Visualização Volumétrica em um Ambiente de Computação Distribuída

ROBERTO DE BEAUCLAIR SEIXAS[1,2]
MARCELO GATTASS[1]

[1] TeCGraf – Grupo de Tecnologia em Computação Gráfica
Depertamento de Informática, PUC-Rio
Rua Marquês de São Vicente, 225
22453-900 Rio de Janeiro, RJ, Brasil
(tron,gattass)@icad.puc-rio.br

[2] LNCC – Laboratório Nacional de Computação Científica
Rua Lauro Müller, 455
22290-160 Rio de Janeiro, RJ, Brasil
tron@lncc.br

Abstract. The high cost of massive parallel computers and availability of LAN-connected workstations have lead researchers to explore distributed computing using independent workstations. This approach may provide better cost/performance than tightly coupled multiprocessors. In this paper we address the feasibility of such a non-dedicated heteregeneous distributed environment to do volume visualizations using direct volume rendering approach.

Keywords: Volume Visualization, Parallel Volume Rendering, Distributed Environment.

1 Introdução

A Visualização Científica tem sido uma importante área de pesquisa nos últimos anos. A forma de visualização mais utilizada é a Visualização Volumétrica. Esta forma de visualização foi primeiramente apresentada em maio de 1988 por Mark Levoy [Levoy (1988)], como uma técnica de visualização de superfícies a partir de dados volumétricos. Sua idéia era conseguir uma técnica de visualização eficiente e precisa, em que se conseguisse "sintetizar" todas as informações contidas em um conjunto de dados volumétricos em uma única imagem, de forma que se tivesse a impressão de estar olhando para os dados reais.

Posteriormente, esta técnica recebeu o nome de Visualização Volumétrica Direta (*Direct Volume Rendering*), tendo como algoritmo principal o algoritmo de *Ray-Casting*. A grande vantagem desta técnica é a de não necessitar decidir quando uma superfície atravessa uma certa região do volume. Essa é uma tarefa essencial para os algoritmos de detecção de superfícies (*Surface-Fitting*), que tentam aproximar uma superfície através de primitivas geométricas e então usar os algoritmos e técnicas tradicionais de visualização [Elvins (1992)].

No entanto, os algoritmos para visualização volumétrica direta possuem um custo computacional muito alto, pois necessitam percorrer todo o volume de dados, que normalmente são matrizes de 256x256x256 contendo um valor escalar, a cada posição de visualização escolhida de observação. Challinger (1993) determinou que a complexidade é $O(n^3+s^2n)$, onde n é a quantidade de valores em cada dimensão do volume e s é o número de *pixels* em cada dimensão da imagem.

Assim, nos últimos anos foram apresentadas várias técnicas de otimizações para o algoritmo de *Ray-Casting*, e alguns outros algoritmos alternativos [Sabella (1988); Upson e Keeler (1988); Westover (1990); Laur e Hanrahan (1991); Kaufman *et al.* (1994), Zuffo e Lopes (1994)].

Entretanto, mesmo com as técnicas de otimizações existentes, os algoritmos atuais, como por exemplo, o apresentado por Lacroute e Levoy (1994), ainda não conseguem a visualização volumétrica direta em tempo real (*real-time*). O tempo mínimo que se conseguiu é chamado de "tempo para viabilizar interação" (*interactive-time*), indicando que os computadores atuais ainda não são rápidos o bastante para gerar imagens de alta resolução com qualidade [Silva and Kaufman (1995)].

Como os computadores que manipulam volumes ainda estão em desenvolvimento, como por exemplo *IBM Power Visualization System*, a tendência para se fazer uma visualização volumétrica direta rápida, é utilizar os computadores paralelos disponíveis. Alguns inclusive já possuem pacotes de visualização de dados volumétricos [Whitman *et al.* (1994)]. Infelizmente estes computadores além de caros possuem características muito particulares, fazendo com que o

desenvolvimento de algoritmos e técnicas que se aproveitam totalmente do alto desempenho do *hardware*, sejam muito complexos.

Para contornar este problema e permitir uma visualização volumétrica direta interativa, optou-se por utilizar computadores interligados em redes locais (processamento distribuído), fazendo com que cada um funcione com um processador (o termo "processador" é utilizado para designar um computador que atue como um elemento de processamento). A vantagem desta abordagem é que pode-se agregar a este "computador paralelo virtual", vários computadores de fabricantes e arquiteturas diferentes.

Diversos trabalhos apresentados no último *Parallel Rendering Symposium* PRS'95 mostraram que várias destas áreas ainda tem problemas a serem resolvidos.

Este artigo trata dos aspectos envolvidos na implementação do algoritmo de *ray-casting* utilizando processamento distribuído num ambiente heterogêneo e não dedicado. Tais aspectos e alguns conceitos referentes a paralelismo da arquitetura utilizada, são descritos na seção 2. A seção 3 descreve o ambiente computacional utilizado. Os resultados obtidos são apresentados na seção 4 e as principais conclusões do trabalho na seção 5.

2 Ray-Casting Distribuído

A visualização volumétrica utilizando a técnica de *ray-casting* envolve a amostragem de valores escalares em todo o conjunto de dados, convertendo-os em valores de cor e transparência. A contribuição de cada amostragem é acumulada ao longo de raios que passa pelo observador e por cada *pixel* da tela, compondo a imagem final.

As otimizações deste algoritmo são obtidas através do uso de várias técnicas de computação gráfica, tais como terminação antecipada do raio (*early ray termination*), decomposição dos dados em *octree* e amostragem adaptativa dos dados. A terminação antecipada do raio é uma técnica que simplesmente termina o caminhamento pelo raio, ou seja, a acumulação das contribuições de cada *voxel*, após a opacidade acumulada ultrapassar um certo valor (*threshold*). A decomposição em *octree* é um técnica de enumeração hierárquica espacial que permite um rápido caminhamento por espaços vazios, diminuindo substancialmente o comprimento dos raios. A amostragem adaptativa tenta minimizar o trabalho pela detecção de partes homogêneas do volume.

As técnicas descritas acima se baseiam apenas na redução do caminhamento pelo segmento do raio. Seixas *et al* (1995) apresentaram técnicas de otimização para as outras etapas do algoritmo de *ray-casting*: (1) o lançamento dos raios; (2) a detecção do segmento de raio que intercepta o volume, utilizando o algoritmo de Cyrus-Beck; (3) a discretização deste segmento e caminhamento através do algoritmo de Bresenham; e (4) na estratégia de preenchimento da tela. Nesta última, é feito um refinamento progressivo da imagem resultando em melhorias perceptuais nos estágios intermediários de exibição.

Na busca da interatividade e da sofisticação das imagens, acaba surgindo a necessidade de se utilizar soluções explorando estratégias de paralelização. É importante notar que nesta adaptação novas estratégias de otimização são necessárias.

2.1 Paralelismo

Normalmente, se obtêm um maior poder computacional pelo uso de paralelismo. No entanto, o desempenho de algoritmos em paralelo é bastante afetado pela topologia e pelas características da conexão entre os processadores. Estas informações interferem diretamente no comportamento e na estrutura dos algoritmos paralelo, podendo, inclusive, torná-los impraticáveis. De maneira geral, o tempo total necessário para uma visualização volumétrica paralela pode ser descrito por:

$$T_{total} = T_{decomposição} + T_{carga} + T_{processamento} + T_{comunicação} + T_{exibição}$$

Nos últimos anos, o uso de paralelismo em visualização vem crescendo muito rapidamente. Em função disso, realizou-se em 1993 o primeiro *Parallel Rendering Symposium* (PRS'93), mostrando que existem pesquisas para os *hardwares* específicos ou para *cluster*[1] de computadores de uso geral. Neste simpósio, ficou claro que existem vários obstáculos ainda a serem superados em áreas como escalabilidade, balanceamento e composição da imagem.

Assim, antes de apresentarmos os resultados obtidos até o momento, é importante descrever as implicações e nomenclatura envolvida quando se trabalha com paralelismo.

2.2 Arquitetura da Rede Local

Um fator que tem grande impacto no desempenho de computação distribuída é a forma de interconexão dos processadores. Na arquitetura de *clusters* de computadores , a forma utilizada é a de Memória Distribuída (*distributed memory MIMD*), onde cada processador tem sua própria memória, podendo ser acessada por outros processadores. Este acesso a esta memória por outros processadores depende dos tipos diferenciados de acesso providos pela rede a diferentes processadores.

[1] O termo *cluster* designa um grupo de computadores interligados por uma rede local (LAN), para a execução de programas em paralelo.

A rede de comunicação entre os processadores também pode ter um grande impacto no desempenho. O estudo de algoritmos para computadores paralelos dependem fortemente das características da rede de comunicação [Dongarra *et al.* (1989)].

Além disso, as redes locais podem ser heterogêneas, compostas de processadores de várias arquiteturas e/ou desempenho diferentes, e homogêneas, quando compostas de processadores idênticos.

Neste trabalho, optamos por utilizar processadores não dedicados (o que altera muito as medidas de desempenho), em uma rede bastante heterogênea. Acreditamos que esta situação seja a mais comum e que os resultados obtidos permitem um avaliação mais geral.

Para avaliar a heterogeneidade, o algoritmo seqüencial foi executado em cada um dos tipos de processadores testados[2] (Figura 1). Tais valores são bastante importantes, pois como será explicado mais adiante, um único processador lento pode afetar o desempenho final do algoritmo se for feito um balanceamento de tarefas inadequado.

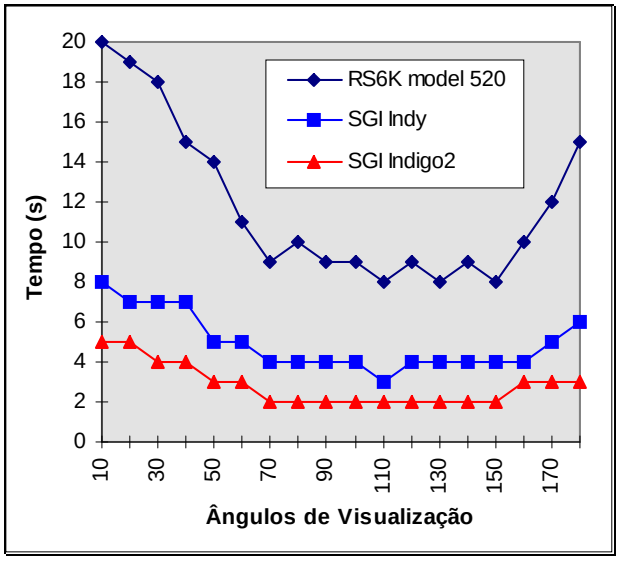

Figura 1: Tempos do algoritmo seqüencial nas diversas arquiteturas utilizadas.

Os tempos ilustrados na Figura 1, confirmam a necessidade de otimizações significativas para que este algoritmo seja interativo.

[2] Os tempos obtidos para os processadores SUN SPARCstation 1+ não constam do gráfico pois foram considerados absurdos. Para ser ter uma idéia, o tempo médio obtido foi de 58,11 segundos, 5 vezes maior que RS6K model 520, 12 vezes maior que SGI Indy e 21 vezes maior que SGI Indigo2. Além disso, nota-se uma variação muito grande nos tempos medidos em função das atividades de rede, tais como NFS, NIS e DNS.

2.3 Indicadores de Desempenho

Quando se projeta um algoritmo para um computador paralelo, deseja-se que ele seja eficiente e utilize toda a potência computacional. Entretanto, a inclusão de mais e mais processadores a um computador paralelo nem sempre implica em uma redução no tempo de execução. A métrica para avaliação de eficiência e potência computacional se dá através da **granulosidade, aceleração** e **eficiência** do algoritmo paralelo em relação ao seqüencial, além da técnica de distribuição dos dados entre os processadores.

Uma vez que o desempenho dos algoritmos paralelos dependem da arquitetura, apenas leva-se em consideração o tempo de decomposição, o tempo de processamento e o tempo para comunicação e sincronização dos processadores, para estimar o desempenho de um algoritmo em uma determinada arquitetura. Os tempos de carga e de exibição são praticamente os mesmos do algoritmo seqüencial, não interferindo na medida de desempenho. Desta forma, assume-se que o algoritmo possui várias tarefas e que cada uma destas pode ser paralelisada, processando-as concorrentemente.

A relação entre o tempo necessário para o processamento desta tarefa (R) e o tempo necessário a comunicação (C) dá-se o nome de **granulosidade** e pode ser usado para determinar o ponto de desempenho ótimo de uma determinada arquitetura:

- se R/C é grande (*coarse-grain*), há pouca comunicação entre os processadores;
- se R/C é pequeno (*fine-grain*), um tempo considerável é gasto em comunicação.

Como regra geral, pode-se dizer que quando a relação é grande, podemos usar mais processadores na expectativa que a comunicação será distribuída entre os processadores. Se a relação é muito pequena, normalmente o número ótimo de processadores é menor.

Outra importante medida de desempenho de um algoritmo paralelo é o seu ***speedup***. Alguns autores definem *speedup* como a relação entre o tempo do melhor algoritmo seqüencial e o tempo do algoritmo paralelo. Outros autores definem como a relação entre o tempo do algoritmo paralelo em um processador e o tempo do algoritmo em n processadores de mesma arquitetura.

A principal dificuldade é que os algoritmos tem seções facilmente paralelizáveis e outras que são inerentemente seqüenciais. Quando um grande número de processadores está disponível, as seções paralelizáveis são rapidamente executadas, mas as seções seqüenciais se tornam gargalos computacionais [Bertsekas (1989)]. Esta observação é conhecida como "Lei de Amdahl" e pode ser quantificada da seguinte

maneira: se um algoritmo consiste de duas seções, uma inerentemente seqüencial e uma totalmente paralelizável, e se a seção seqüencial consome uma fração f do tempo total de processamento, então o seu *speedup* é limitado pela seguinte fórmula:

$$Speedup \leq \frac{1}{f + \frac{(1-f)}{p}} \leq \frac{1}{f}, \forall p$$

onde p é o número de processadores envolvidos.

A **eficiência** de uma paralelização é determinada como uma relação entre a aceleração (*speedup*) obtida e o número de processadores (p) necessários para obtê-la.

2.4 Balanceamento (*load balancing*)

Além das medidas de desempenho discutidos anteriormente, o projeto e a implementação de algoritmos em paralelo dependem da forma de distribuição das tarefas entre os processadores, denominada de balanceamento (*load balancing*).

Nenhum algoritmo paralelo funcionará bem sem um correto balanceamento, pois o desempenho geral ficará comprometido pelos processadores mais lentos e por aqueles que receberem tarefas maiores [Neumann (1994)].

2.5 Técnica de Particionamento

Todos os algoritmos paralelos de visualização volumétrica tem que partilhar dois conjuntos de dados entre os processadores. O primeiro conjunto diz respeito aos dados a serem visualizados, ou seja, neste caso os dados volumétricos propriamente ditos. O outro conjunto refere-se aos dados da imagem, ou seja, localização (x, y) e cor (r, g, b). Assim, os algoritmos paralelos trabalham baseando-se em um dos dois conjuntos. Esta distinção de estratégias cria duas classes de algoritmos paralelos [Neumann (1994)]: particionamento da imagem e particionamento do dados.

Para obter um balanceamento mais uniforme, utilizamos o particionamento da imagem onde os processadores dividem entre si os *pixels* da tela para gerar a imagem completa. Esta divisão pode ser estática ou dinâmica.

Existem várias formas para dividir as tarefas quando se usa esta forma de particionamento: *pixels*, *scanlines* e em blocos [Lacroute (1995)]. O particionamento por *pixels* geraria uma alta comunicação e o particionamento por blocos não permitiria usar a coerência dos dados eficientemente. Assim, o tipo de particionamento usado foi o de *scanlines*, onde podemos pré-calcular os valores de incremento para cada *scanline*, pela coerência dos dados.

Dado que a tarefa básica é calcular um grupo contíguo de *scanlines* de imagens intermediárias, temos que escolher uma forma de atribuir grupos de *scanlines* aos vários processadores, tendo o cuidado de maximizar o balanceamento. Esta atribuição pode ser feita de três formas:

- estática contígua – a imagem é dividida em blocos contíguos de *scanlines*, que são atribuídos a cada processador;
- estática intercalada – a imagem é dividida em *scanlines*, cada uma é atribuída a um processador, de forma intercalada;
- dinâmica – a imagem é dividida em *scanlines*, sendo atribuída a cada processador, de acordo com as suas disponibilidades de processamento, durante o tempo de visualização.

O método escolhido foi um híbrido de particionamento estático intercalado e o dinâmico. Este método resulta num melhor desempenho, mas é necessário que se tenha cuidado na distribuição das *scanlines*. O método utilizado baseia-se na distribuição estática intercalada até que todos os processadores recebam uma *scanline*. A partir deste ponto inicia-se o particionamento dinâmico, fazendo com que, conforme for terminando o processamento, cada processador disponível receba uma nova *scanline*.

A principal vantagem desta técnica é que se consegue minimizar a influência de processadores lentos no desempenho final, fazendo com que os mais rápidos processem mais *scanlines*.

Outro fato importante, é que este método se adapta ao número de processadores existentes, ou seja, se o número de processadores for maior ou igual que o de *scanlines*, somente o particionamento estático intercalado será utilizado.

3 Descrição do Ambiente

O PVM – *Parallel Virtual Machine*, é um ambiente de desenvolvimento e execução de aplicações em paralelo que envolve componentes ou tarefas, relativamente independentes, que interagem entre si. O PVM se propõe de atuar sobre um grupo heterogêneo de computadores interconectados por uma ou mais redes locais. É composto por uma biblioteca de rotinas ou primitivas que podem ser incorporadas às linguagens de programação existentes. Essas rotinas admitem entre outras facilidades, a inicialização e o término de tarefas dentro da rede, a comunicação e a sincronização entre elas.

Cada computador pode endereçar somente sua memória e toda comunicação é efetuada através de troca de mensagens. Assim, para se efetuar uma troca de dados, devemos acessá-los na memória de um

processador providenciando informações de "como" e "para onde" serão enviados. Só então os dados são fisicamente transmitidos e postos na memória de computador receptor. O PVM implementa a parte de comunicação de dados, através do controle de envio destas mensagens entre processadores. Ligado a este fato, temos dois fatores intrínsecos de degradação da eficiência da aplicação: o tempo de comunicação, que é o tempo necessário para a troca de informações; e o tempo de sincronização, que é o tempo de espera para que certas partes do algoritmo sejam completadas a fim de liberar a continuidade da execução. A comunicação e a sincronização entre os processadores é controlada pelo próprio PVM, garantindo a confiabilidade na entrega das mensagens. Ainda como vantagem, fazendo uso do PVM, garante-se a portabilidade do programa para diversas plataformas.

Uma vez determinado o uso de um ambiente heterogêneo distribuído, composto de 11 estações de trabalho, 8 Silicon Graphics modelo *Indy*, 1 Silicon Graphics modelo *Indigo2*, e 2 IBM RS/6000 modelo 520, determinou-se um protocolo a ser implementado através de *message passing*, permitindo a comunicação entre os processos.

Master	**Mensagens**	***Slave***
TaskIDs, NumProc	init →	*wait*
wait	init_ok ←	HostName, MyTaskID
do forever		
RotAng[3]	user_view →	*wait*
while image not completed		
ScanLine-Number	sl_param →	*wait*
wait	sl_result ←	MyID, Results, sl_num

Figura 2: Protocolo de troca de mensagens.

As tarefas de cada processador podem facilmente ser identificadas da seguinte maneira: as partes externas aos *loops* são partes seqüenciais do algoritmo e as internas são as partes paralelas (Figura 2). A parte seqüencial é denominada de *master* e é responsável pelo particionamento e distribuição da imagem. A parte paralela é denominada *slave* e é responsável pelo processamento da imagem (Figura 3).

No entanto, a visualização dos dados fica sem uma definição precisa do seu comportamento. Se por um lado, a visualização da imagem pode ser feita em paralelo, uma vez que a ordem de exibição das *scanlines* não altera a imagem. Entretanto, não se pode iniciar um novo ângulo de visualização sem que todas as *scanlines* anteriores já tenham sido processadas.

No método implementado inicialmente, cada *scanline* calculada é imediatamente exibida. Infelizmente, verificou-se que esta forma de implementação não é eficiente pois o processador que acabou de calcular uma *scanline* tem que esperar que esta seja exibida, antes de receber uma nova para calcular, pois a tarefa de visualização está presente no *master*. Assim, se o processo de visualização dos dados for lenta, todo o mecanismo de paralelização estará prejudicado.

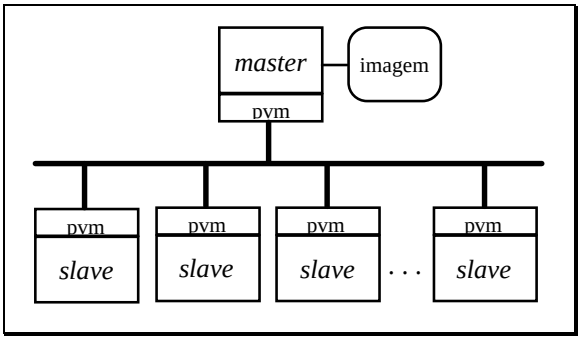

Figura 3: Ambiente de desenvolvimento.

A solução utilizada foi o de se separar as tarefas de particionamento e visualização em processadores diferentes (vide Figura 7). Assim, o tempo necessário à exibição de cada *scanline* não interfere no particionamento e submissão de uma nova *scanline* para o processador que está disponível.

Uma outra vantagem obtida desta separação é o processador responsável pela visualização pode executar um método de *preview* da imagem, de forma sequencial, enquanto a imagem com alta qualidade está sendo calculada pelos demais processadores, aumentando a interatividade com o usuário.

4 Resultados Obtidos

Os resultados mostrados a seguir ilustram as medidas de desempenho descritas anteriormente. Tais resultados são a média das tomadas de tempo escolhendo-se os processadores mais disponíveis em cada execução, uma vez que o ambiente utilizado é heterogêneo e não dedicado.

O algoritmo de *ray-casting* implementado foi derivado do apresentado em Seixas *et al* (1995), que inclui as otimizações do lançamento do raio, na detecção da interseção com o volume de dados e no caminhamento. A estratégia de preenchimento de tela por refinamentos progressivos não foi utilizada.

Com o intuito de se obter uma melhor avaliação comparativa, foram realizados os dois tipos de medição do *speedup* (descrito na seção 2.2), tanto em relação ao algoritmo seqüencial, quanto ao algoritmo paralelo em

um único processador, como ilustra a Figura 4.

Podemos observar que a aceleração foi aumentando conforme se agregavam mais processadores, atingindo um valor máximo com nove processadores (Figura 4). A partir deste ponto, verifica-se que o aceleração diminui conforme se agregam mais processadores mostrando que o tempo de comunicação e sincronização começou a crescer, afetando o desempenho final.

Note-se que a rede local utilizada no teste possui 9 computadores mais velozes que, certamente, influenciaram este resultado.

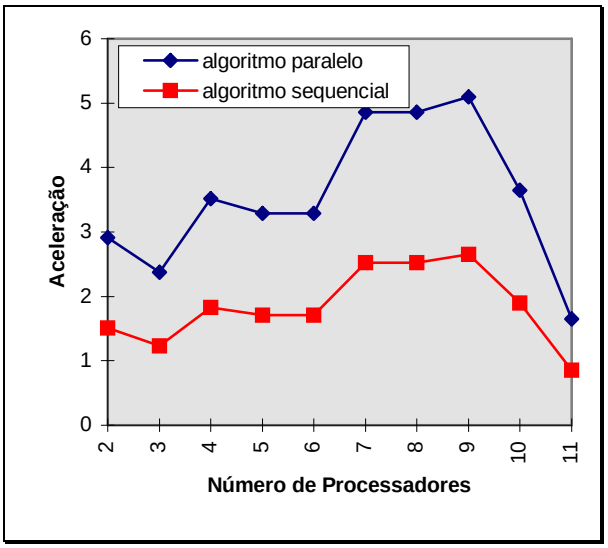

Figura 4: Medidas de Aceleração

As eficiências obtidas, descritas na seção 2.3, são ilustradas pela Figura 5.

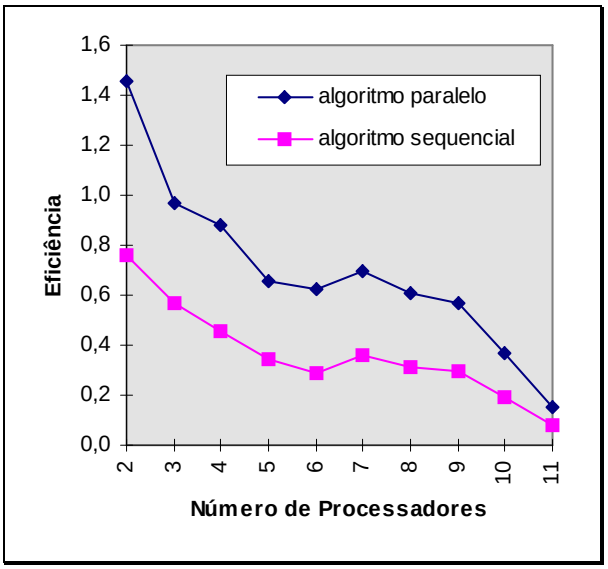

Figura 5: Medidas de Eficiência.

Os tempos finais de processamento para a geração de uma imagem de 200x200 com um conjunto de dados de 128x128x84, é mostrado na Figura 6 em função do número de processdores.

5 Conclusões

Como resultado final do trabalho, obtivemos um sistema interativo de visualização volumétrica, utilizando processamento distribuído e dotado de mecanismo de implementação flexível, que permite a avaliação e testes de novos métodos de particionamento, visualização e a inclusão de otimizações na técnica de visualização volumétrica direta.

Com os resultados obtidos ilustrados na seção 4, pode-se concluir que apesar do *speedup* máximo obtido de 5,2, o tempo final obtido ainda é de aproximadamente 1 *frame* por segundo, o que não é suficiente para causar uma noção de movimento. No entanto, mostrou-se que com a utilização de computadores conectados em rede local, mesmo que partilhados com outros usuários, consegue-se obter *speedups* significativos.

Mais ainda, o mecanismo implementado é totalmente compatível com as propostas de melhorias das técnicas de otimização empregadas até o momento, em todas as etapas do processo da visualização volumétrica.

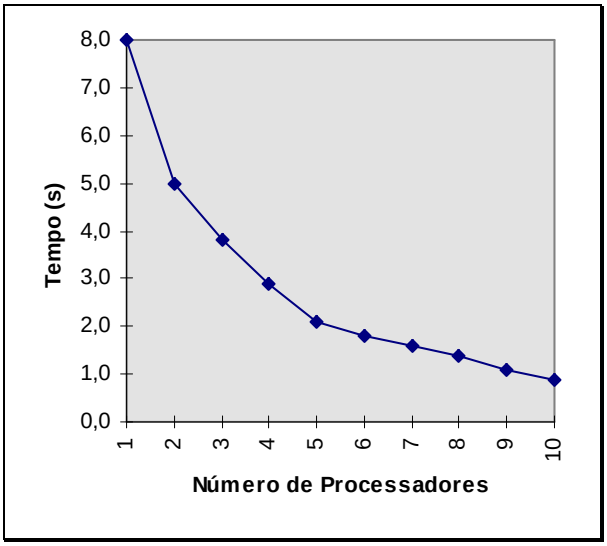

Figura 6: Tempos obtidos.

Nos testes apresentados, algumas otimizações foram incluidas, mas outras ainda podem ser acrescentadas. Para o cálculo das *scanlines*, por exemplo, é interessante o uso de *octrees* ou árvores *min–max*, com o objetivo de minimizar o tempo de caminhamento ou até evitá-lo. Para melhorar a percepção das imagens em construção, pode-se

identificar o método mais adequado de refinamento, progressivo ou adaptativo.

Outra conclusão importante foi a modificação do ambiente para permitir uma independência da tarefa de particionamento da tarefa de visualização (Figura 7). Isto é importante pois se criou um ambiente no qual técnicas de particionamento podem ser testadas, independentemente das técnicas de visualização ou das de cálculo das *scanlines*, e vice-versa.

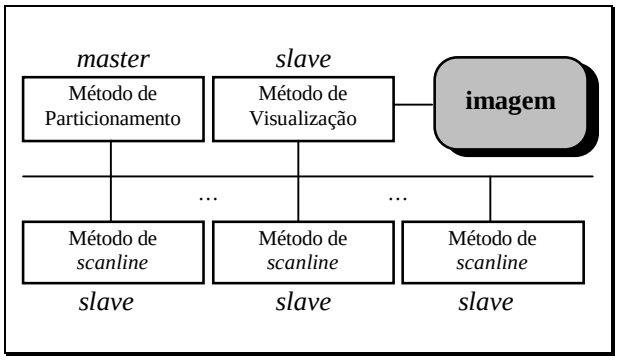

Figura 7: Ambiente final utilizado.

Desta forma, pode-se utilizar num mesmo ambiente, variações de técnicas sem que a haja a necessidade de alteração da sua arquitetura, desde que o protocolo seja mantido. Assim, o resultado das *scanlines* são enviados para o processador responsável pela visualização, determinado durante a inicialização dos processadores *slaves*. Para uma visualização diferente, bastaria informar o novo processadores responsável pela visualização, através de parâmetros globais transmitidos entre os processadores pelo protocolo de mensagens. Para um novo método de *scanline*, basta que o processador responsável pelo particionamento seja informado de quais os processadores que devem ser utilizados na distribuição das tarefas.

A Visualização Volumétrica Direta é uma área que requer experimentação. Para permitir esta experimentação, as aplicações têm que prover ferramentas flexíveis, tais como a definição das funções de mapeamento de cor e opacidade. Quanto mais controles o usuário tiver e mais rápida for a resposta a estes controles, mais útil será a aplicação e maior será a sua gama de utilização em problemas que exijam este tipo de visualização.

Agradecimentos

A Luiz Fernando Martha, pelas proveitosas discussões e contribuições durante a execução deste trabalho.

A motivação para este estudo teve origem no âmbito do convênio CENPES/Petrobrás e teve suporte parcial do projeto temático do CNPq GEOTEC - Geo-processamento: Sistemas e Técnicas.

5. Bibliografia

D. P. Bertsekas, J. N. Tsitsiklis, "Parallel and Distributed Computation", Prentice-Hall International Editions, 1989.

J. A. Challinger, "Scalable Parallel Direct Volume Rendering for Nonrectilinear Computational Grids", Ph.D. Thesis, *University of California at Santa Cruz*, 1993.

J. Dongarra, P. Messina, D. C. Sorensen, R. G. Voigt, "Parallel Processing for Scientific Computing", Proceedings of the Fourth SIAM Conference, December, 1989.

R. A. Drebin, L. Carpenter, P. Hanrahan, "Volume Rendering", Proceedings of SIGGRAPH'88, *Computer Graphics* 22 (1988), pp. 65–74.

T. Elvins, "A Survey of Algorithms for Volume Visualization", Course Notes 1, *SIGGRAPH'92*, 1992, pp. 3.1–3.14.

C. Giertsen, J. Petersen, "Parallel Volume Rendering on a Network of Workstations", *IEEE Computer Graphics and Applications*, November, 1993, pp. 16–23.

A. E. Kaufman, W. Lorensen, R. Yagel, "Volume Visualization: Algorithms and Applications", Course Notes 1, *Visualization'94*, 1994.

P. G. Lacroute, M. Levoy, "Fast Volume Rendering Using a Shear–Warp Factorization of the Viewing Transformation", Proceedings of SIGGRAPH'94, 1994, pp. 451–458.

P. G. Lacroute, "Fast Volume Rendering Using a Shear–Warp Factorization of the Viewing Transformation", Technical Report, *Stanford Univeristy*, September, 1995.

D. Laur, P Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering", Proceedings of SIGGRAPH'91, *Computer Graphics* 25 (), 1991, pp. 285–288.

M. Levoy, "Display of Surface from Volume Data", *IEEE Computer Graphics and Applications* 8 (1988), pp. 29–37.

M. Levoy, "Efficient Ray Tracing of Volume Data", *ACM Transactions on Graphics* 9 (1990), pp. 245–261.

M. Levoy, "A Taxonomy of Volume Visualization Algorithms", Course Notes 11, *SIGGRAPH'90*, 1990, pp. 6–12.

U. Neumann, "Communication Costs for Parallel Volume Rendering Algorithms", *IEEE Computer Graphics and Applications* 4 (1994), pp. 49–58.

P. Sabella, "A Rendering Algorithm for Visualizing 3D Scalar Fields", Proceedings of SIGGRAPH'88, *Computer Graphics* 18 (1988), pp. 51–58.

R. de B. Seixas, M. Gattass, L. H. de Figueiredo, L. F. Martha, "Otimização do Algoritmo de Ray Casting para Visualização de Tomografias", *Caderno de Comunicações do VII SIBGRAPI* (1994), pp. 5–8.

R. de B. Seixas, M. Gattass, L. H. de Figueiredo, L. F. Martha, “Visualização Volumétrica com Otimizações de Ray Casting e Detecção de Bordas”, *Anais do VIII SIBGRAPI* (1995), pp. 281–286.

C. T. Silva, A. E. Kaufman, “Parallel Performance Measures for Volume Ray Casting”, Report, *State University of New York at Stony Brook*, 1994.

C. T. Silva, F. M. Lok, A. E. Kaufman, “Interactive Parallel Volume Rendering Using PVR System”, Technical Report, *State University of New York at Stony Brook*, 1995.

L. Sobierajski, A. Kaufman, “Volumetric Ray Tracing”, *Proceedings of Visualization’94*, 1994, pp. 11–18.

C. Upson, M. Keeler, “V-Buffer: Visible Volume Rendering”, Proceedings of SIGGRAPH’88, *Computer Graphics* **22** (1988), pp. 59–64.

L. Westover, “Footprint Evaluation for Volume Rendering”, *Computer Graphics* **24** (1990), pp. 367–376.

S. Whitman, C. D. Hansen, T. W. Crockett, “Recent Development in Parallel Rendering”, *IEEE Computer Graphics and Applications* 14 (1994), pp. 21–22.

R. Yagel, R. Machiraju, “Data-Parallel Volume Rendering Algorithms”, The Ohio State University, 1994.

M. K. Zuffo, E. T. Santos, R. de D. Lopes, C. A. P. S. Martins, “Visualização de Alto Desempenho”, *Anais do Workshop sobre Computação de Alto Desempenho para Processamento de Sinais*, 1993, pp. 142–161.

M. K. Zuffo, R. Lopes, “A High Performance Direct Volume Rendering Pipeline”, *Anais de VII SIBGRAPI*, 1994, pp. 241–248.