Técnicas Paralelas de *Rendering*

ANSELMO CARDOSO DE PAIVA
Pontifícia Universidade Católica - PUC-Rio - TecGraf
Universidade Federal do Maranhão – UFMA
email: paiva@tecgraf.puc-rio.br

NOEMI RODRIGUEZ
Pontifícia Universidade Católica - PUC-Rio
Departamento de Informática
email: noemi@tecgraf.puc-rio.br

MARCELO GATTASS[1]
Pontifícia Universidade Católica - PUC-Rio – TecGraf
email:gattass@tecgraf.puc-rio.br
PUC-RioInf.MCC34/98 September, 1998

ABSTRACT: This work surveys the parallel techniques applied to the *rendering* process. The basic parallel concepts are revised from the point of view of the rendering process, and some software and hardware implementations are discussed.
Keywords: computer graphics, rendering, parallel algorithms

RESUMO: Este trabalho realiza uma revisão das técnicas paralelas aplicadas ao processo de *rendering*. É apresentada uma visão geral da área de rendering em paralelo, na qual buscamos revisar os conceitos de algoritmos paralelos a partir do ponto de vista do processo de rendering. Ao final são apresentadas algumas implementações de sistemas paralelos para rendering.

Palavras-chave: computação gráfica, rendering, algoritmos paralelos

Sumário

Lista de Figuras

1. Introdução

Muitas das aplicações de computação envolvem a exibição de objetos e cenas tridimensionais. Por exemplo: sistemas de CAD, permitem aos usuários manipularem modelos dos componentes de máquinas, partes de carro, etc.; sistemas de simulação, apresentam uma figura se movendo continuamente, representando o mundo como é visto pelo piloto; sistemas médicos, possibilitam a visualização de aspectos anatômicos e fisiológicos do paciente; sistemas de visualização científica, criam as condições para estudo e compreensão de fenômenos da natureza.

Segundo Watt (Watt,1994), o processo geral que parte de uma base de dados que representa um objeto tridimensional e chega a uma projeção iluminada em uma superfície de visualização 2D é denominado *rendering*.

A produção de uma imagem de uma cena tridimensional em uma superfície de visualização bidimensional resulta em uma série de problemas tais como:

- Como é a profundidade, a terceira dimensão a ser representada na tela?
- Quais partes de que objetos estão escondidas por trás de outros objetos e precisam ser removidas?
- Como a luz, cor, sombras e texturas contribuem para a visualização da cena?

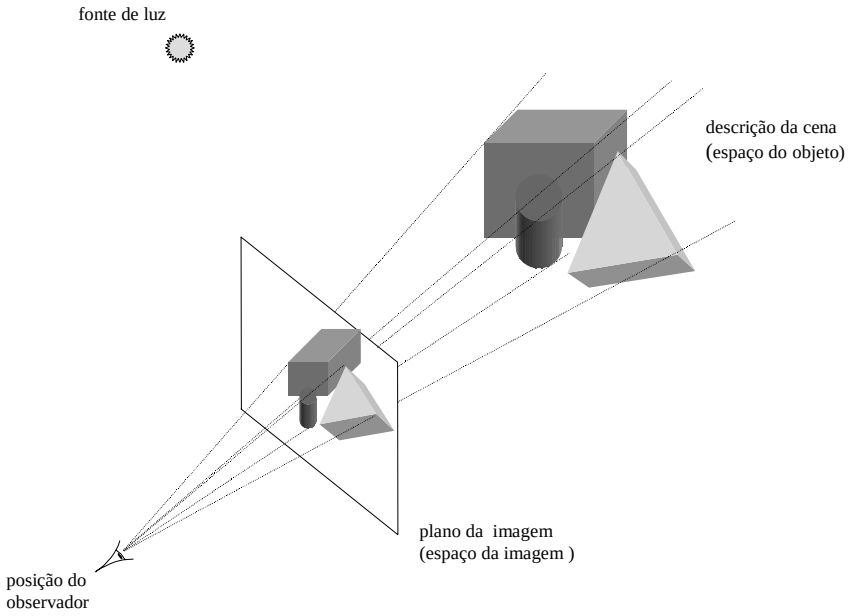


Figura 1.1 Visão Geral do Processo de Rendering

As técnicas de *rendering* buscam resolver estas questões colocadas acima. Em geral é utilizado um modelo de câmera, que tenta reconstruir a imagem da cena da mesma forma como ela seria gerada através da utilização de uma câmera fotográfica.

A Figura 1.1 mostra o esquema característico do problema geral de rendering.

De maneira simples podemos afirmar que, *rendering* é o processo de produção em computadores de imagens de cenas tridimensionais, a partir de uma descrição da cena. Uma dificuldade fundamental nesta produção, é a complexidade do mundo real. Existem muitas texturas de superfícies, gradações súbitas de cor, sombras, e reflexões. O custo computacional para simular todos estes efeitos pode ser muito alto.

Existem muitas formas de representação para os objetos que compõem a cena, e de acordo com a forma de representação temos um processo adequado para realizar o *rendering d*a cena. Assim, podemos afirmar que existem vários conjuntos de técnicas que permitem a geração de imagens 2D a partir de representações de objetos (cenas) tridimensionais. As duas classes mais importantes contêm as técnicas de *rendering* de polígonos e de volumes. Podemos definir como *rendering* de polígonos as técnicas onde a representação dos objetos da cena é toda baseada em objetos geométricos descritos por polígonos. Esta é a categoria de técnicas mais popular na computação gráfica, tendo alto grau de desenvolvimento e várias implementações. Outro conjunto de técnicas bastante popular, embora mais recente, é o *rendering* de volumes, no qual a descrição do objeto é feita através da amostragem de uma função escalar em um *grid* tridimensional, ou seja os objetos da cena são descritos como um função escalar amostrada neste *grid*.

Assim, para cenas complexas e/ou imagens de alta qualidade, o processo de *rendering* é computacionalmente caro. Em aplicações como visualização científica ou realidade virtual, pode ser necessária a performance de centenas de Megaflops e banda de memória de Gigabytes por segundo, ultrapassando a capacidade de um único processador.

Dado que a grande força da Computação Gráfica provém do fato de explorar a capacidade extremamente paralela do sistema humano de visão, é natural que o processamento paralelo se torne uma ferramenta básica para a construção de sistemas de *rendering* de alta performance.

Aplicações como visualização cientifica e simulação em tempo real motivaram inicialmente o desenvolvimento de métodos paralelos de *rendering*, mas atualmente

estes métodos são aplicados a qualquer técnica de geração de imagens usada em computação gráfica.

Uma outra demanda para sistemas paralelos de *rendering* é a necessidade de visualização de dados provenientes de aplicações rodando em computadores de alto desempenho, com grandes quantidades de dados. Surge então a necessidade de migração dos sistemas de visualização para estas máquinas, de modo a permitir lidar com a grande quantidade de dados gerados, além de explorar o paralelismo disponível neste ambiente.

Neste trabalho apresenta-se conceitos básicos sobre técnicas de *rendering* de polígonos e de volumes. Um histórico da utilização de paralelismo no processo de *rendering* será apresentado, assim como diferentes abordagens paralelas utilizadas para este problema. É estudado o relacionamento entre os diferentes tipos de paralelismo e o problema de *rendering.* Em seguida são apresentados os conceitos centrais para desenvolvimento de algoritmos paralelos de *rendering*, com considerações sobre projeto e implementação destes algoritmos. Ênfase é dada a considerações sobre a arquitetura e requisitos das aplicações. Apresenta-se então uma classificação dos algoritmos paralelos de *rendering*, e alguns exemplos de sistemas de *hardware* e *software* para *rendering* em paralelo.

2. O Processo de *Rendering*

Tradicionalmente o processo de *rendering* trata a questão da visualização de um conjunto de objetos que representam uma determinada cena. Com o crescimento da capacidade de processamento foi acrescentada uma nova dimensão a este problema. Esta dimensão surgiu a partir de sistemas de visualização onde há uma definição geral de um mundo, e o usuário (observador) pode navegar neste mundo, (sistemas baseados em imersão do observador). Nesta classe de sistemas podemos citar sistemas de realidade virtual, visualização de grandes extensões de terrenos, etc. Surge então o problema denominado como filtragem da cena, ou seja, a partir da descrição completa do mundo deve ser extraída a descrição da cena a ser visualizada na resolução adequada. Leclerc e Lau Jr. (Leclerc,1994) implementando uma aplicação de visualização de terrenos baseada em dados de altura e imagens de satélite levantaram o problema, que pode ser definido pela necessidade de obter, a partir de um grande conjunto de dados que representa o mundo, o subconjunto de dados que gerará a imagem de uma porção específica do ambiente.

Baseados nesta abordagem, podemos definir genericamente o processo de *rendering* como subdivido em duas etapas básicas. A primeira etapa (pré-processamento) envolve a filtragem da base de dados que representa o ambiente, de modo a obter o subconjunto destes dados que compõem a cena a ser gerada. Estes dados são então carregados para a memória e diponibilizados para a segunda etapa. A segunda etapa corresponde à visão tradicional do problema de *rendering*, que trata a visualização de uma cena a partir dos objetos que a representam. A seguir apresentamos uma idéia geral do processo de rendering para cenas representadas por polígonos e por volumes.

2.1 Rendering de Polígonos

Um dos métodos para realizar o *rendering* de uma cena é partir da descrição da mesma por objetos representados por polígonos. Neste caso, o processo de *rendering* realiza os seguintes passos básicos após a filtragem da base de dados:

i. polígonos são transformados de seu sistema local de representação, para o sistema de coordenadas da cena;
ii. objetos localizados no lado oposto ao observador são removidos(*culling*);
iii. polígonos são recortados em relação ao volume de visualização (*clipping*);

iv. objetos possuem seu cálculo de iluminação realizado;
v. polígonos recortados são projetados no plano da tela;
vi. o problema de visibilidade (que objetos sobrepõem-se a outros) é resolvido.

Como algoritmos gerais para o *rendering* de polígonos podemos citar: z-*buffer*, *ray tracing*. A parte central destes algoritmos é o método de determinação dos polígonos visíveis para uma determinada cena especificada. Também existem modelos de iluminação associados, que tratam a modelagem da interação da luz com os objetos da cena, o que determina a aparência dos mesmos na imagem gerada. Nesta área podemos destacar o algoritmo de radiosidade que busca resolver o problema de iluminação da cena de uma forma mais precisa, necessitando depois da utilização de um dos algoritmos anteriores para solução do problema de visibilidade.

O algoritmo de *z-buffer* desenvolvido inicialmente por Catmull (Catmull,1974) é extremamente simples de implementar tanto em *software* como em *hardware*. Este algoritmo se baseia na existência de um *buffer* (memória adicional) com entradas para cada um dos *pixels* da tela, no qual serão armazenados os valores de profundidade da imagem para cada *pixel*. Inicialmente todas as entradas possuem valor zero, e a imagem está preenchida com a cor de fundo. Assim os polígonos são introduzidos no processo de *rendering* em uma ordem qualquer, e durante a rasterização é feita a determinação, com o auxílio deste *buffer*, das porções visíveis de cada polígono. O pseudo código a seguir descreve este procedimento.

```
Para cada polígono faça
   Para cada pixel na projeção do polígono faça
      Pz = valor z do polígono nas coordenadas do pixel
      Se  pz > Zbuffer do pixel
           Pixel = valor do polígono
          Zbuffer do pixel recebe Pz
     fim se
   fim para
fim para
```

Algoritmo 2.1 Z-buffer

O algoritmo de *ray tracing* ((Appel,1968) e (MAGI,1968)) tenta simular o processo de visualização humano, no qual a imagem da cena é gerada através de raios de luz, que saindo da fonte luminosa, atingem os objetos que compõem a cena e

alcançam o olho do observador. A Figura 2.1 apresenta o esquema de visualização proposto pelo *ray tracing* mostrando que o valor do *pixel* da imagem é formado por raios, que partindo diretamente da fonte de luz, atingiram o objeto visualizado (raios diretos), mais as contribuições de raios que atingiram este objeto via refração e reflexão entre os objetos da cena (raios indiretos). Na sua forma mais simples este algoritmo percorre a trajetória de raios refletidos e transmitidos especularmente através do ambiente onde a cena é definida.

```
Para cada pixel da tela faça
      Lance um raio em direção à cena
      Calcule a interseção do raio com cada uma das faces da cena
      Ordene as interseções em relação à profundidade
      Valor do pixel é o correspondente à interseção mais próxima
fim para
```

Algoritmo 2.2 *Ray Tracing*

O Algoritmo 2.2 mostra como pode ser feita de maneira simples a implementação desta técnica. Devido ao fato de serem lançados muitos raios e calculadas as interseções com todas as faces de todos os polígonos que compõem a cena, este algoritmo é computacionalmente intensivo, sendo bastante apropriado para implementações paralelas.

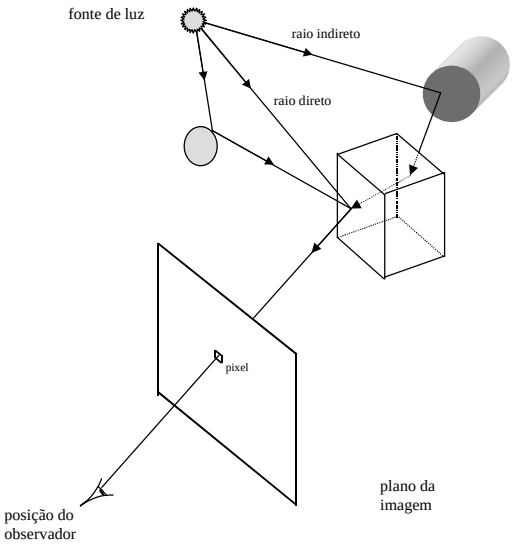


Figura 2.1 Algoritmo de Ray Tracing

Os dois algoritmos anteriores em geral utilizam um modelo para descrever de maneira aproximada a interação da luz com os objetos da cena. Este modelo é, em

geral, baseado em considerações locais, sendo denominados modelos locais de iluminação.

Na tentativa de melhorar a qualidade visual da imagem gerada através de uma melhor modelagem da interação da luz no ambienete, surge o método de radiosidade ((Goral,1984) e (Nishita,1985)). Este método é teoricamente rigoroso, e gera a solução para a interação difusa da luz em um ambiente fechado, baseado nos modelos de engenharia térmica para emissão e reflexão da radiação. Isto resulta em um tratamento bastante preciso da interreflexão entre os objetos da cena. A solução é independente da posição do observador, sendo baseada em uma subdivisão do ambiente em elementos discretos, nos quais assume-se que a intensidade de luz é constante.

Para um ambiente fechado o equilíbrio de energia pode ser estabelecido através de um conjunto de equações lineares que descrevem as intereflexões entre as superfícies. Assim, com a solução destas equações podemos computar o valor da radiosidade B_i de cada elemento discreto que constitui a cena. Após esta etapa é necessária apenas a determinação das superfícies visíveis e a rasterização dos objetos para gerar a cena.

2.2 Rendering de Volumes

Nesta categoria podemos citar as técnicas voltadas para a visualização de funções escalares amostradas no espaço tridimensional, que recebem a denominação de *rendering* de volume.

Estas técnicas geram uma imagem do volume diretamente a partir dos dados volumétricos, sem a necessidade de representações intermediárias. Estes algoritmos são especialmente apropriados para a visualização de volumes representando objetos amorfos tais como nuvens, gás, fluidos, etc.

A visualização em algoritmos de *rendering de volumes* é realizada, após a filtragem da base de dados, através de três etapas básicas, representadas no diagrama da Figura 2.2.

A primeira etapa do processo de *rendering* de volumes é a classificação dos dados, que envolve o mapeamento dos valores escalares a serem visualizados, em valores de cor associados e valores de opacidades. Isto é feito através de funções de mapeamento que são específicas para cada aplicação (funções de transferência).

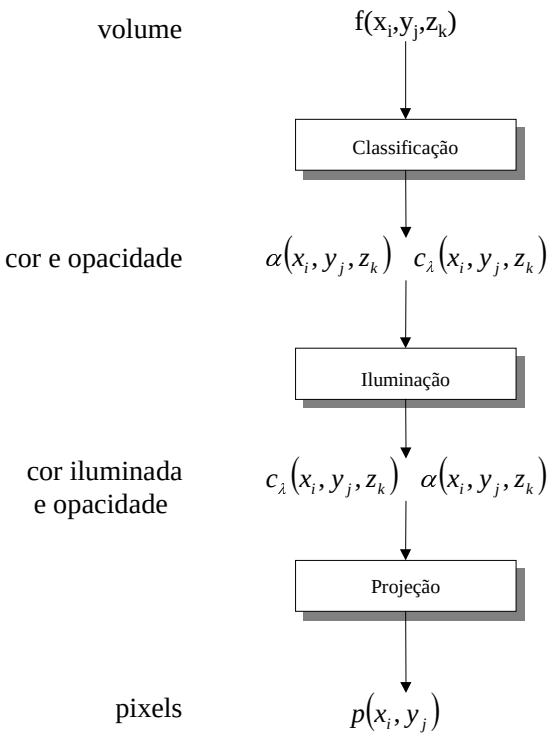


Figura 2.2 Pipeline de *Rendering* de Volumes

O cálculo da iluminação é a etapa do processo de rendering que tem o objetivo de descrever a aparência dos objetos. Este processo envolve a descrição da forma como se dá a interação dos diferentes tipos de luz, de diferentes fontes luminosas, em diferentes posições, com os materiais dos *voxels*[1] que compõem o volume.

A terceira etapa envolve o processo de visualização de um volume de valores de escalares representando a cor de cada *voxel*, já considerando a iluminação. Nesta etapa verifica-se como cada *voxel* contribui para o valor dos *pixels*.

Entre as principais técnicas de *rendering* de volumes podemos citar: *ray casting*, *splatting*, *shear-warping*, etc.

O algoritmo de *ray casting* (Tuy,1984) lança um raio a partir do plano da imagem em direção ao volume. Em um número pré-determinado de pontos ao longo do raio, obtém os valores de cor e opacidade. Estes valores são combinados (composição) de modo a gerar a cor final do pixel. Em geral esta composição é feita a

[1] *voxel* é o análogo 3D de um *pixel*. *Voxels* são paralelepípedos de mesmo tamanho, fortemente agrupados, formados pela divisão do espaço do objeto através de um conjunto de planos paralelos aos eixos principais deste espaço.

medida que o raio avança no volume, levando em consideração efeitos de transparências especificados pelos valores de opacidade.

A técnica *splatting* (Westover,1990) foi desenvolvida para melhorar o desempenho dos algoritmos de *rendering* de volumes. Esta técnica projeta cada um dos *voxels* na tela somando a contribuição em todos os *pixels* na área de influência da projeção do *voxel*. A melhora de desempenho se dá devido à utilização de uma tabela que descrev a projeção de todos os *voxels*, iguais a menos de uma transalação. Esta projeção é em geral efetuada na ordem inversa do *ray casting*, gerando assim um esquema inverso de composição dos valores acumulados no pixel.

O algoritmo de *shear-warping* (Lacroute,1996) realiza uma transformação no volume de modo que todos os *voxels* fiquem alinhados com os *pixels*. Assim o lançamento de raios sobre o volume passa a ser simplesmente um caminhamento na direção longitudinal (ortogonal ao plano da imagem) do volume. O esquema de composição deve ser feito de modo compatível com a ordem de caminhamento.

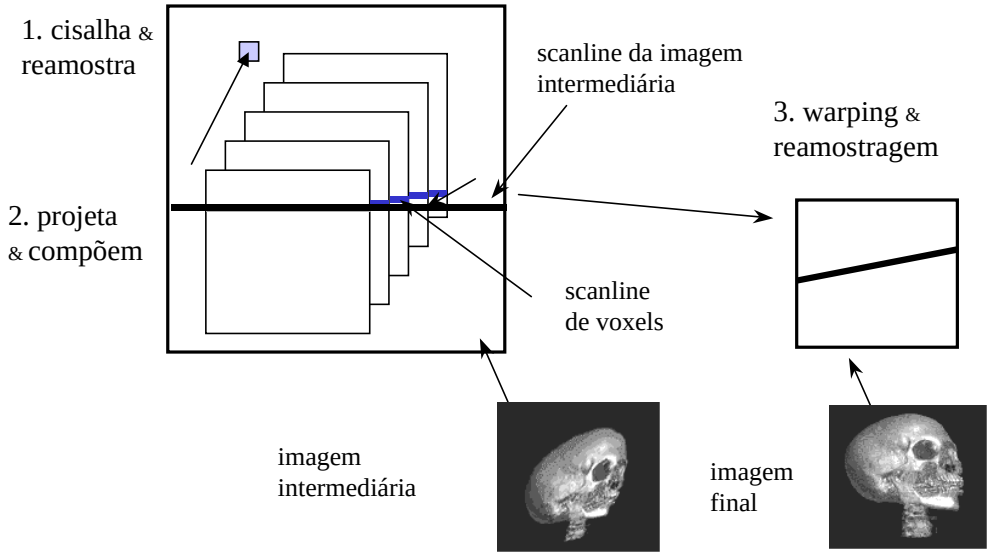


Figura 2.3 Algoritmo de *Shear-warping*

Este algoritmo é descrito em três etapas. Na primeira, a transformação de projeção do volume é decomposta em um cisalhamento 3D das fatias do volume, de modo que os *voxels* fiquem alinhados com o plano da imagem. Em seguida, estes *voxels* são projetados e acumulados em um *bufffer*, onde é criada a imagem intermediária, gerada com deformação em um plano paralelo ao da imagem. A última etapa é a aplicação de uma transformação 2D que corrige esta deformação (*warping*) para gerar a imagem final (Figura 2.3). As fatias do volume cisalhado são percorridas

na ordem *front-to-back* e os *voxels* projetados nos *pixels* correspondentes na imagem intermediária.

O algoritmo de ray-casting é o mais simples de implementar, sendo portanto o mais utilizado atualmente. No entanto, os algoritmos mais recentes como o *shear-warping* e *splatting* apresentam excelentes condições de desempenho, possuindo uma grande área de atuação. Atualmente, todo o esforço é dispendido no sentido de obter melhoria de desempenho nas implementações disponíveis, assim como melhorar a qualidade das imagens geradas.

3. Conceitos Básicos de Computação Paralela

Segundo Quin (Quin,1994) processamento paralelo é o processamento de informações que enfatiza o processamento concorrente de um ou mais conjuntos de dados pertencentes a um ou mais processos de modo a resolver um único problema. Podemos então conceituar computação paralela como a técnica que utiliza mais de uma unidade de processamento ao mesmo tempo para resolver um simples problema.

Considere um grupo de pessoas trabalhando em um único projeto. Algumas das questões que surgem são: Como o trabalho pode ser dividido entre estas pessoas, e em que ordem? Como as pessoas irão se comunicar e coordenar o trabalho? Como tratar pessoas executando tarefas em diferentes velocidades, e terminado-as em tempos diferentes? Como reunir os resultados dos trabalhos das diferentes pessoas para gerar o produto final?

Estas questões básicas surgidas da divisão do trabalho entre várias pessoas, também surgem quando é necessária a divisão do processamento entre vários processadores. As técnicas de processamento paralelo visam responder estas perguntas.

Existem duas características básicas de arquitetura que devem ser levadas em conta no desenvolvimento de sistemas paralelos. A primeira está relacionada ao fluxo de dados no algoritmo, assim podemos definir sistemas SIMD e MIMD. Os sistemas SIMD (*single instruction multiple data*) realizam a mesma operação sobre o conjunto de dados paralelamente. Nos sistemas MIMD (*multiple instruction multiple data)* cada unidade processadora realiza um conjunto de instruções independentes sobre os dados.

A outra característica básica é a forma de organização da memória. De acordo como esta característica podemos dividir as máquinas paralelas em máquinas com memória distribuida e memória compartilhada. Nas máquinas de memória distribuida cada unidade processadora ou grupo de unidades (*clusters*) possui uma memória local associada, podendo somente acessar diretamente os dados localizados em sua memória local. Por outro lado, nas máquinas de memória compartillhada todos os processadores acessam uma memória compartilhada global.

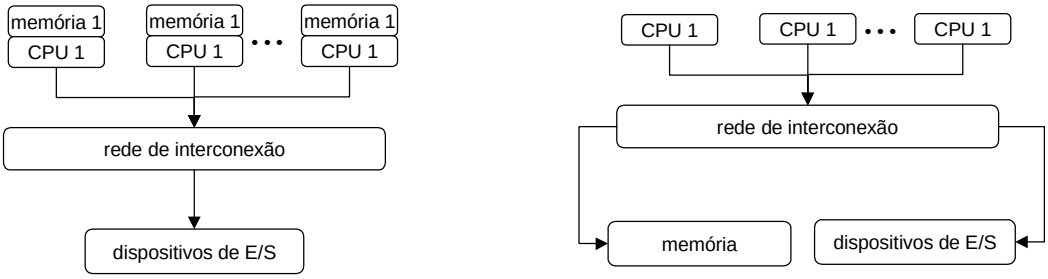


Figura 3.1 Máquinas de Memória Distribuida e Compartilhada

Em geral o paralelismo na solução de um problema pode ser alcançado através de duas abordagens básicas: paralelismo de controle (*pipelining*) e paralelismo de dados. Na primeira abordagem a tarefa a ser executada é subdividida em vários passos, denominados estágios. Cada estágio trabalha na solução de uma parte do problema, e a saída de um estágio se transforma em entrada para o estágio subsequente. O paralelismo de dados é obtido através da utilização de várias unidades processadoras para aplicar conjuntos, iguais ou distintos, de instruções a vários elementos de dados simultaneamente.

Um aspecto importante de sistemas paralelos é sua escalabilidade. Podemos definir escalabilidade como a capacidade de aumento do nível de paralelismo pelo menos linearmente com o aumento do tamanho do problema. Em geral sistemas com paralelismo de dados são mais escaláveis que os com paralelismo de controle.

Da analogia com o trabalho em grupo podemos verificar que um problema central é a divisão do trabalho, referenciado na literatura de processamento paralelo como balanceamento de carga. Este termo se refere ao objetivo de manutenção em sistemas paralelos de uma carga de processamento equilibrada (idealmente igual) entre as várias unidades processadoras. Este problema se torna ainda mais complexo quando além do objetivo de balanceamento da carga, deseja-se minimizar a quantidade de informação trocada entre processadores. Esta abordagem é denominada problema geral do balanceamento de carga. Relacionado a este problema está o conceito de granularidade. A granularidade de um sistema paralelo é uma medida do tamanho das partições do problema. De acordo com este aspecto existem os sistemas de grão grosso e de grão fino.

4. Histórico da Utilização de Paralelismo em *Rendering*.

A incorporação de paralelismo em sistemas de *rendering* tem suas origens nos primeiros dias da computação gráfica. A máquina gráfica Graphic 1, desenvolvida nos Laboratórios Bell no começo da década de 60, usava seu processador interno para gerenciar o dispositivo de visualização e as interações com o usuário, permitindo a operação independente do *mainframe* central (Ninke,1965). Em 1968, Myer e Sutherland (Myer,1968) examinaram a alocação de funcionalidades gráficas em uma configuração multiprocessada, composta por um computador central, um processador para o monitor, e um co-processador para entrada/saída. Neste trabalho foram discutidas as vantagens e desvantagens de utilização de memória compartilhada para realizar a comunicação entre o processador central e o subsistema do monitor, além de ter sido notada a tendência na direção de complexas arquiteturas para dispositivos de visualização. Na mesma época começaram a aparecer máquinas com sistemas gráficos mais sofisticados, incorporando múltiplas unidades funcionais e paralelismo de baixo nível na forma de operações lógicas simultâneas. O clássico "clipping divider" de Sproull e Sutherland (Sproull,1968) representa um exemplo simples destas idéias.

A área de simuladores de vôo em tempo real foi nesta época a mola propulsora do *rendering* em paralelo, propiciando projetos mais ambiciosos, como o de Schumaker *et all* (Schumaker,1969) para a Força Aérea Americana. A arquitetura deste sistema incluía múltiplos processadores e uma variedade de unidades funcionais especializadas, organizadas em três subsistemas de *rendering*: para o terreno, para os objetos, e para fontes de luz pontuais. Durante os anos 70 as aplicações que representaram a grande demanda por processamento gráfico de alta performance foram os simuladores de vôo.

No final dos anos 80 podemos destacar o trabalho de Kaplan e Greenberg (Kaplan,1979) sobre a utilização de algoritmos paralelos para tratar o problema de remoção de superfícies ocultas, que apresentava resultados da implementação em configuração paralela dos algoritmos de *scanline* e subdivisão de áreas (Algoritmo de Warnock (Warnock,1969)). Neste trabalho eles afirmavam vislumbrar um grande futuro para as aplicações de técnicas paralelas neste problema.

A revolução dos VLSI em meados de 1980 representou um ponto de retomada no desenvolvimento das arquiteturas de sistemas gráficos paralelos. A disponibilidade

de processadores compactos, de baixo custo e grande capacidade de memória, tornou possível a utilização de sistemas de alta performance para uso geral. A relativa simplicidade de construção de sistemas replicando componentes encorajou algumas experimentações. Os primeiros projetos baseados neste novo paradigma de *hardware* incluíam sistemas de *rendering* baseados no algoritmo de *z-buffer*, como o sistema proposto por Fuchs e Johnson((Fuchs,1977) e (Fuchs,1979)) e por Parke (Parke,1980), ou então sistemas de *rendering* de polígonos em *pipeline*, como o apresentado por Clark (Clark, 1980).

Durante os anos 80 o uso de múltiplos componentes especializados se tornou a abordagem padrão para a produção de sistemas de *rendering* com alto desempenho em aceleradores gráficos, *workstations* e computadores gráficos especializados. O advento de computadores massiçamente paralelos, contendo de dezenas a centenas de unidades de processamento, adicionou uma nova dimensão ao *rendering* paralelo.

Tão logo as máquinas paralelas de uso geral se tornaram disponíveis em meados de 80, os algoritmos de computação gráfica considerados diretamente paralelizáveis foram implementados nestas máquinas (*ray tracing* e modelagem geométrica por fractais). Em seguida vieram implementações de *rendering* de polígonos e de volumes. Nestas aplicações começou a aparecer o problema básico de computação paralela: “Como particionar um conjunto de dados entre memórias de computadores, de modo a minimizar a comunicação, maximizar a utilização dos processadores, e obter um ganho de velocidade em relação à abordagem seqüencial ?”.

No início dos anos 90 a área de computação gráfica paralela começou a tomar corpo com a realização em Leeds(Inglaterra) da conferência *Parallel Processing for Computer Vision and Display*, seguida em 1993 pelo *Eurographics Rendering Workshop em Bristol* (Inglaterra), e pelo *Parallel Rendering Symposium* organizado pelo IEEE e ACM Siggraph. Estas foram as primeiras conferências na área, reunindo o grande interesse por *rendering* paralelo.

A partir de 1994 com o aparecimento do projeto MAGIC (*Multidimensional Applications GigabitInternet Consortium*) e o início do desenvolvimento de um sistema de visualização de terrenos TerraVision (Leclerc,1994) uma nova dimensão é dada ao problema de *rendering* em paralelo. Neste sistema surge a necessidade de filtragem da base de dados que representa o ambiente para gerar a representação do

subconjunto presente na cena, com a resolução apropriada à posição do observador. Este passo de pré-processamento não aparecia nos sistemas tradicionais de *rendering* que possuíam uma pequena base de dados para representação das cenas. Algumas soluções foram propostas na implementação do sistema e uma nova implementação está sendo gerada na segunda versão do mesmo que está em fase de implementação.

5. Paralelismo no Processo de Rendering.

Se considerarmos o *rendering* de polígonos, temos na Figura 2.1 uma versão padrão das etapas do processamento de uma cena descrita por polígonos. Este processamento consiste de duas partes principais: processamento geométrico (transformações, recorte, iluminação, etc.) e rasterização (transformação para o espaço da imagem, sombreamento, determinação de visibilidade, etc.). Já se observarmos o rendering de volumes (Figura 2.2) verificamos que as principais etapa são a classificação e a projeção dos *voxels*.

Pode-se verificar que diferentes tipos de paralelismo podem ser aplicados ao processo de *rendering*, nos seus vários estágios. Por exemplo: paralelismo de controle pode acelerar computações críticas, e paralelismo de dados pode permitir que vários resultados possam ser obtidos simultaneamente. Alguns tipos de paralelismo são mais apropriados a determinadas aplicações ou a métodos específicos, enquanto outros possuem uma aplicação geral.

A seguir serão apresentados como estes tipos de paralelismo podem ser aplicados no processo de *rendering*.

5.1 Paralelismo de Controle

Uma maneira de obter paralelismo de controle é através da divisão do processo de *rendering* em várias unidades que podem ser aplicadas aos dados em série. Podemos associar a cada etapa do processo uma unidade de processamento, caracterizando a formação do *pipeline* de *rendering*, visto que o grafo de fluxo do processo de rendering é simples e dirigido. A medida que uma unidade termina seu trabalho em um dado, passa este dado transformado à unidade seguinte e pega o próximo dado da unidade anterior. Após o primeiro dado ser tratado por todas as unidades, obtém-se um grau de paralelismo proporcional ao número de unidades processadoras.

Esta abordagem tem sido popular na construção de sistemas gráficos de alto desempenho, desde os anos 60, como foi descrito por Myer (Myer,1968).

James Clark ((Clark,1980) e (Clark,1982)) utilizou esta abordagem para projetar um sistema de computação gráfica altamente paralelo, composto de uma máquina de geometria e um sistema de memória inteligente. A máquina de geometria é composta por 12 *chips* iguais organizados em *pipeline* que realizam as operações de

transformação, recorte e escala. Este módulo pode processar 900 polígonos a cada 1/30 segundos. O outro módulo, o sistema de memória inteligente, é composto por processador pai, um vetor de processadores de memória de imagem e um controle de refrescamento. O vetor de processadores é uma matriz com 8 linhas e 64 colunas, onde cada processador é responsável por 16000 *pixels* de uma tela com resolução de 1024x1024.

Duas características importantes em sistemas em com paralelismo de controle por *pipeline* são: latência e *throughput. Throughput* é a razão total na qual os dados são processados, e latência é o tempo necessário para que um dado simples passe por todo as etapas do fluxo. Algumas aplicações como sistemas gráficos de tempo real (simuladores de vôo) são críticos em relação a latência.

Paralelismo de controle por *pipeline* possui duas grandes limitações :

- a velocidade máxima do sistema. é definida pela unidade mais lenta, propiciando a formação de gargalos;
- o sistema não é escalável, possuindo grau de paralelismo limitado pelo número de estágios no fluxo.

5.2 Paralelismo de Dados

No paralelismo de dados realiza-se uma divisão dos dados em porções, que são processados simultaneamente por diferentes elementos de processamento, como mostrado na Figura 5.1.

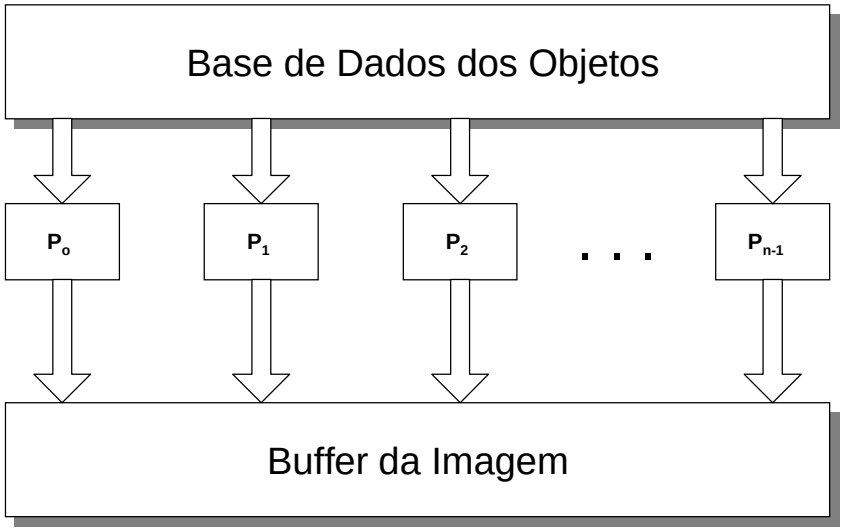


Figura 5.1 Paralelismo de Dados em Rendering

Um sistema deste tipo é altamente escalável, bastando adicionar uma outra unidade processadora e associar a ela um conjunto de dados. Para esta arquitetura a rede de comunicação entre as unidades processadoras é de fundamental importância, sendo a troca de informação efetuada nesta rede o fator limitante da escalabilidade do sistema.

Em função de suas características de escalabilidade, este paradigma tem sido usado pela maioria dos sistemas de rendering desenvolvidos para máquinas massiçamente paralelas de uso geral. Também podem ser desenvolvidos sistemas auto adaptáveis que variem o número de processadores de acordo com características de complexidade da cena, resolução da imagem ou níveis de desempenho desejados.

Assim como os algoritmos de rendering podem trabalhar no espaço da imagem e do objeto, podemos implementar o paralelismo de dados de duas formas: paralelismo de objetos e paralelismo da imagem.

Quando as operações são realizadas de maneira independente no espaço dos objetos que compõem a cena, denominamos paralelismo de objetos. No paralelismo da imagem distribui-se as operações usadas para calcular os valores dos *pixels* para os processadores disponíveis. Em ambos os casos pode-se elevar bastante o nível de paralelismo, uma vez que o número de primitivas[2] para descrever uma cena ou de *pixels* na tela a iluminar, será sempre muito maior que o número de processadores disponíveis.

Em geral, os sistemas de rendering exploram paralelismo de objetos e da imagem, evitando assim o aparecimento de gargalos. No entanto, a definição de balanceamento entre estas duas abordagens é bastante difícil, pois a carga de trabalho envolvida em cada espaço de trabalho é dependente de fatores como: complexidade da cena, área média das primitivas na tela, razão de amostragem, distribuição das áreas das primitivas sobre a tela e resolução da imagem.

Esta abordagem apresenta uma série de vantagens como: a possibilidade de reconfiguração para trabalhar com diferentes algoritmos, e modularidade, que resulta em uma maior facilidade para trabalhar com processadores homogêneos. No entanto, algoritmos baseados em paralelismo de dados necessitam de sincronização e

[2] O termo primitiva será usado para refenciar polígonos ou *voxels*.

balanceamento de carga complexos, além de impossibilitar a utilização de processadores especializados.

6. Projeto de Algoritmos de Rendering Paralelos.

Alguns algoritmos de *rendering* são diretamente paralelizáveis a partir de suas versões seqüenciais, enquanto outros precisam ser completamente reprojetados. Neste processo de desenvolvimento de um algoritmo paralelo para um determinado problema, verifica-se que algumas sobrecargas de trabalho são introduzidas, quando comparado com algoritmos seqüenciais. Em geral estas sobrecargas são provenientes de problemas como :

- comunicação entre processadores;
- atrasos devido a esperas de sincronização;
- computação adicional ou redundante;
- aumento da memória necessária, devido a cópias de dados nas memórias não compartilhadas.

Em geral um algoritmo paralelo pode ser visualizado como contendo quatro etapas básicas, mostradas na Figura 6.1. Esta figura apresenta as principais fontes geradoras das sobrecargas de trabalho geradas pela paralelização.

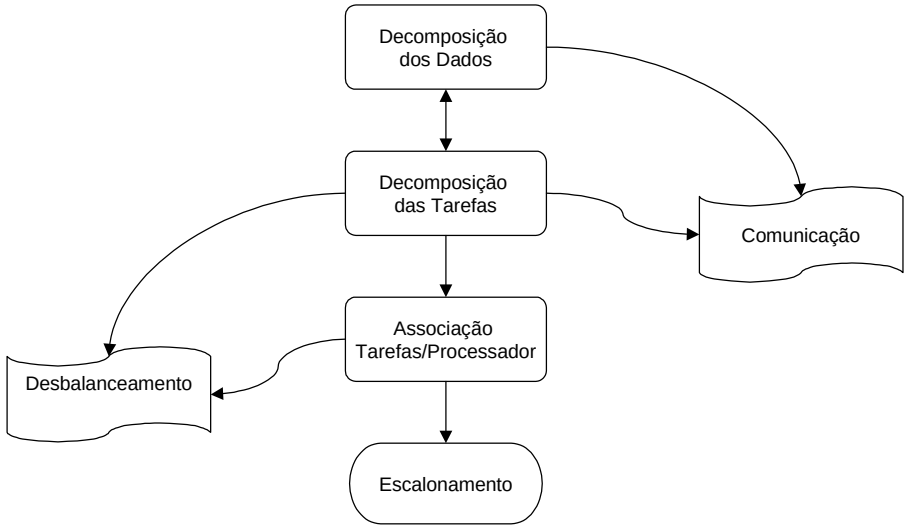


Figura 6.1 Relações Entre Características dos Algoritmos e *Overhead*

Podemos verificar que a base de um algoritmo paralelo é a decomposição do trabalho a ser realizado entre unidades processadoras. Esta decomposição pode ser realizada pela divisão dos dados ou das tarefas a serem realizadas. Em geral estes processos de decomposição geram sobrecargas de comunicação e/ou desbalanceamento nas tarefas a serem realizadas por cada processador. As

sobrecargas de sincronização são em geral derivadas de problemas de desbalanceamento. Estes problemas decorrem da associação inadequada de tarefas a processadores ou da má decomposição do trabalho a ser realizado em cada tarefa. Podemos concluir que os dois problemas centrais no desenvolvimento de algoritmos paralelos são o balanceamento de carga entre processadores e definição da estratégia de comunicação gerada pela divisão de tarefas/dados.

Apresenta-se nesta seção algumas estratégias para solução deste problemas centrais em algoritmos paralelos de *rendering*, além do tratamento de características importantes para a análise e desenvolvimento destes algoritmos.

6.1 Decomposição dos Dados.

A Figura 6.2 apresenta uma classificação proposta por Whitman (Whitman,1992) para estes métodos de decomposição.

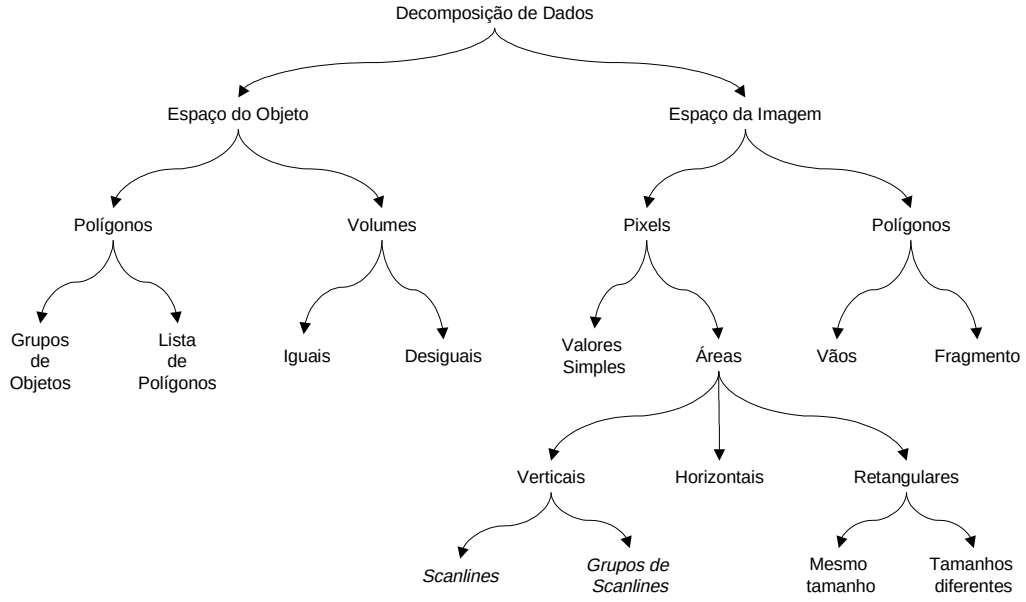


Figura 6.2 Técnicas de Decomposição de Dados

O problema de *rendering* pode ser decomposto em duas fases básicas para efeito de decomposição de tarefas. A primeira fase é o estágio de processamento geométrico (espaço do objeto); nesta fase a base de dados com informações dos objetos que compõem a cena é percorrida, gerando as primitivas da cena. Na segunda fase (espaço da imagem) as primitivas são transformadas para o sistema de coordenadas da imagem, sendo resolvido o problema de visibilidade. Sistemas que

realizam a divisão de tarefas baseados na primeira fase utilizam paralelismo de objetos, enquanto os baseados na segunda fase utilizam paralelismo da imagem.

Um sistema pioneiro na utilização do paradigma de paralelismo de objetos foi o simulador de vôo da GE NASA II (Bunker, 1989) , entregue à NASA em 1967. Este sistema continha várias unidades de *hardware* chamadas *face cards* que processavam polígonos em taxas de vídeo, e na hora de determinar o valor dos *pixels* usava o algoritmo de prioridade por profundidade. Estes sistema utilizava uma decomposição baseada em listas de polígonos, que eram associados a processadores de maneira aleatória.

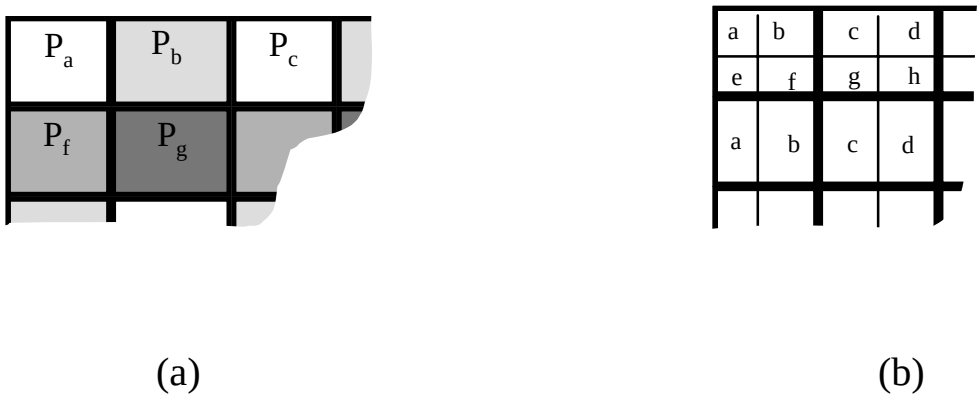
Uma outra utilização deste paradigma está presente na Máquina de *Ray Casting* proposta por Kedem e Ellis (Kedem, 1984). Nesta máquina cada primitiva de um modelo da árvore CSG dos objetos na cena é associada a um processador chamado ***classificador de primitivas*** (CP) e cada operador a um processador chamado ***combinador de classificação*** (CC). A imagem é percorrida na ordem das *scanline, pixel* a *pixel*. Para cada *pixel* o CP calcula o segmento do raio que intercepta a sua primitiva, passando ao CC que realiza as operações necessárias e vai subindo na árvore até chegar à raiz.

Na abordagem baseada no espaço da imagem, duas decisões principais precisam ser tomadas: *Como pode ser particionada a tela? e Quantas partições são necessárias?*

O sistema Pixel Planes 4 (Fuchs,1985) era baseado na divisão por *pixels* do espaço da imagem. Neste sistema cada *pixel* possuía uma ULA (unidade lógica e aritmética) de 1 bit além de uma árvore de acumuladores projetados para calcular eficientemente a equação $F(x, y) = Ax + By + C$. Esta equação é usada para testar se o *pixel* está contido no volume e para cálculos de visibilidade e iluminação. Polígonos são enviados para todos os processadores, e o processador de cada *pixel* determina se é interior ao polígono, realizando os cálculos de iluminação e visibilidade.

A grande maioria dos algoritmos realiza uma divisão dos *pixels* em grupos adjacentes. Duas estratégias obvias são: dividir os *pixels* em blocos contíguos (Figura 6.3a) ou dividir a tela em um padrão como o de um tabuleiro de xadrez. (Figura 6.3b). Nas duas estratégias cada processador é associado a uma partição. Assim durante a rasterização as primitivas são transferidas para os processadores sendo processadas

em relação às partições deles. No caso de lançamento de raios cada processador lança os raios correspondentes à sua porção de tela.



(a) (b)

Figura 6.3 Dois Esquemas para Particionar a Tela

A primeira arquitetura de memória com intervalos, proposta por Fuchs et all ((Fuchs,1977) e (Fuchs,1979)), realizava muitas operações redundantes, pois cada primitiva era completamente processada por cada processador. Clark e Hannah (Clark,1980) propuseram um melhoramento a esta arquitetura para tomar vantagem dos cálculos comuns a múltiplos processadores. A porção de rasterização das máquinas Silicom Graphics de alto desempenho mais recentes usam esta abordagem.

A escolha da abordagem a adotar não é muito fácil. Algoritmos com paralelismo de objetos tendem a distribuir os objetos de forma igual entre os processadores, mas como as primitivas variam em tamanho a rasterização poderá ser desigual. Além disto, primitivas associadas a diferentes processadores podem cair na mesma posição na tela, tornando necessária a realização de uma integração das contribuições individuais para gerar a imagem final. Isto se torna crítico a medida que o número de processadores cresce.

Algoritmos baseados no paralelismo da imagem evitam o passo de integração, mas possuem um outro problema: porções de uma mesma primitiva podem ser mapeadas para diferentes regiões no espaço da imagem. Isto faz com que primitivas, ou porções delas, sejam duplicadas em vários processadores, gerando perda da coerência espacial e resultando em cálculos redundantes.

Para encontrar um equilíbrio entre as abordagens, alguns algoritmos adotam uma abordagem híbrida, incorporando características de ambos os métodos ((Ellsworth,1994), (Crocket,1994), (Salmon,1988) e (Neumann,1994)). Estas técnicas

particionam o espaço da imagem e do objeto, quebrando o *pipeline* de *rendering* no meio e comunicando resultados intermediários entre as tarefas da fase geométrica e da fase de rasterização.

6.2 Aspectos Básicos

Para analisarmos as abordagens paralelas aplicadas ao processo de *rendering*, devemos observar algumas características básicas, tais como:

- natureza do paralelismo;
- Balanceamento de carga;
- nível de granularidade;
- utilização da coerência gráfica;
- acesso aos dados;
- escalabilidade.

A natureza do paralelismo foi abordada no capítulo anterior, onde verificou-se que os algoritmos poderiam utilizar paralelismo funcional ou paralelismo de dados.

As demais características são abordadas a seguir.

6.2.1 Balanceamento de Carga

Nos sistemas paralelos, verifica-se que a utilização efetiva dos processadores depende de uma partição equilibrada do trabalho a ser realizado no sistema. Este objetivo é extremamente difícil de ser alcançado em sistemas de *rendering* em paralelo.

O balanceamento de carga se refere à idéia de que cada processador deve, em um caso ideal, receber a mesma carga de processamento e terminá-lo ao mesmo tempo que os demais. Em geral este problema é tratado através de esquemas de particionamento de tarefas que tentam criar uma divisão igualitária da carga distribuída aos processadores. Estes esquemas podem ser agrupados em duas classes: balanceamento estático e balanceamento dinâmico.

No método estático o número de tarefas T é dividido de maneira equilibrada entre os P processadores, e assume-se que a demanda de processamento será compatível com a capacidade de processamento de cada um. Esta abordagem requer um processamento adicional no inicio da execução do algoritmo, para determinara o tamanho de cada tarefa de maneira compativel com a capacidade de cada processador. Entre as vantagens desta abordagem podemos citar a pequena sobrecarga de

comunicação durante o processamento, a diminuição da sobrecarga no início da execução das tarefas e durante o escalonamento.

Na abordagem dinâmica determina-se *T* muito maior que *P*, e a associação de tarefas a processadores é feita durante a execução. Um processador continua trabalhando nas tarefas até que todas elas acabem, quando então fica inativo até o final do processamento do sistema. Como resultado da pequena carga relativa de cada tarefa, o tempo de inatividade de cada processador é pequeno resultando em impacto mínimo na performance geral do sistema.

Entre as vantagens desta abordagem temos a dispensa da estimativa do tempo de execução das tarefas, a forma dinâmica e adaptável de solução da distribuição de carga, e menor tempo gasto com a determinação do tamanho das tarefas.

Considere um sistema paralelo de *rendering* de polígonos que utilize o espaço do objeto para divisão do trabalho. Primeiro, devido ao diferente numero de vértices de cada polígono, teremos um número diferente de cálculos para operações de transformação e iluminação em cada tarefa. O tamanho das primitivas quando transformadas para a tela também varia resultando em um número de operações diferentes na rasterização. No entanto como o número de primitivas é muito grande e as mesmas podem ser associadas randomicamente aos processadores, em geral não temos grandes variações na carga de trabalho.

Mas há uma outra causa de desbalanceamento, que é mais séria, quando a divisão do trabalho é baseada no espaço da imagem. Nas cenas reais, a distribuição das primitivas sobre o espaço da imagem tende a ser agrupada em determinadas regiões, não sendo uniforme. Assim um processador que esteja responsável por rasterizar uma região densa terá muito mais trabalho que um outro processador que vai somente processar *pixels* com a cor do fundo.

Para tornar mais complicado o problema, verifica-se que o mapeamento entre o espaço do objeto e o espaço da imagem é dependente da projeção. Ou seja, a distribuição de primitivas sobre a imagem pode mudar de um quadro para o outro. Isto é crítico em aplicações interativas. A seguir apresenta-se algumas estratégias para tratar este problema.

6.2.1.1 Estratégias Estáticas.

Estas estratégias se baseiam em um particionamento fixo dos dados de modo a distribuir variações locais por um grande numero de processadores.

A Figura 6.4 mostra diferentes estratégias para particionamento da imagem com diferentes características de balanceamento. É mostrado um esquema de divisão em blocos contíguos de *scanlines* (a), regiões quadradas (b), a *scanlines* intercaladas (c), *pixels* alternados em 2D (d), e partição adaptativa (e).

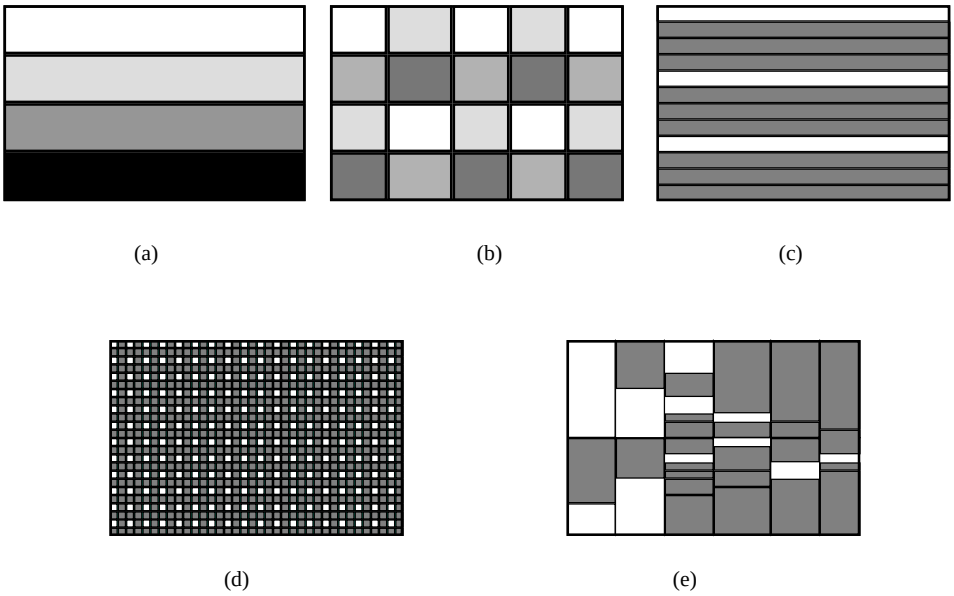


Figura 6.4 Estratégias para Particionamento da Imagem

Existem muitas maneiras de subdividir a imagem. Dois parâmetros nesta divisão são a forma e o número de sub-regiões. Foi mostrado analítica e empiricamente em ((Whelan,1985) e (Whitman,1992)) que regiões quadradas resultam em uma menor perda de coerência. A justificativa para isto reside no fato de regiões quadradas fornecerem um menor fator de sobreposição[3] para polígonos distribuídos randomicamente. Pode ser observado intuitivamente que uma faixa estreita vai produzir um maior número de sobreposições que uma região quadrada com a mesma área.

Encontrar o melhor número de regiões envolve um paradoxo. Por um lado, aumentando-se o número de regiões melhora-se o balanceamento de carga da rasterização. Por outro lado, aumenta-se a necessidade de comunicação entre os processadores, devido a um aumento no número de sobreposições.

[3] sobreposição refere-se à projeção de um polígono em duas ou mais sub-regiões da tela.

6.2.1.2 Estratégias Dinâmicas.

Esquemas de balanceamento dinâmico de carga tentam melhorar as estratégias estáticas, através de uma maior flexibilidade na associação do trabalho ao processador. Existem duas abordagens principais.

A abordagem baseada na demanda decompõe o problema em um grande numero de tarefas independentes, que são associadas a um processador, uma por vez ou em grupos pequenos. Quando o processador completa uma tarefa ele recebe outra, e o processo continua até que todas as tarefas sejam realizadas. Se as tarefas possuem grande variação no tempo de processamento, então as mais caras são iniciadas primeiro, de modo a terem tempo de serem completadas, enquanto os outros processadores estão ocupados com as tarefas menores.

Os tempos de execução de cada tarefa são estimados heuristicamente em um pré-processamento, o qual representa uma sobrecarga de trabalho.

Nesta abordagem verifica-se que ocorrem falhas de balanceamento quando processadores ficam parados, ao final do processamento ou em um instante intermediário, esperando as tarefas longas se completarem. Uma alternativa para evitar este problema é usar um grande número de tarefas pequenas para minimizar a variação de tempo de execução, mas esta alternativa resulta em sobrecarga de processamento devido à perda de coerência e à grande freqüência nas operações de associação de tarefas a processadores.

Uma abordagem adaptativa, baseia-se na tentativa de minimizar a sobrecarga de pré-processamento. Isto é feito através de uma divisão inicial de tarefas de grão fino, permitindo posteriormente que um processador solicite tarefas quando em estado de espera. Neste instante o trabalho restante em cada um dos processadores ativos é dividido e associado aos processadores inativos. Nesta estratégia verifica-se que o particionamento dos dados não é predeterminado, mas sim se adapta à carga computacional.

Este esquema adaptativo foi implementado inicialmente no contexto de métodos de rasterização ((Whelan,1985) e (Whitman,1992)). Métodos similares também foram usados em *ray tracing* e *rendering* de volumes. Um bom exemplo da utilização deste método é apresentado no trabalho de Whitman (Whitman,1992). O sistema de *rendering* de Whitman inicialmente particiona o espaço da imagem em um número relativamente pequeno de tarefas de grão grosso, que são associadas aos

processadores usando a abordagem baseada na demanda. Quando um processador entra em estado de espera e a fila de tarefas está vazia, o sistema procura o processador com maior carga de trabalho por realizar e divide com o processador na espera. A principal sobrecarga dos sistemas baseados nesta abordagem é proveniente da manutenção e recuperação de informações de estado não locais, das tarefas de particionamento, e na migração de dados.

Embora os esquemas dinâmicos de balanceamento de carga ofereçam maior potencial para um balanceamento mais preciso, eles obtêm sucesso somente quando a melhora na utilização dos processadores compensa a sobrecarga computacional gerada. Por esta razão, verifica-se que sistemas com balanceamento dinâmico de carga são mais fáceis de implementar em arquiteturas que fornecem baixo tempo de espera para acesso a memória compartilhada. Em sistemas com troca de mensagens, o alto custo de referências a posições não locais de memória torna muito cara a associação dinâmica de tarefas, a migração de dados e a manutenção de informações globais de estado.

Ellsworth (Ellsworth,1994) tentou ultrapassar esta limitação aplicando um esquema de balanceamento de carga entre quadros. Ao invés de realizar o balanceamento de carga de um único quadro, seu sistema de *rendering* utiliza a distribuição da carga de trabalho de um quadro para o outro para realizar a associação processadores/região da imagem. Este método tenta tirar partido da coerência de quadro, característica de uma sequência consecutiva de imagens de manter a distribuição de polígonos sobre a imagem de maneira similar. Esta estratégia possui a vantagem de realizar o balanceamento de carga em um alto nível de granularidade, com menos sobrecarga. No entanto os experimentos mostraram que o sucesso desta estratégia é parcial, pois a mesma apresenta problemas de escalabilidade na obtenção da informação global da carga de trabalho para um grande número de processadores.

6.2.1.3 Balanceamento de Carga em Sistemas de Lançamento de Raios.

Em algoritmos de lançamento de raios, a maior parte do tempo de execução é gasta no cálculo das interseções entre os raios e as primitivas (*voxels* ou polígonos) que compõem a cena. Nestes sistemas o desbalanceamento de carga pode resultar dos diferentes custos para cálculo de interseção (objetos implícitos , poligonais, etc) e para avaliação dos raios refletidos, que dependem do tipo e da distribuição dos objetos na cena. Em *rendering* de volumes a maior causa de desbalanceamento são as diferentes

profundidades que o raio precisa percorrer no volume, dependente dos valores de opacidade dos *voxels* em seu caminho.

Em sistemas de lançamento de raios com paralelismo de dados existem duas técnicas principais para distribuir os objetos entre os processadores. Em uma abordagem a cena é particionada de acordo com uma árvore ((Caspary,1989) e (Salmon,1988)), enquanto no outro a cena é subdividida em regiões tridimensionais ((Cleary,1983), (Dippé,1984), (Jevans,1989), (Kobayashi,1987) e (Nemoto,1986)).

Na distribuição baseada em árvore, é criada uma árvore hierárquica de caixas de fronteira (*bounding box*), a qual é usada para calcular as interseções. Somente a árvore superior é replicada em cada um dos processadores. Os níveis mais baixos desta árvore são ponteiros para subárvores da hierarquia. Então cada processador controla um subconjunto de *pixels* e uma subárvore correspondendo a um conjunto de primitivas da cena. Para calcular a cor de um pixel, um processador lança um raio primário e segue-o através da árvore até descobrir que processador está encarregado, e caso necessário envia o raio para o processador apropriado. Esta abordagem tem a desvantagem de ter o balanceamento de carga baseado apenas na distribuição dos *pixels*.

Pode-se usar uma divisão espacial da cena (e.g. árvore bsp) como estrutura de dados do objeto, associando-se a cada processador uma ou mais regiões. Uma forma de fazer o balanceamento de carga dinamicamente é através de mensagens de redistribuição que movem os contornos das regiões alocadas a cada processador. Esta abordagem leva a um grande número de operações de ponto flutuante e de mensagens trocadas. O balanceamento de carga estático pode ser feito associando regiões 3D vizinhas a processadores não adjacentes.

6.2.2 Nível de Granularidade

Em computação paralela define-se como tamanho do grão a quantidade de trabalho realizada em uma unidade processadora em relação á quantidade de informação trocada. Quando esta quantidade é pequena, diz-se que o sistema é de grãos finos, e quando é grande de grão grosso.

Este conceito de granularidade está intimamente relacionado com a subdivisão do trabalho entre as unidades processadoras, balanceamento de carga, e consequentemente com a eficiência do sistema. Sistemas de grão fino geralmente necessitam de grande sobrecarga de trabalho para escalonamento das tarefas e

comunicação. Os sistemas de grão grosso minimizam esta sobrecargas, mas são mais suscetíveis a desbalanceamento de carga, impondo limites à quantidade de paralelismo disponível.

6.2.3 Utilização da Coerência Gráfica

Em computação gráfica, coerência se refere à tendência que as imagens possuem de apresentar características semelhantes em pontos próximos espacialmente ou no tempo. Este conceito é importante para *rendering*, pois de uma forma ou de outra todos os algoritmos o exploram. Os sistemas paralelos de *rendering* devem preservar o uso da coerência ao máximo, de modo a evitar um aumento na quantidade de cálculos a serem realizados, minimizar a necessidade de comunicação e melhorar o balanceamento de carga.

Entre os tipos de coerência mais explorados podemos citar :

- de quadro - a imagem não muda muito de um quadro para o outro.
- espacial - um elemento particular da imagem e seus vizinhos em todos os lados tem grande tendência a possuírem as mesmas características.
- de raio - tendência de raios de *pixels* vizinhos interceptarem os mesmos objetos na cena.

No projeto de algoritmos paralelos deve ser observada a probabilidade de uma primitiva interceptar um contorno de uma sub-região gerada pela divisão do trabalho entre os processadores, dependendo do tamanho, forma, e números destas partições. Isto, porque a coerência é perdida nas fronteiras destas regiões, resultando em um acréscimo de carga computacional. Elsworth (Ellsworth,1994) e Molnar *et all* (Molnar,1994) definem uma medida para esta perda de coerência, dada pelo fator de sobreposição de polígonos, o qual é dado pelo número de sub-regiões em que um determinado polígono está sobreposto na tela. Molnar sugere ainda uma fórmula para calcular este fator, dada por $O = ((W+w)/ W)+((H+h)/h)$, onde $W \times H$ é o tamanho da imagem e $w \times h$ é o tamanho típico da caixa de fronteira do polígono.

6.2.4 Acesso aos Dados

Ataulamente o acesso aos dados tem se tornado um gargalo de sistemas de visualização baseados na imersão do observador no ambiente a ser visualizado.

Inicialmente este aspecto estava relacionado aos movimentos de dados entre módulos de memória, e entre memória e disco. Algoritmos gráficos trabalham com

grande quantidade de dados, o que requer um bom gerenciamento da memória para alcançar bons resultados de desempenho. Acesso remotos em ambientes de memória distribuída degradam a performance do algoritmo.

Como os conjuntos de dados que representam as cenas são grandes, não é aconselhável gerar cópias de todos estes dados em cada nó de processamento. Isto está relacionado ao tamanho da memória local do processador e ao esforço gerado pela distribuição de grande quantidade de dados entre eles.

Ultimamente duas variáveis novas foram adicionadas a esta questão. Como filtrar a base de dados que representa todo o ambiente a ser visualizado de modo a gerar somente os objetos que correspondem à cena especificada, e assim evitar o cálculo de iluminação e visibilidade em um grande número de obejtos contidos completamente fora da cena. E, como obter estes objetos da base de dados em uma resolução adequada à posição do observador e à direção de visualização. Este problema adicional se faz presente em sistemas de rendering de grandes extensões de terrenos sintéticos, e em sistemas de realidade virtual baseados em polígonos, nos quais presume-se que uma subdivisão espacial auxiliará na eliminação dos objetos fora da cena, e um esquema de representação dos dados em multiresolução estará associado. Em sistemas deste tipo a paralelização do algoritmo de *rendering* é tão importante quanto o algoritmo de acesso aos dados.

6.2.5 Escalabilidade.

Uma característica importante de um algoritmo paralelo é a capacidade de gerar uma boa aceleração em configurações com grande número de processadores.

Um sistema paralelo é dito escalável se o nível de paralelismo aumenta pelo menos linearmente com o tamanho do problema. Este conceito esta relacionado à capacidade do sistema de fornecer poder de computação adicional a medida que o número de unidades de processamento é aumentado. Este aspecto é bastante importante pois permite ao usuário resolver problemas maiores na mesma quantidade de tempo apenas comprando um número maior de unidades processadoras.

Pode-se verificar que algoritmos com paralelismo de dados, em geral, são mais escaláveis que algoritmos com paralelismo de controle.

A arquitetura do sistema para acesso aos dados é de extrema importância na definição de sua escalabilidade. Os sistemas de memória compartilhada tem sua escalabilidade limitada pela largura de banda do barramento de comunicação ou por

sua rede de conexão processadores/memória. Isto advém do fato de, ao acrescentarmos processadores não aumentarmos a largura de banda do barramento, resultando na saturação dos caminhos entre os processadores e a memória, e em consequente queda de desempenho.

Esta é uma das razões que levam os sistemas que pretendem alta escalabilidade (trabalhar com centenas de processadores) a usarem o modelo de memória distribuída, no qual cada unidade processadora possui a sua memória local associada. Nestes sistemas os conjuntos processadores/memória local são interligados por uma rede de comunicação relativamente escalável. Tais sistemas apresentam a grande desvantagem de que referências a dados não encontrados na memória local podem ser muito mais lentas que referências a dados locais.

6.3 Algoritmos Diretamente Paralelizáveis.

Uma classe de algoritmos paralelos de *rendering* é a classe dos algoritmos diretamente paralelizáveis. Esta classe engloba os algoritmos paralelos que são transformados a partir dos seqüenciais com uma sobrecarga de computação mínima e com baixas taxas de comunicação entre processadores. São geralmente enquadrados nesta categoria os métodos baseados em lançamento de raios *(ray-casting* ou *ray-tracing).* Nestes métodos temos que os valores dos *pixels* são calculados através do lançamento de raios a partir de cada pixel em direção à cena, sendo bastante natural a decomposição das tarefas baseadas no espaço da imagem. Cada raio pode ser processado independentemente, caso os processadores tenham acesso rápido aos dados que definem a cena. Esta abordagem é muito prática para arquiteturas de memória compartilhada, sendo também de muito utilidade quando utilizada em um sistema de memória distribuída com capacidade para replicar os dados em cada porção da memória.

7. Classificação de Sistemas de *Rendering* Paralelo.

7.1 Sistemas de Rendering de Polígonos

Algumas classificações para algoritmos paralelos de *rendering* de polígonos foram propostas por ((Crow,1988), (Gharachorloo,1989), (Molnar,1990), (Whitman,1992)). Estas classificações auxiliam na compreensão dos sistemas mas não possibilitam por si só uma maneira para realizar comparações. Vamos apresentar um sistema de classificação, proposto por Molnar et all (Molnar,1994), baseado no ponto do *pipeline* de *rendering* onde é realizado o mapeamento entre o espaço do objeto e o espaço da imagem. Os algoritmos são classificados em *sort-first*, *sort-midle* e *sort-last*, de acordo com o ponto do *pipeline* de *rendering* onde ocorre o passo de comunicação.

7.1.1 Sort-First

Esta classe de algoritmos distribui as primitivas no inicio do *pipeline* de *rendering* para os processadores que realizam o restante dos cálculos. Isto pode ser feito dividindo a tela em regiões disjuntas e fazendo os processadores responsáveis por todos os cálculos de *rendering* que afetam sua região de *pixels*. Nesta classe associa-se primitivas a um processador, que projeta a primitiva e verifica em que região da tela ela é mapeada. Este processador então envia a primitiva para o processador correto, que realiza a rasterização e os outros cálculos de *rendering*. Esta é a classe de algoritmos menos explorada.

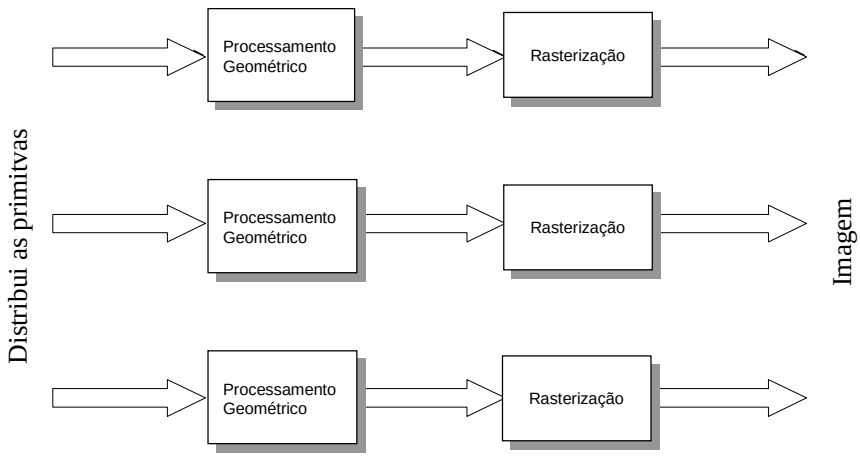


Figura 7.1 *Pipeline* de *Rendering* de Algoritmos *Sort-First*.

Nesta classe de algoritmos a maior parte das primitivas vai estar associada ao processador errado e vai necessitar de redistribuição. No entanto se forem processados vários quadros na seqüência pode-se tomar vantagem da coerência quadro a quadro de modo a reduzir a sobrecarga de comunicação. Para isto basta manter as primitivas que foram finalizadas por cada processador para o próximo quadro.

Pode-se verificar que esta classe de algoritmos possui duas vantagens: requisitos de comunicação são baixos quando a coerência quadro a quadro é utilizada, e um nó de processamento implementa todo o processo de *rendering*.

Estes algoritmos são muito suscetíveis a desbalanceamento de carga e necessitam de um código complexo para poder tirar vantagem da coerência quadro a quadro.

7.1.2 Sort-Middle.

Em algoritmos da classe *sort-middle* as primitivas são redistribuídas no meio do *pipeline* de *rendering* (Figura 7.2), entre o processamento geométrico e a rasterização. No meio do *pipeline* de *rendering* as primitivas estão transformadas em coordenadas da imagem e prontas para rasterização.

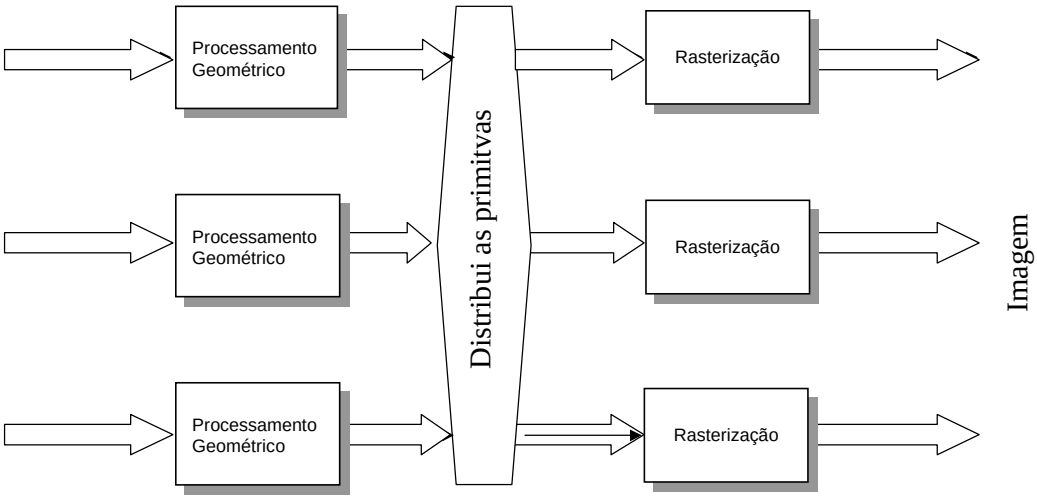


Figura 7.2 *Sort-Middle* no *Pipeline* de *Rendering*

Nesta classe, associa-se subconjuntos arbitrários de primitivas a cada processador geométrico, e aos processadores de rasterização associa-se uma porção da tela. Os dois conjuntos de processadores podem ser distintos ou não. Durante cada quadro os processadores geométricos transformam, iluminam e realizam outras operações geométricas no seu conjunto de primitivas, além de classificar cada

primitiva com respeito as regiões da tela. Então, este processadores enviam todas as primitivas mapeadas na tela para os processadores de rasterização respectivos.

Esta tem sido a abordagem mais comum para sistemas de *hardware* ((Fuchs,1993), (Akeley,1993) e (Deering,1993)) e *software* ((Crockett,1993), (Whitman,1994) e (Ellsworth,1994)). Isto decorre do fato destes algoritmos serem gerais e de fácil implementação, o que se dá em razão da redistribuição ocorrer em um lugar natural. Por outro lado, se o número de primitivas for alto, temos um alto custo de comunicação, e pode ocorrer desbalanceamento devido à concentração de primitivas em regiões da tela.

7.1.3 Sort-Last

Esta classe de algoritmos retarda a redistribuição até que as primitivas estejam rasterizadas em *pixels*. Nestes algoritmos cada processador é associado a um conjunto arbitrário de primitivas. Cada processador calcula os valores dos *pixels* para seu conjunto de primitivas, em seguida envia estes *pixels* para processadores de composição que resolvem o problema de visibilidade dos *pixels*.

Nesta classe de algoritmos cada processador opera independentemente até o estágio de determinação de visibilidade. A comunicação no final do processo pode ser realizada de duas maneiras. No modo denominado *Sl-esparso,* somente os *pixels* produzidos pelos processadores de rasterização são distribuídos, no modo *Sl-full* uma imagem completa é armazenada e enviada por cada processador.

Nesta classe de algoritmos cada nó de processamento implementa o *pipeline* de *rendering* completo para um conjunto das primitivas, sendo menos suscetível a desbalanceamento. No entanto o tráfego de pixels pode ser extremamente alto, principalmente quando forem utilizadas técnicas de *antialiasing* como superamostragem. Alguns sistemas comerciais de *rendering* tem usado esta classe de algoritmos ((Evans,1992) e (Cray,1994)).

7.2 Sistemas de Rendering de Volumes

Foi proposta por Wittenbrink (Wittenbrink,1998) uma classificação das técnicas de rendering de volumes em paralelo baseda em dois aspectos centrais aos algoritmos:

- Transformações usadas entre os espaço do objeto e o espaço da imagem;
- *Hardware* alvo do sistema.

Podemos citar as seguintes transformações possíveis entre o espaço dos objetos (volume) e o espaço da imagem: *forward*, *backward*, domínio e *forward* de multipasso. Estas transformações estão esquematizadas no grafo da Figura 7.3.

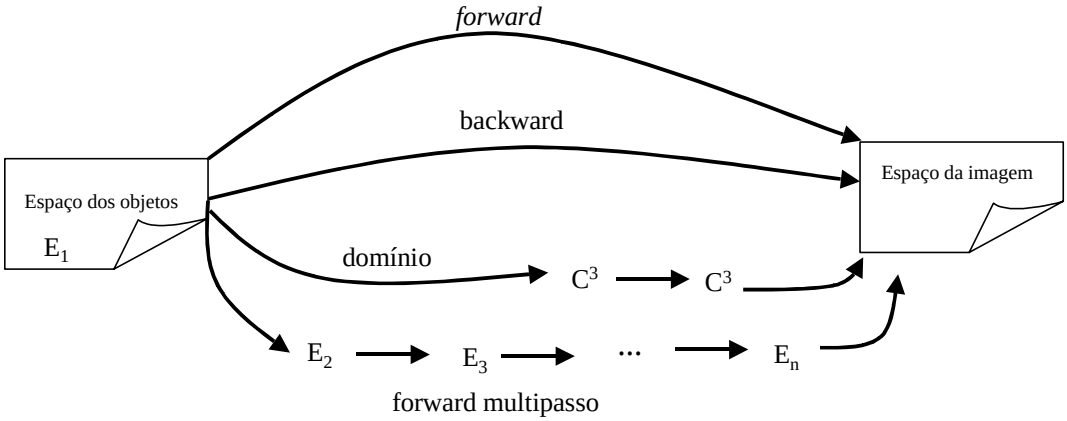


Figura 7.3 Grafo de Transição (espaço dos objetos *x* espaço da imagem).

O método *forward* utiliza a transformação natural *T*, do espaço do objeto no espaço da imagem; para isto transforma cada *voxel* do volume, adicionando sua contribuição ao valor dos *pixels* que se encontram em sua área de influência. A transformação *backward* utiliza a projeção inversa T^{1}. Neste método transforma-se cada *pixel* do volume para o espaço dos objetos, computando a contribuição de todos o *voxels* que contribuem para seu valor final. Um método *forward* multipasso utiliza uma decomposição da transformação *T* em T_1, $T_2 \ldots T_n$, de modo que a contribuição do *voxel* aos valores dos *pixels* é obtida através do cálculo de vários valores de *voxel* intermediários. Os métodos de domínio realizam a transformação do espaço do objeto para um espaço intermediário (frequências de Fourier, coeficientes de *wavelet*, compressão, etc.).

Com relação ao *hardware* alvo, podemos classificar as técnicas de rendering paralelo em:

- Gráfico: *hardware* gráfico especialmente projetado para *rendering* de polígonos;
- Volume: *hardware* gráfico especialmente projetado para *rendering* de volumes;
- Paralelo de Memória Compartilhada: máquinas paralelas de uso geral com memória compartilhada;
- Paralelo de Memória Distribuída: máquinas paralelas de uso geral com memória distribuída;
- Distribuído: máquinas distribuídas de uso geral (e.g. redes de workstations).

Baseada nesta proposta, a Tabela 7-1 apresenta a classificação de alguns sistemas clássicos de rendering de volumes em paralelo.

Referência	Descrição	Hardware Alvo	Tansformação
(Drebin,1988)	Shearing	Gráfico/SIMD	Forward multipasso
(Giertsen,1993)	Ray Casting	Distribuido/MIMD	Backward
(Gunther,1994)	VIRIM	Volume	Backward
(Junhui,1997)	Fourier	Distribuido	Domínio
(Lacroute,1996)	Shear Warping	Paralelo de Memória Compartilhada/MIMD	Forward multipasso
(Ma,1994)	Binary Swap	Distribuido/MIMD	Backward
(Montani,1992)	Ray Casting	Paralelo de Memória Distribuida/MIMD	Backward
(Neumann,1993)	Ray Casting	Paralelo de Memória Compartilhada/MIMD	Backward
(Pfister,1996)	Cube4	Volume/SIMD	Forward multipasso
(Schröeder,1991)	Shearing	Paralelo de Memória Distribuida/SIMD	Forward multipasso
(Schröeder,1992)	Line Drawing	Paralelo de Memória Distribuida/SIMD	Forward multipasso
(Gelder,1996)	3D Texture	Gráfico/MIMD	Backward
(Westover,1989)	Splatting	Gráfico/SIMD	Forward
(Wilhelms,1996)	Cell Projection	Paralelo de Memória Compartilhada/MIMD	Forward
(Wittenbrinck,1997)	Permutation Warping	Paralelo de Memória Distribuida/SIMD	Backward
(Wittenbrinck,1998)	Projected Tetrahedra	Gráfico/MIMD/SIMD	Forward
(Yoo,1991)	Ray Casting	Gráfico/MIMD/SIMD	Backward

Tabela 7-1 Classificação de Algoritmos Paralelos de Rendering de Volumes

8. Sistemas de Rendering de Polígonos em Paralelo

Nesta seção apresentaremos algumas implementações de sistemas de *rendering* de polígonos. Serão apresentadas as características de arquitetura do sistema, sem a preocupação com comparações de desempenho.

8.1 Algoritmos

8.1.1 Z-buffer

Este algoritmo é o mais popular para o *rendering* de polígonos. Tem sido inclusive base para inúmeras aplicações em *hardware*. Em (Crocket,1994) temos um algoritmo baseado em *z-buffer* rodando em máquinas MIMD de uso geral com memória distribuída. Esta implementação trabalha com uma lista de triângulos que descrevem a cena e um *buffer* de memória da imagem, distribuídos regularmente ao longo dos processadores. Para os triângulos utiliza-se uma estratégia do tipo *round robin* enquanto a imagem é dividida por faixas horizontais iguais. Dados comuns a toda cena (posição do observador, das fontes de luz, etc.) são replicados em cada um dos processadores.

```
- Cada p mantém lista de vizinhos
  - Após processar todos os seus triângulos locais p manda
  mensagem para vizinhos avisando que acabou
  - quando pj recebe mensagem de finalização de todos os
   vizinhos manda finalização para o principal
- quando principal recebe finalização de todos, termina
  o algoritmo
```

Algoritmo 8-1 Finalização do *Z-buffer* paralelo

Primeiro, cada processador transforma, ilumina e recorta (*clipping*) seus triângulos locais, o que gera triângulos 2D mapeados no espaço da imagem. Caso necessário, o processador divide o triângulo em trapezóides segundo a faixa de *scanlines* que está responsável. Assim os pedaços de triângulo que não são locais são enviados para o processador correspondente. Finalmente, os pedaços de triângulos são rasterizados em seus processadores respectivos, com um processo *z-buffer*. O padrão de comunicação exibido por este algoritmo é completamente dependente da cena e da posição do observador. A grande dificuldade existente neste algoritmo é a

determinação do instante de terminação do algoritmo, pois não existem pontos de sincronismo. Assim, foi implementado o seguinte algoritmo para detecção da terminação.

Whitman (Whitman,1994) usa uma partição inicial estática do espaço da tela com subsequente partição dinâmica para balanceamento de carga. O espaço da imagem é dividido em regiões iguais com razão de granularidade de 2, ficando cada processador responsável por duas áreas. Utilizando um abordagem *sort-middle*, a geometria é transformada e então enviada para os processadores onde será realizado o *rendering*. Partições dinâmicas ocorrem quando um processador completa o processamento de suas regiões. Este processador divide a lista de tarefas do processador mais carregado, pegando metade de sua carga de trabalho. Em (Le Mair,1993) foi mostrado que esta estratégia está bem próxima da ótima. Uma aceleração linear foi conseguida com a implementação deste algoritmo em uma arquitetura de memória compartilhada distribuída.

O algoritmo de Ellsworth (Ellsworth,1994) foi desenvolvido para *rendering* interativo, usando balanceamento de carga entre quadros. A cada processador são associadas várias regiões da imagem. Um algoritmo guloso baseado no número de polígonos é usado para determinar que regiões associar a que processadores, de modo a conseguir melhor balanceamento de carga. O envio dos polígonos ocorre entre quadros; assim, cada processador realiza o *rendering* somente dos polígonos locais em seu *frame buffer* local. Esta implementação foi realizada em uma máquina Intel Touchstone Delta.

8.1.2 Ray Tracing

Ray tracing foi um dos primeiros algoritmos a serem implementados em computadores paralelos. É um algoritmo bastante eficiente para *rendering*, que pode gerar imagens de muito boa qualidade. No entanto este método requer um grande esforço computacional. Por outro lado, trata-se de um algoritmo diretamente paralelizável, que pode ser paralelizado de diversas formas. Entre as formas de paralelizar um *ray tracing*, podemos destacar :

i. Paralelismo de componentes: os cálculos de um simples raio podem ser paralelizados. Por exemplo, cálculos de reflexão, refração e interseção, envolvem o cálculo de componentes (x ,y, z) de vetores ou pontos. Esta três

componentes podem ser calculadas em paralelo alcançando aceleração de um fator de três.

ii. Paralelismo da imagem: as interseções raio-primitivas podem ser calculadas em processadores separados, pois são independentes. Para tomar partido do paralelismo é necessário que todos os processadores tenham acesso a toda a base de dados dos objetos que compõem a cena, pois cada raio pode atingir qualquer uma das regiões da cena.

iii. Paralelismo de objetos: primitivas na base de dados podem ser distribuídas espacialmente por múltiplos processadores. Cada processador fica responsável por todos os raios que passam por sua região. O processador calcula a interseção raio-objeto se o raio atinge um objeto, e envia o raio para o próximo processador se necessário.

A maior parte das arquiteturas projetadas para *ray tracing* usa um grande número de processadores MIMD. O tipo mais simples de arquitetura paralela por imagem associa um ou mais raios a cada processador e replica a base de dados em cada processador. Esta técnica foi usada no sistema LINKS-1 (Nishimura,1983), construído na Universidade de Osaka. Este sistema foi usado para gerar inúmeras seqüências de animação.

O primeiro sistema proposto com paralelismo de objetos usava subdivisão espacial para associar porções do universo a processadores. Isto resultou em pouca eficiência para cenas com primitivas concentradas em certas sub-regiões. Em razão disto alguns sistemas (Dippé,1984) subdividem a cena adaptativamente para melhorar o balanceamento de carga. Esta técnica torna complicado o mapeamento entre raios e processadores.

Em razão das vantagens oferecidas pelos computadores paralelos comerciais, baixo custo e ambientes maduros de programação, a pesquisa em *ray tracing* paralelo encaminhou-se para o desenvolvimento de algoritmos eficientes que funcionassem nestas máquinas .

Um sistema de *ray tracing* com paralelismo da imagem foi desenvolvido para Thinking Machines (Delany,1988), no qual a base de dados é enviada em *broadcast* para todos os processadores, que realizam os cálculos de interseções entre raios e objetos em paralelos.

Nas implementações em sistemas de memória compartilhada, como BBN Butterfly (Jenkins,1989), a base de dados não precisa ser armazenada em cada

processador, ao invés disto, os processadores solicitam porções da base de dados da memória a medida que é necessário.

Nemoto e Omachi (Nemoto,1986), Kobayashi e Nakamura (Kobayashi,1987), Scherson e Caspary (Scherson,1988), e outros (veja (Jevans,1989)) propuseram vários métodos para associar adaptativamente objetos a processadores e para enviar raios de um processador a outro. Tais sistemas possibilitam vislumbrar a performance possível de ser alcançada em *ray tracing* paralelos.

8.1.3 Radiosidade

Os métodos de radiosidade são apropriados para o cálculo da iluminação global de uma cena contendo superfícies com reflexão difusa.

Neste método as superfícies que descrevem a cena são subdivididas em elementos pequenos que possuem radiosidade aproximadamente uniforme. A radiosidade de um elemento *i* pode ser expressa como uma combinação linear das radiosidades de todos os outros elementos *j*, resultando em um sistema linear de equações. Os coeficientes na combinação linear são os fatores de forma entre elementos, correspondente à fração de energia que sai de *j* e chega em *i*. O cálculo dos fatores de forma é a tarefa computacionalmente mais cara do algoritmo de radiosidade. O número de fatores de forma entre todos os *n* elementos é $O(n^2)$, e cada um destes pares tem que ser testados para intervisibilidade.

Na versão hierárquica deste algoritmo (Hanrahan,1991), a cena é modelada inicialmente por um número *k* de grandes polígonos. As interações da luz são calculadas entre estes polígonos, que são hierarquicamente subdivididos se necessário, para melhorar a precisão.

Podemos encontrar paralelismo neste algoritmo em três níveis: ao longo dos polígonos de entrada; ao longo dos elementos resultantes das subdivisões dos polígonos; e ao longo das várias iterações para calcular o valor de um elemento. Na implementação paralela de Jaswinder (Jaswinder,1994) cada processador possui sua fila de tarefas por realizar, que inicialmente contém um conjunto de interações entre polígonos por calcular. Quando um elemento é dividido, novas tarefas, envolvendo o novo elemento, são colocadas na fila do processador que realizou a divisão. Cada processador consome as tarefas de sua fila até acabar, quando então pega tarefas de outros através da memória compartilhada que mantém as filas.

Um alternativa a este método é o refinamento progressivo (Cohen,1988). Nesta técnica, o elemento com maior nível de energia em cada iteração é escolhido como elemento de emissão, e a energia é transferida deste elemento para os outros no ambiente. Este processo se repete até que o máximo nível de energia não transmitida caia para valores pré-especificados.

Muitos dos métodos paralelos de radiosidade descritos na literatura tentam acelerar o refinamento progressivo através do cálculo das transferências de energia a partir de vários elementos de emissão em paralelo. Isto acontece em razão do tempo para completar uma iteração ser bastante variável, dependendo das relações geométricas entre os elementos, o que pode causar o desbalanceamento de carga com grande perda de performance. Para evitar o desbalanceamento, muitos sistemas usam uma estratégia baseada na demanda existente na computação em paralelo dos fatores de forma.

Uma alternativa é distribuir os dados dos elementos e os cálculos de radiosidade entre os vários processadores. Esta estratégia necessita de métodos eficientes de comunicação global para calcular os fatores de forma e completar a transferência de energia.

A estratégia de processar múltiplos elementos emissores em paralelo perturba a ordem de execução encontrada no algoritmo seqüencial, o que pode levar a uma convergência mais lenta, e ocasionalmente a uma perda dos benefícios do paralelismo.

8.2 Arquiteturas de Hardware

8.2.1 Pixel-Plane 5

A Pixel Plane é uma arquitetura de pesquisa, não comercial, que está focada em uma extensiva paralelização dos estágios de rasterização de modo que a cada pixel corresponde um processador de um bit. Esta arquitetura também apresenta paralelismo em tarefas de *rendering* de alto nível. Pixel Plane 5 (Fuchs,1989), foi um dos primeiros sistemas com *buffers* paralelos virtuais. Esta máquina utiliza de 10 a 16 *buffers* de rasterização com 128 *pixels* de largura. Cada *buffer* é capaz de rasterizar todas as primitivas que caírem na região de 128 por 128 *pixels*. Estes rasterizadores podem ser associados dinamicamente a qualquer uma das 80 regiões em uma tela de 1024 por 1024 *pixels*.

A rasterização nestas máquinas ocorre em duas fases. Primeiro, o módulo inicial (16 a 32 processadores de ponto flutuante) transforma as primitivas e as classifica em relação às regiões. Em seguida, os múltiplos rasterizadores processam a rasterização em paralelo. Quando um rasterizador termina o trabalho na sua região, ele transfere sua porção da imagem para um *frame buffer* e começa a processar uma nova região. Toda a comunicação entre os componentes do sistema é realizada através de uma rede *token ring* com 1.28 Gigabytes por segundo de largura de banda.

Esta máquina alcançou uma melhor utilização dos processadores que as suas verões anteriores, pois a região de 128 por 128 *pixels* é aproximadamente do tamanho da maioria das primitivas. Como os rasterizadores podem ser associados de maneira dinâmica a regiões, os recursos do sistema se concentram nas regiões que mais precisam.

Esta arquitetura apresenta duas dificuldades. Primeiro, a transferência das primitivas para múltiplos *buffers* de rasterização em paralelo requer uma rede de interconexão de alta performance entre o módulo inicial e o subsistema de rasterização, além de necessitar de controle e *software* de sincronização sofisticados. A segunda desvantagem, reside no fato de determinadas imagens com alta concentração de primitivas em uma determinada região ocasionarem um mau aproveitamento dos rasterizadores.

8.2.2 Power Iris 4D/240GTX

A máquina Power Iris 4D/240GTX da Silicon Graphics é uma estação de trabalho de alta performance, projetada para combinar processamento de uso geral com processamento gráfico 3D de alta performance para aplicações científicas e de engenharia.

Esta máquina possui uma CPU de uso geral de alta performance, composta de 4 multiprocessadores fortemente acoplados, que dividem um barramento único de memória. O subsistema gráfico pode realizar o *rendering* a uma velcidade superior a 100 mil quadriláteros por segundo, utilizando cor de 24 bits, método de iluminação de Gourad, e o algoritmo de *z-buffer* (Akkel,1989).

A arquitetura da Power Iris é composta de quatro subsistemas :

i. Subsistema da CPU
ii. Subsistema de Geometria
iii. Subsistema de Conversão

iv. Subsistema de Rasterização

v. Subsistema de Visualização

O Subsistema da CPU é que transforma, recorta e ilumina as primitivas. Ë composto de cinco processadores de ponto flutuante organizados em *pipeline*. Cada um dos processadores, denominado *geometry engine* (GE), contém uma fila de entrada, um controlador, e uma unidade de ponto flutuante com capacidade de 20 MFLOPS. São baseados em um processador comercial o Weitek 3332.

O primeiro GE transforma vértices e normais. O segundo realiza os cálculos de iluminação, suportando até oito pontos de luz. Os teste de aceitação/rejeição do *clipping* são realizados no terceiro. Ao quarto GE restam os cálculos de *clipping* de primitivas que interceptam os contornos da área de *clipping*. A quantização de cores é realizada no quinto GE, que além disto realiza as transformações de coordenadas para as coordenada da tela.

No Subsistema de Conversão as primitivas são rasterizadas em uma arquitetura em *pipeline*. O processador de polígonos classifica os vértices dos polígonos da esquerda para direita no espaço da tela. Estes vértices são então utilizados para decompor o polígono em trapezóides. O par mais alto e o par mais baixo de cada trapezóide são usados para calcular a inclinação que vai ser usada pelo processador de arestas.

O processador de arestas usa as informações dos vértices e das inclinações para calcular as coordenadas (*x, y, z)* e os valores de cor de cada *pixel* que cai na aresta do topo ou da base de cada trapezóide. Em uma coluna vertical, um par de *pixels* determina os pontos do topo e da base do vão (*span*). Este par de *pixels*, junto com a informação de inclinação são passados para o processador de vãos. Cada um dos cinco processadores de vãos atuando em paralelo é responsável por um quinto das colunas da tela. Eles calculam *z, R, G, B e* $\propto$ (para transparência e *antialiasing*) para cada um dos *pixels* no seu vão.

O cache de *pixels* concentra blocos de *pixels* durante as operações de cópia de modo a utilizar completamente a largura de banda do barramento de *pixels*.

No Subsistema de Rasterização é calculada a visibilidade e são escritos os atributos dos *pixels* no *frame buffer*. O subsistema de rasterização é composto por 20 máquinas de imagem, cada uma delas responsável por um vinte avos dos *pixels* na tela, arranjados em blocos de 4x5 *pixels* intercalados. 96 bits são associados a cada

pixel na tela: 2 *buffers* de 32 bits para imagem (R, G, B e $\propto$), um *z-buffer* de 24 bits, 4 planos de bits para *overlay/underlay*, e quadro *bitplanes* para janelas.

As 20 máquinas de imagem trabalham nos *pixels* em paralelo. Cada uma delas mistura os valores baseadas no valor de $\propto$ de cada *pixel*, permitindo a exibição de objetos transparentes e semitransparentes e o emprego de técnicas de *antialiasing* por superamostragem.

O Subsistema de Visualização exibe o conteúdo do *frame buffer* no monitor colorido. É composto de 5 processadores gráficos de multimodos (MGP), sendo a cada um associado 1/5 das colunas da tela. Os MGP lêem a imagem do *frame buffer* de forma concorrente, processam o modo apropriado de exibição (RGB ou pseudocor), e enviam os dados para um conversor digital/analógico.

8.2.3 Stelar GS2000

Sistemas como o da Power Iris utilizam *hardware* dedicado para realizar as computações com imagens mais rapidamente. Muitos destes cálculos, particularmente na fase geométrica, são similares ao tipo de cálculo necessário em computação científica. A máquina Stelar GS2000 tenta disponibilizar seus recursos de computação tanto para geração das imagens como para aceleração de outras tarefas que possam estar rodando na máquina.

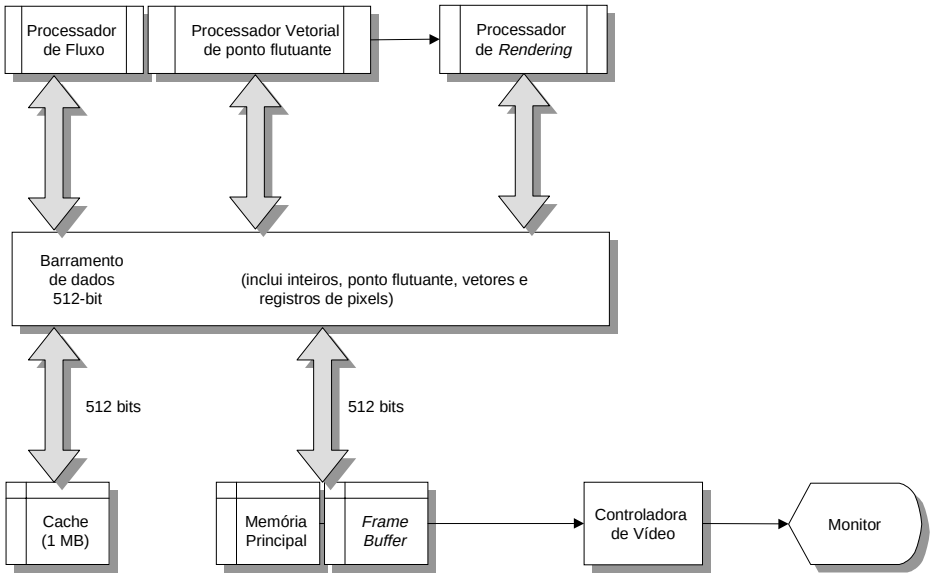


Figura 8.1 Diagrama de blocos da Stelar GS2000 .

A Figura 8.1 apresenta o diagrama de blocos da arquitetura Stelar GS2000. O processador de fluxo é um processador único de alta performance que executa

simultaneamente instruções a partir de quatro fluxos de instruções. Este processador pode realizar até 25 milhões de instruções por segundo. O processador vetorial de ponto flutuante realiza operações escalares e vetoriais de ponto flutuante, podendo realizar 40 milhões de operações de ponto flutuante por segundo. O processador de rendering utiliza uma unidade para realizar os cálculos do processo de rendering de polígonos, e um processador *footprint* 4*x*4 SIMD para as operações com *pixels*. Esta máquina possui a capacidade de processar 30 mil polígonos por segundo com modelo de iluminação de Phong.

Todos os recursos de memória e processamento da máquina ficam organizados em torno de uma estrutura central de comunicação com largura de 512 bits denominada Barramento de Dados (*DataPath*).

Em aplicações gráficas típicas, o processador de fluxos múltiplos percorre a base de dados das primitivas, enviando então dados para o processador vetorial, que realiza os cálculos geométricos. As primitivas transformadas são então enviadas para o processador de *rendering*, que calcula os *pixels* da imagem. Ao invés de armazenar a imagem diretamente no *buffer*, o processador de *rendering* as armazena na memória principal como mapas virtuais de *pixels* (MVP). Em seguida porções dos MVPs são copiadas para o *frame buffer* do dispositivo de visualização.

Em razão do armazenamento dos MVP e do acesso do processador de fluxos a eles, podem ser realizadas uma série de operações de pós-processamento. Por exemplo, para exibir texturas, as unidades de rasterização geram índices de texturas, ao invés da cor final. Depois, um dos processadores de uso geral passa a imagem gerada e substitui os índices de textura pelas cores apropriadas para cada pixel.

8.2.4 Pixel Flow

Esta máquina é desenvolvida na Universidade de Carolina do Norte em Chapel Hill (Molnar,1992). Ela parte do paradigma convencional de *rendering* e explora uma forma de paralelismo denominada composição de imagens. As primitivas geométricas são associadas a diferentes processadores. As imagens geradas por cada um dos processadores é composta sobre uma rede. A vantagem desta abordagem é a escalabilidade linear da performance com o aumento do número de processadores. No entanto, necessita de alta largura de banda na rede de composição, pois cada *pixel* trafega nesta rede várias vezes por quadro.

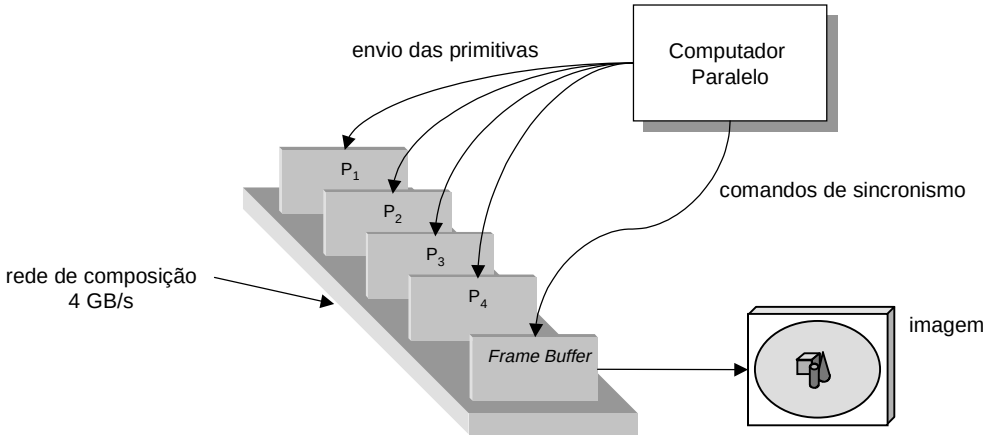


Figura 8.2Arquitetura da Máquina *Pixel Flow*

Existem três diferentes tipos de processadores nesta arquitetura. Os responsáveis pelo rendering são i860XP de uso geral, estas são as máquinas que processam a geometria. Existem os processadores que são responsáveis pelo *shading*, estes processadores são iguais as máquinas de geometria, havendo somente uma memória adicional de textura. O *frame buffer* contém memória de vídeo duplamente bufferizada.

8.2.5 Pixar e Pixar II

Estas máquinas foram projetadas em 1984 com o objetivo de gerar imagens de efeitos especiais e animações para filmes. Toda a computação, assim como o controle de fluxo de *pixels* é realizado por processadores de canal (*channel processors*-CHAP). Um CHAP é um processador vetorial paralelo operando como uma máquina SIMD, executando cada operação sobre 4 operandos simultaneamente. Drebin *et al.* (Drebin, 1988), implementou um método baseado em *shearing* (*forward* de multipasso) em uma máquina PIXAR. A abordagem de *scanlines* percorridas na ordem dos objetos é bastante apropriada para os processadores SIMD

9. Sistemas de Rendering de Volumes em Paralelo

Rendering de volumes é o nome dado ao conjunto de técnicas que permite a visualização de funções escalares tridimensionais amostradas. Um cenário típico de visualização de volumes em aplicações médicas se inicia a partir da digitalização, em seguida a seqüência de fatias transversais 2D é reconstruída em um modelo volumétrico 3D. Então, é usada uma representação baseada em um *grid* 3D de *voxels* ao qual se aplica uma técnica de *rendering* de volumes.

9.1.1 Algoritmos

Inicialmente o rendering de volumes foi desenvolvido em computadores paralelos como um meio de acelerar o grande número de cálculos realizado. Também havia a necessidade de aproveitamento da grande quantidade de memória e largura de banda disponível nestas máquinas para realizar o *rendering* de grandes volumes. Máquinas SIMD (CM-2, MasPar e Princenton Engine) foram inicialmente utilizadas devido à facilidade do modelo de programação desta máquinas. No início o balanceamento de carga foi ignorado, por que o volume de dados mantinha a maioria dos processadores suficientemente ocupados até o final.

9.1.2 Ray Casting

Ray Casting é um algoritmo considerado diretamente paralelizável. Embora seja facilmente implementado em máquinas SIMD, uma das técnicas mais comuns de aceleração, terminação adaptativa do raio, se torna difícil de simular neste ambiente.

O volume a ser visualizado, é representado por um cubo formado por *voxels*. Para cada voxel uma cor e uma opacidade foram calculadas em um passo de pré-processamento. Assim raios são lançados a partir da posição do observador em direção ao volume, passando pelos *pixels* da imagem que será gerada. Cores e opacidade serão calculados para um conjunto de pontos ao longo do caminho do raio dentro do volume, isto é feito através de um processo de interpolação dos valores dos *voxels*. Então este valores serão acumulados para gerar um cor para o raio, ou seja, definindo a cor do pixel. Esses raio são lançado em direção ao volume até que atinjam um determinado valor de opacidade, quando então paramos o caminhamento deste raio (término antecipado). Uma outra forma de acelerar este algoritmo é através da

codificação do volume em uma *octree* definida pela opacidade, de modo a evitar os cálculos de interpolação em regiões transparentes do volume.

Em um ambiente de memória compartilhada cada processador pode diretamente referenciar qualquer *voxel* dentro do volume. Neste ambiente o mais fácil é utilizar o paralelismo do lançamento de raios. Dados *p* processadores, o plano da imagem é particionado em *p* blocos retangulares. Cada bloco é subdividido em subáreas iguais e quadradas, que são colocadas na fila de tarefas de cada um dos processadores. Assim o processador lança os raios correspondentes aos seus blocos de imagem na ordem das *scanlines*. Quando termina pega tarefas de outros processadores ainda sobrecarregados. Este algoritmo foi descrito em (Jaswinder,1994)

Nieh (Nieh,1992) descreve uma implementação baseada na máquina DASH desenvolvida em Stanford. O algoritmo utilizado é o *ray casting* com as mesmas otimizações descritas para o algoritmo anterior. O volume de dados é então subdividido de maneira intercalada entre os *clusters* (memórias dos processadores). Inicialmente, cada processador é responsável por um conjunto de blocos contíguos da imagem. Cada processador lança os raios associados aos seus blocos. Quando terminado o trabalho, o processador verifica se algum dos vizinhos possui trabalho pendente, caso positivo toma para si alguns destes blocos e continua a lançar raios. Esta abordagem permite a utilização da maioria das otimizações usadas no *ray casting.* Isto é possível graças ao fato de cada processador computar todas as contribuições de um dado raio. Como a máquina onde foi implementado este algoritmo possui suas memórias distribuídas ao longo dos processadores, foi necessária a utilização de um esquema de cache para acessar os dados de forma eficiente. Esta implementação conseguiu aceleração quase linear quando o *ray casting* adaptativo não foi usado, e foi verificado que, embora a aceleração não tenha sido muito boa, a melhora no tempo de *rendering* foi bastante considerável. Esta observação leva a um questionamento da aceleração como boa medida de performance.

Uma abordagem diferente para o rendering de volumes via *ray casting* foi proposta por Mackerras (Mackerras, 1994). Seu algoritmo utilizava uma decomposição do espaço da imagem em um computador de memória distribuída (Fujitsu AP-100). Esta abordagem explorava coerência de dados no volume, assumindo que raios adjacentes na imagem penetrariam em *voxels* vizinhos no volume. Neste algoritmo blocos de tamanhos iguais eram associados a cada um dos

processadores. Um processador principal controlava o balanceamento de carga, e um mecanismo de *timeout* permitia que a carga de trabalho de processadores sobrecarregados fosse redistribuída.

Schröder e Stoll (Schröder,1992) introduziram um método que descreve o lançamento de raios como desenho de linhas em um espaço discreto 3D, reduzindo os padrões de comunicação a translações toroidais ao longo de eixos específicos

9.1.3 Splatting

Splatting é uma técnica bastante apropriada para geração de eficientes implementações paralelas. Westover (Westover,1990) o idealizador deste algoritmo propôs uma implementação paralela. Sua versão paralela divide o volume de dados em N subconjuntos. Para uma dada direção de visualização é definida a ordem de caminhamento no volume o que propicia a descoberta dos subconjuntos que dividem a mesma porção do *buffer* de acumulação. Estes subconjuntos são então agrupados e enviados para os *p* processadores. Em cada um destes processadores é executado um algoritmo de *splatting* seqüencial. Quando as imagens estão geradas são enviadas para um processador que as combina de modo a formar a imagem do volume.

Machiraju e Yagel (Machiraju,1993) apresentam um método que utiliza cálculos incrementais para a transformações de *voxels* em *pipeline*. Um *voxel* tem sua posição no espaço da imagem calculada. Em seguida *voxels* em uma faixa longitudinal são transformado de maneira incremental a partir do resultado do primeiro. Depois que esta fatia é toda processada, todos os *voxels* de uma fatia podem ser transformados usando somas vetorizadas baseadas no valor da faixa inicial, e sucessivamente outras fatias podem ser processadas. Esta abordagem funciona bem para volumes com a maior parte dos *voxels* preenchidos. Havendo problemas para sua aplicação a volumes esparsos.

Na abordagem proposta por Elvins (Elvins,1992) o processador principal distribui os dados iniciais para os processadores. Esta implementação é baseada na divisão for fatias do volume, são enviadas três fatias adjacentes do volume para cada processador (para possibilitar o cálculo do gradiente). Cada processador secundário calcula sua imagem intermediária e envia para o processador principal, onde é efetuada a composição.

Neumann (Neumann,1994) implementou uma versão deste algoritmo em uma máquina Pixel-Plane 5. A idéia principal é a divisão de cada uma das fatias do volume

por vez. Os planos são definidos na direção ortogonal à face do volume mais próxima do plano da imagem. Inicialmente cada processador é associado a um conjunto de planos, processando um plano completo por vez.

O processador então transforma, recorta, e gera o núcleo de reconstrução de cada *voxel* que é rasterizado pelos processadores de *rendering* onde é feita a composição dessas fatias.

9.1.4 Shearing

Lacoute (Lacroute,1994) propôs uma técnica baseada no cisalhamento do volume denominada *shear-warping*. Este algoritmo é descrito em três etapas: transformação do volume (cisalhamento); composição da imagem intermediária; *warping* para gerar imagem final.

Jaswinder (Jaswinder,1994) propõem uma paralelização da fase inicial através do particionamento da imagem intermediária entre os vários processadores, de modo que apenas um processador escreve em um determinado *pixel* desta imagem. Esta divisão é feita em grupos de *scanlines*, sendo os grupos associados aos processadores de forma intercalada. Também existe um mecanismo de divisão de trabalho para realizar um balanceamento de carga sob demanda. A fase de *warping* é paralelizada de acordo com uma subdivisão da imagem em grupos de *scanlines*.

9.1.5 Rendering de Terrenos

No *rendering* de terrenos, o problema é gerar uma representação plausível de um terreno real ou imaginário como se estivesse sendo visto de algum ponto sobre ou acima da superfície. Kaba *et all* (Kaba,1993) desenvolveu técnicas paralelas de rendering de terrenos para a Princenton Engine, um sistema SIMD. Seus métodos utilizam uma decomposição de tarefas baseadas nos objetos, distribuindo a imagem de entrada e os conjuntos de valores de elevação entre os processadores através da associação de colunas de *pixels* a processadores. Antes de realizar a projeção, os dados devem ser rotacionados e escalados para concordar com a direção e altitude da visada. A eliminação de superfícies escondidas é realizada rasterizando a transformação de trás para a frente, uma linha por vez. A cada linha da imagem processada, uma linha do horizonte é atualizada.

Li e Curkendal (Li,1992) desenvolveram técnicas para realizar o *rendering* de superfícies planetárias usando uma grande variedade de arquiteturas de memória

distribuída em larga escala. Utilizaram a máquina Intel MPP com memória distribuída e uma decomposição do espaço dos objetos como paradigma. Como no sistema de Kaba, os dados organizados como campos de alturas são projetados na tela. Nas primeiras implementações os dados de entrada eram particionados em fatias horizontais, as quais eram associadas aos processadores de modo intercalado. Nas implementações posteriores os dados são decompostos em regiões quadradas que são associadas randomicamente aos processadores.

Para eliminação de superfícies ocultas, Li e Curkendal usam o algoritmo de *z-buffer*. O *buffer* da imagem final é replicado em cada um dos processadores, com cada processador projetando os *pixels* da sua região no *buffer* local. O passo de montagem da imagem final é necessário para combinar as subimagens de cada processador, o que é feito usando uma técnica parecida com a de Ma *et all* (Ma,1994). Este sistema tem sido usado para produzir imagens de sobrevôo de Marte e Vênus a partir de dados gerados pelas sondas da NASA.

Wright e Hsieh (Wright,1992) apresentam um algoritmo de *rendering* de terrenos em *pipeline*, que foi implementado em *hardware*. Inicialmente este sistema realiza uma projeção para transformar os objetos para o espaço da imagem, sendo os objetos representados por *voxels*. A arquitetura consiste de dois *pipelines*, um para processamento de *voxels* e um para processamento de *pixels*. O *pipeline* de *voxels* percorre a base de dados, gerando colunas de *voxels* que são iluminadas, transformadas para as coordenadas de visualização, e rasterizadas em *pixels*. O *pipeline* de *pixels* projeta o *pixel* na tela, coloca os efeitos de névoa, transparência e cálculos de *z-buffer*. Várias técnicas são aplicadas em diferentes níveis do *pipeline* para reduzir *aliasing* temporal e espacial. O *hardware* desenvolvido é capaz de processar 10 quadros por segundo a uma resolução de 384*x*384 pontos, conseguindo uma aceleração de ordem maior que três sobre uma implementação seqüencial baseada em *software*.

9.1.5.1 Sistemas TerraVision e TerraVision II

TerraVision (Leclerc,1994) é um sistema para navegação interativa em representações 3D de grandes área geográficas. O sistema pode recuperar e combinar grandes volumes de dados oriundos de fontes remotas, incluindo fotos aéreas e de satélites, topografia, dados climáticos, prédios, e outras características culturais. Estes

dados podem ser da ordem de Terabytes, e estarem distribuídos ao longo de uma rede de grande alcance.

Para permitir a interação em tempo real com esta grande quantidade de dados localizados em repositórios distribuídos, foi empregada uma representação em azulejos (*tiles*) baseada em uma representação em multiresolução dos dados. O processamento se dá segundo as seguintes fases:

- testes de visibilidade: em cada quadro, o TerraVision calcula todos os retalhos potencialmente visíveis para o ponto de vista especificado usando um esquema de busca em uma árvore q*uad-tree*. Somente os retalhos do terreno dentro do *frustum* de visão atual são solicitados à rede e usados no *rendering* da cena.
- testes de nível de detalhe: os azulejos visíveis são selecionados baseados no tamanho projetado da tela; como resultado disto, azulejos em mais alta resolução são usados na frente (*foreground*) e os de menor resolução atrás (*background*). Os azulejos são solicitados da resolução mais grossa para a mais fina de modo que o sistema possui sempre dados em baixa resolução da área de interesse.
- predição e pré-seleção: o sistema tenta prever para onde o usuário se encaminhará no próximo quadro e seleciona antecipadamente os azulejos associados a esta região, de modo que eles estejam imediatamente disponíveis para *rendering*.
- *cache* de azulejos: azulejos recentemente recebidos são colocados em um *cache* local por que provavelmente serão solicitados em seguida.

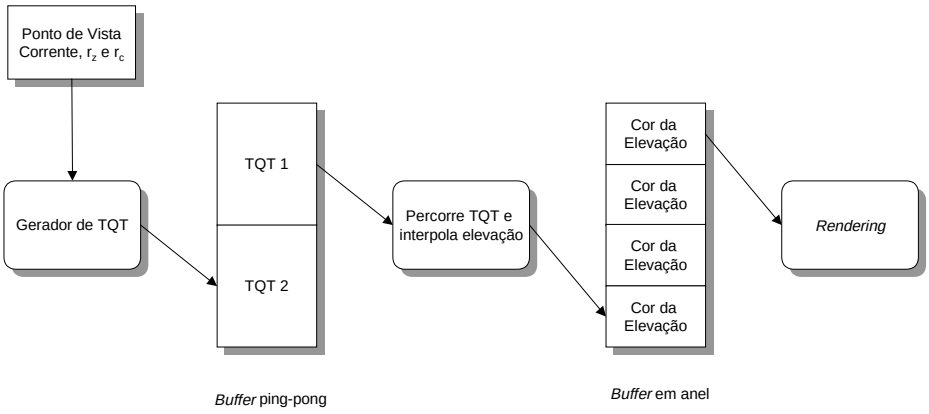


Figura 9.1 Pipeline de Rendering do TerraVision

Baseados nesta arquitetura o sistema TerraVision permite que usuários interajam com um conjunto de dados usando taxas de 10 a 30 quadros por segundos em uma SGI Octane 2.

O paralelismo utilizado no sistema é baseado no pipeline integrado de busca na base de dados e *rendering* da cena.

Dada uma estação de trabalho com uma arquitetura similar às máquinas Silicon Graphics, uma "unidade de *rendering*" eficiente é um azulejo da base de dados com uma cor simples e uma matriz de dados amostrados de elevação (mapa de alturas) associados. O retalho é então colocado na memória de textura e os dados de altura são colocados em uma *quad-strip* e enviados para o *pipeline* de rendering da máquina. O *pipeline* de rendering é dividido em três estágios, como mostrado na Figura 9.1. Cada estágio é implementado como um processo separado, comunicando-se via memória compartilhada.

O primeiro estágio percorre a *quad-tree* do terreno e cria uma estrutura *quad-tree* que contem somente os nós visíveis da estrutura original, somente os nós subdivididos possuem filhos. Esta estrutura é colocada em uma metade do *buffer* pig-pong. O próximo estágio percorre a outra metade do *buffer* ping-pong na memória compartilhada, processa os folhas de cor e elevação preparando para que o estágio final realize o *rendering*. O estágio final tenta constantemente ler novos dados no buffer em anel. E caso seja uma marca de novo quadro, limpa o *frame buffer* e o *back buffer*. Caso contrário pega os valores de elevação e cor do azulejo, e coloca a cor no mapa de textura e converte os dados de altura em uma *quad-strip*.

9.1.6 Arquiteturas

O desafio em implementar um sistema de *rendering* de volumes reside no armazenamento eficiente de uma grande quantidade de dados volumétricos. Assim um sistema que permita interação em tempo real deve empregar *hardware* de alta performance baseado em multiprocessamento e organização paralela da memória.

Nesta seção serão apresentadas cinco arquiteturas de hardware desenvolvidas para visualização de volumes.

9.1.6.1 PARCUM

O sistema PARCUM (***P****rocessing* ***AR****chitecture based on* ***CU****bic* ***M****emory*) tem sido desenvolvido na Universidade de Berlim por Jackel (Jackel,1985). A parte

central do sistema é uma memória 3D especialmente organizada denominada Memória Cubo. A Figura 9.2 apresenta uma visão geral deste sistema.

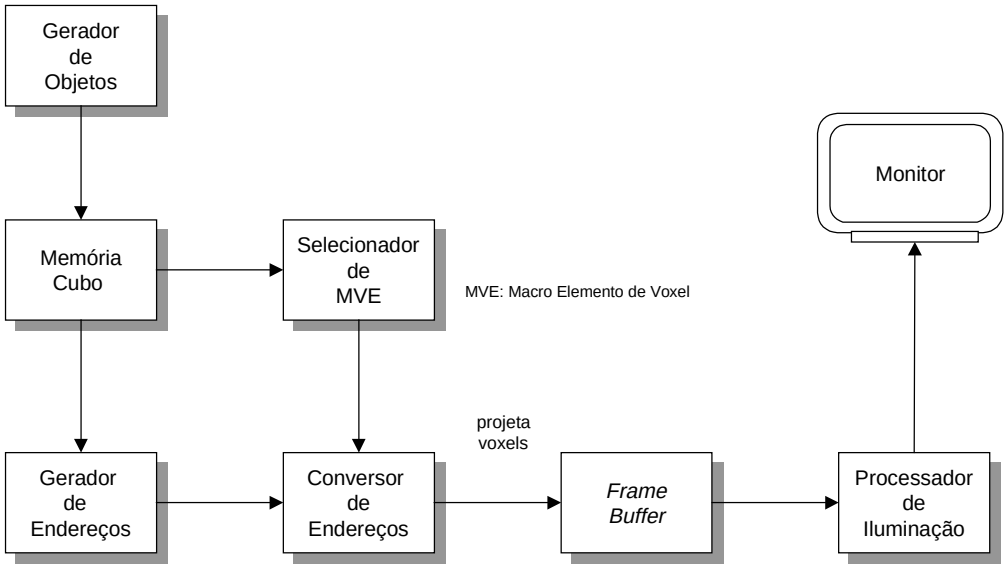


Figura 9.2 Arquitetura do Sistema PARCUM

A memória cubo permite ler e escrever simultaneamente os MVE que são blocos com 4^3 *voxels*. A memória é fisicamente dividida em 64 módulos de memória com capacidade de 2Mbytes cada um. Cada *voxel* de um MVE é associado a um módulo de memória diferente em uma organização intercalada, de modo a permitir acesso a todos os 64 *voxels* de um MVE em paralelo.

O Gerador de Objetos gera objetos de volume na memória cubo. Este volume de dados é projetado usando *ray casting* em paralelo. Uma trajetória de referência localizada no plano dos raios paralelos é calculada pelo gerador de endereços, e ao longo desta trajetória serão acessados os MVE. A seqüência de MVEs é processada no Selecionador, que seleciona os *voxels* visíveis dentro de cada MVE. No gerador de endereços estes *voxels* são projetados em um plano de projeção especificado pelo usuário. Cada *voxel* é transformado pela matriz de projeção e então projetado em um *z-buffer*. A profundidade de cada *voxel* é usada para iluminação baseada na imagem dentro do processador de iluminação.

9.1.6.2 Processador de *Voxels*

A arquitetura do Processador de *Voxels* tem sido projetada por Goldwasser e Reynolds na Universidade da Pensylvania (Goldwasser,1989).

Esta arquitetura introduzida por Goldwasser e Reynolds ((Goldwasser,1983) e (Goldwasser,1984)) é mostrada conceitualmente na Figura 9.3. Ela usa uma subdivisão recursiva do espaço do objeto (*octree*) e uma árvore hierárquica dos processadores de *rendering* e de imagens. Cada subcubo do volume é associado a um processador de *rendering* no primeiro nível de processadores da árvore. Estes subcubos são processados na ordem *back-to-front*. Os processadores de imagens do nível seguinte realizam a composição das imagens geradas por estes subcubos nesta mesma ordem, para gerar a imagem final.

Os dados do volume são armazenados em um computador hospedeiro (principal) que gerencia o acesso aos dados.

O Processador de *Voxels* consegue alcançar a velocidade de 25 quadros por segundo, sendo esta performance atribuída ao paralelismo empregado em três níveis : subdivisão do espaço de *voxels*; aritmética em *pipeline* em cada processador; e paralelismo entre processadores.

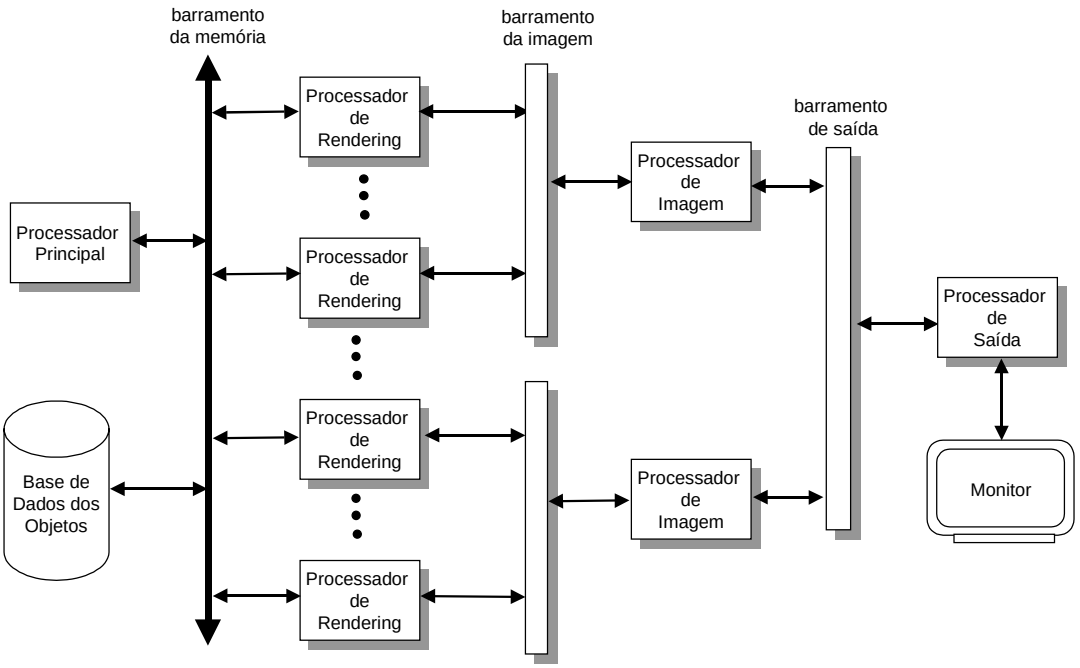


Figura 9.3 Arquitetura do *Voxel Processor*

9.1.6.3 VOGUE

VOGUE (Knittel,1994) é uma unidade compacta e escalável, baseada em *ray casting*, que realiza o *rendering* de volumes em taxas interativas e com um custo de

hardware razoável. A unidade básica é constituída de uma memória intercalada e quatro *chips* VLSI como mostrado na Figura 9.4. Esta máquina possibilita projeções perspectivas arbitrárias, iluminação de Phong, uma fonte de luz livremente posicionável, e uma classificação não binária com a utilização de funções de transferência para cor e opacidade.

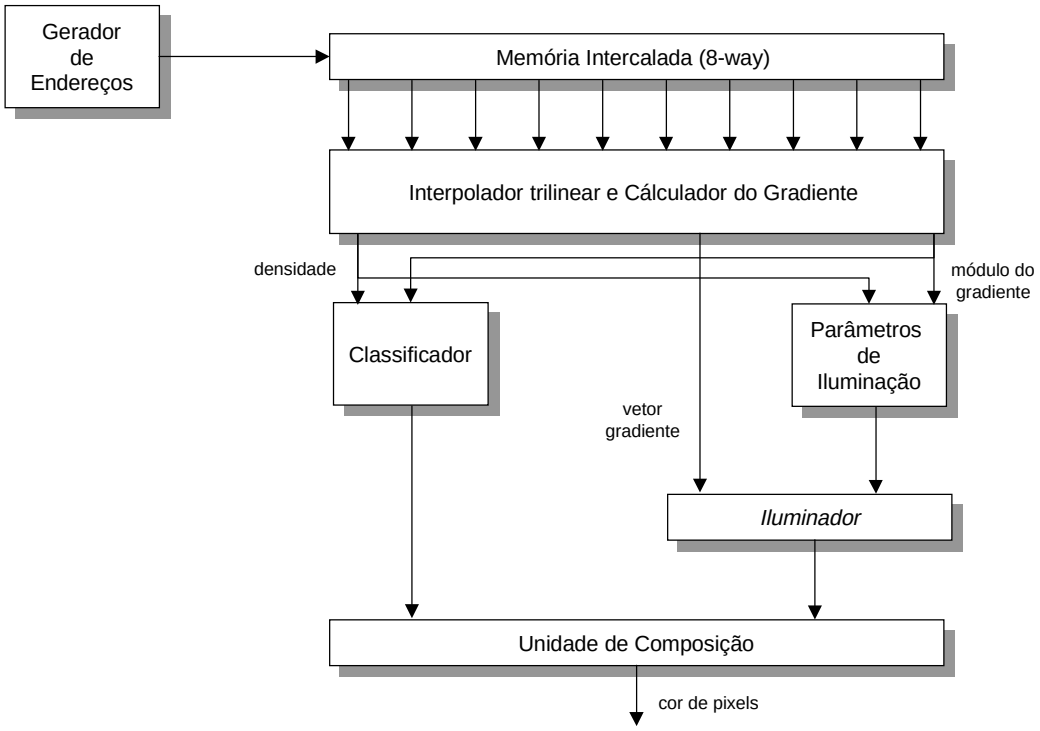


Figura 9.4 Arquitetura VOGUE

Após obter todos os parâmetros do raio, o gerador de endereços sequencialmente gera os pontos de todos os raios, e cada um dos conjuntos de 8 endereços (no máximo sete), é passado para o sistema de memória. A memória de volume (capacidade máxima de 256 Mbytes) é constituída de oito bancos independentes de memória e fornece oito *voxels* por acesso. O interpolador e estimador do gradiente realiza a reconstrução trilinear e calcula a magnitude e direção do gradiente. Estes valores remetem às tabelas LUT de cor e opacidade, para gerar os parâmetros de iluminação. O algoritmo de iluminação de Phong é implementado pelo iluminador para uma fonte de luz. A unidade de composição soma as intensidades ao longo do raio e remete o valor do pixel para o *frame buffer*.

A performance de visualização em tempo real pode ser obtida através de múltiplas unidades paralelas. Assim, o volume de dados é subdividido em subvolumes distribuídos para as diversas unidades. Cada unidade processa um raio ao longo do

seu subvolume e ao final repassa as propriedades definindo o raio para a unidade vizinha. A performance estimada para este sistema é de 2.5 quadros/s para um volume com dimensão de 256^3 .

9.1.6.4 Arquitetura CUBE

Esta arquitetura foi desenvolvida por Kaufman e sua equipe na Universidade de Nova York - Stony Brook (Kaufman,1988). Ela utiliza processamento paralelo maciço em larga escala para suportar interação de volumes 3D de alta resolução (e.g., 1024^3 a 30 quadros por segundo). O paralelismo é obtido através do processamento simultâneo de um feixe de *voxels*.

O paralelismo na fase de projeção é feito através de um barramento especial denominado *Voxel Multiple Write Bus* (VMWB). Um feixe completo de *voxels* é colocado no barramento, com cada *voxel* em um processador distinto, para a escolha do *voxel* opaco que está mais próximo do observador.

Existem três processadores que acessam a memória com os dados do volume (*Cubic Frame Buffer*, CFB), eles são: Processador de Geometria (PG), Processador de *Frame Buffer* 3D (PFB), e o Processador de Visualização 3D (PV).

O PV gera a projeção 2D iluminada das imagens no CFB. Para isto lança um feixe de raios no CFB e através da utilização do VMWB e da organização do CFB determina a projeção ao longo do feixe. Este processador também ilumina a imagem projetada empregando a técnica do gradiente (Cohen,1990).

As subimagens do CFB são manipuladas pelo PFB, podemos ver este processador como uma máquina de *voxblt*, realizando atividades como: manipulação de caixas 3D (transformação, cópia, carga, operações sobre os *voxels*, etc.), entrada de dados de *scanners* 3D, e comunicação entre os dispositivos 3D de entrada e os cursores 3D.

O PG realiza a rasterização dos objetos geométricos na representação baseada em *voxels*.

Aplicações específicas do projeto Cube incluem imagens médicas ou biológicas, CAD, simulação 3D e visualização científica. Este sistema provê suporte para o sistema MediCube 3D, um sistema versátil para manipulação de imagens médicas.

9.1.6.5 Arquitetura 3DP^4

A arquitetura 3DP^4 (*Three Dimensional Perspective Peary* [*Picture Element Array*]), baseada em *voxels* foi proposta na Universidade de Keio (Ohashi,1985). Esta arquitetura é um sistema de animação interativa para modelagem de sólidos e *rendering* de volumes em aplicações biomédicas. A memória é composta por um conjunto de *Pearys*, cada um com 64^3 *voxels* representando uma parte da cena. A arquitetura é em *pipeline* com três estágios. O primeiro emprega um conjunto de processadores em paralelo, um para cada *Peary*, para realizar a projeção. O próximo estágio do *pipeline* é um processador para combinar as imagens projetadas por cada processador da fase anterior. E em um pós processamento realiza-se a iluminação.

10. Sistemas Paralelos de *Rendering*: Considerações.

Verifica-se que a área de algoritmos paralelos de rendering é repleta de objetivos contraditórios. A maneira de resolver os problemas surgidos na tentativa de conciliação destes objetivos depende de uma série de fatores, tais como requisitos da aplicação e características da arquitetura alvo. Nesta seção apresenta-se alguns destes objetivos não conciliáveis e como podem ser tratados.

10.1 Sistemas de Hardware versus Sistemas de Software.

Sistemas de *hardware* empregam circuitos integrados especializados para acelerar o processo de *rendering*. Nos casos mais simples, o *hardware* gráfico consiste de um único processador acoplado ao sistema de memória de vídeo. Em outros casos, circuitos integrados especiais implementam diretamente etapas do processo de *rendering*.

A utilização de *hardware* dedicado tem obtido muito sucesso, embora a alta performance e efetividade do custo do sistema dedicado seja devida à especialização do *hardware*, o que limita a flexibilidade. Verifica-se que modelos de iluminação especializados, tratamento de imagens de alta-resolução, e métodos sofisticados de *rendering*, como *ray tracing* e *radiosidade*, podem ser implementados tranqüilamente em *software* com uma perda de performance correspondente.

Para melhorar a performance dos sistemas de *rendering* implementados em *software* é necessário implementá-los em plataformas paralelas de uso geral, tais como supercomputadores escaláveis ou redes de estações de trabalho. Nestes sistemas os processadores não são otimizados especificamente para operações gráficas e as redes de comunicação possuem limitações de banda. Além disto há a existência de sobrecargas de trabalho dos sistemas de *software* que não estão presentes nos sistemas de *hardware*. O desafio então é desenvolver algoritmos que possam enfrentar com sucesso estas sobrecargas de modo a aproveitar o potencial do *hardware*. Alguns sistemas de *rendering* de polígonos desenvolvidos para o sistema Intel Touchstone Delta (Ellsworth,1994), Thinking Machines CM_200 e CM-5 (Ortega,1993), e Cray T3D (Cray,1994) tem alcançado níveis de performance comparáveis aos da estações gráficas de topo da mesma época como a Silicon Graphic Reality Engine (Akeley,1993).

Também em razão da disponibilidade de grandes massas de dados resultantes de aplicações em sistemas maciçamente paralelos, torna-se necessário a implementação de sistemas de *rendering* nestas arquiteturas.

Redes de estações de trabalho ou de computadores pessoais fornecem um outro tipo de plataforma que pode ser usada por sistemas paralelos de *rendering* baseados em *software*. Estes sistemas não são muito caros, e sua capacidade de processamento e memória tem crescido rapidamente. No entanto, tendem a ser conectados por redes com baixa largura de banda, possuindo grandes problema de espera na comunicação. Por estas razões, estes sistemas são melhor usados com grande granularidade nos cálculos. Exemplos de sistemas baseados em rede incluem os sistemas de rendering de volumes de Ma *et all* (Ma,1994) e Giertsen e Petersen (Giertsen,1993), os sistemas baseado em métodos de radiosidade apresentados por Puech *et all* (Puech,1990) e Recker *et all* (Recker,1990), e o sistema fotorealista NetRenderMan da Pixar (Pixar,1994).

Sistemas de rendering baseados em *hardware* possuem uma vantagem preço/performance sobre os sistemas baseados em *software* que rodam em supercomputadores massiçamente paralelos. Para níveis similares de performance, os sistemas massiçamente paralelos custam de 10 a 100 vezes mais que estações gráficas especializadas.

10.2 Considerações de Arquitetura.

A arquitetura do sistema alvo, incluindo organização de memória e paradigmas de programação, tem grande impacto sobre o projeto de sistemas baseados em *software* para *rendering* em paralelo. Analisa-se a seguir a influência de algumas características da arquitetura alvo sobre o projeto dos sistemas de *rendering*.

10.2.1 Processamento vetorial.

Vetorização é uma forma simples de *pipelining*, que pode ser vista algoritmicamente como uma operação em paralelo sobre elementos de um vetor. Embora a vetorizacão tenha sido utilizada inicialmente em sistemas de computação de alta performance para acelerar operações de ponto flutuante em aplicações numéricas, atualmente também tem sido aplicada a sistemas gráficos, tanto a nível de arquitetura quanto de algoritmo. Sistemas desenvolvidos por Ardent (Diede,1988) e Stellar (Apgar,1988) no final dos anos 80 acoplaram sistemas gráficos de *display* a

processadores com unidade vetoriais de ponto flutuante. As unidades vetoriais eram utilizadas para cálculos no nível dos objetos sobre primitivas geométricas e para cálculos em geral, enquanto a rasterização era realizada usando *hardware* especializado.

10.2.2 Memória distribuída ou Memória compartilhada.

Sistemas de memória compartilhada possuem eficiente acesso a um espaço global de memória. Estes sistemas simplesmente reduzem a necessidade de pré-particionamento das estruturas principais, simplificam a coordenação dos processadores e maximizam o intervalo de algoritmos práticos. A maior desvantagem destes sistemas é sua escalabilidade limitada, que resulta em grandes tempos de espera no acesso à memória à medida que o número de processadores aumenta. Para minimizar estes problemas, bons algoritmos de memória compartilhada devem decompor o problema em subtarefas que evitem muitos acessos simultâneos à memória e mantenham o mínimo de operações de sincronização.

Sistemas de memória distribuída oferecem boas características de escalabilidade, mas geralmente com altos custos para referências remotas. Para esta classe de máquinas, o gerenciamento da comunicação é a consideração básica, especialmente para sistemas de *rendering* que geram grandes volumes de dados intermediários que devem ser mapeados dinamicamente do espaço do objeto para o espaço da imagem.

10.2.3 SIMD ou MIND

Em 1966, Flynn (Flynn,1966) propôs uma classificação para arquiteturas de computadores baseada no número de instruções e de fluxos de entrada de dados nos sistema. Arquiteturas paralelas de uso geral recaem em uma das duas arquiteturas, SIMD (*single instruction multiple data*), ou MIMD (*multiple instruction multiple data*).

Em uma arquitetura puramente SIMD, cada processador executa a mesma instrução em um determinado ciclo de processamento. Sistemas nesta classe tipicamente são compostos de um grande número de processadores simples com suporte a nível de micro-instruções para ativar ou desativar processadores ao longo da rede de interconexão. Podemos citar como exemplos deste tipo de arquitetura as máquinas CM-2 e CM-200 da Thinking Machines e, MP-1 e MP-2 da MasPar.

Na arquitetura MIMD cada processador executa seu conjunto de instruções independentemente dos outros. Processadores são livres para percorrer diferentes caminhos durante a execução do seu programa, ou então executarem programas completamente distintos. As operações de sincronização devem ser realizadas explicitamente sobre controle do *software*. Como representantes desta arquitetura podemos apresentar a máquina Intel Paragon, NCUBE3, CM-5 da Thinking Machines, SP2 da IBM, e T3D da Cray Research.

Apesar da aparente inadaptabilidade entre a variabilidade do processo de *rendering* e a sincronização rígida das máquinas SIMD, observa-se que um bom número de sistemas paralelos de *rendering* demostraram boa performance quando implementados em máquinas desta arquitetura. Existe uma série de razões para isto. Primeiro, devido à grande flexibilidade das arquiteturas MIMD, houve uma grande carga sobre as aplicações e sistemas operacionais, que deveriam estar aptos a lidar com a chegada de dados de fontes remotas em intervalos não prédeterminados e em ordem arbitrária. Isto geralmente resulta em protocolos complicados de comunicação e de *buffering*. Por outro lado, a forma de operação das máquinas SIMD virtualmente elimina esta sobrecarga computacional, resultando em custos baixos de comunicação que estão mais próximos das velocidades dos *hardware* atuais.

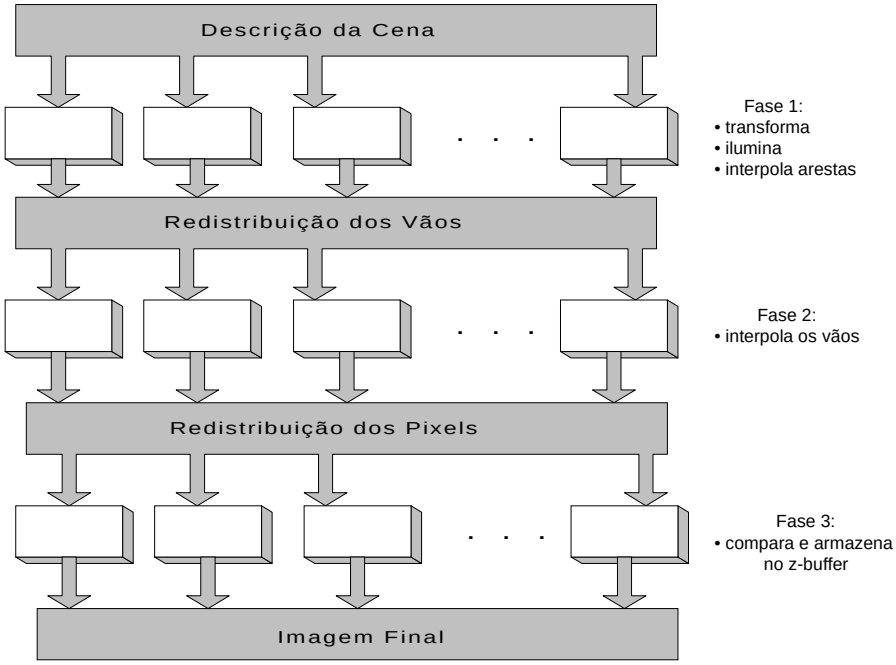


Figura 10.1 *Pipeline* de *rendering* em três fases

Além disto, tem-se a favor das máquinas SIMD a possibilidade de estruturar os algoritmos como um conjunto de fases distintas, onde cada fase opera em um conjunto uniforme de dados. O sistema desenvolvido por Ortega, Hansen e Ahrens (Ortega,1993) ilustra este princípio. Ao invés do passo simples de mapeamento usado pela maioria dos algoritmos de *rendering* de polígonos, nesta abordagem o *pipeline* de *rendering* é quebrado em três fases. A Figura 10.1 mostra um esquema simplificado desta divisão. Esta abordagem em várias fases fornece operações uniformes durante cada fase, e um esquema eficiente de comunicação pode reduzir o impacto da redistribuição extra.

10.2.4 Problemas de Comunicação.

Em sistemas de *rendering* que exploram paralelismo da imagem e do objeto, observa-se um alto volume de comunicação entre processadores. O gerenciamento desta comunicação é um aspecto central no projeto do sistema, e a escolha do algoritmo de comunicação tem um impacto significativo sobre o tempo, volume e padrão da comunicação. Existem três fatores principais que devem ser considerados: espera, largura de banda e contenção. Espera é o tempo necessário para realizar uma comunicação, não interessando a quantidade de dados transmitida. Largura de banda é simplesmente a quantidade de dados que pode ser comunicado por um canal em uma unidade de tempo. Se o sistema de *rendering* tentar jogar mais dados na rede de interconexão que a capacidade da mesma, serão gerados atrasos que diminuirão a performance. Contenção ocorre quando múltiplos processadores tentam enviar dados pelo mesmo segmento da rede simultaneamente e não há largura de banda necessária para suportar esta demanda do conjunto de processadores.

O tempo para um processador enviar dados para outro pode ser expresso pela seguinte fórmula :

$$t_{com} = t_{espera} + t_{transfer} + t_{atraso}$$

, onde o tempo total de comunicação, t_{com} , é a soma do tempo de espera t_{espera}, com o tempo de transferência $t_{transfer}$, e o tempo de atraso devido a contenção t_{atraso}. O tempo de transferência é simplesmente o volume de dados dividido pela largura de banda do canal.

O tempo de espera pode ser melhor compreendido como a soma de três componentes :

$$t_{espera} = t_{envio} + t_{rot} + t_{receb}$$

, onde t_{envio} é o tempo para iniciar a transferencia, t_{rot} é a espera através da rede, e t_{receb} é o tempo para receber o dado na outra ponta.

Uma série de técnicas algorítmicas foram desenvolvidas para lidar com a sobrecarga de trabalho resultante da comunicação em sistemas de *rendering* em paralelo. Uma maneira simples de reduzir a espera é acumular mensagens pequenas em grandes *buffers* antes de enviar, amortizando assim o custo sobre vários itens de dados. No entanto esta técnica não é escalável no caso de mapeamentos do espaço do objeto para o espaço da imagem onde o padrão de comunicação é de muitos para muitos. Isto implica que o número de mensagens por cada processador é $O(p)$, onde p é o número de processadores no sistema. Assumindo que temos uma cena com uma resolução de imagem fixa, e um particionamento dos dados da imagem e dos objetos para p processadores, verifica-se que o número de itens por processador é da ordem de $1/p$. Assim o número de itens de dados por mensagem decresce proporcional a $1/p^2$. Logo a sobrecarga de espera cresce linearmente com o número de processadores, tornando ineficiente a amortização de forma crescente.

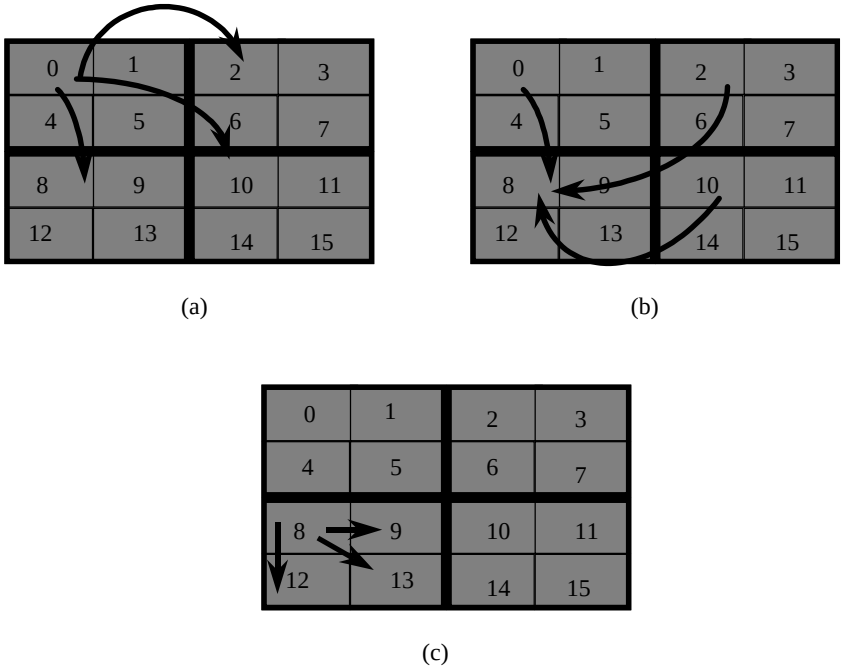


Figura 10.2 Redistribuição de Dados em 2 passos

Uma solução é reduzir a complexidade algorítmica da comunicação usando um esquema de entrega em passos múltiplos, como proposto por Ellsworth (Ellsworth,1994). Com este método, os processadores são divididos em aproximadamente $\sqrt{p}$ grupos, cada um contendo $\sqrt{p}$ processadores de maneira

irregular. Itens de dados direcionados para qualquer dos processadores em um grupo remoto são acumulados em um *buffer* e transmitidos juntos como uma grande mensagem simples para um processador dentro do grupo destino. Este processador copia os itens de dados chegando em um segundo conjunto de buffers baseado no seu destino final, mesclando-os com dados vindos de outras fontes (outros grupos). Após a classificação, estes *buffers* são enviados para seus destinos finais dentro do grupo. A Figura 10.2 ilustra este processo. Nesta figura dados originados nos processadores 0, 2 e 10 são enviados para os processadores 9, 12 e 13, passando primeiro pelo processador 8.

O efeito deste algoritmo na rede é que o número de mensagens gerada por cada processador é reduzida para $O(\sqrt{p})$ e o comprimento de mensagens diminuem de forma mais suave, proporcional a $1/\sqrt{p^3}$, fazendo com que o tempo de espera seja amortizado de forma mais efetiva.

Crocket e Orloff (Crocket,1994) descobriram com seus experimentos em um IPSC/860 da Intel, que *buffers* para acumular grandes mensagens contribuem para reduzir o tempo de espera, mas aumentam os atrasos devidos à contenção. O problema ocorre quando um grande volume de dados é enviado para a rede em um curto período de tempo. Se o tráfego falha em se livrar rapidamente destes dados, processadores deverão esperar pela chegada deste dado e a performance do sistema cai. Este problema se torna mais evidente quando a carga está mais ou menos balanceada entre os processadores e eles tentam se comunicar em torno do mesmo instante.

10.2.5 Memória

Consumo de memória é um outro aspecto bastante relevante no projeto de sistemas de *rendering* paralelos. A aplicação de *rendering* exige o consumo de grande quantidade de memória, especialmente quando lida com cenas complexas e imagens de alta resolução.

A estrutura de um sistema paralelo de *rendering* pode ter grande influência sobre a memória necessária, provendo escalabilidade de dados ou piorando o problema através da replicação dos dados. Um exemplo de boa escalabilidade de dados são os sistemas *Sort-Middle*, pois as estruturas de dados da imagem e dos objetos podem ser particionadas uniformemente por todos os processadores. Por outro lado, alguns sistemas *Sort-Last* necessitam da replicação da imagem inteira em cada

processador, aumentando o custo na proporção direta do número de processadores no sistema.

Para realizar uma utilização satisfatória da memória é necessário encontrar pontos de equilíbrio entre os requisitos da aplicação, objetivos de performance e custo do sistema. Por exemplo, replicando os dados dos objetos podemos eliminar as sobrecargas devidas à comunicação, o que pode ser uma estratégia razoável para cenas moderadamente complexas em um ambiente com tempo alto de espera, tais como redes de estações de trabalho.

10.2.6 Composição e Exibição da Imagem

Sistemas de *rendering* de alta performance produzem uma grande quantidade de dados de saída na forma de imagem. Para um sistema de imagem com resolução de 1280*x*1024, com 24 bits/*pixel*, a 30 quadros por segundo, é necessária uma largura de banda de 120 MB/s no canal de comunicação do *display*. Como todos os sistemas de *rendering* paralelos replicam ou subdividem a imagem, o desafio então é combinar os valores dos *pixels* vindos de múltiplas fontes em altas taxas de quadros por segundo. Esta etapa pode criar um gargalo no sistema de *rendering* no passo de exibição da imagem, limitando a quantidade de paralelismo que pode ser efetivamente utilizada.

10.2.6.1 Soluções de *Hardware*

Este problema é melhor resolvido no nível do *hardware*. Uma abordagem para a solução deste problema seria a integração do *frame buffer* do *display* diretamente com o processo de geração dos *pixels*. Alternativamente, pode-se manter o sistema de rasterização separado do *frame buffer*, com os *pixels* sendo comunicados entre os dois sistemas através de um canal de comunicação de alta velocidade. Um exemplo desta abordagem é o sistema PixelPlane5 (Fuchs,1981) que usa uma rede *token ring* com largura de banda de 640 MB/s para interconectar os componentes do sistema. O sistema PixelFlow (Molnar,1992) usa uma rede de composição da imagem em *pipeline* com uma rede de comunicação entre os estágios com largura de banda de 4 Gb/s.

10.2.6.2 Soluções de *Software*

A maior parte dos sistemas de *rendering* paralelo particiona a imagem pelos processadores, ou replica os dados em todos os processadores.

Na primeira abordagem, as partições da imagem devem ser montadas para produzir a imagem completa. Isto pode ser feito internamente na máquina paralela ou no sistema de *display*. A composição interna implica na alocação de um espaço de memória para armazenar a imagem completa em algum lugar no sistema, o que pode resultar em uma perda de escalabilidade de dados. A composição externa pode ser realizada em muitos lugares em uma máquina central ou no sistema de *display*, ou até em um sistema de armazenamento externo.

Para sistemas que replicam a imagem em cada processador, usualmente é necessário realizar testes de visibilidade de *pixels* provenientes das várias contribuições.

Muitas estratégias algorítmicas estão disponíveis para composição e montagem da imagem. A abordagem mais simples é possuir um processador associado a esta tarefa que aceita as contribuições de todos os outros processadores, realizando as operações de composição necessárias. Para pequeno número de processadores esta abordagem é aceitável, mas não apresenta resultados de escalabilidade razoáveis, por que o processador de composição é um gargalo serial.

Uma melhoria nesta estratégia é realizar a composição da imagem em uma organização em árvore, combinando contribuições em cada nível da árvore. Isto resulta em uma melhor utilização dos processadores e da rede de comunicação, e roda em tempo *O(p)*. Ma *et all* (Ma,1994) observou que mesmo com esta abordagem, os processadores são subtilizados, e os processadores localizados nos níveis mais altos da árvores possuem uma sobrecarga de trabalho. Eles apresentam um esquema alternativo para composição da imagem, que roda em tempo logarítmico, e mantém a maioria dos processadores ocupados em cada estágio. A idéia chave neste método é dividir a imagem em cada passo, com pares de processadores operando em subimagens diferentes. No final do processo, a imagem é particionada entre todos os processadores, necessitando de um passo final de composição da imagem para recuperar todas as partes. A Figura 10.3 ilustra este procedimento com quatro processadores.

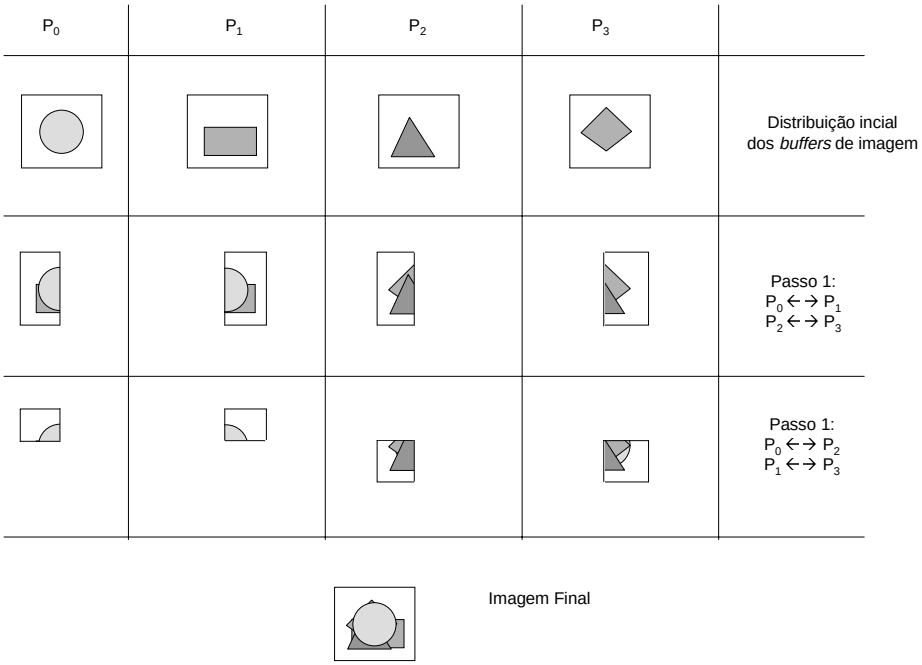


Figura 10.3 Composição de Imagem por Binary Swap

11. Referências

AKELEY, H. “Reality Engine Graphics Graphics”. Proceedings SIGGRAPH’93, pp. 109-116, **1993**.

APGAR, B., BERSACK, B. and MAMMEM, A. “A Display System for the Stellar Graphics Supercomputer Model GS1000”. Computer Graphics, Vol. 22, No. 4, pp. 255-262, **1988**.

APPEL, A. “Some Techniques for Shading Machine Rendering of Solids”. SJCC, pp. 37-45, **1968**.

BADOUEL, D. BOUATOUCH, K. and PRIOL, T. “Distributing Data and Control for Ray Tracing in Parallel”. IEEE Computer Graphics and Applications, Vol. 14, No. 4, pp. 69-77, **1994**.

BUNKER, M. and ECONOMY, R. “Evolution of GE CIG Systems”, SCSD Document, General Eletric Company, Daytona Beach, FL. **1989**.

CAMAHORT, E., and CHAKRAVARTY, I. “Integrating Volume Data Analysis and Rendering on Distributed Memory Architecture”. Proceedings 1993 Parallel Rendering Symposium, ACM Press, pp. 89-96, **1993**.

CASPARY, E. and SCHERSON, I. D.”A Self-Balanced Parallel Ray-Tracing Algorithm”In Parallel Processing for Vision and Display, Addison-Wesley, pp. 408-419, **1989**.

CATMULL, E. “A Subdivision Algorithm for Computer Display of Curved Surfaces”. Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.

CHALLINGER, J. “Parallel Volume Rendering on a Shared-Memory Multiprocessor”. Technical Report UCSC-CRL-91-23, University of California at Santa Cruz, **1991**.

CHALLINGER, J. “Scalable Parallel Volume Raycasting for Nonrectilinear Computational Grids”. Proceedings 1993 Parallel Rendering Symposium, ACM Press, pp. 81-88, **1993**.

CLARK, J. and HANNAH, M. “Distributed Processing in a High-Performance Smart

Image Memory". Lambda (VLSI Design), Vol.1, No. 3, pp. 40-45, **1980**.

CLARK, J. "A VLSI Geometry Processor for Graphics". Computer, Vol. 13, No. 7, pp.59-68, July, **1980**.

CLARK, J. "The Geometry Engine: A VLSI Geometry System for Graphics". Computer Graphics, Vol. 16, No. 3, pp.127-133, July, **1982**.

CLEARY, J., WYVILL, B., BIRTWISTLE, G., and VATTI, R. "Design and Analysis of a Parallel Ray Tracing Computer". Proceedings of Graphics Interface'83, pp. 33-34, **1983**.

COHEN, M. F., CHEN, S. E., WALLACE, J. R., and GREENBERG, D. P. "A Progressive Refinement Approach to Fast Radiosity Image Generation". Computer Graphics, Vol. 22, No. 4, pp. 75-84, **1988**.

COHEN, D., KAUFMAN, A., BAKALASH, R., and BERGMAN, S. "Real Time Discrete Shading". The Visual Computer, Vol. 8, No. 1, pp. 16-27, **1990**.

CRAY Research Inc. "Cray Animation Theather". **1994**.

CROCKET, T. W. and ORLOFF, T. "A Parallel rendering Algorithm for MIMD Architectures", Proceedings Parallel rendering Symposium, ACM Press, New York, pp. 35-42, **1993**.

CROCKET, T. W. and ORLOFF, T. "Parallel Polygon Rendering for Message-Passing Architectures". IEEE Parallel and Distributed Technology. Vol. 2, No. 2, pp. 17-28, **1994**.

CROW, F. "Parallelism in Rendering Algortihms". Proceedings Graphics Inetrface 88, Morgan Kaufman, San Mateo, California, pp. 87-96, **1988**.

DEERING, M. and NELSON, S. R., "Leo: A System fo Cost Effective 3D Shaded Graphics, Proceedings of SIGGRAPH'93, pp. 101-108, **1993**.

DELANY, H. C. "Ray Tracing on a Connection Machine". in Proceedings of the 1988 International Conference on Supercomputing, St. Malo, France, pp. 659-664, **1988**.

DIEDE, T., HAGENMAIER, C., MIRANKER, G., RUBISTEIN, J., and WORLEY, W. "The Titan Graphics Supercomputer Architecture". Computer, Vol. 21, No. 9, pp. 13-30, **1988**.

DIPPÉ, M. and SWENSEN, J. “An Adaptative Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis”. SIGGRAPH’84, pp. 149-158, **1984**.

DREBIN, R. A., CARPENTER, L. and HANRAHAN, P. “Volume Rendering”. Computer Graphics, Vol. 22, No. 4, pp. 65-74, August, **1988**.

ELLSWORTH, D. “A New Algorithm for Interactive Graphics on Multicomputers ”. IEEE Computer Graphics and Applications, Vol. 14, No. 4, pp. 33-40, July, **1994**.

ELVINS, T. “A Survey of Algorithms for Volume Visualization”. Computer Graphics, Vol. 26, No. 3, pp. 194-201, **1992**.

EVANS and SUTHERLAND Computer Corporation. Freedom Series Technical Report. **1992**.

FLYNN, M. J. “Very High Speed Computing Systems”, Proceeding of the IEE, Vol. 54, pp. 1901-1909, **1966**.

FUCHS, H. “Distributing A Visible Surface Algorithm Over Multiple Processors”. Proceedings ACM National Conference, pp. 449-451, October, **1977**.

FUCHS, H. and JOHNSON, B. W. “An Expandable Multiprocessor Architecture for Video Graphics”. Proceedings of the 6th Annual ACM-IEEE Symposium on Computer Architecture, pp. 58-67, April, **1979**.

FUCHS, H. and POULTON, J. “Pixel-Planes : A VLSI-Oriented Design for Raster Graphics Engine”. VLSI Design, Third Quarter, pp. 20-28, **1981**.

FUCHS, H., GOLDFEATHER, J., HULTQUIST, J., SPACH, S., AUSTIN, J., BROOKS, F., EYLES, J., and POULTON, J. “Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes”. SIGGRAPH’85, pp. 111-120, **1985**.

FUCHS, H. et al. “Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Unsing Processor-Enhanced Memories”. Computer Graphics, Vol. 23, No. 3, pp. 79-88, **1989**.

GELDER, A. V., and KIM, K. “Direct Volume Rendering with Shading via 3D Textures”. In ACM/IEEE Symposium on Volume Visualization, pp. 23-30, San Francisco, CA, **1996**.

GHARACHOLOO, N. *et al* “A Characterization of Ten Rasterization Techniques”.

Computer Graphics, Vol. 23, No. 3, pp. 355-368, **1989**.

GIERTSEN, C., and PETERSEN, J. "Parallel Volume Rendering on a Network of Workstation". IEEE Computer Graphics and Applications, Vol. 13, No. 6, pp. 16-23, **1993**.

GOLDWASSER, S. M., and REYNOLDS, R. A. "An Architecture for the Real-Time Display of Three Dimensional Objects". Proceedings International Conference on Parallel Processing, Bellaire, Michigan, pp. 269-274, August, **1983**.

GOLDWASSER, S. M., "A Generalized Object Display Processor Architecture". SIGARCH Newsletter, Vol. 12, No. 3, pp. 38-47, **1984**.

GOLDWASSER, S. M., REYNOLDS., R. A., TALTON, D. A., and WALHS, E. S. "High Performance Graphics Processor for Medical Imaging Applications". In Prallel Processing for Computer Vision and Display. Edited by: P. M. Dew, R. A. Earnshaw, and T. R. Heywood, Addison Wesley, pp. 461-470, **1989**.

GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., and BATTAILE, B. "Modeling the Interaction of Light Between Diffuse Surfaces". SIGGRAPH'84, pp. 213-222, **1984**.

GREEN, S. A., and PADDON, D. J. "A Highly Flexible Multiprocessor Solution for Ray Tracing". The Visual Computer, Vol. 6, No. 2, pp. 62-73, **1990**

GUNTHER, T., POLIWODA, C., REINHARD, C., HESSER, J., MANNER, R., MEINZER, H.-P. , and BAUR, H.-J. "VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine". In Proceedeings of 9th Eurographics Hardware Workshop, pp. 103-108, Oslo, Norway, **1994**.

HANRAHAN, P., SALZMAN, D., and AUPERLE, L. "A Rapid Hierarchical Radiosity Algorithm". Computer Graphics, Vol. 25, No. 4, pp. 197-206, **1991**.

JACKEL, D. "The Graphic PARCUM System: A 3D Memory Based Computer Architecture for Processing and Display of Solid Models". Computer Graphics Forum, Vol.4, No. 4, pp. 21-32, **1985**.

JASWINDER, P. S., GUPTA, A., and LEVOY, M. C, IEEE Computer, Vol. 27, No. 7, pp. 45-55, **1994**.

JENKINS, R. A. "New Approaches in Parallel Computing". Computers in Physics,

Vol. 3, No. 1, pp. 24-32, **1989**.

JEVANS, D. A. “A Review of Multi-Computer Ray Tracing”. Ray Tracing News, Vol. 3, No. 1, pp. 8-15, **1989**.

JUNHUI, D., and ZESHENG, T. “Parallel Frequency Domain Volume Rendering on Workstations Cluster”. Journal Tsinghua University, Scientific Technology, (china, Vol. 37, No. 9, pp. 64-68, **1997**.

KABA, J., and PETERS, J. “A Pyramid-based Approach to Interactive Terrain Visualization”. Proceedings 1993 Parallel Rendering Symposium, ACM Press, pp. 67-70, **1993**.

KAUFMAN, A., and BAKALASH, R. “Memory and Processing Architecture for 3D Voxel-Based Imagery”, IEEE Computer Gaphics and Applications, Vol. 8, No. 11, pp. 10-23, **1988**.

KAPLAN, M., and GREENBERG, D. P. “Parrallel Processing Techniques for HiddenSurface Removal”. Computer Graphics, Vol. 13, No. 2, pp. 300-307, **1979**.

KEDEM, G., and ELLIS, J. L. “The Ray Casting Machine”. in Proceedings of the 1984 Interantional Conference on Computer Design, pp. 533-538, **1984**.

KNITTEL, G. “A Scalable Architecture for Volume Rendering”. In Proceedings of the 9th Eurographics Hardware Workshop, pp. 58-69, September,**1993**.

KNITTEL, G., and Straβer, W. “A Compact Volume Rendering Accelerator”. In Proceedings 1994 Symposium on Volueme Visualization, ACM SIGGRAPH, pp. 67-74, October, **1994**.

KOBAYASHI, H., NAKAMURA, T., and SHIGEI, Y. “Parallel Processing of an Object Synthesis Using Ray Tracing”. Visual Computer, Vol. 3, No. 1, pp. 13-22, **1987**.

LACROUTE, P., and LEVOY, M. “Fast Volume Rendering Using Shear-Warp Factoring of the Viewing Transformation”. SIGGRAPH’94, **1994**.

LACROUTE, P. “Analysis of a Parallel Volume Rendering System Based on the Shear-warp Factorization. IEEE Transactions on Visualization and Computer Graphics, Vol. 2, No. 3, pp. 218-231, **1996**.

LECLERC, Y. G., and LAU Jr., S. Q. “TerraVision: A Terrain Visualization System”.

Technical Note N0. 40, SRI International, Menlo Park, CA, **1994**.

Le MAIR, W., *et all* "Strategies for Dynamic Load Balancing on Highly Parallel Computers". IEEE Transaction on Parallel and Distributed Systems, IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 9, pp. 979-993, **1993**.

LEVOY, M. "Display of Surfaces from Volume Data". IEEE Computer Graphics and Applications, Vol. 8, No. 5, pp. ???, **1988**.

LI, P. P. and CURKENDAL, D. W. "Parallel Three Dimensional Perspective Rendering". Proceedings of the Second European Workshop on Parallel Computing, pp. 320-331, **1992**.

MA, K.-L., PAINTER, J. S., HANSEN, C. D., and KROGH, M. F. "Parallel Volume Rendering Using Binary-Swap Compositing". IEEE Computer Graphics and Applications, Vol. 14, No. 4, pp. 59-68, **1994**.

MACHIRAJU, R. K., and YAGEL, R. "Efficient Feed-Forward Volume Rendering Techniques for Vector and Pararallel Processors". Proceedings of Supercomputing'93, pp. 699-708, **1993**.

MACKERRAS, P. and CORRIE, B. "Exploiting Data Coherence to Improve Parallel Volume Rendering". IEEE Parallel and Distributed Technology, Vol. 2, No. 2, pp. 8-16, **1994**.

MAGI - Mathematical Applications Group Inc. "3D Simulated Graphics Offered by Service Bureau". Datamations, Vol. 13, No. 1, **1968**.

MOLNAR, S., and FUCHS, H. "Advanced Graphisc Architecures". In Computer Graphics Principles and Practice, 2nd ed. J. D. Foley et al. Addison Wesley, Reading, Massachusetts, pp. 855-920, **1990**.

MOLNAR, S., EYLES, J., and POULTON, J. "Pixel-Flow: High Speed Rendering Using Image Composition". Computer Graphics, Vol. 26, No. 2, pp. 231-240, **1992**.

MOLNAR, S. COX, M. ELLSWORTH, D. and FUCHS, H. "A Sorting Classification of Parallel Rendering". IEEE Computer Graphics and Applications, Vol. 14, No. 4, pp. 23-32, July, **1994**.

MONTANI, C., PEREGO, R., and SCOPIGNO, R. “Parallel Volume Visualization on a Hypercube Architecture”. In Proceedings of 1992 Workshop on Volume Visualization, pp. 9-16, Boston, MA, **1992**.

MYER, T. H., and SUTHERLAND, I. E., “On the Design of Display Processors”. Communications of the ACM, Vol. 11, No. 6, pp. 410-414, June, **1968**.

NEMOTO, K., and OMACHI, T. “An Adaptative Subdivision by Sliding Boundary Surfaces for Fast Ray Tracing”. In Proceedings of Graphics Interface’86, pp. 43-48, **1986**.

NEUMANN, U. “Parallel Volume Rendering Algorithm Performance on Mesh Connected Multicomputers”. In Proceedings on the Prallel Rendering Symposium, pp. 97-104, San Jose, CA, **1993**.

NEUMANN, U. “Communication Costs for Parallel Volume-Rendering Algorithms”. ”. IEEE Computer Graphics and Applications, Vol.14, No. 4, pp. 49-58, **1994**.

NIEH, J. and LEVOY, M. “Volume Rendering on Scalable Shared-Memory MIMD Architectures”. 1992 Workshop on Volume Visualization, Boston, MA, pp. 17-24, **1992**.

NINKE, W. H. “Graphic 1 – A Remote Graphical Display Console System”. 1965 Fall Joint Computer Conference, AFIPS Conference Proceedings, Vol. 27, Part 1, pp. 39-846, **1965**.

NISHIMURA, H., OHNO, H., KAWATA, T., SHIRAKAWA, I., and OMURA, K. “LINKS-1 A Parallel Pipelined Multimicrocomputer System for Image Creation”. in Proceedings of the Tenth Interantional Symposium on Computer Architecture, ACM SIGARCH Newsletter, Vol. 11, No. 3, pp. 387-394, **1983**.

NISHITA, T., and NAKAMAE, E. “Continuous Tone Representation of Three Dimensional Objects Taking Account of Shadows and Interreflection”. SIGGRAPH’85, pp 23-30, **1985**.

OHASHI, T., UCHIKI, T. and TOKORO, M. “A Three-Dimensional Shaded Display Method for Voxel Based Representation”. Proceedings EUROGRAPHICS’85, Nice, France, pp. 221-232, **1985**.

ORTEGA, F. A., HANSEN, C. D., and AHRENS, J. P. “Fast Parallel Polygon Rendering”. Proceedings Supercomputing’93, IEEE Computer Society Press, pp.

709-718, **1993**.

PARKE, F. I. “Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems”. Computer Graphics, Vol. 14, No. 3, pp.48-56, July, **1980.**

PFISTER, H., and KAUFMAN, A. “Cube-4 a Scalable Architecture for Real Time Volume Rendering”. In Proceedings 1996 Symposium on Volume Visualization, pp. 17-24, San Francisco, CA, **1996**.

PIXAR, “PhotoRealistic RenderMan Toolkit v3.5 Reference Manual, **1994**.

PUECH, C., SILLION, F., and VEDEL, C. “Improving Interaction with Radiosity-based Lighting Simulation Programs”. Computer Graphics, Vol. 24, No. 2, pp. 51-57, **1990**.

QUINN, M. J., “Parallel Computing: Theory and Practice”, McGraw-Hill, 1994, New York, **1994**.

RECKER, R. J.m GEORGE, D. W., and GREENBERG, D. P. “Acceleration Techniques for Progressive Refinement Radosity”. Computer Graphics, Vol. 24, No. 2, pp. 50-66, March, **1990**.

SALMON, J. and GOLDSMITH, J. “A Hypercube Ray-tracer”. Proceedings of the Third Conference on Hypercube Concurrente Computers and Aplications, Vol. II, Applications, ed. G. C. Fox, ACM Press, pp. 1194-1206, **1988**.

SCHERSON, I. D., and CASPARY, E. “Multiprocessing for Ray-Tracing: A Hierarchical Self-Balancing Approach”. Visual Computer, Vol. 4, No. 4, pp. 188-196, **1988**.

SCHRÖEDER, P., and SALEM, J. B. “Fast Rotation of Volume Data on Data Parallel Architectures. In Proceedings IEEE Visualization, pp. 50-57, San Diego, CA, **1991**.

SCHRÖEDER, P., and STOLL, G. “Data Parallel Volume Rendering as Line Drawing”Proceedings of the 1992 Volume Visualization Workshop, ACM SIGGRAPH, **1992**.

SCHUMACKER, R., BRAND, B., GILLILAND, M., and SHARP, W. “Study for Applying Computer-Generated Images to Visual Simulation”. Report AFHRL-TR-69-14, Air Force Human Resources Laboratory, September, **1969**.

SPROULL, R. F., and SUTHERLAND, I. E. “A Clipping Divider”. 1968 Fall Joint Computer Conference, AFIPS Conference Proceedings, Vol. 33, Part 1, 765-775, December, **1968**.

TUY, H.K., and TUY, L.T., “Direct 2D Display of 3D Objects”, IEEE Computer Graphics ans Applications, Vol. 4, No. 10, pp. 29-33, **1984**.

WARNOCK, J., “A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures”. Technical Report TR 4-15, NTIS AD-753 671, Computer Science Department, University of Utah, Salt Lake Citym UT, June, **1969**.

WATT, A., and WATT, M., “Advanced Animation and Rendering Techniques”, Addison-Wesley, **1994**.

WESTOVER, L. “Interactive Volume Rendering”, In Volume Visualization Workshop, pp. 9-16, Chapell Hill, NC, **1989**.

WESTOVER, L., “Footprint Evaluation for Volume Rendering”, Computer Graphics, Vol. 24, No. 4, pp. 367-376, **1990**.

WHELAN, D. S. “Animac: A Multiprocessor Architecurefor Real-Time Computer Animation”. Ph. D. dissertation, California Institute of Technology, **1985**.

WHILHELMS, J. GELDER, A.V., TARANTINO, P., and GIBBS, J. “Hierarchical and Pralleizable Direct Volume Rendering for Irregular and Multiple Grids”. In Proceedings of Visualization, pp. 57-64, San Francisco, CA, **1996**.

WHITMAN, S. “Multiprocessor Methods for Computer Gaphics Rendering”. Jones and Bartlet, Boston, **1992**.

WHITMAN, S. “Dynamic Load Balancing for Parallel Polygon Rendering”. IEEE Computer Graphics and Applications, Vol. 14, No. 4, pp.41-48, **1994**.

WITTENBRINK, C. M., and SOMANI, K. “Time and Space Optimal Parallel Volume Rendering Using Permutation Warping”. Journal of Parallel and Distributed Computing, Vol. 46, No. 2, pp. 148-164, **1997**.

WITTENBRINK, C. M. “Irregular Grid Volume Rendering with Composition Networks”In Proceedings of IS&T/SPIE Visual Data Exploration and AnalysisV, page In press, San Jose, CA, SPIE, **1998**.

WRIGHT, J. R. and HSIEH, J. C. L. “A Voxel Based, Forward Projection Algorithm

for Rendering Surface and Volumetric Data". Proceedings Visualization'92, IEEE Computer Society Press, pp. 340—348, October, **1992**.