Trabalho de Conclusão de Curso

Visualização Científica Remota: Um Estudo de Caso com o Cluster GradeBR/UFAL

Marco Antonio de Albuquerque Silva
marcoantonio.maas@gmail.com

Orientador:

Leonardo Viana Pereira

Maceió, Fevereiro de 2012

Marco Antonio de Albuquerque Silva

Visualização Científica Remota: Um Estudo de Caso com o Cluster GradeBR/UFAL

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Leonardo Viana Pereira

Maceió, Fevereiro de 2012

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Leonardo Viana Pereira - Orientador
Instituto de Computação
Universidade Federal de Alagoas

Alejandro César Frery Orgambide - Examinador
Instituto de Computação
Universidade Federal de Alagoas

William Wagner Matos Lira - Examinador
Centro de Tecnologia
Universidade Federal de Alagoas

Maceió, Fevereiro de 2012

Resumo

Este documento apresenta uma solução para a visualização remota de grandes volumes de dados gerados em modelagem computacional e simulação em um *cluster* em teraescala. Esta solução faz uso das Placas Gráficas de Propósito Geral (GPGPUs) presentes no cluster e do protocolo VNC. Isto possibilita a visualização remota de terabytes de dados em terminais com pequena capacidade, *tablets* ou *smartphones*. Esta tecnologia foi demonstrada com sucesso em um estudo de caso real.

Abstract

This document presents a solution for remote visualization of large volumes of data generated by computational modeling and simulation in a terascale cluster. It makes use the cluster's General-Purpose computing on Graphics Processing Units (GPGPUs) and VNC protocol. This allows remote visualization of terabytes of data on low capacity terminals, tablets or smartphones. This technology was succesfully implemented in a real case study.

Agradecimentos

Agradeço a Maria Eduarda, minha mãe, por todo esforço que fez para que eu e minha irmã tivessemos uma boa educação, além de todo carinho e paciência.

Agradeço a minha irmã e eterna amiga, Giulianna, por todo carinho e por todas as resenhas durante toda infância até os dias de hoje, e por todas as brigas que eram resolvidas cinco minutos depois.

Agradeço a Leonardo Viana, meu orientador, por ter me dado a oportunidade de participar de um projeto de iniciação científica, pelos puxões de orelhas, pelos conselhos, que não eram restritos somente à área acadêmica e por toda paciência e compreensão.

Aos meus amigos e futuros doutores, Rian e Ivan, com os quais, reza a lenda, fiz a maioria dos trabalhos em grupo da graduação.

Ao meu amigo e quase xará, Marcos Antônio, mais conhecido como Pereira, por sempre ter feito os trabalhos da graduação e compartilhado com a comunidade, pela força e incentivo na graduação, pesquisa, mercado de trabalho, além das caronas para as festas da turma.

Aos meus amigos: Alisson, Alexandre, André, Danilo, Evellyn, Jônathas, Lucas, Paulo, Tamer, Vilker e Wylken, com os quais convivi quase que diariamente durante a graduação.

Agradeço a ANP, a Petrobrás e ao Laboratório de Computação Científica e Visualização da Universidade Federal de Alagoas, especialmente a Baltazar Taveres, pelo suporte e por garantir acesso aos recursos computacionais do cluster GradeBR/UFAL da Rede Galileu.

Marco Antonio

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

Um dos problemas centrais da computação científica é a geração de uma enorme massa de dados. Através de simulações ou experiências são obtidos geralmente um grande volume de dados numéricos. Quando são utilizadas técnicas de modelagem computacional e sensoriamento de fenômenos dinâmicos em sistemas complexos naturais ou artificiais, aliadas a técnicas de computação de alto desempenho, o volume de dados a ser tratado nestes problemas atualmente pode ser da ordem de petabytes (Wu et al., 2009a).

A transformação de dados em forma visual, usualmente 3D com componente dinâmico (tempo ou animação), torna sua análise humanamente factível. Analisar, tirar as informações relevantes, destes dados, muitas vezes associados a fenômenos dinâmicos em três ou mais dimensões, faz parte do escopo da *visualização científica* (Friendly and Denis, 2001). Em projetos de larga escala a armazenagem, transferência e organização destes dados, por si só, se tornam problemas de difícil solução.

A Petrobras financiou, através da Rede de Computação Científica e Visualização – Rede Galileu, a criação do cluster GradeBR/UFAL no Laboratório de Computação Científica e Visualização (LCCV). Uma das metas desta Rede é criar um ambiente de grid de alto desempenho continental, a GradeBR, com desempenho teórico máximo (Rpeak) de 160 Teraflops, capaz de fornecer com eficiência capacidade de processamento para problemas de engenharia computacional aplicados à exploração de óleo e gás. Um dos objetivos também é criar um sistema em teraescala de visualização 3D em tempo real para problemas de mecânica computacional aplicados (Oliveira et al., 2010).

Neste trabalho é apresentado e analisado um protótipo para a visualização remota de grandes volumes de dados gerados em modelagem computacional e simulação no cluster GradeBR/UFAL do Laboratório de Computação Científica e Visualização. Esse protótipo foi desenvolvido integrando tecnologias multiplataformas e de código aberto, experimentando

um pipeline diferente do usado nos grandes centros de pesquisa. Tal pipeline foi motivado pelos recursos e limitações encontrados no ambiente de testes.

2

Visualização Científica

A visualização fotorrealista e interativa é uma ferramenta importante na investigação científica, pois permite que os cientistas façam descobertas visuais em tempo real (Ge et al., 2010).

2.1 Visualização Remota

A moderna ciência computacional está produzindo cada vez mais dados em larga escala, utilizando-se de recursos de alto desempenho computacional (HPC), que geralmente estão distantes do usuário, o que torna a visualização e análise interativa desses dados em uma tarefa desafiadora (Ge et al., 2010).

Existem sistemas de visualização paralela, tais como ParaView (Cedilnik et al., 2006) e VisIt (Childs et al., 2005) que foram projetados para suportar visualização de grande volume de dados, mas que exigem grande poder de processamento. Sendo assim, tais sistemas são utilizados em centros de computação, e mesmo com os dados localizados à distância e possuindo dezenas de gigabytes ou mais, há necessidade de interatividade satisfatória e desempenho no que se diz respeito a visualização destes (Ge et al., 2010).

Uma opção para visualizar esses dados é replicá-los, levando-os para o usuário. No entanto, a replicação de dados é complexa e em muitos casos, o poder de processamento em desktops é inadequado para sistemas de visualização, além do enorme tamanho dos dados o que por si só pode tornar até a transferência inviável. Então se faz necessário o uso de alguma solução que permita a visualização a distância através de computares comuns.

O processo de visualização geralmente consiste em alguns passos, que são chamados de *pipeline* de visualização. Em sistemas de visualização remota é necessário um pipeline que dê suporte a técnicas de visualização sobre redes de longa distância (WANs) (Wu et al., 2009b).

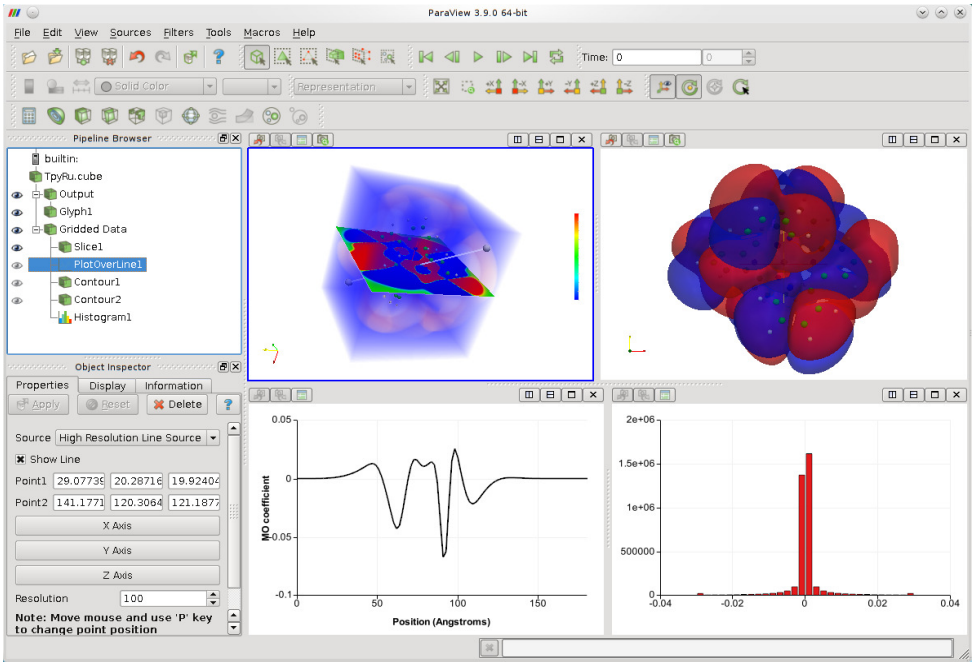


Figura 2.1: Paraview (Cedilnik et al., 2006).

2.2 Visualização em Ultra-Resolução

Muitas imagens em ultra resolução estão sendo obtidas em ritmo acelerado pela ciência e muitos domínios da engenharia, tais como engenharia biomédica, astrofísica, ciências da terra, como bem como ciências sociais e humanas. Com essa avalanche de dados, novos desafios relacionados a visualização surgem. Sistemas convencionais de exibição em tela única impõe limitações a nossa capacidade de analisar esses conjuntos de dados, como os pesquisadores são obrigados a navegar por um espaço enorme usando apenas uma janela minúscula. A situação torna-se progressivamente pior se os pesquisadores pretendem analisar grandes conjuntos de dados, tais como uma coleções de imagens de alta-resolução, cada uma mostrando determinados atributos para o problema estudado, com o objetivo de descobrir as relações entre eles.

Em resposta a estas limitações, os ambientes de tela convencional estão sendo substituídos por sistemas de visualização distribuída, ou seja, em paredes de vídeo formadas por vários monitores, que fornecem ordens de magnitude maiores para as resoluções e capacidade computacional. HIPerSpace (Kuester et al.), por exemplo, pode exibir imagens de até 286 megapixels de resolução. Quando combinado com um middleware adequado, projetado para suportar gráficos distribuídos para alto desempenho, sistemas de visualização distribuída podem fornecer um espaço de trabalho grande e unificado que permite a visualização interativa de imagens mais próximas de suas resoluções nativas, facilitando a análise dos dados em equipe (Yamaoka et al., 2010).

2.3 SAGE: Scalable Adaptive Graphics Environment

Há diversos trabalhos no que diz respeito a visualização remota em alta-resolução, entre esses o SAGE (Jeong et al., 2005) se destaca.

O SAGE (Scalable Adaptive Graphics Environment) é uma arquitetura de computação distribuida que dá suporte a renderização em paralelo e à exibição distribuída em ultra-resolução. Ele aborda a necessidade de apoiar a heterogeneidade e escalabilidade, dissociando renderização e a exibição, porém necessita de uma altíssima largura de banda de rede, para fazer a ponte entre eles. No SAGE, os trabalhos de visualização chamado por um usuário são interpretados por um Broker e expedidos para o recurso de visualização ideal disponível na rede. O sistema escolhido executa a renderização e envia as imagens resultantes para exibição através de streams de vídeo. A meta de adaptatividade SAGE é garantir que esta decisão seja feita de forma transparente.

A atual implementação do SAGE pode executar várias instâncias de aplicações de visualização diferentes e dinamicamente enviar o fluxo de pixels dos aplicativos para exibição em uma parede de vídeo, e as múltiplas janelas de aplicativos podem ser movidas ou redimensionadas livremente entre as várias telas. A maioria dos aplicativos de visualização pode ser facilmente integrados ao framework do SAGE com apenas a adição de algumas linhas de código (Jeong et al., 2005).

Figura 2.2: Parede de Vídeo Gerada pelo SAGE (Eletronic Visualização Laboratory)

Aplicações Suportadas pelo SAGE

Segundo a documentação online do SAGE (Eletronic Visualização Laboratory) atualmente ele dá suporte à algumas aplicações como:

- ImageViewer: uma aplicação de visualização de imagens baseado no toolkit ImageMagick. Suporta a maioria dos formatos de imagem comumente usados (jpg, png, gif, tiff ...)

- Mplayer: MPlayer é um reprodutor de filmes que roda em vários sistemas. Ele suporta os formatos de filmes mais comumente usados. Um plugin de saída para SAGE foi escrito para fornecer streaming de vídeo.

- VLC: VLC Media Player é um player multimídia altamente portátil para vários formatos audio e vídeo (MPEG-1, MPEG-2, MPEG-4, DivX, mp3, Ogg), bem como DVDs, VCDs e vários protocolos streaming. Também pode ser usado como um servidor de fluxo em unicast ou multicast em IPv4 ou IPv6 em uma rede de banda larga. Um plugin de saída para SAGE foi escrito para fornecer streaming de vídeo.

- iHDTV: iHDTV foi originalmente desenvolvido pela ResearchChannel juntamente com a Universidade de Washington, e demonstrado pela primeira vez em 1999. É um conjunto de software composto por módulos que trabalham com componentes disponíveis comercialmente para capturar, empacotar e transportar vídeo de alta definição em vários formatos através da rede, abrange a faixa de níveis de qualidade HDTV.

- VNCViewer: Virtual Network Computing (VNC) sistema de compartilhamento de área de trabalho gráfica usa o protocolo RFB para controlar remotamente outro computador. Foi desenvolvido um protocolo cliente VNC que permite ao usuário levar conteúdo de desktop para o ambiente SAGE. Esse vncviewer é um programa visualizador de VNC modificado que serve como um proxy entre um servidor VNC e o SAGE. Uma vez que pixels são recuperados a partir do servidor VNC, o mesmo pixels são dados para o API do SAGE para exibição imediata. Funcionalidade é algo crítico em um ambiente colaborativo, onde cada cientista com seu laptop precisa compartilhar informações (navegador web, apresentações).

- JuxtaView: é um aplicativo para ver imagens de altíssima resolução em monitores lado a lado. Através do SAGE, JuxtaView permite que um usuário de forma interativa dê zoom e navegue através de terabytes de imagens distribuídas, que são referenciadas espacialmente, como aquelas gerados a partir de microscópios eletrônicos, satélites e fotografias aéreas. Utilizando grandes quantidades de largura de banda.

- Captura de OpenGL: Numerosas aplicações científicas e pacotes de visualização usam a API OpenGL, como OpenDX, VTK, ou Paraview. O sucesso de Chromium e WireGL mostram a necessidade de suportar aplicações nativas OpenGL no modo binário (sem modificação do código fonte). Por isso foi desenvolvida uma biblioteca wrapper para OpenGL baseada no WireGL: usando um mecanismo de biblioteca compartilhada, capturam as chamadas OpenGL para o glSwapBuffer. Os pixels capturados são então

transmitidos para SAGE. Esta é uma maneira eficiente e extremamente fácil de portar aplicações nativas OpenGL para o SAGE. O desempenho é suficiente para rodar aplicações numa resolução de 1280x1200 a uma taxa de quadros interativa. Aplicações conhecidas para trabalhar: Paraview, Enliten de Ensight, iVew3D da IVS, o Google Earth.

- VRA: VRA permite aos cientistas renderizar conjuntos de dados volumétricos muito grandes em paredes de exibição lado a lado. Ele trabalha com distribuição de dados, sistema de memória distribuída, e técnicas de aceleração de hardware para renderização, assim produz uma solução que é escalável em termos de tamanho dos dados de entrada e resolução de saída.

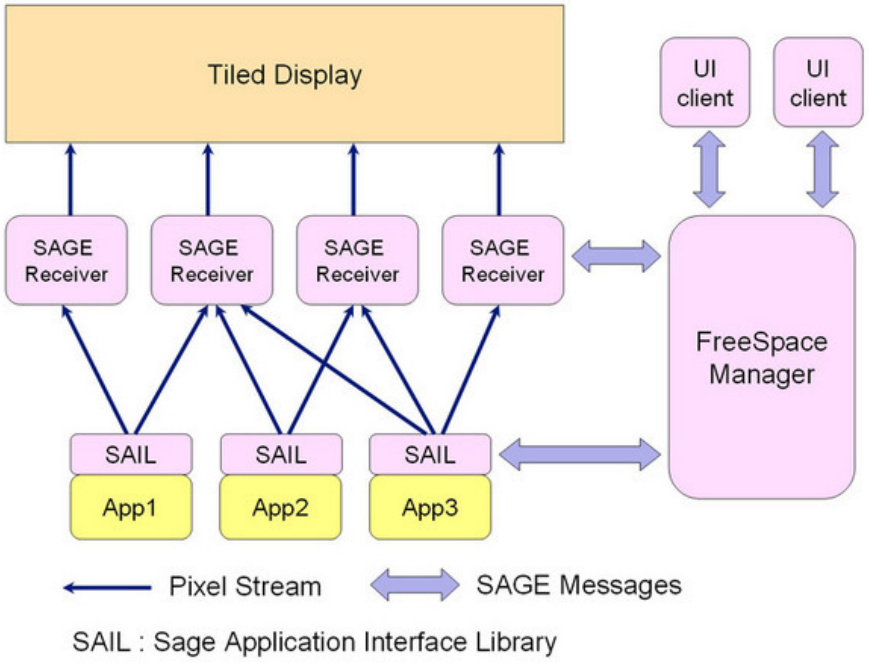


Figura 2.3: Arquitetura do SAGE (Eletronic Visualização Laboratory)

Interfaces de Usuário do SAGE

- SAGE User Interface: A interface do usuário SAGE (UI) é uma interface gráfica multiplataforma para gerenciamentos de janelas. Em vez de depender de uma ferramenta de linha de comando para manipular as janelas, uma representação gráfica da configuração de exibição distribuída e da localização da janela pode ser manipulada diretamente por um usuário a partir seu laptop. SAGE UI é escrito em Python e usa o toolkit wxWidget, por isso é portável (roda em Windows, MacOSX, Linux). Ela contém um painel para iniciar/parar o SAGE, um painel para configurar a maioria dos serviços do

SAGE, e controle sobre um lançador de aplicativos. Uma das características do SAGE é que a comunicação entre qualquer cliente SAGE e fsManager é feita usando mensagens de texto. Usando um método baseado em texto de comunicação, clientes SAGE diferentes podem interagir com fsManager, permitindo assim uma maior flexibilidade na linguagem de programação usada para criar o cliente SAGE. Já que o SAGE foi concebido com a colaboração em mente, vários UIs SAGE pode controlar um ambiente SAGE, ao mesmo tempo. Cada UI SAGE mostra o estado da mesma, isto é, o estado atual da parede de vídeo. Na Figura 2.3 é demonstrada a arquitetura do SAGE.

- Web User Interface: A interface de controle Web do SAGE é uma aplicação web, construída usando AJAX e Javascript, que permite aos usuários o controle do SAGE. O aplicativo funciona em navegadores web padrão, como FireFox, Internet Explorer e Safari (Mac). Foi desenvolvido principalmente para que os usuários com conhecimento mínimo de SAGE pudesem usar o SAGE para mostrar para os seus trabalhos na parede de vídeo. Ele se conecta a sessões de SAGE simultaneamente. Mover, redimensionar, minimizar e maximizar janelas do aplicativo são funções implementadas. Pode-se compartilhar uma área de trabalho através de VNC (Nota: a partilha da área de trabalho requer um servidor VNC rodando na máquina local). O lado do servidor é implementado usando o 'Apache Tomcat Servlet Container'.

Neste capítulo falamos sobre o que é visualização científica, sobre a sua importância, sobre visualização científica remota e ultra-resolução. Também fizemos um breve resumo sobre o SAGE (Jeong et al., 2005), que é um sistema de referência para visualização científica em ultra-resolução.

3

OpenGL e X Window System

NESTE capítulo fazemos um breve resumo sobre o OpenGL, X Window System e sobre o GLX, a extensão do OpenGL para o X Window System.

3.1 OpenGL

O OpenGL (Open Graphics Library) é uma API livre utilizada na computação gráfica, para desenvolvimento de aplicativos gráficos, ambientes 3D, jogos, entre outros (Wikipedia).

3.1.1 História do OpenGL

Na década de 80, cada fabricante de hardware tinha seu próprio conjunto de instruções para desenho de gráficos 2D e 3D. Construir aplicações com essas tecnologias que suportassem vários hardwares era um verdadeiro desafio. O esforço era constantemente duplicado e havia pouco espaço para companhias menores e sem capital para tamanho investimento.

Em meados dos anos 80, um padrão surgiu na indústria, chamado PHIGS (sigla de Programmer's Hierarchical, Graphics System) e começou a ser adotado por grandes fabricantes da época. Entretanto, embora tenha sido reconhecido por entidades como a ANSI e a ISO, o padrão começou a ser considerado complicado e desatualizado.

No final dos anos 80, a Silicon Graphics Inc. (SGI), criou um padrão chamado IRIS GL, que não só começou a chamar atenção da indústria, como também foi considerado o estado da arte de uma API gráfica. Consideravelmente mais fácil de usar, a API começou a tornar-se um padrão de fato na indústria.

Entretanto, grandes empresas produtoras do hardware, como a Sun Microsystems e a IBM, ainda eram capazes de fabricar hardware adotando o padrão PHIGS. Isso levou a SGI a tomar uma decisão para que a adoção de sua API fosse impulsionada: torna-la um padrão público, para que todos os fabricantes de hardware pudessem adota-lo.

A SGI considerou que a API Iris continha muito código proprietário e, portanto, não poderia ser aberta. Ela também lidava com questões não tão relacionadas ao desenho 2D e 3D, como gerenciamento de janelas, teclado e mouse. Ainda assim, era do interesse da Silicon Graphics manter os seus antigos clientes comprando seu hardware.

Como um resultado disso, o padrão OpenGL foi criado em 1992. Desde 1992, o padrão é mantido pelo ARB (Architecture Review Board), um conselho formado por empresas como a 3DLab, ATI, Dell, Evans&Sutherland, HP, IBM, Intel, Matrox, NVIDIA, Sun e, logicamente, a Silicon Graphics. O papel desse conselho é manter a especificação e indicar quais recursos serão adicionados a cada versão. A versão mais atual do OpenGL até hoje é a 4.2.

Os projetistas do OpenGL sabiam de duas coisas. A primeira é que fabricantes de hardware gostariam de adicionar recursos próprios, sem que tivessem que esperar para esses recursos estarem oficialmente aceitos no padrão. Para resolver esse problema, eles incluíram uma maneira de estender o OpenGL. Muitas vezes, essas extensões são interessantes o suficiente para que outros vendedores de hardware as adotem. Então, quando elas são consideradas suficientemente abrangentes, ou suficientemente importantes, a ARB pode promovê-las para que façam parte do padrão oficial. Praticamente todas as modificações recentes do padrão começaram como extensões, a grande maioria devido ao mercado de jogos.

Em segundo lugar, os projetistas também sabiam que muitos hardwares não seriam poderosos o suficiente para abraçar todo o padrão. Por isso, também incluíram extensões de software, que permitiam emular essas funcionalidades. Isso resolvia um problema sério que ocorria com a Iris: o da aplicação simplesmente não rodar pela falta de um ou outro recurso (Wikipedia).

3.1.2 Características da API OpenGL

Segundo Pozzer, essas são as principais características do OpenGL:

- É uma interface de software (API) com o hardware gráfico escrita em linguagem C.
- Multi-plataforma;
- Arquitetura bem definida;
- Não oferece comandos para gerenciamento de janelas, nem comandos para geração de modelos 3D complexos.
- Conjunto de primitivas geométricas simples;

3.1.3 Arquitetura da API OpenGL

Um programa OpenGL consiste de um laço infinito onde a cada passo a aplicação, geralmente escrita em linguagem C, passa para a API um conjunto de dados que devem ser processados. Esses dados mudam à medida que o usuário interage com a aplicação (Pozzer), como pode ser visto na Figura 3.1.

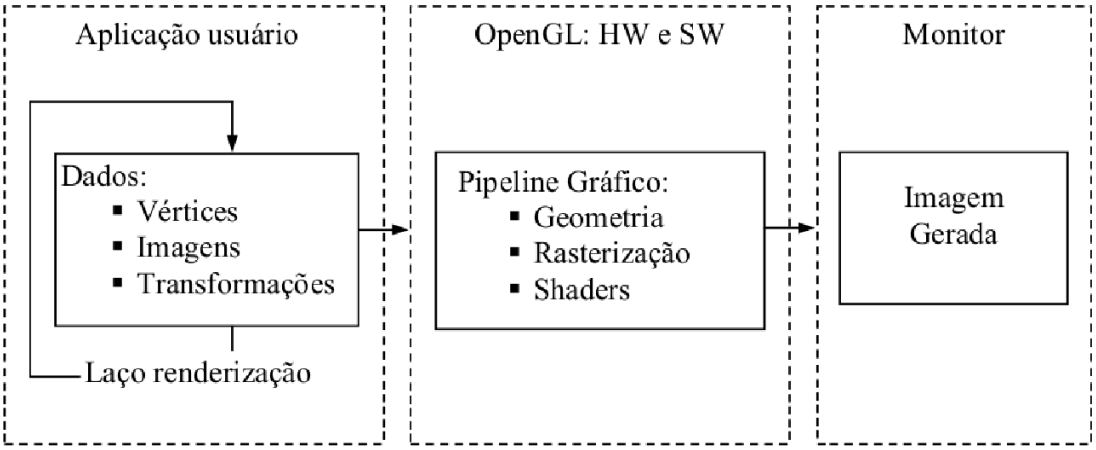


Figura 3.1: Pipeline de uma Aplicação OpenGL (Pozzer)

3.2 X Window System

O X Window System é um sistema cliente-servidor de janelas, transparente e que funciona em uma ampla gama de hardwares. A implementação da X. Org. Foundation é relativamente simples para ser construída e distribuída para a maioria dos sistemas compatíveis com POSIX. Implementações comerciais também estão disponíveis para uma ampla gama de plataformas (The X.Org Foundation).

Ele cria uma camada de abstração de hardware onde o software é escrito para usar um conjunto generalizado de comandos, permitindo a independência de dispositivo e reutilização programas em qualquer computador que implemente o X.

Originalmente chamado simplesmente de X, foi desenvolvido no MIT em 1984. Atualmente está na versão 11, publicada em setembro de 1987, e por isso carrega no nome este número.

3.2.1 Arquitetura do X Window System

Servidores do X Window System são executados em computadores com displays formado por um mapa de bits (bitmap). O servidor distribui a entrada do usuário e aceita pedidos de saída de vários programas cliente através de uma variedade de diferentes canais de comunicação entre processos. Embora o caso mais comum é para os programas clientes que

rodam na mesma máquina que o servidor, os clientes podem rodar de forma transparente a partir de outras máquinas (incluindo máquinas com arquiteturas e sistemas operacionais diferentes).

O X (como é comumente chamado) funciona segundo o modelo cliente-servidor: o servidor X recebe os pedidos via uma porta, um cliente X conecta-se ao servidor X e envia seus pedidos utilizando o protocolo X através da biblioteca X (Xlib). Este modelo de comunicação permite o uso de janelas de modo transparente através da rede.

O número de programas que usam X é bastante grande. Programas distribuídos pela X. Org Foundation incluem: um emulador de terminal, xterm, um gerenciador de janelas, twm; um gestor de display, xdm; um console gráfico xconsole; uma interface de e-mail, xmh; um editor de bitmap, bitmap; ferramentas de visualização e edição de recursos, appres, editres; programas de controle de acesso, xauth, xhost e iceauth; programas de configuração de preferência do usuário , xrdb, xcmsdb, xset, xsetroot, xstdcmap e xmodmap; relógios, xclock e oclock; um visualizador de fontes, xfd; utilitários para obter informações sobre fontes, janelas e displays, xlsfonts, xwininfo, xlsclients, xdpyinfo, xlsatoms e xprop; utilitários de manipulação de imagens, xwd, xwud e xmag; um utilitário de medição de desempenho, x11perf; um compilador de fontes, bdftopcf ; um servidor de fontes e utilitários relacionados, xfs, fsinfo, fslsfonts, fstobdf; um servidor de display e utilitários relacionados, Xserver, rgb, mkfontdir; um servidor de impressão e utilitários relacionados, Xprt, xplsprinters; utilitários de execução remota, RSTART e xon; uma prancheta, xclipboard gerente; compilador de teclado descrição e utilitários relacionados, xkbcomp, xkbprint, xkbbell, xkbevd, xkbvleds e xkbwatch; um utilitário para terminar clientes, xkill; um protocolo otimizado para proxy X, lbxproxy; um proxy de segurança e firewall, xfwp; um gerente de proxy para controlá-los, proxymngr; um utilitário para encontrar proxies, xfindproxy; Plugins para o Netscape Navigator, libxrx.so libxrxnest.so; e um utilitário para fazer com que parte ou a totalidade da tela seja redesenhada, xrefresh (The X.Org Foundation).

3.2.2 OpenGL Extensions to the X Window System (GLX)

GLX (OpenGL Extensions to the X Window System) é uma interface que proporciona a conexão entre a biblioteca OpenGL e o X Window System. Ele permite que programas utilizem OpenGL junto com as janelas do X Window System.

O GLX consiste de três partes:

- Uma API que proporciona funcionalidades para uma aplicação X Window System.
- Uma extensão do protocolo X, que permite ao cliente (a aplicação OpenGL) enviar comandos de renderização para o servidor X (o software resposável pela exibição). O cliente e o servidor podem rodar em computadores diferentes.

- Uma extensão do protocolo X que recebe o comandos de renderização do cliente e os repassa para a biblioteca OpenGL instalada (se a aceleração por hardware não está disponível normalmente é utilizada a biblioteca Mesa 3D, que emula este comportamento em software).

Se o cliente e o servidor estão sendo executados no mesmo computador e uma placa de aceleração 3D com seu respectivo driver estiver instalado a renderização será efetuada através do DRI (Direct Rendering Infrastructure) e o programa cliente terá permissão à acesso direto ao hardware de vídeo (Karlton and Womack, 2005).

Neste capítulo foi apresentado um breve resumo sobre o OpenGL, X Window System e a forma como os dois interagem, através do GLX. Esse capítulo é necessário para se entender a adoção das ferramentas que são apresentadas no próximo capítulo.

4

Ferramentas

NESTE capítulo apresentaremos de forma breve as ferramentas adotadas na construção do nosso protótipo para visualização remota e distribuída.

4.1 VNC

O *VNC* é um sistema baseado no protocolo RFB. Permite o uso remoto de máquinas, evitando que o usuário tenha que carregar consigo o hardware. O VNC permite que um único computador seja acessado simultaneamente a partir de vários locais, assim permitindo o trabalho cooperativo. A tecnologia por trás do VNC é um protocolo simples e é a simplicidade desse protocolo que torna o VNC tão poderoso. O protocolo VNC é totalmente independente de sistema operacional, sistema de janelas e aplicativo (Richardson et al., 2002). O Projeto VirtualGL fornece uma versão acelerada do VNC, chamado "TurboVNC", que é feita para ser usado com o VirtualGL. A combinação dos dois oferece uma solução de alto desempenho para o 3D remoto, mesmo em redes lentas. O TurboVNC também fornece recursos de colaboração, permitindo simultaneamente a vários utilizadores interagir com o mesmo aplicativo 3D (VirtualGL, 2010).

4.1.1 Protocolo RFB

RFB ("remote framebuffer") é um simples protocolo de acesso remoto a interface gráfica do usuário. Por trabalhar no nível de framebuffer ele é aplicável a qualquer sistemas de janelas e aplicações, incluindo o X11, Windows e Macintosh.

O ponto onde o usuário está sentado (local onde se localiza o monitor, teclado e mouse) é chamado de cliente RFB. Já o ponto onde o framebuffer é alterado é conhecido como servidor RFB.

RFB é verdadeiramente um protocolo "thin client". A enfase no planejamento do protocolo RFB era requerer muito pouco do lado cliente. Desta forma, clientes podem ser executados a partir de uma enorme gama de tipos de hardware diferentes, pois a tarefa de desenvolver um cliente é a mais simples possível.

O protocolo também se utiliza de um cliente que não mantém estado. Se um cliente desconectar de um servidor e posteriormente reconectar ao mesmo servidor, o estado da interface do usuário estará preservado. Além disso, um outro cliente pode ser usado para conectar ao mesmo servidor RFB. A partir desse novo cliente, o usuário irá ver exatamente a mesma interface gráfica que o primeiro usuário está visualizando a partir do cliente original (Richardson, 2005).

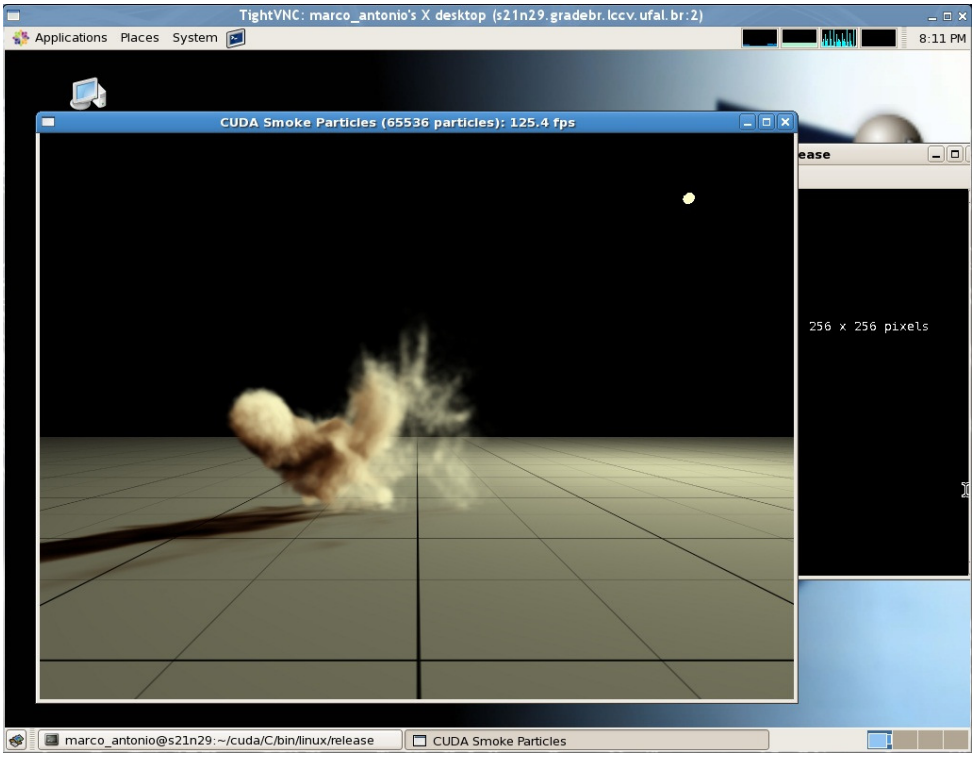


Figura 4.1: SmokeParticles (Nvidia Corporation, 2007) Através do Cliente VNC

4.1.2 Encodings

O servidor envia pequenos retângulos do framebuffer para o cliente. Em sua forma mais simples, o protocolo VNC pode usar muita largura de banda, por isso vários métodos foram desenvolvidos para reduzir a sobrecarga de comunicação. Por exemplo, existem várias codificações (métodos para determinar a maneira mais eficiente para a transferência desses retângulos). O protocolo VNC permite que o cliente e o servidor negociem a codificação e a paleta de cores que será usada.

Estes são os principais tipos de encodificação usadas:

Raw encoding

A codificação mais simples, que é suportada por todos os clientes e servidores, é a codificação Raw (Bruta), onde os dados são enviados em pixels da esquerda para a direita de cima para baixo, e depois que a tela original for transmitida por completo, apenas transfere outra tela se algum pixel tiver sido alterado.

Hextile encoding

Nesta codificação a tela é divida em 16x16 pedaços, onde cada pedaço é monitorado individualmente, havendo alteração na tela, os pedaços que sofreram alteração são enviados. Essa codificação funciona muito bem se apenas uma pequena porção da tela mudar de um quadro para o próximo (como um ponteiro do mouse se movendo através de um desktop, ou texto a ser escrito na posição do cursor), mas a demanda por banda ficar muito alta, se um monte de pixels mudarem ao mesmo tempo.

Zlib encoding

A codificação ZLib utiliza a biblioteca multi-plataforma de compressão de dados zlib para comprimir os dados codificados pela codificação Raw. Esta codificação alcança alta taxa de compressão, mas consome muito tempo de CPU.

Tight encoding

Assim como a codificação Zlib, a codificação Tight se utiliza da biblioteca *zlib* para compressão das imagens, mas estas são pre-processadas de forma a maximizar as taxas de compressão e minimizar o uso de CPU. Também faz uso da compressão JPEG, onde o nível de compressão pode ser ajustado pelo cliente. Essa codificação se mostra como a mais eficiente para desktops complexos e mudanças completas na tela, quando o objetivo é poupar banda.

4.2 VirtualGL

O VNC e outros ambientes de Desktop Remoto para Unix e Linux não suportam a execução de aplicativos OpenGL ou forçam os aplicativos OpenGL a serem executados sem o benefício da aceleração 3D via hardware. Já quando uma aplicação OpenGL é executada através de um servidor remoto X, seus comandos OpenGL são transmitidos para o cliente, que ficará encarregado pelo processamento gráfico, como pode ser observado na Figura 4.2.

O VirtualGL é um pacote de código aberto que permite que um Linux remoto execute aplicações OpenGL com aceleração via hardware. Com ele comandos OpenGL são renderizados no servidor e as imagens da aplicação 3D são geradas (VirtualGL, 2010). O VirtualGL pode ser combinado com um servidor VNC, esse esquema é exibido na Figura 4.3. Dessa

forma podemos visualizar aplicações OpenGL atráves de um cliente VNC, sendo somente enviada as imagens resultantes da renderização.

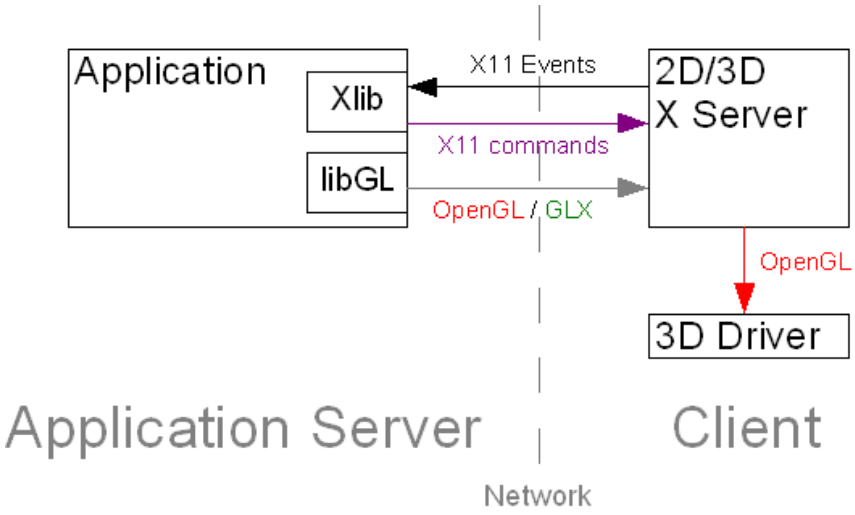


Figura 4.2: Renderização indireta (VirtualGL, 2010)

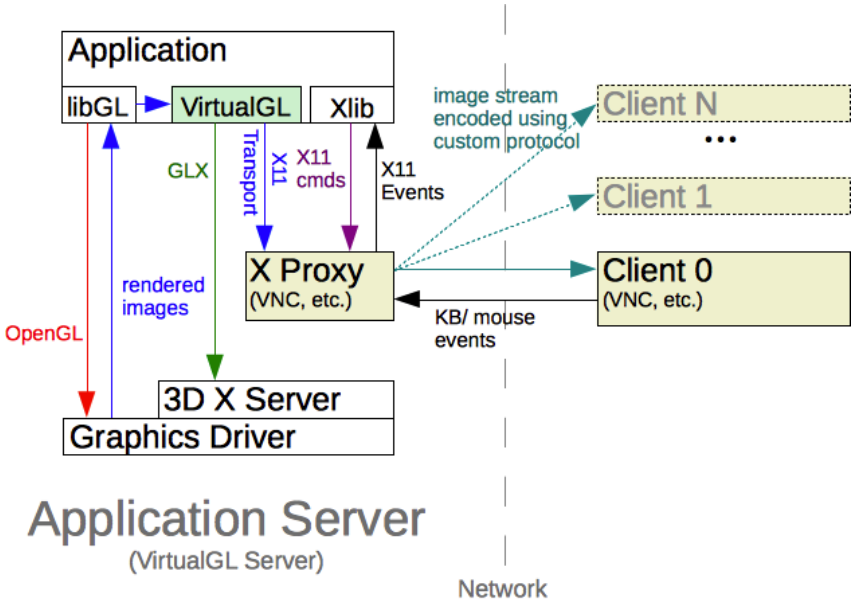


Figura 4.3: VirtualGL e Servidor VNC (VirtualGL, 2010)

4.3 Chromium

O Chromium é um sistema de código aberto para manipulação do fluxo de comandos da API gráfica. As unidades de processamento de fluxo (SPUs) do Chromium são implementadas como módulos que geralmente podem ser combinados de maneira arbitrária. As SPUs são escritas em C e podem ser estendidas. Esta flexibilidade permite que os usuários possam resolver problemas diversos que vão além da escalabilidade, além de permitir a integração

com uma interface já existente. O Chromium alcança boa escalabilidade em clusters, além de abstrair completamente a arquitetura do processamento gráfico, topologia de rede, e a semântica de processamento dos comandos da API (Humphreys et al., 2002).

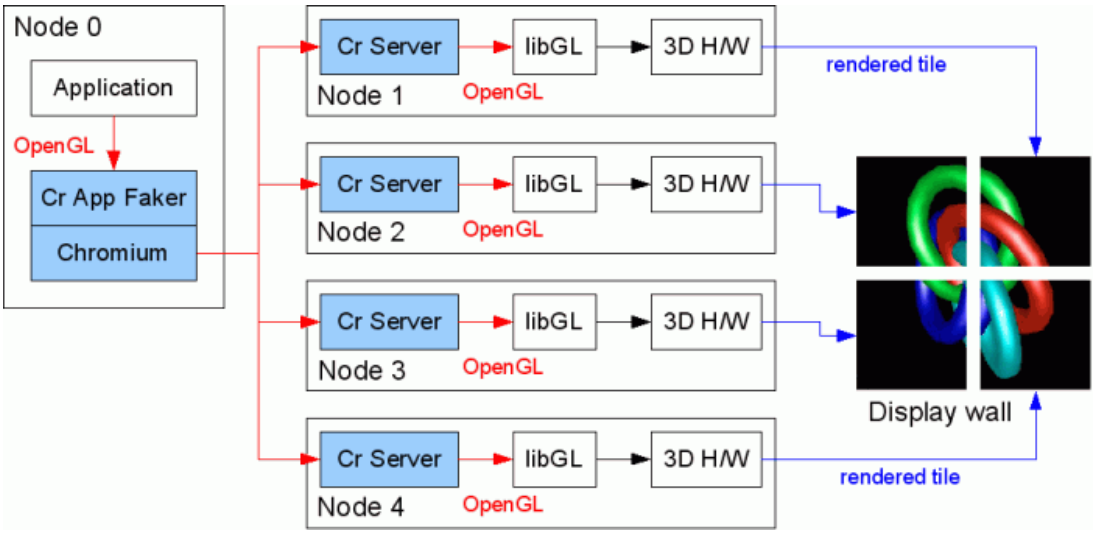


Figura 4.4: Modelo Sort-first (Humphreys et al., 2002)

Entre as características do Chromium, estas se destacam:

- Sort-first rendering - o frame buffer é subdividido em pedaços retangulares que podem ser renderizados em paralelo, até mesmo em máquinas diferentes.
- Sort-last rendering - o conjunto de dados 3D é dividido em N partes que são renderizadas em paralelo por N processadores. As imagens resultantes são juntas compondo a imagem final.
- Filtro de comandos OpenGL - os fluxos de comando OpenGL podem ser interceptados e modificados por uma unidade de processamento de fluxo (SPU) para implementar uma renderização não-fotorealista.
- Muitos programas OpenGL podem ser utilizados com o Chromium sem necessidade de modificação do código. O Chromium intecepta os comandos que a aplicação envia para o driver OpenGL, assim obtendo a fonte de comandos.

Neste capítulo foram apresentadas as ferramentas utilizadas na elaboração do protótipo.

5

Resultados

NESTE capítulo é apresentado o ambiente computacional utilizado para os experimentos, a banda utilizada e o número de quadros por segundo transmitidos nas diversas configurações apresentadas abaixo.

5.1 Ambiente de Testes: O Cluster GradeBR/UFAL

Figura 5.1: Cluster GradeBR/UFAL

O *cluster* GradeBR-UFAL, instalado nas dependências do LCCV, é uma das partes que compõe a maior grade de computadores da América Latina, possui a função de realizar cálculos e processar os dados necessários a exploração de petróleo na área do Pré-Sal. Os nós do cluster são compostos de 2 processadores de 4 núcleos com 24 GB RAM compartilhada, sendo 15 nós ligados a um mesmo switch. Existem 16 switches em topologia hipercubo 4D (Torus 4D 2x2x2x2), essas ligações são feitas em uma malha InfiniBand 4xQDR com largura de banda de 40 Gbps. Um dos servidores em cada switch está ligado a duas unidades GPGPU através de porta PCI-Express 16x. Esta mesma rede InfiniBand também interliga os servidores responsáveis pelo sistema de arquivos paralelo, o LustreFS. Este sistema é composto de 2 servidores de MetaDados (MDS) e 4 servidores de armazenagem de objetos (OSS) que são ligados a diferentes nós da topologia. Resultados preliminares geraram eficiência HPL superior a 85% e desempenho em IOR-POSIX de 6,8 e 2,7 Gbps em R/W, respectivamente, de disco (Vanderlei and Pereira, 2010).

Um resumo da capacidade computacional do cluster GradeBR é apresentado na Tabela 5.1.

INFORMAÇÃO	DADOS
Nome	GradeBR-UFAL
Tipo	Hipercubo 4D
Tecnologia	Sun Fire 2X2S
Processadores (CPUs)	Intel Xeon X5560/W5580 2,8/3,2 GHz de 4 núcleos
Processadores Gráficos de Uso Geral (GPGPU)	nVidia Tesla S1070
Tamanho	1.752 núcleos de processamento em CPU (1496 Intel Nehalem 2,8 GHz, 256 Intel Nehalem 3,2 GHz) e 7.680 núcleos de processamento em GPGPU (Nvidia cores)
Rede	Infiniband QDR (40 Gbps)
Memória	3 GB de RAM por núcleo de CPU (total de 5,256 TB)
Armazenamento	90 TB em disco (50 TB LustreFS + 40 TB NFS)
Desempenho máximo teórico (Rpeak)	20,07 Tflops em CPUs + 32 Tflops em GPGPU com precisão simples

Tabela 5.1: Informações sobre o cluster

Arquitetura de Rede InfiniBand

InfiniBand (IB) é uma arquitetura ponto-a-ponto que define um canal de entrada e saída (I/O Channel) de alta velocidade para interligar servidores, equipamentos de infra-estrutura de comunicação, sistemas de armazenamento e embarcados (de Melo Carvalho Filho et al., 2010). Por oferecer baixa latência, elevada largura de banda e carga de CPU extremamente baixa com uma ótima relação custo/benefício, o InfiniBand se tornou a mais implantada interconexão de alta velocidade. A arquitetura InfiniBand foi projetada para fornecer escalabilidade para dezenas de milhares de nós e múltiplos núcleos de CPU por servidor e para

fornecer eficiente utilização dos recursos computacionais de processamento. Atualmente, InfiniBand fornece throughput de até 40 Gb/s servidor para o servidor e 120Gb/s switch para switch. Essa largura de banda de alto desempenho é aliada com a baixíssima latência, e este desempenho tem quase custo zero em termos de utilização da CPU. (Grun, 2010)

Lustre

Na procura de um sistema de arquivos que possibilite um maior desempenho, aumentando o throughput (taxa de transferência) de dados e garantindo que esses dados possam ser os mesmos para uma grande quantidade de processos ou usuários, foram encontrados sistemas de arquivo que provêm maior desempenho. Dentre esses sistemas, se destaca o Lustre (da Silva et al., 2010). O Lustre é um sistema de arquivos paralelo distribuído de código aberto desenvolvido e mantido pela Sun Microsystems Inc. (Oracle). Devido à arquitetura extremamente escalável do sistema de arquivos Lustre, o seu uso é popular nas aplicações de computação científica, bem como na indústria de óleo e gás, manufatura, mídia e no setor financeiro. O Lustre apresenta uma interface POSIX aos seus clientes com capacidade de acesso paralelo. Boa parte dos supercomputadores mais rápidos em todo o mundo usam o sistema de arquivos Lustre para a alta desempenho (Wang et al., 2009).

5.2 Protótipo

Os testes foram realizados usando apenas um dos 16 nós do cluster que estão, cada um, conectados a duas GPGPUs Nvidia Tesla C1070. A GPGPU não possui saída gráfica física, o servidor gráfico foi configurado para gerar uma saída gráfica virtual. O proxy do servidor gráfico foi configurado e integrado a um software de renderização local de OpenGL, o VirtualGL. "Virtualiza-se"o hardware de gráficos 3D, permitindo que ele seja co-localizado com maiores recursos de computação e armazenamento, também permitindo que o hardware gráfico seja compartilhado entre vários usuários. O mais importante, é que o VirtualGL elimina a rede como barreira ao tamanho dos dados. O usuário agora pode visualizar Terabytes de dados em tempo real sem a necessidade de copiar qualquer um dos dados através da rede ou sentar na frente da máquina que está processando os dados.

No modelo apresentado, como pode ser visto na Figura 5.2, o armazenamento (LustreFS) e o servidor de renderização estão no mesmo ambiente e se comunicam através da rede InfiniBand, ou seja não há transferência externa de dados. O servidor de renderização se utiliza de um servidor VNC (aplicação) combinado com o VirtualGL para enviar somente as imagens geradas pela aplicação 3D para o(s) cliente(s), além disso o VNC permite a interação entre o cliente e a tal aplicação.

O sistema ainda está em fase de prototipação, havendo instabilidade, pode ser acessado através de qualquer conexão de banda larga e por dispositivos com pouco poder de proces-

samento, como *tablets* e celulares.

O acesso e a usabilidade do protótipo são bastantes simples, quando se manipula uma única sessão com resolução máxima limitada a 2560x1600 pixels. Já para a exibição em uma parede de vídeo, se faz necessária a inicialização de várias sessões VNC e a criação de um arquivo de configuração no Chromium indicando a resolução, quantidade de telas e sua ordem. Em comparação com o SAGE, o protótipo se mostra mais flexível pois possibilita a visualização em uma sessão única e através de um cliente VNC qualquer. Quando tratamos de visualização em um paredão de vídeo, o SAGE possui características que facilitam a gestão, enquanto o protótipo aqui apresentado se mostra pouco amigável. Estudos estão sendo feitos no sentido de facilitar o uso do protótipo em paredões de vídeo, dado que o Chromium possui integração com o DMX (Distributed Multihead X), possibilitando a criação de uma única área de trabalho com a visualização distribuída em várias sessões, assim sendo possível mover e redimensionar janelas em um paredão, tendo uma sessão VNC como ponto de controle.

Somente foram adotadas tecnologias multiplataformas e de código aberto para a construção deste protótipo.

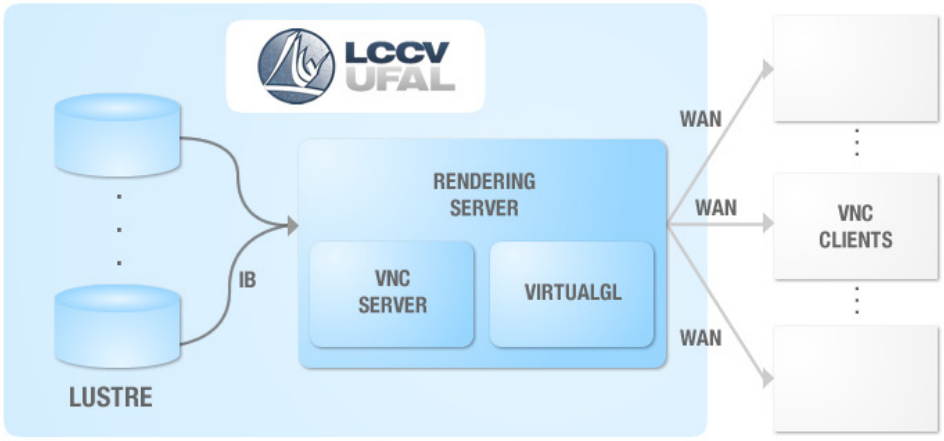


Figura 5.2: Pipeline do Protótipo para Exibição

Na Figura 5.3 é apresentado o pipeline de visualização do *eaviv*, os servidores de dados e de renderização estão distribuídos fisicamente. Primeiro os dados são transferidos do servidor de dados para o servidor de renderização, a renderização é feita em paralelo, depois as imagens geradas são transferidas do servidor de renderização para os usuários locais.

No *pipeline* referente ao protótipo aqui apresentado, a visualização é gerada no mesmo lugar que as simulações são feitas, não existindo transferência de dados para outro cluster específico para renderização, o diferenciando de Ge et al. (2010). E como o protocolo VNC foi usado, não há necessidade de rede de alta velocidade, possibilitando a visualização a longa distância.

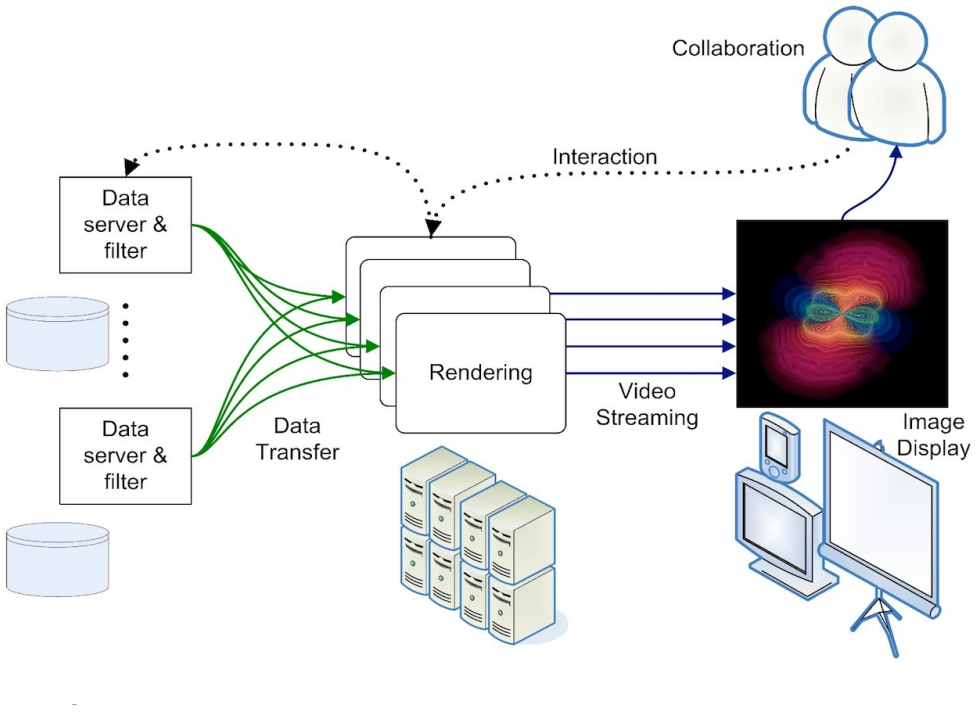


Figura 5.3: Pipeline de Visualização do *eaviv* (Ge et al., 2010)

5.3 Análise de Desempenho

Uma análise do desempenho e da largura de banda utilizados pelo pipeline proposto é apresentada abaixo. O servidor (TightVNC 1.3.9 + VirtualGL 2.1) foi executado em um dos nós do cluster, o qual foi conectado a interface ethernet gigabit, permitindo acesso direto ao mesmo. O cliente TightVNC 1.3.10 foi executado em um Notebook com processador Intel Core i5-2430M de 2.40 GHz, com 4Gb de memória RAM e com sistema operacional ArchLinux 64bits.

5.3.1 Metodologia

Para analisar o desempenho do protótipo, foi proposto testa-lo com a transmissão de um filme em duas resoluções: 1280x720 e 1920x1080, que são as resoluções do padrão HDTV. O filme escolhido (Trailer de "Os Simpsons: O Filme") foi transmitido por 100 segundos através do *Mplayer* redirecionando sua saída gráfica para o hardware gráfico, o que possibilita o VirtualGL interceptar a saída. O filme foi transmitido na duas resoluções HDTV e com a taxa de atualização de 24 quadros por segundo. Sendo analisado o desempenho com os 4 encodings mais conhecidos do protocolo VNC: *Raw*, *Hextile*, *ZLib* e *Tight*.

5.3.2 Monitoramento

Trafégo de Rede

Para medir o tráfego de dados foi utilizado o *vnStat*, que é um monitor de tráfego de rede para Linux e BSD. Este usa a interface de estatística de rede provida pelo kernel Linux. Com o *vnStat* temos informações como a banda máxima, mínima e a média, durante o período em que ele está monitorando a interface escolhida. A comparação da banda média de rede exigida em cada configuração avaliada pode ser vista na Figura 5.4.

Devido aos resultados apresentados na Figura 5.4 e 5.8, a codificação Tight foi escolhida para ser utilizada em testes com variação na paleta de cores, onde foi monitorada a largura de banda utilizada variando a quantidade de bits de profundidade de cor. Este teste foi motivado pelo fato que nem sempre é necessária grande variedade de cores na visualização.

A avaliação do consumo médio da banda de rede consumida pela codificação Tight, usando a resolução 1920x1080 e variando a quantidade de bits que representam a quantidade de cores entre 6, 16 e 24 bits pode ser visualizada na Figura 5.6. E o resultado da variação de cores pode ser vislumbrado na Figura 5.5.

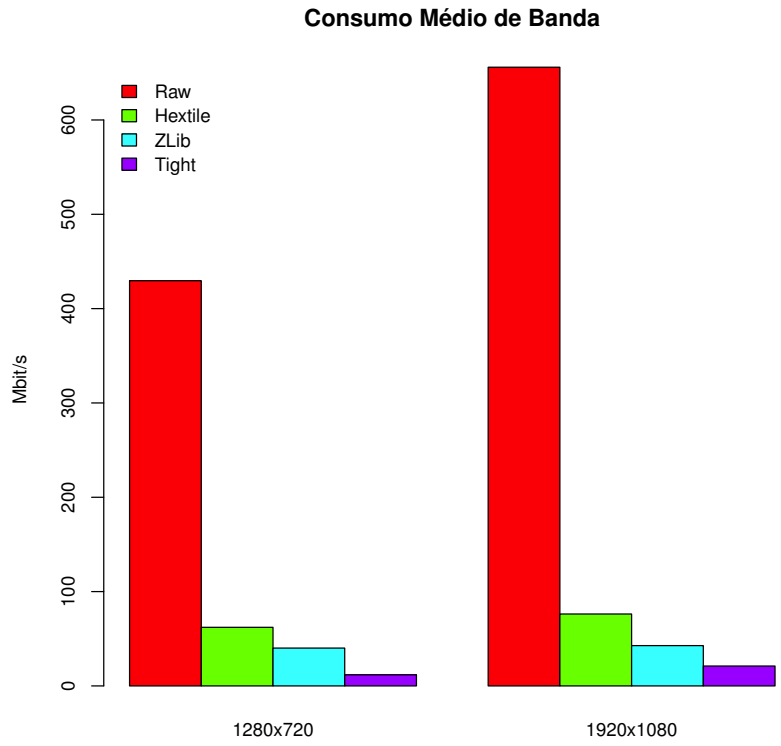
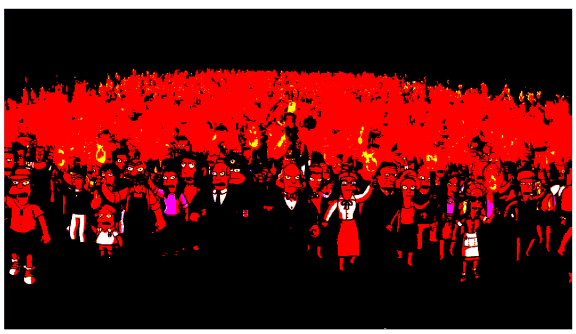
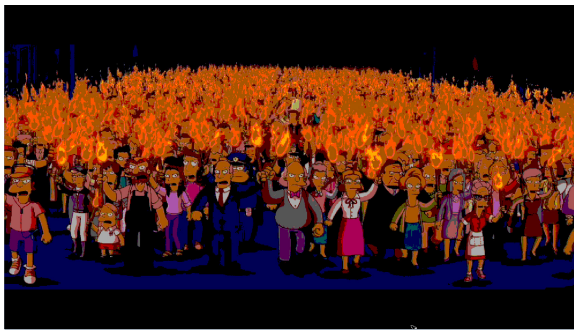


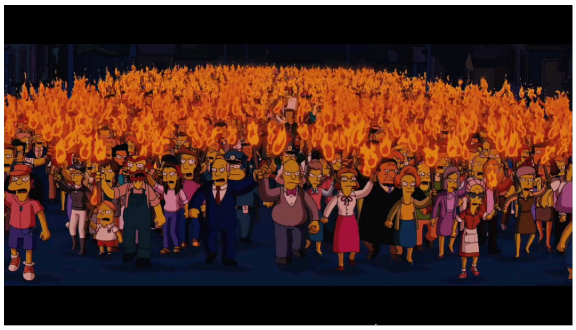
Figura 5.4: Média do Consumo de Rede



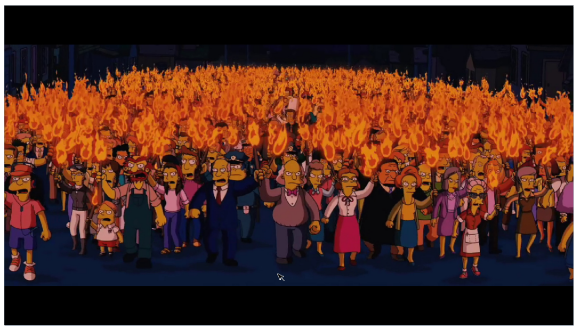
(a) 3 bits



(b) 6 bits



(c) 16 bits



(d) 24 bits

Figura 5.5: Screenshot do Vídeo com Variação na Paleta de Cores

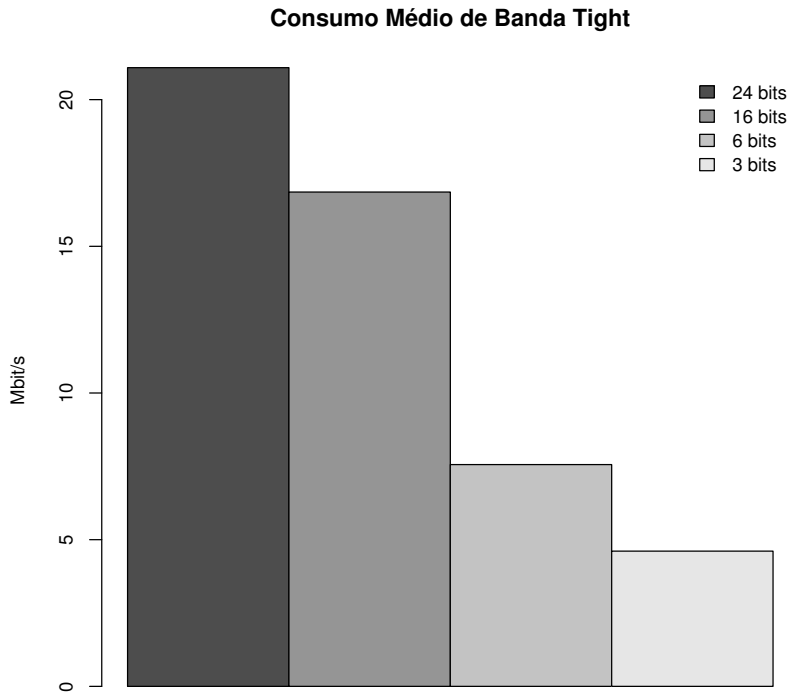


Figura 5.6: Média do Consumo de Rede Variando a Paleta de Cores

Medição de FPS no Cliente VNC

Para medição dos frames por segundo (FPS) se fez necessária uma pequena modificação no código fonte do cliente TightVNC 1.3.10 (versão em C), onde é incluindo um trecho de código responsável por contar as atualizações de quadros feitas por segundo.

Nas Figuras 5.7 e 5.8 pode-se visualizar a variação do FPS durante os 100 (cem) primeiros segundos de exibição com as quatro codificações, nas resoluções de 1280x720 e 1920x1080, respectivamente.

Já nas Tabelas 5.2 e 5.3 é feito um comparativo entre as codificações, no que diz respeito à média de frames por segundo transmitidos, taxa média de transferência, e ao tamanho de cada frame, nas resoluções de 1280x720 e 1920x1080, respectivamente.

Dada a instabilidade na quantidade de frames por segundo avaliada nas Figuras 5.7 e 5.8, o experimento foi repetido mais quatro vezes e foi constatado que os picos seguem um padrão, como pode ser visto na Figura 5.9.

Encoding	Média FPS	Transf. Média (Mb/s)	Tamanho Frame (MB)	Tam. Frame %
Raw	19.83	429.59	2.71	100
Hextile	8.36	62.13	0.93	34.30
ZLib	8.51	40.11	0.59	21.75
Tight	20.87	11.75	0.07	2.59

Tabela 5.2: Comparação entre os Encodings na Resolução de 1280x720

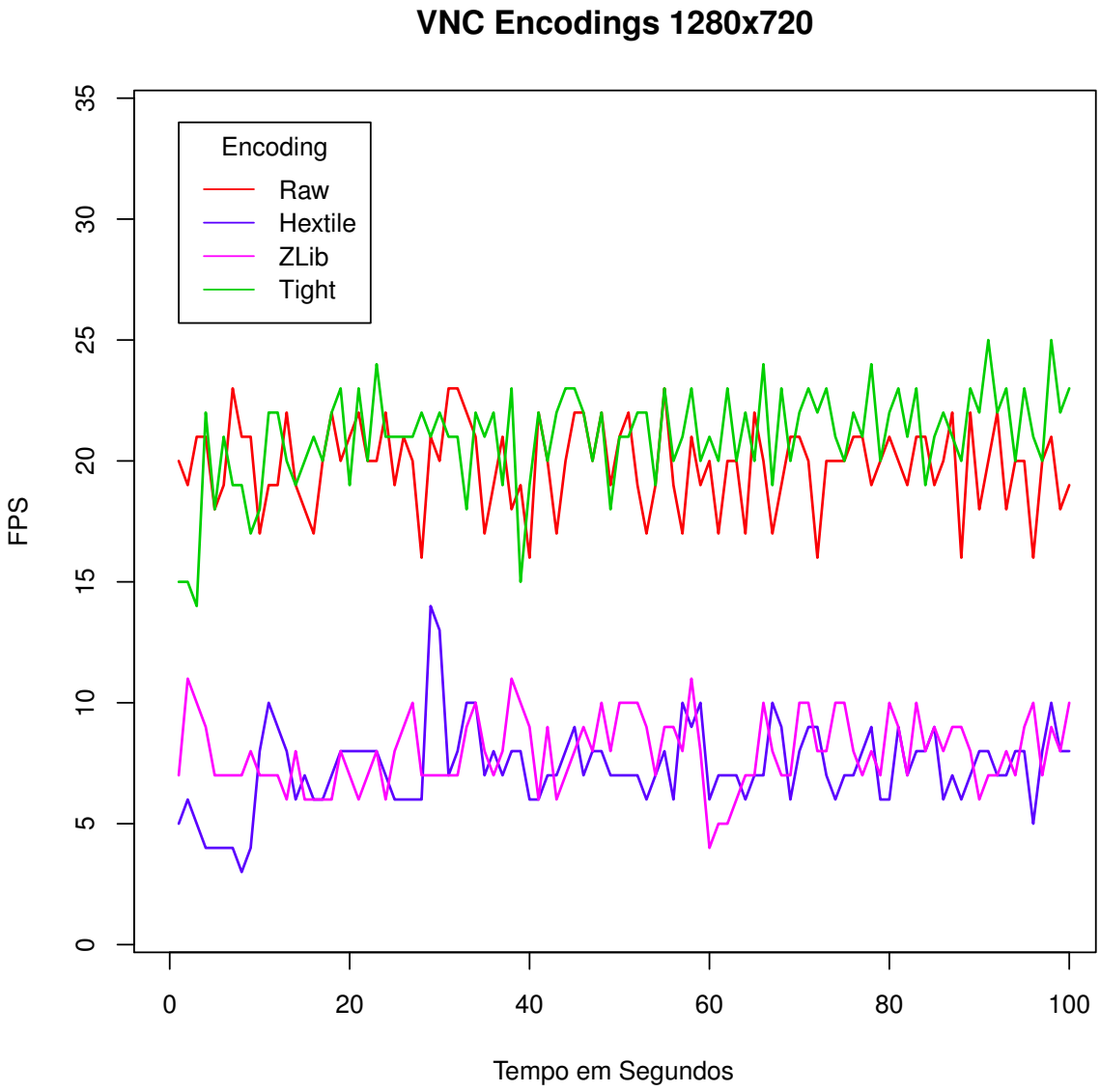


Figura 5.7: Medição do FPS na Resolução 1280x720

Encoding	Média FPS	Transf. Média (Mb/s)	Tamanho Frame (MB)	Tam. Frame %
Raw	13.68	655.94	6.00	100
Hextile	4.93	76.25	1.93	32.26
ZLib	4.29	42.74	1.25	20.77
Tight	17.48	21.09	0.15	2.52

Tabela 5.3: Comparação entre os Encodings na Resolução de 1920x1080

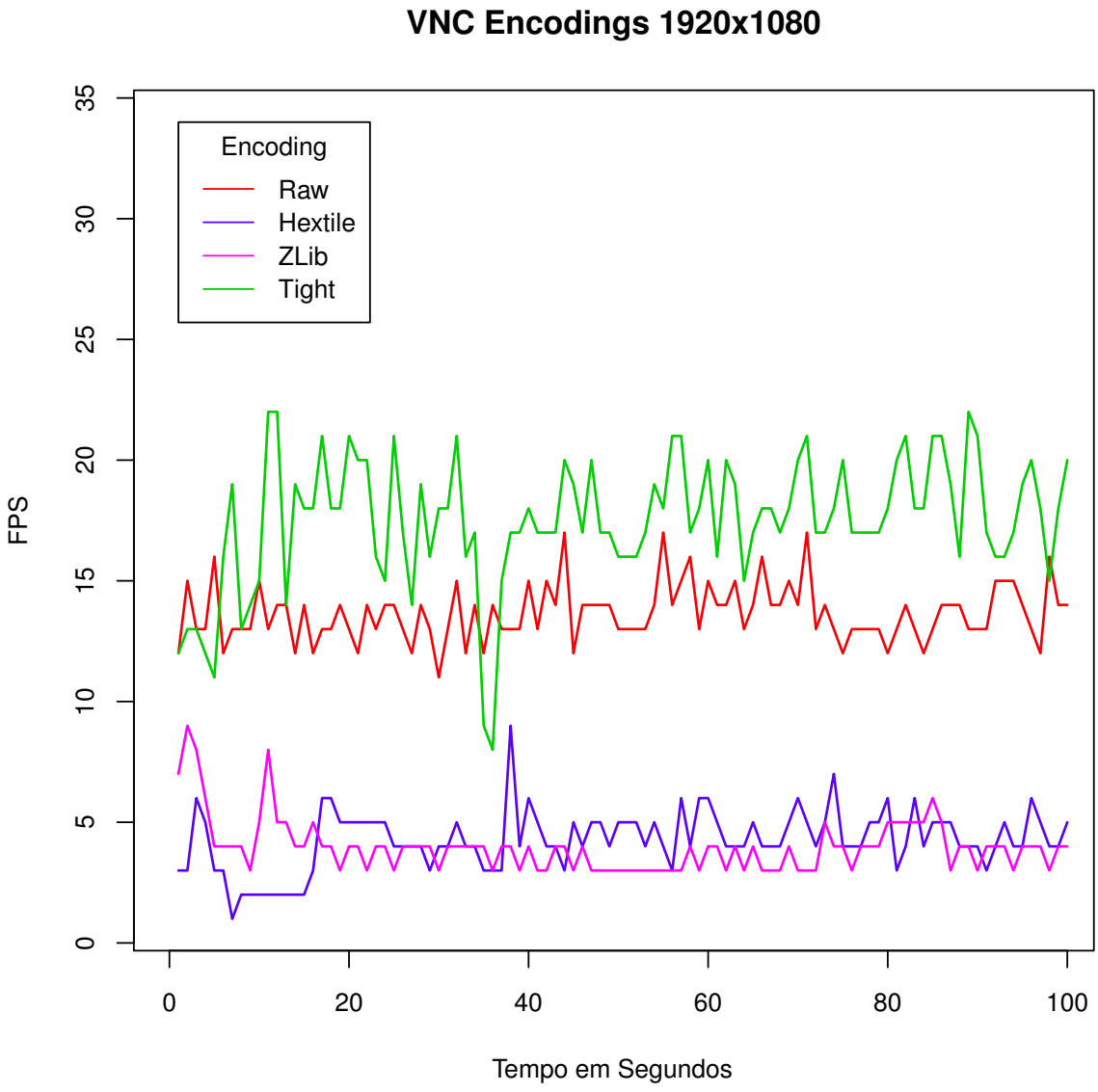


Figura 5.8: Medição do FPS na Resolução 1920x1080

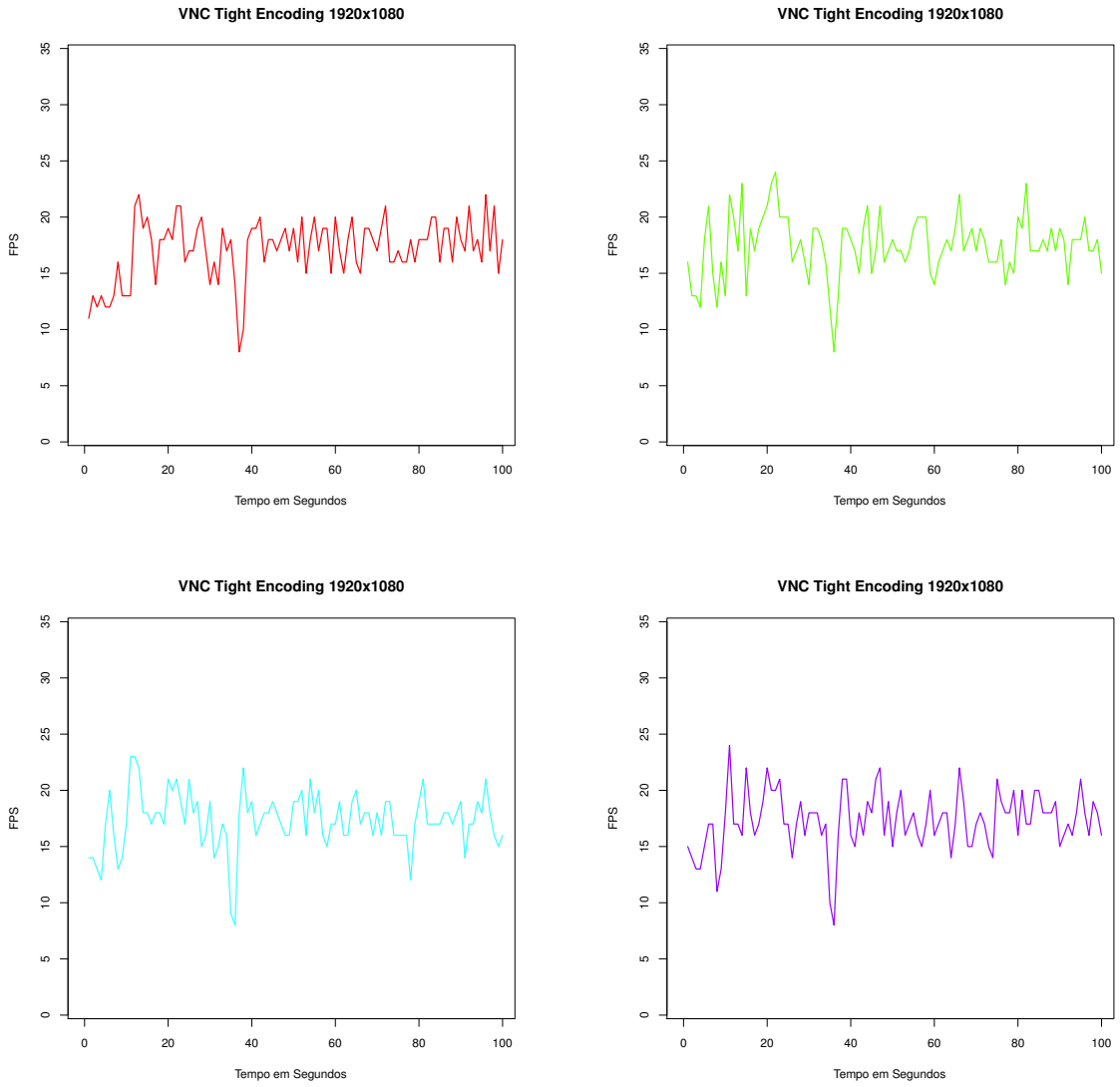


Figura 5.9: Medições do FPS na Resolução 1920x1080 Usando a Codificação Tight

Neste capítulo é apresentado o protótipo para visualização remota, além dos dados provenientes do monitoramento (FPS e Tráfego de Rede) deste usando várias configurações diferentes na sessão VNC.

6

Considerações Finais

Foi possível visualizar remotamente dados renderizados no Cluster GradeBR/UFAL. As visualizações remotas foram feitas até fora da rede da UFAL, demonstrando que até em conexões residenciais de banda larga ela é aceitável, assim foi demonstrada a viabilidade das tecnologias adotadas. Na Figura 6.1 é demonstrado o baixo custo de processamento para a visualização remota, graças à simplicidade do cliente VNC podemos visualizar os dados através de dispositivos com relativamente pouco poder de processamento, como *tablets* e *smartphones*.

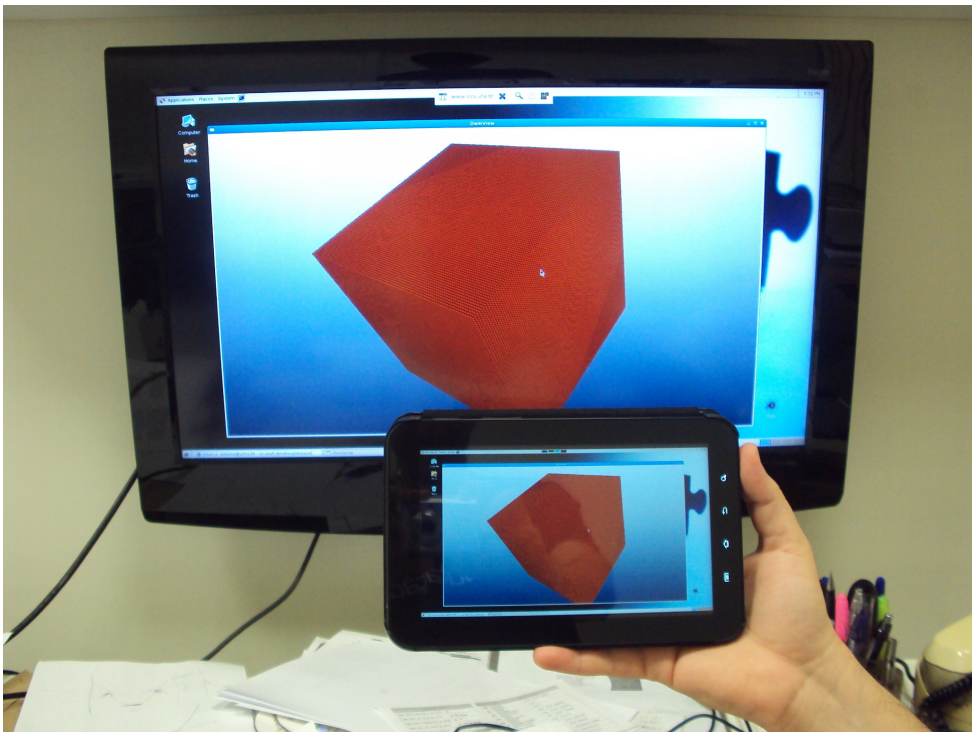


Figura 6.1: Demonstração do Cliente VNC em um Tablet

Avaliando os resultados do capítulo anterior, nota-se que a codificação Tight se mostra mais eficiente que as outras codificações, pelo menos no caso avaliado, tanto na taxa de quadros por segundo (FPS), quanto na utilização do trafégo de rede. Das quatros codificações é a única que apresenta perda de dados, resultantes da compressão JPEG, sendo que a codificação dá suporte a escolher o nível de compressão.

Quanto a variação no FPS, conclui-se que seja provocada pela própria natureza do protocolo RFB, onde um novo frame só é enviado quando há alguma mudança em relação ao frame anterior. Nota-se visualmente que há algumas pequenas paradas no vídeo no momento entre 36 e 39 segundos, que resultam no pico de valor mínimo global visto na Figura 5.9. Repetindo o monitoramento com as mesmas configurações e com o mesmo trecho de vídeo, é possível notar um padrão, principalmente no que diz respeito as quedas na taxa de frames por segundo. Um estudo mais aprofundado sobre essa variação deve ser feito futuramente.

Com os resultados obtidos conclui-se que a transmissão de imagens em alta-resolução pelo protótipo é viável, em termos de qualidade e largura de banda. Sendo possível a construção de uma parede de vídeo usando esta configuração. Vários servidores VNCs podem rodar em um único nó e com o Chromium, usando o modelo Sort-first, pode-se dividir a renderização de uma aplicação para várias sessões VNC em vários nós. Alguns testes com essa configuração estão sendo realizados, como pode ser visto na Figura 6.2, mas o trabalho ainda está muito prematuro.

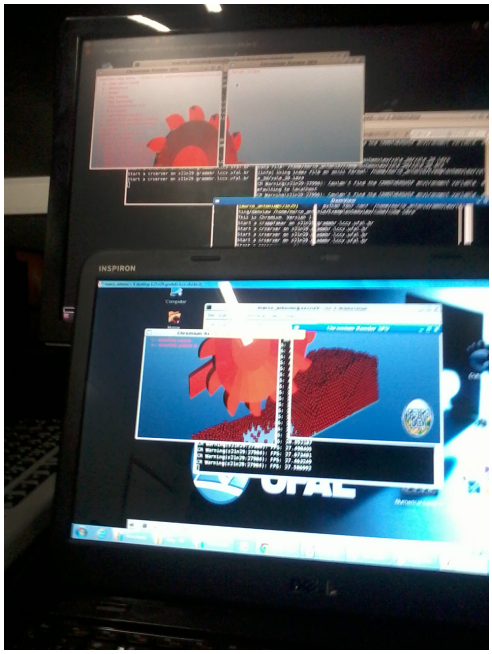


Figura 6.2: Teste com o Modelo Sort-First

Referências bibliográficas

A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre. Remote large data visualization in the paraview framework. *Proceedings of the Eurographics Parallel Graphics and Visualization*, pages 162–170, 2006.

H. Childs, E. Brugger, K.S. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A contract based system for large data visualization. In *Proceedings of IEEE Visualization*, volume 2005, pages 190–198, 2005.

Anderson Santos da Silva, Baltazar Tavares Vanderlei, Joílnen Batista Leite, and Leonardo Viana Pereira. Sistema paralelo de arquivos em rede de alto desempenho no cluster gradebr/ufal. *Anais do VII Congresso Acadêmico da Universidade Federal de Alagoas*, 2010.

Ademir de Melo Carvalho Filho, Diego Cedrim Gomes Rego, Leonardo Viana Pereira, and Baltazar Tavares Vanderlei. Arquitetura infiniband: características e estudo de caso do cluster gradebr/ufal. *Anais do VII Congresso Acadêmico da Universidade Federal de Alagoas*, 2010.

Eletronic Visualização Laboratory. Sage. URL http://www.sagecommons.org/. Último acesso realizado no dia 01/02/2012.

M. Friendly and D.J. Denis. Milestones in the history of thematic cartography, statistical graphics, and data visualization. 174, 2001. URL http://www.math.yorku.ca/SCS/Gallery/milestone.

Jinghua Ge, Andrei Hutanu, Cornelius Toole, Robert Kooima, Imtiaz Hossain, and Gabrielle Allen. An experimental distributed visualization system for petascale computing. *IEEE Computing in Science and Engineering*, September/October:78–82, 2010.

Paul Grun. Introduction to infiniband for end users. 2010. URL http://members.infinibandta.org/kwspub/Intro_to_IB_for_End_Users.pdf.

G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner, and J. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Transactions on Graphics*, 21(3):693–702, 2002. ISSN 0730-0301.

B. Jeong, R. Jagodic, L. Renambot, R. Singh, A. Johnson, and J. Leigh. Scalable graphics architecture for high-resolution displays. In *IEEE Information Visualization Workshop*, 2005.

P. Karlton and P. Womack. Opengl graphics with the x window system, 2005.

Falko Kuester, Kai-Uwe Doerr, So Yamaoka, Jason Kimball, Kevin Ponto, Tom Wypych, Daniel Knoblauch, Tom DeFanti, Greg Dawe, and Larry Smarr. The highly interactive parallelized display space project (hiperspace). URL http://vis.ucsd.edu/projects/hiperspace.

Nvidia Corporation. Compute unified device architecture programming guide. *NVIDIA: Santa Clara, CA*, 2007.

Alyson Leandro Costa Oliveira, Baltazar Tavares Vanderlei, and Leonardo Viana Pereira. Infraestrutura de rede para o grid continental gradebr: Proposta para o nó gradebr/ufal. *Anais do VII Congresso Acadêmico da Universidade Federal de Alagoas*, 2010.

C.T. Pozzer. Opengl–conceitos básicos.

T. Richardson. The rfb protocol. *AT&T Labs Cambridge Whitepaper*, 2005.

T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. Virtual network computing. *Internet Computing, IEEE*, 2(1):33–38, 2002. ISSN 1089-7801.

The X.Org Foundation. X man page. URL http://ftp.x.org/pub/X11R6.8.2/doc/X.7.html. Último acesso realizado no dia 01/02/2012.

Baltazar Tavares Vanderlei and Leonardo Viana Pereira. Implementação de um supercomputador: cluster gradebr/ufal. *Anais do VII Congresso Acadêmico da Universidade Federal de Alagoas*, 2010.

VirtualGL. User's guide for virtualgl. 2010. URL http://www.virtualgl.org/vgldoc/2_1_4/. Último acesso realizado no dia 01/02/2012.

F. Wang, S. Oral, G. Shipman, O. Drokin, T. Wang, and I. Huang. Understanding lustre filesystem internals. 2009.

Wikipedia. Opengl. URL http://en.wikipedia.org/wiki/OpenGL. Último acesso realizado no dia 01/02/2012.

Q. Wu, J. Gao, Z. Chen, and M. Zhu. Pipelining parallel image compositing and delivery for efficient remote visualization. *Journal of Parallel and Distributed Computing*, 69(3): 230–238, 2009a. ISSN 0743-7315.

Q. Wu, M. Zhu, Y. Gu, and N.S.V. Rao. System design and algorithmic development for computational steering in distributed environments. *IEEE Transactions on Parallel and Distributed Systems*, 2009b. ISSN 1045-9219.

S. Yamaoka, K.U. Doerr, and F. Kuester. Visualization of high-resolution image collections on large tiled display walls. *Future Generation Computer Systems*, 2010.

Este trabalho foi redigido em LaTeX utilizando uma modificação do estilo IC-UFAL. As referências bibliográficas foram preparadas no JabRef e administradas pelo BibTeX com o estilo plainnat. O texto utiliza fonte Fourier-GUT*enberg*em corpo de 12 pontos. A numeração dos capítulos segue com a familia tipográfica Art Nouveau Caps.