Revista de Ciências Exatas e Tecnologia
Vol. III, Nº. 3, Ano 2008

IMPLEMENTAÇÃO E AVALIAÇÃO DE DESEMPENHO DE UM CLUSTER BASEADO EM SOFTWARE LIVRE PARA RENDERIZAÇÃO DISTRIBUÍDA EM REDE LOCAL

Eberval Oliveira Castro
Faculdade Anhanguera de Jundiaí
eberval.oliveira@unianhanguera.edu.br

Rodrigo Amaral Rocha
Faculdade Anhanguera de Jundiaí
prof.rodrigorocha@yahoo.com

RESUMO

A exemplo de algumas produções cinematográficas, animações e jogos em ambientes tridimensionais virtuais, problemas que exigem grande quantidade de processamento sejam pela complexidade das operações envolvidas ou pela quantidade de dados a ser tratada podem, geralmente, ser resolvidos utilizando-se a estratégia de processamento distribuído. O uso desta técnica torna-se fundamental quando não se dispõe de unidades computacionais com capacidade de processamento individual suficiente para se realizar produtivamente as tarefas em questão. A implantação e avaliação de desempenho de um sistema para renderização distribuída sobre uma rede local baseada em *software* livre (código aberto) são apresentadas neste trabalho. O modelador 3-D utilizado é o Blender 2.46 e a renderização distribuída é realizada pelo DrQueue 0.60.0 ambos licenciados sobre a GPL (*General Public License*). Fatores de *speedup* aproximadamente lineares foram encontrados demonstrando o bom desempenho do sistema implementado.

Palavras-Chave: Computação gráfica, renderização, processamento distribuído, *software* livre.

ABSTRACT

As some cinematographic productions, animations and games in virtual tridimensional environments, issues that require a great quantity of processing because of the complexity of the involved operations or because of the large number of data to be treated, these could generally be solved using the strategy of distributed processing. The use of this technique becomes fundamental when the processing units don't offer sufficient computing resources to accomplish the tasks mentioned. The implementation and evaluation of the performance of a system used for the distributed rendering over a local network based in a free open source code presented in this work. The 3D modeler used was Blender 2.46 the distributed rendering was done using DrQueue 0.60.0 both licensed under the GPL (General Public License). Linear speedup factors were found demonstrating a good performance of the implemented system.

Keywords: Graphical computing, rendering, distributed processing, free software.

Anhanguera Educacional S.A.

Correspondência/Contato
Alameda Maria Tereza, 2000
Valinhos, São Paulo
CEP. 13.278-181
rc.ipade@unianhanguera.edu.br

Coordenação
Instituto de Pesquisas Aplicadas e Desenvolvimento Educacional - IPADE

Informe Técnico
Recebido em: 15/7/2008
Avaliado em: 27/10/2008

Publicação: 8 de dezembro de 2008

1. INTRODUÇÃO

A criação de imagens em cenas tridimensionais com uso de computador demanda duas tarefas principais, a saber: a modelagem e a renderização. A primeira consiste em descrever o objeto do ponto de vista geométrico, determinando como será a sua forma, tamanho, cor, em fim, sua aparência. A renderização, por sua vez, consiste em produzir os pontos de cor em cada quadro partindo dos modelos dos objetos. Em outras palavras, o processo de renderização é responsável por gerar as imagens propriamente ditas, recebendo como entrada os modelos dos objetos, posições de câmera entre outras informações (MCREYNOLDS; BLYTHE, 2005).

Particularmente, a renderização de imagens com efeitos de partículas, iluminação e refletividade, assim como muitas outras questões na área de computação gráfica, necessitam de grande capacidade computacional na sua solução. Estúdios cinematográficos têm lançado mão de técnicas de processamento distribuído através de *render farms* (*clusters* dedicados à computação gráfica) na produção de diversos filmes, em especial, as animações gráficas como a série "Shrek", "Mundo de Inseto" e "Elephants Dream" (vide quadro da Figura 1) ou mesmo filmes com muitos efeitos especiais por computador como é o caso de "Homem Aranha" e "Transformers". Para a renderização de seus quadros, obras como as citadas necessitaram de diversas horas ou mesmo dias de processamento.

Figura 1. Quadro do filme Elephants Dream criado em Blender 3D.

Na indústria de publicidade há demanda por soluções computacionais igualmente sofisticadas uma vez que também fazem uso de processamento massivo na produção de animações tridimensionais em propagandas comerciais.

Outro ramo da indústria que se beneficia do processamento gráfico distribuído é o de criação de jogos 3D. Nos últimos anos o número de lançamentos de títulos que fazem uso de recursos gráficos tridimensionais tem aumentado significativamente.

Diversas universidades no exterior como Berckley e Princeton, e no Brasil, como USP e Unicamp mantêm *clusters* dedicados à realização de computação massiva em suas diversas linhas de pesquisa, o que inclui modelagem de semicondutores, dinâmica biomolecular, dentre outras.

Este trabalho discute a implantação e avaliação de desempenho de um sistema para renderização distribuída sobre uma rede local usando soluções de código aberto (*software* livre). O modelador e animador 3D utilizado é o Blender 2.46, que é distribuído segundo a licença GPL (*General Public License*) (BLENDER, 2008). O gerenciador de renderização distribuída utilizado é o DrQueue 0.60.0, que também é licenciado sob a GPL (DRQUEUE, 2008).

Embora sejam *softwares* livres e não forneçam qualquer tipo de garantia, um amplo suporte pode ser obtido através de fóruns e comunidades na Internet. A contratação de suporte técnico também é possível por meio de empresas especializadas.

Na próxima seção deste artigo será apresentada uma descrição geral do Blender. A terceira seção discorre sobre o DrQueue, suas vantagens e desvantagens. A quarta seção apresenta a estrutura do *render farm*. Na quinta seção é analisado o desempenho do sistema através de métricas consagradas. Na última seção são apresentadas as discussões dos resultados e conclusões finais do trabalho.

2. BLENDER

Blender é um *software* livre multiplataforma de modelagem 3D que oferece recursos que o tornam uma solução adequada para a utilização profissional, competindo inclusive com aplicativos proprietários, como 3D Studio, Cinema 4D, Maya e Lightware.

Blender foi originalmente um projeto de modelagem e animação da companhia alemã NeoGeo. Em 1998, seu arquiteto Ton Roosendaal fundou a empresa "*Not a Number*" (NaN) e decidiu distribuir o Blender livremente. O financiamento para este

projeto vinha da venda de manuais e do licenciamento chamado C-Key (somente usuários registrados podiam utilizar algumas funções) (LIN, 2001). Hoje, o Blender pode ser baixado livremente e recebe patrocínio de algumas empresas financiadoras. A Figura 2 mostra a interface gráfica do Blender 3D durante a modelagem de um objeto.

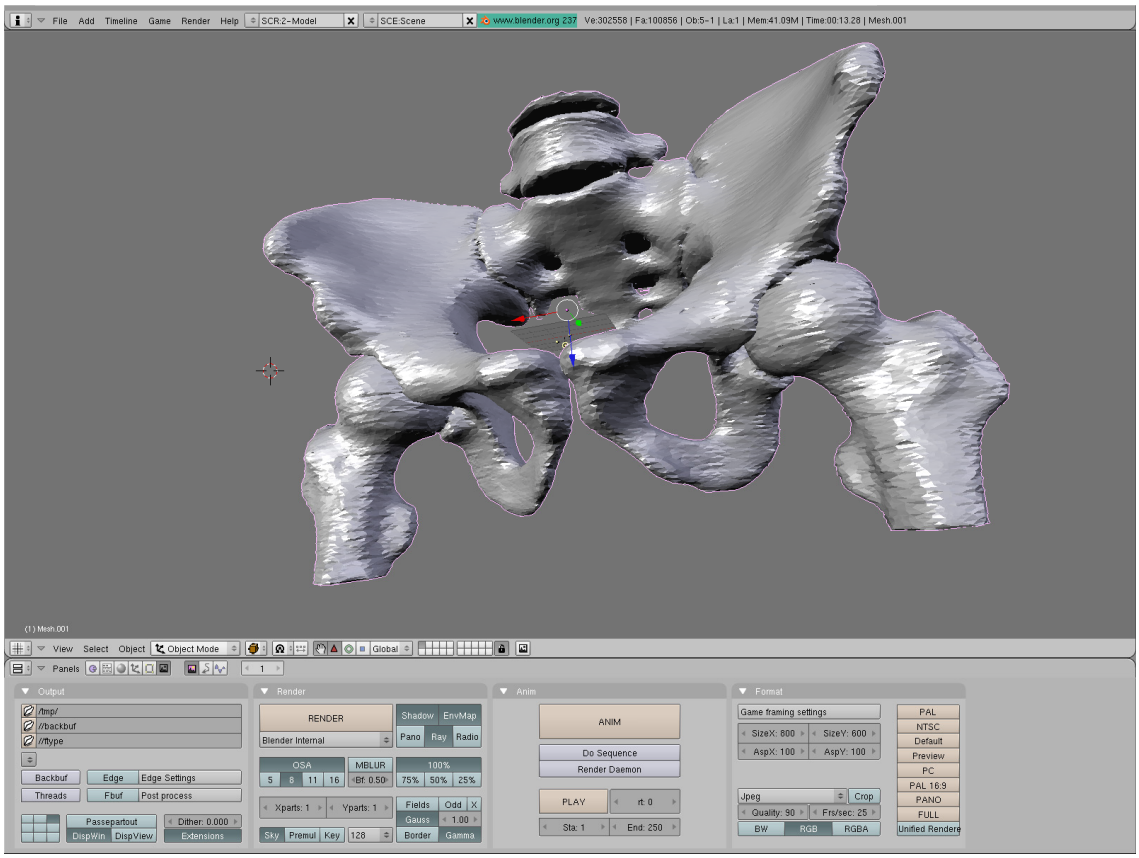

Figura 2. Interface do Blender 3D. (FONTE: http://kldp.org)

3. DRQUEUE

DrQueue é uma ferramenta baseada em *shell* para distribuição de tarefas como renderização de imagens suportando atualmente, além do Blender: Maya, Mental Ray, BMRT (*Blue Moon Rendering Tools*), 3Deligth, Aqsis, Pixie, After Effects e outros.

Sendo um *software* livre distribuído sobre GPL, o DrQueue é compatível com diversas plataformas como Linux, FreeBSD, Irix, Mac OS X além do MS Windows.

A utilização e configuração do DrQueue se dá por meio de três ferramentas principais: `master`, `slave` e `drqman`.

O `drqman` é uma GUI (*Graphical User Interface*) usada para controlar as tarefas e computadores. Com esta ferramenta é possível repriorizar ou parar tarefas, reiniciar quadros específicos, alterar quadros a serem renderizados entre outras possibilidades.

Dentre as vantagens do DrQueue podemos citar:

- O baixo custo de instalação e manutenção pelo fato de ser um programa livre;
- Ampla documentação disponível on-line sem custos;
- Suporte a redes heterogêneas, inclusive com sistemas operacionais distintos.
- Representa uma excelente forma para reciclagem de máquinas antigas utilizando sistemas operacionais leves (baseados em Knoppix, por exemplo).

Dentre as principais desvantagens observadas na utilização do DrQueue podemos citar:

- A dificuldade de operação em redes com IP dinâmico. Cada máquina da rede local deve possuir uma variável de ambiente chamada DRQUEUE_MASTER que contém o endereço IP (*Internet Protocol* – Protocolo Internet) da máquina-mestre. Embora possa ser contornado com configuração adicional, em uma rede que faz uso de DHCP (*Dynamic Host Control Protocol* – Protocolo de Controle Dinâmico de *Host*) isto se torna um inconveniente num primeiro momento, pois o endereço IP da máquina-mestre pode mudar;
- A necessidade de manter ao menos dois diretórios com permissões de acesso totalmente liberadas, somado ao fato de que qualquer usuário deve poder montar o diretório remoto representam um importante problema de segurança para a rede local a ser considerado.

4. ESTRUTURA HIERÁRQUICA E DISTRIBUIÇÃO DE TAREFAS NO CLUSTER

O sistema implementado neste trabalho tem uma estrutura hierárquica de dois níveis. No primeiro nível está o processo `drqueue.master` (ou servidor), rodando na máquina-mestre; no segundo nível, os processos clientes, dentre eles o `drqueue.slave`, rodando nas máquinas-escravo e também na máquina-mestre.

A Figura 3 mostra a estrutura hierárquica de dois níveis do sistema de renderização distribuída implementado neste trabalho.

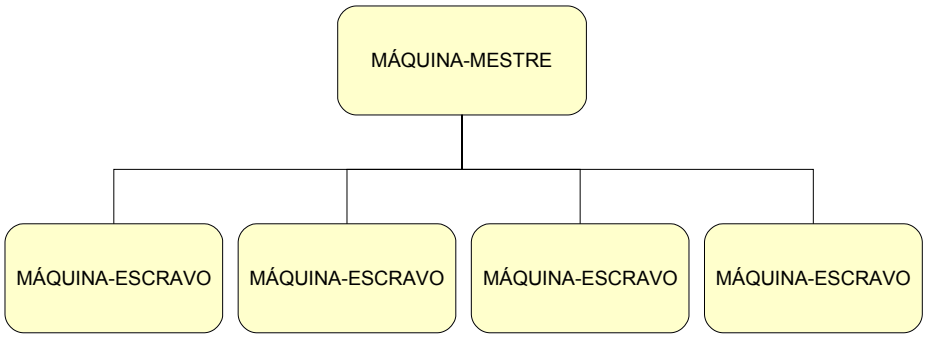

Figura 3. Estrutura hierárquica do *render farm*.

A comunicação interprocessos deste sistema distribuído se dá por meio de requisições em porta TCP (*Transfer Control Protocol* – Protocolo para Controle de Transferência) e de um diretório compartilhado usando tecnologia NFS (*Network File System* – Sistema de Arquivo de Rede).

O processo mestre mantém informações atualizadas sobre as máquinas da rede tais como: tarefas executadas, carga computacional média e tempo de CPU disponível. Ele fornece estas informações aos processos escravos e aos clientes quando assim requisitado. Tais informações são úteis para manter os escravos ocupados quando houver CPU disponível.

As requisições são realizadas por meio de uma porta TCP que deve estar aberta na rede para recebê-las. Uma questão de segurança a ser levada em conta é que todas as requisições são realizadas sem autenticação. Desta forma, a porta só deve estar aberta para máquinas confiáveis (GREVE, 2005).

As informações de *log* geradas pelo processo mestre são armazenadas em um diretório compartilhado na rede com permissão de leitura e escrita de forma que seja possível o acesso remoto destes dados.

Cada processo escravo é responsável por manter suas informações de estado e periodicamente reportá-las ao processo mestre.

No momento em que estiver ocioso e disponível para renderização, será feita a requisição de uma tarefa ao mestre que lhe atribuirá uma que seja adequada ao perfil do escravo. Para a atribuição de tarefas, o mestre leva em consideração o sistema operacional, a memória livre, a utilização de CPU e outras informações semelhantes.

Quando uma tarefa é atribuída a um escravo, este recebe todas as informações relacionadas a ela, incluindo localização do *jobscript*, tipo de tarefa, número de quadros e outras informações relativas ao modelador. Estas servirão para que o escravo crie um

novo processo no cliente com as variáveis de ambiente exigidas e execute o *jobscript* apontado para aquela tarefa.

O *jobscript* precisa estar localizado em algum lugar do diretório compartilhado de forma que todos os escravos possam lê-lo com o mesmo caminho.

O novo processo criado iniciará e armazenando o arquivo de *log* da tarefa no diretório de *logs* do DrQueue.

Enquanto o novo processo permanece em execução, o escravo periodicamente verifica sua existência para saber se houve uma finalização anormal. Quando a tarefa termina, o processo escravo recebe o estado de saída (*exit status*) do processo filho que acaba de finalizar e envia toda esta informação para o mestre.

Dependendo das informações recebidas daquela tarefa, ela é marcada como "Finalizada" ou "Erro". Esta informação torna-se disponível para todos os clientes.

Quando um escravo cria um novo processo filho para executar o *jobscript*, é importante notar que este último será executado localmente e que todas as variáveis devem ser válidas localmente. Isto inclui caminhos, arquivos executáveis que o *script* possa chamar, *plugins*, texturas, cenas, diretórios de saída etc. Logo, para que a renderização seja realizada com sucesso é necessário que tudo tenha a mesma localização no sistema de arquivos. Esta é a principal razão pela qual deve-se usar um diretório compartilhado.

Também é possível realizar renderização em ambiente de plataforma cruzada. Nestes casos, o *jobscript* assume a responsabilidade por traduzir os valores que são recebidos do sistema operacional do processo escravo na configuração necessária. Isto permite a utilização de redes heterogêneas envolvendo estações Sun, Linux e Windows, por exemplo.

5. IMPLEMENTAÇÃO DO *RENDER FARM*

A plataforma escolhida para a implantação do sistema foi o Ubuntu Linux pela sua popularidade e ampla documentação. A implementação, baseada em documentação disponível na Internet (COSTA, 2008), será descrita em detalhes nesta seção.

Considerando que o Sistema Operacional (Ubuntu Linux 8.04 LTS) esteja instalado e funcional, que a rede esteja corretamente configurada, e que se tenha acesso normal à Internet, o primeiro passo para a implementação do *render farm* é instalar os

programas que serão utilizados, neste caso, o Blender e o DrQueue, o que pode ser feito inserindo em uma janela de terminal os seguintes comandos:

```
$ sudo apt-get update
$ sudo apt-get install blender drqueue nfs-kernel-server
```

O DrQueue será instalado automaticamente no sistema e um diretório especial será criado em `/var/lib/drqueue` com toda a estrutura de subdiretórios necessária ao funcionamento do programa. Contudo, ainda é necessário criar um diretório para conter os projetos que serão processados pelo *render farm*. Nos comandos a seguir será criado um diretório para armazenar os projetos e serão configuradas as permissões para o diretório de operação do DrQueue.

```
$ sudo mkdir /var/lib/drqueue/projects
$ sudo chmod -R 777 /var/lib/drqueue
```

O DrQueue demanda também a existência das variáveis de ambiente `DRQUEUE_ROOT` e `DR-QUEUE_MASTER`, que indicam o diretório de instalação do DrQueue e o endereço IP da máquina-mestre, respectivamente, devem ser criadas. Os comandos abaixo, quando executados com privilégios de superusuário (root), inserem duas linhas no arquivo de configuração do sistema `/etc/bash.bashrc`, automatizando a criação das variáveis de ambiente durante a inicialização:

```
# echo -e "\nexport DRQUEUE_ROOT =\"/var/lib/drqueue\"" >> /etc/bash.bashrc
# echo -e "\nexport DRQUEUE_MASTER= \"192.168.2.101\"" >> /etc/bash.bashrc
```

5.1. Máquina-Mestre

As configurações específicas da máquina-mestre são: o NFS, para exportar o diretório de operação; e o conteúdo do arquivo de configuração mestre do DrQueue (`master.conf`).

O correto funcionamento do *render farm* depende do compartilhamento do diretório de operação do DrQueue. Neste caso, foi escolhido o NFS para esta função. Para tanto, é necessário inserir, na máquina-mestre, o diretório a ser compartilhado no arquivo de configuração `/etc/exports`, que pode ser configurado através dos comandos seguintes com privilégios de superusuário:

```
# echo -e "\n/var/lib/drqueue
192.168.2.0/24(rw,sync,no_wdelay,no_subtree_check)" >>
/etc/exports
```

```
# exportfs -a
```

5.2. Máquina-Escravo

O diretório que foi exportado pela máquina-mestre precisa ser montado localmente nas máquinas-escravo. Para o bom funcionamento do *render farm*, é importante que o ponto de montagem local possua a mesma localização do diretório na máquina remota. Os comandos a seguir inserem uma linha de configuração no `/etc/fstab`.

```
# echo -e "\n192.168.2.101:/var/lib/drqueue
/var/lib/drqueue nfs rw,users,noauto 0 0\n">> /etc/fstab
```

É importante notar a presença da opção `users` na linha de configuração do `fstab`. Ela faz com que qualquer usuário do sistema possa montar e desmontar este diretório. Na verdade, o mesmo usuário que executará o processo do DrQueue escravo deve montar o diretório remoto. Caso isto não seja feito, erros poderão ocorrer durante a execução dos processos escravos pela falta de privilégios por parte do usuário. Por este mesmo motivo é utilizada a opção `noauto`, para que o diretório não seja montado automaticamente durante a inicialização do sistema, forçando o próprio usuário a fazê-lo. Desta forma, na máquina-escravo deve ser executada uma operação de montagem antes de se iniciar o processamento distribuído através do comando:

```
$ mount /var/lib/drqueue
```

5.3. Inicialização dos Serviços

Para iniciar os processos do *render farm* na máquina-mestre, basta executar em um terminal de console os comandos:

```
$ drqueue.master
```

```
$ drqueue.slave
```

Nas máquinas-escravo, inicia-se o processo do DrQueue por um terminal de console através do comando:

```
$ drqueue.slave
```

Neste ponto, a *reder farm* está configurada e preparada para receber as tarefas (*jobs*) a serem realizadas.

6. METODOLOGIA UTILIZADA

As máquinas utilizadas possuem processador AMD Sempron 3000+, 1 GB de memória principal, placa de rede *on-board* de 10/100 Mbps e sistema operacional Ubuntu Linux 8.04 LTS.

A rede de interconexão entre as máquinas usa um Switch Ethernet Planet de 24 portas não-gerenciável.

Utiliza-se, durante os testes, a GUI (*Graphical User Interface* – Interface Gráfica do Usuário) padrão do Ubuntu, ou seja, Gnome, contudo qualquer outra interface pode ser utilizada.

Para realização da medição foi utilizada uma animação simples criada no Blender com 50 frames, todos eles de carga computacional exigida para renderização aproximadamente semelhante. A priorização da tarefa foi configurada para "normal".

Para inserir um *job* usa-se a ferramenta `drqman`, que deve ser executada na máquina-mestre através do comando:

```
$ drqman
```

Será invocado através deste comando o gerenciador de tarefas do DrQueue, cuja janela pode ser vista na Figura 4.

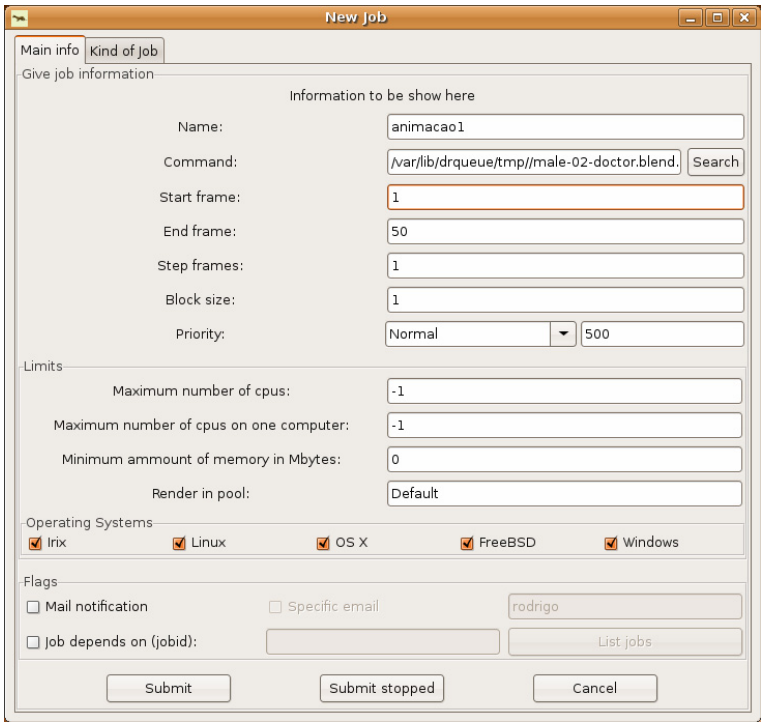

Figura 4. Janela do gerenciador de tarefas do DrQueue (`drqman`).

Através do comando anterior, são configurados os parâmetros da tarefa a ser executada no sistema de renderização distribuída, incluindo o número de quadros, prioridade da tarefa, parâmetros para cascateamento de tarefas, limitação quanto ao número de máquinas utilizada entre outras opções.

7. ANÁLISE DE DESEMPENHO

Considera-se para efeitos de análise neste trabalho o modelo de computação paralela com seções seriais (MAGGS; MATHESON; TARJAN, 1995), onde a tarefa a ser realizada é divisível em duas partes, sendo uma fração γ serial (não-paralelizável) e outra paralelizável dada por $1 - \gamma$.

O fator de *speedup* é uma métrica bastante utilizada no meio científico para avaliação do ganho de desempenho obtido com a paralelização de uma determinada tarefa. Segundo Amdahl (1967), o *speedup* relativo para a execução de uma certa tarefa sobre p processadores é definido como a razão entre o tempo gasto para sua execução em um único processador e o tempo gasto para sua execução sobre p processadores. Considerando o tempo de execução de uma determinada tarefa sobre p processadores como sendo $T(p)$, pode-se expressar o fator de *speedup* $S(p)$, em termos matemáticos, como:

$$S(p) = \frac{T(1)}{T(p)} \tag{1}$$

Esta métrica sugere que a fração serial da tarefa possa ser dividida de forma igualitária entre os processadores. Do ponto de vista prático, isto nem sempre é uma verdade, fazendo com que, para determinada porção do código γ_i seja alcançado um certo fator de *speedup* S_i associado. A generalização do fator de *speedup* calculado levando-se em conta esta limitação leva à expressão:

$$S(p) = \frac{1}{\sum_{i=1}^{p} \frac{\gamma_i}{S_i}} \tag{2}$$

A equação (2) é conhecida como "Lei de Amdahl Generalizada".

Uma outra métrica de interesse na avaliação da computação paralela é a eficiência da paralelização. Esta medida nada mais é do que a razão entre o fator de *speedup* e o número de processadores utilizados, ou seja, é a normalização do fator de *speedup* em relação ao número de processadores. A eficiência ξ (p) pode, portanto, ser escrita como:

$$\xi(p) = \frac{S(p)}{p} \tag{3}$$

A eficiência fica normalmente limitada entre 0 e 1. Em casos práticos, entretanto, é possível verificar valores de eficiência superiores a 1 quando algoritmos paralelos superam os seriais de forma sinergética, especialmente para de fatores de *speedup* reais, em detrimento dos relativos (CASTRO, 2007).

Foi calculado o fator de *speedup* real (AMDAHL, 1967) e a eficiência para diversos números de processadores utilizando a animação de teste para fins de avaliação de desempenho gerada em Blender. A Tabela 1 apresenta um resumo dos resultados obtidos.

Tabela 1. Fator de speedup real e eficiência de paralelismo calculados até 10 processadores.

p	T(p)	S(p)	ξ(p)
1	12:21,0	1,000	100,0%
2	06:08,0	2,014	100,7%
3	04:06,0	3,012	100,4%
4	03;06,0	3,984	99,6%
5	02:30,0	4,940	98,8%

p	T(p)	S(p)	ξ(p)
6	02:05,0	5,928	98,8%
7	1:47,0	6,925	98,9%
8	1:34,0	7,883	98,5%
9	1:24,0	8,821	98,0%
10	1:15,0	9,880	98,8%

É importante notar a presença de duas medidas onde o *speedup* ligeiramente superlinear é observado. Este é um dos casos reais onde o ganho por paralelismo apresenta sinergia, ou seja, quando o fator de *speedup* apresenta características superlineares.

Nota-se que a eficiência se mantém muito próxima de 100% para todos os testes realizados, mostrando que o DrQueue possui um bom algoritmo de renderização distribuída mesmo representando uma solução de software livre.

8. CONCLUSÕES

A Figura 5 apresenta os resultados obtidos na forma gráfica. Da Figura 5a nota-se claramente que o fator de *speedup* desta tarefa apresenta comportamento linear para os parâmetros utilizados.

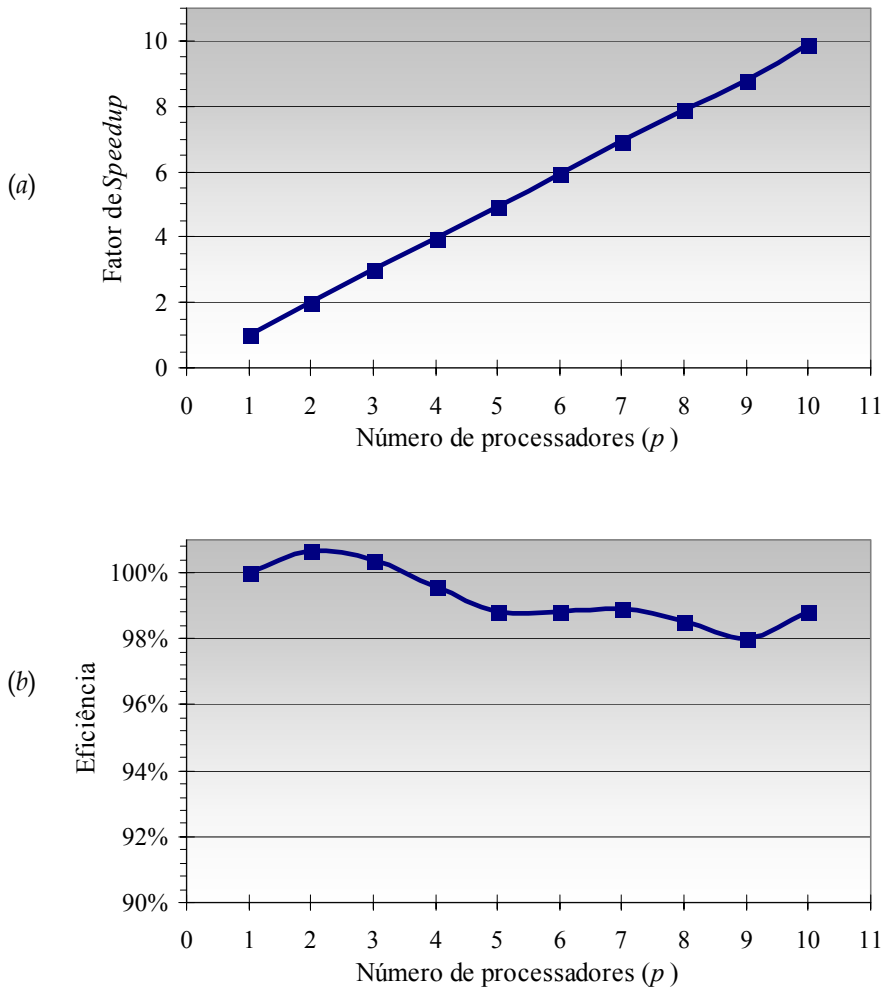

Figura 5. Fator de *speedup* (a) e eficiência (b) medidos no sistema.

Para o número de máquinas avaliado não foi possível notar um impacto significativo do *overhead* de comunicação no desempenho do sistema. Isto mostra que para pequenos *render farms*, os resultados obtidos aproximam-se muito do fator de *speedup* linear, ou seja, o caso prático ideal. Acreditamos que as diferenças notadas entre os fatores de *speedup* e eficiência para diferentes números de máquinas-escravo utilizadas devem-se principalmente ao fato de o DrQueue não fracionar quadros na distribuição de tarefas, ou seja, a menor tarefa que pode ser alocada a um processo escravo é um quadro. Isto provoca um impacto direto nas métricas de desempenho dependendo do número de quadros da animação e do número de máquinas utilizado. Como no teste realizado foram utilizados quadros muito semelhantes, ou seja, a demanda de computação é praticamente a mesma para todos os quadros, os resultados tornam-se bastante dependentes da relação numérica entre a quantidade de quadros e a quantidade de

processadores. Em resumo, quanto mais máquinas ociosas se têm no último ciclo de renderização (já que não há quadros para todos), pior será o desempenho.

Por outro lado, para tarefas que possuem quadros com grande divergência na exigência computacional de renderização, o *speedup* ficará limitado ao *frame* mais complexo da animação, ou seja, aquele que levará mais tempo para ser renderizado.

REFERÊNCIAS

AMDAHL, G. Validity of the single processor approach to achieving large-scale computing capabilities. In: **AFIPS Conference Proceedings**. [S.l.: s.n.], 1967. p. 483-485.

BLENDER. v.2.46. Disponível em: <http://www.blender.org>. Acesso em: mar. 2008.

CASTRO, E. O. **Multiprocessador em eletrônica reconfigurável para aplicações robóticas**. Dissertação de Mestrado. UNICAMP, 2007.

COSTA, A. S. **Dr. queue com Blender**: um render farm GNU. Disponível em: <http://ana-kinpendragon.files.wordpress.com/2007/02/blen der-drqueue2.pdf>. Acesso em: mar. 2008.

DRQUEUE. v.0.64.3. Disponível em: <http://www.drqueue.org>. Acesso em: mar. 2008.

GREVE, G. C. F. BRAVE GNU world. **Linux Magazine**. n. 52, mar. 2005.

KLDP. Open source, Geek, IT… Disponível em: <http://kldp.org>. Acesso em: nov. 2008.

LIN, Norman. **Linux 3D graphics programming**. EUA: WordWare Publishing, 2001.

MAGGS, B.; MATHESON, L.; TARJAN, R. Models of parallel computation: a survey and synthesis. System Sciences. In: **Proceedings of the Twenty-Eighth Hawaii International Conference on**, v. 2, 1995.

MCREYNOLDS, T.; BLYTHE, D. **Advanced graphics programming using OpenGL**. San Francisco: Elsevier, 2005.