Adail Antônio Pinto Júnior

Blender e Drqueue – Uma Render Farm Open Source no Linux

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Joaquim Quinteiro Uchôa

Lavras
Minas Gerais – Brasil
2009

Adail Antônio Pinto Júnior

Blender e Drqueue – Uma Render Farm Open Source no Linux

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em 20 de Abril de 2009

_____
Me. Prof. Denilson Vedoveto Martins

_____
Ma. Professora Kátia Uchôa

_____
Me. Prof. Joaquim Quinteiro Uchôa (Orientador)

Lavras
Minas Gerais – Brasil
2009

Dedico este trabalho a Deus e a todos àqueles que me auxiliam em minha caminhada nesta terra.

Agradecimentos

Agradeço primeiramente a Deus, por dar-me força e inteligência para completar este trabalho, aos meus pais por tudo o que fizeram e fazem por mim e à toda a equipe da ARL, pelo excelente trabalho e pela oportunidade de completar um curso de pós-graduação a distância.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A produção de imagens, animações e filmes 3D em computadores, demanda grande poder computacional e sofre com o tempo gasto para se renderizar uma produção. Este trabalho tem o intuito de apresentar uma *render farm open source* como solução, tanto para o problema da demanda de poder computacional, como para o problema da demora no tempo de renderização. O trabalho traz conceitos sobre sistemas distribuídos, mais especificamente *render farms.* Apresenta os aplicativos Blender e Drqueue, como aplicativos principais na *render farm.* E mostra os passos necessários para se estruturar e configurar uma *render farm open source.*

Palavras-chave: *Render farm,* sistemas distribuídos, Blender, Drqueue, animação, renderização, imagens, 3D, *cluster*.

Capítulo 1

Introdução

A criação de imagens, animações, comerciais e filmes 3D em computadores, é uma das tarefas que mais requer poder de processamento e tempo para serem concluídas. Dependendo do tamanho e do nível de detalhes empregados no projeto, este pode demorar meses para ser processado.

Existem diversas formas, para se tentar contornar a insaciável necessidade de poder de processamento, inerente a projetos de criação e renderização de conteúdo 3D. Entre essas formas, pode-se citar: a compra de uma única máquina com grande poder de processamento e alto custo; a utilização de serviços que vendem ciclos de processamento, para tarefas determinadas, a preços variados; ou mesmo, a montagem de *clusters* locais compostos de vários *desktops* comuns.

O objetivo deste trabalho é apresentar os conceitos básicos e uma forma de implementação de uma *render farm open source,* apresentando-a como uma solução, ao problema da necessidade de poder de processamento e da demora no tempo de renderização. Como aplicativo para criação, renderização e editoração de conteúdo 3D, será utilizado o Blender. E para o gerenciamento e distribuição do trabalho de renderização pela rede de computadores, será utilizado o Drqueue.

Como objetivo secundário, este trabalho tem o intuito de mostrar a maturidade do software *open source* Blender, na consecução de obras e trabalhos de criação, editoração, renderização e modelagem 3D.

O trabalho esta organizado como se segue. O Capítulo 2 apresenta os conceitos e definições utilizados e ainda introduz o Blender e o Drqueue como soluções. O Capítulo 3 apresenta os componentes, a estrutura e a forma de implementação, configuração e utilização da *render farm.* O Capítulo 4 apresenta os testes efetuados e os resultados obtidos. E o Capítulo 5 apresenta as conclusões obtidas e aponta para trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta os conceitos básicos sobre sistemas distribuídos, mais especificamente sobre *render farms* e introduz os procedimentos envolvidos na consecução de um projeto de criação de animações. Este capítulo faz ainda, uma apresentação do Blender e o do Drqueue.

2.1 – Conceitos e definições

2.1.1 - Sistemas Distribuídos

Computadores podem efetuar operações e cálculos matemáticos em frações de segundo. Computadores são ideais para aplicações científicas, em que fórmulas matemáticas de alta complexidade são utilizadas, ou para aplicações comerciais, onde a quantidade de dados e de acessos por minuto é muito grande. A velocidade dos computadores também pode ser empregada em aplicações 3D, na criação de animações e filmes, ou mesmo em aplicações críticas, onde um determinado evento, tem que ser monitorado incessantemente.

Apesar da velocidade com que os computadores conseguem executar cálculos e do constante aumento da capacidade de processamento dos mesmos, a demanda por mais poder de processamento cresce incessantemente. E cresce em um ritmo tão acelerado, que impossibilita a indústria de prover máquinas que atendam a toda essa demanda.

Essa demanda por poder computacional é oriunda das mais diversas causas. Entre as causas da crescente demanda por poder computacional, pode-se citar:

- o crescimento das empresas e a incapacidade de seus servidores em suprir a demanda de acessos
- o desenvolvimento de sistemas de pesquisa que requerem cada vez mais processamento, para cálculos cada vez mais complexos
- a produção de filmes e animações por empresas cinematográficas, que desejam efeitos especiais mais realistas e personagens em animações, com cada vez mais detalhes.

Aumentar a velocidade em que um computador trabalha demanda pesquisas nas mais diversas áreas do conhecimento humano. Desde pesquisas em busca do desenvolvimento de substâncias, que sejam melhores condutoras elétricas, até pesquisas por melhores formas de miniaturização de componentes. Estes estudos levam tempo e não acompanham a demanda de mercado por poder computacional.

Uma solução cada dia mais utilizada para tentar sanar o problema da demanda de poder computacional, sem aumentar a velocidade com que um computador específico trabalha, é colocar vários desses mesmos computadores para trabalharem em conjunto. Ou seja, essa solução distribui o trabalho entre várias máquinas, aumentando assim a velocidade com que o trabalho é realizado e não propriamente a velocidade dos computadores envolvidos.

Um sistema computacional onde é possível distribuir a carga de trabalho

(*workload)* a vários computadores é chamado de sistema distribuído**.** Na definição de [Couloris, 2007] um **sistema distribuído** é "*aquele no qual os componentes de hardware ou software, localizados em computadores interligados em rede, se comunicam e coordenam suas ações apenas enviando mensagens entre si."*

Existem vários níveis e várias formas diferentes de se criar um sistema distribuído. [Couloris, 2007] afirma que pode-se dividir os sistemas distribuídos em dois modelos de arquitetura básicos: cliente/servidor e *peer-to-peer*. A Figura 2.1 traz uma representação destes modelos básicos.O modelo de arquitetura de um sistema computacional define a forma pela qual as entidades comunicantes que compõem o sistema interagem entre si.

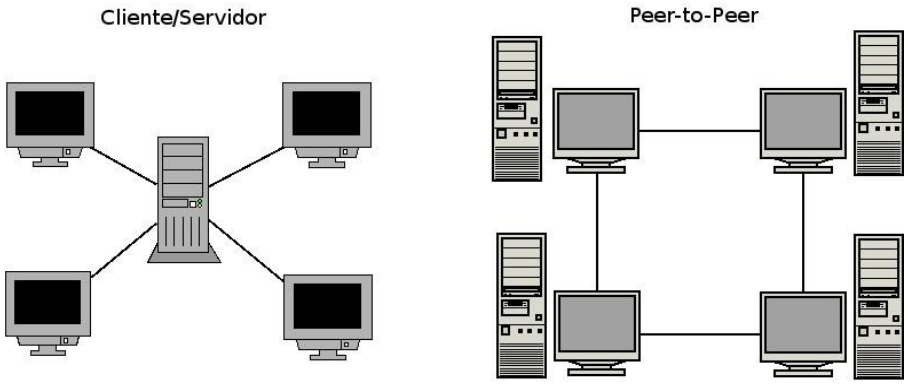

Figura 2.1: Arquiteturas de Rede

No modelo cliente/servidor existe um ou mais computadores entitulados de "servidor". A função do servidor é prover serviço aos clientes. Este serviço geralmente é a execução de um determinada tarefa e a devolução de um resultado. Mas este serviço, pode ser também um trabalho de gerência de uma

determinada tarefa complexa, que é fragmentada e distribuída a um grupo de clientes, que por sua vez, executam individualmente suas frações da tarefa original. Terminado o trabalho, os clientes devolvem as suas frações da tarefa ao servidor, que apresenta o trabalho como um todo, ao usuário o qual a solicitou.

No modelo *peer-to-peer* essa divisão cliente/servidor deixa de ter significado. Nessa arquitetura, todos os processos envolvidos em uma tarefa ou atividade, desempenham funções semelhantes interagindo como pares (*peers*). Nestes pares existem processos clientes e processos servidores. Pode-se dizer que na arquitetura *peer-to-peer,* os pares provêem serviço, da mesma forma como solicitam serviços, não havendo distinção entre as funções dos mesmos.

É importante notar que essa denominação de cliente ou servidor, é dada ao processo (programa/software) que solicita e/ou fornece informações, e não ao computador propriamente dito. Um mesmo computador pode funcionar como cliente de um serviço web, utilizando um navegador como *software* cliente. E o mesmo computador pode atuar, utilizando o *software* Samba, como um servidor de arquivos.

Segundo [Bookman, 2003], computação distribuída *"é a habilidade de executar vários trabalhos através da potência de processamento de ambientes, principalmente heterogêneos"*. Existem sistemas distribuídos que se espalham por todo o globo terrestre e são constituídos de diversos tipos e classes de equipamentos. Muitas vezes estes equipamentos, utilizam a própria internet para se conectar e propiciam serviços a milhões de usuários. Como exemplos destes sistemas distribuídos, podem ser citados os sistemas de busca da Google e os sistemas de vendas da Amazon, que possuem vários servidores espalhados pelo mundo. Pode-se citar, como exemplos de sistemas distribuídos de menor abrangência, uma intranet[1], uma *render farm* ou um software de gestão de filiais, entre tantos outros.

[1] Rede interna de uma empresa ou instituição, que provê serviços específicos

2.1.2 Renderização de imagens e a *Render Farm*

Um setor que utiliza massivamente computação distribuída para a consecução de um objetivo, é a indústria de filmes e propagandas. É cada dia mais comum na produção de um filme longa metragem, a utilização de computadores interligados em rede para criar efeitos especiais, personagens digitais e até mesmo mundos inteiros virtuais. Basta ligar a televisão, para encontrar em um comercial, um personagem ou uma efeito criado virtualmente.

Este processo de criação de efeitos ou de animações digitais, segue um certo roteiro. Se hipotéticamente, será criado digitalmente um arranha-céus, primeiramente é criado um *wireframe*. Um *wireframe* é um esboço com poucos detalhes do objeto que se deseja modelar/criar, construído apenas com linhas (arames/*wire*) e/ou superfícies simples. No caso do arranha-céus, o *wireframe* apresentaria apenas seu esboço, possibilitando visualizar sua estrutura básica.

É através do *wireframe* que serão atribuídas aos objetos as suas características especiais, como a textura dos materiais utilizados ou os reflexos luminosos em sua superfície. Porém estas características não são consideradas na imagem apresentada ao usuário, o que o usuário realmente vê, são apenas linhas delimitando as formas.

A **renderização** é o processo que dá acabamento e realismo à cena criada. Na definição de [Andauer, 2004], *"renderização é o processo final da computação gráfica(CG) e é a fase em que a imagem correspondente a sua cena 3D é finalmente criada"*. É durante a renderização que são consideradas, calculadas e aplicadas todas as características especiais de cada objeto. Pode-se citar como características especiais: a incidência de raios luminosos; a extensão e intensidade de sombras; as texturas e os índices de maleabilidade dos objetos; o movimento e choque de partículas; entre outros. Segundo [Gonzáles-Morcillo, 2007] é na fase de renderização que "*os algoritmos de renderização recebem*

uma definição da geometria, materiais, texturas, fontes de luzes e câmera virtual, como entrada e produzem uma imagem (ou conjunto de imagens no caso de animações) como saída."

No caso do exemplo do arranha-céus, seria a renderização que possibilitaria perceber as janelas de vidro, os reflexos em superfícies espelhadas, o volume dos objetos, os tipos de materiais utilizados e a sombra dos objetos. Seria na renderização, que o arranha-céus se tornaria mais ou menos "real", dependendo apenas da perícia do criador do projeto e da qualidade e dos propósitos do software utilizado. Na Figura 2.2 é apresentada Susane, mascote do Blender, exibida em *wireframe*, em *wireframe* com faces e por fim renderizada.

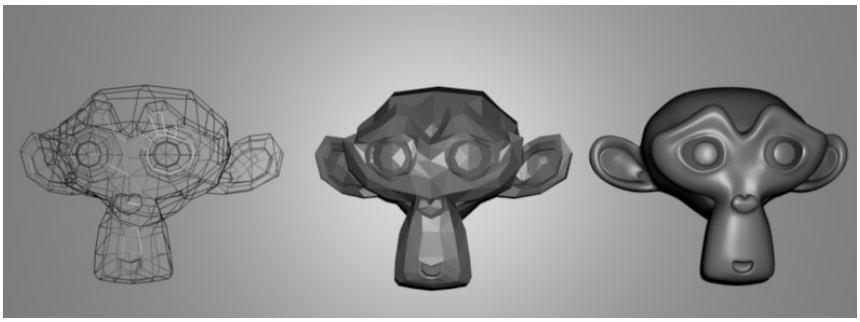

Figura 2.2: *Wireframe, wireframe* com faces e renderização.

Devido à todos os fatores enunciados, a renderização é um processo muito demorado. Dependendo do projeto proposto, ou seja, da riqueza de detalhes e da quantidade de cenas, a fase de renderização pode ser mais longa que todas as fases anteriores do projeto.

Para ilustrar o problema do tempo da renderização, consideremos uma animação, que apresenta a trajetória de uma bola de tênis sendo liberada do 8º andar de um edifício e depois tocando o solo. A animação em questão tem exatos seis segundos.

Uma **animação** é um conjunto ordenado de imagens, que exibido em

uma determinada velocidade e seqüência estabelecidas pelo seu criador, dá a impressão de que os objetos estão em movimento. [*Mullen*, 2007] define animação como "*uma série de figuras levemente alteradas que, quando vistas em uma rápida seqüência, criam a ilusão de movimento"*[1]. Cada imagem que compõe uma animação, é chamada de *frame.* A velocidade de uma animação é medida pela quantidade de imagens que é exibida por segundo e é representada pela sigla FPS – *frames por segundo*.

No exemplo da bola de tênis, a velocidade da animação será de 25FPS, ou seja, para cada segundo de animação serão exibidos 25 *frames*.

Multiplicando 25 *frames* x 6 segundos será obtido o tatal de 150 *frames* para serem renderizados. Se o tempo levado para renderizar cada *frame*, for de 34 minutos, multiplicando-se o total de *frames* pelo tempo gasto para renderizar cada *frame* (150x34), será obtido o total de 5100 minutos ou 85 horas de renderização. Para uma animação de 6 segundos, 5100 minutos de renderização é bastante tempo.

No cálculo anterior foi levado em conta apenas um processador, que pode renderizar um *frame* por vez. Se fossem utilizados dois computadores iguais ou um computador com dois processadores, e a cada um fosse delegado metade da animação para renderizar, o trabalho seria terminado em aproximadamente metade do tempo.

Existem aplicativos que tratam exatamente da tarefa de dividir o *workflow* (fluxo de trabalho), entre os computadores. Um sistema computacional que permite dividir o trabalho de renderização de um grupo de imagens, entre vários computadores usando um software de controle de *jobs* (trabalhos) por *queue* (fila), é chamado de ***render farm***.

Uma *render farm* é um tipo de *cluster*, que trabalha com processamento paralelo. Segundo [Deitel, 2005], "*Clustering interconecta nodos dentro de uma LAN de alta velocidade para funcionar como um único computador paralelo".*

[1] Tradução do autor.

Na definição de [Gonzáles-Morcillo, 2007] *render farm* é "*um cluster de computadores de uma empresa em que cada frame da animação é calculado independentemente por um único processador."*

A principal função do software de controle de *jobs*, é controlar o fluxo de trabalho. As tarefas são distribuídas e o trabalho é delegado aos nodos[1] que não estão em uso pela *render farm* ou aos que já terminaram a tarefa anterior. Outra característica de uma *render farm,* é o fato de que os computadores que fazem parte da mesma, não precisam ser dedicados. Estes computadores podem funcionar perfeitamente como estações de trabalho, enquanto os *jobs* são processados *em background* (segundo plano).

Retomando a situação hipotética da animação da bola de tênis, se fosse utilizada uma *render farm* com 10 computadores, a renderização terminaria em aproximadamente 8 horas e meia, o que seria um ganho excepcional de tempo.

Estúdios famosos de animação como a Pixar e a Dreamworks , dispõem de *render farms* com 1024 processadores ou mais. Muitas vezes esses grupos de computadores renderizam filmes durante meses, alcançando níveis de detalhes altíssimos.

Um bom exemplo do tempo gasto para se renderizar uma grande produção cinematográfica é o caso da trilogia Shrek, produzida pela Dreamworks. Em 2001, o primeiro Shrek gastou 5 milhões de horas de processamento na renderização. Em 2004, Shrek 2 gastou mais de 10 milhões de horas de processamento. E em 2007, Shrek 3 gastou mais de 20 milhões de horas de processamento na renderização. Para a renderização de Shrek 3, a Dreamworks utilizou sua *render farm* que conta com mais de 3000 processadores. São utilizados servidores HP DL145 G2, com 2 GB de memória para cada núcleo, sendo que cada servidor possui quatro núcleos e sistema operacional Red Hat Linux.[2]

---

[1] Termo utilizado para se referenciar um computador que faz parte de um grupo de computadores que dividem tarefas em uma rede.

[2] Dados disponíveis em http://www.linuxjournal.com/article/9653

2.2 – *Software*

2.2.1 - Blender

O Blender[2] é uma suíte de criação de conteúdo 3D, que possui uma grande variedade de ferramentas e funcionalidades para a criação de imagens e animações de alta qualidade. [Kreusel, 2006] define o Blender como "*um aplicativo de código aberto capaz de lidar com todos os passos da criação de um modelo digital: desde a sua construção, passando pela composição da superfície, até a renderização da cena tridimensional."*

Como exemplos de ferramentas para modelagem pode-se citar: a possibilidade de modelar *meshes* por *vertex, edges* ou *faces*; os deformadores de pilha como *lattice*, curvas e armaduras; o sistema Catmull-Clark de subdivisão de objetos para melhora da suavidade das formas; entre outras. Além de vários tipos de objetos, como meshes, superfícies NURBS (*Non Uniform Rational Basis Spline)* e curvas.

Estão presentes vários controles diferentes de eventos físicos. Existe o controle e simulação de partículas, que permite a simulação de vento, vortex e choque de partículas. O simulador de fluídos que permite a criação de líquidos e a definição de viscosidade e o efeito da gravidade nos mesmos. Entre outras ferramentas que permitem a simulação de eventos físicos.

O Blender possui suporte a criação de *scripts* Python, para a automação de tarefas. O software também trás um sistema de composição para edição dos vídeos, que permite a mixagem de áudio e vídeo, cortes e etc. Estão presentes várias ferramentas para sombreamento, como difusores de sombreamento e

[2] Site do projeto: http://www.blender.org

editores de nodes, que servem para a criação e mistura de materiais complexos.

Além do renderizador interno, o Blender suporta vários renderizadores externos e possui suporte integral a máquina de renderização *open source* Yafaray, que renderiza imagens traçando o caminho percorrido pela luz em uma cena 3D. Segundo [Dos Anjos, 2006], "*o Blender possui uma característica em especial que lhe possibilita o uso em ambientes de renderização distribuída: ele permite a execução em background de renderizações, sem que o ambiente de modelagem seja iniciado".* É esta característica especial citada por [Dos Anjos, 2006], que permite a criação de uma *render farm* utilizando o Blender como motor de renderização.

O Blender possui ainda uma engine de jogos integrada e totalmente funcional, que permite a criação de jogos de alta qualidade. A engine é bem completa e possui várias funcionalidades interessantes, como suporte ao *"Bullet Physics Library",* biblioteca *open source* de detecção de colisão e de dinâmica de corpo rígido para o *Playstation 3* da Sony. Outra funcionalidade interessante da engine de jogos é a API de *scripts* Python para controle sofisticado das ações, Inteligência Artificial e definição de lógica avançada de jogos.

O Blender oferece suporte para as principais plataformas do mercado atual, suportando Windows 2000, XP, Vista, Mac OS X (PPC e Intel), Linux (i386 e PPC), FreeBSD 5.4 (i386), SGI Irix 6.5 e Sun Solaris 2.8 (sparc). O pacote de distribuição do software possui apenas 14 MB e é possível utilizar o software sem nem mesmo instala-lo, bastando para isso, descompactá-lo e executar o binário.

A interface de trabalho do Blender é inovadora e foi elaborada buscando-se propiciar o maior nível possível de produtividade. Apesar de causar uma pequena sensação de impotência nos primeiros contatos, a interface realmente propicia excelentes condições de trabalho. E a medida em que o usuário se familiariza com a interface, percebe o quanto é bem elaborada e funcional. A Figura 2.3 apresenta a interface do Blender.

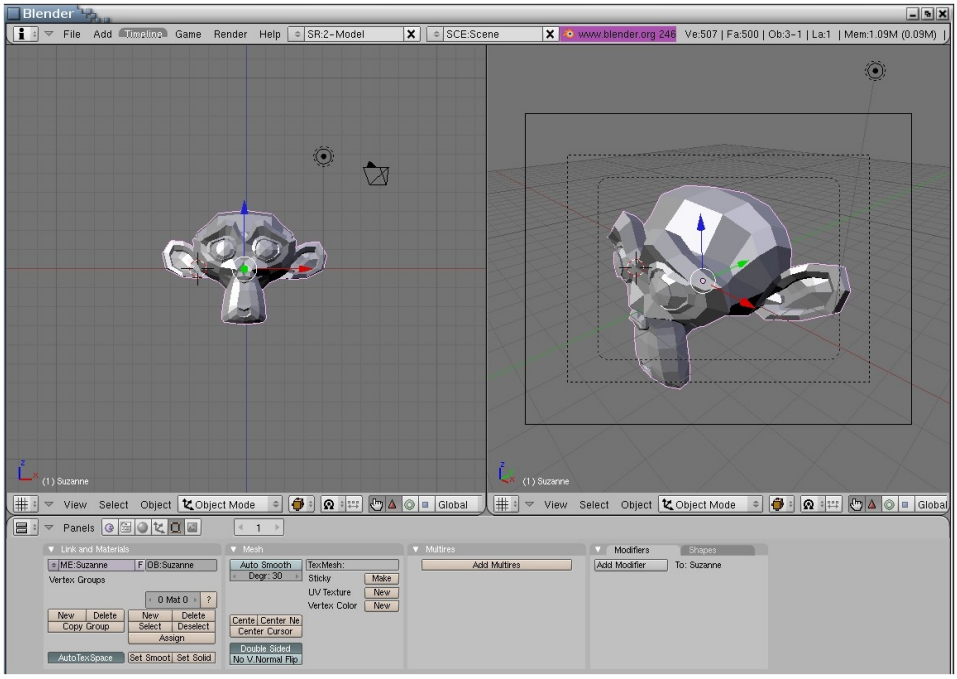

Figura 2.3: A interface do Blender

Existem vários grandes trabalhos criados utilizando somente o Blender ou tendo-o como ferramenta principal no projeto. Como exemplos de trabalhos criados inteiramente no Blender pode-se citar os *open movies* "Big Bucky Bunny"(Figura 2.4) e "Elephants Dreams"(Figura 2.5)[1].

[1] Os dados apresentados podem ser encontrados em: http://www.blender.org/features-gallery/blender-open-projects/

Figura 2.4: Imagem do filme Big Buck Bunny

Fonte: http://www.bigbuckbunny.org/index.php/media-gallery/

Figura 2.5: Imagem do filme Elephants Dream

Fonte: http://orange.blender.org/wp-content/themes/orange/images/media/

2.2.2 Drqueue

O Drqueue[1] é um sistema *open source* para gerenciamento de uma *render farm* distribuída. O software, cuida da distribuição, enfileiramento (*queueing)*, monitoramento e gerenciamento das tarefas em uma rede de computadores. Na definição de [Greeve, 2005], o *"Drqueue distribui a tarefa de renderizar múltiplas imagens individuais ou animações em um pool de computadores."*

Um *job* é um conjunto de tarefas a serem distribuídos pela *render farm.* Uma tarefa é composta de um *script* com os dados para renderização do *frame* ao qual ele se refere. O Drqueue gerencia a distribuição dos *frames* aos nodos que vão ficando desocupados. O numero de tarefas é correlato ao numero de *frames* a renderizar. Uma animação com 100 *frames* a renderizar, será um *job* com 100 tarefas. A medida em que as tarefas vão ficando prontas em cada um dos nodos, estes devolvem o *frame* já renderizado ao servidor.

Todo o processo é baseado em enfileiramento, que significa criar uma fila de tarefas, que vão sendo executadas pelos nodos à medida em que eles vão ficando disponíveis.

A Figura 2.6 ilustra o funcionamento do Drqueue., onde um cliente cria um *job* com 10 tarefas a serem distribuídas. Na ilustração os nodos já renderizaram três imagens, estão recebendo mais três para renderização e o buffer do servidor ainda possui quatro tarefas para serem enviadas.

O Drqueue suporta Linux, MacOS X, Irix, FreeBSD e Windows, como sistemas operacionais para ser instalado. E oferece suporte a vários *renderers,* incluindo suporte ao Blender, Maya, Lightwave, Mental Ray, Softimage XSI, Mantra/Houdini, entre outros. Basicamente, se o aplicativo de renderização pode

[1] Site do projeto: http://www.drqueue.org

ser executado sem interface gráfica, é possível utilizar o Drqueue para criar os *scripts* de renderização e gerenciar as tarefas distribuídas.

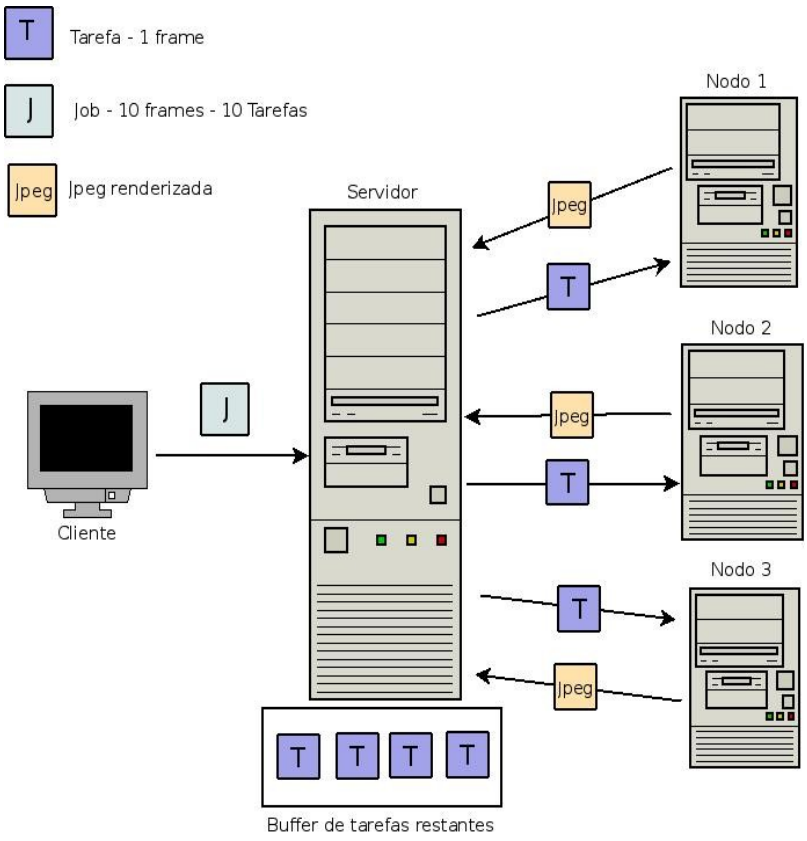

Figura 2.6: Funcionamento Drqueue

O Drqueue pode ser gerenciado pelo drqman (Figura 2.7), que é a interface padrão do sistema. Pelo drqman pode-se iniciar *jobs,* pará-los, estabelecer níveis de prioridades, mudar os *frame*s a serem renderizados entre outras funcionalidades. Além do drqman o projeto já conta com o *DrKeewee,* que é um pequeno serviço *web* baseado em Python, que permite a checagem do

status da *render farm* pelo navegardor *web.* O Drqueue possui, em andamento, um sub-projeto para criar uma nova interface *web* utilizando *RubyOnRails.*

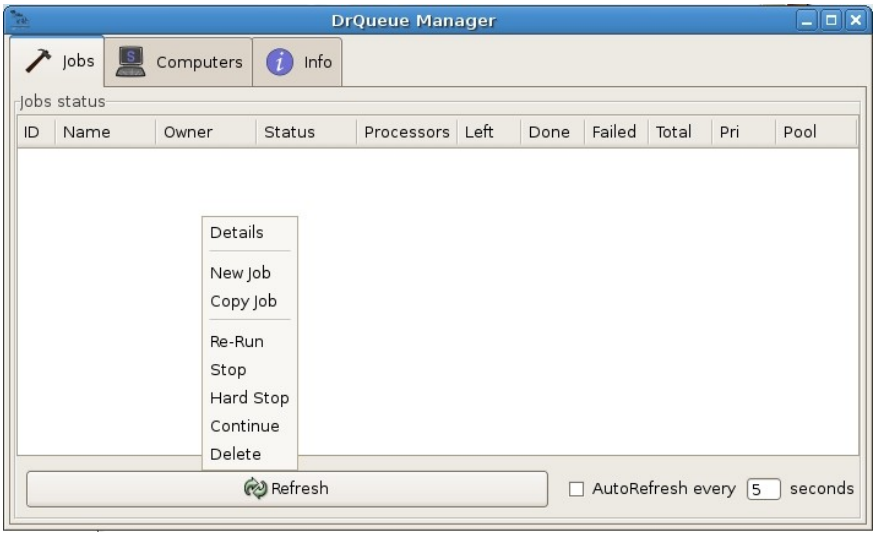

Figura 2.7: drqman – Drqueue Manager

O software possui todo um conjunto de chamadas, objetos e tipos de dados, em linguagem Python e de Ruby On Rails, que permitem a criação de ferramentas, *interfaces* e *daemons*, que não estejam incluída no pacote principal.

Como exemplos de grandes produções que utilizaram o Drqueue estão: “Elephants Dream”, “Pirates of the Caribbean: Dead Man's Chest”, “Exorcist, The Beginning” e ”Bee Season”[1].

[1] Informações disponíveis em: https://ssl.drqueue.org/project/wiki/SomeDrQueueUsers

Capítulo 3

Metodologia

Este capítulo apresenta a estrutura e os componentes da *render farm open source.* Apresenta ainda informações a respeito dos requisitos de sistema necessários para implementar o projeto e as informações para a instalação e configuração dos componentes, nas máquinas que farão parte da *render farm*. Por fim este capítulo apresenta os passos necessários, para iniciar os trabalhos de renderização, na *render farm.*.

A forma de apresentação das informações e instruções neste capítulo, presume conhecimento básico sobre a instalação e a configuração de sistemas operacionais Linux. Somente as informações absolutamente necessárias e direcionadas à implementação da *render farm,* serão apresentadas.

Por motivos práticos, presume-se acesso a internet por todas as máquinas, durante a fase de instalação e configuração da *render farm.*

Durante a apresentação de linhas de comandos, o simbolo "#" significa que o comando deve ser executado com privilégios de super-usuário e "$" significa privilégios de um usuário comum.

3.1 – Componentes da *Render Farm*

A *render farm open source,* proposta neste trabalho, possui quatro componentes básicos:

- O sistema operacional no qual será implementada
- O motor de renderização das animações
- O aplicativo de controle de *jobs*
- E o aplicativo para unir as imagens renderizadas em um filme

3.1.1 – Sistema Operacional

A *render farm* precisa de um sistema operacional, que seja compatível com os demais componentes, para gerenciar o *hardware* no qual será instalada e prover os recursos de sistema e de rede necessários.

O sistema operacional escolhido para implementação do trabalho, é a distribuição GNU/Linux Debian versão 4.0r7 – Codinome Etch[1]. Sua escolha se deve a sua ampla aceitação e facilidade em obtenção de pacotes. Praticamente todos os componentes utilizados na implementação deste trabalho estão disponíveis na lista de repositórios padrão do Debian.

Além dos motivos enunciados acima, é uma distribuição que adota profundamente a política do *software* livre. E todos os componentes que serão utilizados são *open source* e livres.

[1] Debian: http://www.debian.org

3.1.2 – Motor de renderização - *Renderer*

Um dos componentes principais em uma *render farm*, é o motor de renderização ou *renderer*. É ele quem irá gerar as imagens renderizadas, ou seja, quem aplicará texturas, efeitos especiais e dará acabamento as imagens e animações criadas.

O motor de renderização não precisa ser o mesmo software que foi utilizado para criar e modelar a obra. Porém nesta *render farm* em específico, o *renderer* e o aplicativo de modelagem 3D são o mesmo aplicativo.

O Blender foi escolhido para ser o *renderer* nesta *render farm,* por ser uma suíte de criação de conteúdo 3D completa. E pelo fato de que ele pode ser utilizado como um *renderer* externo, por uma ferramenta de controle de jobs. Além de ser um aplicativo *open source* e livre, que atendará plenamente às necessidades deste projeto.

O Blender pode ser obtido no site do projeto ou pelos repositórios da Debian. Neste trabalho, foi utilizada a versão 2.42a, disponível nos repositórios da Debian

3.1.3 – Software de controle de *jobs*

Outro componente essencial a consecução deste projeto, é o *software* de controle de *jobs*. É ele quem irá distribuir o trabalho de renderização pelos computadores que compõem a *render farm.*

O motivo pela escolha do Drqueue, para a tarefa de gerenciar esta *render farm,* se deve ao fato de que é um software *open source* e livre. Que possui suporte a gerenciamento de vários *renderers* diferentes, incluindo o Blender e uma interface amigável para controle dos *jobs*.

A versão utilizada neste trabalho é a 0.64.3. A versão disponível no repositório da Debian tem o uso desencorajado. Por normalmente, ser bem mais antiga que versão disponível no site dos mantenedores do projeto do Drqueue.

3.1.4 – Conversor/codificador de vídeo

O produto final da distribuição do trabalho pela *render farm* não é um vídeo, mas uma sequência de arquivos de imagem, geralmente em formato jpeg. Como o objetivo de uma animação é um vídeo, e não um aglomerado de imagens. É necessário um ultimo aplicativo, para que a *render farm* esteja completa, o conversor de imagens em vídeo.

Para este fim foi escolhido o Mencoder. O mencoder é codificador de vídeo que possui suporte a vários *codecs* de áudio e vídeo diferentes. Além de possibilitar a transformação de um aglomerado de imagens em um vídeo. E é justamente essa funcionalidade que será utilizada neste trabalho.

O Mencoder faz parte do pacote do Mplayer e está disponível no site do projeto ou pelos repositórios da Debian. A versão utilizada neste trabalho é a 1.0-rc1svn20070225-0.3etch1, disponível no repositório debian-multimedia.

3.2 – Requisitos de sistema

O *hardware* necessário, para se implementar a *render farm* proposta neste trabalho, não precisa ser robusto. Basicamente qualquer computador que possua um processador de 450Mhz, 256 MB de RAM, disco rígido de 10 GB e uma interface de rede funcional, pode hospedar o servidor sem problemas. Porém é necessário ressaltar, que quanto mais numerosos e robustos forem os

nodos, mais poderosa será a *render farm*.

Seguem algumas considerações e especificações, a respeito dos componentes da *render farm.*

SISTEMA OPERACIONAL

O Linux é um sistema operacional conhecido por ser capaz de funcionar, basicamente, em qualquer ambiente computacional. Desde de *mainframes* até relógios. Portanto, a recomendação de *hardware* do início deste capítulo, já supre quaisquer questionamentos.

BLENDER

O Blender não exige uma máquina muito poderosa para funcionar. O poder de processamento da máquina influenciará nas respostas do sistema e no tempo de renderização.

As seguintes orientações constam no endereço <http://www.blender.org / features-gallery/requirements/> do projeto :

Configuração mínima

- 300 Mhz CPU
- 128 MB Ram
- 20 MB disco rígido livre
- 1024 x 768 px com 16 bit de cor
- Mouse de 3 botões (emulação do 3° botão possível)
- Placa de vídeo compatível com OpenGL e com 16 MB Ram

Configuração recomendada

- 2 Ghz de CPU de núcleo duplo
- 2 GB de Ram
- 1920 x 1200 px com 24 bit de cor

- Mouse de 3 botões
- Placa de vídeo compatível com OpenGL com 256 MB de Ram

Existe uma recomendação para que não sejam utilizadas placas de vídeo *on-board*, mesmo que compatíveis com OpenGL. Por possuírem desempenho e instruções OpenGL reduzidos. Porem esta recomendação, só se aplica aos nodos trabalhadores. Pode ser desconsiderada para o servidor, caso o mesmo não execute trabalho de renderização.

DRQUEUE

O Drqueue não faz exigências com relação ao equipamento a ser utilizado. O porte das máquinas utilizadas deverá ser baseado no tipo e no nível de detalhes do projeto que se busca concretizar.

Basicamente, pode-se utilizar o Drqueue em qualquer equipamento em que esteja instalado um sistema operacional compatível e exista uma conexão de rede funcional.

3.3 – Estrutura da R*ender Farm*

A *render farm* proposta neste trabalho utiliza a arquitetura cliente/servidor. A Figura 3.1 ilustra a estrutura da *render farm.* A máquina denominada *"RenderServer"* será o servidor da *render farm*. Ou seja, a máquina responsável por gerenciar e distribuir as tarefas de renderização. Também exercerá a função de servidor de arquivos. Exportando um diretório chamado *"/ usr/local/drqueue"*.

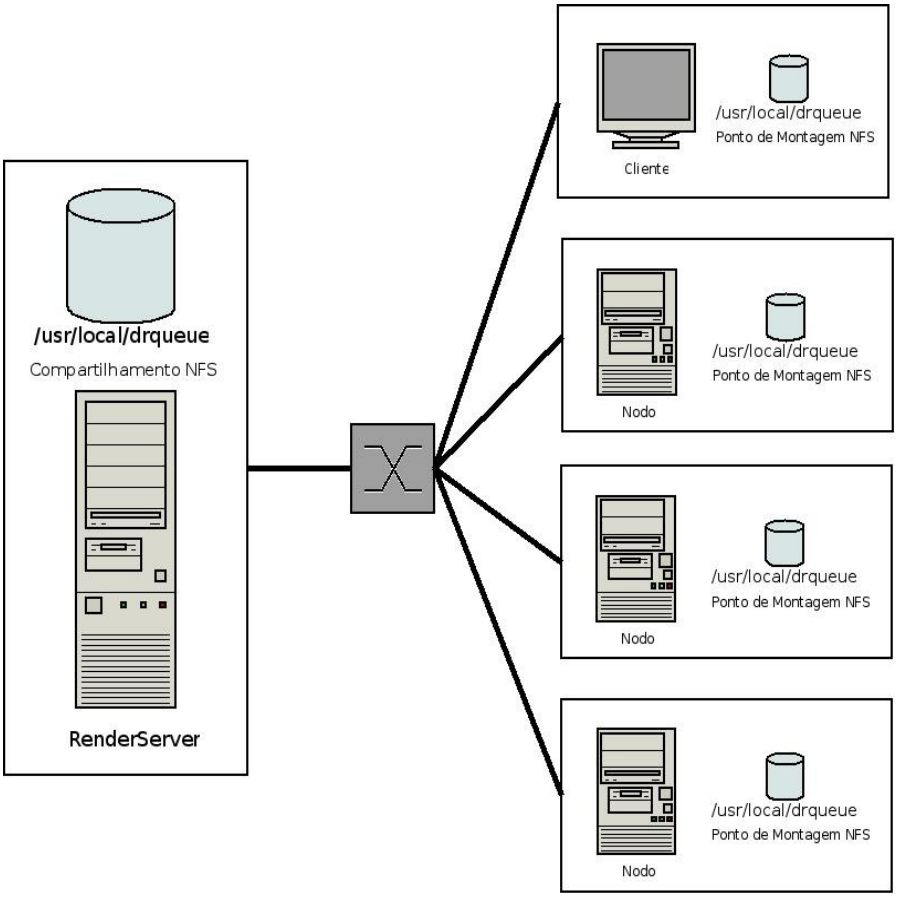

Figura 3.1: Estrutura da *render farm*

No RenderServer, será feita a instalação do GNU/Linux Debian, do Drqueue e do Mencoder. A instalação do Blender no servidor só será efetuada, caso deseje-se que o servidor exerça também a função de nodo trabalhador. Participando então do processo de renderização. O RenderServer desempenhará também a função de servidor de arquivos, compartilhado via NFS (*Network File System),* o diretório "*/usr/local/drqueue* ".

Neste diretório e em seus sub-níveis, estarão os binários, os arquivos de

configuração e os *scripts* do Drqueue. É também o local onde será armazenado o arquivo ".blend". Que é o arquivo gerado pelo Blender quando se salva um trabalho criado no mesmo. Este arquivo possui todos os dados relativos a animação. Nele existem informações sobre as texturas, sobre os objetos criados, sobre luzes, enfim. É ainda neste diretório que serão armazenadas as imagens já renderizadas. Ou seja, o produto da distribuição do trabalho, devolvido pelos nodos trabalhadores.

Os nodos são as máquinas que executarão o trabalho de renderização dos *jobs*. Neles devem estar instalados: o GNU/Linux Debian, o Blender e o pacote para montagem de compartilhamentos remotos NFS. Será feita a montagem do compartilhamento "*/usr/local/drqueue"*, com o mesmo nome exportado pelo servidor. Pois é a partir deste diretório que serão executados os *scripts* do Drqueue.

No cliente, máquina onde será utilizada a interface de gerenciamento do Drqueue (drqman*),* será instalado o GNU/Linux Debian e deve ser montado o compartilhamento "*/usr/local/drqueue"*. Também com o mesmo nome exportado pelo servidor. É pela máquina cliente, que se cria as tarefas e as remete para o RenderServer iniciar a distribuição do trabalho. A Tabela 3.1 apresenta um resumo das funções e componentes instalados em cada uma das máquinas.

3.3.1 – Considerações

Apesar da distinção, servidor/cliente/nodo, adotada por este projeto. Todas as máquinas envolvidas, podem executar trabalho de renderização, enviar e criar *jobs* e ainda trabalhar como servidor. Caso esta configuração, seja interessante para o cenário no qual será empregada a *render farm.* Bastando para isso, que executem o *scripts* pertinentes.

Tabela 3.1: Componentes instalados e funções de cada máquina

Máquina	Componentes instalados	Função
RenderServer	Drqueue, Debian, Mencoder e Blender (opcional)	- Distribuir e gerenciar o trabalho de renderização pela *render farm.* - Prover o compartilhamento NFS para acesso às funcionalidades da *render farm.*
Nodo(s)	Debian, Drqueue e Blender	- Executar o trabalho de renderização dos *frames* enviados pelo RenderServer.
Cliente(s)	Debian e Drqueue	- Criar e enviar as tarefas de renderização no RenderFarm pela interface drqman.

Esta *render farm* possui um ambiente homogêneo. Ou seja, todos os componentes da mesma utilizam o mesmo sistema operacional. Porem isto não é uma exigência. O Drqueue trabalha bem em ambientes heterogêneos. Onde existem máquinas com Linux, com Windows e até mesmo com Solaris. A principal alteração estrutural, em caso de ambiente heterogêneo, é o tipo de sistema de arquivos exportado pelo servidor. Que deve ser acessível a todos os componentes da *render farm.*

3.4 - Instalação

As instruções se aplicam aos componentes e sua forma de instalação, em todas as máquinas que compõem a *render farm*. Quando existirem divergências, nos procedimentos de uma máquina para a outra, estas serão apontadas.

3.4.1 – Debian

A única recomendação na instalação do Debian, é que na tela "Seleção de software" sejam marcadas pelo menos as opções:

- Ambiente Desktop – Para que as máquinas possuam um ambiente desktop e aplicativos.
- Servidor de arquivos – Para que as ferramentas e módulos de compartilhamento de arquivos já estejam disponíveis.
- Sistema Básico – Para a instalação das partes básicas do sistema.

3.4.2 – Blender

A instalação do software Blender, em um sistema GNU/Linux Debian, é efetuada, por meio da ferramenta de obtenção e resolução de dependências de pacotes, *apt-get*.

3.4.3 – Drqueue

A instalação do Drqueue, será feita pela compilação do seu código-fonte. Este pode ser obtido, em formato "tgz", na página do projeto. Neste trabalho, considera-se o *download* do arquivo "drqueue.0.64.3.tgz".

Antes da compilação e instalação do Drqueue, é necessário criar o diretório no qual ele será instalado. Neste trabalho, foi seguido o padrão

encontrado no site do projeto do Drqueue. Sendo assim, será utilizado o diretório "/usr/local/drqueue".

A compilação do Drqueue depende de um ambiente de desenvolvimento instalado (GCC, o G++, o pkg-config e o **libgtk2.0-dev) e da ferramenta de construção** *scons*. Todas as ferramentas necessárias à compilação do código-fonte do Drqueue, podem ser obtidas via ferramenta *apt-get.*

O comando a seguir, deve ser executado dentro do diretório, onde o arquivo "drqueue.0.64.3.tgz", foi descompactado. Este comando compila e depois instala o Drqueue, no diretório /usr/local/drqueue.

```
#scons PREFIX=/usr/local/drqueue install
```

Se a compilação e a instalação forem bem sucedidas, o Drqueue já estará instalado no sistema. A instalação do Drqueue, só deve ser efetuada no RenderServer. As demais máquinas, acessam as suas funcionalidades, pelo compartilhamento NFS.

3.4.4 – Mencoder

Para efetuar a instalação do mencoder no Debian, deve-se primeiramente inserir o repositório debian-multimedia na lista de fontes. Depois de configurado o repositório, é necessário instalar o aplicativo através da ferramenta *apt-get*.

3.5 – Configuração

A configuração da *render farm,* se resume basicamente, na configuração do compartilhamento, na preparação das variáveis de sistema e na configuração dos arquivos ".conf" do Drqueue.

3.5.1 – Compartilhamento NFS

O primeiro passo no processo de configuração da *render farm,* é ativar o compartilhamento de arquivos no servidor RenderServer. Como apresentado, no item 3.3 Estrutura da *render farm*, o servidor RenderServer deve compartilhar o diretório "/usr/local/drqueue". O compartilhamento deve ser feito utilizando o *Network File System.* A árvore de diretórios do Drqueue é dividida em cinco diretórios principais:

- bin – binários
- etc – configurações
- logs – arquivos de log
- tmp – arquivos temporários
- db – base de dados

Todos estes diretórios serão utilizados por um mais componentes do Drqueue. Sendo que as permissões dos mesmos diferem entre si. O diretório tmp deve possuir permissão de leitura e escrita para todos os computadores da *render farm.* O diretório logs, permissão de escrita para todos os computadores da *render farm.* Os diretórios etc e bin, devem permitir a leitura por todos os

integrantes da *farm*. E por fim o diretório db que deve estar acessível somente ao servidor.

Depois de compartilhado no servidor, o diretório "/usr/local/drqueue" deve ser montado em cada um dos nodos e no cliente da *render farm.* Exatamente com o mesmo nome. Ou seja, o diretório "/usr/local/drqueue" exportado no servidor, é acessado no cliente e nos nodos pelo mesmo caminho, "/usr/local/drqueue".

3.5.2 – Drqueue – variáveis e configurações

O Drqueue faz uso de duas variáveis de sistema. A variável "DRQUEUE_ROOT ", que armazena o caminho do diretório de trabalho do Drqueue. E a variável "DRQUEUE_MASTER", que armazena o nome ou o endereço IP do servidor. Sem estas variáveis, os *scripts* do diretório "/usr/local/drqueue/bin" não funcionam.

Para se configurar o Drqueue, é necessário deixar estas variáveis sempre disponíveis nas máquinas que compõem a *render farm.* Para isso, deve-se adiciona-las ao arquivo "/etc/profile".

No RenderServer, estas são as linhas, a serem adicionadas ao arquivo supra citado:

```
DRQUEUE_ROOT=/usr/local/drqueue
DRQUEUE_MASTER=localhost
export DRQUEUE_ROOT DRQUEUE_MASTER
```

E em cada nodo e no cliente, as linhas, seriam as seguintes:

```
DRQUEUE_ROOT=/usr/local/drqueue
DRQUEUE_MASTER=RenderServer
export DRQUEUE_ROOT DRQUEUE_MASTER
```

Se o *Domain Name Server* da rede local não estiver ativo, o nome do servidor deve ser substituído pelo seu endereço IP, na variável DRQUEUE_MASTER no cliente e nos nodos.

Uma alternativa à configuração do arquivo "/etc/profile", é o *script* Python, que fica no diretório "/usr/local/drqueue/bin", chamado "writedefaultenv.py". Que tem a função de automatizar a disponibilização das variáveis.

Este *script* gera um segundo *script*, chamado "default_env". Que quando executado, exporta as duas variáveis supra citadas com seus devidos valores. E adiciona o caminho "/usr/local/drqueue/bin" à variável PATH do sistema. A utilização deste método é opcional, pois pode não detectar corretamente as variáveis a serem exportadas. Caso isto aconteça, será necessário editar o arquivo "default_env" e corrigir os erros manualmente.

Além das variáveis, existem três arquivos que devem ser editados. Os arquivos de configuração:

- master.conf – servidor
- slave.conf - nodos
- drqman.conf - interface do cliente.

Que contém os caminhos para os diretórios do Drqueue. Estes arquivos se encontram no diretório "/usr/local/drqueue/etc". Para configurá-los, é necessário substituir as strings "/path/to/shared/" e "/path/to/local/", pelo caminho da instalação do Drqueue. Neste trabalho foi utilizado "/usr/local/drqueue/".

Neste ponto, pode-se definir configurações globais e também configurações individualizadas. Bastando colocar os caminhos para os arquivos de configuração em diretórios locais. Para simplificar este projeto, todas as configurações serão globais e estarão no diretório previamente citado. Esta observação também é válida, para os arquivos "/etc/drqueue/master.conf" e "/etc/drqueue/drqman.conf". Que podem ser configurados localmente. A Figura 3.2 apresenta o arquivo "master.conf", já configurado. Os demais arquivos seguem a mesma estrutura.

```
#
#
# Lines starting with '#' will not be parsed
# Use the following as examples (without the '#' character)
#
logs=/usr/local/drqueue/logs
tmp=/usr/local/drqueue/tmp
db=/usr/local/drqueue/db
bin=/usr/local/drqueue/bin
etc=/usr/local/drqueue/etc
```

Figura 3.2: Exemplo do arquivo master.conf

O arquivo "slave.conf", possui a opção de determinar o nome do *pool*, ou seja, do grupo ao qual aquele nodo pertence. Neste trabalho, convencionou-se que todos os nodos fariam parte do *pool* "drqueue". Para alterar este campo, basta adicionar ou modificar o valor, após o sinal de igualdade:

pool=Default

Com as variáveis ativas no sistema e os arquivos ".conf" modificados, o Drqueue já esta instalado e funcional.

3.6 - Utilização da *render farm*

3.6.1 – Inicializando o servidor

Para que o servidor comece a aceitar conexões, deve-se executar o seguinte comando, em um terminal:

```
$/usr/local/drqueue/bin/master.Linux.i686
```

Algumas mensagens serão exibidas na tela, informando o status do servidor. Se a string “Waiting for connections...” aparecer no final da listagem, o servidor está operacional.

3.6.2 – Inicializando os nodos

Para inicializar os nodos, é utilizado o seguinte comando em um terminal:

```
$/usr/local/drqueue/slave.Linux.i686
```

A Figura 3.3 apresenta um exemplo de saída de dados, após a inicialização de um nodo. No inicio da mensagem, percebe-se uma notificação, de que não foi possível encontrar o arquivo '/etc/drqueue/slave.conf'. Esta mensagem não é um problema. Na linha seguinte, o Drqueue notifica que está

recebendo as configurações do arquivo "/usr/local/drqueue/etc" previamente configurado.

```
Could not open config file: '/etc/drqueue/slave.conf'
Parsing config at: /usr/local/drqueue/etc/slave.conf
Logs on: '/usr/local/drqueue/logs'
Tmp on: '/usr/local/drqueue/tmp'
Pools are: 'drqueue'
HWINFO Report
Name:                   DQserver
Architecture:           Intel
OS:                     Linux
Processor type:         UNKNOWN
64bit-32bit cpu:        32bit
Processor speed:        1998 MHz
Number of processors:   1
Memory:                 1012 Mbytes
Working silently...
```

Figura 3.3: Exemplo de um nodo em funcionamento

Neste trabalho, para simplificar o processo, foi preferida a configuração diretamente no compartilhamento. Os arquivos de configuração utilizados estão no diretório "/usr/local/drqueue/etc". O que permite centralizar as configurações e arquivos de *log*. Desta forma todos os integrantes da *render farm,* indiferentemente da função exercida por aquela máquina, possuem os mesmos arquivos de configuração.

3.6.3 – Interface do cliente - drqman

Para abrir o Drqueue Manager, ou drqman, utiliza-se o seguinte comando, em um terminal:

$/usr/local/drqueue/bin/drqman.Linux.i686

Na Figura 3.4 pode-se ver o drqman em funcionamento, em uma *render farm open source.* A *render farm* é composta de um servidor, que também renderiza imagens, e de mais dois nodos.

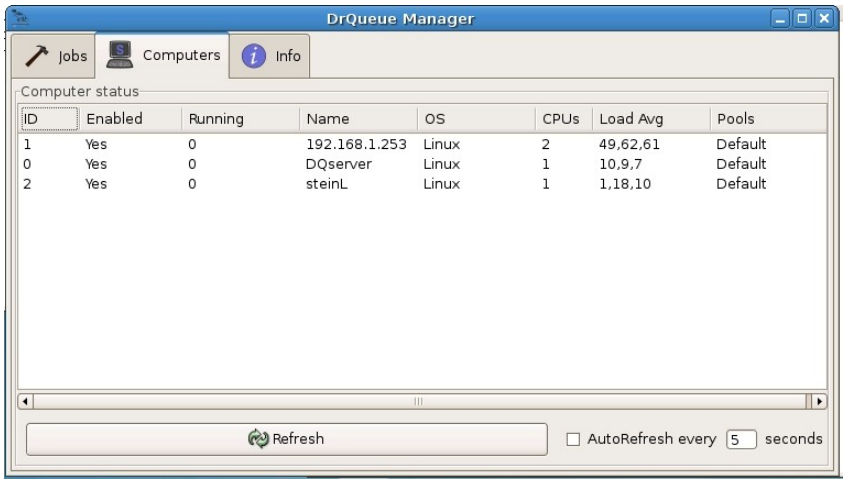

Figura 3.4: drqman em funcionamento.

Com o drqman em funcionamento, pode-se começar a submissão de *jobs*. A criação de *jobs* é executada pela aba "*jobs*", do drqman.

Com um clique do botão direito do mouse, em qualquer área da janela, será apresentado um menu. Ao clicar-se em "*New job*", uma nova tela com os campos para preenchimento será apresentada. Deve-se escolher o tipo de *job*, no caso deste trabalho, o tipo de *job* é "Blender". A Figura 3.5 apresenta a tela para criação de *jobs* para o Blender. Preenchidos os dados necessários, basta clicar em "submit" para remeter o *job* para a renderização.

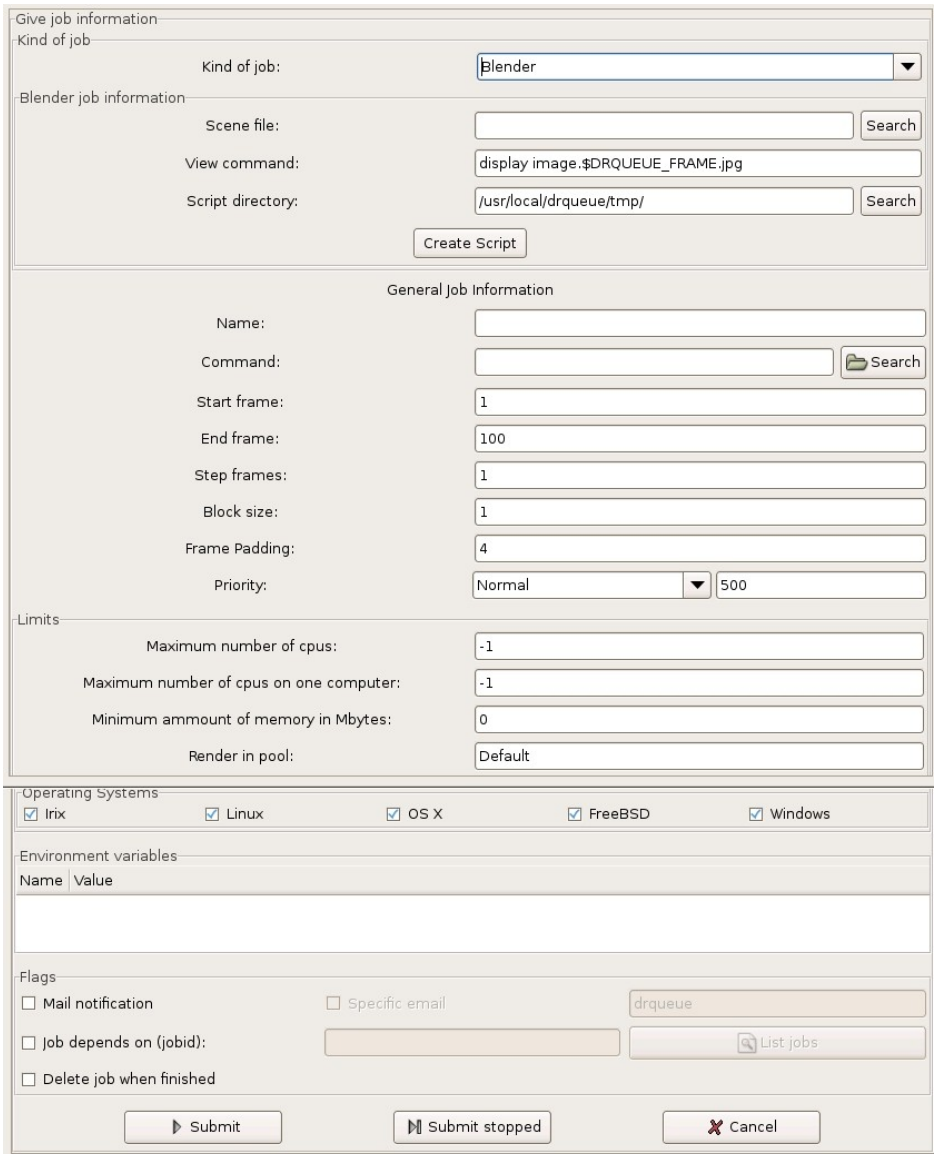

Figura 3.5: Tela de criação de jobs para o Blender

É importante salientar, que o arquivo ".blend", deve estar em algum subdiretório, dentro do diretório "*/usr/local/drqueue/*". Para que o mesmo esteja acessível à todos os nodos. Neste trabalho, sugere-se a criação de um diretório

chamado “blend”. A Figura 3.6, apresenta um *job* em processo de renderização.

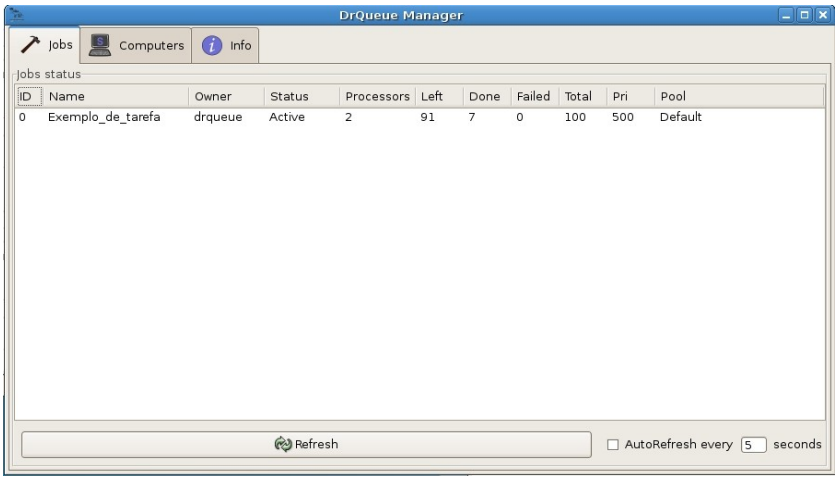

Figura 3.6: Job em execução

3.6.4 – Criação do vídeo com as imagens geradas - mencoder

Como apresentado anteriormente, o produto final da *render farm* criada, é um conjunto de imagens. Após a distribuição do trabalho, as imagens estarão no diretório “/usr/local/drqueue/tmp”.

Os comandos a seguir, retirados de [Mplayer, 2009], concatenam essas imagens e as codificam em um vídeo.

Para criar um Motion JPEG (MJPEG) de todos os arquivos no diretório corrente, pode ser utilizado:

```
$mencoder mf://*.jpg -mf w=800:h=600:fps=25:type=jpg -ovc
copy -oac copy -o output.avi
```

Para criar um filme no formato MPEG-4 de todos os arquivos no diretório corrente, pode ser utilizado:

```
$mencoder mf://*.jpg -mf w=800:h=600:fps=25:type=jpg -ovc lavc
```

-lavcopts vcodec=mpeg4:mbd=2:trell -oac copy -o output.avi

Em [Mplayer, 2009] podem ser encontrados mais detalhes, sobre a utilização do mencoder.

Capítulo 4

Testes e resultados

Este capítulo apresenta os testes realizados e os resultados obtidos.

4.1 - Descrição do ambiente de testes

Neste trabalho, para a execução dos testes, a *render farm* foi estruturada segundo a Figura 3.1 do Capítulo 3. No Teste #1 foram utilizadas três máquinas reais e uma máquina virtual como servidor. As configurações dos equipamentos envolvidos são as seguintes:

- steinL – *Desktop*, Pentium IV 1.8 Ghz, 1 GB de memória RAM, *hard-disk* de 160 GB, vídeo GeForce4 MX4000 e sistema operacional Debian 5 (lenny).
- vostro – *Notebook*, Core 2 Duo T7250 2.0 Ghz, 4 GB de memória RAM, *hard-disk* 250 GB, vídeo GeForce 8400M GS e sistema operacional Debian 5 (lenny).
- matrixL – *Desktop*, Core Duo E4600 2.4 Ghz, 2 GB de memória RAM, *hard-disk* 250 GB, vídeo GeForce 8400 GS e sistema operacional Debian 4 (etch). Hospedeiro da máquina virtual DrqueueServer.

No Teste #2 foram utilizadas duas máquinas reais e uma máquina virtual como servidor. As configurações dos equipamentos envolvidos são as seguintes:

- vostro – (configuração previamente mencionada)
- steinL – (configuração previamente mencionada)
- 3i3i – *Notebook*, Core 2 T5300 1.73 Ghz, 1 GB de memória RAM, *hard-disk* 160 GB, vídeo GeForce 7300M GS e sistema operacional Ubuntu 7.10.

Uma das características de uma *render farm* é que não existe a necessidade de se deixar máquinas dedicadas. Durantes os testes efetuados as máquinas envolvidas estavam sendo utilizadas em tarefas comuns, como navegar na internet, editar textos, utilizar comunicadores instantâneos e etc. Mesmo o hospedeiro da máquina virtual estava realizando tarefas corriqueiras durante todos os testes.

4.2 - Animações utilizadas

Foram utilizadas três animações com 96 *frames* a 24 FPS, totalizando quatro segundos de duração para cada animação. A animação entitulada "Cube" (Figura 4.1) é uma réplica da animação descrita no tutorial de nível básico encontrada em [olivS, 2005]. A animação "Cube, spheres and monkeys" (Figura 4.2) é uma adaptação da animação "Cube" onde são adicionadas quatro esferas nas extremidades da base da animação e duas "Susanne" em faces opostas do cubo. E a animação "Cube and a lot of stuff" (Figura 4.3) adiciona mais quatro esferas e coloca a mascote "Susanne" em todas as faces do cubo. Os objetos adicionados na segunda e na terceira animação, estão utilizando a ferramenta *subsurface* e o filtro Catmull-Clark para suavizar as suas formas e aumentar a demanda de processamento.

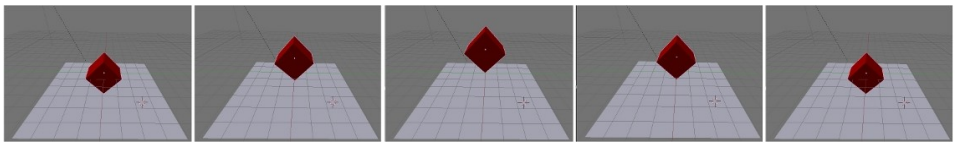

Figura 4.1: “Cube”

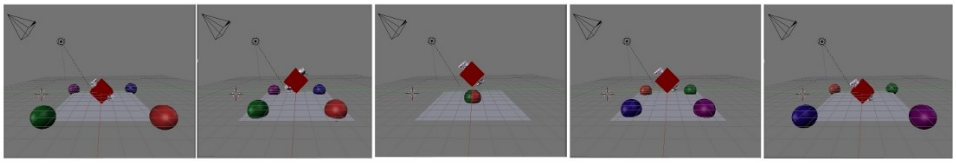

Figura 4.2: “Cube, spheres and monkeys”

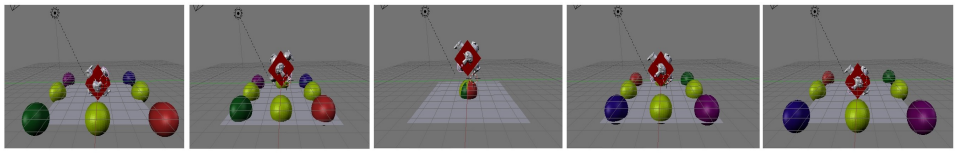

Figura 4.3: “Cubes and a lot of stuff”

4.3 – Resultados obtidos

No Teste #1 foi efetuado um comparativo entre o tempo gasto para renderização das animações, utilizando a *render farm* com todos os computadores trabalhando e utilizando somente o *notebook* vostro. O comparativo foi efetuado para as três animações citadas. Na Tabela 4.1 são apresentados cinco amostras de tempos obtidos na renderização das animações propostas.

Tabela 4.1: Tempo de renderização das animações – Teste #1

Método de renderização	Tempos de renderização - "Cube"					Tempo Médio
vostro	6'00"	6'10"	5'59"	5'52"	6'03"	6'01"
Render farm	3'58"	3'34"	3'31"	3'17"	3'37"	3'35"
Método de renderização	Tempos de renderização - "Cube, spheres and monkeys"					Tempo Médio
vostro	9'21"	10'30"	10'41"	9'31	9'34"	10'17"
Render Farm	4'18"	4'15"	4'31"	4'10"	4'23"	4'19"
Método de renderização	Tempos de renderização - "Cube and a lot of stuff"					Tempo Médio
vostro	14'20"	15'32"	14'27"	14'39"	14'55"	14'53"
Render Farm	5'57"	6'01"	5'04"	5'49"	6'10"	5'53"

As Figuras 4.4, 4.5 e 4.6, apresentam as cenas renderizadas das animações propostas "Cube", "Cube, spheres and monkeys" e "Cube and a lot of stuff", respectivamente.

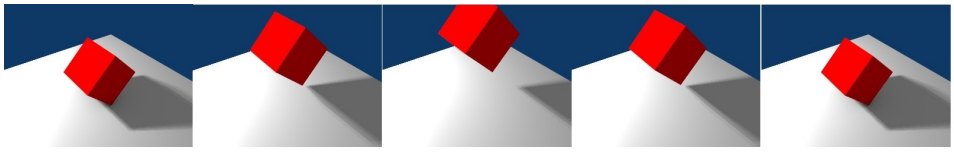

Figura 4.4: "Cube" renderizado

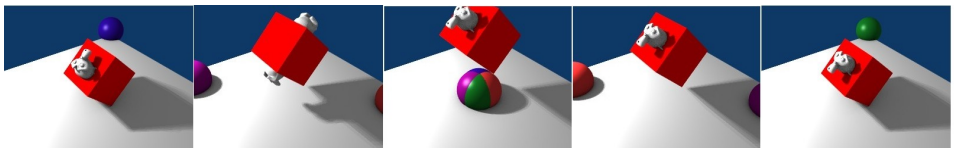

Figura 4.5: "Cube, spheres and monkeys" renderizado

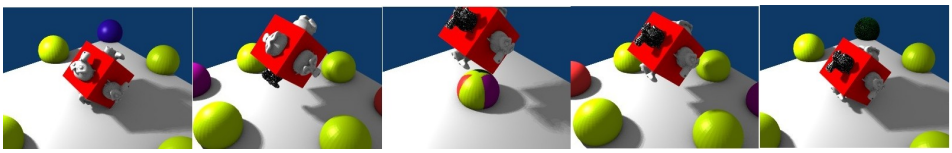

Figura 4.6: "Cube and a lot of stuff" renderizado

Tomando como referência os tempos médios obtidos para a renderização das três animações pode-se observar nitidamente que a diferença de tempo, entre a renderização em uma máquina e a renderização utilizando a *render farm,* aumenta a medida em que a complexidade da tarefa aumenta.

Na renderização da animação "Cube" a diferença no tempo de renderização do vostro para o tempo de renderização da *render farm* é de aproximadamente 45%, sendo que esta animação consiste em apenas um plano e um cubo em rotação. Apesar de ser um ganho considerável, o objetivo da *render farm* é propiciar um ganho mais significativo. Na segunda animação "Cube, spheres and monkeys" a diferença de tempo encontrada é de aproximadamente 60%. Tendo em vista que são utilizados três computadores na *render farm* e que houve uma modificação significativa na complexidade da animação, o ganho obtido no tempo de renderização é um valor que pode ser considerado aceitável. Na renderização da terceira animação "Cube and a lot of stuff" o ganho foi de quase 70%. A animação quase dobrou o seu nível de complexidade em relação a segunda animação, pois conta com o dobro de esferas e o triplo de "Sussane". O ganho de tempo alcançado na terceira animação é um valor que se espera alcançar, quando se triplica o numero de máquinas que efetuarão trabalho de renderização.

O objetivo do Teste #2 é comparar o desempenho de uma única máquina, com grande poder de processamento, contra o desempenho da *render farm.* Para isso foi utilizado o recurso de limitação do *clock* dos computadores que pertencem a *render farm*. O intuito da limitação de *clock* é simular uma

render farm com quatro máquinas de pequeno porte ("vostro" e "3i3i", ambos de núcleo duplo, com *clocks* limitados a 800Mhz). O computador com grande poder de processamento utilizado foi o "matrixL" com o *clock* de 2,4 Ghz. As animações utilizadas no Teste #2 são as mesmas utilizadas no Teste #1. A Tabela 4.2 apresenta os resultados obtidos.

Tabela 4.2: Tempo de renderização das animações – Teste #2

Método de renderização	Tempos de renderização - "Cube"					Tempo Médio
matrixL	5'39"	6'10"	5'29"	5'22"	6'03"	5'56"
Render farm	4'59"	7'03"	4'12"	9'07"	5'07"	7'39"
Método de renderização	Tempos de renderização - "Cube, spheres and monkeys"					Tempo Médio
matrixL	9'11"	10'40"	10'01"	9'11	9'14"	10'03"
Render Farm	6'14"	6'37"	6'58"	5'18"	6'20"	6'32"
Método de renderização	Tempos de renderização - "Cube and a lot of stuff"					Tempo Médio
matrixL	13'50"	14'32"	14'37"	14'29"	14'15"	14'13"
Render Farm	6'03"	6'43"	7'21"	5'16"	6'30"	6'48"

No Teste #2 foi possível perceber que mesmo quando a somatória dos *clocks* das máquinas que pertencem a *render farm* é correspondente ao *clock* de uma única máquina mais poderosa, o desempenho da *render farm* na renderização ainda é mais satisfatório.

Levando em conta os resultados obtidos, é facilmente perceptível que o emprego ou não de uma *render farm* em um projeto, está diretamente ligado ao nível de complexidade do mesmo. Ou seja, quanto mais detalhes e mais poder de processamento for requerido para um projeto, mais é indicado o uso de uma *render farm* com várias máquinas para diminuir o tempo gasto para renderiza-lo.

Capítulo 5

Conclusão

O objetivo deste trabalho foi apresentar uma solução aos problemas de demanda de poder computacional e da demora na renderização, em projetos de animações 3D. A solução apresentada foi a criação de uma *render farm open source.*

Esta *render farm* utiliza o GNU/Linux Debian como sistema operacional e é composta pelos aplicativos Blender e Drqueue. O Blender como uma ferramenta gratuita e *open source*, tanto para a criação e edição de conteúdo 3D, quanto como um motor no processo de renderização das imagens. E o Drqueue na função de distribuir e gerenciar o trabalho de renderização dentro da *render farm.*

A interface do Drqueue para gerenciamento das tarefas é realmente amigável e permite o acompanhamento do status da *render* farm e a intervenção do usuário sempre que necessário.

A infra-estrutura necessária para a implementação da *render farm*, não demanda investimentos volumosos em equipamentos. Podendo ser implementada mesmo em máquinas consideradas obsoletas no mercado atual. Situação esta, que permite que o projeto seja implementado por um público muito mais heterogêneo.

Outro ponto positivo que pode ser observado, é a facilidade de criação

de uma *render farm* heterogênea. Tanto a nível de sistema operacional, quanto a nível de motor de renderização. Depois de implementada, a *render farm* pode renderizar trabalhos de vários *renderers* diferentes e em vários ambientes diferentes.

A instalação e configuração da *render farm*, não se mostraram fases trabalhosas. A maioria dos aplicativos, foram instalados sem problemas pela ferramenta de gerência de pacotes "apt-get". E as configurações que foram feitas manualmente, não podem ser consideradas complexas, para um usuário intermediário.

Devido aos resultados obtidos com os testes, percebe-se que com animações muito simples o ganho de tempo, com a utilização da *render farm,* não é tão acentuado. Porém foi possível perceber que a medida em que as animações se tornam mais complexas a diferença no tempo gasto para a renderização de um *job,* de uma só estação para o tempo de renderização obtido com a *render farm,* se torna muito grande. Dados estes resultados, o emprego da *render farm* em projetos complexos é indicado como real solução para a redução no tempo de renderização de animações e filmes em 3D.

A desvantagem encontrada no projeto é o fato de que o produto da distribuição do trabalho de renderização, não é um vídeo pronto para visualização. Mas sim um aglomerado de imagens, geralmente em formato jpeg. Porém este contra-tempo pode ser facilmente remediado com a utilização do mencoder para criar o vídeo almejado.

Ao final do projeto é possível afirmar que o resultado obtido foi satisfatório e que a *render farm open source* apresentada atende plenamente aos objetivos definidos.

Ao longo do trabalho algumas idéias para projetos futuros foram vislumbradas. Dentre elas, as seguintes são consideradas de maior relevância:

- A criação de um *script* ou módulo para que ao final do processo

de renderização, já esteja disponível um vídeo do trabalho executado e não o aglomerado de imagens.

- O Drqueue lida apenas com *frame rendering*, que é a renderização de um *frame* por vez em cada processador. A implementação de um *script* ou módulo que permita *bucket rendering* , que é a renderização de várias partes da mesma imagem, distribuídas pela *render farm,* é na opinião do autor de grande relevância.

Referências Bibliográficas

Andauer, Claudio

Blender Documentation Volume I – User Guide/ Claudio Andauer, Manuel Bastioni, *et all.* [on-line], Ultima modificação, 01 Setembro 2004. Copyright by Stichting Blender Foundation. Disponível em <http://www.blender.org/documentation/htmlI/book1.html>. Acessado em 11/03/2009

Bookman, Charles

Agrupamento de computadores em Linux / Charles Bookman – Rio de Janeiro : Editora Ciência Moderna Ltda., 2003.

Couloris, George

Sistemas distribuídos : conceitos e projeto/ George Colouris, Jean Dolimore, Tim Kinderberg ; Tradução João Tortello – 4. ed. - Porto Alegre : Bookman, 2007

Deitel, H.M.

Sistemas operacionais: terceira edição / H. M. Deitel, P. J. Deitel, D. R. Choffnes – São Paulo : Pearson Prentice Hall, 2005.

Dos Anjos, Thiago Curvelo

OurBlender: Um Grid Livre para Renderização Distribuída de Animações 3D. Webmidia – Artigos Resumidos, Págs. 1 – 3. 2006. Disponível em: <http://www.lavid.ufpb.br/~curvelo/papers/ourblender_-_webmedia2006 .pdf>. Acessado em 12/03/2009

Drqueue

Drqueue Wiki Documentation. Disponível em: <https://ssl.drqueue.org/project/wiki/Documentation>. Acessado em 12/03/2009.

Greve, Georg C. F.

BRAVE GNU WORLD, Linux-Magazine,USA, Edição 52, págs.94 - 95, Mar. 2005. Disponível em: <https://www.linuxmagazine.com/issue/52 /Brave_GNU_World.pdf>. Acessado em 12/03/2009.

Gonzáles-Morcillo, Carlos

3D Distributed Rendering and Optimization using Free Software/ Carlos Gonzáles-Morcillo, Gerhard Weiss, David Vallejo, *et al.* - Escuela Superior de Informática, University of Castilla - La Mancha Paseo de la Universidade - Ciudad Real. Pág. 52-66 - 2007 – Espanha. Disponível em <http://grid.ntu-kpi.kiev.ua/files/3d_rendering.pdf>. Acessado em 12/03/2009

Kreusel, Peter

Curvas Suaves – Modelagem 3D com o Blender, Linux-Magazine,Brasil,Edição 18, Págs. 60 – 66, Mar. 2006. Disponível em: <http://www.linuxnewmedia.com.br/article/curvas_suaves>. Acessado em 12/03/2009.

Mplayer

Mplayer - The Movie Player: Manual page for MPlayer and Mencoder, Copyright 2000-2009 MPlayer team. Disponível em: <http://www.mplayerhq.hu/DOCS/HTML/en/index.html>. Acessado em 11/03/2009.

Mullen, Tony

Introducing character animation with Blender/ Tony Mullen – 1. ed.-Indianápolis, Indiana: Wiley Publishing, Inc., 2007

olivS

Tutorial: Simple animations/ olivS - Blender Tutorials – Feeblemind. Março de 2008. Disponível em: <*http://www.blender*.org/education-help/tutorials/animation/>. Acessado em 09/04/2009.