesta técnica de antialiasing o valor de n no interior do triângulo é sempre 1 e nas bordas decresce de 1 a 0, de acordo com o afastamento do pixel à borda, dando uma sensação de leve desfocamento nas bordas dos triângulos.

No capítulo II vimos e criticamos algumas técnicas para a obtenção de n. A técnica que aqui utilizaremos se aproveita da disponibilidade das distâncias às arestas, D1, D2 e D3. "n" é obtido a partir destas distâncias fazendo-se com que decresça linearmente, de 1 a 0, à medida que a distância do pixel externo ao triângulo varia de 0 a 1 (Di da aresta mais próxima variando de 0 a -1 respectivamente). Para afastamentos maiores que 1 (Di < -1) n será sempre 0 e para os pixels internos será sempre 1.

Para isso passaremos D1, D2 e D3 por uma função que chamaremos de Norm (normalização), cuja definição está na figura 4.9.

$$\text{Norm}(x) = \begin{cases} 1 \text{ para } 0 \leq x & \text{(interior)} \\ x+1 \text{ para } -1 < x < 0 & \text{(transição)} \\ 0 \text{ para } x \leq -1 & \text{(exterior)} \end{cases}$$

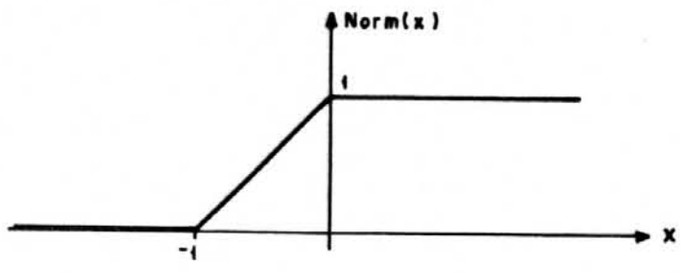


Fig. 4.9 - A função Norm(x)

Com essa função obtemos N1, N2 e N3 por:

N1 = Norm(D1)
N2 = Norm(D2)
N3 = Norm(D3)

Agora n deve ser obtido de maneira que: se N1, N2 e N3 = 1 então n = 1; se N1, N2 ou N3 = 0 então n = 0; se somente um deles é diferente de 1, n assume seu valor; e se mais de um N é menor que 1 este pixel estará nas imediações de um vértice do triângulo e aí duas arestas interagem para a determinação do antialiasing de maneira que o efeito de um vértice intensifica o outro, como um produto entre seus efeitos.

A operação para a obtenção de n em função de N1, N2 e N3 que desempenha o comportamento descrito acima é simplesmente:

$$n = N1.N2.N3$$

A figura 4.10 apresenta um exemplo do comportamento dessa técnica numa porção de um polígono exemplo. A esquerda da figura temos a borda do polígono representada por uma linha grossa e o valor de n escrito no centro de cada pixel. A direita temos o mesmo que na esquerda com o valor de n representado pela área dos círculos negros.

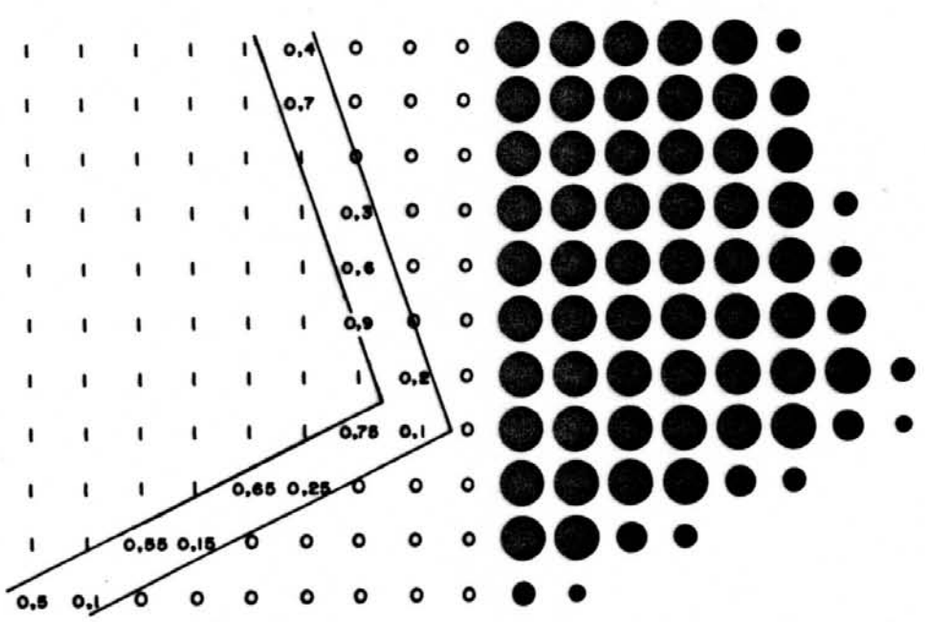


Fig. 4.10 - Antialiasing com a técnica descrita

Podemos ver que, nas imediações do vértice do triângulo, onde n é gerado pelo efeito de duas arestas, este não é devido à menor distância até o triângulo e sim pelo produto dos efeitos das distâncias entre esses pixels e a

projeção dessas duas arestas. Nesta região n decresce quadraticamente com a distância ao triângulo.

A figura 4.11 apresenta outros comportamentos possíveis para a função Norm(x), conforme o efeito que se deseje.

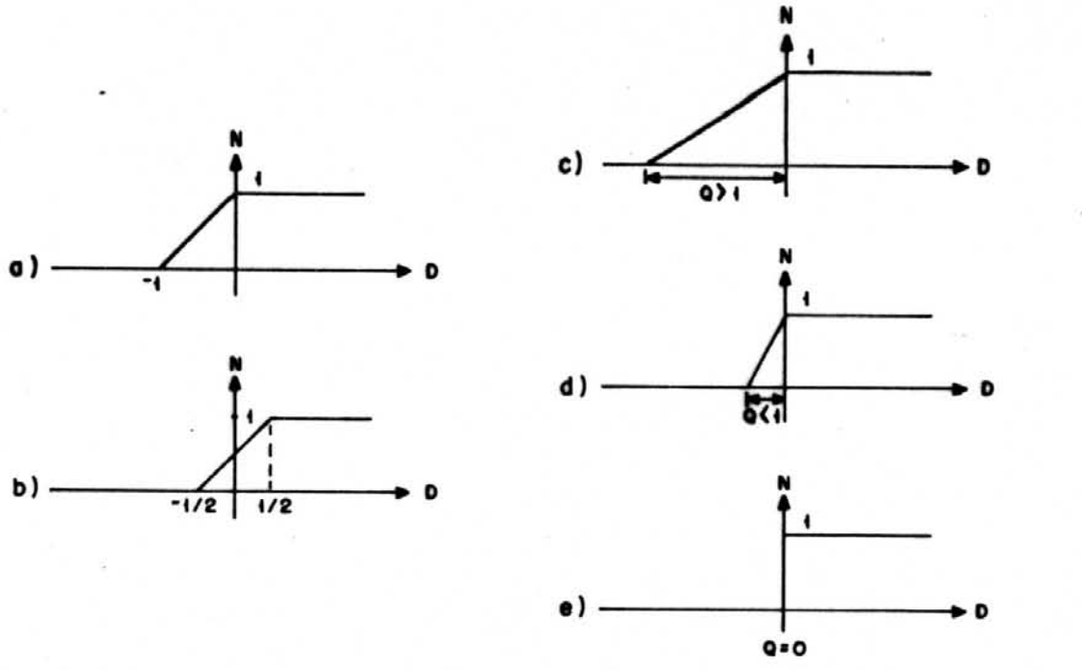


Fig. 4.11 - Comportamentos para a função Norm(x)

A figura 4.11 a) mostra o comportamento de n aqui proposto. Entretanto, pode-se desejar um comportamento diferente, como nos gráficos seguintes. Em b) temos algo semelhante a a) com a linha média do antialiasing coincidindo com a aresta do triângulo. Essa forma é desejável porque o tamanho aparente dos polígonos é o real enquanto que em a) o aspecto dos polígonos é de $1/2$ pixel maior em todas as direções. O único problema com o comportamento em b) é que se formarmos polígonos ou imagens compostos por triângulos justapostos, as linhas de contacto entre estes triângulos ficarão

visíveis por uma pequena interferência com a cor de trás dos polígonos pois nesta linha de contacto o n resultante de dois triângulos justapostos é 0,75 e não 1, como era de se esperar. A figura 4.12 mostra a interferência entre triângulos justapostos conforme os comportamentos b) e a) respectivamente. Aí vemos a descontinuidade da resultante de n.

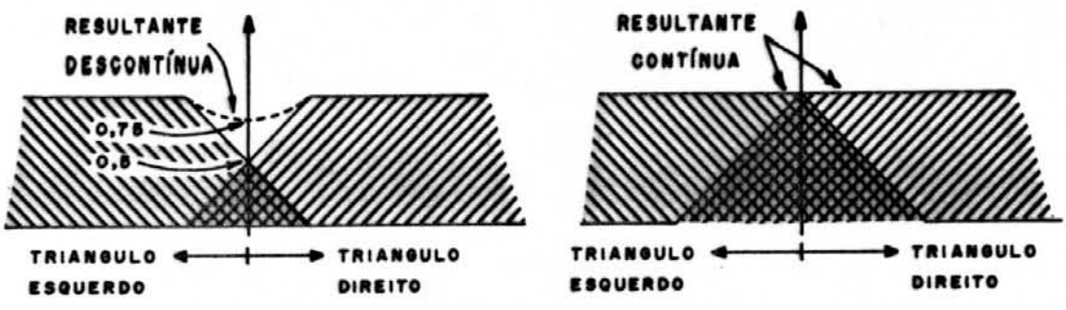


Fig. 4.12 - Intersecções de faces com antialiasing

Em função disso optou-se por garantir n=1 na intersecção, usando o modelo de comportamento a), mesmo com figuras aumentadas de $^1/_2$ pixel, o que na prática só é percebido em detalhes muito pequenos da imagem.

Na figura 4.11 c) tem-se uma outra possibilidade que seria o alargamento da margem de atuação do antialiasing (fator Q>1 pixel, em contraste com Q=1 em a) e b)) dando uma impressão de imagem mais desfocada. Isso equivale a diminuir a resposta de freqüência dessa pré-filtragem.

Em d), em oposição a c), temos a redução do efeito antialiasing (0<Q<1) e em e) a eliminação total do anti-aliasing (Q=0).

Para se obter os outros comportamentos não é necessário alterar-se o DDA. Pode-se obtê-os alterando-se as equações lineares que expressam as distâncias D_1, D_2 e D_3 da seguinte forma:

1) - para um comportamento como em b) deve-se subtrair 0,5 das três equações de distâncias;

2) - para um Q diferente de 1 deve-se dividir todos os coeficientes das equações de distâncias por Q.

4.2.5.1 - A implementação da função Norm(x)

A figura 4.13 apresenta um circuito que transforma D em Norm(D) e as alterações necessárias para gerar os sinais requeridos pelo seqüenciador do DDA para identificar as fronteiras dos triângulos.

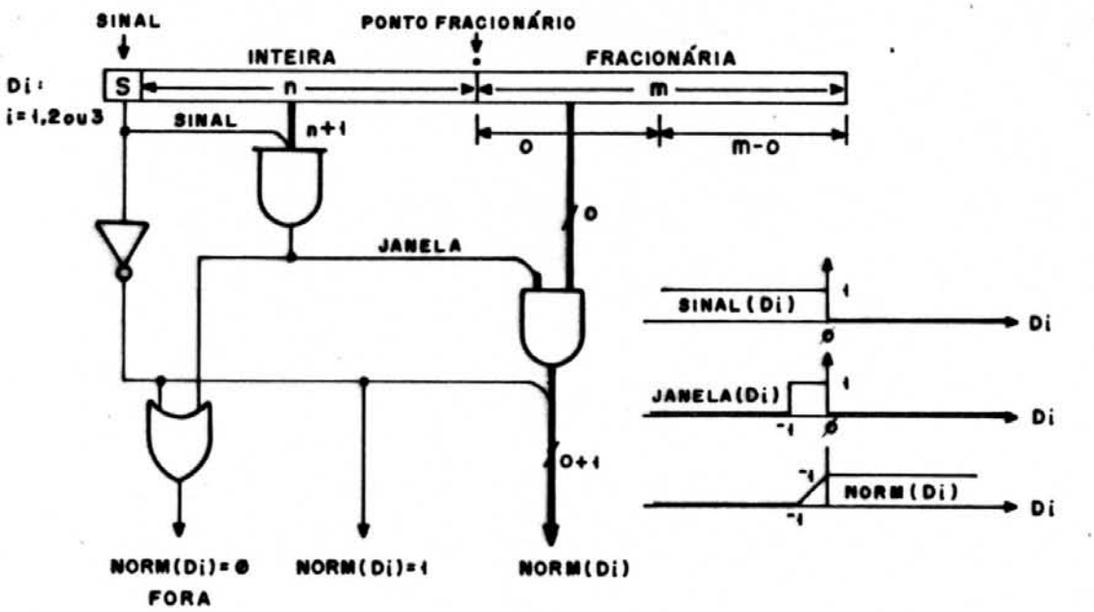


Fig. 4.13 - Circuito da função Norm(D)

A saída "Norm(D_i):0" agora é a identificação de pixel fora do triângulo. As três saídas "Norm(D_i):0" e "n_i" são combinadas conforme a figura 4.14.

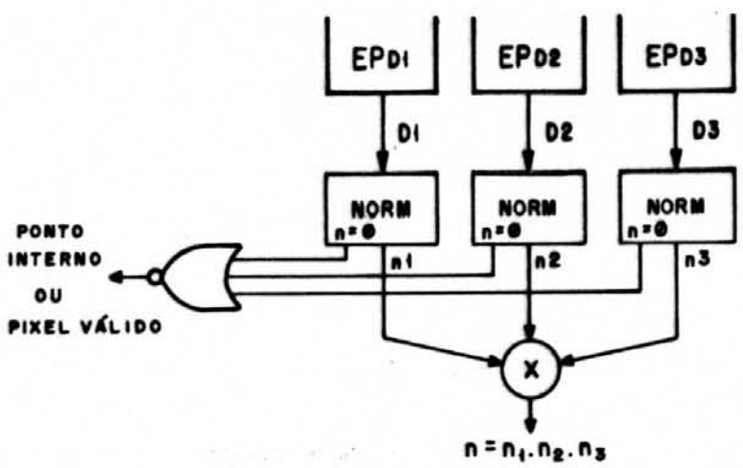


Fig. 4.14 - Geração de "n" e de "pixel interno"

A função Norm é então aplicada às três distâncias geradas pelo DDA obtendo-se n_1, n_2 e n_3, que serão multiplicados entre si para produzir "n".

Quando n=1 os valores R, G e B calculados pelo DDA passarão direto para a memória de vídeo, caso este pixel tenha passado pelo teste Z-buffer. Com n<1 um ciclo adicional de leitura da memória de vídeo é necessário para a operação da média ponderada. Portanto o preenchimento de pixels com n<1 provavelmente será mais lento.

4.2.5.2 - A função de X_{min}, X_{max} e Y_{max} no antialiasing

Em um vértice formado por um ângulo pequeno poderíamos ter um problema de alongamento deste para fora do triângulo devido à extensão da área de intersecção das

margens de antialiasing das duas arestas deste vértice, como mostra a figura 4.15.

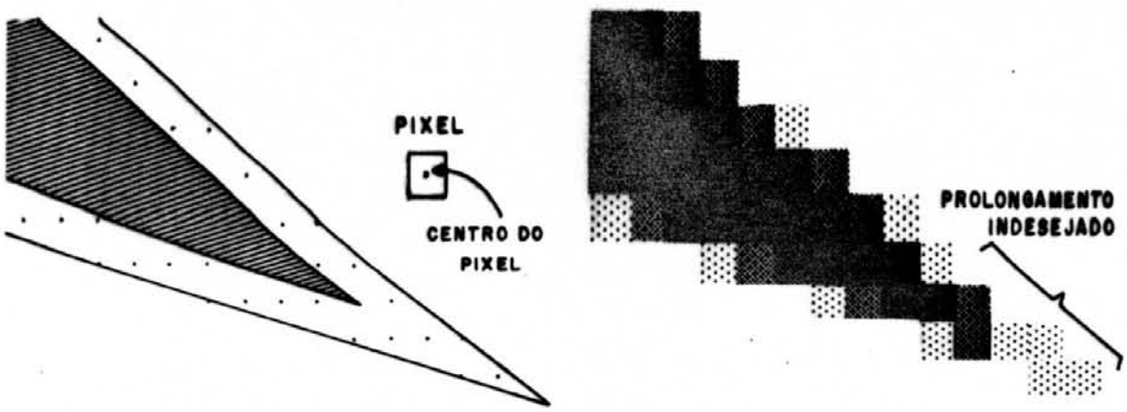


Fig. 4.15 - Efeito do alongamento dos vértices agudos

O controle de janela efetuado pelo gerador de pixels (ULAs), através da verificação dos limites de preenchimento de pixels X_{min}, X_{max} e Y_{max}, evita esse efeito da seguinte forma: esses limites, em conjunto com Yi (ou Ymin) correspondem aos valores mínimo e máximo encontrados entre as coordenadas dos vértices do triângulo, sendo determinados pelo estágio anterior, o preparador do DDA. Estes limites então representam o menor retângulo possível, ou janela, com lados horizontais e verticais, que contenha o triângulo em questão. Conseqüentemente pode-se afirmar que qualquer vértice cujo ângulo é menor que 90° toca as bordas dessa janela e o alongamento do antialiasing desses vértices será recortado por essa janela.

4.2.6 - Transparência

A transparência das faces pode ser obtida permitindo que o "n" seja menor do que 1 ao longo do restante dos pixels dos triângulos, como ocorre com o anti-aliasing nas bordas dos triângulos. Assim o preenchimento de um triângulo manterá uma dose controlada da imagem contida na tela anteriormente.

A transparência é o fator T, que irá atuar também sobre o efeito de n. Este fator também pode ser gerado incrementalmente para uma transparência suavizada, necessitando portanto de um EP adicional para o seu cálculo. Assim como R, G e B podem variar, ao longo de uma superfície curva, o mesmo pode ocorrer para a transparência, de acordo com os ângulos que os feixes de luz atravessam os corpos.

Convém salientar que com esse recurso podemos simular superfícies transparentes e não sólidos transparentes, o que implicaria também em considerar a refração do material e a transparência dependente também da espessura do material, o que só pode ser obtido por um sistema ray-tracing. De qualquer maneira existem muitas aplicações para esse tipo de transparência, especialmente de objetos ocos e transparentes como copos ou jarros delgados.

É importante salientar também que a geração de imagens com transparência exige que os triângulos sejam previamente ordenados, em ordem de profundidade decrescente, para uma correta formação da imagem. Portanto para um correto funcionamento da transparência, inclusive do próprio antialiasing quando este for usado, devemos dispor de um estágio a mais no pipeline, responsável pela ordenação em profundidade dos triângulos.

Como T também será gerado incrementalmente necessitamos mais um EP, com os mesmos quatro registradores e a ULA, para processar a equação $T = T_0 + T_x' . x + T_y' . y$.

T também irá atuar, então, junto a n para formar a transparência final $n_f = T.n = T.n_1.n_2.n_3$.

4.2.7 - Implementação do cálculo da transparência final

Como vimos, para o cálculo de n_f, necessitamos de pelo menos três multiplicações por pixel. Podemos adiantar que todos os dados envolvidos nestas operações não precisam ser representados por mais de 8 bits, e portanto o cálculo pode ser feito usando-se multiplicação em hardware com circuitos comerciais de 8x8 bits, que são bastante comuns. A disposição destes, para o cálculo do n_f, é mostrada na figura 4.16.

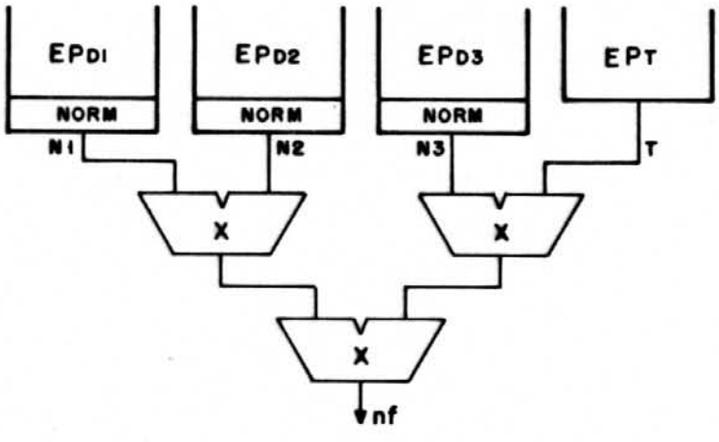


Fig. 4.16 - Disposição dos multiplicadores para a obtenção de nf

Exemplos desses circuitos comerciais são o 74S557, com um tempo máximo de multiplicação de 60ns, e o ADSP 1080A cujo atraso máximo é de 33ns. Essas operações de

multiplicação devem acompanhar o ritmo de produção de pixels do restante do DDA. O circuito multiplicador mais rápido possui um atraso de 33ns. Se dispusermos os multiplicadores como na figura 4.16 teremos um atraso de 66ns, que está acima do ciclo das ULAs, que pretendemos que seja menor do que os 50ns/pixel, com uma folga para compensar uma certa percentagem de ciclos que não produzem pixels válidos.

Pode-se dispor esses multiplicadores na forma de pipeline, colocando-se registradores entre eles, sincronizados com o clock. Assim o atraso continua o mesmo mas a velocidade pode ser dobrada para um resultado de multiplicação a cada 40ns (considerando um tempo adicional de 7ns para os registradores, baseado no registrador 74AS374).

A disposição dos multiplicadores em forma de pipeline exige que se coloquem registradores também nas outras saídas dos EPs para que estas saiam em paralelo com nf, representando o mesmo pixel. O circuito fica como na figura 4.17.

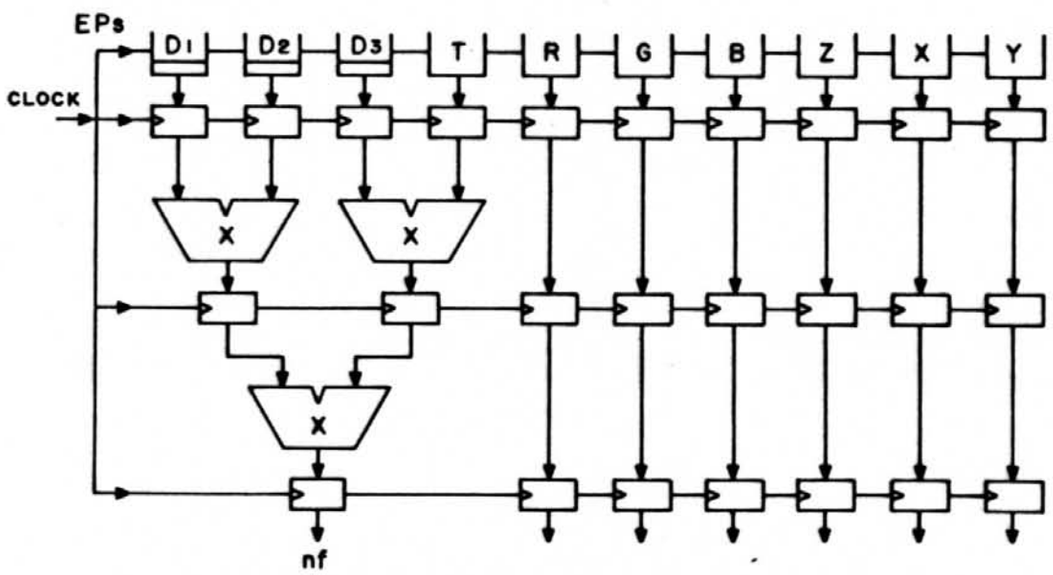


Fig. 4.17 - Inserção dos registradores pipeline

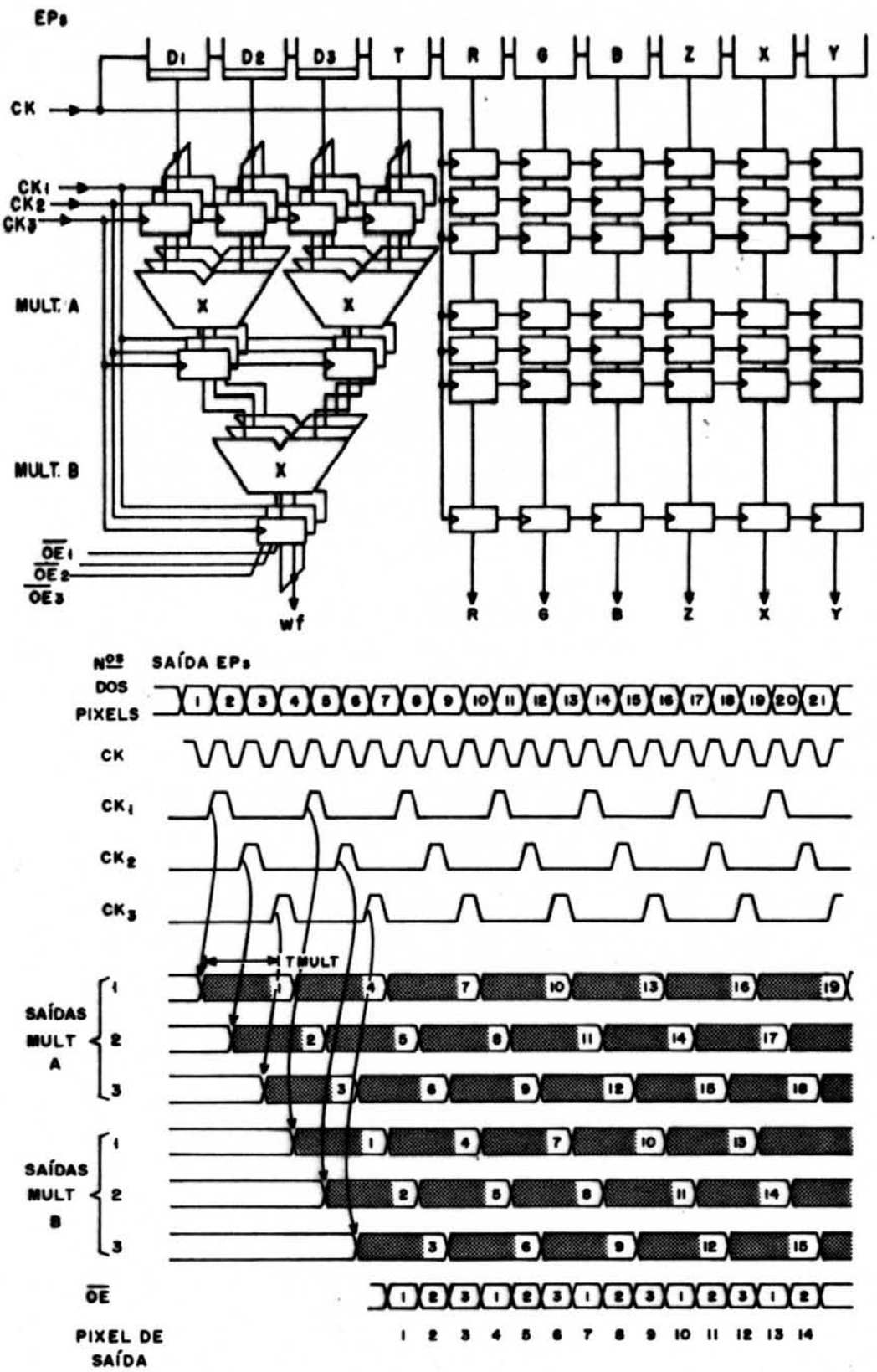


Fig. 4.18 - Paralelização dos multiplicadores

Se a velocidade de geração de pixels ainda estiver limitada pelo circuito de multiplicação poderemos dividir por um fator inteiro o tempo de cada multiplicação repetindo-se esse circuito tantas vezes quanto este fator em paralelo. A figura 4.18 mostra um esboço dessa idéia e o diagrama de tempos de clock para um fator igual a 3.

Esse circuito de multiplicação agora possui 9 multiplicadores e 63 registradores de tamanho variável mas é capaz de entregar resultados a cada 20 ns (33ns/3 + 9ns dos registradores). Essa solução é bastante complicada e volumosa além de exigir sinais de controle muito complexos.

Uma observação mais apurada do algoritmo permite considerações que podem trazer uma grande simplificação nessa tarefa de multiplicação sem perdas significativas de desempenho. Para simplicidade de explanação vamos estabelecer uma variável U que representa a quantidade de dados entre n_1, n_2, n_3 e T que sejam diferentes de 1. A tabela a seguir mostra U em função da situação de um determinado pixel:

	sem transparência	com transparência
a) interior do triângulo sem antialiasing	U = 0	U = 1
b) borda do triângulo com antialiasing	U = 1	U = 2
c) vértice do triângulo antialiasing duplo	U = 2	U = 3

U = 0 ocorre na grande maioria das atualizações de pixel, que são internas aos triângulos e sem antialiasing e

nem transparência. Neste caso o produto n_f não precisa ser calculado pois sabe-se que é igual a 1 ($n_1 = n_2 = n_3 = T = 1$).

U = 1 é também bastante freqüente. Ocorre ou quando se usa transparência, no interior dos triângulos, ou no antialiasing normal, sem transparência. Neste caso o produto n_f também não precisa ser calculado pois somente um dado é diferente de 1, podendo-se obter n_f pela adoção deste valor.

U = 2 ocorre com muito pouca freqüência. Somente ocorre nos poucos pixels perto de um vértice (antialiasing duplo) ou no antialiasing simples de triângulos transparentes, mas a multiplicação deve ser feita. Entretanto se, nesses casos, sacrificarmos a precisão dos dados a serem multiplicados, restringindo para 5 bits por exemplo, não teremos perdas visiveis na qualidade da imagem. A vantagem disto é que com argumentos de 5 bits poderemos usar PROMs para a operação de multiplicação, em lugar ds multiplicadores comerciais. Os dois argumentos de 5 bits entrando como as 10 linhas de endereço de uma PROM, dispararão um produto de 8 bits na saida de dados, já gravado previamente na PROM. Existem PROMs com registradores internos capazes de responder a clocks de 20ns que satisfazem plenamente ao ritmo do DDA. Alguns exemplos dessas PROMs são as fornecidas pelas TEXAS, para uso como memórias de microseqüenciadores rápidos:

512x8 - TPB28R45 e TPB28R46
1Kx8 - TBP28R85A e TPB28R46A
2kx8 - TPB28R165A e TPB28R166A

Esses dados que se quer multiplicar são números reais entre 0 e 1, inclusive. Para isso há a necessidade de pelo menos 1 bit para a representação da parte inteira e o

restante para a parte fracionária. Em 8 bits estes dados teriam 1 bit para a parte inteira e 7 para a fracionária, não necessitando de bit de sinal pois os dados são todos positivos. Se usarmos 5 bits 1 é para a parte inteira e 4 para a fracionária. A representação desses dados em 5 bits é como segue:

1 decimal = 1.0000 binário

0 decimal = 0.0000 binário

0 < 0.XXXX binário < 1

1.XXXX só é válido se XXXX=0000

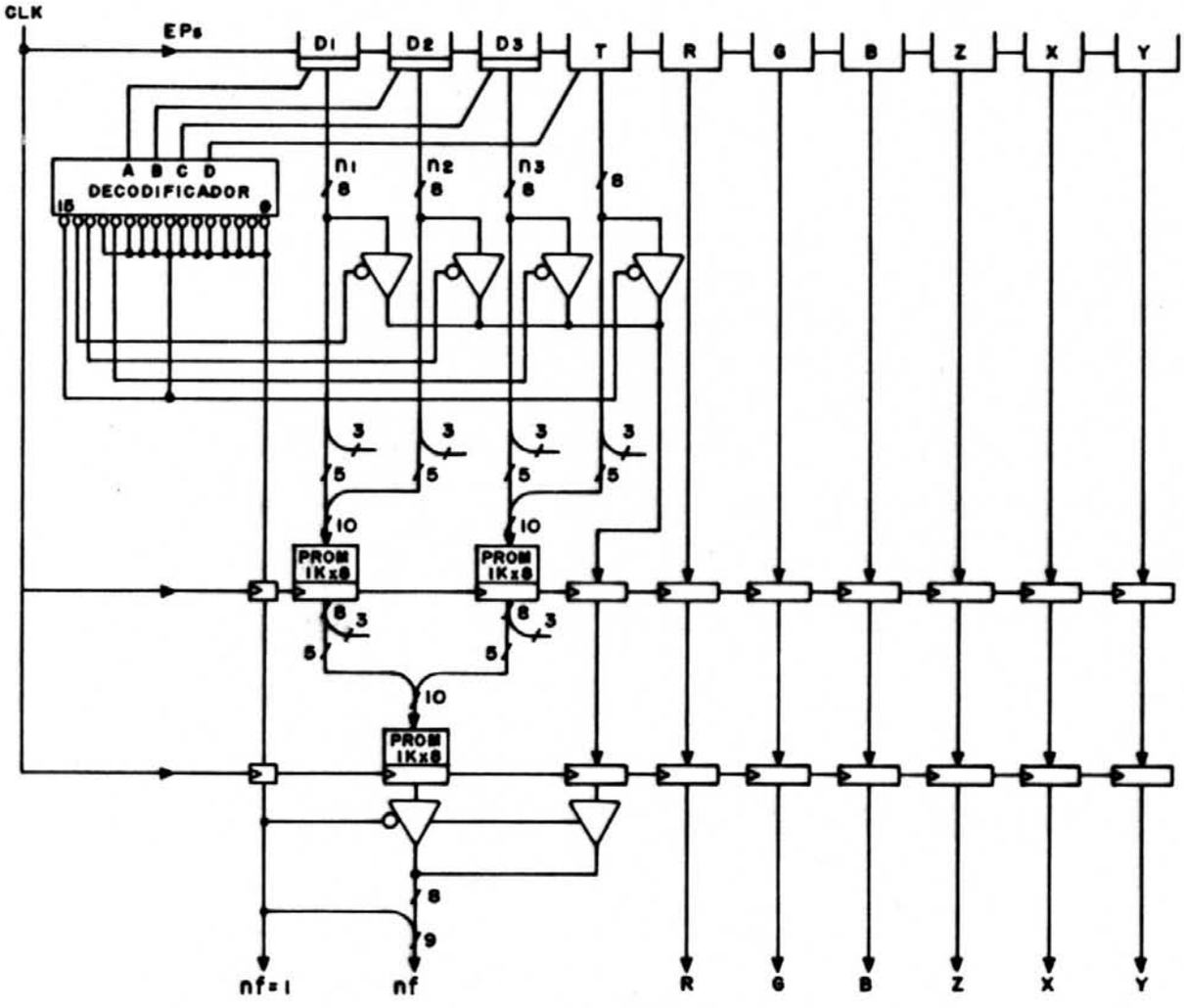


Fig. 4.19 - Circuitos dos multiplicadores com PROMs

O dado pode então admitir 17 valores diferentes entre 0 e 1, com uma precisão de 0,0625.

O circuito de multiplicação com o uso de PROMs é mostrado na figura 4.19. Note-se que estas PROMs já incorporam um registrador interno, dispensando registradores pipeline externos.

O valor U é avaliado por um decodificador 4:16. As quatro linhas de entrada deste decodificador são os sinais "N=1" provenientes dos quatro operandos. Em função desses quatro sinais o decodificador decide o encaminhamento dos dados para a obtenção ou cálculo de n_f, ativando ou não as PROMs, conforme a seguinte tabela:

	$n_1=1$	$n_2=1$	$n_3=1$	T=1	encaminhamento
(0)	0	0	0	0	n_f =mult
(1)	0	0	0	1	n_f =mult
(2)	0	0	1	0	n_f =mult
(3)	0	0	1	1	n_f =mult
(4)	0	1	0	0	n_f =mult
(5)	0	1	0	1	n_f =mult
(6)	0	1	1	0	n_f =mult
(7)	0	1	1	1	$n_f = n_1$
(8)	1	0	0	0	n_f =mult
(9)	1	0	0	1	n_f =mult
(10)	1	0	1	0	n_f =mult
(11)	1	0	1	1	$n_f = n_2$
(12)	1	1	0	0	n_f =mult
(13)	1	1	0	1	$n_f = n_3$
(14)	1	1	1	0	n_f =T
(15)	1	1	1	1	n_f =1=T

4.2.8 - Considerações sobre a precisão interna dos EPs

Os valores calculados pelos EPs devem ter uma precisão (nº de bits) suficiente para que não se propague erros significativos ao longo das iterações de soma e subtração com as derivadas. As saídas R, G e B, por exemplo, são números inteiros sem sinal de 8 bits. Entretanto, para a sua correta geração, um certo número de bits fracionários são necessários, em função do método de calculo adotado. A dúvida é: "quantos bits serão necessários?". Quanto mais bits, menor será o erro propagado. Um erro propagado igual a 1.0 é aceitável e imperceptível em uma escala de 0 a 255, como ocorre em R, G, B e T. O erro máximo propagável em cada iteração de soma ou subtração é a metade da precisão, se usarmos arredondamento simétrico nas derivadas. Numa tela de 1024x1024, que é o nosso caso, o número máximo de iterações em uma linha horizontal é de 1024.

Cabe aqui observar que o erro máximo propagado não depende do número de pixels do triângulo e sim de suas dimensões, isso porque o erro gerado por uma soma é o mesmo, com sinal oposto, ao de uma subtração; assim o erro propagado em uma série de somas no sentido positivo da direção X, por exemplo, é compensado quando se anda em sentido contrário, com subtrações sucessivas. Para efeitos de propagação de erro podemos considerar então como 1024 o número máximo de iterações, embora raramente os triângulos atinjam essas dimensões.

Portanto, para que em 1024 iterações seja propagado um erro máximo igual a 1, teremos que ter uma resolução de $2x(1,0)/1024 = 2^{-9}$, ou seja, precisaremos de pelo menos 9 bits fracionários. No caso dos EPs R, G, B e T teremos então 8 (inteiro) + 9 (fracionário) = 17 bits ao todo. Acrescentamos mais 3 bits na porção inteira para

evitar a passagem de 255 para 0, em caso de overflow, e de 0 para 255, em caso de underflow, resultando um total de 20 bits.

Há situações em que o DDA pode gerar overflow ou underflow de 8 bits no cálculo do antialiasing nas bordas dos triângulos. Com a adição de mais 3 bits, com o mais significativo podendo ser o de sinal, os dados R, G, B e T, internamente aos EPs, poderão variar entre -1024 e +1024 enquanto que na saída estes deverão ser limitados entre 0 e 255 por saturação, controlando assim as situações de underflow e overflow. Um circuito para desempenhar essa função de saturação é mostrado na figura 4.20. Este pode ser colocado às saídas dos EPs R, G, B e T.

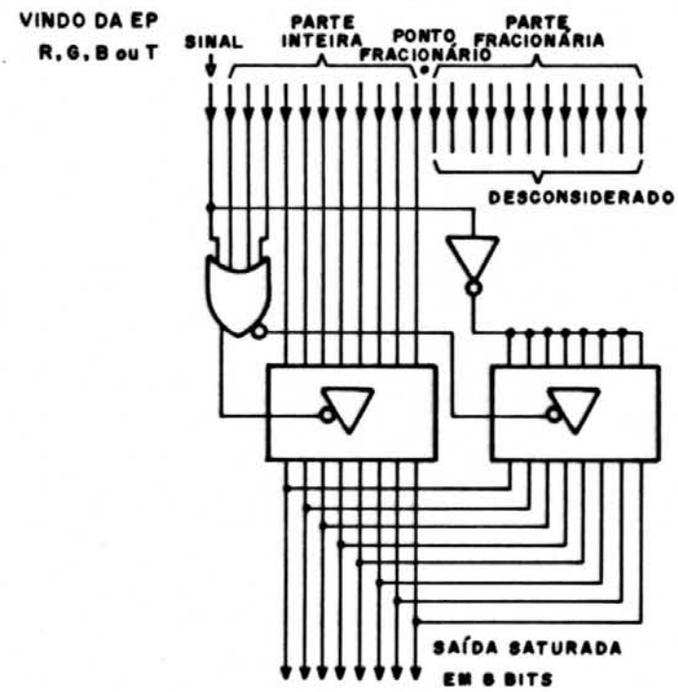


Fig. 4.20 - Circuito de saturação

A respeito da precisão interna dos EPs D1, D2 e D3, sabemos que em antialiasing normal (Q=1), seus resultados têm uma variação igual a 1,0 para cada afastamento equivalente à dimensão de 1 pixel na direção ortogonal à sua respectiva aresta. Um erro de uma unidade em D1, D2 e D3 corresponde a um deslocamento de até 1 pixel na determinação das bordas dos triângulos. Esse desvio de 1 pixel, em um triângulo isolado, é imperceptível, mas em triângulos que juntos formam uma superfície sombreada, a linha de contacto entre esses triângulos poderá se tornar visível caso eles fiquem afastados de um pixel. Com o afastamento, entra em ação o antialiasing neste espaço vazio.

Ao contrário de R, G e B, um erro de uma unidade em D1, D2 ou D3 é intolerável, ou seja, os 9 bits fracionários não são suficientes. Por outro lado temos que ter o cuidado de não usarmos palavras muito longas nos EPs pois isto diminui a velocidade de suas ULAs e estas se tornarão mais lentas que as outras.

Um bom compromisso seria o de acrescentar 3 bits à parte fracionária, reduzindo o erro máximo em 1024 iterações para 0,125. Com esse erro o menor fator T de transparência gerada pelo antialiasing, no pior caso de afastamento errôneo das bordas de triângulos adjacentes, será de $1-(0,125/2)^2 = 0,996$ ou 0,4% de transparência, que é equivalente ao erro de uma unidade em R, G e B (1/256). A figura 4.21 demonstra graficamente a obtenção deste valor de erro a partir das funções Norm(D) das duas arestas dos triângulos em contacto por um afastamento de 0,125. Assim o número de bits da parte fracionária de D1, D2 e D3 deverá conter 12 bits.

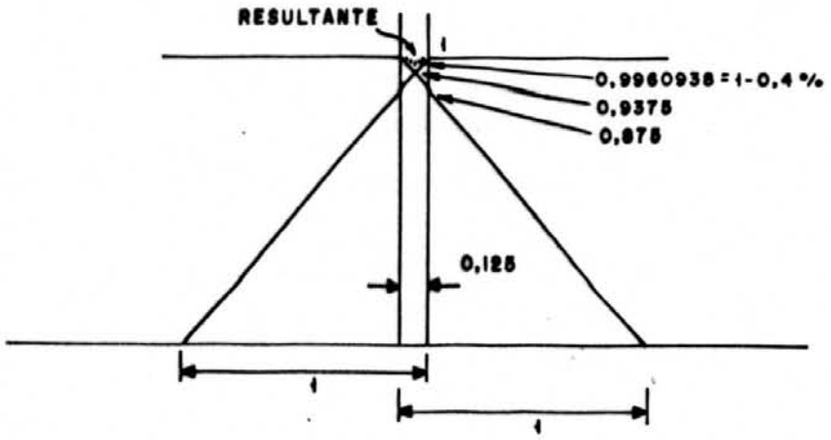


Fig. 4.21 - Intersecção dos antialiasing

Foi dito que esses dados variam de 1 unidade para cada afastamento da ordem de 1 pixel, e portanto em uma tela inteira podem variar no máximo o equivalente à diagonal da tela, que é 1024 x $\sqrt{2}$ ≈ 1448, que necessita de 11 bits para ser representado. Acrescentando-se mais um bit de sinal, para que os limites de representação dos triângulos se afastem um pouco das bordas da tela, temos então 12 bits. Somando-se os 12 fracionários temos um total de 24 bits para a representação interna dos dados nos EPs das distâncias.

As saídas de endereço X e Y são inteiros sem sinal de 10 bits cada, para a representação das coordenadas de tela que estão entre 0 e 1023. Estes bits justapostos formarão 20 bits de endereço que acessarão 1Mpixel de memória de vídeo. X e Y não precisam ter representação fracionária pois variam sempre em passos unitários. Estas coordenadas porém poderiam ter alguns bits adicionais na parte mais significativa para permitir, dentro do DDA, a representação de triângulos fora da tela. Acrescentamos pois 6 bits para completar o número 16, que é convencional em circuitos de incremento ou decremento.

Os EPs X e Y, fazendo cálculos somente de incremento e decremento em apenas 16 bits, são mais rápidos

que os outros EPs mas, considerando que a cada ciclo deve-se dispor da comparação entre esses valores e os limites X_{min}, X_{max} e Y_{max}, o restante do tempo é consumido por essas comparações.

O EP Z, por sua vez, gera a profundidade do pixel com 16 bits de precisão. Para a sua representação interna não há a necessidade de bits adicionais na porção mais significativa pois as profundidades dos vértices dos triângulos são calculadas para que a maior profundidade seja representada por 65535 e a menor por 0. A remoção de partes ocultas com Z-buffer de 16 bits é usual e tem demonstrado ser plenamente satisfatória. Bits fracionários também não são necessários mas colocaremos 8 para igualar-se com os EPs das distâncias e dar uma precisão adicional, ou seja 24 bits ao todo.

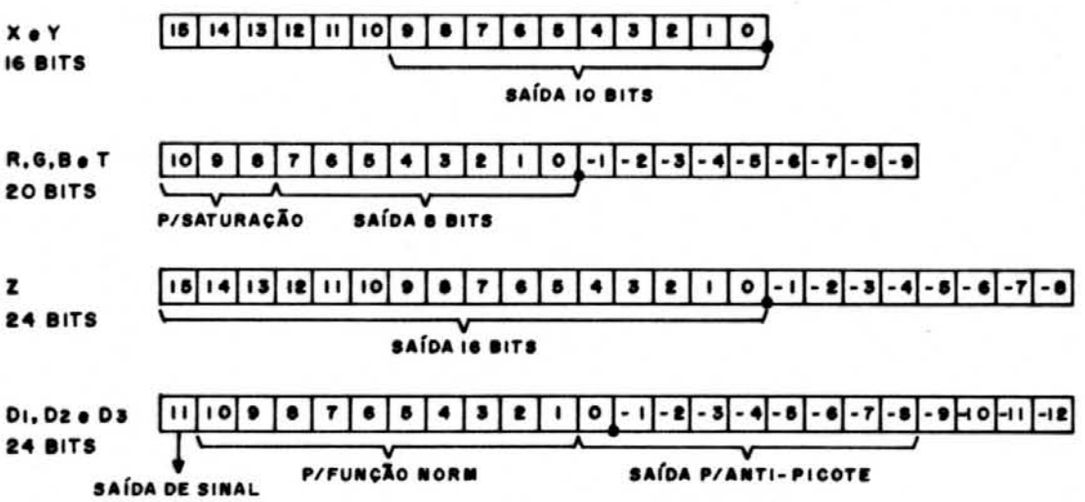


Fig. 4.22 - Organização interna dos registradores dos EPs

A posição do ponto fracionário nos registradores é mera convenção e passa desapercebida pelas ULAs, que só

enxergam números inteiros. O ponto fracionário fixo só é considerado na saída dos EPs, para saber aonde truncar os dados, e no estágio anterior, onde os números foram convertidos de ponto flutuante para ponto fixo.

A organização interna dos registradores dos EPs fica resumida como na figura 4.22.

4.2.9 - O algoritmo DDA na forma de máquina de estados

Conhecendo-se agora o bloco operacional do DDA, onde realizar-se-á a geração de pixels e seus atributos, podemos imaginar o bloco de controle do DDA, representado pelo seu diagrama de estados, para nos fornecer uma noção mais precisa de seu desempenho. O diagrama de estados é apresentado na figura 4.23.

Alguns comandos do algoritmo original foram colocados em execução paralela, em um mesmo estado, enquanto outros tiveram que ser colocados em separado, consumindo tempo morto, devido às limitações do nosso bloco operacional. O importante é observar quais estados efetivamente geram pixels, que são aqueles marcados em negrito na figura 4.23, e determinar-se qual a proporção de ocorrência desses em relação a todos os estados, ou seja, a eficiência média de geração efetiva de pixels.

E somente nesses estados que o bloco controlador emite pulsos de clock para os estágios pipeline seguintes, a fim de não enfileirar pixels que não devam ser atualizados, ou porque são externos ao triângulo ou porque pertencem a estados do DDA que não correspondem a novos pixels.

Consequentemente a taxa média de produção de pixels é menor que a taxa de clock do DDA.

Esta taxa depende de diversos fatores. O principal é o tamanho do triângulo. Quanto maiores os triângulos menos são levados em conta os estados não efetivos porque maior é a quantidade de pixels dentro dos laços de pixels válidos. O número de estados responsáveis por pixels válidos cresce quadraticamente com a dimensão do triângulo enquanto que os

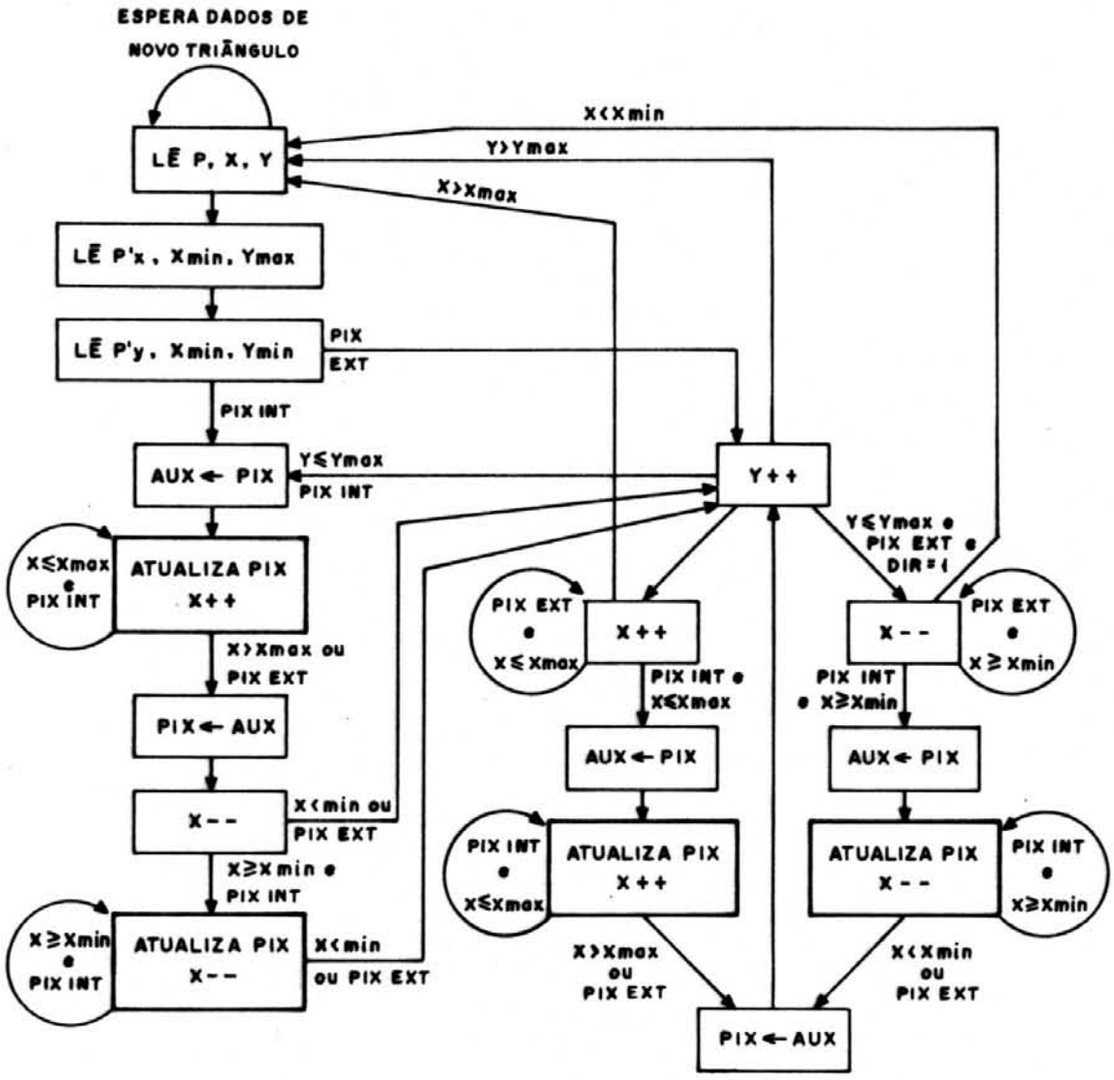


Fig. 4.23 - Diagrama de estados do DDA

restantes crescem somente linearmente, pois são proporcionais ao perimetro do triângulo. Analisando-se o diagrama de estados, podemos estabelecer o número de estados que não geram pixels como função das dimensões das arestas dos triângulos da seguinte maneira: cada aresta cujo vetor normal externo aponta para cima contribui com $2.\Delta y$ estados a mais e para arestas com o vetor normal para baixo o número de estados não efetivos é de $\Delta x + 3.\Delta y/2$, onde Δx e Δy são as diferenças entre os vértices das respectivas arestas do triângulo. Como é difícil avaliar esta taxa média entre as aplicações mais comuns, considera-se, para efeitos de cálculo, um triângulo equilátero bastante pequeno, dentro da média de triângulos que compõem uma imagem típica (o que produz uma alta taxa de estados não efetivos no DDA), de 200 pixels (aresta com um comprimento de aproximadamente 21 pixels). Com a análise feita acima, pode-se estimar que o DDA perde por volta de 86 estados para exibir seus 200 pixels, ou seja, uma eficiência de geração de pixels de 1 - 86/(200 + 86) = 70%, que adotaremos como média.

Com esse dado podemos esperar que, para atingirmos uma taxa média de 50ns por pixel, devemos ter um clock com período 70 % de 50ns, ou seja 35ns.

4.2.10 - Estimativa prática dos EPs do DDA

Aqui se faz uma avaliação de componentes eletrônicos comerciais para a implementação dos EPs para que possam desempenhar a geração de pixels a um clock de 35ns, velocidade estipulada no item anterior.

Estas ULAs e seus circuitos agregados, tais como registradores, circuitos de saturação, a função Norm e os comparadores em X e Y, são pequenas unidades de

processamento ideais para serem implementadas em micro-eletrônica, ao invés do uso de componentes discretos, pois são circuitos simples e as ULAs só necessitam executar somas ou subtrações inteiras (ponto fixo). Entretanto um projeto em micro-eletrônica foge ao escopo deste trabalho. Por razões práticas avaliaremos o desempenho desses circuitos baseando-nos em componentes discretos comerciais e em tecnologias disponíveis.

Entre os nossos 10 EPs temos praticamente 2 tipos de EPs: aqueles dois para gerar X e Y, cujas ULAs são representadas pelos circuitos de incremento e decremento e os circuitos de comparação, e os outros oito restantes, em que as ULAs calculam somas e subtrações.

As ULAs do primeiro tipo podem ser compostas de registradores com auto-incremento e auto decremento, tais como os conhecidos contadores up-down. Para se obter um contador desses de 16 bits usa-se 4 contadores de 4 bits interligados através de um LACG (Look Ahead Carry Generator) para melhora na velocidade. O contador up-down TTL de 4 bits mais rápido é o 74AS169. Quatro desses mais um LACG 74AS264 podem contar 16 bits a uma freqüência de 30MHz, ou 33ns por contagem. Agora levando em conta que os comparadores devem operar dentro do mesmo ciclo, usando por exemplo 4 comparadores 74F85 em cascata, um atraso adicional de 28ns ocorre. O atraso total de 61ns inviabiliza esta solução por passar dos 35ns máximos.

Poderemos dispor estas duas operações em forma de pipeline, separando do mesmo ciclo as operações de incremento/decremento da operação de comparação. Assim as duas operações não ultrapassam o limite de 35ns mas a informação de ultrapassagem por X_{max}, X_{min} ou Y_{max} é recebida um clock atrasada pelo bloco de controle. Assim o

DDA perde 2 clocks a mais para cada linha horizontal do triângulo.

As ULAs dos EPs do segundo tipo devem efetuar somas e subtrações de até 24 bits. Utilizando-se 6 ULAs do tipo 74AS881 e 2 LACG 74AS882 conseguimos um atraso típico de 19ns e máximo de 36ns. Somando-se os atrasos dos registradores, 9ns para o 74AS374 (8 bits), os tempos começam a se tornar críticos novamente.

Podemos usar esses componentes, com um desempenho 20% abaixo do desejado, ou empregar lógica ECL. Exemplos desses componentes em versão ECL e seus atrasos típicos são:

100166 comparador 9 bits cascateável 3ns
100181 ULA 4 bits 18 funções 10ns
100179 LACG 4ns

Com o uso desses componentes ECL pode-se atingir com folga os ciclos desejados de 35ns, o que equivale a uma taxa mínima de 20Mpixels/s (triangulos de 200 pixels) e máxima teórica de 28Mpixels/s (100% de estados efetivos).

4.2.11 - Hardware de Atualizacão de Display - HAD

4.2.11.1 - Considerações iniciais

Até aqui temos especificado o bloco operacional do DDA construído em uma estrutura pipeline capaz de gerar, a uma velocidade de 28MHz, pixels candidatos a serem escritos na memória de vídeo. Cada pixel é representado por, além de suas coordenadas, seus atributos de cor R, G, B, e transparência nf e a profundidade Z, que definirá se o pixel

deve ou não ser atualizado na memória. Estes dados são capturados pelo circuito de atualização de memória de vídeo, o HAD, para preenchimento da tela. Ao HAD cabe decidir se o pixel recebido pelo DDA deve ou não ser escrito na memória de vídeo, conforme a comparação de seu Z com o Zb, lido da memória de vídeo no mesmo endereço (X,Y). Também cabe a este calcular os novos R, G e B, caso a transparência nf seja diferente de 1, forçando a obtenção da média ponderada com os R, G e B lidos da memória. Outras funções básicas do HAD são o refresh da memória de vídeo e a varredura para a geração dos sinais de vídeo.

Se o pixel passar no teste de profundidade, um novo R-G-B deverá ser calculado, conforme nf. Se nf for igual a 1 o R-G-B emitido pelo DDA será transferido imediatamente para a memória. E se nf for menor que 1 a média ponderada deverá ser calculada. Um ciclo adicional de leitura da memória de vídeo se faz necessário para obter os R-G-B já presentes no pixel, que entrarão no cálculo desta média ponderada. Esta média é obtida pelas operações:

$$Rb = R.nf + Rb.(1-nf)$$
$$Gb = G.nf + Gb.(1-nf)$$
$$Bb = B.nf + Bb.(1-nf)$$

Estas operações são mais demoradas do que a transferência direta de R, G e B para a memória de vídeo, além de necessitar de um ciclo adicional de leitura.

As memórias de vídeo disponíveis no comércio possuem, no melhor caso, um tempo de acesso de 200ns o que é incompatível com a velocidade do DDA. Disso surge a necessidade da intercalação de uma memória cache antes da memória de vídeo. O acesso à cache pode ser feito na velocidade do DDA havendo entretanto a necessidade, de

tempos em tempos, de uma espera para troca da página da memória cache. Nesse instante, toda o conteúdo da memória cache é transferido para a memória de vídeo e em seguida a nova página é transferida da memória de vídeo para a memória cache.

4.2.11.2 - Uso de uma FIFO entre o DDA e o HAD

As considerações acima evidenciam um comportamento assíncrono entre o HAD e DDA. O HAD terá momentos de espera para a troca de página e ciclos maiores quando nf<1. O DDA por sua vez nem sempre emitirá pulsos de clock que representam pixels válidos e portanto a emissão de pixels pode ser fragmentada. Para que o HAD não retarde o DDA durante suas poucas operações mais demoradas e nem que o DDA faça o HAD esperar o fim de seus estados sem pixels, uma FIFO faz-se necessária entre esses dois processadores. Com isso diminui-se bastante a dependência temporal entre esses e promove-se um fluxo mais equilibrado de dados entre o DDA e HAD. A taxa média de pixels desses dois processadores deve ser semelhante para uma utilização ótima da FIFO.

Uma FIFO de 16 posições, bastante comum comercialmente, é suficiente pois os tempos de espera de cada estágio nunca são maiores do que 16 ciclos do DDA. Por segurança um fraco sincronismo ainda deve ser assegurado nesta construção, através dos sinais de FIFO-cheia e de FIFO-vazia. O sinal de escrita (WR) da FIFO é sincronizado com o clock do DDA, enquanto que o DDA pode ficar em estado de espera pelo sinal de FIFO-cheia. Enquanto isso o HAD lê dados da FIFO através do sinal de leitura (RD) e também pode ficar em estado de espera pelo sinal FIFO-vazia. Esta FIFO atuará como um registrador pipeline entre esses dois processadores, com uma profundidade variável de acordo com a necessidade.

A FIFO terá que ter uma largura de 69 bits (R:8 + G:8 + B:8 + nf:9 + X:10 + Y:10 + Z:16). Para isso podemos dispor de 17 FIFOs de 4 bits de largura, como o 74S222, ou de 14 FIFOs de 5 bits de largura, o 74S225. Ambas são comuns e possuem uma profundidade de 16 posições.

O HAD possui um seqüenciador, mais simples que o do DDA, que executa a remoção de pixels ocultos, emite ciclos de acesso à memória cache, ativa a transferência entre a memória cache e a memória de vídeo e executa o refresh de memória e de vídeo.

4.2.11.3 - Circuito de transparência n_f

O HAD mantém atenção ao sinal FIFO-vazia esperando que este indique a presença de dados válidos que então serão lidos. Desses dados X e Y comporão o endereço de memória de vídeo. Essa composição é direta, pela concatenação de X com Y, ficando as linhas de endereço A0-A9 oriundas de X e A10-A19 de Y, perfazendo um total de 20 bits que endereçam 1 Mpixel. Z é comparado com Zb, lido da memória, e se for menor dispara a escrita do pixel. Para isso o seqüenciador avalia se n_f é igual a 1. Neste caso ele permite a transferência direta de R, G e B para a memória cache. Caso contrário é ativado o circuito de transparência, para o cálculo dos novos R, G e B. Considerando que esta operação é mais demorada, o HAD dará um tempo maior para que seja executada. O uso do circuito de transparência é mais demorado mas sua ocorrência é bem mais rara (antialiasing ou transparência), o que praticamente não reduz o desempenho médio. A figura 4.24 mostra um esboço do circuito do HAD.

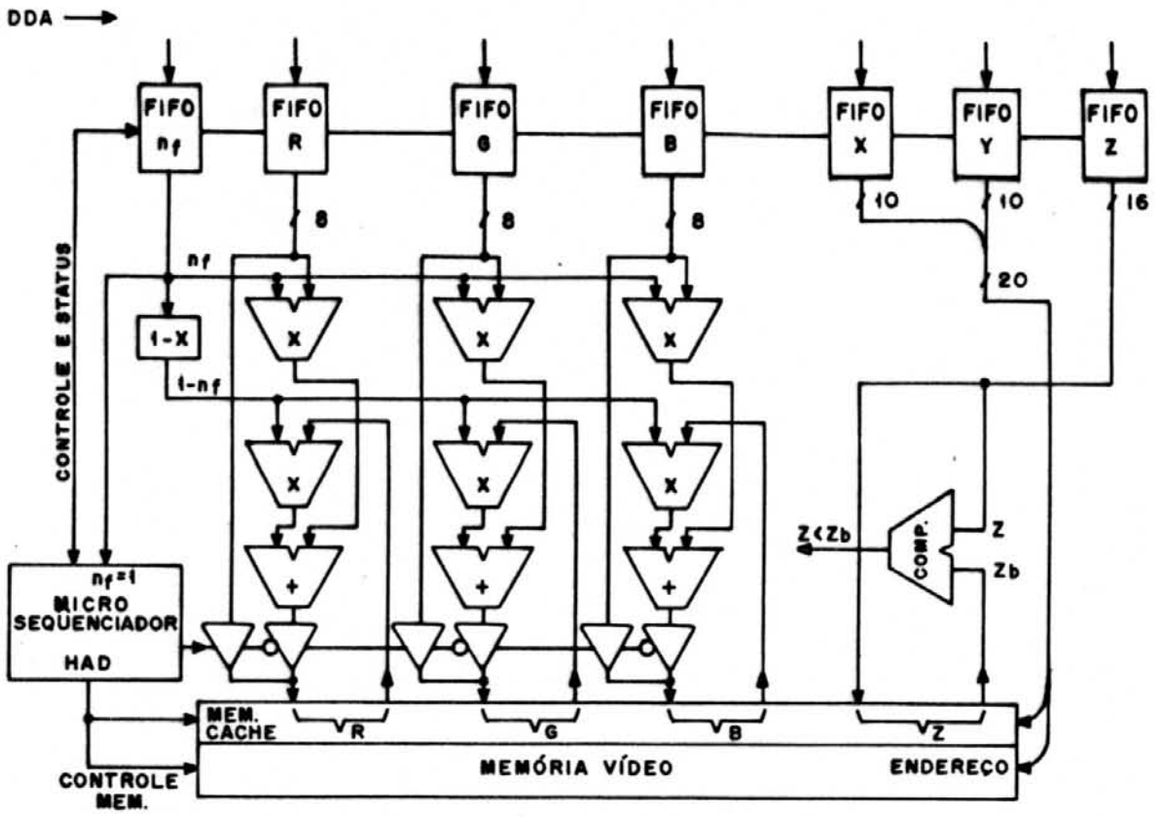


Fig. 4.24 - Circuito do HAD

R, G e B, assim como Rb, Gb e Bb, são inteiros de 8 bits e podem assumir valores de 0 a 255. n_f possui 9 bits, um inteiro e 8 fracionários. Seu bit mais significativo é usado para indicar que $n_f = 1$. A parte fracionária é multiplicada por R, G e B, da mesma forma que $(1-n_f)$ é multiplicado por Rb, Gb e Bb. Os multiplicadores de 8x8 possuem 16 linhas de saída de resultado. Sabe-se que o número de digitos fracionários de uma multiplicação é a soma do número de digitos fracionários dos multiplicandos, portanto temos 8 bits fracionários nos nossos produtos. Então para obtermos somente a porção inteira usa-se somente os 8 bits mais significativos do resultado. A média ponderada é então obtida pela soma desses resultados parciais.

Para a obtenção de (1-nf) deve-se ter um inversor de todos os bits fracionários de nf seguido de um somador com 1, como mostra a seguinte tabela:

nf	$\overline{nf}$	1+nf = (1-nf)
1.00000000	1.11111111	0.00000000
0.11111111	0.00000000	0.00000001
0.11111110	0.00000001	0.00000010
0.11111101	0.00000010	0.00000011
.	.	.
.	.	.
0.00000001	0.11111110	0.11111111
0.00000000	0.11111111	1.00000000

Pode-se dispensar o somador com 1 e usar só os inversores gerando um imperceptível erro de 0,4% na cor do pixel. Não se usando o somador só poderia ocorrer um erro grande quando nf=1 mas neste caso o circuito de transparência não é usado pois o caminho de R, G e B é direto.

A memória cache tem barramentos separados de leitura e escrita de maneira a poder fazer tanto a leitura como a escrita num mesmo ciclo. No caso de haver nf<1 o seqüenciador deve contar alguns ciclos de clock para aguardar o atraso dos multiplicadores e somadores. Esse tempo é tipicamente de 33ns a 60ns para os multiplicadores e de 15 ns para as somas, utilizando-se componentes comerciais de tecnologia Schottki.

4.2.11.4 - Memórias cache e de video

O objetivo é a obtenção de uma imagem de 1024x1024 pixels, ou seja 1 Mpixel. Cada pixel possui 3 bytes para a cor (R, G e B) mais 2 bytes para a profundidade Z dando um total de 5 Mbytes. E desejável também, em equipamentos

voltados à animação, que a memória de vídeo seja duplamente bufferizada, ou seja, pode-se exibir um buffer enquanto outro está sendo preenchido, de maneira que o preenchimento não perturbe a continuidade das cenas. Para isto duplica-se a porção R, G e B da memória (Z não é necessário duplicar) perfazendo um total de 8 Mbytes.

O tempo de leitura e escrita das memórias dinâmicas de vídeo comerciais situa-se por volta de 220ns que será usado como base para nossas análises. Se não for usada uma memória cache os pixels consumirão cada um 440ns o que dará uma taxa máxima de 2.27 Mpixel/s, que está nove vezes abaixo do requerido (20 Mpixel/s).

Se a transferência entre a memória cache e a memória de vídeo for totalmente paralela, ela poderá ser executada em um único ciclo de 220ns ao invés de transferir-se seqüencialmente pixel por pixel da memória cache. Para isto a memória de vídeo tem que ser particionada para permitir um acesso paralelo a varios pixels simultaneamente, tantos quantas forem as posições de memória cache. As memórias de vídeo VRAMs mais difundidas no mercado são as de 64Kx4 bits (tamanhos diferentes são raros nesse tipo de memória de vídeo de dupla porta). Entre elas pode-se citar alguns exemplos e seus fornecedores como o TMS44C65 da TEXAS, Am90C64 da AMD, MB81461 da Fujitsu, MT42C4064 da Micron, μPD41264C da NEC e HM53462 da Hitachi, todas elas com dupla porta organizadas em 64Kx4 bits na porta de acesso randômico e 256x4 bits na porta de acesso serial. 64K é 1/16 de 1M de maneira que para completar 1M serão necessários pelo menos 16 chips para cada plano de memória. Estes 16 chips possuem cada um 16 linhas de endereço (multiplexadas 2x8, como DRAMs convencionais de 64K) faltando 4 linhas para completar um endereçamento de 1M. Estas 4 linhas

selecionarão um dos 16 chips, ou grupos de chips, para o acesso a um único pixel.

Pode-se, ao invés de selecionar um dos 16 grupos, ter acesso direto a todos os 16 grupos de uma só vez, o que significa acessar 16 pixels em um só ciclo de leitura ou escrita à memória de vídeo. Fazendo-se esse acesso paralelo transferir dados entre as memórias e um conjunto de 16 registradores de dupla porta, um para cada pixel, teremos constituída a memória cache de 16 pixels, que seria acessada através da segunda porta desses registradores a uma velocidade bem maior que aquela conseguida pelo acesso direto às memórias de vídeo.

A memória cache deverá ter uma largura de 40 bits (R + G + B + Z) e 16 posições enquanto que a memória de vídeo deverá ter uma largura de 64 bits (2x(R+G+B) + Z) e 1M posições requerendo para isso 256 chips do tipo acima descrito.

Se for necessária uma memória cache maior deverão ser usadas memórias de vídeo menores e em maior quantidade. Por exemplo, 1024 chips de 16Kx4 (também comuns) possibilitam uma memória cache de 64 posições. Veremos adiante que 16 posições para a memória chache são suficientes para o nosso objetivo.

A cada instante a memória cache estará representando um grupo de 16 pixels da imagem ou página da memória de vídeo. A distribuição desses 16 pixels ao longo da imagem deve ser tal que possibilite a maior utilização desses 16 pixels num único acesso à página na memória de vídeo. Considerando-se que o algoritmo do DDA acessa pixels prioritariamente na horizontal, ou seja, não pula para a linha seguinte enquanto não preencher todos os pixels da linha horizontal interna ao triângulo atual, deve-se então dispor os pixels de cada página também na horizontal, na

forma de retângulos de largura de 16 pixels e altura de 1 pixel, como mostra a figura 4.25.

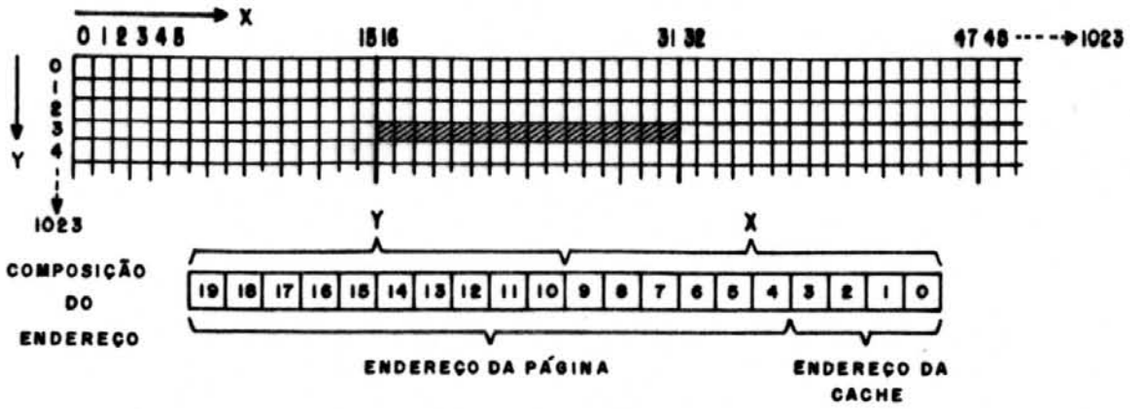


Fig. 4.25 - Area atendida pela mem. cache num dado instante

Esta configuração é obtida dividindo-se as 20 linhas de endereço A0-A19 em duas partes: A0-A3 selecionam um dos 16 pixels da memória cache ou página e os restantes A4-A19 selecionam uma das 64K páginas, que são as 16 linhas que endereçam todos os chips da memória de vídeo.

No instante em o HAD recebe um pixel cujo endereço cai fora da página corrente, o que é detectado pela não coincidência das linhas A4-A19 do novo pixel com o anterior, deverá haver uma troca de páginas entre as memórias cache e de vídeo. A página corrente é devolvida à memória de vídeo e em seguida é buscada uma nova página, cujo endereço é o novo A4-A19. Essa operação de troca de páginas toma um tempo grande comparado com o tempo gasto para cada pixel na memória cache. Durante esse período a emissão de novos pixels por parte do HAD deve ser interrompida. A figura 4.26 apresenta uma idéia dos tempos de acesso às memórias, durante a troca de páginas e durante a operação normal.

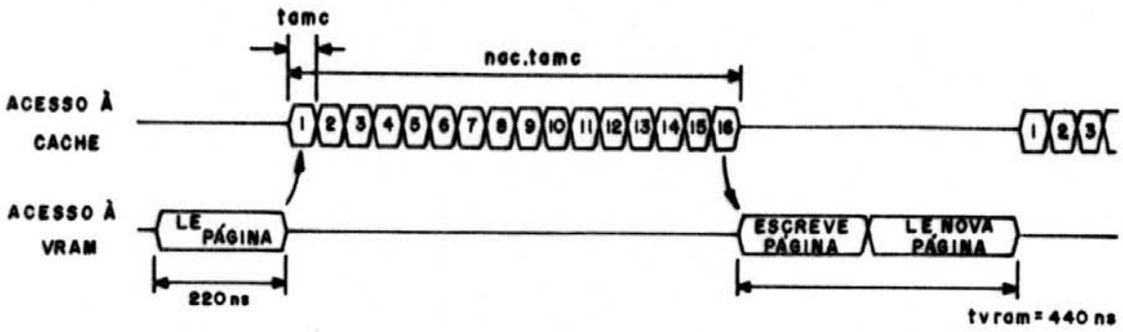


Fig. 4.26 - Diagrama de tempos de acesso às memórias

O tempo médio de acesso aos pixels é dado por:

$$t_{amp} = t_{amc} + t_{vram}/nac$$

onde:

t_{amc} = tempo de acesso à memória cache

t_{vram} = tempo gasto com a VRAM = 440ns (2 acessos)

nac = número médio de acessos por página

Como exemplo tomaremos t_{amc} = 25ns. No melhor caso, com nac=16, teremos um tempo médio de 52.5ns e uma eficiência de 48%. Com nac=1 o tempo médio cresce para 465ns e a eficiência cai para 5,4%. No capitulo seguinte veremos como reduzir t_{vram} aumentando a eficiência de uso da memória cache.

A figura 4.27 mostra um esboço do sistema de memórias.

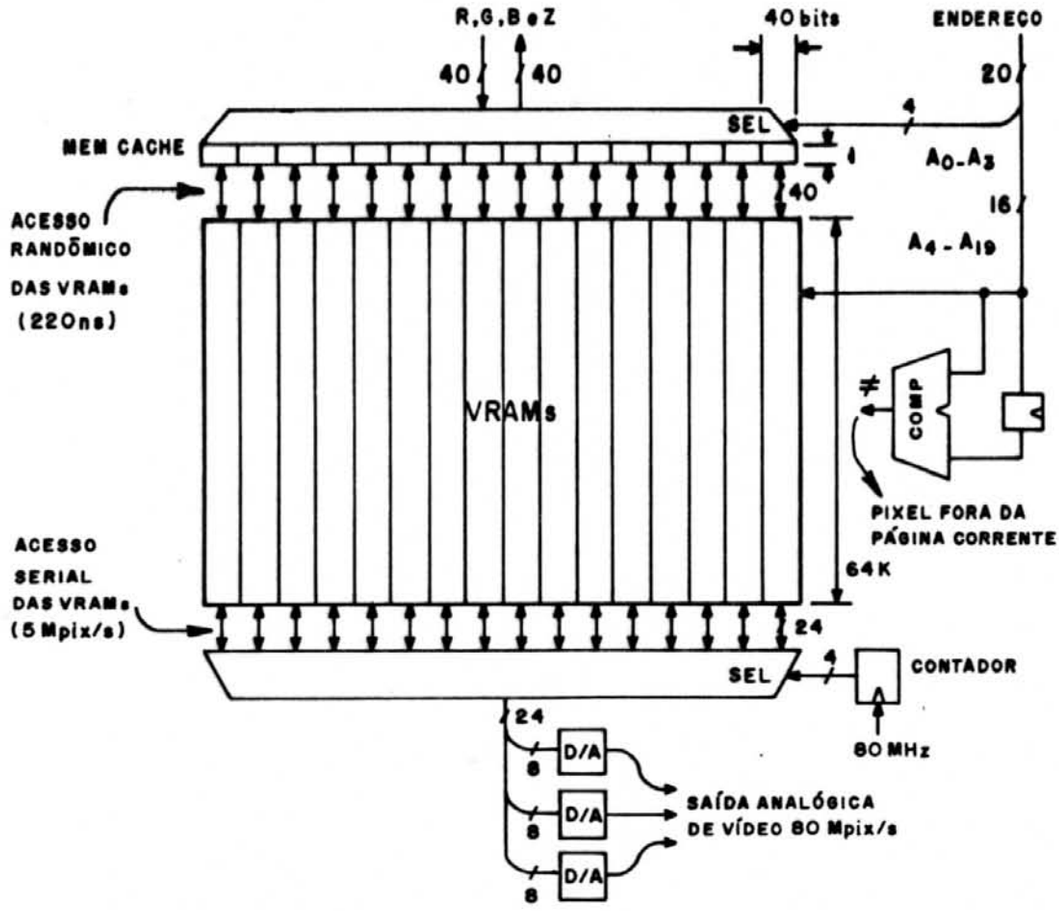


Fig. 4.27 - Esboço dos circuitos de memória cache e de vídeo

O acesso série das VRAMs, a princípio, é destinado à geração do sinal analógico de vídeo através de um conversor Digital-Analógico. Algumas memórias VRAM permitem acesso bidirecional em sua porta série, ou seja, também pode-se escrever na memória de vídeo através desta. Isso pode ser aproveitado para fazer-se um preenchimento bastante rápido de toda a memória. Um preenchimento desse tipo é necessário antes de cada geração completa da imagem. R, G e B recebem a cor de fundo e Z a profundidade máxima. Preenche-se inicialmente todas as 256 posições do shift-register interno às VRAMs e depois transfere-se esta informação para cada uma das 256 linhas das memórias em 256 ciclos de acesso às

VRAMs, ou seja, 256x200ns para preencher os shift-registers (máx 5MHz) mais 256x220ns para transferir seus dados para as linhas internas das memórias. Desta mmaneira o preenchimento de todas as memórias de vídeo toma apenas 1080s. Se fossem usados ciclos de memória isto levaria 220ns x 1M ≈ 230ms. Por intermédio da memória cache também se pode preencher as memórias, levando 16 vezes mais tempo, ou seja 14ms. Na figura 4.27 vemos que o acesso serial das memórias é bidirecional. O caminho de saída passa pelos conversores D-A, gerando o sinal de vídeo; e o de entrada para o preenchimento dos dados contidos em três registradores de cor de fundo.

Como as duas portas dos registradores da cache nunca são acessadas simultaneamente, um circuito de chaveamento é suficiente para desempenhar a função de memória de dupla porta. Este circuito é apresentado na figura 4.28, para um bit da cache. Desta forma os acessos de leitura e escrita do lado do HAD são simultâneos, permitindo que ocorram em um mesmo ciclo.

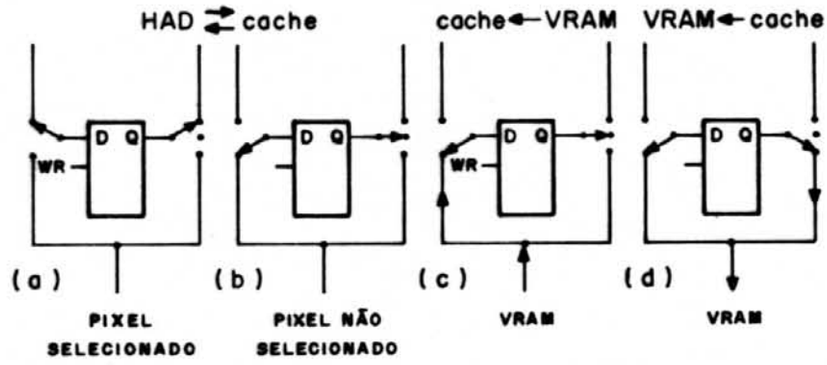


Fig. 4.28 - Circuito de chaveamento da memória cache

Dada a grande quantidade de componentes envolvidos nesse circuito de memória (256 chips VRAMs!) é importante se fazer uma estimativa do aspecto construtivo desses

circuitos. Aparentemente, como vimos na figura 4.26, entre a memória cache e a memória de vídeo existe uma ligação por meio de 640 vias paralelas, o que é proibitivo em termos de circuito impresso. Porém, dada a independência de ligações entre os blocos de memórias associadas a cada posição de memória cache (blocos verticais na figura 4.27) pode-se dispor de toda a memória de vídeo em 16 placas contendo cada uma um dos registradores da cache e 16 VRAMs de 64Kx4, referentes às posições de memória de vídeo associadas a essa posição da cache, perfazendo um total de 64Kx64 bits por placa. Assim o problema da quantidade excessiva de ligações fica resolvido e o circuito de memórias é visto como um agrupamento das memórias relacionadas com a sua posição na memória cache. A interconexão e seleção dessas placas fica minimizada por uma quantidade pequena de ligações, como apresentado na figura 4.29.

As linhas de controle e endereço são todas comuns a cada placa, sendo portanto ligadas em paralelo. As vias de acesso de dados à memória cache e o acesso de entrada e saída série de cada placa são também ligados em paralelo, já que o acesso por estas é sempre feito individualmente para cada placa. A seleção de cada saída é feita por circuitos tri-state, locais a cada placa, ativados por linhas independentes. Para a seleção de uma placa para entrada de dados são empregadas linhas individuais de controle de escrita.

As saídas série de cada placa são colocadas em paralelo e selecionadas por um acionamento tri-state individual de cada saída, ciclicamente a uma velocidade 16 vezes superior à freqüência de saída das portas série das VRAMs, cujo limite situa-se por volta dos 5MHz. Portanto o sinal de vídeo pode atingir uma velocidade de 80MHz.

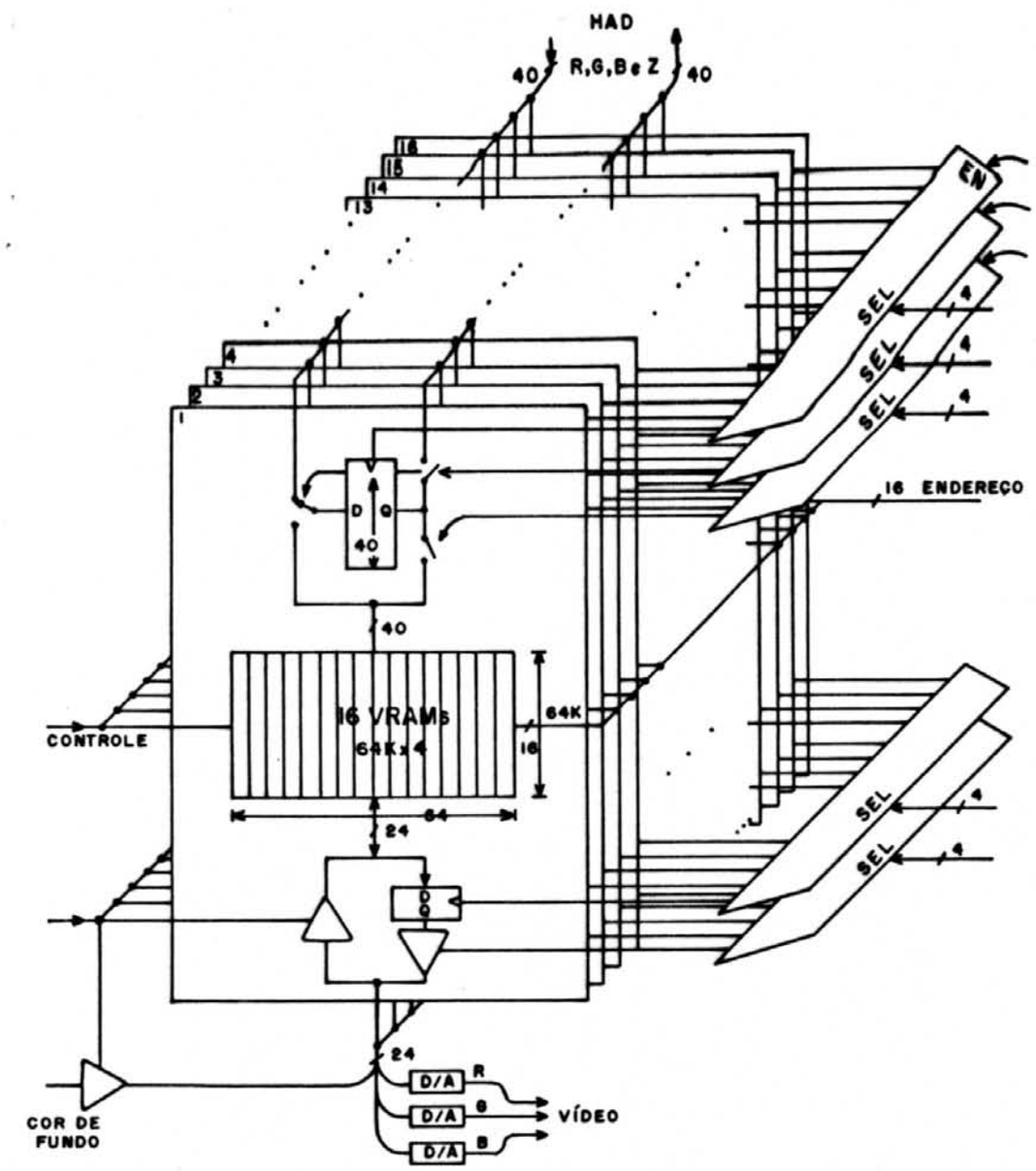


Fig. 4.29 - Distribuição da memória em placas e sua seleção

A geração dos sinais de seleção de placas é feita por demultiplexadores externos, como aqueles na figura 4.29. Com todos esses sinais, o seqüenciador, ou outro circuito independente, pode controlar todas as operações desejáveis

sobre estas memórias - o refresh, o controle individual de cada um dos dois buffers de memória R-G-B, a seqüencialização do sinal de vídeo, a inicialização rápida das memórias com a cor de fundo e profundidade Z máxima, as leituras e escritas rápidas de posições individuais da memória cache e as transferências entre toda a memória cache e a memória de vídeo em um único ciclo.

O circuito de chaveamento da memória cache pode ser implementado por registradores comerciais de 8 bits, como sugere a figura 4.30.

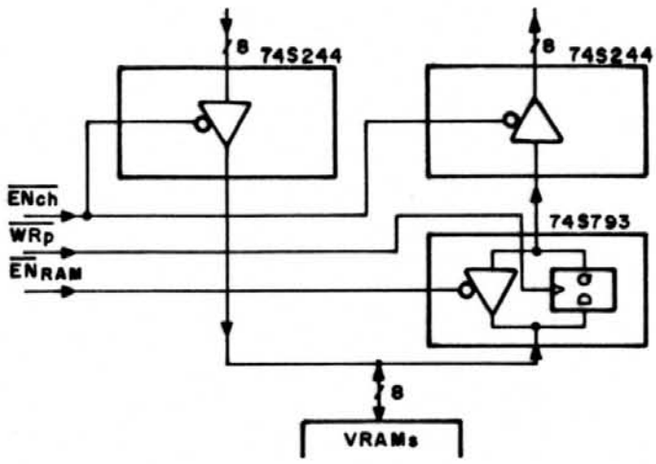


Fig. 4.30 - Implementação prática de 8 bits da cache

Somente com finalidade ilustrativa apresentamos na figura 4.31, na página seguinte, um esboço do circuito de uma das 16 placas de memória empregando componentes práticos, e na figura 4.32 um diagrama resumido de toda a arquitetura dos processadores DDA e HAD.

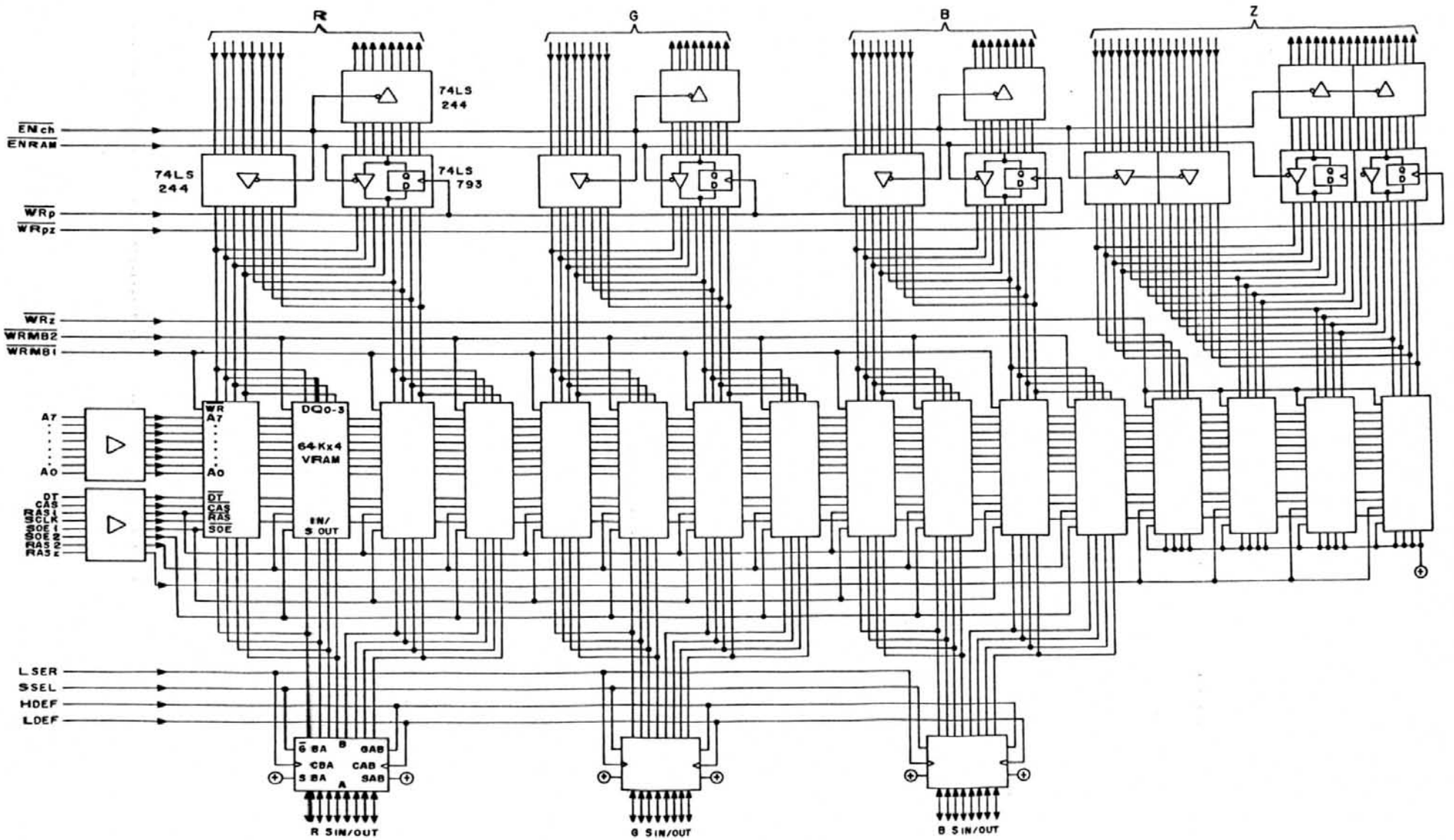


Figura 4.31 - Circuito de uma Placa de Memória 64Kx64 1/16 da Memória de Vídeo e Cache.

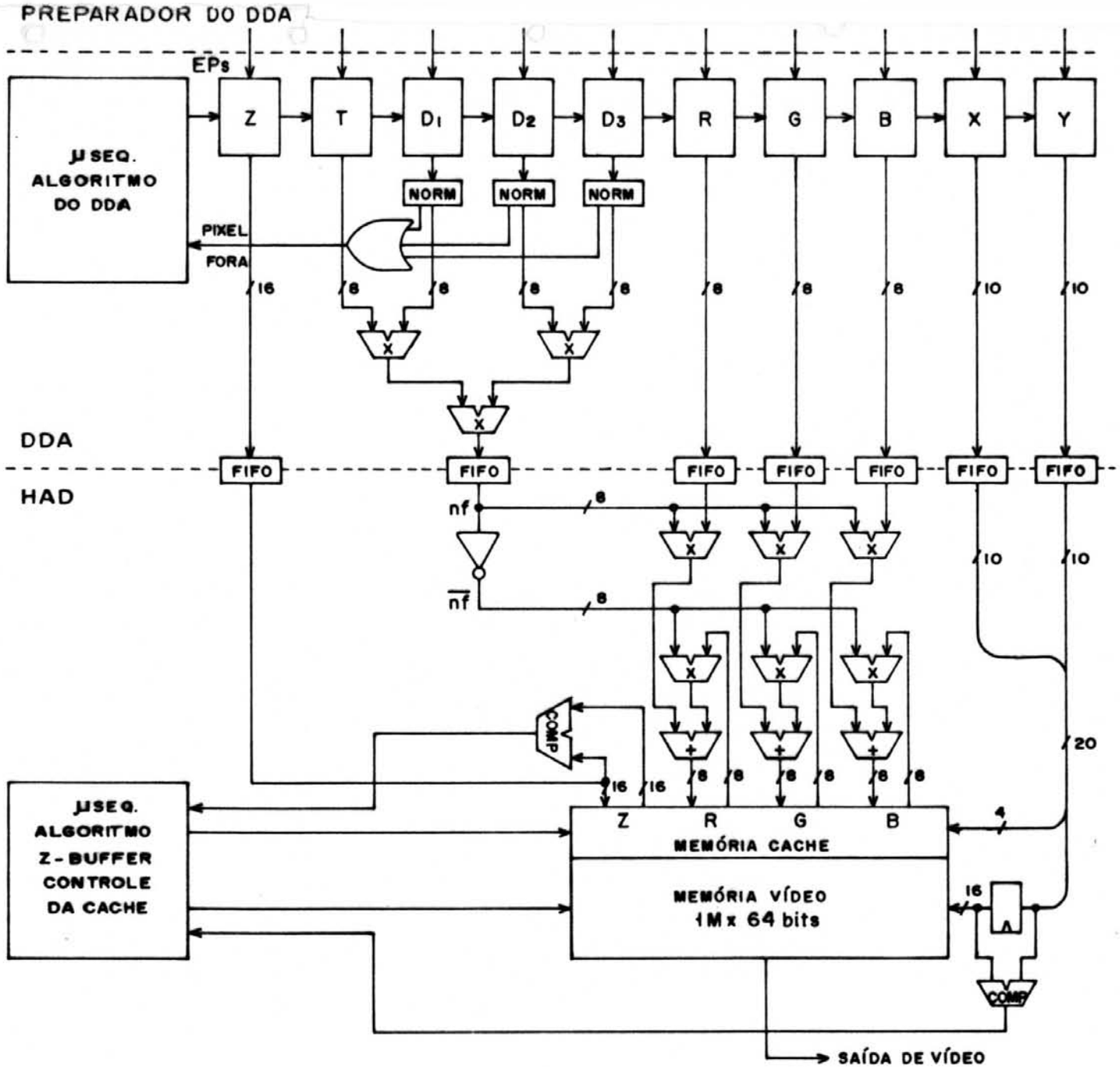


Figura 4.32 - Diagrama Completo do DDA+HAD .

4.3 - Estágios anteriores ao DDA

Esta seção se preocupa em expor o restante dos estágios do pipeline gráfico. Como o objetivo deste trabalho foi o de apresentar um ante-projeto de um DDA de alto desempenho, esta seção existe apenas com a intenção de deixar uma idéia mais completa do GIRTD, uma vez que algumas peculiaridades do DDA podem ser justificadas ou reforçadas pela caracteristicas destes estágios, e estes, reciprocamente, possuem simplificações importantes decorrentes da filosofia de funcionamento do DDA. Para atender esse objetivo, a apresentação desses estágios não será dada com a mesma profundidade aplicada na apresentação do DDA. Apenas serão mostrados seus algoritmos adaptados, de forma a se poder ter também uma noção das exigências de processamento de cada estágio. Esta seção tem um enfoque mais de "propostas para futuros trabalhos", contendo apenas sugestões ou estímulos para a continuação deste projeto.

Como o primeiro estágio do GIRTD apresentado foi o último na direção do fluxo de dados no pipeline - o DDA - apresentaremos o restante dos estágios na ordem inversa a esse sentido, partindo do estágio preparador do DDA, como se fizéssemos o projeto nesta ordem, postergando ao estágio anterior a pronta preparação e entrega dos parâmetros que cada um necessita.

A configuração completa do pipeline do GIRTD é apresentada na figura 4.33.

fig. 4.33 - Pipeline do GIRTD

4.3.1 - Velocidade do fluxo de dados no pipeline

Para que se possa ter uma noção da demanda de dados (polígonos, triângulos ou vértices) que cada estágio impõe ao seu antecessor devemos avaliar a demanda do DDA.

Sabemos que o tempo que o DDA consome em cada triângulo não é constante. Este depende principalmente do número de pixels (visíveis ou não) dos triângulos, a uma vazão média de 50ns/pixel. Consideremos como pior caso (menor triângulo formador de uma superfície triangularizada) um triângulo de 500 pixels, que consumirá 50ns/pixel x 500 pixels = 25μs. Assim sabemos que o preparador do DDA deve estar apto a entregar 40.000 triângulos por segundo. Com triângulos maiores do que 500 pixels esta taxa é reduzida e o gargalo do pipeline então passa a ser o DDA. Com essa demanda máxima de 40.000 triângulos por segundo podemos idealizar o poder de processamento necessário em cada estágio, conforme seus algoritmos.

4.3.2 - O uso de FIFOs entre os estágios

E recomendável que se use uma FIFO entre cada par de estágios para minimizar os tempos de espera entre esses. Cada estágio consome tempos bastante variados, dependendo das caracteristicas dos dados sendo manipulados. O que pode requerer bastante processamento em um determinado estágio pode ser trivial em outro. Dessa maneira, o balanceamento do fluxo dos dados ao longo do pipeline fica bastante prejudicado, pois assim, cada estágio tem que esperar pela disponibilidade do estágio seguinte, eventualmente. Intercalando-se FIFOs entre estes estágios minimiza-se enormemente os tempos de espera acarretados por essa irregularidade nos tempos de execução dos estágios. Assim, o fluxo de dados ao longo do pipeline se torna mais equilibrado. Os estágios passam mais tempo ocupados, garantindo um melhor desempenho do sistema.

A profundidade das FIFOs deve ser estipulada de maneira que a irregularidade de tempos de processamentos entre os estágios tenha pouca influência sobre o balanceamento do pipeline.

4.3.3 - Preparador do DDA

Como foi visto, para o preenchimento de cada triângulo, o DDA recebe parâmetros que descrevem as condições iniciais (coordenadas, componentes de cor, transparência e as três distâncias às arestas), e parâmetros que descrevem o comportamento desses dados ao longo da área do triângulo (as derivadas em relação a X e a Y). A função do preparador do DDA é a de calcular esses parametros a partir das informações contidas em cada um dos 3 vértices do triângulo.

4.3.3.1 - limites X_{max}, X_{min} e Y_{max}

Estes valores são obtidos por comparação entre as coordenadas do triângulo e os limites da área de visualização. Eles são os valores máximos e mínimos de X e Y pertencentes à área do triângulo que é capturada pela janela de visualização. Esta janela é definida pelos seus limites X_{Jmin}, X_{Jmax}, Y_{Jmin} e Y_{Jmax}, que não são necessariamente iguais aos limites da área visível da tela gráfica. Estes valores são entregues ao preparador do DDA diretamente do utilitário gráfico em execução no hospedeiro pois determinam a área da tela que se quer ocupar com a imagem gráfica.

As seguintes expressões determinam X_{min}, X_{max} e Y_{max}.

$$X_{min} = \max (\min(X_1, X_2, X_3) , X_{Jmin})$$

$$X_{max} = \min (\max(X_1, X_2, X_3) , X_{Jmax})$$

$$Y_{max} = \min (\max(Y_1, Y_2, Y_3) , Y_{Jmax})$$

4.3.3.2 - Ponto de partida X_0, Y_0

O ponto de partida para o preenchimento de um triângulo deve ser aquele com menor Y possível, desde que caia dentro da janela de visualização. A obtenção deste ponto não fica ao encargo do preparador do DDA. Este é calculado no estágio de recorte, anterior a este e passado direto ao DDA. X_0 e Y_0 são utilizados neste estágio para a obtenção dos atributos iniciais, R_0, G_0, B_0, T_0, Z_0, D_{1_0}, D_{2_0} e D_{3_0}.

4.3.3.3 - Derivadas em relação a X e Y.

As derivadas, para triângulos com arestas não coincidentes, são obtidas a partir dos atributos contidos nos três vértices, respectivamente P_1, P_2 e P_3, segundo as seguintes equações:

$$P'_x = \frac{(P_2 - P_1).(Y_3 - Y_1) - (P_3 - P_1).(Y_2 - Y_1)}{(X_2 - X_1).(Y_3 - Y_1) - (X_3 - X_1).(Y_2 - Y_1)}$$

$$P'_y = \frac{(P_3 - P_1).(X_2 - X_1) - (P_2 - P_1).(X_3 - X_1)}{(X_2 - X_1).(Y_3 - Y_1) - (X_3 - X_1).(Y_2 - Y_1)}$$

onde $P \in \{ R, G, B, T, Z \}$.

O denominador das expressões é o produto vetorial de qualquer par de arestas do triângulo no sentido anti-horário. Para evitar-se duas divisões este produto pode ser calculado separadamente, tirado seu inverso e multiplicado nas expressões de cima das frações.

4.3.3.4 - Cálculo das derivadas das distâncias D1, D2 e D3

Esse cálculo é feito no estágio de seleção e recorte, anterior ao preparador do DDA, porque para o recorte essas informações são necessárias. No entanto, por se tratar de cálculos da mesma natureza do restante - preparação das derivadas para o DDA, estes serão aqui descritos.

Para obtermos Di'_x, Di'_y e Di_0, onde i=1, 2 ou 3, devemos ter em mente a equação das arestas a partir das coordenadas de seus vértices, na forma:

$$A.x + B.y + C = 0$$

onde (para i=1)

$$A = Y_2 - Y_1$$
$$B = X_1 - X_2$$
$$C = -A.X_1 - B.Y_1$$

A equação para a distância de um ponto a um reta pode ser dada por:

$$D = \frac{A.x + B.y + C}{\sqrt{(A^2 + B^2)}}$$

O denominador é o comprimento da aresta, que chamaremos L, portanto:

$$Di'_x = A/L \quad Di'_y = B/L \quad Di_\theta = C/L$$

Di_θ é a distância da aresta i à origem (0,0). Precisamos agora é obter Di_0, que não é o mesmo que Di_θ. Di_0 é a distancia da aresta ao ponto de partida do DDA, que é (X_0, Y_0). Para isso aplica-se a equação obtida nesse ponto:

$$Di_0 = Di'_x . X_0 + Di'_y . Y_0 + Di_\theta$$

4.3.3.5 - Valores iniciais dos atributos de pixel

Os atributos de pixel para o disparo do preenchimento são aqueles que ocorrem no ponto de partida $P_0 (X_0, Y_0)$. Como P_0 não é necessariamente um dos vértices, onde os atributos já são conhecidos, estes devem ser

calculados. Toma-se um vértice qualquer e seu atributo P_i, calcula-se o afastamento de P_0 a esse vértice em termos de diferença em X e em Y e acrescenta-se a esse atributo P_i as variações de P devidas ao efeito desse afastamento com as derivadas em X e Y. Em resumo:

$$P_0 = P_i + (X_0 - X_i).P'_x + (Y_0 - Y_i).P'_y$$

4.3.3.6 - Avaliação do poder de computação requerido

Levando em conta a demanda do DDA de 25μs por triângulo e a quantidade de cálculos por triângulo necessários neste estágio, conclui-se que isso não pode ser feito por um processador simples. A grande maioria dos cálculos envolve ponto flutuante, conforme a lista a seguir:

3	radiciações
4	divisões
51	multiplicações
81	somas, subtrações ou comparações

Para efeitos estimativos consideremos somas, subtrações, comparações e multiplicações equivalentes a 1 opf (Operação de Ponto Flutuante) cada; divisões 4 opfs e radiciações 8 opfs cada. Isso corresponde a um total de 181 opfs necessárias por triângulo. Dessas 181, 57 correspondem ao cálculo das derivadas das distâncias, que é feito no estágio de seleção e recorte, restando 124 opfs para este estágio. Como cada triângulo deve levar no máximo 25μs para ser preparado para o DDA, este estágio requer 124/25 = 4.96 Mflops (Floating Point Operation per Second) de desempenho. Os DSPs existentes hoje (p/ex. a série ADSP4100 da Analog Devices), com seus processadores de ponto flutuante, podem atingir 10 Mflops, dependendo do grau de paralelismo usado. Assim o uso de um ou dois desses

processadores pode tranqüilamente suprir o desempenho necessário.

4.3.4 - Seleção e recorte

Aqui os triângulos devem ser recortados contra a janela de visualização, ou descartados caso não interceptem a janela. Tradicionalmente o recorte implica em transformar uma figura geométrica em outra correspondente à intersecção da área interna desta com a área interna da janela de visualização. Este é um processo complexo e demorado. Um triângulo pode ser transformado em um polígono de até 7 lados e deverá ser novamente transformado em triângulos. Por outro lado, caso não se faça o recorte, o DDA poderá perder muito tempo tentando preencher pixels externos à janela.

Devido à natureza do preenchimento dos triângulos por parte do DDA, podemos executar o recorte simplesmente especificando ao DDA um ponto de partida conveniente para o preenchimento. Assim, ao invés de recortarmos realmente os triângulos, estamos somente evitando que o DDA perca tempo com pixels externos à janela.

O preenchimento se dá de cima para baixo (y crescente), uma linha horizontal após a outra. Portanto o ponto de partida deve ser aquele mais alto (menor y) que pertença ao triângulo e esteja dentro da janela. Dessa maneira recortamos a porção do triângulo fora da janela acima deste ponto. As porções do triângulo abaixo deste ponto e que ainda estão fora da janela são automaticamente recortadas pelo próprio algoritmo DDA, quando este testa os limites Xmin, Xmax e Ymax, sem perder nenhum tempo com os pixels destas porções.

A figura 4.34 apresenta alguns casos de disposições de triângulos sobre a janela de recorte e as soluções para o ponto de partida de preenchimento.

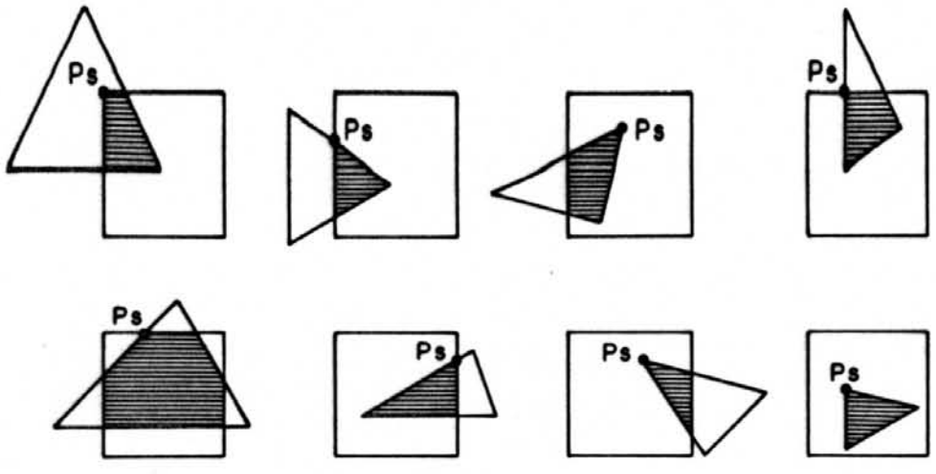


fig. 4.34 - Exemplos de escolhas para o ponto de partida

Para a seleção e recorte, então, este estágio inicialmente calcula as derivadas das distâncias D1, D2 e D3, conforme decrito em 4.3.3.4. Estes dados representam as 3 equações das arestas do triângulo que serão usadas pelo algoritmo para identificar as intersecções destas retas com os limites da janela de recorte, além de serem usados pelo DDA, na identificação dos limites do triângulo. Em seguida testa-se se o canto superior à esquerda da janela (XJmin, YJmin) cai dentro do triângulo. Caso positivo, este é escolhido como ponto de partida pois este serve como tal para qualquer triângulo que o contenha, inclusive triângulos que contenham toda a janela de recorte. Caso negativo, são escolhidos pontos candidatos para uma posterior seleção. Estes candidatos são os vértices do triângulo que estejam dentro da janela e as intersecções das arestas com os limites da janela. Após estes candidatos serem todos empilhados, será escolhido como ponto de partida (Xo, Yo) aquele que tiver menor Y. E caso não haja candidatos, tem-se

uma indicação segura de que o triângulo não intercepta a janela, e é então finalmente descartado.

4.3.5 - Recorte de profundidade

A projeção em perspectiva por si só não elimina as porções da imagem que caem atrás da câmera sintética. Caso interceptem a dupla pirâmide de visualização, como mostra a figura 4.35, serão exibidas na imagem projetada tanto as porções à frente da câmera como as de trás sendo que as de trás aparecerão invertidas na tela.

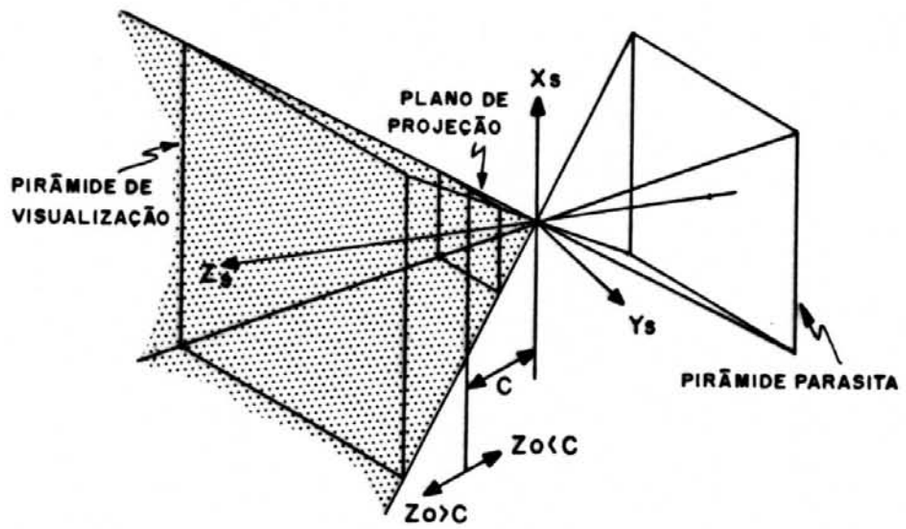


fig. 4.35 - Pirâmide dupla de visualização

As porções indesejadas, que são capturados pela pirâmide parasita atrás da câmera, podem ser eliminadas pela observação de suas profundidades Zs, nas coordenadas de câmera, que são menores que C, que é a distância focal, mostrada na figura 4.35. Costuma-se eliminar elementos com Zs>C ao invés de Zs>0 para evitar que a câmera visualize objetos com dimensões projetadas infinitas, no caso de uma grande proximidade da câmera com estes objetos. Ocorrem também casos em que um objeto intercepta ambos os lados do

plano Zs=C podendo inclusive interceptar as duas pirâmides de visualização. Um exemplo disto é o que ocorre em um simulador de vôo, onde a câmera sintética está associada a um avião imaginário que sobrevoa um grande plano que é a superfície da terra. A pirâmide da frente da câmera captura a superfície da terra normalmente, como se estivesse abaixo do avião, e se a pirâmide de trás não for eliminada, a parte de trás da superfície da terra também será capturada e aparecerá invertida na imagem como se existisse um teto no céu paralelo ao chão, com um aspecto semelhante a este.

Um objeto que está tanto na frente como atrás da câmera deve ser seccionado em duas partes, a anterior e a posterior, e somente a posterior será exibida. A isso chamamos de recorte de profundidade. No nosso caso os objetos são triângulos e o recorte de profundidade em triângulos pode se dar em 4 casos, conforme o número de vértices que tenham Zs > C - 0, 1, 2 ou 3 - como o ilustrado na figura 4.36.

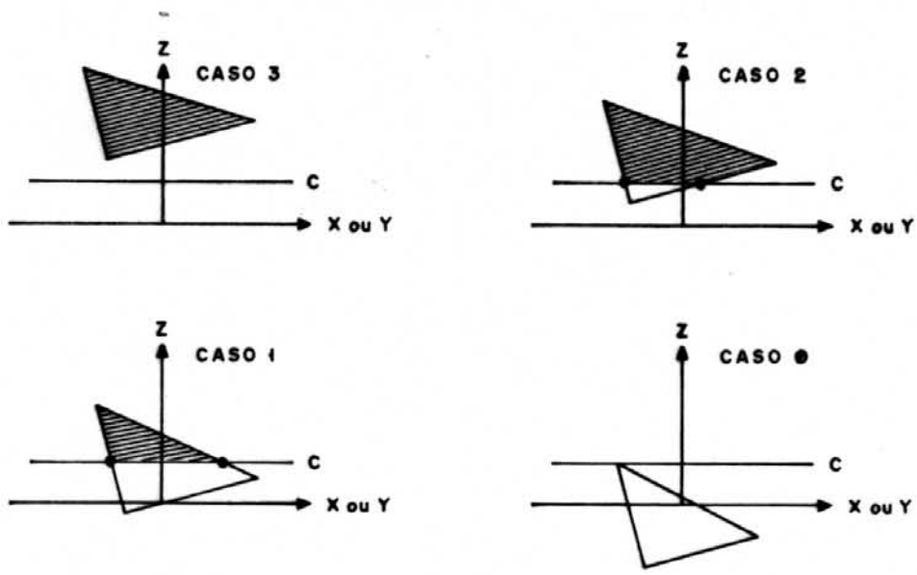


fig. 4.36 - Quatro casos de recorte em profundidade

O caso 0 significa que nenhum vértice pertence ao semi-espaço Zs > C de maneira que todo o triângulo é ignorado. No caso 3 todos os vértices pertencem a esse semi-espaço de maneira que o triângulo é selecionado integralmente como candidato ao estágio posterior. Nos casos 1 e 2 o triângulo intercepta parcialmente o semi-espaço de maneira que a parte restante do triângulo deverá ser eliminada através do processo de recorte que aqui será descrito.

Esse recorte implica em alteração geométrica e topológica dos triângulos. No caso 1 o triângulo será transformado em outro com dimensões diferentes e no caso 2, como o resultado do recorte é um quadrilátero, esse quadrilátero deverá ser transformado em 2 triângulos pela criação de uma nova aresta em uma das diagonais deste quadrilátero, já que os estágios posteriores só trabalham com triângulos.

Para os novos vértices gerados pela intersecção do triângulo com o plano limite Zs=C deverão ser recalculados os atributos R, G, B, T e Z por interpolação linear dos atributos contidos nos vértices da aresta que contém este novo vértice (Note que Z é a profundidade no sistema de coordenadas da tela e Zs é a profundidade no sistema de coordenadas da câmera).

O algoritmo é simples e não será mostrado em forma de programa, apenas serão descritos verbalmente seus procedimentos.

Inicialmente são analisados os 3 vértices com seus atributos X, Y, Z, R, G, B e T, incluindo os Zs da câmera. Caso os 3 Zs sejam maiores que C, que é um parâmetro global ao pipeline, o triângulo é entregue ao estágio seguinte assim como foi recebido, excluindo as informações Zs, que

não são importantes nos estágios posteriores. Se os 3 Zs forem menores que C o triângulo é ignorado e nada é passado ao estágio seguinte. Se cair no caso 1, as duas arestas que atravessam o plano Zs=C geram pontos que substituirão os vértices com Zs<C. Seus atributos serão também substituídos por aqueles calculados pela interpolação entre os vértices separados pelo plano Zs=C, na forma:

$$P_i = \frac{P1.(Zs1-C) + P2.(C-Zs2)}{Zs1-Zs2}$$

P1 - atributos do vértice Zs1>C

P2 - atributos do vértice Zs1<C

e este novo triângulo é então entregue ao preparador do DDA. No caso 2, somente um vértice cai no semi-espaço Zs<0 de maneira que este será substituído por 2 vértices gerados pela intersecção das 2 arestas que cortam o plano Zs=C, que, em conjunto com os outros 2 vértices já existentes dá um total de 4 vértices, ou seja, um quadrilátero. O algoritmo agora procede transmitindo ao estágio seguinte 2 triângulos, um de cada vez, contendo cada um 3 desses 4 vértices de maneira que esses 2 triângulos se completam formando o ~~tetraedro.~~ quadrilátero

Este é certamente um algoritmo bem mais simples que os existentes nos estágios de recorte e preparação do DDA não havendo preocupações relacionadas com velocidade.

4.3.6 - Triangularização

Este é o estágio que adapta a representação B-rep poligonal à B-rep triangular, que é o princípio da filosofia do GIRTD. Assim o GIRTD pode receber descrições baseadas em

qualquer tipo de polígono. A triangularização de um polígono de n arestas é feita subdividindo-se este em n-2 triângulos pela inserção de arestas internas ao polígono, sem a necessidade de acrescentar-se algum novo vértice. Como todos os triângulos resultantes usam os mesmos vértices do polígono original não há a necessidade de calcular-se nem os atributos e nem as coordenadas de novos vértices ou alterar os já existentes.

O triangularizador, ao receber um polígono do seu estágio anterior, avalia inicialmente o número de arestas que esse possui. Se este número for 3 não há trabalho a se fazer e esse triângulo é passado adiante ao estágio seguinte. Se o número for maior que 3 então são calculados os produtos vetoriais dos vértices, conforme figura 4.37, para identificação de algum vértice responsável por uma concavidade no polígono. Este vértice é identificado pelo fato do sinal da componente Z do seu produto vetorial ser diferente dos demais.

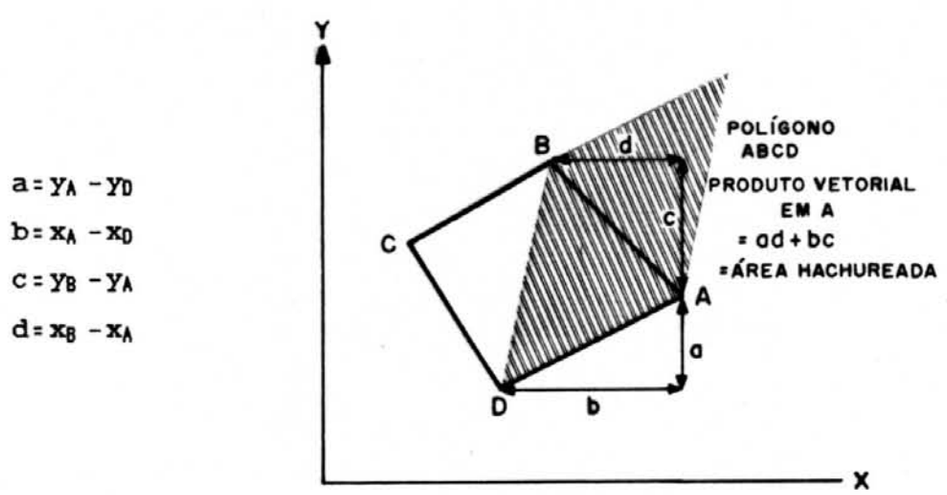


fig. 4.37 - Definição do produto vetorial de vértice

O produto vetorial de um vértice é definido pelo produto vetorial dos vetores formados pelas arestas que contêm o vétice em questão. No exemplo da figura 4.37, o produto vetorial do vértice A é o produto vetorial dos vetores D-A e A-B. Como estes vetores estão restritos ao plano XY o resultado do produto vetorial só possui a componente Z. Seu módulo equivale a área indicada na figura 4.37. O sinal da componente Z, que é o que nos interessa, pode ser obtido pela correta obtenção dos termos a, b, c e d.

Se não houver vértice côncavo a solução é imediata. Escolhe-se um vértice qualquer do qual partirão novas arestas a todos os vértices que ainda não são ligados, através de arestas, a esse vértice escolhido. Caso se encontre apenas um vértice com seu sinal diferente dos outros usa-se o mesmo procedimento anterior, mas com esse vértice. E se for encontrado mais de um vértice com sinal diferente dos outros o tratamento já não é trivial e requer um processamento especial. Na prática a grande maioria dos poligonos não possui mais do que uma concavidade de maneira que o tempo consumido com estes casos especiais pouco influi no tempo médio representado pelos casos mais simples.

A triangularização supõe que os vértices representados no plano X-Y-P sejam coplanares. Caso isto não ocorra diferentes escolhas de arestas internas produzirão diferentes comportamentos dos atributos P durante a interpolação no DDA.

Maiores detalhes teóricos sobre triangularização podem ser vistos em [WOR 83].

4.3.7 - Transformação de vista e projeção

Aqui se aplica as transformações de vista necessárias para a correta captura da imagem contida no universo.

A transformação de vista consiste num produto matricial aplicado às coordenadas de cada vértice dos poliedros no universo pela matriz de transformação de vista, preservando suas topologias. Esta matriz representa o efeito dos 3 ângulos de visualização α, β e γ, e as coordenadas da câmera Xs, Ys e Zs, conforme apresentado no capitulo II. Os valores dessa matriz são constantes ao longo da geração de uma imagem completa, de maneira que não necessitam ser calculados constantemente. Eles precisam ser calculados entre uma imagem e outra quando se quer trocar o ponto de vista, que é o que acontece dinamicamente durante a utilização do GIRTD como simulador de vôo. Esses componentes da matriz não precisam ser calculados neste estágio podendo ser fornecidos pelo próprio computador hospedeiro, já que a velocidade de sua obtenção não é critica no desempenho do pipeline.

O produto matricial da transformação é dado por

$$\begin{bmatrix} X_s & Y_s & Z_s \end{bmatrix} = \begin{bmatrix} X_u - X_c & Y_u - Y_c & Z_u - Z_c \end{bmatrix} x \begin{bmatrix} T_{1\,1} & T_{1\,2} & T_{1\,3} \\ T_{2\,1} & T_{2\,2} & T_{2\,3} \\ T_{3\,1} & T_{3\,2} & T_{3\,3} \end{bmatrix}$$

onde

X_u, Y_u, Z_u = ponto segundo as coordenadas do universo;

X_s, Y_s, Z_s = ponto segundo as coordenadas da câmera;

X_c, Y_c, Z_c = centro da câmera segundo as coordedadas do universo.

X_s, Y_s correspondem às coordenadas segundo uma projeção paralela.

Para um realismo mais fiel, a projeção cônica também deve ser feita. Com isso é obtido, através da transformação de perspectiva, um efeito de profundidade, onde os objetos mais distantes aparecerão proporcionalmente menores. Nessa transformação, esta proporção é dada pelo inverso da profundidade Z_s, conforme as expressões

$$X_p = \frac{X_s}{Z_s} \times \frac{D}{S} \qquad Y_p = \frac{Y_s}{Z_s} \times \frac{D}{S}$$

onde D/S é uma constante de proporcionalidade explicada no capitulo II.

A profundidade Z_s também é convertida em Z_p, conforme mostrado no capitulo II, pelas razões que aqui repetiremos:

- melhor aproveitamento da precisão disponivel para a representação da profundidade;
- corrigir o efeito de curvatura causado pela não linearidade do termo $1/Z_s$;
- permitir o uso de interpolação linear da profundidade.

Essa conversão é dada por:

$$Z_p = \frac{Z_{max}}{Z_{max} - Z_{min}} \times \left(1 - \frac{Z_{min}}{Z_s}\right) \times P$$

Z_{min} e Z_{max} são a menor e a maior profundidade encontradas no conjunto de vértices dos objetos, compreendidos no semi-espaço $Z_s > C$.

Z_e variando de Z_{min} até Z_{max} faz com que Z_p varie de 0 até P, de uma maneira não linear para compensar a não linearidade das expressões anteriores.

4.3.8 - Modelagem de iluminação

Este é o estágio responsável pela realização do cálculo dos atributos de cor e transparência dos vértices. E o coração do realismo, onde se efetua uma aproximação matemática dos fenômenos de propagação de luz que ocorrem em ambientes reais. Diversos modelos matemáticos foram desenvolvidos para representar esses fenômenos - a reflexão, a refração, a difusão e o sombreamento - e alguns desses modelos foram exemplificados no capitulo II. Aqui não nos preocuparemos em obter um realismo quase perfeito, como alguns sistemas "ray-trace" conseguem, mas em obter uma aproximação que nos permita ter uma noção dos objetos sem ambigüidades e uma melhor compreensão da profundidade e disposição dos objetos no espaço.

A escolha do grau de realismo é um compromisso entre precisão e custos computacionais. Algumas imprecisões já foram estabelecidas por aproximações, no decorrer do projeto, que são:

- modelo geométrico aproximado por poliedros
- aproximação linear da luminosidade ao longo dos poligonos
- não há suporte para geração de sombras
- desconsidera-se a refração dos materiais transparentes

No GIRTD consideraremos somente a reflexão, a difusão e a transparência calculadas nos vértices dos poliedros, mantendo-se assim uma compatibilidade com o poder

computacional dos outros estágios. Sistemas gráficos para realismo com essas caracteristicas tem ampla atilização nos ambientes de CAD e simulação de vôo, ou seja, nas aplicações a que o GIRTD se propõe.

O modelo de sombreamento usado aqui é uma simplificação daquelas técnicas apresentadas no capitulo II. Este modelo pode ser decomposto em quatro partes, uma contribuição da luz difusa do ambiente, uma contribuição do efeito da reflexão difusa das superficies, o efeito da reflexão especular de uma fonte de luz e, por último, o efeito de transparência dos materiais.

A luz difusa do ambiente é aquela que se propaga em todas as direções, decorrente de múltiplas reflexões no ambiente. E o efeito que ilumina as partes dos objetos que não estão sendo iluminados diretamente por uma fonte de luz. Consideraremos a luz difusa constante em todo o espaço e em qualquer direção. Esta é representada por $[I_d]$ (Intensidade de luz Difusa) que é um vetor cujos 3 elementos são as intensidades individuais R, G e B de luz difusa no ambiente. O efeito desta luz sobre um corpo cuja cor (ou coeficientes de reflexão da luz) é [R] é o produto vetorial

$$[E_d] = [R] \times [I_d]$$

ou em termo de componentes de cor R, G e B

$$E_{dr} = R_r \cdot I_{dr}$$
$$E_{dg} = R_g \cdot I_{dg}$$
$$E_{db} = R_b \cdot I_{db}$$

onde E_d significa "Energia luminosa num dado ponto devido ao efeito da luz difusa sobre este".

A reflexão difusa de luz é um efeito causado por uma fonte de luz sobre um corpo que reflete esta uniformemente em todas as direções, de acordo com a sua cor. Esta reflexão é explicada pela lei de Lambert, que estabelece que a energia de luz que atinge uma superficie é proporcional ao cosseno do ângulo de incidência desta luz. Portanto a contribuição da reflexão difusa em um ponto da superficie é dado por:

$$[E_{rd}] = [R] \times [I_f] . \cos(i)$$

$[I_f]$ é a intensidade de luz oriunda de uma fonte luminosa no infinito e i é o ângulo de incidência desta luz sobre a superficie. Cos(i) pode ser dado também pelo produto escalar entre o vetor unitário **I**, que representa a direção de irradiação da fonte de luz, e o vetor normal da superficie **N**:

$$\cos(i) = \mathbf{I} . \mathbf{N}$$

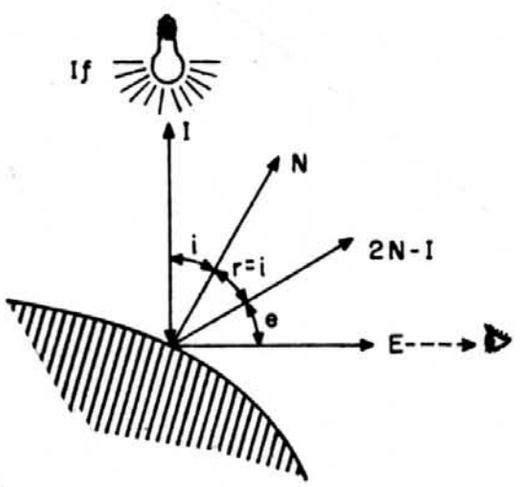


fig. 4.38 - Definição dos ângulos e vetores referenciados no texto

portanto $[E_{rd}]$ pode ser expresso por:

$$[E_{rd}] = [R]x[I_f].(I.N)$$

A figura 4.38 apresenta o significado dos ângulos de incidência "i", de reflexão "r" e de reflexão especular "e" e dos vetores, normal à superfície **N**, incidência **I**, reflexão **R** e reflexão especular **E**.

A reflexão especular caracteriza-se por não depender da cor do objeto e por concentrar-se predominantemente em torno do vetor de reflexão **R**, decaindo, à medida que nos afastamos deste vetor, a uma velocidade que depende de um fator de polimento da superfície. Esse fator n nos identifica quão polida é a superfície ou o quanto esta se comporta de maneira semelhante a um espelho. Um n grande (n=200) causa um comportamento muito similar a um espelho enquanto que um n pequeno (n=1) faz com que a reflexão especular se assemelhe à reflexão difusa de uma superfície fosca, como uma folha de papel.

A contribuição de energia luminosa por reflexão especular a um dado ponto da superfície é:

$$[E_{re}] = CRE(i).\cos^n(e).[I_f]$$

CRE(i) é uma função que expressa o comportamento da intensidade de luz refletida segundo o vetor **R** em função do ângulo de incidência, o que caracterizará o tipo de material das superfícies. Esta função pode ser entregue a este estágio na forma de uma tabela em que são plotados valores de CRE(i) para alguns valores de i entre 0° e 90° para interpolação linear.

O ângulo "e" é aquele formado entre os vetores **R** e **E**. Cosseno de "e" pode então ser dado pelo produto escalar (**R**.**E**).**R**, que por sua vez pode ser obtido por (2.**N** - **I**). Reescrevendo-se o cos(e) temos:

$$\cos(e) = (2.\mathbf{N}-\mathbf{I}).\mathbf{E}$$

Assim a expressão para $[E_{re}]$, sem o uso de funções trigonométricas, fica:

$$[E_{re}] = CRE(i).((2.\mathbf{N}-\mathbf{I}).\mathbf{E})^n.[I_f]$$

A contribuição da transparência será feita no DDA, uma vez que aqui ainda não temos condições de saber a intensidade de luz emanada por detrás das superficies. Portanto a transparência será calculada aqui mas só será aplicada no estágio do DDA.

O resultante das componentes R, G e B, que será atribuido aos vértices dos poliedros, é obtido então da soma de todas as contribuções, exceto a transparência, assim:

$$[R\ G\ B] = [E_d] + [E_{rd}] + [E_{re}]$$

Quanto à transparência, sabe-se, pela prática, que um material semi-transparente plano possui a maior transparência quando a luz incide ortogonalmente a este e que esta transparência diminui à medida em que inclinamos este ângulo de incidência. Isso pode ser justificado pelo aumento do caminho que a luz tem que atravessar, quando inclinamos esta placa semi-transparente, o que é proporcional ao cosseno do ângulo de incidência. Podemos, a partir dessa consideração, criar um modelo intuitivo em que a 0° tem-se uma transparência máxima T_{max} e a 90° ou -90° tem-se uma transparência minima T_{min} e em pontos

intermediários tem-se valores de transparência entre T_{min} e T_{max}, segundo uma dosagem dada pelo cosseno do ângulo t de transmissão. Isso pode ser escrito da seguinte forma:

$$T_f = T_{min} + (T_{max} - T_{min}) . |\cos(t)|$$

O cosseno é dado em módulo para permitir também a transparência de faces opostas à câmera de visão. O cos(t) pode ser dado pelo produto escalar entre a normal à superfície **N** e o vetor unitário **E**, que representa a direção em que a câmera está. Podemos então reescrever a expressão na forma:

$$T_f = T_{min} + (T_{max} - T_{min}) . |\mathbf{N} . \mathbf{E}|$$

Para um polígono opaco, faz-se $T_{max} = T_{min} = 0$.

Tf varia de 0 a 1, respectivamente da transparência nula a total. O valor T usado pelo DDA é definido de maneira contrária sendo que T deve ser definido por $1 - T_f$.

Todos esses cálculos necessitam do conhecimento do vetor normal à superfície **N**, nos vértices do poliedro. Esses podem ser calculados pela média dos vetores normais dos polígonos, de uma mesma superfície, que possuem este vértice.

Por definição, uma superfície é um conjunto de polígonos e os objetos são compostos de superfícies e não diretamente por polígonos. O sombreamento deve ser contínuo somente dentro de uma superfície e não entre superfícies justapostas, portanto polígonos justapostos e que pertencem a superfícies distintas não devem ter continuidade de sombreamento entre si. Os vértices comuns a essas duas

superfícies devem ter referências isoladas para que possam ter 2 vetores normais, um referente aos polígonos de uma superfície que contêm este vértice, e o outro referente aos polígonos da outra superfície e que também contêm este vértice. Além disso, quem contêm as especificações que caracterizam o material de que é feito o objeto, os parâmetros de cor, reflexão e transparência, são as superfícies e não os polígonos. Essa distinção entre superfícies e polígonos é garantida pela estrutura de dados de entrada do GIRTD, descrita no ítem seguinte.

Todos os cálculos dos vetores normais aos polígonos e posteriormente aos vetores normais dos vértices somente são feitos uma primeira vez, antes dos processamentos de mudanças de vista. Estes só precisam ser recalculados se houver alguma alteração na estrutura de dados, como a inserção, remoção ou alteração de partes dos objetos. Inicialmente são calculadas as normais dos polígonos. Escolhendo-se arbitrariamente 3 vértices deste polígono, calcula-se o produto vetorial e divide-se o resultado pelo seu módulo. A seguir percorre-se os polígonos e para cada um percorre-se os vértices por ele referenciados, somando neles o vetor normal do polígono e incrementando um contador de referências por polígono. No final, em cada vértice, divide-se o acumulador de vetores normais pelo contador e tem-se então o vetor normal de cada vértice.

4.3.9 - Estrutura de dados de entrada do GIRTD

A figura 4.39 apresenta um esquema da estrutura de dados de partida do GIRTD.

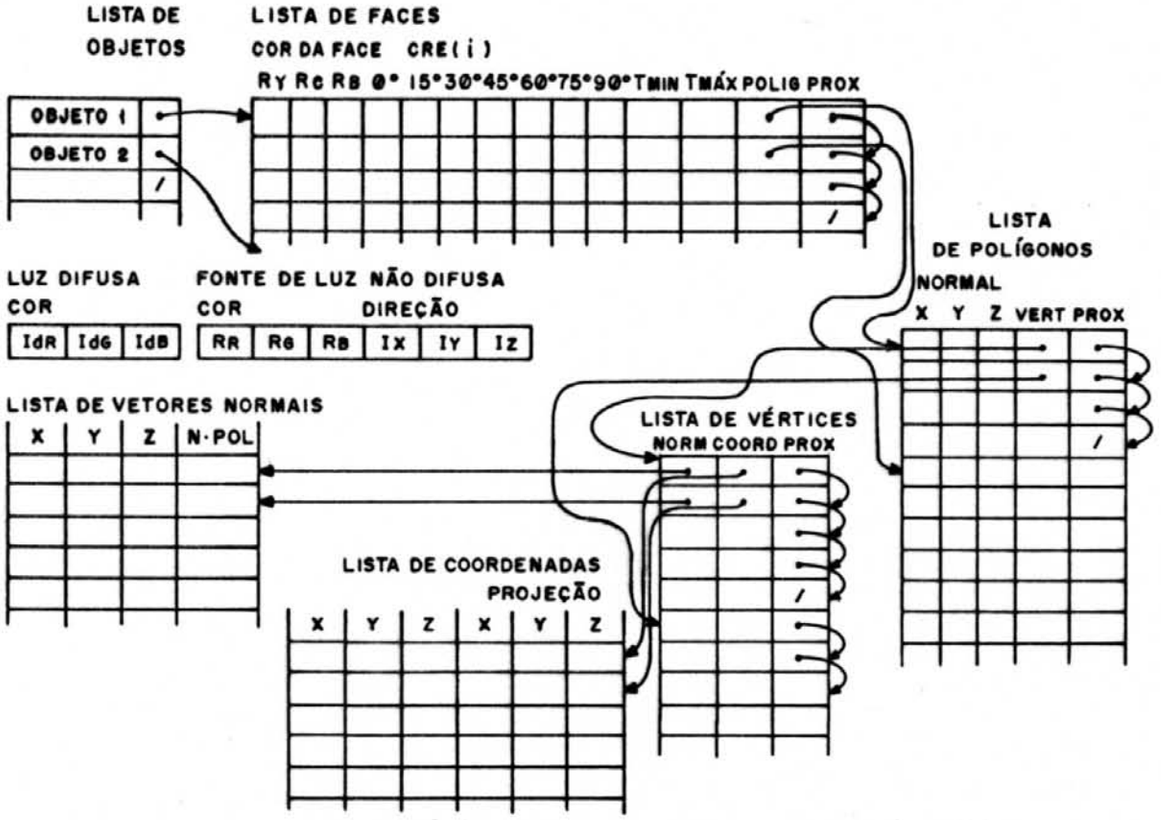


fig. 4.39 - Estrutura de dados usada pelo GIRTD

Uma imagem é composta de objetos. Um objeto é definido pelas suas superfícies. As superfícies possuem atributos que caracterizam o seu comportamento em relação à luz caracterizando o tipo de material de que é feita a superfície, inclusive a função CRE(i) tabelada para valores de i entre 0° e 90°, em passos de 15°. Um objeto pode então ser feito de materiais diferentes, cada um representado por uma superfície.

A curvatura das superficies é aproximada por um conjunto de poligonos. Essa curvatura ficará evidente após esses poligonos serem exibidos com remoção de partes ocultas e sombreamento continuo. Associado a cada poligono está seu vetor normal, que é calculado no GIRTD após cada modelo de objetos que este recebe.

Cada vértice pode ser referenciado por um ou mais poligonos, por isso possuem um apontador para uma lista de vetores normais de poligonos para o cálculo da média entre esses. Os vértices apontam também para uma tabela de coordenadas, para um melhor uso de memória, já que cada posição dessa tabela pode ser referenciada por diversos vértices, o que também possibilita diminuir o tempo de cálculo da mudança de vista e projeção.

4.3.10 - Comentários a respeito dos estágios do pipeline

Para uma correta execução dos poligonos transparentes e do antialiasing estes deveriam entrar no DDA em ordem de profundidade decrescente. Por isso, faz-se necessário um estágio de ordenação de poligonos por algum processo de comparações de sobreposições. Esta é uma tarefa dispendiosa e não pode funcionar na mesma filosofia dos outros estágios, a de trabalhar um poligono ou triângulo por vez, porque precisa dispor de todos os poligonos pertencentes à imagem para a comparação destes entre si. Mesmo assim podemos implementar uma técnica de ordenação que se aproveite da coerência entre uma imagem e outra em uma seqüência de imagens geradas para animação, as quais tem entre si pouca diferença, na suscessão de imagens.

Esse algoritmo é simples e funciona da seguinte forma: uma lista ordenada de n apontadores para poligonos

contêm a seqüência, previamente ordenada, de polígonos, segundo a sua profundidade; a cada geração de imagens compara-se todos os elementos A_i da lista com o o seu sucessor A_{i+1} para ver se este ainda é de menor profundidade; caso não, o elemento A_{i+1} deve ser movimentado para trás, na lista, até que o novo polígono sucessor a este tenha uma profundidade menor. Assim é feito até que se chegue ao fim da lista. Caso não haja alteração na ordem, o número máximo de comparações é n-1.

Uma outra tarefa que deve ser feita antes da projeção cônica é a obtenção de Z_{min} e Z_{max}, que entram nas expressões de conversão de coordenadas da câmera para coordenadas da tela. Para isso deve-se pesquisar todos os valores de Z_e, após a mudança de sistema de referência, e encontrar o maior e o menor valor de Z_e que sejam maiores que C. Isso exige também a investigação de todos os polígonos antes de começar a entregá-los aos estágios seguintes.

Estas tarefas, caso forem implementadas como estágios do pipeline, funcionarão como uma grande FIFO, do tamanho do número de polígonos, que só começará a descarregar polígonos aos estágios seguintes quando já tiver recebido todos do estágio anterior. Isso em termos de pipeline não é nenhum gargalo, apenas se comporta como se fosse um pipeline de comprimento (número de tarefas sendo executadas em paralelo) da ordem do número de polígonos retidos nestes estágios. O número de imagens geráveis por segundo não é prejudicado por isso, apenas ocorre um atraso maior entre o comando, no início do pipeline do GIRTD, e o efeito, na imagem gerada.

5 - CONCLUSOES E MELHORAMENTOS DO GIRTD

5.1 - Conclusões

O objetivo deste trabalho era o de discutir o emprego de arquiteturas especiais de computadores na geração de imagens de objetos tridimensionais em uma tela de computador, tendo como meta principal o projeto de um sistema gráfico para tal, fazendo uso de arquitetura paralela.

O capitulo II introduz a teoria necessária para a compreensão das técnicas e algoritmos empregados na geração de imagens tridimensionais, enquanto que o capitulo III dá uma visão do que tem sido feito nesta busca de melhores desempenhos na geração de imagens de sólidos, através de artigos referentes ao emprego de arquiteturas paralelas.

Para o projeto final foi colocado como meta a obtenção de uma arquitetura que atendesse as aplicações de modelagem e animação de sólidos representados por B-rep, a tempo real. Para tal adotou-se uma estrutura em forma de pipeline, pois as diversas tarefas na geração de imagens por B-rep podem ser dispostas nesta forma, para execução em paralelo. Nesta disposição, o cálculo dos pixels para preenchimento, na memória de vídeo, de faces sombreadas dos objetos ficou para o último estágio do pipeline, o DDA. Para que o sistema atendesse as aplicações de animação de cenas, ficou estabelecido que o DDA deve ser capaz de calcular pixels a uma taxa mínima de 20 Mpixels/segundo. Esta grande demanda torna o DDA um estágio crítico no pipeline. Por esta

razão o enfoque principal deste trabalho foi projetar um DDA para gerar, a cada 50ns em média, pixels para preenchimento dos poligonos.

Esta meta foi atingida com o emprego de uma estrutura especial de hardware contendo dez elementos processadores rodando em paralelo sob o controle de um algoritmo especial para preenchimento de triângulos, semelhante a uma estrutura SIMD. Com isto obtivemos um DDA capaz de suprir as necessidades de um sistema de modelagem de sólidos com respostas instantâneas de imagens gráficas providas de realismo, obtido pelo sombreamento continuo das suas superficies visiveis. A velocidade da montagem das cenas é tal que permite a animação a tempo real de cenas com o mesmo grau de realismo.

Cada elemento de processamento, ou EP, executa operações bastante básicas requerendo para isso um conjunto pequeno de recursos, que são quatro registradores e uma ULA para somas ou subtrações. E um circuito bastante atrativo para uma implementação em um único circuito integrado, ou em PLA, podendo incluir a FIFO de entrada.

A eficiência do DDA se deve especialmente ao algoritmo que controla seus recursos. Tinha-se como ideal um algoritmo que utilizasse um ciclo de clock para a geração de cada pixel interno aos triângulos sem perda de tempo com pixels ou ciclos inválidos, o que para nós significa uma eficiência de 100 %. Também era desejável que o algoritmo percorresse os pixels por saltos unitários com a finalidade de restringir os recursos utilizados por este, evitando o uso de operações de multiplicação. O algoritmo proposto é resultado de um balanceamento entre o desempenho e o custo computacional. Fazendo uso de unidades aritméticas para somas e subtrações e alguns poucos registradores obtêve-se

uma eficiência teórica de 70% (30% de pixels ou ciclos não efetivos), o que pode-se dizer que é um ótimo compromisso entre custo e eficiência.

O pipeline que antecede o DDA deve possuir condições de suportar a demanda imposta por este. Não era proposta deste trabalho projetar cada um dos estágios do pipeline, mas no entanto foi feito um levantamento que determinou que cada um requer no máximo 10 Mflops para poder desempenhar suas tarefas de maneira compatível com o fluxo de dados no pipeline. Isto pode ser obtido por processadores modernos de alto desempenho ou pelo emprego de construções especiais e/ou dedicadas de processamento paralelo. A implementação dos estágios do pipeline não foi levada em conta pois cada uma mereceria um trabalho à parte, embora suas tarefas sejam quase que puramente operações aritméticas.

Da mesma forma, a memória de vídeo deve ser capaz de aceitar a velocidade com que o DDA emite pixels para o seu preenchimento. Sabe-se que esta velocidade é superior àquela suportável pelas memórias, limitada pelo seu grande tempo de acesso. Para compatibilizar estas diferentes velocidades a memória de vídeo foi particionada em 16 partes para acesso simultâneo a 16 pixels. Como o DDA gera somente um pixel por vez, uma memória cache garante a distribuição rápida destes pixels gerados para as 16 posições da memória de vídeo. Estas 16 posições de memória de vídeo foram dispostas de maneira a otimizar o aproveitamento da memória cache, em linhas de 16 pixels vizinhos na horizontal, pois esta é a disposição em que o algoritmo do DDA tem um melhor aproveitamento de utilização da memória cache.

5.2 - Possibilidades de melhoramentos do GIRTD

Uma série de sugestões para melhorar o desempenho e acrescentar novos recursos para o GIRTD são apresentados aqui, valendo como propostas para futuros trabalhos.

5.2.1 - Uso de dados em ponto fixo no pipeline

Um estudo mais aprofundado pode ser aplicado sobre os algoritmos envolvidos no pipeline para a avaliação das possibilidades de se ter operações matemáticas somente sobre dados em ponto fixo. O emprego de aritmética inteira implica em processamento mais rápido sob um custo computacional menor.

5.2.2 - Melhora na velocidade de troca de dados entre memórias VRAM e cache

Na seção 4.2.11.4 foi apresentada a interligação entre a memória cache e a memória de vídeo VRAM, do estágio do DDA. Foi visto que, no melhor caso, tem-se 16 acessos à memória cache para cada 2 acessos à VRAM, um de escrita e outro de leitura. Esse tempo de acesso à VRAM ainda é um fator limitante, uma vez que o acesso à cache e o acesso à VRAM nunca se dão simultaneamente. Um meio de melhorar-se o fator de eficiência do acesso à memória cache é o de sobrepor, no tempo, estes dois acessos, com um conseqüente aumento na complexidade do circuito de controle.

Um primeiro passo seria acrescentar um registrador a mais, em cada célula da cache, para armazenamento temporário do último conteúdo da cache, enquanto a cache

busca dados da VRAM. Após a busca de novos dados, as informações armazenadas nesse registrador são escritas na VRAM, ao mesmo tempo em que o acesso à cache é permitido.

O novo circuito de um bit para memória cache é representado na figura 5.1, incluindo o diagrama de tempos para a troca de dados entre os registradores.

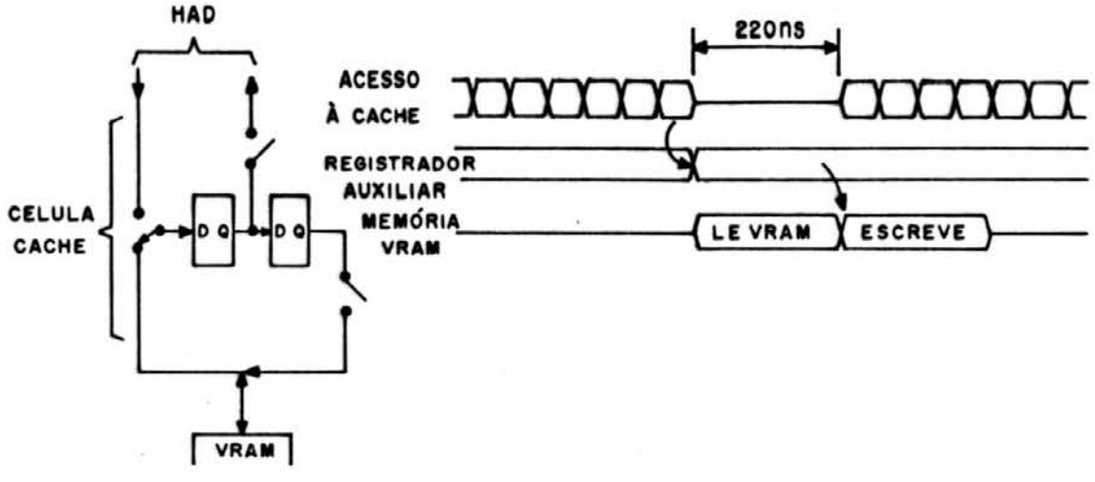


fig. 5.1 - Acréscimo de um registrador auxiliar à cache

Com isso consegue-se um acesso simultâneo à memória cache durante um dos dois ciclos da VRAM, reduzindo-se assim à metade o tempo gasto com a VRAM. Evidentemente alguns cuidados devem ser tomados na implementação para prever casos em que o número de acessos a pixels em uma mesma página é pequeno. Pode ocorrer a solicitação de uma nova página enquanto o registrador auxiliar ainda está escrevendo na VRAM; um mecanismo deve fazer com que a memória cache espere pelo final da escrita na VRAM.

Pode-se agora sobrepor o ciclo de escrita da VRAM com os acessos à cache, com o uso de um registrador auxiliar. Com isso reduziu-se à metade o tempo gasto pela transferência entre a cache e a VRAM. Para ser possível sobrepor também o ciclo de leitura teríamos que ler anteci-

padamente da VRAM uma página de 16 bits, antes que essa fosse referenciada pelo HAD, e colocar este dado em mais um registrador auxiliar. Prever a próxima página não é impossivel pois existem registradores pipeline no HAD para o endereço do pixel e estes contêm os próximos endereços. O problema é determinar em que nivel da cadeia de registradores pipeline capturar esta nova página pois estes proximos endereços podem estar ou não dentro da mesma página. Uma maneira garantida de se prever uma nova página e de se ter tempo para lê la antes que sua referência chegue na cache, é capturá-la em um registrador pipeline tantos niveis antes quantos pulsos de clock cabem num ciclo de acesso à VRAM. A lógica de controle fica bastante complexa mas consegue-se quase 100% de acessos à cache sem clocks de espera à VRAM.

5.2.3 - Instalação de um cursor 3-D

Um recurso que pode ser bastante interessante para a interpretação da profundidade da imagem é o uso de um cursor 3-D, conforme sugere o artigo [FUJ 84]. Este pode ser um plano, continuo ou em forma de grade, controlado interativamente, que atravessa a imagem gerada, tendo suas partes obscurecidas caso estas atravessem os objetos. Isso pode ser acrescentado ao GIRTD sem grandes custos, com o acréscimo de um plano a mais na memória de vídeo para armazenar este cursor. Este plano terá, para cada pixel da imagem, seu bit indicando a condição de cursor visível ou não. O HAD poderá ser chaveado para atualizar a memória de vídeo ou somente o plano do cursor. O traçado do cursor usará os mesmos recursos do DDA e HAD sem atualizar o z-buffer, apenas lendo-o para decidir se o cursor está dentro ou fora dos objetos da imagem. Também não terá sombreamento e nem antialiasing.

Este cursor poderá movimentar-se através dos objetos sem a necessidade de reexibir-se a imagem da tela, pois esta permanece intacta na memória.

5.2.4 - Visualização wire-frame

Em algumas aplicações pode-se desejar aproveitar os recursos de remoção de faces ocultas do GIRTD para gerar imagens na forma de wire-frame. Isso pode ser feito facilmente alterando-se a função NORM, apresentada no item 4.2.5.1 e usada no cálculo do antialiasing, para um comportamento sugerido na figura 5.2.

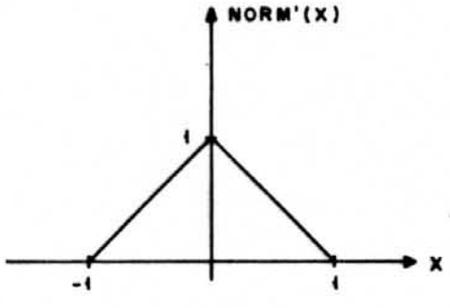


fig. 5.2 - Função NORM para wire-frame

Assim somente as bordas dos polígonos são desenhadas, aproveitando-se ainda do recurso de antialiasing. Apenas deve-se ter o cuidado de implementar-se também um meio de marcar aquelas arestas criadas pelo triangularizador, que não devem ser exibidas, e um meio de serem detectadas pelo DDA para um tratamento diferenciado. Para que ocorra a remoção de partes ocultas na geração de uma imagem em forma de wire-frame, os valores R, G e B que o HAD lê da memória de vídeo para a dosagem da transparência final devem, ao

invés disso, ser lidos de um registrador fixo que contenha a mesma informação de cor de fundo.

5.2.5 - Problemas com a técnica de sombreamento de Gouraud

A técnica de sombreamento usada no GIRTD é a sugerida por Gouraud [GOU 71], onde a luminosidade é interpolada a partir dos vetores normais nos vértices, calculados pela média das normais dos seus polígonos. Esta técnica, embora simples, apresenta algumas deficiências. Quando usada em uma seqüência de movimento, o sombreamento parace mudar de maneira irregular. Isto ocorre porque a interpolação é feita na superfície da tela, ao invés da superfície do objeto.

Outra desvantagem do sombreamento de Gouraud é a introdução do efeito "Mach band", um efeito causado pela capacidade da visão humana de perceber descontinuidades na primeira derivada do sombreamento. Estas descontinuidades ocorrem devido à interpolação linear dentro de cada polígono e por não haver suavização da derivada de um polígono para outro. Isto pode ser solucionado fazendo-se interpolação quadrática com um DDA que avalie diferenças finitas de segunda ordem.

Na figura 5.3 está ilustrado um outro problema decorrente do fato de calcularmos as normais por média. Neste exemplo, apesar dos polígonos não serem paralelos, estes terão um sombreamento igual, ao longo destes.

Tanto esse problema como os outros citados acima podem ser amenizados com o aumento da precisão da descrição, implicando no custo de descrever-se as superfícies com mais polígonos.

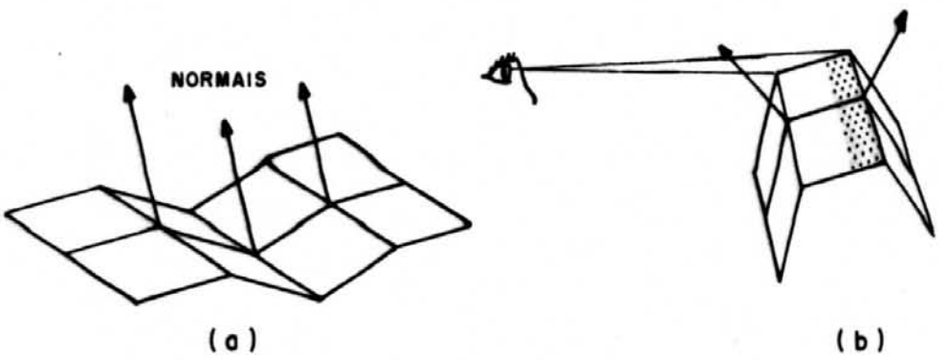


fig. 5.3 - Anomalias do sombreamento causadas pela determinação das normais por média

Phong sugere uma técnica de sombreamento que minimiza alguns dos problemas inerentes à técnica de Gouraud [NEW 79]. Nesta técnica a interpolação é aplicada aos vetores normais, em lugar de interpolar-se a luminosidade calculada nos vértices. Isso pode ser feito de forma semelhante à implementação do DDA sugerida neste trabalho. As três componentes do vetor normal são determinadas incrementalmente. O efeito de luminosidade deve ser calculado a partir desse vetor normal calculado pelo DDA, o que requer um HAD mais complexo.

5.2.6 - Estudo da possibilidade de geração de sombras

O próximo recurso de realismo que poderia desejar-se, a partir dos já existentes no GIRTD, é a geração de sombras. As sombras em uma imagem 3-D são efeitos bastante desejados em simulação realística pois intensificam enormemente a percepção de profundidade. Estas podem ser causadas por mais de uma fonte de luz.

Em um sistema inicialmente baseado em remoção de partes ocultas por z-buffer é difícil a inclusão de tal recurso. Brotman [BRO 84] sugere um algoritmo baseado em z-buffer para geração de sombras. Para este algoritmo são inseridos mais planos de memória na imagem para a representação de ponteiros para estruturas que facilitarão a determinação das porções da imagem que devem ser escurecidas. E uma técnica interessante mas exige demoradas pesquisas e cálculos nestas estruturas.

No projeto "pixel-planes" [FUC 85], apresentado no capítulo anterior, a remoção de partes ocultas também é por algoritmo Z-buffer, e o sistema se aproveita da persistência da informação de profundidade Z, em cada pixel, para a geração de sombras. No GIRTD esta mesma técnica é difícil de ser implementada, ou muito lenta, pois para a determinação dos pixels sombreados, todos os pixels da imagem devem ser testados, para cada polígono, como é feito , simultaneamente para cada pixel, no array do "pixel-planes". O algoritmo do DDA do GIRTD determina um pixel de cada vez, dentro de uma área esperada ser a área resultante, que deve ser convexa e conexa. A área de sombreado pode não ser convexa e ainda ser desconexa e o DDA não tem condições de determinar eficientemente os pixels desta área.

5.2.7 - O uso de processamento genérico local

Uma tendência bastante forte, em estações gráficas de trabalho, é a migração do processamento para execução local nas próprias estações. Isso libera o computador central de muito processamento e alivia as redes de comunicação pois informações gráficas implicam normalmente em troca de enorme quantidade de dados. Cada vez mais o computador central torna-se dedicado somente à gerência do

banco de dados enquanto que as estações gráficas executam localmente as mais diversas e pesadas tarefas tais como cálculo de elementos finitos, checagem de regras de formação, simulação lógica, cálculo de superficies, spline, graftals, fractals, etc. Atualmente já assume-se que uma estação gráfica moderna deva ter localmente um poder de processamento de pelo menos 10Mflops.

Acompanhando a tendência, sugere-se a intercalação entre o pipeline e o computador hospedeiro de um processador de alto desempenho, para que o hospedeiro não venha a tornar-se o gargalo do sistema. Esse processador armazenará e manipulará toda a estrutura de dados que descreve o universo, além de poder executar rotinas emitidas pelo hospedeiro com alta velocidade.

Um exemplo de um procesador adequado para esse fim é o obtido pelo emprego de chips da Weitek, onde consegue-se um endereçamento praticamente ilimitado de dados e código de 32 bits, com barramentos separados e concorrentes para dados e programa, e que pode atingir um desempenho de 10 Mflops com circuitos de ponto flutuante.

E interessante que este processador tenha acesso a todos os estágios do pipeline para que possa executar tarefas em qualquer nivel, inclusive o preenchimento pixel a pixel da tela.

5.2.8 - Textos na tela gráfica

O GIRTD é um equipamento dedicado. Seus propósitos são unicamente a visualização de imagens 3-D com um certo grau de realismo e velocidade. Ele é portanto inconveniente

para, por exemplo, ser empregado para imagens com saída de textos.

Para que seu uso não fosse tão restrito poder-se-ia colocar em paralelo com o pipeline um controlador gráfico moderno, tal como o 82786 da Intel ou o 34010 da Texas, para aplicações mais genéricas, sem que isto represente um aumento significativo no custo relativo. Este poderia ter a sua própria memória de maneira que seus dados se sobreponham aos da imagem 3-D sem alterá-la. Assim consegue-se, por exemplo, colocar-se textos de referência sobre a imagem realística gerada ou fazer-se um desenho gráfico dos controles e indicadores de um painel de avião, quando empregado como simulador de vôo.

BIBLIOGRAFIA

[BEN 85] BENNET, J. Raster Operations BYTE, Peterborough , 10(12):187-203, Nov. 1985.

[BRO 84] BROTMAN, L. S. Generating Soft Shadows with a Deep Buffer Algorithm IEEE Computer Graphics & Aplications, Los Alamos, 4(10):5-12, Oct. 1984.

[BUR 87] BURGOON, D. Pipelined Graphics Engine Speeds 3-D Image Control Electronic Design, Rochelle Park ,35(17):143-50 July 1987.

[DEM 85] DEMETRESCUL S. Moving Pictures BYTE, Peterborough, 10(12):207-17, Nov. 1985.

[ENG 86] ENGLAND, N. A Graphic System Architeture for Interactive Application-Specific Display Functions IEEE Computer Graphics & Aplications, Los Alamos, 6(1):60-70, Jan. 1986.

[FUC 85] FUCHS, H. et alii. Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixels-Planes Computer Graphics, New York, 19 (3):111-20, July 1985.

[FUJ 83] FUJIMOTO, A. & IWATA, K. Jag-Free Image on Raster Displays IEEE Computer Graphics & Aplications, Los Alamos, 3(9):26-34, Dec. 1983.

[FUJ 84] FUJIMOTO, A.; PERROT, C. G.; IWATA, K. A 3-D Graphics Display System wuth Deep Buffer and Pipeline Processor IEEE Computer Graphics & Aplications, Los Alamos, 4(6):11-23, June 1984.

[GOL 86] GOLDFEATHER, J. et alii. Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffer Memory IEEE Computer Graphics & Aplications, Los Alamos, 6(1):48-59, Jan. 1986.

[GOU 71] GOURAUD, H. Continuous Shading of Curved Surfaces IEEE Transactions on Computers, New York, C20(6):623-9, June 1971.

[GUT 86] GUTTAG, K. et alii. Requeriments for a VLSI Graphic Processor IEEE Computer Graphics & Aplications, Los Alamos, 6(1):32-47, Jan. 1986.

[HEA 86] HEARN, D. & BAKER, M. P. Computer Graphics London, Prentice-Hall, 1986.

[HWA 84] HWANG, K. & BRIGGS, F. A. Computer Architecture and Parallel Processing, New York, McGraw-Hill Book Company, 1984

[IKE 84] IKEDO, T. High-Speed Techniques for a 3-D Color Graphics Terminal IEEE Computer Graphics & Aplications, Los Alamos, 4(5):46-58, May 1984.

[JEF 85] JEFFERY, T. The uPD7281 Processor BYTE, Peterborough, 10(12):237-246, Nov. 1985.

[MOK 87] MOKHOFF, N. Custom VLSI Chips soup up Graphics Engines Electronic Design, Rochelle Park, 35(17):33-35, July 1987.

[NEW 79] NEWMAN, W. M. & SPROUL, R. F. Principles of interactive computer graphics 2. ed. Auckland, McGraw-Hill, 1979.

[NII 84] NIIMI, H. et alii. A Parallel Processor System for Three-Dimensional Color Graphics Computer Graphics, New York, 18(3):67-76, July 1984.

[SAT 85] SATO, H. et alii. Fast Image Generation of Construtive Solid Geometry Using a Cellular Array Processor, Computer Graphics, New York, 19(3):95-102, July 1985.

[SIE 82] SIEWIOREK, D. P. et alii. Computer Structures: Principles and Examples, Auckland, McGraw-Hill, 1982

[SUT 74] SUTHERLAND, I.; SPROULL, R. F.; SCHUMACKER, R. A. A caracterization of Ten Hidden-Surface Algorithms Computing Surveys, New York, 6(1):1-55, Mar. 1974.

[TUN 87] TUNICK, D. Powerful Display System revs up Image and Graphics Processing Electronic Design, Rochelle Park, 35(17):58-62, July 1987.

[WOR 83] WORDENWEBER, B. Surface Triangulation for Picture Production IEEE Computer Graphics & Aplications, Los Alamos, 3(8):45-51, Nov. 1983.

ANEXO: Validação dos algorítmos do GIRTD

VALIDAÇÃO DOS ALGORÍTMOS DO GIRTD

Este anexo apresenta o pacote gráfico utilizado na demonstração dos recursos do GIRTD e na validação do algorítmo do DDA. As rotinas aqui escritas, implementadas para execução em IBM-PC, perfazem a função dos três últimos estágios do pipeline do GIRTD - o recorte, o preparador do DDA e o DDA propriamente. O recorte e o preparador do DDA foram incluídos pra facilitar a comunicação com o DDA, pois seus parâmetros fazem mais sentido do que os parâmetros do DDA. O preparador do DDA, representado pela rotina "PREPDDA(p1,p2,p3)" e que já inclui o recorte, recebe do programa principal a descrição dos três vértices do triângulo a ser preenchido na imagem. Esta descrição inclui as coordenadas X e Y, a profundidade Z, as componentes de cor R, G e B, e a transparência T, de cada vértice. O preparador do DDA se encarrega então de calcular as derivadas da cor, da profundidade, da tranparência e das distâncias, bem como seus valores iniciais para o disparo da iteração do algorítmo de preenchimento do DDA.

A própria rotina do preparador do DDA chama o algorítmo DDA e entrega a este parâmetros - as derivadas e as condições iniciais - para seu correto funcionamento. O DDA então calcula pixels incrementalmente, segundo o algorítmo DDA apresentado na figura 4.2 do capítulo IV, para preenchimento sombreado, com antialiasing, transparência e remoção de pixels ocultos por Z-buffer, de triângulos sobre a tela gráfica.

Para a apresentação dos pixels calculados pelo DDA, na falta de um sistema de alta resolução que permita apresentar a vasta gama de cores necessária para um bom desempenho do sombreamento, utilizou-se uma técnica chamada "halftoning" que permite aumentar a gama de cores disponível em um sistema de alta resolução, com uma consequente e preoporcional redução da resolução fornecida pelo sistema. Esta técnica utiliza um grupo de pixels próximos para formar um pixel quadrado maior composto de pixels básicos. As intensidades de cores itermediárias são obtidas pelas diferentes configurações de pixels acesos e apagados deste pixel composto, de maneira a se obter tantas intensidades de cor diferentes quantos pixels que compõem o pixel maior.

Para tal utilizou-se a placa ARTIST, baseada em controlador de vídeo NEC 7220, que dispõe de uma resolução de 1024x768 pixels, podendo exibir simultaneamente 16 cores de um conjunto de 4096 cores. Dessas 16 cores utilizamos 8, uma para cada combinação de intensidades máximas e mínimas de cada componente de cor R, G e B. Com pixels compostos por 64 pixels básicos, dispostos em matriz 8x8, obtém-se 65 níveis de cor para cada componente, resultando em um total de 274.000 cores possíveis simultaneamente. Com a utilização

de pixels de dimensão 8 nas direções X e Y a resolução fica reduzida para 128x96, que é bastante pequena para a representação realística de imagens, mas, uma vez que o objetivo deste pacote é de demonstrar o correto funcionamento do DDA, atende perfeitamente os objetivos.

A exibição de pixels compostos, na placa ARTIST, é relativamente lenta - por volta de 40 pixels/segundo, de maneira que é possível acompanhar o andamento do algorítmo de preenchimento do DDA. O tempo de exibição está limitado pelo número de pixels da imagem, consequentemente a exibição de uma imagem relativamente complexa pode levar de 5 a 60 segundos.

As configurações de pixels acesos e apagados na matriz 8x8, para cada intensidade entre 0 e 64, foram cuidadosamente escolhidas para que a repetição suscessiva destes, ao longo do plano da tela, introduza o mínimo possível de sensação de um padrão repetitivo ou de linhas paralelas.

Os programas foram divididos em dois módulos, um escrito em linguagem "C" e outro escrito em assembly do 8088. Os algorítmos de recorte, preparação do DDA e o proprio DDA estão escritos em "C" enquanto que outras rotinas críticas, para uma maior eficiência foram escritas em assembly, tais como a remoção de partes ocultas, o preenchimento dos pixels compostos e a inicialização da placa ARTIST, entre outras que também fazem acesso direto as portas de I/O do controlador 7220.

```
#define Pix_Interno ( PIX_INTERNO())
#define Pix_Externo (!PIX_INTERNO())

int X, Y;
long R, G, B, T, Z, D1, D2, D3;
int Xmax, Ymax;
long RX, GX, BX, TX, ZX, D1X, D2X, D3X;
int Xmin, Ymin;
long RY, GY, BY, TY, ZY, D1Y, D2Y, D3Y;
int Xaux, Yaux;
long Raux, Gaux, Baux, Taux, Zaux, D1aux, D2aux, D3aux;
int QQ;
char R_BUF[0x3000], G_BUF[0x3000], B_BUF[0x3000];
int  Z_BUF[0x3000];
struct {
        int R;
        int G;
        int B;
        int Nf;
        int Z;
        int X;
        int Y;
        } REG;

typedef struct
        {
        float x;
        float y;
        float z;
        float r;
        float g;
        float b;
        float t;
        } ponto;

const float E16=65536.0;
const float E24=16777216.0;
const float E32=4.2949E+9;
const float KC=63.5;
const float UM=1.0;
const float TQ=0.75;

double sqrt(double x);
long ftol(float f);
```

```
/*********************************************************
 *                      PREPDDA                          *
 *                                                       *
 *     Esta rotina executa a funcao do estagio do *
 * pipeline que prepara os parametros do DDA. Sao *
 * calculadas as derivadas e os pontos iniciais das *
 * informacoes R, G, B, T, Z, D1, D2 e D3, para *
 * disparar o preenchimento do triangulo, por parte *
 * do DDA. Tem como argumentos de entrada os tres *
 * pontos que definem o triangulo, contendo em cada *
 * um as informacoes dos seus atributos R, G, B, T e *
 * Z, e os utiliza para o calculo de suas derivadas *
 * em relacao a X e Y.                                   *
 *********************************************************/
prepdda(p1,p2,p3)
ponto *p1, *p2, *p3;
{
float pv, dx, dy, di, x21, x31, y21, y31, xs1, ys1;
float px, py, pp, max, min;
float p1x,p2x,p3x,p1y,p2y,p3y;
float d1x,d1y,d1,d2y,d2x,d2,d3y,d3x,d3;
float xs, ys;
int q;

/* converte para as coordenadas de tela */
p1x=KC*(UM + p1->x);
p2x=KC*(UM + p2->x);
p3x=KC*(UM + p3->x);
p1y=KC*(TQ - p1->y);
p2y=KC*(TQ - p2->y);
p3y=KC*(TQ - p3->y);

/* calcula o produto vetorial do triangulo */
pv = (p2x-p1x)*(p3y-p1y)-(p3x-p1x)*(p2y-p1y);
if (pv==0) return(0);
q=(pv)0)?0:-1;
pv=1/pv; /* reciproca do produto vetorial */

/* calcula derivadas e valor inicial de D1 */
dx = p1y - p2y;
dy = p2x - p1x;
di=sqrt(dx*dx+dy*dy);
if (di==0)
    {
    d1x=0;
    d1y=0;
    d1=UM;
    }
else
    {
    di=1/di;
    d1x=di*dx;
    d1y=di*dy;
    d1=-d1x*p1x-d1y*p1y;
```

```
    if (q)
        {
        d1x=-d1x;
        d1y=-d1y;
        d1=-d1;
        }
    }
/* calcula derivadas e valor inicial de D2 */
dx = p2y - p3y;
dy = p3x - p2x;
di=sqrt(dx*dx+dy*dy);
if (di==0)
    {
    d2x=0;
    d2y=0;
    d2=UM;
    }
else
    {
    di=1/di;
    d2x=di*dx;
    d2y=di*dy;
    d2=-d2x*p2x-d2y*p2y;
    if (q)
        {
        d2x=-d2x;
        d2y=-d2y;
        d2=-d2;
        }
    }

/* calcula derivadas e valor inicial de D3 */
dx = p3y - p1y;
dy = p1x - p3x;
di=sqrt(dx*dx+dy*dy);
if (di==0)
    {
    d3x=0;
    d3y=0;
    d3=UM;
    }
else
    {
    di=1/di;
    d3x=di*dx;
    d3y=di*dy;
    d3=-d3x*p3x-d3y*p3y;
    if (q)
        {
        d3x=-d3x;
        d3y=-d3y;
        d3=-d3;
        }
    }
```

```
/* A seguir e' executado o processo de recorte,
que nada mais e' do que a determinacao do ponto
de partida Xs, Ys para a iteracao inicial do DDA. */

/* Inicialmente e' testado o ponto de menor Y. Se
este for interno ao triangulo sera' o ponto de
partida */
ys=p1y;
xs=p1x;
if (p2y < ys)
    {
    ys=p2y;
    xs=p2x;
    }
if (p3y < ys)
    {
    ys=p3y;
    xs=p3x;
    }

/* se o menor y esta' abaixo da tela, e'
   um triangulo invisivel */
if (ys>95) return(0);

/* e se for interno `a janela, este
   ponto e' escolhido como o ponto de
   partida */
if (xs>=0 && xs<=127 && ys>=0) goto recorteok;

/* verifica se os pontos (0,0) ou (127,0) sao
   internos ao triangulo. Caso algum deles seja
   e' considerado como ponto de partida */
ys=0; xs=0;
if (d1>=0 && d2>=0 && d3>=0) goto recorteok;
xs=127;
if ((d1+d1x*xs)>=0 &&
    (d2+d2x*xs)>=0 &&
    (d3+d3x*xs)>=0) goto recorteok;

/* Testa interseccao entre arestas e o limite
   superior da tela Y=0 ; 0<X<128 */
ys=0;
if (d1x!=0)
    {
    xs=-d1/d1x;
    if ( (d2x*xs+d2)>=0 &&
         (d3x*xs+d3)>=0 &&
         xs>0 && xs<128 ) goto recorteok;
    }
if (d2x!=0)
    {
    xs=-d2/d2x;
    if ( (d1x*xs+d1)>=0 &&
```

```
          (d3x*xs+d3)>=0 &&
          xs>0 && xs<128 ) goto recorteok;
     }
if (d3x!=0)
     {
     xs=-d3/d3x;
     if ( (d1x*xs+d1)>=0 &&
          (d2x*xs+d3)>=0 &&
          xs>0 && xs<128 ) goto recorteok;
     }

/* Testa interseccao com os limites laterais
   da tela. */
ys=96;
px=0;
if (d1y!=0)
     {
     py=-d1/d1y;
     if ( (d2y*py+d2)>=0 &&
          (d3y*py+d3)>=0 &&
          py>0 && py<95 && ys>py)
        {
        ys=py;
        xs=px;
        }
     }
if (d2y!=0)
     {
     py=-d2/d2y;
     if ( (d1y*py+d1)>=0 &&
          (d3y*py+d3)>=0 &&
          py>0 && py<95 && ys>py)
        {
        ys=py;
        xs=px;
        }
     }
if (d3y!=0)
     {
     py=-d3/d3y;
     if ( (d1y*py+d1)>=0 &&
          (d2y*py+d2)>=0 &&
          py>0 && py<95 && ys>py)
        {
        ys=py;
        xs=px;
        }
     }
px=127;
if (d1y!=0)
     {
     py=-(d1+d1x*px)/d1y;
     if ( (d2x*px+d2y*py+d2)>=0 &&
          (d3x*px+d3y*py+d3)>=0 &&
```

```
          py>0 && py<95 && ys>py)
        {
        ys=py;
        xs=px;
        }
    }
if (d2y!=0)
    {
    py=-(d2+d2x*px)/d1y;
    if ( (d1x*px+d1y*py+d1)>=0 &&
         (d3x*px+d3y*py+d3)>=0 &&
          py>0 && py<95 && ys>py)
        {
        ys=py;
        xs=px;
        }
    }
if (d3y!=0)
    {
    py=-(d3+d3x*px)/d1y;
    if ( (d1x*px+d1y*py+d1)>=0 &&
         (d2x*px+d2y*py+d2)>=0 &&
          py>0 && py<95 && ys>py)
        {
        ys=py;
        xs=px;
        }
    }
if (ys>95) return(0);
ys-=1;

recorteok:
/* converte coeficientes das equacoes das
   distancias para dados do tipo long */
D1 =ftol(E16*(d1+d1x*xs+d1y*ys));
D1X=ftol(E16*d1x);
D1Y=ftol(E16*d1y);
D2 =ftol(E16*(d2+d2x*xs+d2y*ys));
D2X=ftol(E16*d2x);
D2Y=ftol(E16*d2y);
D3 =ftol(E16*(d3+d3x*xs+d3y*ys));
D3X=ftol(E16*d3x);
D3Y=ftol(E16*d3y);

x21=pv*( p2x - p1x );
y21=pv*( p2y - p1y );
x31=pv*( p3x - p1x );
y31=pv*( p3y - p1y );
xs1=(xs - p1x );
ys1=(ys - p1y );

/* calcula derivadas e valor inicial de R */
px=(p2->r-p1->r)*y31-(p3->r-p1->r)*y21;
py=(p3->r-p1->r)*x21-(p2->r-p1->r)*x31;
```

```
pp=p1->r+px*xs1+py*ys1;
RX=ftol(E24*px);
RY=ftol(E24*py);
R=ftol(E24*pp);

/* calcula derivadas e valor inicial de G */
px=(p2->g-p1->g)*y31-(p3->g-p1->g)*y21;
py=(p3->g-p1->g)*x21-(p2->g-p1->g)*x31;
pp=p1->g+px*xs1+py*ys1;
GX=ftol(E24*px);
GY=ftol(E24*py);
G=ftol(E24*pp);

/* calcula derivadas e valor inicial de B */
px=(p2->b-p1->b)*y31-(p3->b-p1->b)*y21;
py=(p3->b-p1->b)*x21-(p2->b-p1->b)*x31;
pp=p1->b+px*xs1+py*ys1;
BX=ftol(E24*px);
BY=ftol(E24*py);
B=ftol(E24*pp);

/* calcula derivadas e valor inicial de T */
px=(p2->t-p1->t)*y31-(p3->t-p1->t)*y21;
py=(p3->t-p1->t)*x21-(p2->t-p1->t)*x31;
pp=p1->t+px*xs1+py*ys1;
TX=ftol(E16*px);
TY=ftol(E16*py);
T=ftol(E16*pp);

/* calcula derivadas e valor inicial de Z */
px=(p2->z-p1->z)*y31-(p3->z-p1->z)*y21;
py=(p3->z-p1->z)*x21-(p2->z-p1->z)*x31;
pp=p1->z+px*xs1+py*ys1;
ZX=ftol(E32*px);
ZY=ftol(E32*py);
Z =ftol(E32*pp);

/* pontos iniciais para iteracao */
X=xs;
Y=ys;

/* determina valores minimos e maximos de X e Y */
if (p1x > p2x) max=p1x; else max=p2x;
if (max < p3x) max=p3x; max++;
if (max > 127) max=127;
if (p1x < p2x) min=p1x; else min=p2x;
if (min > p3x) min=p3x; min--;
if (min < 0) min=0;
Xmax=max;
Xmin=min;
if (p1y > p2y) max=p1y; else max=p2y;
if (max < p3y) max=p3y; max++;
if (max > 95) max=95;
Ymax=max;
```

```
/* passa o controle ao DDA */
DDApix();
return(1);
}

/*************************************
 * esta rotina serve  para  corrigir *
 * um defeito na conversao de  float *
 * para long,  na  atual  versao  do *
 * library da Borland - para numeros *
 * negativos da' errado!             *
 *************************************/
long ftol(float f)
{
if (f<0) return(-(long)(-f));
    else return( (long) (f));
}


/*************************************************************
 *                          DDA                              *
 *                                                           *
 *      Esta rotina desempenha exatamente o algoritmo        *
 * do DDA, conforme descrito na fig. 4.2. As  opera-         *
 * coes aritmeticas sao todas sobre numeros em ponto         *
 * fixo e somente somas e subtracoes. Os registrado-         *
 * res das EPs sao posicoes de memoria de  32  bits,         *
 * onde os 16 menos significativos sao fracionarios.         *
 * As operacoes paralelas das EPs sao  representadas         *
 * pelas    rotinas   Inc_X(),   Inc_Y(),   Dec_X(),         *
 * Mov_Pix_Aux()  e  Mov_Aux_Pix(),  descritas  mais         *
 * adiante. Algumas rotinas foram definidas no modu-         *
 * lo em assembler - ZBUF.ASM - para maior  eficien-         *
 * cia; sao o ATU_DSP, que atualiza um pixel na ima-         *
 * gem conforme a descricao do HAD; o DIR,  que  de-         *
 * volve a direcao a tomar, segundo a secao 4.2.4; e         *
 * PIXEL_INTERNO, que devolve a  condicao  de  pixel         *
 * interno ao triangulo.                                     *
 *      OBS: E' impossivel implementar este algori-          *
 * tmo sem o emprego de GOTOs                                *
 *************************************************************/
DDApix()
{
if Pix_Interno
    {
lb1:
    Mov_Aux_Pix();
    do  {
        ATU_DSP();
        Inc_X();
        if (X)Xmax) break;
        } while Pix_Interno;
    Mov_Pix_Aux();
```

```
    do  {
        Dec_X();
         if (X<Xmin) break;
         if Pix_Interno ATU_DSP();
             else break;
         } while (1);
    }
lb2:
Inc_Y();
if (Y>Ymax) goto fim;
if Pix_Interno goto lb1;
if (DIR())
    {
    do  {
        Dec_X();
        if (X<Xmin) goto fim;
        } while Pix_Externo;
    Mov_Aux_Pix();
    do  {
        ATU_DSP();
        Dec_X();
        if (X<Xmin) break;
        } while Pix_Interno;
    }
else
    {
    do  {
        Inc_X();
        if (X>Xmax) goto fim;
        } while Pix_Externo;
    Mov_Aux_Pix();
    do  {
        ATU_DSP();
        Inc_X();
        if (X>Xmax) break;
        } while Pix_Interno;
    }
Mov_Pix_Aux();
goto lb2;
fim:
}

/*************************************************************
 *      As seguinte rotinas sao utilizadas pelo              *
 * algoritmo do DDA. Estas representam as operacoes          *
 * paralelas de transferencias de dados e aritmeticas        *
 * que ocorrem simultaneamente nas EPs.                      *
 *************************************************************/

Mov_Aux_Pix()
{
Raux=R; Gaux=G; Baux=B; Taux=T;
Xaux=X; Yaux=Y; Zaux=Z;
D1aux=D1; D2aux=D2; D3aux=D3;
```

```
}

/*****************************************/

Mov_Pix_Aux()
{
R=Raux; G=Gaux; B=Baux; T=Taux;
X=Xaux; Y=Yaux; Z=Zaux;
D1=D1aux; D2=D2aux; D3=D3aux;
}

/*****************************************/

Inc_X()
{
X++;
R+=RX; G+=GX; B+=BX; Z+=ZX;
D1+=D1X; D2+=D2X; D3+=D3X; T+=TX;
}

/*****************************************/

Dec_X()
{
X--;
R-=RX; G-=GX; B-=BX; Z-=ZX;
D1-=D1X; D2-=D2X; D3-=D3X; T-=TX;
}

/*****************************************/

Inc_Y()
{
Y++;
R+=RY; G+=GY; B+=BY; Z+=ZY;
D1+=D1Y; D2+=D2Y; D3+=D3Y; T+=TY;
}

/*************************************************************
 *     As seguintes rotinas sao interfaces para o uso        *
 * das primitivas graficas por  parte  dos  programas        *
 * escritos em "C".                                          *
 *************************************************************/

/* desenha um pixel 8x8 diretamente na tela */

Des_Pix_8x8(r, g, b, x, y)
int r, g, b, x, y;
{
REG.R=r; REG.G=g; REG.B=b;
REG.X=x; REG.Y=y;
DESPIX8X8();
}
```

```
/* desenha um pixel 8x8 passando pelo teste de
   profundidade e utilizando a transparencia T */

Had_Pix(r, g, b, t, z, x, y)
int r, g, b, t, z, x, y;
{
REG.R=r; REG.G=g; REG.B=b;
REG.X=x; REG.Y=y; REG.Z=z;
REG.Nf=t;
HADPIX();
}

/* preenche a memoria de video com a cor
especificada em R, G e B, e inicia o Z-Buffer
com a profundidade maxima */

Fill_Mem(r, g, b)
int r, g, b;
{
REG.R=r; REG.G=g; REG.B=b;
FILLMEM();
}
```

```
        name    zbuf
        page 60,132
;*****************************************************************
;                           ZBUF.ASM
;
;       Esta listagem apresenta as rotinas do pacote grafico
;de demosntracao do GIRTD que foram implementadas em
;assembler do 8088 para uma maior eficiencia. Sao as rotinas
;mais basicas, que fazem o acesso aos enderecos de I/O da
;placa grafica ARTIST, cujo controlador e' o 7220, da NEC.
;       Aqui e' simulado o HAD, descrito no capitulo IV, onde
;sao calculados a transparencia final, seu efeito sobre as
;componentes R, G e B da imagem, e a remocao de pixels
;ocultos pelo teste de profundidade. Para tal e' mantido uma
;area de memoria para armazenar os valores re R, G, B e
Z,
;de tamanho suficiente para represetar todos os pixels da
;imagem.
;       A placa ARTIST, nesta configuracao, possiu
ate'
;1024x768 pixels enderecaveis, embora somente com 4 planos
;de cores, limitando a exibicao ate' 16 cores simultaneas.
;Para poder demostrar o sombreamento e o antialiasing do DDA
;optou-se por utilizar-se a tecnica de "halftoning" para a
;obtencao de uma mair gama de cores. Nesta tecnica utiliza
;se pixels maiores, constituidos de um conjunto de pixels
;basicos. Neste pacote os pixels sao representados por
;pixels maiores de 8x8 pixels básicos, ou 64. Acendendo-se,
;para cada componente R, G e B independentemente, de 0 a 64
;pixels basicos deste pixel maior, pode-se controlar a
;intensidade em 65 niveis de cor, para cada componente,
;perfazendo um total de 256k cores. Embora a resolucao fique
;reduzida para 128x96, podemos demonstrar os recursos de
;sombreamento, transparencia e antialiasing de triangulos
;oferecidos pelo DDA.

;A maior parte das rotinas recebe parametros apontados pelo
;registrador SI, segundo "offsets" pre-estabelecidos, de
;acordo com a tabela abaixo:
word_R equ      word ptr [si]
word_G equ      word ptr [si+2]
word_B equ      word ptr [si+4]
word_Nfequ      word ptr [si+6]
word_Z equ      word ptr [si+8]
word_X equ      word ptr [si+10]
word_Y equ      word ptr [si+12]
```

```
byte_Rlequ      byte ptr [si]
byte_Rhequ      byte ptr [si+1]
byte_Glequ      byte ptr [si+2]
byte_Ghequ      byte ptr [si+3]
byte_Blequ      byte ptr [si+4]
byte_Bhequ      byte ptr [si+5]
byte_T equ      byte ptr [si+6]
byte_Tlequ      byte ptr [si+7]
byte_Zlequ      byte ptr [si+8]
byte_Zhequ      byte ptr [si+9]
byte_Xlequ      byte ptr [si+10]
byte_Xhequ      byte ptr [si+11]
byte_Ylequ      byte ptr [si+12]
byte_Yhequ      byte ptr [si+13]

_TEXT   segment byte public 'CODE'
DGROUP  group   _DATA,_BSS
        assume  cs:_TEXT,ds:DGROUP
_TEXT   ends

_DATA   segment word public 'DATA'
_d@     label   byte
_DATA   ends

_BSS    segment word public 'BSS'
_b@     label   byte
_BSS    ends

_DATA   segment word public 'DATA'
_s@     label   byte
_DATA   ends


_DATA   segment word public 'DATA'
        extrn   _R_buf:byte
        extrn   _G_buf:byte
        extrn   _B_buf:byte
        extrn   _Z_buf:word
        extrn   _D1x:dword, _D2x:dword, _D3x:dword
        extrn   _X:word,  _Y:word,  _Z:dword
        extrn   _R:dword, _G:dword, _B:dword
        extrn   _REG:word
        extrn   _D1:dword, _D2:dword, _D3:dword, _T:dword
        extrn   _QQ:word
_DATA   ENDS

_TEXT   segment byte public 'CODE'
```

```
;***********************************************
;     As seguintes rotinas sao utilizadas pelo
;algoritmo DDA, enscrito em "C"
;***********************************************
;***********************************************
;                PIX_INTERNO
;     Retorna estado de pixel interno conforme
;as distancias D1, D2 e D3 sejam todas maiores
;que -1. Os dados de entrada sao os dados glo-
;bais do DDA, D1, D2 e D3. A condicao de pixel
;interno e' devolvida em AX.
; - invocada por "DDApix()"
;***********************************************
        public _PIX_INTERNO
_PIX_INTERNO proc near
        mov     ax,word ptr _D1+2
        or      ax,ax
        jz      pint1   ;0 < D1 < 1
        inc     ax
        or      ax,ax
        js      pext    ;D1 < -1
pint1:
        mov     ax,word ptr _D2+2
        or      ax,ax
        jz      pint2   ;0 < D2 < 1
        inc     ax
        or      ax,ax
        js      pext    ;D2 < -1
pint2:
        mov     ax,word ptr _D3+2
        or      ax,ax
        jz      pint3   ;0 < D3 < 1
        inc     ax
        or      ax,ax
        js      pext    ;D3 < -1
pint3:
        mov     ax,1    ;indicacao de pix interno
        ret
pext:
        mov     ax,0    ;indicacao de pix externo
        ret

_PIX_INTERNO endp
```

```
;************************************************
;                    DIR
;     Obtem a informacao de direcao a tomar, a
;partir do sinal da derivada  x  da  distancia
;negativa, conforme explicado na secao 4.2.4.
;     Os parametros de entrada sao  globais  e
;o resultado DIR e' devolvido em AX  (8000h  =
;direita e 0000h = esquerda.
; - invocada por "DDApix()"
;************************************************
        public  _DIR
_DIR    proc    near
        mov     ax, word ptr _D1+2
        inc     ax
        or      ax,ax
        jns     dd2    ;D1 > -1
        mov     ax, word ptr _D1x+2 ;sinal da derivada
        jmp     filtdr
dd2:
        mov     ax, word ptr _D2+2
        inc     ax
        or      ax,ax
        jns     dd3    ;D2 > -1
        mov     ax, word ptr _D2x+2 ;sinal da derivada
        jmp     filtdr
dd3: ;se nem D1 e nem D2 e' negativo usa D3x
        mov     ax, word ptr _D3x+2
filtdr:and      ax, 8000h    ;filtra bit de sinal
        ret
_DIR    endp


;************************************************
;                   ATU_DSP
;    Esta rotina serve de preparacao dos dados
;correntes do DDA para o calculo da transparen-
;cia final (rotina NF), e para a chamada da ro-
;tina simuladora do HAD - HadPix.
; - invocada por "DDApix"
; - parametros de entrara: dados globais do DDA
; - chama "HadPix"
;************************************************
        public  _ATU_DSP
_ATU_DSP proc   near
        push    si
        push    di
        mov     si,offset _REG
        mov     ax,word ptr _X
        mov     word ptr word_X,ax
        mov     ax,word ptr _Y
        mov     word ptr word_Y,ax
        mov     ax,word ptr _Z+2
        mov     word ptr word_Z,ax
        mov     ax,word ptr _R+2
```

```
        mov     word ptr word_R,ax
        mov     ax,word ptr _G+2
        mov     word ptr word_G,ax
        mov     ax,word ptr _B+2
        mov     word ptr word_B,ax
        call    _NF
        mov     word ptr word_Nf,ax
        call    _HadPix
        pop     di
        pop     si
        ret
_ATU_DSP endp


;*******************************************************
;                       HadPix
;   Simula a funcao do HAD - Hardware de atualizacao
;de display - descrito na secao 4.2.11.
;   A rotina recebe um pixel emitido pelo DDA,
;executa o teste de profundidade z e calcula o
;efeito da transparencia nf sobre R, G e B.
;   Se for o caso, o pixel  e' atualizado na imagem.
; - invocada por "ATU_DSP" e main()
; - Dados de entrada apontados por SI
;SI -> [R][G][B][Nf][Z][X][Y] - 2 bytes por dado
;SI +  0  2  4  6   8  10 12  - enderecos dos dados
;. - chama: "Des_Pix_8x8"
;*******************************************************
        public  _HadPix
_HadPix proc    near
        mov     si,offset _REG
        mov     ax,ds
        mov     es,ax

;------- if x>127 or x<0 then fim : pixel fora da tela
        mov     ax,word_X
        test    ax,0ff80h
        jnz     fim

;------- if y<0 or y>95 then fim : pixel fora da tela
        mov     ax,word_Y
        test    ah,80h
        jnz     fim
        cmp     ax,96
        jge     fim

;------- ender = Y(6-0):X(6-0) -> DI
;concatena X e Y para a obtencao do endereco de pixel
        mov     ah,al
        mov     al,byte_X1
        sal     al,1
        shr     ax,1
        mov     di,ax
```

```
;------- if z >= Z_buf(end) then fim : teste de profundidade
        mov     bx,di    ;ender
        sal     bx,1     ;ender*2 p/word
        mov     ax,_Z_buf[bx]
        cmp     ax,word_Z
        jbe     fim

;------- if R < 0 then R=0 : saturacao (fig. 4.20) em R
;-------            else if R>255 then R=255
        mov     ax,word_R
        or      ax,ax
        jns     jr1
        mov     word_R,0
        jmp     jg0
jr1:    or      ah,ah
        jz      jg0
        mov     byte_R1,255

;------- if G < 0 then G=0 : saturacao em G
;-------            else if G>255 then G=255
jg0:    mov     ax,word_G
        or      ax,ax
        jns     jg1
        mov     word_G,0
        jmp     jb0
jg1:    or      ah,ah
        jz      jb0
        mov     byte_G1,255

;------- if B < 0 then B=0 : saturacao em B
;-------            else if B>255 then B=255
jb0:    mov     ax,word_B
        or      ax,ax
        jns     jb1
        mov     word_B,0
        jmp     jt0
jb1:    or      ah,ah
        jz      jt0
        mov     byte_B1,255

;------- if Nf < 0 then fim : pixel invisivel
jt0:    mov     ax,word_Nf
        or      ax,ax
        jz      fim
        jns     jt1
fim:
        ret
jt1:

;calcula o efeito da transparencia sobre R, G e B
;------- if Nf >=1 then Nf=1
        test    ah,7fh
        jnz     Tovf
```

```
;------- T1 = 1-Nf
        neg     al
        mov     byte_T1,al ;1-Nf

;------- if QQ nao executa antialiasing
        mov     ax,_QQ
        or      ax,ax
        jnz     fim

;------- R = Nf*R + (1-Nf)*R_buf(end)
        mov     al,byte_R1
        mul     byte_T
        mov     cl,ah
        mov     al,_R_buf[di]
        mul     byte_T1
        add     ah,cl
        mov     byte_R1,ah

;------- G = Nf*G + (1-Nf)*G_buf(end)
        mov     al,byte_G1
        mul     byte_T
        mov     cl,ah
        mov     al,_G_buf[di]
        mul     byte_T1
        add     ah,cl
        mov     byte_G1,ah

;------- B = Nf*B + (1-Nf)*B_buf(end)
        mov     al,byte_B1
        mul     byte_T
        mov     cl,ah
        mov     al,_B_buf[di]
        mul     byte_T1
        add     ah,cl
        mov     byte_B1,ah
        jmp     NatuZ
Tovf:
;------- Z_buf(end) = Z
        sal     di,1
        mov     ax,word_Z
        mov     _Z_buf[di],ax
        shr     di,1
NatuZ:
;------- R_buf(end) = R
        mov     al,byte_R1
        mov     _R_buf[di],al

;------- G_buf(end) = G
        mov     al,byte_G1
        mov     _G_buf[di],al

;------- B_buf(end) = B
        mov     al,byte_B1
        mov     _B_buf[di],al
```

```
        jmp     DesPix8x8

_HadPix endp


;*********************************************
;                       NF
;calcula a transparencia final  Nf=N1*N2*N3*T
;     Esta rotina opera de maneira semelhante
;ao circuito da fig. 4.19, onde  so' se  usa
;operacao de multiplicacao se houverem  mais
;de dois dados diferentes de 1
; - invocada por "ATU_DSP"
; - parametros de entrada globais do DDA
;*********************************************
        public  _NF
_NF     proc    near
        mov     ax,word ptr _D1+2
        add     ax,1
        js      NfEq0   ;funcao NORM(D1)
        jnz     N1Eq1
        mov     ch,0
        mov     cl,byte ptr _D1+1 ;nf=D1
        jmp     N1Ls1
N1Eq1:  mov     cx,100h             ;nf=1
N1Ls1:
        mov     ax,word ptr _D2+2
        add     ax,1
        js      NfEq0   ;funcao NORM(D2)
        jnz     N2Eq1
        or      ch,ch
        jz      Nf1Ls1
        mov     ch,0
        mov     cl,byte ptr _D2+1   ;nf=D2
        jmp     N2Eq1
Nf1Ls1:mov      al,byte ptr _D2+1
        mul     cl                  ;nf=nf*D2
        mov     cl,ah
N2Eq1:
        mov     ax,word ptr _D3+2
        add     ax,1
        js      NfEq0   ;funcao NORM(D3)
        jnz     N3Eq1
        or      ch,ch
        jz      Nf2Ls1
        mov     ch,0
        mov     cl,byte ptr _D3+1   ;nf=D3
        jmp     N3Eq1
Nf2Ls1:mov      al,byte ptr _D3+1
        mul     cl                  ;nf=nf*d3
        mov     cl,ah
N3Eq1:
        mov     ax,word ptr _T+2
```

```
        or      ax,ax
        js      NfEq0   ;funcao saturacao(T)
        jnz     NfOK
        or      ch,ch
        jz      Nf3Ls1
        MOV     Ch,0
        mov     cl,byte ptr _T+1 ;nf=T
        jmp     NfOK
Nf3Ls1:mov      al,byte ptr _T+1
        mul     cl                ;nf=nf*T
        mov     cl,ah
NfOK:   mov     ax,cx
        ret
NfEq0:
        xor     ax,ax
        ret

_NF     endp

;************************************************
;                  DesPix8x8
;desenha um pixel 8x8 na tela ARTIST de acordo
;com a tabela "TabMask", definida  no  fim  da
;listagem, para a obtencao de 64 tons interme-
;diarios de cor para cada componente R, G, e B,
;segundo a tecnica de "halftoning".   O  pixel
;fica com dimensoes 8 vezes maior reduzindo  a
;resolucao da imagem para 128x96  pixels,  com
;a disponibilidade de 256k cores.
; - invocado por "FillMem", "HadPix" e main()
; - parametros de entrada apontados por SI
; - chama "tab", "comando" e "parametro"
;************************************************
;SI -> [R][G][B]...
        public  _DesPix8x8
_DesPix8x8 proc near
DesPix8x8:
        mov     si,offset _REG
        mov     cl,0    ;plano=0
lpdp1:  mov     bl,cl
        mov     bh,0
        sal     bx,1
        mov     bl,[si+bx]
        and     bl,0fch ;trunca em 6 bits (64 cores)
        rol     bx,1
        add     bx,offset TabMask ;obtem padrao para
halftoning

        mov     ah,78h  ;preenche controlador com  o padrao
        call    comando
        mov     ch,8
lpdp2:  mov     ah, cs:[bx]
        call    parametro
```

```
        inc     bx
        sub     ch,1
        jnz     lpdp2

        push    cx      ; tab(x1, y1, plano)
        mov     bl,byte_X1
        mov     bh,byte_Y1
        call    tab
        pop     cx

        mov     ah,4ch
        call    comando
        mov     ah,12h
        call    parametro
        mov     ah,8
        call    parametro
        mov     ah,0
        call    parametro
        mov     ah,8
        call    parametro
        mov     ah,0
        call    parametro

        mov     ah,68h
        call    comando
        inc     cl
        cmp     cl,3
        jb      lpdp1
        ret
_DesPix8x8 endp

;************************************************
;                  FILLMEM
;   Inicializa area de imagem  com  a  cor  de
;fundo, a area de z-buffer com a  profundidade
;maxima, e atualiza display com cor de  fundo.
; - invocado pelo programa principal main()
; - chama "Des_Pix_8x8","comando","parametro"
; - parametros de entrada apontados por SI
;************************************************
        public  _FillMem
_FillMem proc   near
        push    si
        push    di
        mov     si,offset _REG
        cld
        mov     ax,ds
        mov     es,ax
; - - - - - - - - - - - - - -
        mov     cx,12288
        mov     al,byte_R1
        mov     bl,al
        mov     di,offset _R_buf
        rep stosb
```

```
; - - - - - - - - - - - - - -
        mov     cx,12288
        mov     al,byte_G1
        or      bl,al
        mov     di,offset _G_buf
        rep stosb
; - - - - - - - - - - - - - -
        mov     cx,12288
        mov     al,byte_B1
        or      bl,al
        mov     di,offset _B_buf
        rep stosb
; - - - - - - - - - - - - - -
        mov     cx,12288
        mov     ax,0ffffh
        mov     di,offset _Z_buf
        rep stosw
; - - - - - - - - - - - - - -
        or      bl,bl
        jz      fill0   ;se R=G=B=0 entao
                        ;apaga com rotina rapida

        mov     byte_Y1, 0
fillp1:mov      byte_X1, 0
fillp2:call     _DesPix8x8
        inc     byte_X1
        cmp     byte_X1,128
        jb      fillp2
        inc     byte_Y1
        cmp     byte_Y1,96
        jb      fillp1
        pop     di
        pop     si
        ret
; - - - - - - - - - - - - - -
fill0:
        mov     cl,3    ;for plano = 0 to 3
flp1:
        xor     bx,bx   ;tab(0,0,plano)
        push    cx
        call    tab
        pop     cx

        mov     ah,04ah ;seta mascara = 0xFFFF
        call    comando
        mov     ah,0ffh
        call    parametro
        call    parametro

        mov     ch,0    ;for linha = 0 to 3
flp2:
        mov     ah,04ch
        call    comando
        mov     ah,002h
```

```
        call    parametro
        mov     ah,0ffh
        call    parametro
        mov     ah,03fh
        call    parametro

        mov     ah,022h ;limpa linha
        call    comando
        mov     ah,001h
        call    parametro
        call    parametro

        inc     ch      ;next linha
        cmp     ch,4
        jb      flp2

        inc     cl      ;next plano
        cmp     cl,4
        jb      flp1

        pop     di
        pop     si
        ret

_FillMem endp

;*************************************************
;                    INIT
;inicializa controlador NEC7220, da placa ARTIST,
;no modo 1024x768 e programa as cores na memoria
;de pallette.
; - invocado pelo programa principal "main()"
; - chama "comando", "parametro" e "progcor"
;*************************************************
        public  _init
_init   proc    near
        mov     dx,02e6h
        mov     al,0ffh
        out     dx,al   ;ZOOM = 1
        mov     dx,02e1h
        xor     al,al
        out     dx,al   ;reset controlador 7220

        mov     ah,0    ;COMANDO DE RESET
        call    comando
        mov     ah,01fh ;1 - sem flicking, intrelaced
        call    parametro
        mov     ah,03eh ;2 - 64 word/linha
        call    parametro
        mov     ah,0e5h ;3 - HSW=6 VSW=7
        call    parametro
        mov     ah,00ch ;4 - HFP=4
        call    parametro
        mov     ah,007h ;5 - HBP=8
```

```
        call    parametro
        mov     ah,001h ;6 - VBP=1
        call    parametro
        mov     ah,080h ;7 - 384 linhas/campo
        call    parametro
        mov     ah,069h ;8 - VBP=26
        call    parametro

;VSYNC
        mov     ah,06fh ;MASTER MODE SINCRONIZATION
        call    comando

        mov     ah,047h ;PITCH COMMAND
        call    comando
        mov     ah,040h ;x e' 64 words
        call    parametro

        mov     ah,070h ;PRAM COMMAND
        call    comando
        mov     ah,000h ;start addrs. = 0
        call    parametro
        call    parametro
        call    parametro
        mov     ah,030h ;Upper Portion = 768
        call    parametro

        mov     ah,04bh ;CCHAR COMMAND
        call    comando
        mov     ah,000h
        call    parametro
        mov     ah,0c0h
        call    parametro
        mov     ah,000h
        call    parametro

        mov     ah,046h ;ZOOM COMMAND
        call    comando
        mov     ah,000h ;ZFD=ZFW=1
        call    parametro

;comando de START
        mov     ah,06bh
        call    comando

;programa cores na tabela LUT
        mov     bx,offset tabcor
        mov     cl,0
prgc1:  mov     dx,02e8h
        mov     ah,cl
        call    progcor
        inc     dx
        inc     dx
        mov     ah,cs:[bx]
        call    progcor
```

```
        inc     dx
        inc     dx
        inc     bx
        mov     ah,cs:[bx]
        call    progcor
        inc     cl
        inc     bx
        cmp     cl,16
        jb      prgc1

        mov     ah,20h  ;replace mode
        call    comando

        mov     ah,70h ;rola tela
        call    comando
        mov     ah,0
        call    parametro
        call    parametro


        ret
_init   endp

;************************************************
;                   PROGCOR
;          programa uma cor na LUT
;           endereco de I/O em DX
;                  cor em AH
; - invocado por "init"
;************************************************
progcor proc near
        push    dx
        mov     dx,02e0h
corlp1:in       al,dx
        test    al,40h
        jnz     corlp1  ;espera fim retraco Hor.
corlp2:in       al,dx
        test    al,40h
        jz      corlp2  ;espera retraco Hor.
        pop     dx
        mov     al,ah
        out     dx,al
        ret
progcor endp
```

```
;***********************************************
;                     TAB
;              posiciona cursor
;     X = BL        Y = BH        plano = Cl
; - invocado por "Des_Pix_8x8"
; - chama "comando" e "parametro"
;***********************************************
tab     proc    near
        inc     bh
        and     cl,3
        rol     cl,1
        rcr     bl,1
        rcr     cl,1
        and     bl,63
        rol     bh,1
        mov     ah,49h
        call    comando
        mov     ah,bl
        call    parametro
        mov     ah,bh
        call    parametro
        mov     ah,cl
        call    parametro
        ret
tab     endp

;***********************************************
;                   COMANDO
;emite comando ao controlador
;comando em AH
;***********************************************
comando proc    near
        mov     dx,02e0h
espc:   in      al,dx    ;le status
        test    al,2
        jnz     espc     ;se FIFO full espera
        mov     al,ah
        inc     dx
        out     dx,al    ;larga comando
        dec     dx
        ret
comando endp
```

```
;***********************************************
;                 PARAMETRO
;emite parametro ao controlador
;parametro em AH
;***********************************************
parametro proc near
espp:   in      al,dx    ;le status
        test    al,2
        jnz     espp     ;se FIFO full espera
        mov     al,ah
        out     dx,al    ;larga parametro
        ret
parametro endp


;*****************************************************
tabcor:
;     A memoria de cores da placa  artist  permite
;a escolha de uma das 4096 cores  disponiveis  no
;pallette para cada uma das 16 cores exibiveis.
;     A configuracao das 16 cores, escolhida aqui,
;e' tal que atribui a cada  um  dos 4  planos  de
;memoria uma das componentes de cor R, G  e  B  o
;valor maximo (r, g ou b  =  0000b)  ou  o  valor
;minimo  (r,  g  ou  b  =  1111b),  conforme  o
;respectivo bit do plano seja 1 ou  0  (A  tabela
;recebe os bits invertidos).
;
;             conteudo da mem.       endereco
;                 Xr   gb          plano   3210
;                 ||   ||                  ||||
        db      0ffh,0ffh        ;cor 0  (0000)
        db      0f0h,0ffh        ;cor 1  (0001)
        db      0ffh,00fh        ;cor 2  (0010)
        db      0f0h,00fh        ;cor 3  (0011)
        db      0ffh,0f0h        ;cor 4  (0100)
        db      0f0h,0f0h        ;cor 5  (0101)
        db      0ffh,000h        ;cor 6  (0110)
        db      0f0h,000h        ;cor 7  (0111)
        db      0ffh,0ffh        ;cor 8  (1000)
        db      0f0h,0ffh        ;cor 9  (1001)
        db      0ffh,00fh        ;cor 10 (1010)
        db      0f0h,00fh        ;cor 11 (1011)
        db      0ffh,0f0h        ;cor 12 (1100)
        db      0f0h,0f0h        ;cor 13 (1101)
        db      0ffh,000h        ;cor 14 (1110)
        db      0f0h,000h        ;cor 15 (1111)
```

```
;*************************************************
TabMask:
;    A seguinte tabela contem as 65 configuracoes
;de bits para o "halftoning". A intensidade de
;cor e' conseguida pela intensificacao de 0 a 64
;pixels do pixelzao 8x8. A configuracao foi esco-
;lhida de maneira a que a justaposicao destes, na
;tela, nao produza padroes repetitivos visiveis.

; 0
        db      00000000b
        db      00000000b
        db      00000000b
        db      00000000b
        db      00000000b
        db      00000000b
        db      00000000b
        db      00000000b
; 1
        db      00000000b
        db      00000000b
        db      00000000b
        db      00010000b
        db      00000000b
        db      00000000b
        db      00000000b
        db      00000000b
; 2
        db      00000000b
        db      00000100b
        db      00000000b
        db      00000000b
        db      00000000b
        db      01000000b
        db      00000000b
        db      00000000b
; 3
        db      00000000b
        db      00010000b
        db      00000000b
        db      00000000b
        db      00000100b
        db      00000000b
        db      01000000b
        db      00000000b
; 4
        db      00000000b
        db      01000000b
        db      00000000b
        db      00000100b
        db      00000000b
        db      01000000b
        db      00000000b
        db      00000100b
```

```
; 5
        db      00000000b
        db      00000010b
        db      00000000b
        db      00010000b
        db      00000001b
        db      00000000b
        db      00000100b
        db      01000000b
; 6
        db      00000000b
        db      00000100b
        db      01000000b
        db      00000010b
        db      00010000b
        db      00000000b
        db      00000010b
        db      01000000b
; 7
        db      00000000b
        db      00000100b
        db      01000000b
        db      00000010b
        db      00010000b
        db      10000000b
        db      00000100b
        db      01000000b
; 8
        db      10001000b
        db      00000000b
        db      00100010b
        db      00000000b
        db      10001000b
        db      00000000b
        db      00100010b
        db      00000000b
; 9
        db      10001000b
        db      00000000b
        db      00100010b
        db      00000000b
        db      00001001b
        db      00100000b
        db      10000100b
        db      00000000b
; 10
        db      10000100b
        db      00010000b
        db      00000010b
        db      00100000b
        db      00001001b
        db      01000000b
        db      00000010b
        db      00010000b
```

```
; 11
        db      10000100b
        db      00010000b
        db      10000100b
        db      00100000b
        db      00001001b
        db      01000000b
        db      00000010b
        db      00010000b
; 12
        db      10000100b
        db      00010000b
        db      01000010b
        db      00001000b
        db      00100001b
        db      00001000b
        db      01000010b
        db      00010000b
; 13
        db      10000100b
        db      00010000b
        db      01000010b
        db      00001000b
        db      00100001b
        db      00000100b
        db      10010000b
        db      00100010b
; 14
        db      00010001b
        db      01000000b
        db      00001001b
        db      00100010b
        db      10001000b
        db      00100100b
        db      10010000b
        db      00000100b
; 15
        db      10001000b
        db      00100010b
        db      10001000b
        db      00100010b
        db      10001000b
        db      00100100b
        db      10000000b
        db      00010001b
; 16
        db      10001000b
        db      00100010b
        db      10001000b
        db      00100010b
        db      10001000b
        db      00100010b
        db      10001000b
        db      00100010b
```

```
; 17
        db      10001000b
        db      00100010b
        db      10001000b
        db      00100010b
        db      10010100b
        db      00100010b
        db      10001000b
        db      00100010b
; 18
        db      10001000b
        db      00100010b
        db      01001001b
        db      00100010b
        db      10001000b
        db      00100010b
        db      10010100b
        db      00100010b
; 19
        db      10001000b
        db      01010010b
        db      10001000b
        db      00100010b
        db      10010100b
        db      00100010b
        db      10010100b
        db      00100010b
; 20
        db      01001001b
        db      00100010b
        db      10010100b
        db      00100010b
        db      01001001b
        db      00100010b
        db      10010100b
        db      00100010b
; 21
        db      01001001b
        db      00100010b
        db      10010100b
        db      00101010b
        db      01000001b
        db      00101010b
        db      10010100b
        db      00100010b
; 22
        db      01001001b
        db      00100010b
        db      10010100b
        db      01001001b
        db      10010100b
        db      00100010b
        db      01001001b
        db      10010100b
```

```
; 23
        db      01000001b
        db      00101010b
        db      10010100b
        db      01001001b
        db      10010100b
        db      00101010b
        db      01000001b
        db      10101010b
; 24
        db      01010101b
        db      00100010b
        db      01010101b
        db      10001000b
        db      01010101b
        db      00100010b
        db      01010101b
        db      10001000b
; 25
        db      01010101b
        db      00100010b
        db      01010101b
        db      10101000b
        db      01010101b
        db      00100010b
        db      01010101b
        db      10001000b
; 26
        db      01010101b
        db      00100010b
        db      01010101b
        db      10101000b
        db      01010101b
        db      00100010b
        db      01010101b
        db      10001010b
; 27
        db      01010101b
        db      00100010b
        db      01010101b
        db      10101010b
        db      01010101b
        db      00100010b
        db      01010101b
        db      10001010b
; 28
        db      01010101b
        db      00100010b
        db      01010101b
        db      10101010b
        db      01010101b
        db      00100010b
        db      01010101b
        db      10101010b
```

```
; 29
        db      01010101b
        db      00100010b
        db      01010101b
        db      10101010b
        db      01010101b
        db      10100010b
        db      01010101b
        db      10101010b
; 30
        db      01010101b
        db      00101010b
        db      01010101b
        db      10101010b
        db      01010101b
        db      10100010b
        db      01010101b
        db      10101010b
; 31
        db      01010101b
        db      10101010b
        db      01010101b
        db      10101010b
        db      01010101b
        db      10100010b
        db      01010101b
        db      10101010b
; 32
        db      01010101b
        db      10101010b
        db      01010101b
        db      10101010b
        db      01010101b
        db      10101010b
        db      01010101b
        db      10101010b
; 33
        db      01010101b
        db      10101010b
        db      01010101b
        db      10101010b
        db      01011101b
        db      10101010b
        db      01010101b
        db      10101010b
; 34
        db      11010101b
        db      10101010b
        db      10101010b
        db      10101010b
        db      01011101b
        db      10101010b
        db      01010101b
        db      10101010b
```

```
; 35
        db      11011101b
        db      10101010b
        db      01010101b
        db      10101010b
        db      01011101b
        db      10101010b
        db      01010101b
        db      10101010b
; 36
        db      11011101b
        db      10101010b
        db      01010101b
        db      10101010b
        db      11011101b
        db      10101010b
        db      01010101b
        db      10101010b
; 37
        db      11011101b
        db      10101010b
        db      01010101b
        db      10101010b
        db      11011101b
        db      10101010b
        db      01110101b
        db      10101010b
; 38
        db      11011101b
        db      10101010b
        db      01010111b
        db      10101010b
        db      11011101b
        db      10101010b
        db      01110101b
        db      10101010b
; 39
        db      11011101b
        db      10101010b
        db      01010111b
        db      10101010b
        db      11011101b
        db      10101010b
        db      01110111b
        db      10101010b
; 40
        db      11011101b
        db      10101010b
        db      01110111b
        db      10101010b
        db      11011101b
        db      10101010b
        db      01110111b
        db      10101010b
```

```
; 41
        db      11010101b
        db      01101011b
        db      10110110b
        db      01101011b
        db      11010101b
        db      10111110b
        db      01010101b
        db      10111110b
; 42
        db      11011101b
        db      01101011b
        db      10110110b
        db      01101011b
        db      11011101b
        db      10110110b
        db      01101011b
        db      10110110b
; 43
        db      11011101b
        db      01101011b
        db      11010101b
        db      10111110b
        db      11010101b
        db      10010100b
        db      11011101b
        db      10110110b
; 44
        db      11011101b
        db      01101011b
        db      11011101b
        db      10110110b
        db      11011101b
        db      01101011b
        db      11011101b
        db      10110110b
; 45
        db      10101101b
        db      01110111b
        db      11011101b
        db      01101011b
        db      11011101b
        db      01101011b
        db      11011101b
        db      01110111b
; 46
        db      11011101b
        db      10110110b
        db      11011101b
        db      01110111b
        db      11011101b
        db      01101011b
        db      11011101b
        db      01110111b
```

```
; 47
        db      11011101b
        db      01110111b
        db      11011101b
        db      01101011b
        db      11011101b
        db      01110111b
        db      11011101b
        db      01110111b
; 48
        db      11011101b
        db      01110111b
        db      11011101b
        db      01110111b
        db      11011101b
        db      01110111b
        db      11011101b
        db      01110111b
; 49
        db      11011101b
        db      01110111b
        db      11011101b
        db      01110111b
        db      11011011b
        db      01111111b
        db      11101110b
        db      01110111b
; 50
        db      10111111b
        db      11110110b
        db      11011101b
        db      01110111b
        db      11011011b
        db      01101111b
        db      11111011b
        db      11101110b
; 51
        db      11101111b
        db      10111101b
        db      11110111b
        db      11011110b
        db      11111011b
        db      01101111b
        db      11011101b
        db      01111011b
; 52
        db      11101111b
        db      10111101b
        db      11110111b
        db      11011110b
        db      11110111b
        db      10111101b
        db      11101111b
        db      01111011b
```

```
; 53
        db      11101111b
        db      01111011b
        db      11011111b
        db      11110110b
        db      10111111b
        db      11111101b
        db      11101111b
        db      01111011b
; 54
        db      11101111b
        db      11111101b
        db      11011111b
        db      11110110b
        db      10111111b
        db      11111101b
        db      11101111b
        db      01111011b
; 55
        db      11111111b
        db      11011101b
        db      11111111b
        db      11110110b
        db      11011111b
        db      01111011b
        db      11111111b
        db      01110111b
; 56
        db      11111111b
        db      11011101b
        db      11111111b
        db      01110111b
        db      11111111b
        db      11011101b
        db      11111111b
        db      01110111b
; 57
        db      11111011b
        db      10111111b
        db      11111101b
        db      11101111b
        db      01111111b
        db      11111011b
        db      10111111b
        db      11111111b
; 58
        db      11111011b
        db      10111111b
        db      11111101b
        db      11101111b
        db      11111111b
        db      11111110b
        db      10111111b
        db      11111111b
```

```
; 59
        db      11111101b
        db      11111111b
        db      11101111b
        db      11111110b
        db      11111111b
        db      11111011b
        db      10111111b
        db      11111111b
; 60
        db      10111111b
        db      11111111b
        db      11111011b
        db      11111111b
        db      10111111b
        db      11111111b
        db      11111011b
        db      11111111b
; 61
        db      11101111b
        db      11111111b
        db      11111111b
        db      11111011b
        db      11111111b
        db      10111111b
        db      11111111b
        db      11111111b
; 62
        db      11111011b
        db      11111111b
        db      11111111b
        db      11111111b
        db      10111111b
        db      11111111b
        db      11111111b
        db      11111111b
; 63
        db      11111111b
        db      11111111b
        db      11101111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
; 64
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
```

```
_TEXT   ends
        end
```