Paralelização do Módulo de Ray-tracing na Ferramenta POV-Ray

Bruno Gonçalves de Jesus[1], Douglas D. Fernandez das Neves, Denise Stringhini
Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática
{00cpxxx, dougneves}@gmail.com
dstring@mackenzie.br

Resumo

O principal objetivo dessa pesquisa foi a análise de estratégias de paralelização para o módulo que implementa o ray-tracing *na ferramenta POV-Ray, que possui código aberto. Com isso, pretendeu-se a elaboração de novos algoritmos que unam o poder da computação paralela de um* cluster *à alta necessidade de processamento gerada pela renderização de imagens na computação gráfica, diminuindo o tempo final do processo. Este artigo apresenta as limitações impostas pela ferramenta POV-Ray ao paralelismo, o método de paralelização empregado e os testes preliminares de desempenho num* cluster *de PCs com sistema operacional Linux.*

1. Introdução

A síntese de imagens na computação é um processo demorado e que exige considerável poder de processamento da máquina utilizada. Cada vez mais os algoritmos de *renderização* estão se aperfeiçoando e hoje é possível criar imagens com alto grau de realismo, de forma que é cada vez mais difícil diferenciar uma imagem feita no computador de uma foto. Mas isso tem conseqüências, e mesmo nas máquinas mais modernas o processamento de uma única imagem pode levar até dias, dependendo da complexidade da imagem.

Renderizar uma imagem significa converter modelos tridimensionais em imagens bidimensionais que possam ser visualizadas e impressas, efetuando transformações geométricas, projeções, mapeamento de texturas, iluminação, efeitos especiais e *rasterização*. Isso se faz a partir da geometria da cena, das informações sobre os materiais de que são feitos os objetos (cores, texturas e transparências), das condições de iluminação ambiente e da posição de observação da cena (denominada câmera virtual ou observador).

Atualmente, o *ray-tracing* é uma das mais populares técnicas de síntese de imagens e possui simples implementação. Ele pode usar a representação de cenas complexas com muitos objetos e muitos efeitos diferentes. O princípio do *ray-tracing* é simular a geometria ótica envolvida no trajeto de feixes de luz que viajam pelo espaço da cena [1, 9]. O POV-Ray [8], objeto de estudo deste trabalho, é uma ferramenta que implementa esta técnica de síntese de imagens.

Esses algoritmos de computação gráfica necessitam de grande capacidade de processamento computacional. Devido à complexidade das cenas e objetos, às vezes, são necessários muitas horas – até mesmo vários dias - para que uma boa *renderização* seja computada.

O objetivo geral deste trabalho é estudar a utilização de *clusters* de computadores para aumentar o poder de processamento de modo eficaz e de baixo custo em aplicações de Computação Gráfica.

Mais especificamente, entretanto, o presente artigo faz parte de um estudo preliminar sobre a paralelização de módulos da ferramenta POV-Ray. O módulo de r*ay-tracing* foi o escolhido para um primeiro experimento por ser um algoritmo cuja paralelização já é objeto de estudo há algum tempo [2, 3]. Além disso, pretende-se estudar a paralelização de métodos mais modernos, como o de *photon-mapping*, por exemplo [4].

[1] Participante do PIVIC/Mackenzie (Programa Institucional Voluntário de Iniciação Científica)

Os testes de desempenho realizados mostraram o potencial dos algoritmos podem ficar mais rápidos quando executados em paralelo.

2. Ambiente POV-Ray

O POV-Ray, objeto de estudo deste trabalho, é uma ferramenta que implementa diversas técnicas de síntese de imagens. Nos sistemas Unix e suas variações não existe um ambiente gráfico oficial para a ferramenta, mas sim diversos programas *opensource*. De fato, o POV-Ray não precisa de nenhuma GUI (*Graphical User Interface*) ou qualquer outra parte gráfica para ser executado. Diversos argumentos ligados ao executável do programa resolvem esse problema e a edição de códigos pode ser feita em qualquer editor de texto comum respeitando-se as regras da linguagem de modelagem do POV-Ray.

Há mais de uma maneira de utilizar o programa para *renderizar* uma imagem. Variações no tempo de *renderização* são notáveis de acordo com a escolha da qualidade desejada. Uma imagem bem elaborada com dezenas de objetos, reflexos e pontos de luz *renderizada* com uma grande dimensão e *anti-alias* pode levar horas ou minutos dependendo da qualidade desejada.

Para demonstrar as etapas do processo realizadas pelo POV-Ray para gerar uma *renderização* temos o modelo da figura 1 que demonstra o processo desde seu início com a construção da cena até o final, com a saída na tela, em arquivo ou em ambos. As etapas onde são aplicadas as técnicas de *radiosidade* e *photon mapping* servem para dar mais realismo à cena, mas não fazem parte do escopo deste trabalho.

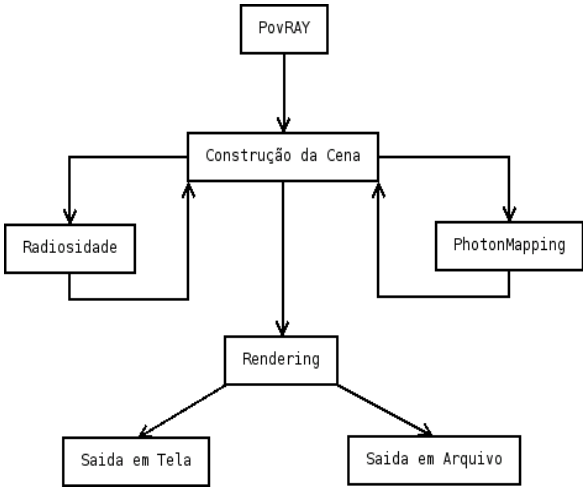

Figura 1. Estrutura do POV-Ray

3. Paralelização do módulo de *rendering*

O objetivo deste trabalho é alterar a forma como é feita essa *renderização*, modificando o algoritmo do programa de forma que distribua partes da *renderização* em diversas máquinas de um *cluster*, criando uma nova forma do modelo a qual é representada na figura 2.

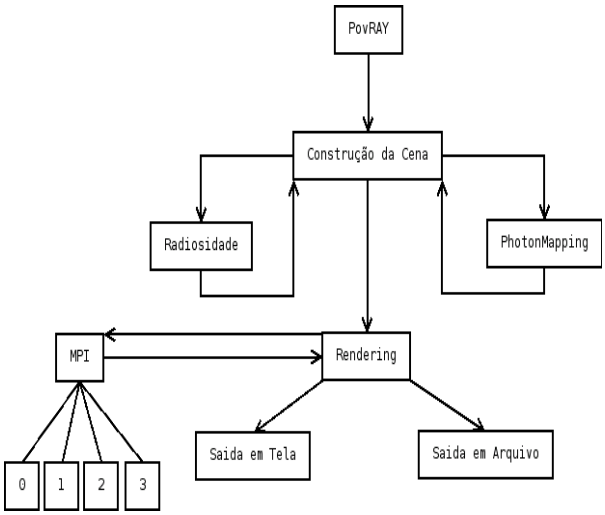

Figura 2. POV-Ray com paralelização

A biblioteca escolhida para implementação foi a MPI [6, 7] que, além de portável, pode ser integrada à linguagem C/C++ (que é a linguagem na qual o POV-Ray foi escrito). A versão do MPI adotada é a LAM-MPI. LAM (*Local Area Multicomputer*) [5] é uma implementação aberta do padrão MPI que foi desenvolvida há mais de uma década.

Durante a fase inicial do trabalho foi realizado um estudo sobre as técnicas de paralelização de algoritmos existentes elegendo a melhor entre essas para construção de algoritmos executados em *clusters*.

Em um estudo teórico sobre como paralelizar a *renderização* de uma imagem, analisou-se os seguintes métodos:

- Particionamento por tiras (*strip partitioning*): a partir do número de computadores, dividem-se as tarefas de *renderização* em linhas horizontais ou verticais.
- Grade Estática (*checkerboarding*) a tela é dividida em uma grade baseada no número de computadores.

- Grade Dinâmica: cada computador recebe uma pequena área da tela de tamanho proporcional à complexidade da *renderização* ou à sua capacidade de processamento.

Estas técnicas não se mostraram adequadas no caso da *renderização* numa análise preliminar. No primeiro caso, por exemplo, uma imagem que tem a metade superior com poucos ou nenhum objeto e a metade inferior repleta de objetos não seria uniformemente *renderizada*, já que os nós aos quais fossem atribuídas as primeiras tiras terminariam mais rapidamente e praticamente não teriam trabalho, ao contrário dos nós aos quais fossem atribuídas as faixas da metade inferior. Dessa forma, foi descartada a hipótese de se paralelizar com linhas horizontais e verticais.

A grade estática também foi descartada, pois seria necessária a criação de um *pool* de tarefas. Um *pool* de tarefas consiste em haver uma máquina especial que serve tarefas para as demais máquinas do *cluster*. Neste caso específico, o *pool* iria enviar os blocos que cada nó deveria *renderizar*, causando sobrecarga na rede e atrasando o processo de envio de dados.

O terceiro método em especial (grade dinâmica) foi desconsiderado, pois é impossível medir com antecedência a complexidade de uma cena sem disparar raios contra ela. Não é suficiente apenas saber onde os objetos estão posicionados.

Como explicado acima, nenhuma das hipóteses iniciais para a paralelização seria eficiente. Foi então que um estudo mais aprofundado do POV-Ray indicou uma nova direção. Verificou-se que cada raio trabalha independentemente dos outros, seja este ao lado, acima ou abaixo. Partindo dessa descoberta um método alternativo foi adotado. Ele baseia-se no fato de que uma imagem complexa contém *pixels* complexos próximos de forma que se cada máquina calculasse um *pixel* e saltasse um número de *pixels* igual ao número de máquinas no *cluster*, todas as máquinas calculariam *pixels* de todas as "complexidades".

Por exemplo, assumindo que o primeiro *pixel* seria o *pixel 0*, este seria calculado pelo nó mestre já que o mestre normalmente tem o *rank* 0 em aplicações MPI. Com isso tem-se que cada nó deveria calcular como *pixel* inicial aquele correspondente ao seu próprio *rank* no *cluster* e os demais resultando da soma do número de nós do *cluster*.

4. Resultados

Os testes de execução com os resultados mais significativos estão descritos abaixo para 1, 2 e 4 máquinas. Num primeiro momento, os testes foram realizados num *cluster* homogêneo de configuração modesta[2] (4 máquinas Celeron, com 96Mb de RAM cada uma, conectadas por uma rede Ethernet). A distribuição Linux montada no laboratório é baseada no Debian 3.0. Novos testes serão realizados nos dois *clusters* localizados em outros laboratórios da Universidade em breve.

O parâmetro de qualidade da imagem é dada pelo parâmetro **Q** do POV-Ray e deve estar entre 0 e 9. De 0 a 9 as opções vão melhorando a qualidade da imagem adicionando cada vez mais recursos para a renderização (sombras, reflexos, efeitos especiais).

A figura 3 apresenta um gráfico de desempenho para o teste realizado. Nota-se que o aumento de desempenho com os quatro nós subiu 2,08 vezes. O gráfico exibe a relação do tempo e qualidade das imagens.

Um dos problemas encontrados e que afetou diretamente o desempenho do algoritmo foi o modo de armazenamento temporário da *renderização* feita pelo POV-Ray. Ao contrário do que se pensava, o POV-Ray não guarda uma matriz inteira de *pixels* com as cores de cada um, mas sim a linha atual que ele está *renderizando*. Assim que ele termina a linha, guarda no arquivo e perde essas informações passando para a próxima linha. Essa função é um ponto de sincronização entre os processos distribuídos, pois não se pode escrever dados fora de ordem (nem na tela e nem no arquivo).

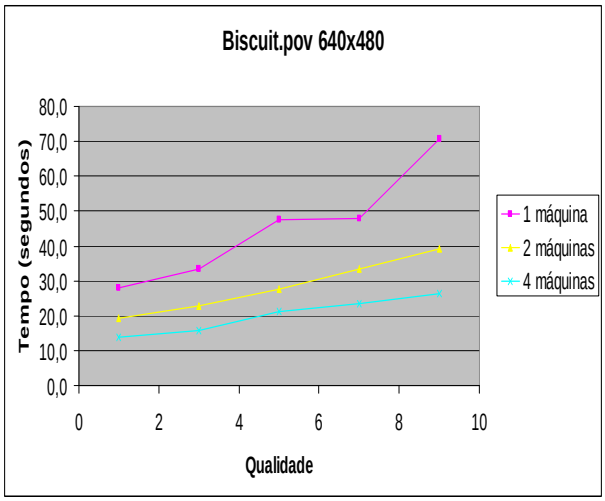

Figura 3. Gráfico de desempenho.

[2] Este *cluster*, assim como o laboratório experimental onde está inserido, possui sua manutenção realizada exclusivamente por alunos de graduação, estagiários da Universidade Mackenzie.

5. Conclusões

Concluindo a pesquisa, os testes realizados mostraram que é possível aumentar o desempenho da *renderização* pelo método de *ray-tracing* gerando imagens mais rapidamente que o comum no POV-Ray.

Como projeto futuro, um método assíncrono deve ser implementado utilizando algum tipo de estrutura como uma lista ou mesmo uma matriz de dados para que não seja necessário aguardar a conclusão linha-a-linha do processo aumentando o tempo final da *renderização*.

A implementação do método de *anti-aliasing* paralelo poderia ser proposta de forma que ao fim do processo completo (guardando a informação de todos os *pixels* da imagem) uma sincronização fosse feita com todas as máquinas de forma que cada uma tendo a imagem completa pudesse aplicar o *anti-alias* e devolver o resultado para o nó mestre dividindo o processo em duas etapas (*renderização* e aplicação do *anti-alias*).

Referências

[1] Azevedo, Eduardo; Conci, Aura. Computação Gráfica: Teoria e Prática. Rio de Janeiro: Elsevier, 2003.

[2] Badouel, D.; Priol, T. Distributing Data and Control for Ray-tracing in Parallel. IEEE Computer Graphics & Applications, Vol. 14, Num. 4, 1994 (jul), pp. 69-77

[3] Green, Stuart A., Parallel Processing for Computer Graphics, MIT Press/Pitman Publishing, Cambridge, Mass./London, 1991.

[4] Jensen, Henrik W. Realistic Image synthesis using photon mapping. Stanford University, 2001.

[5] LAM-MPI. Disponível em: http://www.lam-mpi.org/ (acesso em maio, 2006)

[6] MPI Fórum. Disponível em http://www.mpi-forum.org/ (acesso em maio, 2006)

[7] Pacheco, Peter S. Parallel programming with MPI. California, Morgan Kauffmann Publishers, 1997.

[8] POV-Ray. Disponível em http://www.povray.org/ (acesso em maio, 2006)

[9] Watt, A. e Watt, M. Advanced Animations and Rendering Techniques: Theory and Practice, ACM Press, New York, NY, 1992.