Uso da Heterogeneidade para aceleração de *Ray Tracing*

Maurício Antônio de Castro Lima* Wagner Toledo Corrêa† Wagner Meira Júnior‡
Márcio Luiz Bunte de Carvalho§

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais
Caixa Postal 702 – Belo Horizonte – MG – CEP 30.161-970

Resumo

O objetivo deste trabalho é analisar os ganhos que podem ser obtidos adaptando-se um algoritmo computacionalmente caro a uma nova arquitetura de computadores. No presente caso, o algoritmo objeto da análise é a técnica de síntese de imagens denominada *ray tracing*, conhecida por produzir imagens com alto grau de realismo, mas a um elevado custo computacional. A estratégia utilizada para se acelerar tal técnica foi explorar o paralelismo inerente a ela num ambiente heterogêneo constituído por máquinas SIMD e MIMD. As implementações mostraram resultados bastante satisfatórios.

Abstract

The aim of this work is to analyse the gains that can be obtained by adapting a computationally expensive algorithm to a new computer architecture. In the present case, the analysed algorithm is the image synthesis technique called ray tracing known to produce realistic images but a high computational cost. The strategy used to accelerate this technique was to explore the inherent parallelism of it in an heterogeneous environment consisting of SIMD and MIMD machines. The implementations showed quite satisfactory results.

Trabalho financiado com recursos da FAPEMIG (proc. TEC 1113/90) e do CNPq (proc. 502353/91-0(NV).

*Bacharelando em Ciência da Computação (UFMG); Algoritmos paralelos, Computação Gráfica; E-mail: mlima@dcc.ufmg.br

†Bacharelando em Ciência da Computação (UFMG); Algoritmos paralelos, Computação Gráfica; E-mail: wagner@dcc.ufmg.br

‡Mestrando em Ciência da Computação (UFMG); Algoritmos paralelos, redes neuronais, programação matemática; E-mail: meira@dcc.ufmg.br

§Doutorando em Pesquisa Operacional (UC Berkeley); Algoritmos paralelos, programação matemática algebra linear numérica, redes neuronais; Professor do Departamento de Ciência da Computação – UFMG; E-mail: mlbc@dcc.ufmg.br

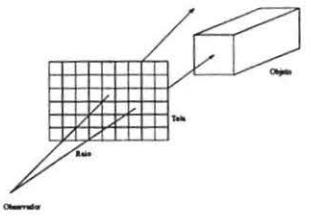

Figura 1: Princípio básico do algoritmo de Ray Tracing

1 Introdução

Dá-se o nome de "síntese de imagens" à criação, na tela de um computador, de uma figura bidimensional de um mundo tridimensional. A cena do mundo tridimensional é modelada utilizando-se "primitivas gráficas" como esferas, cilindros, cones e poliedros, todas elas sujeitas a transformações como mudança de escala, rotação em torno dos eixos coordenados, translação e cisalhamento que são as denominadas transformações afins. Essas transformações são definidas por matrizes 4×4, denominadas "matrizes de transfomação" ([Whi80], [Gla89], [Hil90] e [FvDFH90]). A cada objeto, também são associados uma cor definada pelas contribuições das cores básicas (vermelho, verde e azul) e coeficientes de reflexão difusa, de reflexão especular, de transparência e de brilho, proporcionando um alto grau de flexibilidade para a modelagem da cena. A questão principal em síntese de imagens é a determinação da cor de cada *pixel* da tela. *Ray tracing* é uma técnica de síntese de imagens poderosa e conceitualmente simples, pois permite modelar de forma simples efeitos complexos como reflexões, refrações e sombras. Porém apresenta a desvantagem de possuir um alto custo computacional. Com objetivos de diminuir esses custos, foram feitas implementações em ambientes paralelos, procurando explorar as potencialidades de um ambiente heterogêneo. Na primeira parte desse trabalho, a técnica de *ray tracing* é descrita. A seguir, são abordados os principais métodos para aceleração da técnica. Por fim, as implementações em ambientes paralelos são apresentadas, seguidas dos resultados que foram obtidos.

2 O algoritmo básico de *Ray Tracing*

Para entender o algoritmo básico de *ray tracing*, pode-se imaginar que um observador está olhando para uma cena através da tela do computador e que raios de luz provenientes da cena atingem o olho do observador passando pelos centros dos *pixels*. A cor de um *pixel* será a cor do raio proveniente da cena que passa pelo seu centro. Como nem todo raio proveniente da cena atinge o observador, na realidade, o processo é feito em sentido inverso. Para cada *pixel* da tela, lança-se um raio, passando pelo seu centro, partindo do olho do observador em direção à cena, conforme representado na figura 1. Determina-se qual objeto da cena o raio atinge primeiro e em que ponto ocorreu a interseção. Esse processo resolve automaticamente o problema de superfícies escondidas, uma vez que a primeira superfície atingida pelo raio é o objeto mais próximo ao olho do observador. Levando-se em conta as fontes de luz presentes na cena, um modelo de iluminação é aplicado ao ponto de interseção. As diversas componentes da luz são calculadas e somadas. A cor resultante é a cor do *pixel*.

Normalmente, um modelo de iluminação determina a quantidade de luz que chega em um ponto somando três componentes básicas. A primeira é a contribuição local, que depende apenas de fontes de luz reais. A segunda componente é a quantidade de luz que chega ao ponto devido a reflexão da luz por um objeto brilhante. A última contribuição é a quantidade de luz que é transmitida por refração através de um objeto transparente.

Chama-se "raio primário" aquele que parte do olho do observador e segue em direção à cena passando pelo centro de algum *pixel*. Na figura 2, **O** representa o olho do observador e **p** o ponto de interseção mais próximo de um raio primário com os objetos da cena. A luz I que chega a **O** vinda de **p** é soma da luz $L1$ local em **p**, da luz $R1$ devida a reflexão e da luz $T1$ devida a refração. Da mesma forma que I, $R1$ e $T1$ também são formadas a partir de suas próprias três componentes. A luz $R1$ é a soma da componente $L2$

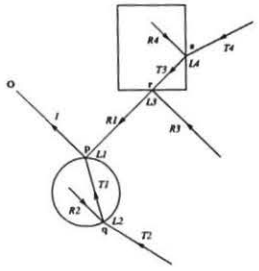

Figura 2: A luz que chega ao observador é a soma das componentes local, refletida e transmitida.

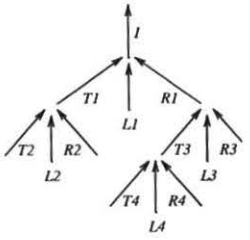

Figura 3: A árvore de contruibuições de luz.

local a **q**, da luz refletida $R2$ e da luz transmitida $T2$. Analogamente, $L3$, $R3$ e $T3$ se juntam em **r** para formar $T1$. $R3$, por sua vez, é a soma de $L4$, $R4$ e $T4$ em **s** e assim sucessivamente.

A figura 3 mostra as várias componentes da luz organizadas em uma árvore de contribuições de luz, com as componentes transmitidas nos troncos esquerdos, as refletidas nos trocos direitos e as locais nos troncos centrais. A quantidade de luz que chega a cada nodo é igual a soma das quantidades de luz que chegam em cada um de seus filhos.

Na figura 2, para se determinar o ponto **r** onde a componente refletida $R1$ é calculada, é lançado um novo "raio secundário" a partir de **p**. A direção desse raio é tal que o ângulo de incidência é igual ao ângulo de reflexão. Analogamente, para se determinar o ponto **q** onde a componente transmitida $T2$ é calculada é lançado outro raio secundário a partir de **p** na direção de refração.

Outra utilidade dos raios secundários é determinar quando um ponto está à sombra em relação a uma fonte de luz. Nesse caso, um raio secundário é lançado do ponto à fonte de luz. Se o raio interseccionar algum objeto opaco entre o ponto e a fonte de luz, o ponto estará à sombra. Se forem encontrados objetos transparentes no caminho, a intensidade da luz é diminuída, levando em conta algum índice que indique o grau de transparência do objeto.

A partir de uma modelagem baseada na ótica geométrica, é possível obter imagens com alto grau de realismo, contendo efeitos visuais interessantes produzidos, por exemplo, por espelhos, lentes e aquários. Podem existir reflexões múltiplas em que um raio é refletido por vários objetos brilhantes antes de atingir o observador e combinações complexas de reflexão, refração e sombras.

2.1 Intersecção de um raio com um objeto

A cena é representada por uma "lista de objetos" na qual estão armazenados os objetos com as suas respectivas matrizes de transformação. A principal tarefa em *ray tracing* é determinar onde um raio intersecciona um objeto. Essa tarefa deve ser executada para todos os objetos da cena. Um raio **r** qualquer é dado

5.1 A Zephyr Wavetracer

5.1.1 Arquitetura

A Zephyr Wavetracer (WT) é uma máquina SIMD (Single Instruction Multiple Data) com 8192 processadores de 1 bit cada. Esses 8K processadores são dispostos em duas placas, cada uma com 4K processadores. Essa máquina é ligada a uma *workstation* hospedeira por uma interface SCSI.

A memória total da máquina (da ordem de 256 megabytes) é igualmente dividida entre os processadores. Cada processador possui dois tipos de memória, a primeira é da ordem de 2 Kbits e é interna ao processador, já a segunda é externa a cada processador possui 32 Kbytes.

Os processadores podem ser organizados bi ou tridimensionalmente via *software*, no caso, para uma máquina com 8k processadores, o arranjo bidimensional tem dimensões 64 × 128 e o tridimensional 16 × 32 × 16.

Às dimensões do arranjo de processadores denomina-se espaço de solução, assim, num arranjo de 64 × 128 pode-se manipular simultânea e identicamente 8192 elementos.

5.1.2 Processamento Virtual

Para suportar um espaço de soluções com dimensões maiores que as dos arranjos de processadores, o controlador particiona a memória de cada processador em n partes iguais e associa cada uma delas a um nodo no espaço de soluções. Assim, sempre que uma instrução for gerada para o arranjo de processadores, o controlador instrui a sua execução para cada uma das partições via alterações de endereçamento, ou seja, o controlador trata cada processador real como n processadores virtuais.

5.1.3 Linguagem MultiC

Um programa de aplicação para a WT é escrito como um programa para a *workstation* hospedeira em MultiC, uma extensão da linguagem Ansi C. Tendo em vista que o programa executa na *workstation* e não no controlador da WT, o programa de aplicação compilado tem acesso às facilidades da *workstation*, como sistema de arquivos, rede e ambientes gráficos. As dimensões do arranjo de processamento virtual são especificadas em tempo de execução, sem o prévio conhecimento da configuração do arranjo físico. Como resultado, um programa pode rodar em máquinas com quaisquer quantidades de processadores sem necessidade de recompilação.

As principais extensões ao Ansi C providas pelo MultiC são as seguintes:

1. O especificador de tipos multi, que declara uma variável que tem um valor numérico independente em cada processador virtual. O especificador de tipo uni também foi acrescentado à linguagem com a finalidade de declarar variáveis que tenham apenas um valor que é armazenado na *workstation*.
2. Um operador de comunicação interprocessador que permite o trânsito e intercâmbio de dados entre processadores. Deve ser ressaltado que o fluxo de dados é idêntico para todo o espaço de soluções virtual.
3. Operadores de redução que aplicam a operação indicada a todos os valores da expressão multi alvo nos processadores virtuais ativos, condensando o resultado em um único tipo uni. As operações de redução são: soma, subtração, multiplicação, divisão, operações lógicas (e, ou , ou-exclusivo) e operadores de máximo e mínimo.

5.2 Arquitetura PVM

O PVM (*Parallel Virtual Machine*) [GBD+93] é um pacote de programas e bibliotecas que permite que uma rede de computadores heterogêneos UNIX seja usada como um único e grande computador paralelo. Assim, grandes problemas computacionais podem ser resolvidos usando o poder agregado de vários computadores.

Executar operações concorrentemente: consiste em aproveitar o paralelismo inerente ao processo de *ray tracing*. O objetivo dessa técnica é explorar o fato de que a cor de um *pixel* não depende da cor de outros *pixels*, isto é, os raios lançados para o interior da cena são independentes uns dos outros. Assim, o processamento de um raio pode ocorrer concorrentemente ao processamento de outros raios. [Max81, PB85, GS85, NOK+83, KNS87, DW85, NO86, CWBV85, Ull83]

4 Estratégia proposta

Para reduzir o tempo gasto na geração de uma imagem utilizando *ray tracing* exaustivo, foi escolhida a estratégia de aproveitar o paralelismo inerente ao processo. O objetivo dessa estratégia é investigar o impacto da utilização de uma nova arquitetura na solução de um problema computacionalmente caro, de forma que sejam aproveitadas melhor as suas características. Para tanto, foram implementadas três técnicas: (i) utilizando uma máquina SIMD (Single Instruction Multiple Data); (ii) utilizando uma máquina MIMD (Multiple Instruction Multiple Data); (iii) adotando uma técnica híbrida, implementada num ambiente heterogêneo, aproveitando os pontos positivos de cada uma das técnicas anteriores.

Na primeira técnica, os raios primários recebem um tratamento coletivo para toda a cena por parte de uma máquina de arquitetura SIMD. Todos os raios primários são lançados de uma só vez. Isso é possível graças ao fato de que, para cada objeto da cena, todos os raios são submetidos à mesma transformação. A lista de objetos pode ser única para todos os processadores e cada um desses processadores pode tratar um único raio. Isso caracteriza uma situação ideal para uma abordagem SIMD. Uma vez processados os raios primários, os raios secundários são tratados pelo processador anfitrião da máquina SIMD. Esse processador recebe como informação uma matriz cujas posições representam os *pixels* da imagem. Cada posição contem um identificador do primeiro objeto atingido pelo raio primário correspondente, a direção desse raio (necessária para a determinação dos raios secundários) e o tempo de intersecção. Por fim, o cálculo das cores dos *pixels* é realizado seqüencialmente pelo processador anfitrião. Essa primeira técnica será referenciada como método SIMD/seqüencial.

Na segunda técnica, referenciada como método distribuído, utiliza-se uma máquina MIMD. O processador pai divide a imagem em *frames* retangulares que são distribuídos entre os processadores filhos. Cada um desses processadores, por sua vez, realiza a confecção da imagem referente ao *frame* que recebeu, executando seqüencialmente todas as operações do algoritmo: lançamento de raios primários e secundários(reflexões e refrações). Com isso, os processadores filhos apenas necessitam trabalhar os raios primários referentes à parte da imagem que lhes cabe. Para o processamento dos raios secundários, vale lembrar que cada processador filho necessita de uma cópia completa da estrutura de dados para verificar a interação desses raios com os demais objetos da cena.

A terceira técnica, referenciada como método heterogêneo, combina as potencialidades das duas anteriores. Primeiro, a máquina SIMD processa os raios primários. Depois, a máquina MIMD processa os raios secundários. Nessa etapa, uma otimização adicional é implementada: o processador pai recebe a matriz de informações que contem os identificadores dos objetos atingidos pelos raios primários, as direções desses raios e os tempos de interseção fornececida pela máquina SIMD e então distribui os *frames* aos processadores filhos somente se houver objetos naquele *frame*, evitando que esses processadores sejam ocupados por *frames* degenerados. Essa informação é obtida durante o processamento dos raios primários, da seguinte forma: se em um *frame*, nenhum raio primário interseccionou algum objeto, então esse *frame* é degenerado, não exigindo nenhum processamento adicional.

5 Descrição do Ambiente

As técnicas descritas anteriormente exigem para a sua implementação a disponibilidade de uma máquina SIMD e de outra MIMD. A máquina SIMD utilizada foi a *Zephyr Wavetracer*. Já a máquina MIMD, foi emulada utilizando a rede *workstations SUN* do DCC - UFMG a partir da utilizaçaõ do pacote para programação paralela denominado PVM. As características dessas máquinas são abordadas a seguir.

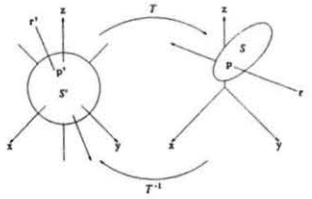

Figura 4: Intersecção de um raio com um objeto transformado (Hill).

parametricamente em função do tempo t por

$$\mathbf{r}(t) = \mathbf{s} + \mathbf{d}t \tag{1}$$

onde $\mathbf{s}$ é o vetor posição do ponto inicial do raio e $\mathbf{d}$ é o vetor direção do raio. Então, determinar em que ponto um raio atinge um objeto se reduz a determinar em que tempo o raio atinge o objeto. Seja S um elipsóide que foi gerado a partir da aplicação da tranformação T sobre uma esfera primitiva S' (vide figura 4). Deseja-se determinar a intersecção do raio original, $\mathbf{r}(t)$, com o elipsóide. Em lugar de se transformar a esfera e interseccioná-la com $\mathbf{r}(t)$, submete-se $\mathbf{r}(t)$ à transformação T^{-1} e faz-se a intersecção do raio obtido, $\mathbf{r}'(t)$, com a esfera primitiva. Se o raio original passa através do ponto $\mathbf{p}$ em S no tempo t_0, pode-se garantir que o raio transformado passa através do ponto $\mathbf{p}'$ em S' no mesmo tempo t_0, ou seja, o tempo de intersecção do raio transformado com a esfera primitiva é idêntico ao tempo de intersecção do raio original com o elipsóide.

É fato conhecido que, numa cena típica, o custo dos cálculos de interseção é bem maior do que a soma dos custos das tarefas restantes como, por exemplo, cálculos de sombreamento. De acordo com Whitted [Whi80], esse custo pode chegar a representar mais de 95% do custo total da geração da imagem para cenas complexas. Isso ocorre devido ao fato de que o processo de intersecção equivale a resolver uma equação algébrica. Mesmo no caso de superfícies simples, como esferas, essa equação possui grau 2. No caso de superfícies mais complicadas, como o toro, essa equação possui grau 4.

A implementação mais direta do algoritmo de *ray tracing* é conhecida por "*ray tracing* exaustivo". Esse nome, dado por Glassner [Gla89], é devido ao fato de que cada raio lançado, seja primário, seja secundário, é interseccionado com todos os objetos da cena. O excessivo número de cálculos de intersecção torna o *ray tracing* exaustivo muito inefeciente. Este trabalho, como vários outros, alguns dos quais serão abordados na próxima secção, tem como objetivo apresentar um algoritmo mais eficiente para reduzir o tempo gasto na geração de uma imagem utilizando a técnica de *ray tracing*.

3 Técnicas de otimização

A maioria das técnicas de aceleração de *ray tracing* concentra-se no pròblema do cálculo de intersecções, assumindo que apenas uma parte desprezível do tempo é gasta nas demais tarefas. Basicamente, existem quatro estratégias para se acelerar o processo de *ray tracing*:

Intersecões com cálculos mais rápidos: o objetivo dessa estratégia é reduzir o custo médio de interseccionar um raio com a cena. Isso pode ser conseguido utilizando algoritmos eficientes para objetos primitivos específicos ou diminuindo o número de intersecções de raios com objetos a serem feitas. [Han83, SA84, Whi80, RW80, Ger86, Rot82, KK86, GS87, Gla84, Kap85, Jan86, FTI86, HG86, OM, AK87]

Lançar menos raios: o objetivo dessa estratégia é reduzir o número de raios a serem interseccionados com a cena, sejam os raios primários ou secundários. [HG83, Coo86, Kaj83, LRU85, CPC84, Pur86]

Raios generalizados: nessa estratégia, raios são agrupados sob a forma de feixes [HH84], cones [Ama84] ou lápis [STN87], para obter mais rapidez no cálculo das interseções. Porém restrições são impostas a classe de objetos primitivos e a precisão no cálculo das interseções é prejudicada.

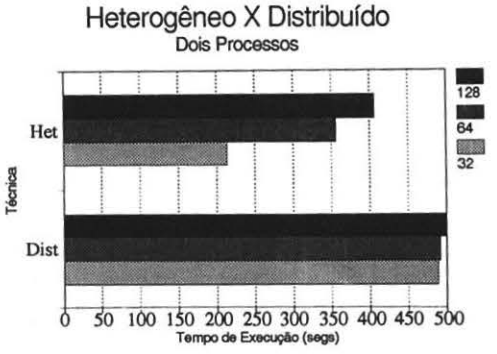

Figura 7: Comparação entre os métodos heterogêneo e distribuído, usando dois processos filhos

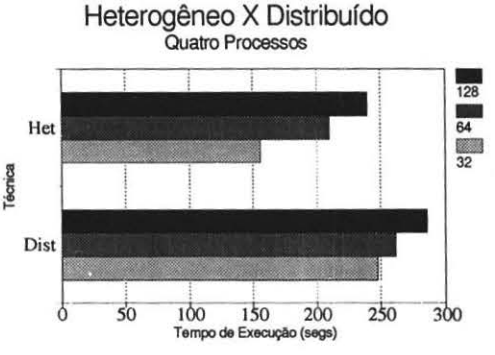

Figura 8: Comparação entre os métodos heterogêneo e distribuído, usando quatro processos filhos

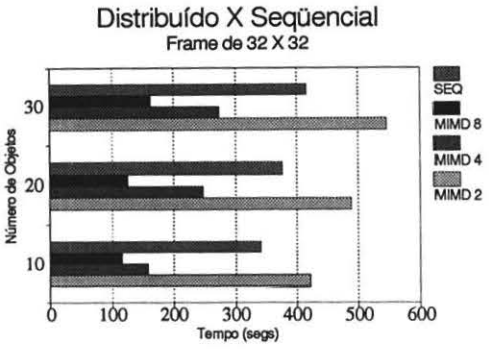

Figura 6: Comparação entre o distribuído e o seqüencial

entre processador pai e processadores filhos dos métodos distribuído e heterogêneo em função das dimensões dos *frames* para a geração de imagens de 384 × 384. Pode-se notar o fluxo de dados no método distribuído é sempre menor do que no método heterogêneo que é sempre constante (imagem processada). Para o método heterogêneo, os processadores filhos recebem informações sobre o processamento dos raios primários (objeto em que o raio interseccionou primeiro, a direção do raio primário e tempo de intersecção) e retornam a imagem processada. Verifica-se também que à medida em que as dimensões dos *frames* diminuem, o fluxo também diminui devido ao fato de que *frames* degenerados não são enviados aos processadores filhos.

Já o gráfico da figura 11 representa o número total de *pixels* processados em função das dimensões dos *frames* em cada um dos métodos: 128, 64 e 32. Verifica-se que à medida em que as dimensões dos *frames* tornam-se menores, uma quantidade menor de *pixels* é processada no método heterogêneo. O gráfico foi produzido a partir da amostragem da quantidade desses *frames* degenerados para as diversas dimensões. Para o método distribuído, sempre são processados todos os *pixels*, devido a ausência de informações quanto à existência de *frames* degenerados. Para *frames* grandes, os números de *pixels* processados nos dois métodos tendem a ser iguais devido ao fato de que a probabilidade de existir um *frame* degenerado diminui com o aumento do tamanho do *frame*, forçando o método heterogêneo processar a mesma quantidade de *frames* que o método distribuído.

Pode-se concluir que as execuções realizadas no ambiente heterogêneo são mais eficientes do que as realizadas no distribuído, pois no primeiro caso foi explorada toda a potencialidade da máquina *Zephyr wavetracer* no processamento dos raios primários e evitou-se o processamento de *frames* degenerados por parte das *workstations*. Isso justifica a união das abordagens MIMD e SIMD, estabelecendo ganhos satisfatórios no processo de geração de imagens.

6.5 Comparação das técnicas

O gráfico da figura 12 representa a comparação dos tempos de execução dos quatro métodos: seqüencial, SIMD/seqüencial, distribuído com 8 processadores filhos e heterogêneo com 8 processadores filhos, em função do número de objetos na cena. Verifica-se que o seqüencial apresentou sempre os maiores tempos, seguido do SIMD/Sequencial, MIMD e Heterogêneo, que foi o mais rápido.

7 Análise da técnica heterogênea

Dentre as três técnicas apresentadas nesse trabalho, como visto na seção anterior, a que apresentou melhores resultados foi a heterogênea. A seguir serão feitas algumas considerações com relação a esta técnica de

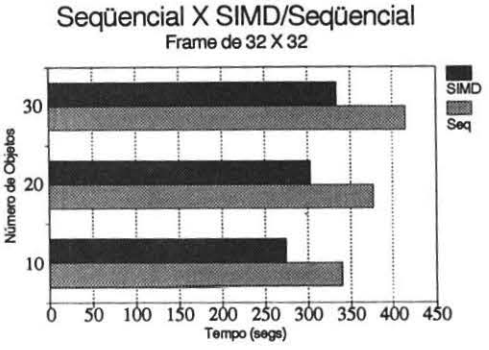

Figura 5: Comparação entre o seqüencial e o SIMD/seqüencial

os raios primários. Então o processador pai distribui os *frames* entre os processadores filhos somente se houver objetos naquele *frame*, evitando que *frames* degenerados (sem objetos) sejam processados.

6.2 Seqüencial x SIMD/Seqüencial

O gráfico da figura 5 mostra a comparação dos tempos de execução dos métodos seqüencial e SIMD/seqüencial (primeira técnica), para a geração de imagens de 384 × 384 *pixels* em função do número de objetos. Pode-se notar que para ambos os métodos, o tempo de execução diminui à medida em que o número de objetos diminui. Além disso, percebe-se que a abordagem SIMD/seqüencial apresentou sempre melhores tempos, demonstrando ganho no processamento coletivo dos raios primários por parte da máquina *Zephyr Wavetracer*.

6.3 Seqüencial x Distribuído

O gráfico da figura 6 mostra a comparação dos tempos de execução dos métodos seqüencial e distribuído (segunda técnica), com 8, 4 e 2 processadores filhos, para a geração de imagens de 384 × 384 *pixels* em função do número de objetos. Novamente pode-se notar que para ambos os métodos, o tempo de execução diminui à medida em que o número de objetos diminui. Além disso, percebe-se que a abordagem distribuída apresentou sempre melhores tempos e que à medida em que o número de processadores filhos é aumentado os ganhos do processamento são maiores. Isso demonstra que esta segunda técnica também promove melhoras ao algoritmo de *ray tracing*.

6.4 Distribuído x Heterogêneo

Os gráficos das figuras 7, 8, 9 mostram a comparação dos tempos de execução dos métodos heterogêneo (terceira técnica) e distribuído (segunda técnica) para a geração de imagens de 384 × 384 *pixels* para dimensões de *frames* variados: 128 × 128, 64 × 64, 32 × 32. Pode-se notar que o tempo de processamento aumenta com a diminuição do número de processadores utilizados, em virtude da diminuição do grau de paralelismo da aplicação. Além disso, verifica-se que o tempo de execução decresce à medida em que o número de *frames* é aumentado, ou seja, à medida em que as dimensões dos *frames* tornam-se menores. Isso deve-se basicamente ao fato de que embora haja um fluxo maior de *frames*, a quantidade de dados transmitida por *frame* é menor, além disso o percentual de *frames* degenerados quando da utilização de *frames* menores é maior.

Isso pode ser visto no gráfico da figura 10, que mostra a comparação do fluxo total de dados realizado

Sob o PVM, o usuário define uma coleção de computadores seriais, paralelos e vetoriais que se comportarão como um único computador paralelo com memória distribuída. Desta forma, denomina-se máquina virtual esses computadores lógicos com memória distribuída e hospedeiro cada um dos computadores reais. O PVM provê funções para disparar tarefas automaticamente nas máquinas virtuais e permitir que as tarefas se comuniquem e sincronizem entre si. Uma tarefa é definida como sendo a unidade de computação do PVM de forma análoga a processos UNIX. Freqüentemente ela é um processo UNIX, mas não necessariamente. Aplicações, que podem ser escritas em C ou FORTRAN 77, podem ser paralelizadas usando construções para trocas de mensagens comuns a muitos computadores de memória distribuída. Um bom exemplo de utilização do PVM é na implementação de aplicações do tipo "cliente - servidor".

PVM suporta heterogeneidade nos níveis de aplicação, de máquina e de rede. Em outras palavras, ele permite que tarefas de aplicação explorem a arquitetura mais adequada à solução do problema enfocado. Todas as conversões de dados necessárias são automaticamente efetuadas para viabilizar a comunicação entre computadores que usem diferentes representações de inteiros e reais.

O sistema PVM é composto de duas partes. A primeira parte é um *daemon*, chamado *pvmd3* que reside em todos os computadores formando a máquina virtual. (Um exemplo de um *daemon* é o programa *sendmail* que manipula todas as mensagens eletrônicas - *mails* que chegam e saem em um sistema UNIX.) *pvmd3* foi projetado de tal forma que qualquer usuário com um *login* válido possa instalá-lo em uma máquina. Ao rodar uma aplicação PVM, ele executa *pvmd3* em um dos computadores, disparando o *daemon* em cada um dos outros computadores e criando a máquina virtual definida pelo usuário. A aplicação PVM pode então ser disparada de qualquer dessas máquinas como uma tarefa UNIX normal. Usuários podem configurar máquinas virtuais com sobreposição, e cada um pode executar várias aplicações PVM simultaneamente.

A segunda parte do sistema é uma biblioteca de rotinas. Essa biblioteca contem rotinas que podem ser chamadas pelo usuário para troca de mensagens, disparo de processos, coordenação de tarefas e modificação da máquina virtual. Os programas de aplicação devem incluir essa biblioteca para usar o PVM.

6 Resultados

A seguir serão apresentados os resultados das análises das três técnicas de aceleração descritas e da implementação sequencial quanto a vários dos fatores que podem influenciar a geração de imagens.

As cenas para os testes foram criadas utilizando-se esferas posicionadas aleatoriamente no espaço, com finalidade de analisar os ganhos e as perdas de cada método.

6.1 Ambiente de execução

Os testes foram realizados na rede de *workstations SUN* do DCC - UFMG, na máquina *Zephyr Wavetracer* de arquitetura SIMD ligada a uma *Sparc Station 2* (processador anfitrião) também conectada à rede do departamento. As execuções foram realizadas da seguinte forma:

Método seqüencial: como processador anfitrião foi utilizado uma *Sparc Station 2*;

Método SIMD/seqüencial: foi utilizada a máquina *Zephyr Wavetracer* de arquitetura SIMD ligada a um processador anfitrião (*Sparc Station 2*). Nesse método, inicialmente, a máquina *Zephyr Wavetracer* processa os raios primários de forma coletiva. Depois, a *Sparc Station 2* processa sequencialmente os raios secundários;

Método distribuído: implementado através de uma rede de *workstations* dos tipos *Sparc Station SLC* emulando uma máquina MIMD através do pacote PVM. Nesse método, o processador pai (*Sparc Station 2*) distribui os *frames* aos processadores filhos (*Sparc Station SLC*). Estes processam as etapas do algoritmo de *ray tracing* (raios primários e secundários) de forma sequencial somente na porção de imagem do *frame* que recebeu e retornam essa imagem ao processador pai.

Método heterogêneo: implementado usando a máquina *Zephyr Wavetracer*, uma *Sparc Station 2* como processador anfitrião/pai e uma rede de *workstations* do tipo *Sparc Station SLC* como processadores filhos se comunicando via PVM. Nesse método, a máquina *Zephyr Wavetracer* processa de forma coletiva

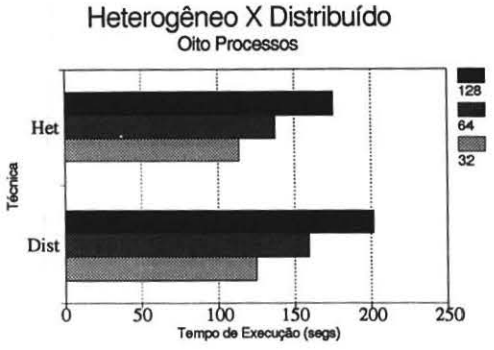

Figura 9: Comparação entre os métodos heterogêneo e distribuído, usando oito processos filhos

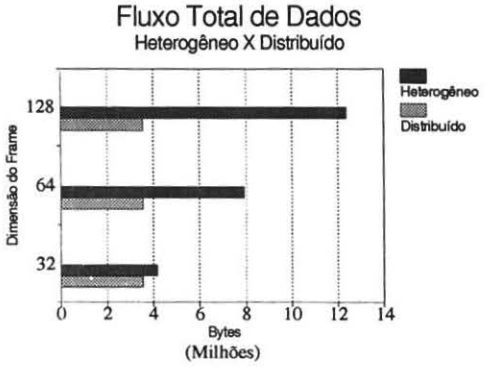

Figura 10: Comparação do fluxo total de dados entre os métodos

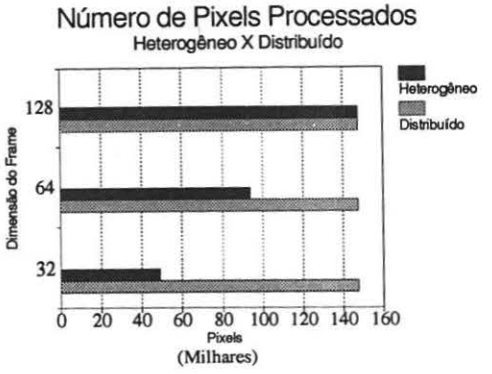

Figura 11: Número total de pixels processados em cada métodos

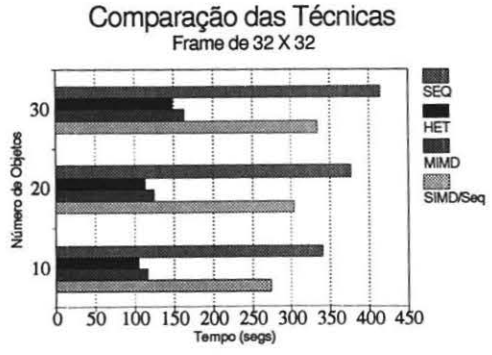

Figura 12: Comparação das técnicas

aceleração.

- A técnica não impõe restrições à classe de objetos primitivos, ao contrário de outras estratégias como a de Raios Generalizados.
- Os raios primários e secundários são tratados de maneiras diferentes. Os raios primários são tratados pela máquina SIMD (*Zephyr Wavetracer*) e os secundários pela máquina MIMD emulada (*workstations*).
- A técnica não exige nenhum pré-processamento, ao contrário de outras estratégias como a de Lançar menos Raios.
- Não há ganho em termos de ordem de complexidade, isto é, o tempo de execução ainda varia linearmente com relação o numero de objetos, mas é reduzido por um fator constante.
- O ganho em termos de tempo é otido em detrimento de um gasto adicional em memória para armazenar as informações processadas na primeira etapa (raios primários) pela máquina SIMD necessárias a máquina emulada MIMD na segunda etapa (raios secundários).
- A técnica é bastante simples, mas sua implementação depende do ambiente heterogêneo disponível e exige que os algorimos sejam adaptados para se adequar à arquitetura. No caso estudado, o algoritmo de *ray tracing* exaustivo foi reescrito em código Multi C para a máquina *Zephyr WaveTracer*, aproveitando a facilidade de trasmissão de mensagens fornecida pelo pacote PVM. Para se obter os melhores resultados, é necessário determinar o tamanho ótimo do *frame* que é distribuído pelo processador pai (*Sparc Station 2*) aos processadores filhos (*Sparc Station SLC*).

8 Conclusão

A implementação do *ray tracing* em ambiente heterogêneo apresentou resultados satisfatórios com relação as demais implementações. Dessa forma, conclui-se que realmente vale investir em implementações heterogêneas, explorando novas arquiteturas, como a da máquina *Zephyr Wavetracer*, procurando adaptar os algoritmos de forma a melhor utilizar as potencialidades das arquiteturas diferentes, a fim de reduzir os custos computacionais existentes na implementação seqüencial. Como perspectiva futura, pretende-se avaliar os ganhos do uso da heterogeneidade combinado com as demais técnicas de aceleração, além da investigação da técnica em outras arquiteturas.

Referências

[AK87] J. Arvo and D. Kirk. Fast ray tracing by ray classification. *Comput. Graph.*, 21(4):55–64, July 1987.

[Ama84] J. Amanatides. Ray tracing with cones. *Comput. Graph.*, 18(3):129–135, July 1984.

[Coo86] R. L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1), January 1986.

[CPC84] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *Comput. Graph.*, 18(3):137–145, July 1984.

[CWBV85] J. G. Clearly, B. M. Wyvill, G.M. Birtwistle, and R. Vatti. Multiprocessor ray tracing. *Comput. Graph. For.*, 5:3–12, 1985.

[DW85] M. Dippe and E. H. Wold. Antialiasing through stochastic sampling. *Comput. Graph.*, 19(3):68–78, July 1985.

[FTI86] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated ray-tracing system. *IEEE Comput. Graph. Appl.*, 6(4):16–26, April 1986.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley Publishing Company, 2nd edition, 1990.

[GBD+93] Al Geist, Adam Beguelim, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vandy Sunderam. *PVM 3.0 User's Guide and Reference Manual.* Oak Ridge National Laboratory, Feb 1993.

[Ger86] M. Gervautz. Three improvements of the ray tracing algorithm for csg trees. *Comput. Graph.*, 10(4):333–339, 1986.

[Gla84] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Comput. Graph. Appl.*, 4(10):15–22, October 1984.

[Gla89] Andrew S. Glassner, editor. *An Introduction to Ray Tracing.* Academic Press Inc., 1989.

[GS85] J. Goldsmith and J. Salmon. A ray tracing system for the hypercube, 1985. Caltech Concurrent Computing Project Memorandum HM154, California Institute of Technology.

[GS87] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl.*, 7(5):14–20, May 1987.

[Han83] P. Hanrahan. Ray tracing algebraic surfaces. *Comput. Graph.*, 17(3):83–89, July 1983.

[HG83] R. A. Hall and D. P. Greenberg. A testbed for realistic image synthesis. *IEEE Comput. Graph. Appl.*, 3(10):10–20, November 1983.

[HG86] E. A. Haines and D. P. Greenberg. The light buffer: a shadow testing accelerator. *IEEE Comput. Graph. Appl.*, 6(9):6–16, September 1986.

[HH84] P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. *Comput. Graph.*, 18(3):119–127, July 1984.

[Hil90] Francis S. Hill. *Computer Graphics.* Macmillan Publishing Company, Department of Electrical and Computer Engineering – University of Massachusetts, 1990.

[Jan86] F. W. Jansen. Data structures for ray tracing. In L.R.A. Kessener, F.J. Peters, and M.L.P. Lierop, editors, *Data Structures for Raster Graphics, Proceedings Workshop*, pages 57–73. Springer Verlag, 1986. Eurographics Seminars.

[Kaj83] J. T. Kajiya. New techniques for ray tracing procedurally defined objects. *Comput. Graph.*, 17(3):91–102, July 1983.

[Kap85] M. R. Kaplan. Space tracing a constant time ray tracer. state of the art in image synthesis. 11, July 1985. Siggraph '85 Course Notes.

[KK86] T. L. Kay and J. Kajiya. Ray tracing complex scenes. *Comput. Graph.*, 20(4):269–278, August 1986.

[KNS87] H. Kobayashi, T. Nakamura, and Y. Shigei. Parallel processing of an object space for image synthesis using ray tracing. *The Visual Computer*, 3(1):13–22, 1987.

[LRU85] M. Lee, A. R. Redner, and S. P. Uselton. Statistically optimized sampling for distributed ray tracing. *Comput. Graph.*, 19(3):61–67, July 1985.

[Max81] N. L. Max. Vectorized procedural models for natural terrain: waves and islands in the sunset. *Comput. Graph.*, 15(3):317–324, August 1981.

[NO86] K. Nemoto and T. Omachi. An adaptative subdivision by sliding boundary surfaces. In *Proc. of Graphics Interface '86*, pages 43–48, Vancouver, B. C., May 1986.

[NOK+83] H. Nishimura, H. Ohno, H. Kawata, T. Shirakawa, and K. Omura. Links-1: a parallel pipelined multimicrocomputer system for imagem creation. In *Proc. of the 10th Symposium on Computer Architecture*, pages 387–394, 1983.

[OM] M. Ohta and M. Maekawa. Ray coherence theorem and constant time ray tracing algorithm. In T. L. Kunni, editor, *Computer Graphics 1987*, pages 303–314. Proc. of CG International '87.

[PB85] D. J. Plunkett and M. J. Bailey. The vectorization of a ray-tracing algorithm for improved execution sppeed. *IEEE Comput. Graph. Appl.*, 5(8):52–60, August 1985.

[Pur86] W. Purgathofer. A statistical method for adaptative stochastic sampling. In A. A. G. Requicha, editor, *Proc. Eurographics '86*, pages 145–152, Elsevier(North–Holland), 1986.

[Rot82] S. D. Roth. Ray casting for modeling solids. *Comput. Graph. Image Process.*, 18:109–144, 1982.

[RW80] S. Rubbin and T. Whitted. A three–dimensional representation for fast rendering of complex scenes. *Comput. Graph.*, 14(3):110–116, July 1980.

[SA84] T. W. Sederberg and D. C. Anderson. Ray tracing of steiner patches. *Comput. Graph.*, 18(3):159–164, July 1984.

[STN87] M. Shinya, T. Takahashi, and S. Naito. Principles and applications of pencil tracing. *Comput. Graph.*, 21(4):45–54, July 1987.

[Ull83] M. K. Ullner. *Parallel machines for computer graphics.* PhD thesis, California Institute of Technology, Pasadena, California, 1983.

[Whi80] T. Whitted. An improved illumination model for shaded display. *Communi. of ACM*, 23(6):343–349, June 1980.