Boletim Técnico da Escola Politécnica da USP

Departamento de Engenharia Eletrônica

BT/PEE/94-10

" Ray Tracing" Paralelo

Eduardo Toledo Santos
João Antonio Zuffo

São Paulo - 1994

O presente trabalho é um resumo da dissertação de mestrado apresentada pelo Eng. Eduardo Toledo Santos, sob orientação do Dr. João Antonio Zuffo: "Avaliação do Algorítmo de Ray Tracing em Multicomputadores", defendida em 29/06/94, na Escola Politécnica.

A íntegra da dissertação encontra-se à disposição com o autor e na biblioteca de Engenharia Civil da Escola Politécnica/USP.

Santos, Eduardo Toledo
"Ray tracing" paralelo / E.T. Santos, J.A. Zuffo. -- São Paulo : EPUSP, 1994.
11p. -- (Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia Eletrônica, BT/PEE/94-10)

1. Processamento paralelo (Computação) 2. Computação gráfica I. Zuffo, João Antonio II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Eletrônica III. Título IV. Série

CDU 681.322.06
519.674

"Ray Tracing" Paralelo

EDUARDO TOLEDO SANTOS
JOÃO ANTONIO ZUFFO

Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, travessa 3, n.158
05508-900 - São Paulo, SP, Brasil
toledo@lsi.usp.br
jazuffo@lsi.usp.br

Abstract. This paper discusses some techniques for paralellizing the ray tracing algorithm on multicomputers.

1 Introdução

A *Computação Gráfica* é um campo que está recebendo cada vez mais a atenção da comunidade científica internacional. Da mesma forma, o interesse na utilização comercial de técnicas de computação gráfica também é crescente. Este interesse é devido aos enormes avanços nas áreas de microeletrônica e arquitetura de computadores que permitiram que muitas aplicações antes inviáveis computacionalmente pudessem finalmente ser atendidas. Restrições quanto à resolução espacial e de cores nos monitores e velocidade ou memória disponível nos computadores estão bastante menores do que eram há alguns anos atrás.

Atualmente a Computação Gráfica já é parte integrante do dia-a-dia da sociedade como se vê em comerciais de TV, cinema, vídeo-games, etc. Com a disponibilidade de equipamentos de alta desempenho e custo mais baixo, rapidamente surgiram aplicações em campos como publicidade, desenho industrial, visualização de modelos tridimensionais complexos, estudo de iluminação, CAD, animação, efeitos especiais, realidade virtual, visualização médica, editoração, apresentações e simulação de vôo, entre tantos outros. Em muitas destas aplicações cada vez mais é necessário gerar imagens mais sofisticadas, mais realísticas e mais rapidamente.

Genericamente, a Computação Gráfica procura transformar dados em imagens. Um segmento particular da Computação Gráfica conhecido por *Síntese de Imagens Foto-realísticas* procura gerar imagens que se aproximem daquelas que seriam obtidas fotografando-se a realidade. Estas imagens, normalmente, apresentam sombras, reflexos, transparências, texturas, etc. Dentre as técnicas usadas neste segmento destaca-se aquela denominada "ray tracing" pois é a que permitiu gerar as imagens mais realísticas até hoje produzidas por computador e inclui todos os efeitos mencionados. O "ray tracing" surgiu em 1968 (APPEL, 1968) mas permaneceu no esquecimento até praticamente 1980 pois, até então, não havia máquinas com poder de processamento suficiente para gerar imagens sofisticadas num período de tempo viável.

Apesar da grande evolução na capacidade de processamento dos equipamentos, a demanda computacional das aplicações é crescente. A síntese de imagens de objetos tridimensionais é muito cara computacionalmente e o uso de técnicas mais realísticas, com cenas de alta complexidade amplia ainda mais o custo por imagem.

Além da Computação Gráfica, outro campo que vem apresentando resultados bastante promissores é a *Computação Paralela*. Esta área já vem sendo estudada a bastante tempo, contando com vários congressos especializados realizados anualmente em todo o mundo. Apesar de menos desenvolvida comercialmente devido a fatores relacionados à dificuldade de programação dos computadores paralelos, vem paulatinamente se firmando com a presença de várias máquinas multiprocessadoras no mercado e a decisão de gigantes como *IBM* e *Cray* de montarem laboratórios e/ou grupos para pesquisar e desenvolver computadores maciçamente paralelos (BELL, 1992), tendo esta última recentemente lançado um modelo nesta linha.

A computação paralela tem-se mostrado como uma solução alternativa aos supercomputadores com um ou poucos processadores ultra rápidos já que os computadores maciçamente paralelos apresentam relação desempenho/custo mais favorável. Isto permite que se construam computadores com desempenho muito alto a custos comparativamente baixos. Também tem sido aplicada em conjunto com a computação gráfica com sucesso (BURKE & LELER, 1990).

2 "Ray Tracing"

O algoritmo básico de "ray tracing" (WHITTED, 1980) é muito simples e trata basicamente da simulação dos raios de luz que percorrem o ambiente da cena a ser sintetizada (fig. 1).

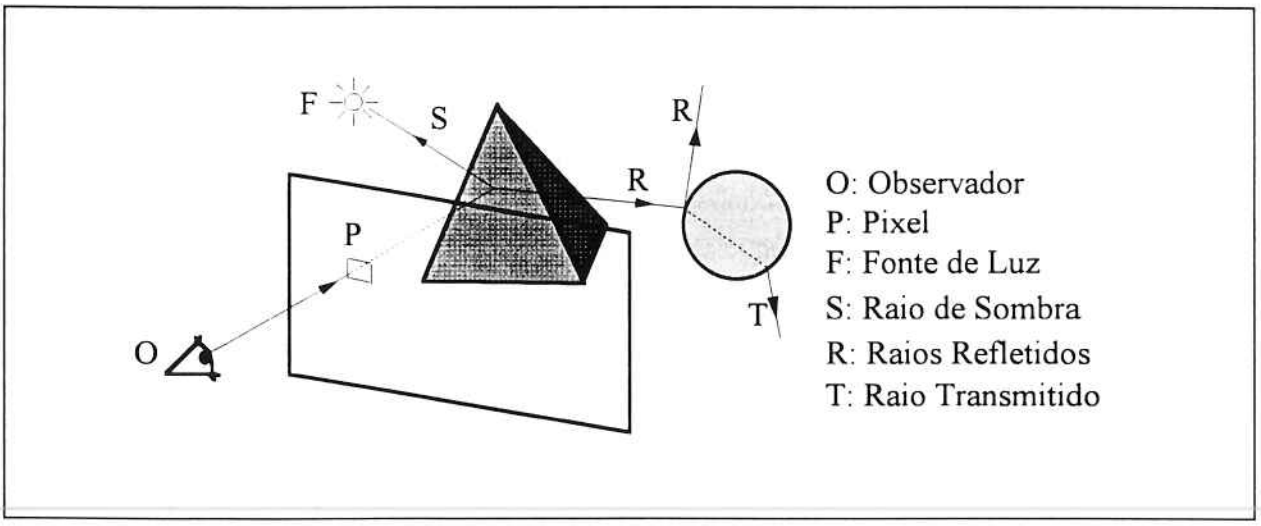

Figura 1 - "Ray Tracing"

A geração da imagem é feita de forma recursiva, segundo o algoritmo a seguir:

```
ray_tracing()
    para cada pontel faça
    •calcule direção do raio primário (do observador ao pontel),prim ;
    •cor do pontel = lança_raio(prim);
fim

lança_raio(raio)
    •intersec = det_intersecção(raio);
    •calcule iluminação local (ilum_local) no ponto de intersecção intersec;
    •calcule direcão do raio refletido (refl) para gerar a componente
    especular;
    •calcule direção do raio refratado (refr) para gerar a componente
    transmitida;
    •comp_espec = lança_raio(refl);
    •comp_trans = lança_raio(refr);
    •cor_final = ilum_local + k_s.comp_espec  + k_t.comp_trans;
    •retorne cor_final;
fim

det_intersecção(raio)
    •para cada objeto
        •calcule a intersecção do raio com objeto;
    •retorne a intersecção mais próxima da origem do raio;
fim
```

Note-se que o algoritmo basicamente lança raios recursivamente e calcula a intersecção destes com os objetos da cena. O cálculo da iluminação local exige o lançamento de "raios de sombra" em direção às fontes de luz para que se verifique quais as que iluminam o ponto interceptado. A rotina de raios de sombra é praticamente a mesma usada no lançamento de raios de luz, terminando ao detectar-se qualquer intersecção com o raio.

"Ray tracing" é um algoritmo elegante e resolve de maneira extremamente simples problemas como reflexão, refração e sombras que são de difícil solução na técnica abordada na secção anterior. Além destes, outros problemas só podem ser resolvidos de forma correta com "ray tracing", tais como profundidade de foco, penumbra, translucência e reflexão nublada.

O preço a pagar por todas as qualidades do "ray tracing" é sua enorme demanda por processamento. Por outro lado, é um algoritmo naturalmente paralelizável, o que sugere o uso de arquiteturas concorrentes para sua aceleração.

Desenvolveu-se várias técnicas para aceleração do cálculo de "ray tracing". Grande parte delas pode ser diretamente transportada para arquiteturas tipo multicomputador.

Uma discussão sobre os aspectos envolvidos na paralelização do "ray tracing" para ambientes paralelos fracamente acoplados (memória distribuída) é apresentada a seguir.

3 "Ray Tracing" Paralelo

A motivação para implementar-se um algoritmo qualquer sob forma paralela é sempre o desejo de acelerar-se o tempo de execução total de um trabalho. Procura-se obter máxima eficiência, na forma de ganho de velocidade linear. Com o algoritmo de "ray tracing" não é diferente, porém outros aspectos também são fundamentais como se verá a seguir.

A técnica de paralelização denominada *"processor farm"* é uma das mais simples e também é a que proporciona os melhores ganhos de velocidade para "ray tracing". Um processador mestre envia, a cada nó, pacotes de pontéis a serem processados por processadores escravos. Estes, ao concluirem suas tarefas, enviam ao mestre um pacote de pontéis já calculados. Esta solução apresenta baixas sobrecarga e comunicação e bom balanceamento. Não impõe praticamente nenhuma restrição ao aproveitamento de técnicas de aceleração desenvolvidas para o algoritmo seqüencial. Por outro lado, esta mesma técnica é a que apresenta o pior desempenho em outra característica importante: utilização de memória. Isto ocorre pela exigência de duplicação de todos os dados de descrição de uma cena na memória local de cada um dos nós de processamento de um multicomputador já que o cálculo de iluminação em "ray tracing" é global. A tendência atual de sintetizar-se cenas cada vez mais complexas torna esta característica altamente indesejável. A busca de maior realismo nas cenas exige um modelamento detalhado de cada objeto, adicionando-se texturas e geometrias complexas que consomem grandes quantidades de memória.

A replicação total dos dados faria com que, num multicomputador com, por exemplo, 256 processadores e 4 Mbytes de memória local por nó, a cena mais complexa que poderia ser processada tivesse apenas 4 Mbytes, sendo o total disponível 1 Gbyte !

Ao abordar-se este problema, procura-se chegar o mais próximo possível da *solução ideal* (SANTOS, 1994): para N processadores, tempo de execução T/N e memória requerida M, sendo T e M, respectivamente, tempo e memória do algoritmo seqüencial.

Na busca de atingir-se o desempenho da solução ideal, cuidado deve ser dado a aspectos como balanceamento de carga, fragmentação, sobrecargas de comunicação e pré-processamento, redundância de dados, etc.

A seguir são apresentadas algumas propostas de solução para o problema de "ray tracing" paralelo.

3.1 Partição da Imagem

As chamadas "soluções no espaço imagem" são aquelas em que praticamente todo o processamento relativo a um dado pontel é feito por um único processador. A solução *"processor farm"* mencionada antes se enquadra nesta categoria.

Os processadores cooperativamente dividem entre sí os vários pontéis da tela para gerar a imagem completa. Esta divisão pode ser estática ou dinâmica.

A divisão *estática* é feita antes do processamento e pode ou não usar algum tipo de heurística para balancear a carga atribuída a cada processador. Algumas técnicas de balanceamento estático procuram fazer uma pré-amostragem da cena para verificar qual o processamento requerido em cada região. Trechos de imagem mostrando só a cor-de-fundo terão tempo de processamento mínimo enquanto outros mostrando objetos complexos e com muitas reflexões e transparências apresentarão tempos mais longos. O tamanho de cada partição é variado de acordo com os resultados dessa amostragem, procurando-se um balanceamento.

Soluções com divisão dinâmica dos pontéis apresentam balanceamento de carga em geral muito melhor que a estática. Durante o processamento, os pontéis vão sendo calculados seqüencialmente por qualquer processador que se encontre desocupado.

Obter balanceamento uniforme (e ganho de velocidade linear) é mais simples em soluções no espaço imagem do que nas outras soluções discutidas adiante. Por outro lado, estas em geral requerem algum tipo de duplicação de dados.

A duplicação de dados permite que se elimine quase totalmente a comunicação entre processadores, que fica restrita ao envio de pacotes de pontéis processados ao nó conectado à memória de quadro. Esta pode ser uma boa solução se o custo de comunicação é muito elevado.

Para evitar a duplicação total de dados, duas metodologias foram propostas. A primeira, adequada apenas para uma arquitetura específica, envia constantemente os dados geométricos a todos os processadores e estes usam os que lhe interessam em dado instante (*"broadcasting de dados"*) (GAUDET et alli, 1988). Só o computador hospedeiro mantém a descrição completa dos objetos da cena. A outra solução (GREEN, 1990), mais geral, aloca um trecho da memória local de cada nó para fazer a função de um "cache" de objetos. Cada processador armazena em sua memória principal a descrição de uma parcela dos objetos. Ao precisar de certo dado relativo a um objeto o processador verifica se ele é o "proprietário" daquele dado. Em caso negativo verifica sua "memória cache". Persistindo a ausência, o dado é solicitado, via troca de mensagens, ao processador "proprietário" do dado.

Além da facilidade de balanceamento, as soluções no espaço imagem em geral não impõem restrições ao aproveitamento de técnicas de aceleração ou aumento de realismo desenvolvidas para máquinas seqüenciais.

3.2 Partição do Espaço

Soluções que empregam o particionamento do espaço atuam, basicamente, dividindo o espaço tridimensional do cenário em subregiões disjuntas. Cada nó do sistema recebe a descrição dos objetos contidos nas subregiões a ele alocadas.

Os raios percorrem sua trajetória através das subregiões; ao atravessar uma fronteira, o processador responsável faz o cálculo de intersecção do raio somente com os objetos presentes naquela subregião, otimizando o desempenho em relação ao algoritmo clássico. Se não for encontrada nenhuma intersecção ou forem gerados raios de reflexão ou refração, os raios são "transmitidos" à região seguinte através de comunicação entre processadores.

O primeiro aspecto a considerar nesta abordagem é o tipo de divisão do espaço, que pode ser uniforme ou adaptativa.

A divisão uniforme não leva em consideração a forma, dimensão ou posição dos objetos no cenário, prejudicando o balanceamento de carga entre os processadores. Por outro lado, o cálculo de intersecção do raio com as fronteiras e a determinação do processador responsável pela próxima subregião são facilitados.

A comunicação na divisão uniforme tende a ser apenas entre vizinhos, para propagação de raios.

Para minimizar o problema de balanceamento na divisão uniforme pode-se variar a alocação de subregiões aos processadores como mostrado na figura 2 (KOBAYASHI et alli, 1988). O balanceamento na *alocação distribuída* é duas a três vezes melhor do que aquele apresentado pela *alocação em bloco* já que qualquer região "complexa" do espaço será dividida entre os processadores. Por outro lado, o problema de *fragmentação* é maior na *alocação distribuída* pois cada fronteira pode levar potencialmente à duplicação de dados.

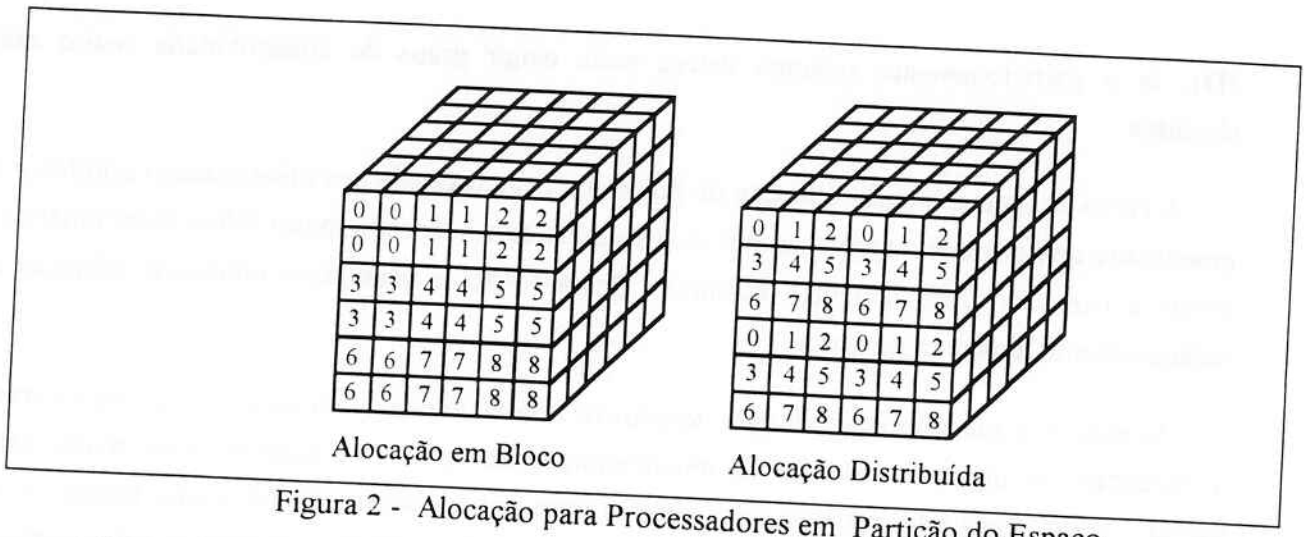

Figura 2 - Alocação para Processadores em Partição do Espaço

A fragmentação consiste no fato de que, ao particionar-se o espaço, objetos "cortados" por fronteiras pertencem a duas ou mais subregiões e deverão estar presentes nas memórias dos respectivos nós, aumentando a redundância de dados. A divisão uniforme sempre apresenta maior fragmentação do que a adaptativa.

A *divisão adaptativa* procura melhorar o balanceamento apresentado pelas soluções de particionamento do espaço. A divisão adaptativa pode ser feita segundo "cubos genéricos" (regiões topologicamente equivalentes ao cubo) ou em forma de octree (fig. 3).

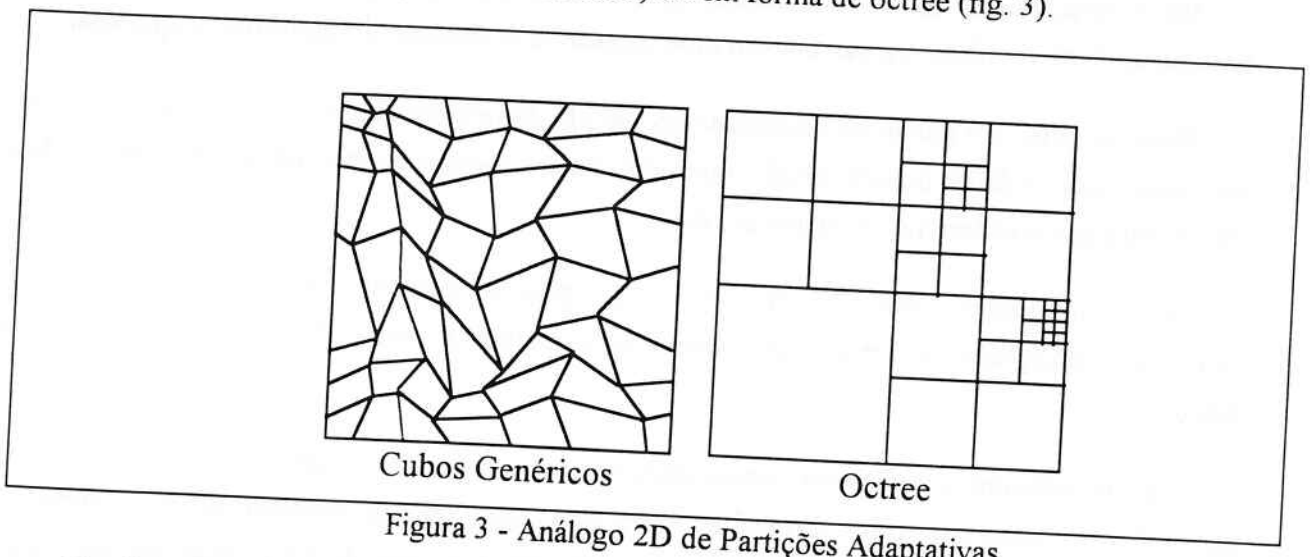

Figura 3 - Análogo 2D de Partições Adaptativas

Na divisão octree, o espaço é recursivamente dividido em octantes até que cada subregião tenha um número equivalente de objetos. Se a divisão é feita por cubos genéricos, estes são deformados, mantendo-se planas as faces, de forma a uniformizar-se o número de objetos em cada subregião e minimizar-se a fragmentação. O *pré-processamento* exigido por estas modalidades é um pouco maior que aquele do particionamento uniforme, porém ainda desprezível.

O uso de células em forma de cubos genéricos permite que a comunicação ocorra somente entre nós vizinhos, se dispuser-se de um multicomputador com conectividade de grau 6 (grade

3D). Já o particionamento segundo octree pode exigir graus de conectividade muito mais elevados.

A carga de uma subregião é função do número e complexidade dos objetos nela contidos e da quantidade de raios que a atravessa. Por esta segunda dependência é muito difícil determinar-se *a priori* a carga de uma subregião qualquer. Dessa forma é necessário utilizar-se técnicas de balanceamento dinâmico de carga.

As soluções que empregam *divisão adaptativa dinâmica* procuram balancear a carga durante a execução do algoritmo através da movimentação de fronteiras. Regiões com muita carga devem, assim, transmitir parte de seus objetos para processadores vizinhos que passam a ser responsáveis por aquela região do espaço. Consegue-se dessa forma um melhor balanceamento porém com grande sobrecarga de comunicação para transmitir os objetos "expulsos" de uma subregião. A validade desta solução dependerá das características de comunicação do multicomputador (conectividade, latência e largura de banda de comunicação).

3.3 Partição de Objetos

Da mesma forma que a paralelização por divisão do espaço, também a metodologia de particionamento de objetos é baseada em uma técnica de aceleração do algoritmo seqüencial.

Pode-se obter um ganho de velocidade no cálculo de intersecções envolvendo-se cada objeto com outro cujo cálculo de intersecção com raio seja mais simples, denominado envoltório. Bons envoltórios são a esfera, paralelepípedos, etc.

Sendo necessário verificar-se uma intersecção, primeiro testa-se o envoltório. Somente caso este seja interceptado, procede-se ao cálculo mais complexo com o objeto real, pertencente ao cenário.

Ganhos adicionais podem ser conseguidos se envolver-se grupos de envoltórios com outros envoltórios, e estes por outros. O resultado será uma "hierarquia de envoltórios". Os volumes em posições mais altas na hierarquia são testados primeiro. Em caso de haver mais de um volume interceptado o mais próximo da origem do raio é selecionado e os outros descartados. Esta técnica reduz drasticamente o número de intersecções a serem testadas para cada raio.

Ao transportar-se esta solução para um sistema paralelo, atribui-se a cada nó um ou mais envoltórios (e seus conteúdos). Cada novo raio disparado é testado por todos os processadores contra seus envoltórios. Encontrada uma intersecção com um objeto, é feito o cálculo de iluminação local e eventualmente novos raios (reflexão e/ou refração) são disparados.

O particionamento de objetos evita completamente o problema da fragmentação resultando tanto em menor consumo de memória quanto diminuição de sobrecarga devido ao processamento de objetos duplicados.

Por outro lado, o número de envoltórios testados no particionamento de objetos é muito maior que o de células na técnica de partição do espaço, como exemplifica a fig. 4 (adaptada de (SCHERSON & CASPARY, 1987)). Supondo-se que o volume 3 é o único interceptado pelo raio, uma hierarquia de envoltórios (em forma de árvore binária, no exemplo) exigiria o teste dos volumes 15, 13, 14, 9, 10, 4 e 3 (nesta ordem) enquanto que o raio no particionamento do espaço poderia percorrer apenas os volumes 1, 2 e 3, por exemplo.

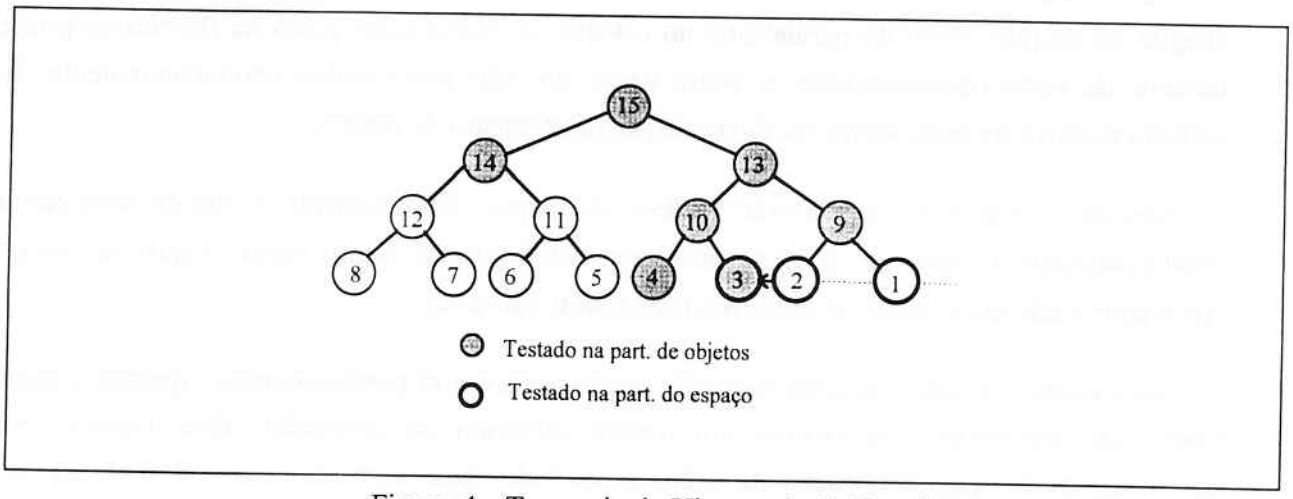

Figura 4 - Travessia da Hierarquia de Envoltórios

A construção de uma boa hierarquia de volumes é tarefa complexa de ser realizada antes da simulação dos raios. Algumas heurísticas para este fim são sugeridas em (GOLDSMITH & SALMON, 1987).

O balanceamento de carga nesta metodologia depende da escolha de uma boa hierarquia. Uma alternativa eficiente para balanceamento dinâmico é a replicação dos primeiros níveis da hierarquia em todos os nós (SCHERSON & CASPARY, 1988). Desta forma, qualquer processador ocioso poderia processar intersecções nos primeiros níveis (*demand-driven*), direcionando o raio aos processadores encarregados da intersecção de volumes e objetos em níveis mais baixos (*data-driven*). Esta proposta implementa balanceamento dinâmico bastante eficiente porém à custa de alguma redundância em memória.

Um aspecto bastante negativo da partição de objetos é que a comunicação é não local. Uma implementação eficiente exigiria uma topologia com alto grau de conectividade entre os nós tornando-a menos atraente para arquiteturas em grade 2D mais modulares que a hipercubo, por exemplo.

3.4 Outras Soluções

Outros tipos de particionamento são possíveis, dependendo de condições mais particulares.

Se o modelamento geométrico empregado na descrição do cenário for o CSG[1], é possível adotar-se uma solução com topologia em árvore. Os processadores em nós-folha da árvore CSG armazenam a descrição de sólidos primitivos enquanto nós internos contém a operação booleana a ser realizada entre seus nós filhos. Todos os objetos nos nós-folha são interceptados pelo mesmo raio. Os resultados destas intersecções são transmitidos aos nós pais que executam a operação booleana adequada e enviam o resultado a seus pais, até que o resultado da intersecção correta chegue ao nó-raiz. Além do paralelismo no cálculo de intersecções ainda há *pipelining* pois um número de raios correspondente a altura da árvore são processados concorrentemente. Esta solução poderia ser enquadrada na classe de particionamento de objetos.

Dispondo-se de hardware vetorial, pode-se interceptar, paralelamente, feixes de raios com um objeto diferente a cada vez. A eficiência desta metodologia é muito baixa já que não se pode aproveitar a coerência espacial como feito nas outras soluções.

Se é necessário gerar-se uma animação, pode-se fazer um particionamento quadro-a-quadro, com cada processador calculando um quadro diferente da animação. Esta técnica tem a desvantagem de exigir replicação de dados e nós individuais com muita memória. É adequada para processamento por estações de trabalho conectadas por rede.

O particionamento funcional é pouco adequado ao "ray tracing". Existem propostas que dividem as operações de cálculo de intersecção com envoltórios, intersecção com objetos e cálculo de iluminação entre grupos de processadores diferentes. Este tipo de enfoque em geral só é adequado quando se dispõe de processadores específicos para cada tarefa.

4 Conclusões

A paralelização do algoritmo de "ray tracing" em arquiteturas de memória distribuída apresenta sutilezas e está condicionada a inúmeros compromissos envolvendo memória e tempo de processamento.

O desempenho de cada proposta de solução paralela a este problema está sempre condicionada a determinadas características da cena não havendo uma solução vencedora em todos os casos. Um sistema de síntese de imagem eficiente deve ser capaz de escolher o melhor algoritmo e seus parâmetros para gerar cada cena. Em animações, experiência pode ser adquirida na geração de quadros anteriores.

---

[1] *Constructive Solid Geometry*

Ampla discussão sobre o algoritmo de "ray tracing" em multicomputadores pode ser encontrada em (SANTOS, 1994).

Referências

APPEL, A. Some Techniques for Shading Machine Renderings of Solids. in: **Spring Joint Computer Conference Proceedings,** AFIPS, p. 37-45, 1968.

BELL, G. Ultra Computers - A Teraflop Before It's Time. **Comm. of the ACM,** v.35, n.8, p.27-47, 1992.

BURKE, A. ; LELER, W. Paralellism and Graphics: An Introduction and Annotated Bibliography. In: **SIGGRAPH'90 Course Notes #28 - Parallel Algorithms and Architectures for 3D Image Generation.** p.111-140, 1990.

GAUDET, S. ; HOBSON, R. ; CHILKA, P. ; CALVERT, T. Multiprocessor Experiments for High-Speed Ray Tracing. **ACM Transactions on Graphics,** v.7, n.3, p. 151-179, 1988.

GOLDSMITH, J. ; SALMON, J. Automatic Creation of Object Hierarchies for Ray Tracing. **IEEE Computer Graphics and Applications,** v. 7, n. 5 , p.14-20, 1987.

GREEN, S. ; PADDON, D. A Highly Flexible Multiprocessor Solution for Ray Tracing. **The Visual Computer**, v.6, 62-73, 1990.

KOBAYASHI, H. ; NISHIMURA, S. ; KUBOTA, H. ; NAKAMURA, T. ; SHIGEI, Y. Load Balancing Strategies for a Parallel Ray-Tracing System Based on Constant Subdivision. **The Visual Computer,** v.4, p.197-209, 1988.

SANTOS, E. T. **Avaliação do Algoritmo de "Ray Tracing" em Multicomputadores**. Dissertação (Mestrado), Escola Politécnica da USP, São Paulo, 1994.

SCHERSON, I. D. ; CASPARY, E. Multiprocessing for Ray Tracing: a Hierarchical Self-Balancing Approach. **The Visual Computer,** v.4, p.188-196, 1988.

SCHERSON I. D. ; CASPARY, E. Data Structures and the Time Complexity of Ray Tracing. **The Visual Computer,** v.3, p.201-13, 1987.

WHITTED, T. An Improved Illumination Model for Shaded Display. **Comm. of the ACM,** v. 23, n.6, p. 343-349, 1980.

BOLETINS TÉCNICOS - TEXTOS PUBLICADOS

BT/PEE/93-01 - Oscilador a HEMT - 10 GHz - FÁTIMA S. CORRERA, EDMAR CAMARGO

BT/PEE/93-02 - Representação Senoidal da Voz através dos Polos do Filtro Preditor - MARCELO B. JOAQUIM, NORMONDS ALENS

BT/PEE/93-03 - Blindagens por Grades Condutoras: Cálculo do Campo Próximo - LUIZ CEZAR TRINTINALIA, ANTONIO ROBERTO PANICALI

BT/PEE/93-04 - Sistema de Otimização e Controle de Produção em Minas de Pequeno e Médio Porte - TSEN CHUNG KANG, VITOR MARQUES PINTO LEITE

BT/PEE/94-01 - Determinação das Frases de Aplicação Forense para o projeto NESPER e Tese de Mestrado IME/94, com Base em Estudos Fonéticos - MARCONI DOS REIS BEZERRA, EUVALDO F. CABRAL JUNIOR

BT/PEE/94-02 - Implementação e Teste de uma Rede Neural Artificial do Tipo KSON (Kohonen Self-Organizing Network) com Entradas Bidimensionais - MARCELO YASSUNORI MATUDA, EUVALDO F. CABRAL JR.

BT/PEE/94-03 - Transformada de Walsh e Haar Aplicadas no Processamento de Voz - ALEXANDRE AUGUSTO OTTATI NOGUEIRA, THIAGO ANTONIO GRANDI DE TOLOSA, EUVALDO F. CABRAL JÚNIOR

BT/PEE/94-04 - Aplicação de Redes Neurais ao Problema de Reconhecimento de Padrões por um Sonar Ativo - ALEXANDRE RIBEIRO MORRONE, CRISTINA COELHO DE ABREU, EDUARDO KOITI KIUKAWA, EUVALDO F. CABRAL JR.

BT/PEE/94-05 - Tudo que se Precisa Saber sobre a Prática da FFT - Transformada Rápida de Fourier (Inclui Software) - ROGÉRIO CASAGRANDE, EUVALDO F. CABRAL JR.

BT/PEE/94-06 - A Survey on Speech Enhancement Techniques of Interest to Speaker Recognition - CELSO S. KURASHIMA, EUVALDO F. CABRAL JR.

BT/PEE/94-07 - Identificação de Pulsos Decádicos em Linhas Telefônicas - ANTONIO P. TIMOSZCZUK, MÁRCIO A. MATHIAS, EUVALDO F. CABRAL JR.

BT/PEE/94-08 - Implementação e Teste de Filtros do Tipo Adaptativo e "Notch" para a Remoção de Interferência de 60 Hz em Sinais de Eletrocardiograma - FLÁVIO ANTÔNIO MENEGOLA, JOSÉ AUGUSTO DE MATTOS, JOSÉ GOMES G. FILHO, SIDNEY SILVA VIANA, EUVALDO F. CABRAL JR.

BT/PEE/94-09 - Compressão de Sinais de Voz utilizando Transformadas de Karhunen-Loève, Fourier e Hadamard - IVAN LUIS VIEIRA, LUIZ FERNANDO STEIN WETZEL, EUVALDO F. CABRAL JR.