MARCELO DE PAIVA GUIMARÃES

Um Ambiente para o Desenvolvimento de
Aplicações de Realidade Virtual baseadas em
Aglomerados Gráficos

Tese apresentada à Escola
Politécnica da Universidade de
São Paulo para a obtenção do
título de Doutor em Engenharia
Elétrica.

São Paulo
2004

MARCELO DE PAIVA GUIMARÃES

Um Ambiente para o Desenvolvimento de Aplicações de Realidade Virtual baseadas em Aglomerados Gráficos

Tese apresentada à Escola
Politécnica da Universidade de
São Paulo para a obtenção do
título de Doutor em Engenharia
Elétrica.

Área de Concentração
Sistemas Eletrônicos

Orientador
Prof. Dr. Marcelo Knörich Zuffo

São Paulo
2004

FICHA CATALOGRÁFICA

Guimarães, Marcelo de Paiva

Um ambiente para o desenvolvimento de aplicações de realidade virtual baseadas em aglomerados gráficos / Marcelo de Paiva Guimarães. -- São Paulo, 2004.

126 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1. Sistemas distribuídos 2. Realidade virtual 3. Computação gráfica 4. Multiprogramação e multiprocessamento I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II. t.

Dedico este trabalho à minha esposa Rita
e à minha mãe Oscalina, pelo amor e
incentivos dispensados comigo.

*“Há três coisas que nunca voltam atrás:
A flecha lançada, a palavra pronunciada e a oportunidade perdida.”*
Provérbio chinês

*“É melhor estar preparado para uma oportunidade e não ter nenhuma,
do que ter uma oportunidade e não estar preparado.”*
Whitney Young Jr.

AGRADECIMENTOS

A escrita dos agradecimentos é uma tarefa difícil, talvez até mesmo mais difícil que escrever esta tese, pois este trabalho foi realizado com apoio de muitos. Então, aqui irei apenas tentar, e os que não foram citados me perdoem.

A Deus, pela vida e pelas oportunidades;

À minha mãe, pelo apoio incondicional, amor, amizade e presença constante, mesmo que distante, durante estes anos de estudo;

À minha esposa Rita, pelo grande amor, amizade e ajuda técnica;

Ao meu orientador, Professor Marcelo Knörich Zuffo, pela atenção que sempre me dispensou, e por ser um grande mestre e amigo;

Ao meu amigo Bruno Barberi Gnecco, pela amizade e por acreditar que poderíamos trabalhar juntos para alcançar os nossos objetivos. A sua participação foi fundamental neste trabalho;

Aos amigos da CAVERNA^{TM} Digital, o Hilton, Paulo, Márcio, Celso, Leo, Luciano, Lucas, Glauco e estagiários, pela ajuda em diversos momentos;

Aos grandes amigos que sempre acreditaram em mim, o Sandro, Zé, Renato, Bianchi, Ana Paula, Eli, Dani, Valéria, Ana Cláudia, Rogério, Cláudia e Lilian;

Aos professores e funcionários do Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo, pela criação de um ambiente de trabalho amigável e agradável, que ajudaram na minha formação profissional e pessoal.

Muito obrigado.

M.P.G.

FINANCIAMENTO

Esta tese foi possível graças ao financiamento da FINEP aos seguintes projetos:

- SISCOMPRO: **Sistemas de Comunicação e Processamento Avançados e Aplicações Interativas** - Convênio 01.02.0288.00 - Ref. 2430/02.
- HSI: **Desenvolvimento de Aplicações e Novas Gerações de Hipersistemas Integráveis** –Convênio 23.02.0464.00 - Ref. 1395/02.

RESUMO

A busca por ambientes de alto desempenho e com baixo custo vem impulsionando as pesquisas relacionadas com aglomerados de computadores. Aliado a este fato está o aumento da necessidade e interesse pelas aplicações de Realidade Virtual. Isso motiva a pesquisa e o desenvolvimento de aglomerados de computadores, bibliotecas de desenvolvimento e de estratégias de desenvolvimento de aplicações de Realidade Virtual. Este trabalho apresenta um ambiente para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos, que são aglomerados de computadores específicos para a Realidade Virtual. O ambiente desenvolvido é voltado principalmente para a construção de aplicações de multiprojeção, como as executadas em CAVERNAs^{TM} Digitais. Para testar o ambiente criado, foram desenvolvidas diversas aplicações. Além disso, são mostradas as pesquisas que estão sendo realizadas na área; uma revisão dos problemas relacionados ao uso de aglomerados gráficos; a Glass, que é a biblioteca desenvolvida neste projeto; e estratégias para o desenvolvimento das aplicações.

ABSTRACT

The search for high performance and low cost environments has stimulated the field of computer clusters. Allied to this fact is the increasing interest and the need for Virtual Reality applications. Together, they motivate the research and the development of computer clusters, libraries and of strategies for the development of Virtual Reality applications. This thesis presents an environment for the development of Virtual Reality applications, based on graphic clusters, which are computer clusters tailored for Virtual Reality. This environment was targeted at multi-projection applications, such as those executed in CAVEs (CAVE Automatic Virtual Environment). Several multi-projection applications were developed to test it. Moreover, this thesis presents an overview of the current state of the research in the area; a study of the problems related with the use of graphic clusters; Glass, a software library developed in this project; and strategies for the development of those applications.

SUMÁRIO

Lista de figuras.....	1
Notação	3
Lista de abreviaturas e siglas.....	4
1. Introdução	6
1.1 Motivação.....	9
1.2 Objetivo.....	10
1.3 Trabalhos correlatos	10
1.4 Estrutura da tese	11
2. Realidade Virtual	13
2.1 Introdução	13
2.2 A CAVERNA™ Digital	16
2.3 Trabalhos relacionados.....	17
2.3.1 Syzygy.....	20
2.3.2 OpenSG	22
2.3.3 Chromium	24
2.3.4 Net Juggler	26
2.4 Conclusão.....	29
3. Aglomerados gráficos	30
3.1 Introdução	30
3.2 Os aglomerados gráficos	30
3.2.1 A sincronização.....	32
3.2.1.1 Sincronização dos sinais de vídeo.....	32
3.2.1.2 Sincronização de dados	33
3.2.1.3 Sincronização da conclusão dos quadros	34
3.2.2 A comunicação entre os nós do aglomerado gráfico	35
3.3 O aglomerado gráfico utilizado.....	39
3.4 Requisitos das bibliotecas para aglomerados gráficos.....	40
3.5 Conclusão.....	42

4. Biblioteca para programação distribuída	44
4.1 Introdução	44
4.2 A Glass	44
4.2.1 O arcabouço Glass	47
4.2.1.1 Componente Instanciação	49
4.2.1.2 Componente Protocolo	52
4.2.1.3 Componente Plugins	54
4.2.2 Aplicações	63
4.2.2.1 Aplicações Glass	64
4.2.2.2 Aplicações de Suporte	66
4.3 Testes	76
4.3.1 Barreiras	76
4.3.2 Compartilhamento	77
4.3.3 Eventos	79
4.4 Conclusão	80
5. Desenvolvimento de aplicações	82
5.1 Estratégias de desenvolvimento	82
5.2 Estratégia de Desenvolvimento Utilizando Bibliotecas para Programação Distribuída	85
5.3 Exemplos de aplicações	91
5.4 Conclusão	97
6. Considerações finais	98
6.1 Os aglomerados gráficos na Realidade Virtual	99
6.2 A biblioteca Glass	99
6.3 A estratégia de desenvolvimento	100
6.4 Trabalhos futuros	101
6.5 Publicações	102
7. Referências bibliográficas	104

LISTA DE FIGURAS

Figura 1 – CAVERNA™ Digital gerenciada pelo aglomerado gráfico.....	8
Figura 2 – Capacete de visualização	15
Figura 3 – Luvas	15
Figura 4 – CAVERNA™ Digital da Universidade de São Paulo.....	17
Figura 5 – Arquitetura Sem Replicação	18
Figura 6 – Arquitetura Com Replicação	19
Figura 7 – Arquitetura do Syzygy	21
Figura 8 – Aplicação Syzygy	22
Figura 9 – Arquitetura da OpenSG	24
Figura 10 – Aplicação OpenSG	24
Figura 11 – Exemplo de aplicações <i>tiled</i>	26
Figura 12 – Aplicação Chromium	26
Figura 13 – VR Juggler (linhas contínuas) e Net Juggler (linhas pontilhadas)	28
Figura 14 – Aplicação Net Juggler	28
Figura 15 – Sincronização de vídeo – <i>genlock</i>	32
Figura 16 – <i>Framelock</i> via <i>software</i>	34
Figura 17 – <i>Framelock</i> via <i>hardware</i>	35
Figura 18 - O aglomerado gráfico Perfect Cluster	40
Figura 19 – Arquitetura Glass	46
Figura 20 – Diagrama de classe do arcabouço Glass.....	49
Figura 21 – Inicialização das aplicações Glass.....	50
Figura 22 – Aplicação Glass na arquitetura Sem Replicação	51
Figura 23 – Aplicação Glass na arquitetura Com Replicação.....	51
Figura 24 – Diagrama de classes Protocolo	53
Figura 25 – Processo de sincronização	55
Figura 26 – Barreira de sincronização	56
Figura 27 – Variável compartilhada.....	58
Figura 28 – Compartilhamento de variável de tipo criado.....	59
Figura 29 – Associação de métodos.....	60
Figura 30 – Entrada/Saída da variável <i>Event</i>	62
Figura 31 – Variável <i>Event</i>	63

Figura 32 – Utilização do padrão XML para integração de aglomerados remotos ...	65
Figura 33 – Escolha da aplicação no GTracer	67
Figura 34 – Depuração no GTracer.....	68
Figura 35 – GEditor	70
Figura 36 – Tipos de <i>hot spot</i>	71
Figura 37 – Aplicações com execução no PDA.....	71
Figura 38 – Camadas de <i>software</i>	72
Figura 39 – Código base para o tratamento de eventos	73
Figura 40 – O PDA é um nó do aglomerado.....	73
Figura 41 – Arquitetura do GVoicer	74
Figura 42 – Gramática.....	75
Figura 43 – GVoicer	75
Figura 44 – Barreiras (1000 sincronizações)	77
Figura 45 – Compartilhamento (1000 sincronizações) – de 1 byte até 64 kb.....	78
Figura 46 – Compartilhamento (1000 sincronizações) – de 128 kb a 4096 kb.....	78
Figura 47 – Eventos (1000 sincronizações) – de 1 byte até 64 kb.....	79
Figura 48 – Eventos (1000 sincronizações) – de 128 kb a 4096 kb.....	80
Figura 49 – Desenvolvimento Automático	83
Figura 50 – Desenvolvimento Utilizando Biblioteca para Programação Distribuída	84
Figura 51 – Imagem da aplicação em um único nó	86
Figura 52 – Código resumido da aplicação a ser portada	87
Figura 53 – Aplicação portada	88
Figura 54 – Código da aplicação portada.....	90
Figura 55 – Aplicação executada no PDA	92
Figura 56 – Usuários navegando na aplicação Celestia.....	93
Figura 57 – Celestia controlado por comandos de voz	94
Figura 58 – Imagens do Celestia.....	95
Figura 59 – Usuário navegando na aplicação Cathedral.....	95
Figura 60 – Simulador de avião GL-117.....	96
Figura 61 – Visualizador baseado no OpenSceneGraph.....	96

NOTAÇÃO

A seguir, a notação utilizada neste trabalho:

A) Termos em inglês, cuja tradução direta para o português não foi encontrada até o momento da escrita deste trabalho, são apresentados em itálico.

Exemplo: *socket*

B) Termos em inglês traduzidos são apresentados em itálico e entre parênteses à direita do termo em português.

Exemplo: rastreamento (*tracking*)

C) Quando for definido um novo termo, este é apresentado grifado.

Exemplo: rastreamento é o mecanismo de determinação de posição do usuário no interior do sistema.

D) Citações da bibliografia serão apresentadas entre colchetes, seguindo esta regra: maiúsculo as três primeiras letras do sobrenome do primeiro autor concatenado ao ano de publicação.

Exemplo: [GUI00]

Quando o autor possuir mais de uma publicação no mesmo ano, será acrescentada uma letra do alfabeto, em minúsculo, após o ano de publicação.

Exemplo: [GUI01a]

E) O código fonte das aplicações é apresentado na seguinte fonte e tamanho.

Exemplo: `int main (int argc, char **argv)`

LISTA DE ABREVIATURAS E SIGLAS

3D – Tridimensional

API – *Application Programming Interface*

CAVE – *CAVE Automatic Virtual Environment*

CORBA – *Common Object Request Broker Architecture*

E/S – Entrada/Saída

HMD – *Head Mounted Display*

IP – *Internet Protocol*

JNI – *Java Native Interface*

LSI – Laboratório de Sistemas Integráveis

MPI – *Message Passing Interface*

ORB – *Object Request Broker*

PC – *Personal Computers*

PDA – *Personal Digital Assistant*

PVM – *Parallel Virtual Machine*

RAM – *Random Access Memory*

RSTP – *Real-Time Streaming Protocol*

RSVP – *Resource Reservation Protocol*

SANs – *System Area Networks*

SPUs – *Stream Processing Units*.

TCP – *Transmission Control Protocol*

UDP – *User Datagram Protocol*

UML – *Unified Modeling Language*

VIA – *Virtual Interface Architecture*

XML – *Extensible Markup Language*

XTP – Xpress Transport Protocol

X3D – Extensible Markup Language - enabled 3D

Capítulo 1

---

1. INTRODUÇÃO

Apesar de os primeiros trabalhos em Realidade Virtual terem sido propostos na década de 60, apenas recentemente tecnologias de baixo custo têm sido disponibilizadas para a implementação de aplicações efetivas. A Realidade Virtual possibilita a interação e envolvimento imersivo de usuários com simulações, seja, por exemplo, por intermédio de dispositivos de visualização 3D estereoscópicos, seja por dispositivos de sensação de toque (dispositivos *haptic*), seja por dispositivos que capturam os movimentos dos usuários. Diversas áreas do conhecimento podem se beneficiar das aplicações de Realidade Virtual; a Engenharia, por exemplo, pode reduzir ou evitar a construção de protótipos físicos, e simular ambientes inacessíveis ou perigosos. A Medicina pode utilizar a Realidade Virtual para interpretar dados, monitorar pacientes e planejar cirurgias. Contudo, os custos têm inviabilizado o desenvolvimento dessas aplicações, em particular as de multiprojeção.

Tradicionalmente, os sistemas de multiprojeção, como as CAVERNAs^{TM} Digitais¹ (CAVE^{TM} - *CAVE Automatic Virtual Environment*), têm como núcleo de processamento gráfico os sistemas multiprocessados, também conhecidos como sistemas fortemente acoplados (*tightly coupled systems*)², de alto custo, o que os limita a grandes centros de pesquisas. Em contra partida, o desempenho de processamento geral e gráfico dos computadores pessoais (PC's), de baixo custo, estão aumentando. Sendo assim, a melhor solução e a tendência atual constitui na substituição dos sistemas fortemente acoplados por aglomerados (sistemas fracamente acoplados / *cluster* / *loosely coupled*)³, que são compostos por

---

¹ Caverna^{TM} Digital é a denominação dada ao primeiro sistema de multiprojeção totalmente imersivo e interativo desenvolvido na América Latina. Esse sistema foi construído pelos pesquisadores do Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo.

² Nos sistemas fortemente acoplados, existem vários processadores, compartilhando uma única memória e controlados por apenas um sistema operacional [MAC92].

³ Os sistemas fracamente acoplados caracterizam-se por possuir dois ou mais sistemas de computação, conectados através de uma linha de comunicação. Cada sistema tem seu próprio processador, memória, dispositivos de Entrada/Saída (E/S) e sistema operacional, podendo funcionar independentemente [MAC92].

computadores pessoais, para o gerenciamento, processamento geral e desenvolvimento de sistemas de multiprojeção [LIN02].

A principal diferença entre os aglomerados e os sistemas fortemente acoplados é a forma de distribuição e sincronização dos dados. No caso dos sistemas fortemente acoplados, a circuitaria (*hardware*) e os programas (*software*) foram projetados para eles; por isso, a distribuição e sincronização dos dados são realizadas automaticamente pelo sistema operacional, via barramento interno; no caso dos aglomerados, a distribuição e a sincronização dos dados são realizadas por *softwares* de comunicação via rede, sendo a latência maior.

Dessa maneira, os aglomerados parecem ter diversas desvantagens, mas contêm substanciais benefícios também. Os sistemas fortemente acoplados possuem limitações na escalabilidade (memória, placas, dispositivos de saída), com reflexos na melhora de desempenho. Por outro lado, utilizando-se um *software* apropriado, os aglomerados não sofrem da limitação de escalabilidade, podendo ter número ilimitado de nós⁴.

As vantagens e desvantagens dos aglomerados vêm sendo estudadas pelos pesquisadores de Realidade Virtual [LIN02, ZUF01, ZUF02a, SGI01, SCH02c]. Eles têm realizado importantes passos para permitir a solução dos problemas, de modo que se possa, de fato, ter ambientes virtuais de propósito geral baseados em aglomerados. Os aglomerados projetados para as aplicações de Realidade Virtual são compostos por computadores pessoais que dispõem de *hardware* dedicado a esse tipo de problema, como placas gráficas, então denominados aglomerados gráficos (VRCluster).

A Figura 1 mostra o ambiente de pesquisa deste trabalho, que é formado por um aglomerado gráfico (heterogêneo - sistema operacional e arquitetura de *hardware* -), o qual gerencia e processa os dados que são apresentados na CAVERNA™ Digital. Neste ambiente, os projetores e espelhos estão dispostos atrás de cada parede. Os usuários se posicionam dentro da CAVERNA™ Digital para visualizarem as imagens estereoscópicas, através do uso de óculos obturadores (*shutter glasses*).

---

⁴ Esta capacidade de escalabilidade dos aglomerados (memória, placas, dispositivos de saída) não garante o ganho de desempenho das aplicações, pois este também depende de fatores como latência de comunicação e dependência entre as tarefas [HWA93].

Figura 1 – CAVERNA™ Digital gerenciada pelo aglomerado gráfico

São apresentados nesta pesquisa os principais problemas e soluções do desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos. Além disso, apresenta a biblioteca desenvolvida neste projeto, a Glass, que visa facilitar o desenvolvimento de tais aplicações. Esta biblioteca fornece recursos de comunicação de dados, suporte a processamento distribuído, gerenciamento distribuído, acesso distribuído dos dados e de distribuição dos dados dos dispositivos, e ferramentas que auxiliam o desenvolvimento de aplicações ou de novos recursos para esta biblioteca. Por fim, apresentam-se uma estratégia específica para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos e exemplos de aplicações desenvolvidas. Assim, criou-se um ambiente que permite o desenvolvimento e execução de aplicações de Realidade Virtual baseadas em aglomerados gráficos.

Diferentes áreas do conhecimento estão envolvidas nessa pesquisa, por exemplo: protocolos de rede, paralelismo, sistemas distribuídos, computação gráfica, interfaces e outras. O estudo dessas áreas é necessário para resolver problemas como: gerenciamento de consistência das informações compartilhadas, interatividade em tempo real, processamento e geração de imagens.

Essa pesquisa não irá substituir as soluções tradicionais, que são sistemas computacionais limitados à capacidade de processamento de uma máquina (sistemas fortemente acoplados) ou as ferramentas de processamento distribuído ou paralelo, pois existem diversas aplicações que foram desenvolvidas com a sua utilização [OLS02, OVR03]. Contudo, a procura de uma melhor escalabilidade e custo,

estimula a exploração dos recursos fornecidos pelos aglomerados gráficos, para que a Realidade Virtual possa ser utilizada pelos centros de pesquisas, indústria, comércio, educação e em outras áreas.

As aplicações de Realidade Virtual baseadas em aglomerados gráficos dependem de um conjunto amplo e sempre evolutivo de tecnologias, que inclui desde a infra-estrutura de comunicação até a capacidade de geração e apresentação de imagens de maneira sincronizada pelos diversos nós. Se por um lado, esta dependência por novas tecnologias causa problemas, por outro cria uma estimulante demanda por serviços desta área e de outras, pois os motivos para o crescimento dessas aplicações são mais fortes, como a seção a seguir apresenta.

1.1 Motivação

A Realidade Virtual é uma área do conhecimento que oferece inúmeras oportunidades de investigação científica e inovação tecnológica. A possibilidade aberta para que as comunidades científicas e industriais possam criar ambientes virtuais imersivos e interativos tem sido considerada a fonte de motivação para o estudo do uso dos aglomerados gráficos. Além disso, as seguintes perspectivas motivam esta pesquisa:

- A diminuição dos custos, pois os aglomerados gráficos são relativamente baratos quando comparados com os sistemas fortemente acoplados;
- A criação de um ambiente que forneça os recursos para o desenvolvimento e execução de aplicações de Realidade Virtual baseadas em aglomerados gráficos. Na literatura internacional científica, somente nos últimos anos começaram a surgir pesquisas nesta área, o que permite observar o enorme potencial de crescimento; e
- Os recentes investimentos das indústrias na produção de *hardware*, *software* e dispositivos de Entrada/Saída têm gerado o crescimento acelerado da Realidade Virtual nos últimos anos e indicado perspectivas bastante promissoras para os diversos segmentos vinculados com a área [KIR03];

Para que essas motivações façam parte das aplicações de Realidade Virtual, é imprescindível que o processo de desenvolvimento de tais aplicações seja facilitado e que as ferramentas de desenvolvimento atendam os requisitos específicos destas aplicações, como: baixa latência de rede, alta largura de banda e suporte para sincronização e comunicação de dados. Alguns desses requisitos podem ser encontrados em algumas APIs (*Application Programming Interface*) para a Realidade Virtual ou para aglomerados, mas, geralmente, estas APIs são difíceis de utilizar ou exigem o redesenvolvimento das aplicações.

1.2 Objetivo

Esse projeto tem como objetivo a pesquisa e desenvolvimento de um ambiente para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos, que permita ao desenvolvedor facilmente criar/portar aplicações para serem executadas em ambientes de multiprojeção, como as executadas em CAVERNAs^{TM} Digitais. Este ambiente deve fornecer características como: facilidade de uso, desempenho, flexibilidade e portabilidade. Além disso, esse trabalho tem como objetivo demonstrar a viabilidade do uso de aglomerados gráficos para as aplicações de Realidade Virtual.

1.3 Trabalhos correlatos

Essa tese insere-se junto a projetos de pesquisas relacionados com Realidade Virtual de alto desempenho, que estão sendo desenvolvidos na área de Meios Eletrônicos Interativos no Laboratório de Sistemas Integráveis da Universidade de São Paulo. A seguir, a descrição sucinta de alguns projetos correlatos:

- **Visualização Volumétrica aplicada em aglomerados gráficos de computadores convencionais:** este projeto implementa técnicas de distribuição de dados em sistemas computacionais para sintetização de dados volumétricos. Ele apresenta como gerenciar estes dados em aglomerados de computadores, conectados em rede, utilizando o máximo da hierarquia de memória [BRE01];

- **Uma Plataforma para Geração Interativa de Imagens por traçado de raios (*ray tracing*):** este trabalho se propõe a desenvolver uma plataforma capaz de gerar imagens via traçado de raios em tempo real, podendo ser manipuladas interativamente. Este projeto também utiliza a Glass (biblioteca proposta e desenvolvida nessa tese);
- **JINX: Um navegador X3D (*Extensible Markup Language - enabled 3D*) para simulações de Realidade Virtual:** este projeto visa fornecer mecanismos para o desenvolvimento de aplicações de Realidade Virtual no formato X3D e um navegador X3D para visualização dessas aplicações [SOA04b].

O restante deste capítulo apresenta uma visão geral sobre a organização desta pesquisa e um sumário sobre o conteúdo dos capítulos posteriores.

1.4 Estrutura da tese

O Capítulo 2 apresenta uma introdução aos diversos aspectos relacionados à Realidade Virtual. A seguir é mostrada a CAVERNA^{TM} Digital, que é o ambiente de multiprojeção utilizado neste trabalho. Ao final, são apresentados diversos trabalhos relacionados a esta pesquisa.

O Capítulo 3 introduz os conceitos gerais sobre os aglomerados gráficos e apresenta a plataforma Perfect Cluster, que é o aglomerado gráfico utilizado nesta pesquisa. A seguir, apresenta-se uma análise de problemas a serem solucionados, desde os requerimentos relativos ao aspecto distribuído do sistema, como a comunicação, até a sincronização das imagens. Por fim, apresentam-se os requisitos que uma biblioteca de desenvolvimento para aglomerados gráficos deve oferecer.

O Capítulo 4 mostra a especificação e a implementação da biblioteca Glass, tendo como base os requisitos apresentados no capítulo anterior. Além disso, são apresentados os resultados dos testes realizados nos componentes da Glass, que são analisados para se caracterizar o comportamento do sistema e verificar a validade desta biblioteca.

O Capítulo 5 apresenta, inicialmente, as estratégias de desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos. A seguir, mostra a estratégia criada para o desenvolvimento utilizando bibliotecas de programação para a computação distribuída, como a Glass, e exemplos de aplicações desenvolvidas.

Por fim, o Capítulo 6 expõe as principais conclusões, bem como as contribuições geradas pelo trabalho, e também enfatiza alguns tópicos importantes para estudos futuros.

Capítulo 2

---

2. REALIDADE VIRTUAL

Este capítulo apresenta inicialmente uma visão geral dos conceitos de Realidade Virtual e, em seguida, o dispositivo de multiprojeção utilizado nesta pesquisa, a CAVERNA^{TM} Digital. Ao final, são apresentados alguns trabalhos que implementam soluções de interesse dessa pesquisa.

A Realidade Virtual tem fornecido novos meios para melhorar a interface e a interação dos usuários com os sistemas computacionais. Tem permitido a imersão e a interação dos usuários com ambientes sintéticos gerados por computador, através da exploração dos sentidos fundamentais do corpo humano, como a visão, a audição, o tato e o olfato.

2.1 Introdução

A Realidade Virtual é um acervo de conhecimentos e tecnologias relacionadas com a imersão e interação dos usuários com mundos sintetizados. Estes mundos podem ser explorados, visualizados e manipulados em tempo real. As principais características deles são a imersão e interatividade: a imersão pelo seu poder de prender a atenção do usuário; e a interatividade no que diz respeito à capacidade que o sistema tem de dar respostas às ações do usuário [KIR03, KIR01, FOR97, CAP99]. Alguns autores, como [GRI94], definem a Realidade Virtual como uma simulação que utiliza a computação gráfica para criar visões realísticas de mundos.

Para ser caracterizado como sistema de Realidade Virtual, as aplicações devem possuir algumas características fundamentais. Elas não precisam necessariamente explorar ao máximo todas estas características, mas a ausência total de um desses itens pode comprometer a classificação do sistema como sendo de Realidade Virtual [NET02, COS00, PIN96]:

- **Interface de alta qualidade:** a interface da aplicação deve ser bem projetada a fim de oferecer uma interação amigável e o mais natural possível, de modo que o usuário se sinta encorajado e bem à vontade para usá-la. Esta interface

é tida como a de mais alto nível, pois permite que o homem interaja intuitivamente com a máquina, por reproduzir o que acontece na interação deste com o mundo real;

- **Alta interatividade:** o ambiente deve permitir que as interações dos usuários sejam em tempo real, com uma pequena ou nenhuma consciência de que estão utilizando uma interface. Por exemplo, em uma simulação de corrida, o acelerador deve responder às ações do piloto imediatamente após a ação (millessegundos). Esta característica deve ser atendida para que interfaces de alta qualidade sejam construídas;
- **Imersão:** uma aplicação de Realidade Virtual deve ser capaz de permitir que o usuário se sinta dentro do mundo virtual, seja com o seu corpo físico (por exemplo, dirigindo um veículo dentro de um cockpit) ou não (por exemplo, interagindo com um robô remoto). A imersão dos usuários pode ser obtida com o uso de capacete de visualização, luvas, som 3D e outros dispositivos;
- **Analogia com o mundo real:** refere-se à exploração da intuição dos usuários, possibilitando, assim, novas formas de interação. Por exemplo, o desenvolvedor de um sistema de simulação de veículo deve projetá-lo utilizando a experiência do usuário com o mundo real, como, o conhecimento das sensações de mudança de marchas e giro do volante. O fato de poder utilizar a intuição dos usuários faz com que a aplicação de Realidade Virtual atue como uma transferência do mundo real para o virtual;
- **Adição de aspectos não existentes no mundo real:** trata-se de acrescentar aspectos que não existem no mundo real. Por exemplo, em uma biblioteca virtual o sistema pode destacar os volumes que acredita serem do interesse do usuário, enquanto ele se locomove e olha as estantes, aproveitando assim a habilidade de locomoção do usuário.

O atendimento dessas características é essencial para a construção de aplicações de Realidade Virtual cada vez mais sofisticadas, ou seja, que possuam um grande número de recursos, alto nível de interação e imersão e, ao mesmo tempo, que satisfaçam os usuários.

Para obter a satisfação dos usuários, as aplicações de Realidade Virtual envolvem um controle tridimensional altamente interativo de processos computacionais. Os usuários entram no espaço virtual das aplicações, visualizam, manipulam e exploram os dados das aplicações em tempo real, usando seus sentidos, particularmente os movimentos naturais tridimensionais do corpo. Por isso, o conhecimento intuitivo dos usuários a respeito do mundo físico pode ser utilizado para manipular o mundo virtual [MEI99]. Para permitir a manipulação e/ou visualização, os usuários utilizam dispositivos não convencionais como capacete de visualização (HMD [*Head mounted display*], Figura 2), luvas (*Gloves*, Figura 3), trajes, *mouses*, entre outros [ARC01, KIR03, OLS02, VIR03, CRU92, BUR94].

Figura 2 – Capacete de visualização [VIR03]

Figura 3 – Luvas [VIR03]

Dispositivos como esses dão aos usuários a impressão de que a aplicação está funcionando no ambiente real, permitindo a exploração e a manipulação natural dos objetos com o uso das mãos, por exemplo, para apontar, pegar, e realizar outras ações. A exploração e manipulação do ambiente requerem um alto desempenho computacional para que os usuários se sintam imersos em um ambiente real.

Desde os primórdios da Realidade Virtual, o fator desempenho sempre mereceu grande atenção dos desenvolvedores, lado a lado com requisitos como realismo gráfico e ergonomia. O desempenho tem influência direta na qualidade de uma

aplicação e é geralmente medido em número de quadros (*frames*) por segundo. Um baixo desempenho pode prejudicar a usabilidade ou, em casos mais graves, até mesmo inviabilizar a implementação de uma boa idéia.

A próxima seção apresenta a CAVERNA^{TM} Digital. Explorando a visão, audição, fala e o tato dos seres humanos, ela é um dispositivo de visualização que busca atender as principais características das aplicações de Realidade Virtual, a imersão e interatividade.

2.2 A CAVERNA^{TM} Digital

A CAVERNA^{TM} Digital do Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo é um sistema de visualização para a Realidade Virtual de alta resolução, formada por um sistema de multiprojeção, que disponibiliza um ambiente virtual totalmente imersivo e interativo [ZUF01, SAN01, GUI02a]. Ela é composta por teclado, sistema de luz e som, projetores, óculos obturadores, rastreadores de posição, telas para projeção das imagens e chinelos apropriados para pisar na superfície da tela, já que todos os cinco lados da instalação são forrados por tela produzida em lona especial capaz de distribuir homogeneamente a imagem e som, sem perda de qualidade.

A CAVERNA^{TM} Digital possui uma dimensão útil de 3x3x3 metros. Cada uma das paredes é capaz de exibir imagens com resolução superior a 2.000 x 2.000 pixels. Os usuários visualizam as imagens em um ângulo que varia de acordo com a distância entre os olhos deles; em média, eles visualizam em um ângulo de 110° sem interrupções. Isso é possível devido o campo visual não estar limitado às dimensões da tela de projeção, visto que esta envolve o usuário. Com isso, pode-se estar dentro do objeto a ser estudado, - seja ele microscópico, como uma célula, ou macroscópico, como uma galáxia -, e estudar com detalhes todo seu funcionamento, interagindo com o mundo projetado nas telas.

Para gerar as imagens, existem duas configurações que podem ser usadas: a primeira, baseada em um aglomerado gráfico denominado Perfect Cluster (descrito na seção 3.3); e a segunda, em um único supercomputador Silicon Graphics Onyx

III. Toda a instalação é conectada por meio de fibra óptica no padrão Gigabit Ethernet.

Até seis pessoas podem estar no interior da CAVERNA^{TM} Digital e interagir com um mundo simulado por computador. A Figura 4 (a) mostra um usuário na porta da CAVERNA^{TM} Digital e um desenvolvedor manipulando o sistema gerenciador. A Figura 4 (b) apresenta o protótipo construído para simular a posição dos espelhos e a abertura da porta da CAVERNA^{TM} Digital.

(a)

(b)

Figura 4 – CAVERNA^{TM} Digital da Universidade de São Paulo

A próxima seção apresenta um estudo de suportes existentes para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos, que foi realizado como forma de obter uma visão geral sobre a área.

2.3 Trabalhos relacionados

O objetivo do estudo de trabalhos que implementam soluções de interesse desta pesquisa foi buscar idéias e soluções já experimentadas de problemas, ou simplesmente buscar material para novas soluções. Nesse estudo, notou-se que:

- Existem diversas soluções válidas para o desenvolvimento de aplicações de Realidade Virtual, sendo cada uma delas melhor em determinadas situações;
- Não existe uma solução comum para todas as aplicações de Realidade Virtual;
- O reuso de componentes pode facilitar o desenvolvimento, manutenção e o gerenciamento dos aplicativos;

- A transmissão de dados, a sincronização de dados e o gerenciamento das aplicações são os maiores desafios para a execução de aplicações de Realidade Virtual em aglomerados gráficos;
- O processo de desenvolvimento de novas aplicações e de porte das aplicações legadas (desenvolvidas para sistemas fortemente acoplados) deve ser facilitado. Somente assim, os aglomerados gráficos serão amplamente utilizados.

Os projetos encontrados na literatura foram implementados utilizando-se a arquitetura Mestre/Escravo Com Replicação ou a Mestre/Escravo Sem Replicação. Na arquitetura Sem Replicação, apresentada na Figura 5, não há replicação da aplicação nos nós do aglomerado gráfico. O nó Mestre trata as interações do usuário, realiza a sincronização do ambiente, constrói e envia as primitivas gráficas para os nós Escravos, os quais geram (*render*) e apresentam as imagens. Nesta solução, cada nó recebe apenas as primitivas gráficas referentes às imagens que irá gerar. Existem variações na implementação desta arquitetura, por exemplo [CHE01b] apresenta uma solução onde o nó Mestre trata as interações do usuário, realiza a geração das imagens e envia os pixels para os nós Escravos apresentarem. Por isso, a capacidade necessária de processamento e gráfica dos nós depende da solução adotada e da complexidade da aplicação.

Figura 5 – Arquitetura Sem Replicação

As bibliotecas para aglomerados gráficos que permitem a criação de aplicações Sem Replicação possibilitam que os programas sejam projetados como se fossem ser executados em apenas um nó, sendo o processo de envio das primitivas gráficas ou pixels para os outros nós transparente para o desenvolvedor, ou seja, sem a necessidade de criação de rotinas de transmissão ou codificação de dados. Estas bibliotecas buscam fornecer serviços capazes de obter todas as primitivas gráficas da aplicação e fazer a divisão destas entre os nós Escravos.

Na arquitetura Com Replicação, apresentada na Figura 6, cópias idênticas da mesma aplicação são executadas em cada nó. O nó Mestre recebe as interações do usuário e envia para os nós Escravos de forma sincronizada, assegurando a consistência de dados entre as diversas instâncias. Os nós Escravos recebem estas interações, tratam-nas, gerando as imagens e apresentam de maneira sincronizada, garantindo então a apresentação dos quadros ao mesmo tempo. Esta arquitetura pode variar, por exemplo, em algumas soluções, como na Glass, o nó Mestre pode também ser um nó Escravo [GUI04].

Figura 6 – Arquitetura Com Replicação

As bibliotecas que possibilitam o desenvolvimento de aplicações na arquitetura Com Replicação exigem que o desenvolvedor adicione o código referente à distribuição e tratamento dos dados nas aplicações.

A arquitetura Sem Replicação exige mais largura de banda e é menos flexível (cada nó tem uma função específica) do que a Com Replicação. Entretanto, a Com Replicação exige que todas as mudanças de estados que possam afetar a execução do programa sejam tratadas e enviadas para todos os nós. Assim, números aleatórios gerados, temporizadores, ou qualquer variável que possa influenciar a execução do programa devem ser distribuídos e sincronizados entre os nós. Todos os nós desta arquitetura devem ter alta capacidade de processamento, pois executam a aplicação e geram as imagens (pode-se incluir o nó Mestre).

A escolha da arquitetura a ser utilizada depende da avaliação de um conjunto de fatores, como: a complexidade de implementação, a velocidade dos processadores, as interações, a capacidade de comunicação, o custo, a sincronização e a capacidade de processamento gráfico dos nós.

Essa taxonomia apresentada como arquitetura Mestre/Escravo Com Replicação e Mestre/Escravo Sem Replicação não está de acordo com a classificação apresentada por [SCH02b, SCH02c, STA03, CHE01a, CHE01b]. Esses autores nomeiam a arquitetura Com Replicação como sendo arquitetura Cliente/Servidor e a arquitetura Sem Replicação como sendo arquitetura Mestre/Escravo. Entretanto, como os nós que apresentam as imagens são dependentes das instruções vindas dos nós Mestres, independentemente do modelo de implementação, ou seja, eles executam apenas as instruções recebidas (o Mestre detém todas as tarefas), então denominou-se nesta pesquisa as duas arquiteturas como sendo Mestre/Escravo, tendo como diferença a replicação ou não da aplicação nos nós.

2.3.1 Syzygy

Desenvolvida na Universidade de Illinois em Urbana-Champaign, a Syzygy é uma solução projetada para a execução de aplicações de Realidade Virtual em aglomerados gráficos [SCH02b, SCH02c]. A Figura 7 apresenta a arquitetura desta biblioteca. A primeira camada, a de Comunicação, fornece ferramentas para a construção de protocolos de comunicação, realiza a conversão automática de diferentes formatos de dados, e fornece um ambiente de rede confiável com robusta conexão, desconexão e reconexão. Os programadores definem um dicionário de moldes (*template*) de mensagens, o qual define a semântica das mensagens que

podem ser transmitidas na rede. Os objetos transferidos são compostos por informações de gerenciamento da conexão e por dados que serão tratados pelas aplicações. Esta camada permite que as aplicações trabalhem com TCP (*Transmission Control Protocol*), UDP (*User Datagram Protocol*) ou com memória compartilhada (*pipes*).

Figura 7 – Arquitetura do Syzygy [SCH02b]

A segunda camada é formada pelos Protocolos de Mídia que provêm métodos para a realização da sincronização, distribuição e geração das imagens de grafos de cenas; manipulação e execução do banco de dados de som; e comunicação com os dispositivos de Entrada/Saída. Além disso, essa camada é composta pelo Phleet, que é um sistema operacional distribuído (simplificado) para objetos de mídia, como tocadores de som e geradores de imagens remotos de grafo de cena. O Phleet também permite que as aplicações sejam inicializadas ou finalizadas remotamente, utilizando-se comandos de linha ou em uma interface Java.

A terceira camada, a de Objetos Gráficos, é composta por programas de gerenciamento, como, o szgserver, que gerencia as aplicações Syzygy. Este aplicativo armazena todos os atributos dos nós, dados dos usuários e informações sobre outros programas. Outro exemplo de programa de gerenciamento é o SoundRender, que distribui o som para os nós do aglomerado gráfico.

Por fim, na última camada, a de Aplicações Syzygy, tem-se as aplicações que podem ser implementadas na arquitetura Com Replicação. Para isso, as aplicações podem ser implementadas utilizando-se: uma API que distribui as modificações do grafo de cena para cada nó conectado, cabendo a cada um deles utilizar esta

informação para desenhar o próximo quadro (*frame*); ou um arcabouço⁵(*framework*) que fornece recursos para a replicação das aplicações. Syzygy suporta os sistemas operacionais Windows, Linux e SGI Irix.

Syzygy envolve-se em vários problemas a fim de atingir o objetivo proposto, como: distribuição dos objetos gráficos, distribuição do som, gerenciamento de sistemas distribuídos, acesso a dados distribuídos e gerenciamento dos dispositivos de Entrada/Saída. Existem vários sistemas que resolvem cada um destes problemas de forma separada, mas nenhum de forma geral. Syzygy é uma tentativa de identificação e implementação da infra-estrutura comum para aplicações baseadas em aglomerados gráficos.

A Figura 8 apresenta um exemplo de uma aplicação Syzygy. O dispositivo de multiprojeção utilizado é uma CAVERNA^{TM} Digital.

Figura 8 – Aplicação Syzygy [SCH02b]

2.3.2 OpenSG

Desenvolvida por diversos grupos de pesquisa, sendo a maioria da Alemanha, a OpenSG é uma biblioteca que utiliza grafo de cena para a geração de imagens em

---

⁵ Aplicações baseadas em um arcabouço são construídas customizando suas classes. Reusar um arcabouço, normalmente, significa adaptar sua estrutura para necessidades específicas. Além da reutilização de código, que é oferecido por classes e objetos, os arcabouços também oferecem a reutilização de projeto, liberando o desenvolvedor dos aspectos comuns daquele domínio de aplicação [SOU98].

tempo real, que pode ser estendida para aglomerados gráficos e que fornece suporte para estereoscopia [OPE03a]. O seu principal objetivo é fornecer um ambiente de manipulação de estrutura de dados em múltiplos processos-leves seguros (*multithreaded-safe*), que não comprometa o desempenho. Para isso, o controle dos processos-leves (*threads*) é realizado de maneira explícita, ou seja, os desenvolvedores devem se preocupar com os travamentos (*lock*) e destravamentos (*unlock*) dos processos-leves, o que dificulta o uso desta biblioteca. A OpenSG fornece recursos para que um grafo de cena seja manipulado por processos-leves independentes [REI02a, REI02b, REI02c].

Esses processos-leves geram a necessidade de replicação de dados, mas o esquema de organização de dados (dados separados da estrutura da cena) faz com que apenas os dados necessários sejam replicados. Além disso, cada uma delas possui uma lista de atributos alterados. Esta lista é utilizada para que a sincronização seja feita apenas dos dados que foram alterados.

OpenSG possui diversos recursos para a criação de objetos, sendo possível construir um vetor com a indicação de como as estruturas devem ser criadas, como por exemplo, triângulos, linhas, pontos e malhas de triângulos. Além disso, permite a reorganização dos dados por intermédio de índices.

Essa biblioteca fornece a possibilidade de ser estendida. Por exemplo, pode ser utilizada para escrever novos leitores de formatos de imagem. Atualmente, ela disponibiliza leitores de VRML e OBJ [OPE03a]. A sua utilização em aglomerados gráficos é uma extensão que utiliza a tecnologia de *sockets* para informar cada nó as alterações dos valores dos dados, e para fazer a sincronização das cópias remotas com a local (arquitetura Com Replicação) [CLU04].

A Figura 9 apresenta a arquitetura da OpenSG, em cuja base há as Bibliotecas Básicas, que são compostas por funções de baixo nível e classes bases, por exemplo, vetores e volumes. Ainda, contém uma abstração do sistema operacional e empacotadores (*wrappers*) básicos para os múltiplos processos-leves. A camada intermediária, a de Bibliotecas de Sistema, contém os objetos de alto nível, como, os métodos de manipulação de grafos de cena, os leitores de imagens e cenas, e as classes de manipulação de janelas. Por fim, na camada de Aplicações OpenSG, estão

as aplicações que podem ser executadas atualmente nos sistemas operacionais Irix, Linux e Windows.

Figura 9 – Arquitetura da OpenSG

A Figura 10 apresenta um exemplo de aplicação OpenSG. O dispositivo de multiprojeção utilizado é uma parede de multiprojeção (*PowerWall*) -(5 x 4).

Figura 10 – Aplicação OpenSG [REI02c]

2.3.3 Chromium

Desenvolvida na Universidade de Stanford, Chromium é um sistema que manipula fluxos (*streams*) de comandos gráficos em aglomerados gráficos. Ela oferece uma interface OpenGL [OPE03a, OPE03b] para a geração de imagens, permitindo a execução em taxas interativas, mesmo com o aumento das resoluções [HUM99, HUM00, HUM01, HUM02, LEC03].

Esta biblioteca utiliza a arquitetura Sem Replicação, suportando Windows, Irix e Linux. As aplicações não sofrem modificações para serem executadas com a

utilização do sistema Chromium. Para isso, o nó Mestre obtém os comandos OpenGL da aplicação local, transformam-os para o formato proprietário Chromium e enviam-os para os nós Escravos. Estes nós recebem os comandos gráficos da rede e geram as imagens, o que exige alta largura de banda de rede. As aplicações podem ser executadas utilizando-se a técnica de geração de imagem paralela *sort-first* ou a *sort-last* [PAI98]. Toda comunicação é baseada em *sockets*.

A Chromium permite que a quantidade de nós Mestres e Escravos varie (n -Mestres X n -Escravos) [HUM02]. Isso é possível porque ela foi construída a partir de três passos que foram identificados no envio do fluxo de comandos: o primeiro passo é a classificação dos comandos de acordo com o particionamento realizado na imagem (*tiles*); o próximo é o envio dos fluxos de comandos para uma implementação local de OpenGL; e o último, a leitura do acumulador de quadros (*framebuffer*) e a criação de novos fluxos de comandos para a geração de imagens.

Para a execução desses passos, os nós foram divididos em duas partes: os de transformação e os de serialização. Os de transformação (nós Mestres) têm como responsabilidade receber um único fluxo de comandos OpenGL da aplicação, e produzir um ou mais fluxos de comandos OpenGL como saída. Os de serialização (nós Escravos) têm como finalidade consumir um ou mais fluxos de dados e produzir um único fluxo de comandos OpenGL como saída (apresentar as imagens).

Essa biblioteca é formada por SPUs (*Stream Processing Units*), que são programas que realizam atividades específicas, como, execução de comandos OpenGL, inicializações e finalizações. Assim, as transformações e serializações são realizadas por SPUs específicas. Chromium fornece diversas SPUs, por exemplo, uma que implementa a técnica *sort-first* (SPU Tilesort) e outra que implementa a *sort-last* (SPU readback).

A Figura 11 mostra dois exemplos de *tiled display*. A Figura 11(a) mostra a projeção sendo realizada em um PowerWall e a Figura 11(b) em uma CAVERNA^{TM} Digital. Nestes exemplos, o nó Mestre executa uma aplicação e a SPU Tilesort. Esta SPU obtém os comandos OpenGL da aplicação, classifica-os e transformam-os no formato Chromium. Após isso, o nó Mestre envia os comandos gráficos para os nós Escravos, para que estes gerem as imagens. Os nós Escravos geram as imagens

utilizando os SPUs Render (*server nodes*) e as enviam a um dispositivo de saída (geralmente um projetor). A sincronização da apresentação dos quadros é realizada diretamente entre os nós Escravos.

(a) PowerWall [ZUF02a]

(b) CAVERNA™ Digital

Figura 11 – Exemplo de aplicações *tiled*

A Figura 12 apresenta um exemplo de aplicação desenvolvida que utiliza a Chromium. O dispositivo de multiprojeção utilizado é um *PowerWall* (3 x 4).

Figura 12 – Aplicação Chromium [HIG04]

2.3.4 Net Juggler

Desenvolvida na Universidade de Orleans, Net Juggler é uma camada escrita sobre a biblioteca VR Juggler, que gerencia a comunicação dos nós de um

aglomerado gráfico para a geração de uma única imagem. Ela encapsula as funcionalidades da VR Juggler e adiciona os recursos necessários para a execução em aglomerados gráficos, como soluções de comunicação e sincronização [ALL02a, BIE00].

A Net Juggler replica a aplicação VR Juggler em cada nó do aglomerado gráfico e distribui (*broadcast*) os eventos dos dispositivos de entrada para os nós, garantindo assim a coerência de dados entre as cópias (arquitetura Com Replicação).

Para enviar os dados, a Net Juggler usa MPI (*Message Passing Interface*) [ALL02a]. MPI é um protocolo tradicional desenvolvido para a comunicação de aplicações de alto desempenho. O uso dessa biblioteca dificulta modificações dinâmicas na estrutura do aglomerado gráfico ou na configuração da aplicação que está executando. Além disso, o uso de MPI introduz uma nova ferramenta que os desenvolvedores precisam aprender para desenvolver as aplicações.

VR Juggler é um arcabouço que esconde a complexidade da infra-estrutura e das especificações dos nós (dispositivos, monitores e computadores). As aplicações são independentes dos dispositivos de Entrada/Saída. Os desenvolvedores podem acessar diretamente as API gráficas (OpenGL ou OpenGL Performer) e utilizar diversos dispositivos como luvas, *mouses*, monitores e capacetes de visualização [ALL02a, OLS02, BIE00]. Segundo [OLS02], é possível que as aplicações sejam construídas utilizando-se a arquitetura Com Replicação ou a Sem Replicação.

A Figura 13 mostra a VR Juggler e os recursos adicionados para a execução em aglomerados gráficos (Net Juggler). Este ambiente é composto por um núcleo e por diversos componentes denominados gerentes. Cada gerente comanda um conjunto de detalhes específicos do sistema, por exemplo, os dispositivos de entrada são controlados pelo Gerente de Entradas e as API gráficas são encapsuladas no Gerenciador de Desenho. O núcleo controla a execução do sistema e a comunicação entre os diferentes gerentes. Uma interface gráfica, a VjControl, é fornecida para configurar os dispositivos de *hardware*, como: monitores e dispositivos de entrada [ALL02c].

Figura 13 – VR Juggler (linhas contínuas) e Net Juggler (linhas pontilhadas) [ALL02c]

Net Juggler suporta os mesmos sistemas operacionais que o VR Juggler (IRIX, Linux, Windows, Free BSD e Solaris) [OLS02, ALL02a, BIE00, NET04].

A Figura 14 apresenta um exemplo de aplicação Net Juggler. O dispositivo de multiprojeção utilizado é uma CAVERNA^{TM} Digital.

Figura 14 – Aplicação Net Juggler [NET04]

As bibliotecas apresentadas nesta seção oferecem soluções para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos, entretanto elas são complexas, muitas não suportam processos-leves, não são portáteis e/ou interoperáveis; e algumas são dependentes de bibliotecas e/ou de protocolos específicos. Isso ocorre principalmente porque elas foram adaptadas ou

podem ser adaptadas a este tipo de problema. Isso motivou essa pesquisa a desenvolver a biblioteca Glass. Além disto, motivou a criação da estratégia de desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos.

2.4 Conclusão

O desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos envolve estudos e recursos ligados com *hardware*, *software*, percepção, interface do usuário, fatores humanos e aplicações. Para auxiliar o desenvolvimento, diversas bibliotecas estão sendo desenvolvidas, mas, até o momento, nenhuma oferece todos os recursos necessários para este domínio de aplicação.

Este capítulo apresentou inicialmente uma visão geral da Realidade Virtual e as bibliotecas mais relevantes no momento para o desenvolvimento de aplicações dela baseadas em aglomerados gráficos. As que utilizam a abordagem Com Replicação requerem que todos os nós sejam capazes de gerar imagens e uma baixa largura de banda. Nesta abordagem, um nó provê a mesma entrada de dados para todos os outros e realiza a sincronização. Além disso, ocorre o reuso de código, pois as aplicações são replicadas em todos os nós. Por outro lado, as que utilizam a abordagem Sem Replicação necessitam de alta largura de banda de rede e nós com funções específicas. Essa taxonomia, a arquitetura Mestre/Escravo Com Replicação e Mestre/Escravo Sem Replicação, é uma das contribuições dessa pesquisa.

As bibliotecas apresentadas foram inicialmente projetadas para gerarem um único ponto de vista, utilizando sistemas fortemente acoplados. Mas, devido à necessidade de sistemas de multiprojeção, como a das CAVERNAs^{TM} Digitais, e ao interesse do uso de aglomerados, pesquisas estão surgindo a fim de estendê-las, para que seja possível criar vários pontos de vista da mesma aplicação de Realidade Virtual baseada em aglomerados gráficos.

Capítulo 3

---

3. AGLOMERADOS GRÁFICOS

O uso de aglomerados tem sido estudado por diversas áreas da engenharia e ciência da computação. Neste capítulo, serão apresentados os aglomerados projetados para as aplicações de Realidade Virtual, os aglomerados gráficos. Serão apresentados os conceitos básicos; os problemas a serem resolvidos, como a comunicação e a sincronização; e os requisitos que as bibliotecas devem atender para permitir o seu uso no desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos.

3.1 Introdução

O desenvolvimento acelerado de arquiteturas de redes e dos computadores pessoais propiciou avanços significativos na área de processamento de alto desempenho. Os sistemas fortemente acoplados passaram a serem substituídos por aglomerados. Os aglomerados são caracterizados por uma coleção de nós interconectados por intermédio de uma rede local denominada SAN (*System Area Network*)⁶, dando ao usuário a ilusão de um único sistema [KIW01]. Essa arquitetura expandiu-se rapidamente devido, principalmente, ao seu baixo custo, alta flexibilidade e escalabilidade [BUY99].

3.2 Os aglomerados gráficos

Essa pesquisa utiliza um tipo especial de aglomerado, os gráficos, os quais diferem dos tradicionais. Os tradicionais dividem a tarefa em pequenas tarefas, as distribuem para os nós, e logo após o processamento das tarefas, realizam a sincronização das informações no final. Por outro lado, os aglomerados gráficos têm como objetivo oferecer uma visão múltipla do mesmo conjunto de dados, para isto,

---

⁶ *System Area Networks (SAN)* – rede de interligação de computadores que tem uma abrangência de poucos metros, altíssimas taxas de transferências (*Giga bits* ou *Giga bytes/segundo*) com uma ocorrência de erros muito baixa (próxima de zero) [DAN02].

cada nó processa apenas os dados referentes à sua parte de interesse, e então gera a imagem apenas daquela parte [SGI01].

Além disso, os nós dos aglomerados tradicionais são geralmente idênticos. Tal característica tem sido no geral boa, pois existe apenas uma tarefa que é executada em uma base de dados enorme, eventualmente dividida entre os diversos nós. Em Realidade Virtual é diferente, pois as aplicações requerem muitas tarefas variadas para serem realizadas, e em muitas aplicações, os dados são relativamente pequenos quando comparados com os dados dos aglomerados tradicionais. Por isso, os aglomerados gráficos não podem restringir-se a ter apenas nós idênticos, alguns deles podem ser específicos apenas para entradas, outros para saídas e outros para processamento.

O aspecto de tempo real é outra importante diferença entre eles. Os aglomerados gráficos devem realizar as tarefas em tempo real, enquanto que os tradicionais, geralmente, realizam em forma de lotes [SGI01]. A interatividade e a geração de imagens em tempo real das aplicações gráficas requerem que as tarefas sejam executadas respeitando-se alguns limites de tempo. Por exemplo, as aplicações de multiprojeção precisam ser capazes de receber uma entrada, processar os dados, e produzir uma saída (para vários projetores ou monitores), mais de 15 vezes por segundo (preferencialmente 30 vezes, sem estereoscopia), enquanto que as aplicações que utilizam os aglomerados tradicionais não possuem este comprometimento. Isso é conhecido como taxa de atualização de quadros [CAR03, OLS02]. Se a aplicação for estereoscópica, a taxa de atualização de quadros sobe para no mínimo 60 por segundo, sendo preferencialmente 120, pois para cada olho do usuário deve ser gerada uma imagem. Por isto, a sincronização das imagens e a comunicação estão entre os principais desafios a serem superados.

A comunicação é necessária para trocar informações entre os nós. E quando há comunicação, devem existir operações de sincronização, para fornecer controle de seqüência e controle de acesso. O controle de seqüência (também chamado de sincronização condicional ou sincronização de atividades) é utilizado para que se determine uma ordem na qual os processos (ou parte deles) devem ser executados. O controle de acesso é necessário quando há competição entre processos para a

manipulação de algum recurso (deve-se garantir que acessos concorrentes a recursos sejam controlados), para que se mantenha a consistência [SOU96].

3.2.1 A sincronização

O desafio da sincronização em aglomerados gráficos é que esta deve ser feita dentro de uma limitação de tempo. Especificamente, existem três tipos de sincronização que devem ser analisados: a de vídeo (*genlock*); a de dados (*datalock*) e a de conclusão de quadros (*framelock/swaplock*) [SGI01, LIN02, GUI02b, ZUF02a, SCH00].

3.2.1.1 Sincronização dos sinais de vídeo

Todos os computadores gráficos contêm um sinal gerador que conduz a saída para um sistema de visualização, como um monitor ou projetor. Este sinal controla a apresentação (função geralmente controlada por *hardware*) das imagens e é conhecido como *genlock*. Em [ALL02b] é apresentada uma solução via *software* que não precisa nenhum *hardware* específico (placa gráfica com suporte a *genlock*).

A sincronização dos sinais de vídeo assegura que cada *pixel* do sistema de visualização esteja sincronizado. A Figura 15(a) mostra dois monitores nos quais a sincronização não está acontecendo; se estivesse, os feixes de luzes estariam desenhando os pixels nos mesmos lugares. Na Figura 15(b), o *genlock* está sendo utilizado, assim os feixes de luzes estão desenhando os pixels nos mesmos lugares.

(a) – Sem *genlock*

(b) – Com *genlock*

Figura 15 – Sincronização de vídeo – *genlock* [ZUF02a]

O *genlock* está relacionado principalmente com a sincronização para sistemas com estereoscopia ativa. Neste caso, o redesenho das imagens pelos projetores deve ser realizado ao mesmo tempo para o correto funcionamento dos óculos ativos de

estereoscopia (óculos obturadores), pois eles executam batimentos de oclusão da lente, que estão sincronizadas com a imagem de vídeo [CAR03].

3.2.1.2 Sincronização de dados

Existem dois tipos de dados que podem mudar em cada quadro (*datalock*): os de controle, os quais determinam o que desenhar (por exemplo, a direção do ponto de vista do observador em cada nó); e os de mudança do conjunto dos dados (por exemplo, o modelo deve ser atualizado, pois ocorreu uma mudança na textura de um objeto). Sempre que houver uma mudança nos dados, todos os nós devem se sincronizar para que imagens inconsistentes não sejam geradas. Essas mudanças ocorrem devido a estímulos das aplicações em tempo real, por exemplo: eventos de *mouse*, de luvas e de teclados.

Existem três abordagens para a implementação da sincronização de dados em aglomerados gráficos:

- **Distribuição de estímulos:** os estímulos do nó Mestre são transmitidos para os nós Escravos do aglomerado gráfico (arquitetura Com Replicação). Assim, quando o usuário gera um estímulo, por exemplo, movimenta o *mouse* ou pressiona uma tecla, este estímulo é enviado para todos os nós Escravos. Ao recebê-lo, os nós executam as rotinas de tratamento para atualização dos dados e geração das imagens;
- **Cálculo centralizado dos resultados e distribuição:** os estímulos são tratados localmente pelo nó Mestre, sendo os resultados enviados para os nós Escravos, que geram e apresentam as imagens (arquitetura Com Replicação). Esta abordagem é geralmente utilizada quando o problema envolve muitos cálculos que resultam em poucos dados, como em simuladores de vôos. Assim, o meio de comunicação do aglomerado gráfico é utilizado apenas para transmitir os resultados dos cálculos;
- **Cálculo centralizado dos dados gráficos e distribuição:** o nó Mestre trata os estímulos, realiza o particionamento dos dados - utilizando técnicas como a *sort-first*, *sort-middle* ou *sort-last*-, e envia para os nós Escravos gerarem as imagens (arquitetura Sem Replicação).

3.2.1.3 Sincronização da conclusão dos quadros

Como cada nó gera imagens de visões diferentes, então o trabalho de cada um será diferente. Por exemplo, a imagem que está sendo gerada por um nó pode ter centenas de polígonos, enquanto a do outro pode ter milhares. Assim, provavelmente, estes nós finalizarão a tarefa de geração das imagens em tempos diferentes, o que pode acarretar apresentação das imagens de maneira inconsistente. Para isso, há a necessidade da sincronização da conclusão dos quadros (*framelock/swaplock*) para garantir que todos eles sejam apresentados por todos os nós ao mesmo tempo.

Essa sincronização é realizada utilizando-se a técnica denominada acumuladores duplos (*double-buffering*). Ela desenha uma imagem no segundo acumulador (*buffer*) enquanto apresenta uma imagem previamente desenhada no primeiro acumulador. Isso está disponível via *software* (o sinal de sincronização é transmitido via rede conexão de um aglomerado gráfico) ou via *hardware* (o sinal de sincronização é transmitido por um cabo que interconecta as placas gráficas). Em [GUI02b] é apresentada uma solução via *software*, que é um dos resultados dessa pesquisa. Para tal solução, utilizou-se CORBA (*Common Object Request Broker Architecture* - [ORF94, ORF96]) como mecanismo de comunicação entre os nós.

A Figura 16 apresenta a solução via *software* de [GUI02b]. A Figura 16(a) mostra o momento em que o nó Mestre está aguardando a mensagem de *pronto* dos nós (cada nó envia uma mensagem *pronto* para o Mestre, indicando que o desenho está gerado no acumulador). Após receber as mensagens de *pronto*, o nó Mestre apresenta a imagem já gerada em seu acumulador e distribui (*broadcast*) uma mensagem *desenhar* para os outros nós (Figura 16 (b)), indicando que é para apresentarem o conteúdo do acumulador.

Figura 16 – *Framelock* via *software* [GUI02b]

A Figura 17 apresenta a solução via *hardware*, na qual a Figura 17(a) mostra o momento em que o nó Mestre recebeu o sinal que o Nó 1 acabou de gerar a imagem no acumulador. Para isso acontecer, o Nó 2 terminou de gerar a imagem e enviou o sinal para o Nó 1; em seguida o Nó 1 terminou de gerar a imagem e enviou o sinal para o Mestre. A Figura 17(b) mostra o momento em que o nó Mestre apresenta a imagem contida no acumulador e envia o sinal de *desenhar* para o Nó 1, o qual, por sua vez, mostra a imagem contida em seu acumulador e envia outro sinal de *desenhar* para o Nó 2, e assim, sucessivamente. Atualmente, a solução via *hardware* está disponível apenas em poucas placas gráficas, como nas 3DLabs WildCat [USI02] e na Nvidia Quadro FX 3000g [REV04], que são de alto custo.

(a) Mestre recebe o sinal de pronto dos nós (b) Mestre envia o sinal de *desenhar* para os nós
Figura 17 – Framelock via *hardware* [ZUF02a]

3.2.2 A comunicação entre os nós do aglomerado gráfico

Da mesma forma que a diversidade e sofisticação dos computadores sofreram uma considerável expansão em um período bastante curto, também a tecnologia de redes e comunicação propiciou um espantoso aumento no volume e tipo de interações entre os nós dos aglomerados.

Esses nós devem se comunicar, utilizando-se primitivas de comunicação, para que haja a troca de dados e sincronização entre as tarefas em execução, pois não existem espaços de endereçamento de memórias globais. As tarefas devem ter um tamanho (granularidade) e meio de transmissão suficiente para que o tráfego gerado não seja maior do que o processamento e que, conseqüentemente, haja uma queda no desempenho dos processos em execução.

Uma queda no desempenho dos processos afeta diretamente o tempo de reação dos eventos disparados pelos usuários, o que pode gerar falta de interatividade. Por

isso, os requisitos de tempo real aumentam principalmente com o grau de sincronismo das interações dos usuários [GUI02c, FER99].

Para atender os requisitos de tempo real, é necessário garantir uma latência máxima, isto é, o tempo decorrido entre o momento em que o usuário executa uma ação e o momento em que a aplicação apresenta o resultado desta ação. As fontes de latência nos sistemas de Realidade Virtual podem ser divididas da seguinte maneira [WLO95]:

- **Latência do dispositivo de entrada:** tempo total gasto na transmissão dos dados do dispositivo de interação para a aplicação. Esta transmissão pode ser, por exemplo, via comunicação sem fio ou cabo serial;
- **Latência dependente da aplicação:** tempo total gasto pela aplicação para receber um dado e transformá-lo no formato apropriado. Por exemplo, quando a aplicação recebe os dados no formato X3D e os transforma em primitivas gráficas antes de gerar as imagens [SOA04a];
- **Latência de geração de imagens:** tempo total gasto desde o envio dos dados para a placa gráfica, até a apresentação da imagem no dispositivo de visualização;
- **Latência de sincronização:** tempo total gasto na espera da finalização das tarefas que estão sendo executadas em paralelo para poderem, então, coordenar os resultados ou trocarem dados e reinicializar novas tarefas em paralelo; e
- **Latência induzida pela taxa de quadros:** tempo total gasto na percepção da imagem que está sendo apresentada no dispositivo de visualização pelo cérebro do usuário.

No caso de uma aplicação executada em um aglomerado gráfico, a latência de sincronização também inclui a latência de rede, que é o tempo gasto na transmissão dos dados na rede de comunicação. A latência total de uma aplicação de Realidade Virtual baseada em aglomerados é resultado da soma dessas latências.

A literatura apresenta várias técnicas utilizadas na redução da latência em aplicações de Realidade Virtual [CAR97, WLO95, SIN99, HAN01, FER99], como o algoritmo *Dead Reckoning*, que calcula a posição futura de cada objeto a fim de diminuir o número de mensagens de atualização na rede [SIN99]. Esse trabalho tem o foco principal na latência de transmissão de dados e de sincronização, na qual a maioria das soluções atuais baseia-se em MPI (*Message Passing Interface* – [THE03a, INT03]) e PVM (*Parallel Virtual Machine* - [PAR03]). Estas são as bibliotecas de passagem de mensagens mais usadas, entretanto, a interoperabilidade de *hardware* e *software* e os serviços disponíveis em CORBA têm conduzido as novas aplicações a usá-lo [DIE01]. Recentemente, os desenvolvedores têm adotado diversas bibliotecas para a mesma aplicação de Realidade Virtual. A escolha é determinada pelo tipo de dado a ser transmitido, confiabilidade, desempenho e número de destinatários [SIN99].

O uso de bibliotecas de comunicação visa tentar minimizar os problemas decorrentes da necessidade de sincronização, pois as tarefas executadas em paralelo aguardam a finalização mútua para poderem então coordenar os resultados ou trocar dados e reiniciar novas tarefas em paralelo [ALV02]. As principais bibliotecas disponíveis são apresentadas a seguir [BUY99]:

- **PVM:** A idéia principal por trás desta biblioteca é utilizar um conjunto de nós heterogêneos interconectados, como um recurso virtualmente único. Cada nó existente na rede pode ser utilizado como um nó da máquina paralela virtual. Este ambiente é constituído de uma biblioteca de rotinas em C e Fortran, onde o desenvolvedor pode escolher qual protocolo de camada de transporte deseja utilizar, TCP ou UDP, no envio e recebimento de mensagens. Em ambos os casos as mensagens são empacotadas antes do envio e desempacotadas pelo processo receptor. Esses procedimentos geram uma sobrecarga extra, mas resolvem o problema da heterogeneidade de ambientes. A PVM é composta por duas partes. A primeira é um programa chamado *pvm3*, que reside em todos os nós, criando uma máquina paralela virtual. A segunda parte é uma biblioteca de rotinas que são executadas pelas aplicações para a passagem de mensagens, criação de processos, sincronização de tarefas e modificação da máquina virtual [PAR03].

- **MPI:** é um padrão que define um conjunto de rotinas que facilitam a comunicação (troca de dados e sincronização) entre processos em memória, e é composta de ferramentas de análise de desempenho. Este padrão possui diversas implementações (MPICH, LAM, MPL e outras), sendo portátil para a maioria das arquiteturas, mas não interoperável. MPI possui rotinas para programas em linguagem C/C++ e Fortran. Cada processo executa e comunica-se com outras instâncias do programa, possibilitando a execução no mesmo nó ou entre diferentes nós. A comunicação básica consiste em enviar e receber dados de um nó para outro [THE03a].
- **CORBA:** é a especificação de uma arquitetura para ambientes distribuídos e heterogêneos, que tem diversas implementações (Orbacus, Orbix, TAO e outras). As implementações CORBA possibilitam que as aplicações façam solicitações a objetos de uma forma transparente, sendo independente da linguagem, sistema operacional ou considerações de localização. Tal arquitetura define um módulo intermediário entre clientes e servidores, o ORB (*Object Request Broker*). No modelo (Cliente/Servidor) implementado pelo CORBA, os clientes requisitam serviços às implementações de objetos nos Servidores por intermédio de um ORB. O ORB é responsável por todos os mecanismos requeridos para encontrar o Servidor onde o objeto está implementado, pela preparação da implementação do objeto para receber a requisição, e pela execução das requisições. A vantagem do CORBA é que, além da comunicação, ele dispõe de diversos serviços que podem ser importantes em aglomerados, como o serviço de eventos, o de controle de concorrência, replicação, transação e outros. Entretanto, ele tem sido criticado devido ao desempenho na comunicação, mas implementações mais robustas e com características de tempo real têm sido criadas [SCH02a, AHM01].

Em [ESQ01], é apresentada uma comparação entre CORBA (TAO e ORBacus), MPI (LAM e MPICH) e PVM. Os resultados mostraram que MPICH, seguido de PVM, apresenta o melhor desempenho para dados do tamanho de 2KB até 4 KB. Com o aumento do tamanho dos pacotes, a degradação do desempenho é rápida, sendo superada pelo ORBacus e TAO, quando o tamanho ultrapassa 32KB.

Nota-se que CORBA apresenta uma solução competitiva em termos de desempenho de comunicação, quando comparados com MPI e PVM. Além disso, o ORB oferece um melhor gerenciamento de diversas comunicações simultâneas do que MPI e PVM. Outro ponto relevante é que pesquisadores, como [KIW01], têm proposto o uso de CORBA para a construção de sistemas de imagem única (*Single System Image*), e realizado avaliações de escalabilidade, de disponibilidade e de suporte para programação paralela.

O desempenho das aplicações não depende somente do sistema de comunicação, como PVM, MPI ou CORBA, pois diversas aplicações podem estar executando concorrentemente nos nós e utilizando a rede. Além disso, essas aplicações podem estar se comunicando com diferentes nós destinos [SIN99]. Por isto, o gerenciamento do aglomerado é essencial para que se atinja o desempenho requerido. Neste gerenciamento, inclui-se todo o controle sobre os nós, como a inicialização, finalização e monitoramento das aplicações.

3.3 O aglomerado gráfico utilizado

Para a implementação e análise desta pesquisa, este projeto utilizou o aglomerado gráfico denominado Perfect Cluster, desenvolvido no Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo e apresentado na Figura 18. Este aglomerado gráfico é composto por 6 nós Intel-Xeon Xeon 3.06 GHz com 75 GBytes de disco rígido e 2 GByte de memória RAM, 1 MB de memória *cache*, interligados por uma rede Gigabit Ethernet.

O sistema de interconexão é baseado em comutadores Gigabit Ethernet NetStructure 470F Intel^{TM} disponibilizando 8 portas ópticas 1000 Base-SX. Os nós são interconectados através de placas adaptadoras de redes ópticas GigaBit Ethernet. Utilizando-se um comutador adicional, o aglomerado gráfico é conectado ao sistema de armazenamento de massa de dados, no caso, uma estação Dual Pentium III Xeon 833 MHz com um banco de discos de 250 GBytes (responsável pelo armazenamento centralizado de dados gráficos) [SAN01, ZUF01].

Figura 18 - O aglomerado gráfico Perfect Cluster [SAN01, ZUF01]

O sistema de exibição gráfica de cada nó é baseado em placas aceleradoras gráficas Nvidia Quadro FX 3000g para aplicações gráficas de propósito geral, e placas Volume PRO VP500-2X Double Memory para a aceleração do processamento em aplicações de visualização volumétrica.

Para se valer dos recursos fornecidos pelos aglomerados gráficos, como o Perfect Cluster, é necessário utilizar bibliotecas de desenvolvimento que facilitem a criação e execução das aplicações. A próxima seção mostra os principais requisitos que uma biblioteca de desenvolvimento de aplicações de Realidade Virtual baseada em aglomerados gráficos deve apresentar.

3.4 Requisitos das bibliotecas para aglomerados gráficos

Existem fatores desejáveis em uma aplicação, como: facilidade de uso, confiabilidade, desempenho, legibilidade, robustez, compatibilidade e reusabilidade. Esses fatores devem servir como requisitos e preocupações a serem avaliados durante o processo do desenvolvimento de qualquer aplicação.

Para atender esses requisitos e preocupações, são necessários ambientes de desenvolvimento que acompanham a rápida evolução a que está sujeita a área computacional. Estes ambientes devem direcionar o desenvolvimento de aplicações de alta flexibilidade e qualidade, a fim de que possam adequar-se com mais

facilidade ao dinamismo das tecnologias e dos ambientes onde são utilizadas. No desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos, essas preocupações também devem estar presentes, e os principais requisitos que um ambiente de desenvolvimento deve fornecer são [BIE00]:

- **Desempenho:** o ambiente de desenvolvimento deve fornecer recursos que possibilitem alto desempenho das aplicações, caso contrário, pode causar baixa interatividade e falta de imersão. Além disto, o baixo desempenho do sistema pode causar uma baixa taxa de apresentação dos quadros, o que pode gerar problemas conhecidos como *cybersickness* (dores de cabeça, náusea e instabilidade postural) aos usuários [CYB04];
- **Extensibilidade:** como novos recursos surgem constantemente, é importante que o ambiente de desenvolvimento possibilite a extensão das funcionalidades já disponíveis. Viabilizando que, por exemplo, um novo tipo de rastreador seja usado nas aplicações ou que novos recursos de uma nova placa gráfica sejam utilizados;
- **Flexibilidade:** o ambiente deve permitir que os novos nós que forem adicionados no aglomerado gráfico possam ser utilizados pelas aplicações. É desejável a possibilidade de modificação de parâmetros das aplicações em tempo de execução; fácil incorporação ou modificação de funcionalidades. Assim, as últimas tecnologias ou requisitos específicos podem ser incorporados facilmente;
- **Simplicidade:** embora as aplicações de Realidade Virtual sejam naturalmente complexas, pois envolvem conhecimento de diversas tecnologias, o ambiente de desenvolvimento não precisa ser. O ambiente deve possibilitar a criação de aplicações da forma mais fácil possível, permitindo então o desenvolvimento de aplicações sem o conhecimento de todas as particularidades do ambiente;
- **Robustez:** as aplicações de Realidade Virtual são complexas, o que possibilita a ocorrência de algumas falhas. Por isso, é necessário que o ambiente forneça recursos para o tratamento das possíveis ocorrências de falhas, de *hardware* ou de *software*, permitindo então que as aplicações continuem executando mesmo após falhas, como de rede ou de sistema operacional de algum nó;

- **Portabilidade:** esta característica não pode ser considerada propriamente um requisito, porém é recomendável. Ela consiste na possibilidade de o ambiente fornecer recursos para que as aplicações de Realidade Virtual possam ser instaladas e facilmente portadas para diferentes arquiteturas de *hardware* (*Personal Computers* - *PCs*, *Personal Digital Assistant* - *PDA*s) e de *software* - sistemas operacionais (Linux, Irix, Windows), linguagem (C/C++, Java)-. Todavia, outras características podem ser consideradas determinantes de portabilidade, como suporte a dispositivos, a habilidade de abrir e converter uma variedade de formatos de entrada;
- **Processos-leves seguros (*Thread safety*):** suporte de forma simples e automática a processos-leves, que não exige um trabalho ou a utilização de outras soluções em vez de processos-leves;
- **Heterogeneidade:** se possível, é desejável que a mesma aplicação possa ser executada em nós heterogêneos. Assim, pode-se, por exemplo, alguns nós da mesma aplicação podem estar sendo executados no Windows, enquanto outros no Linux.

Visando principalmente o uso comercial, em que os ambientes geralmente não são de alta qualidade como os dos laboratórios de pesquisas, esses requisitos tornam-se características fundamentais para um ambiente de desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos.

3.5 Conclusão

Diversas vantagens têm permitido o crescimento dos aglomerados, dentre elas: a tolerância a falhas, descentralização, partilha de carga de processamento, ganhos de desempenho, economia de recursos e melhor integração com componentes de programas. Entretanto, os aglomerados também possuem algumas desvantagens, como: poucos aplicativos disponíveis; necessidade da utilização de programas para fazer a manipulação das mensagens na rede; caso contrário, pode-se perder mensagens ou o sistema pode ser acessado sem permissão; necessidade de uma rede de interconexão que dê vazão à demanda [SIL01, TAN95]. Por isso, os aglomerados

exigem soluções da área de Engenharia de *Software* mais sofisticadas do que para os sistemas fortemente acoplados.

A necessidade de uma biblioteca específica para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos é evidente. Contudo, estas bibliotecas devem atender os requisitos básicos de *software*, como: usabilidade, suporte a novas tecnologias, desempenho, manutibilidade, tolerância a falhas, portabilidade e reusabilidade. Além disso, devem fornecer recursos específicos para a área de Realidade Virtual, como a sincronização e comunicação de dados em tempo real.

Capítulo 4

---

4. BIBLIOTECA PARA PROGRAMAÇÃO DISTRIBUÍDA

Este capítulo apresenta uma das contribuições dessa pesquisa, a Glass, que é uma biblioteca para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos. Essa biblioteca foi construída a partir do estudo de várias áreas do conhecimento, como transmissão de dados, sistemas distribuídos, dispositivos de visualização, dispositivos de rastreamento e sincronização de dados.

Inicialmente, o capítulo apresenta uma visão geral da Glass, e, em seguida, mostra os seus componentes. Ao final, apresentam-se os testes realizados nos seus componentes.

4.1 Introdução

A Glass foi inicialmente criada no Laboratório de Sistemas Integráveis do departamento de Engenharia de Sistemas Eletrônicos da Universidade de São Paulo, e denominava-se DICElib, tendo como objetivo facilitar o desenvolvimento de aplicações que necessitavam de sincronização entre os nós de um aglomerado gráfico [GNE01]. A versão inicial tinha foco na comunicação de dados e na sincronização entre os nós de um aglomerado gráfico, entretanto não oferecia recursos suficientes, como: mecanismos de comunicação e sincronização eficazes para o problema; facilidade para uso de dispositivos; facilidade de configuração e tolerância a falhas. A Glass adicionou esses novos recursos e aprimorou os existentes, tornando-se assim, um ambiente avançado para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos [GNE03, GUI03a].

4.2 A Glass

A Glass consiste em um conjunto escalável de componentes que podem ser utilizados pelas aplicações. As aplicações são construídas reutilizando-se, conforme a necessidade, os componentes que estão disponíveis, habilitando-se assim a execução dos serviços oferecidos pela biblioteca. Apesar dessa biblioteca ter como objetivo

agilizar e facilitar o processo de desenvolvimento de aplicações de Realidade Virtual, ela pode ser utilizada para desenvolver qualquer aplicação baseada em aglomerado. Além disso, pode ser utilizada para portar aplicações de Realidade Virtual já desenvolvidas para sistemas fortemente acoplados para serem executadas em aglomerados gráficos sem a necessidade de refazer todo o projeto. A documentação, o código fonte e exemplos da Glass estão disponíveis no SourceForge.net, no endereço <http://libglass.sourceforge.net/>, e no site dos Meios Eletrônicos Interativos do Laboratório de Sistemas integráveis da Universidade de São Paulo, no endereço <http://www.lsi.usp.br/interativos/>.

Cabe ressaltar que a biblioteca Glass teve, como uma de suas principais metas, a criação de um ambiente de fácil uso, tanto para o desenvolvimento de novas aplicações quanto para as existentes. Isso se tornou um requisito de destaque, pois a maioria das soluções disponíveis requer um grande número de modificações no código fonte, e, às vezes, na arquitetura da aplicação.

A implementação da Glass foi feita em C/C++. Essa linguagem facilitou o seu desenvolvimento, devido a seus recursos de orientação a objetos. Entre os benefícios destaca-se a estruturação das classes de tal maneira que se torna possível a construção de sistemas complexos e a possibilidade de uso de diferentes tipos de dados (incluindo estruturas e classes) de forma simples.

A Figura 19 apresenta uma visão geral dos seus componentes, cujos detalhes serão apresentados nas seções seguintes. Inicialmente, tem-se o arcabouço Glass, que é composto pelos componentes Instanciação, Protocolo e o Plugins. O Instanciação tem como objetivo inicializar as aplicações conforme a arquitetura interna da Glass, que permite a aplicação ser Cliente ou Servidor. O componente Protocolo encapsula (*wrapper*) bibliotecas, escondendo as diferenças entre os protocolos de comunicação TCP, UDP, MPI, VIA (*Virtual Interface Architecture* [MVI04]) ou outros. As aplicações Glass podem usar diferentes protocolos sem que a biblioteca Glass seja modificada ou recompilada. Este componente possui uma infra-estrutura de empacotamento/desempacotamento de mensagens, que suporta todos os tipos básicos (*integer*, *float*, *string* e outros); além disto, podem-se definir os próprios tipos. Esse empacotador/desempacotador garante a interoperabilidade entre sistemas

operacionais. Assim, pode-se, por exemplo, executar alguns nós da mesma aplicação no Linux e outros, no Windows⁷ e no Mac OS X⁸.

Figura 19 – Arquitetura Glass

O outro componente do arcabouço Glass é o Plugins, que tem como responsabilidade gerenciar todos os *plugins* da Glass. Os *plugins* fornecem serviços de transmissão de Eventos (*plugin* Eventos), de compartilhamento de dados (*plugin* Compartilhamento), de barreiras de sincronização (*plugin* Barreiras) e de associação de funções (*plugin* Alias – Pseudônimo). A Glass não está restrita apenas a esses *plugins*, outros podem ser adicionados conforme a necessidade, sem a modificação de sua estrutura interna. Para isso, o componente Plugins gere a arquitetura de rede interna; os nós que estão conectados; e o envio/recebimento dos dados, de tal forma que o envio seja sempre apenas para os nós que tenham interesse no dado, o que reduz a banda utilizada.

Além do arcabouço Glass, existe o componente Aplicações, que é composto pelas Aplicações Glass e pelas Aplicações de Suporte. As aplicações Glass são aplicações de Realidade Virtual baseadas em aglomerados gráficos. Elas podem ser desenvolvidas em C/C++ ou em Java, estando a integração (*binding*) com Java

⁷ O porte da Glass para o sistema operacional Windows foi realizado pelo Dr Russell Coggrave da Universidade de Loughborough (c.r.coggrave@phasevision.com).

⁸ O porte da Glass para o sistema operacional Mac OS X foi realizado pelo Engenheiro Lucas Peetz Dulley do Laboratório de Sistemas Integráveis da Universidade de São Paulo (dulley@lsi.usp.br).

disponível por meio da utilização de JNI (*Java Native Interface*) [JAV04]. As Aplicações Glass foram testadas nas arquiteturas PC, Silicon e PDA. As outras aplicações Glass, as de Suporte (GTracer, o GEditor e o GVoicer), são ferramentas que auxiliam o desenvolvimento de Aplicações Glass.

As próximas seções apresentam a Arquitetura da Glass de maneira detalhada.

4.2.1 O arcabouço Glass

As aplicações orientadas a objetos são construídas por classes que cooperam entre si, por meio da troca de mensagens, para realizar as tarefas do sistema. Por intermédio do paradigma de Orientação a Objetos, é possível reutilizar uma aplicação ou parte desta, somente redefinindo o comportamento de algumas subclasses. A partir da reutilização de uma aplicação existente, podem-se obter diferentes aplicações, reutilizando-se tanto o código como o projeto geral dessa aplicação. Analisando-se o comportamento esperado das aplicações, é possível detectar que existe uma parte da atuação destas que é comum e que pode ser generalizada. Assim, pode ser criada uma nova classe, superclasse das anteriores, que contenha todo o código comum a ambas, enquanto as originais implementam a parte específica do comportamento abstrato [SOU98].

As classes abstratas representam conceitos genéricos relativos a uma família de objetos relacionados. Cada objeto representa um caso particular da abstração e será representado por uma subclasse concreta, que fornecerá uma variante específica do comportamento abstrato definido na classe abstrata. Dessa maneira, as classes abstratas trabalham como um modelo para as suas subclasses. Assim sendo, um projeto constituído por classes abstratas funciona como um "molde" para aplicações. Um projeto constituído por classes abstratas é denominado um arcabouço de aplicação orientado a objetos ou simplesmente arcabouço [ARA97, LEV99, CAM96, PIZ98].

Os arcabouços fornecem a reutilização de código, que é oferecido por classes e objetos, e também oferecem a reutilização de projeto, liberando o desenvolvedor dos aspectos comuns daquele domínio de aplicação [SOU98]. A reutilização de projeto permite uma reutilização em larga escala, que é uma condição necessária para o

aumento de produtividade na atividade de desenvolvimento de uma aplicação. Além disso, ela é mais importante que a reutilização de código, porque o projeto é mais difícil de criar e recriar do que o código.

O arcabouço Glass consiste em um modelo de classes que representa uma infraestrutura reutilizável e adaptável para as aplicações de Realidade Virtual baseadas em aglomerados gráficos. Este arcabouço visa atender as diversas propriedades dessa técnica de desenvolvimento de *software*. Dentre as propriedades, tem-se a denominada *Extensível*, que é a capacidade de ser ampliado, ou seja, de permitir que os desenvolvedores de aplicações possam facilmente modificar e adicionar funcionalidades. Além de atender esta propriedade, os arcabouços devem ser *Completos*, ou seja, devem fornecer as características desejadas pelos desenvolvedores e uma implementação padrão sempre que possível; ser *Flexíveis*: os arcabouços devem permitir que suas abstrações sejam usadas em diferentes contextos; e ser *Compreensíveis*: as interações entre o arcabouço e o código das aplicações devem ser claras, e o arcabouço bem documentado [SOU98].

A Figura 20 apresenta o Diagrama de classe do arcabouço Glass, cujo ponto chave é a classe abstrata `Glass`. As classes filhas `Cliente` e `Servidor` definem os objetos a serem instanciados pelas aplicações de acordo com a responsabilidade no sistema, ou seja, de ser uma aplicação `Cliente` ou `Servidor`. Outra classe abstrata é a `Protocolo`, utilizada pela classe `Glass`, que define os métodos a serem implementados pelas classes filhas. Assim, é possível ter diversos protocolos no sistema, sendo apenas necessário a implementação dos métodos da classe pai. A classe `Gerenciador de Plugin` é responsável pelo gerenciamento dos *plugins* garantindo que os pacotes enviados pelos nós sejam processados pelos *plugins* correspondentes. A classe abstrata pai `Plugin` define os métodos a serem implementados pelas classes filhas, cuja responsabilidade é fornecer recursos para o desenvolvimento de aplicações, como barreiras de sincronização, transmissão de eventos e criação de tipos de dados. Os *plugins* na Glass são módulos independentes um dos outros.

Figura 20 – Diagrama de classe do arcabouço Glass

No arcabouço Glass nota-se a flexibilidade desse projeto, pois novos *plugins*, protocolos e arquitetura da aplicação podem ser adicionados, conforme a necessidade, estendendo-se, então, a biblioteca com novos serviços [GNE03, GUI03a]. Por exemplo, no componente Instanciação, podem-se adicionar recursos que possibilitem a construção de aplicações com conexão ponto-a-ponto, permitindo, então, no futuro, a integração de aglomerados remotos.

A seguir, são descritos os detalhes da implementação desse arcabouço, expondo todos os serviços fornecidos por ele, as classes envolvidas, e como ele pode ser expandido para englobar novas funcionalidades. Para isso, apresentam-se os detalhes de seus componentes, o Instanciação, o Protocolo e o Plugins, exibidos na Figura 19 e detalhados na Figura 20.

4.2.1.1 Componente Instanciação

Este componente tem como objetivo criar as instâncias das aplicações Glass. Internamente, utiliza-se uma arquitetura Cliente/Servidor, que não influencia no desenvolvimento das aplicações. Quando uma aplicação Glass é executada, inicialmente é executado um Servidor Glass, que, depois de iniciado, permanece na espera de comunicações. Ele gerencia todas as aplicações Clientes (conexão e desconexão) e as mensagens de comunicação (envio e recebimento). Todas as mensagens transmitidas são filtradas pelo Servidor a fim de reduzir as atualizações

redundantes. Tendo o Servidor inicializado, as aplicações Clientes podem ser inicializadas, tomando, então, as iniciativas de comunicação quando desejam algum serviço. Este é o comportamento atual desse componente, entretanto, no futuro, ele será modificado, pois está previsto o suporte de outras arquiteturas além da Cliente/Servidor, como a ponto-a-ponto.

A Figura 21 mostra um exemplo de inicialização de uma aplicação que pode ser Cliente ou Servidor. Inicialmente, incluiu-se o código da Glass (`#include "libglass.h"`). Na função `main`, declarou-se a variável `g` do tipo `Glass` (`Glass *g;`), que tem como objetivo instanciar o nó. Se a aplicação não receber nenhum parâmetro na inicialização, cria-se então uma instância do tipo Servidor (`g = new GlassServer(...);`), caso contrário, do tipo Cliente (`g = new GlassClient(...);`). Por fim, a instância é finalizada (`delete g;`) e o programa terminado (`return 0;`). Nota-se que a sintaxe utilizada é semelhante à das linguagens de programação C/C++, o que facilita o uso dessa biblioteca.

```
#include "libglass.h"
...
int main(int argc, char **argv) {
    Glass      *g;
    if (argc == 1) {
        g = new GlassServer(...);
    } else {
        g = new GlassClient(...);
    }
    ...
    delete g;
    return 0;
}
```

Figura 21 – Inicialização das aplicações Glass

A Figura 22 exemplifica o funcionamento interno da Glass com a arquitetura Sem Replicação (apresentada na seção 2.3). O nó Cliente recebe as interações do usuário, constrói e divide as primitivas gráficas entre os nós do aglomerado gráfico, e a seguir, envia todas estas primitivas para Servidor Glass (Mestre). Quando o Mestre as recebe, envia para os nós Escravos, para que gerem e apresentem as imagens. Essa arquitetura de funcionamento pode variar, por exemplo, no futuro, a divisão das primitivas gráficas poderá ser realizada no Servidor. Desta maneira, o nó Cliente terá apenas como responsabilidade o recebimento das interações dos usuários.

Figura 22 – Aplicação Glass na arquitetura Sem Replicação

A Figura 23 exemplifica o funcionamento interno da Glass com a arquitetura Com Replicação (apresentada na seção 2.3). Neste tipo de aplicação o nó Cliente trata as interações do usuário e gera primitivas de controle. Estas são enviadas para o Servidor Glass, que as envia para todos os nós tratarem e gerarem as respectivas imagens. Portanto, a arquitetura interna da Glass (Cliente/Servidor) não interfere na arquitetura da aplicação (Com Replicação ou a Sem Replicação).

Figura 23 – Aplicação Glass na arquitetura Com Replicação

Assim, podem-se desenvolver aplicações baseadas na arquitetura Com Replicação ou na Sem Replicação, tendo a comunicação e a sincronização entre os nós totalmente transparente para os usuários. Nas seções seguintes, ao se referir ao nó Mestre, trata-se do Servidor Glass, que recebe as interações de um determinado Cliente e que coordena os nós Escravos.

4.2.1.2 Componente Protocolo

A possibilidade de escolha de um protocolo específico entre vários disponíveis pela mesma solução é uma tendência no meio acadêmico e industrial, pois a utilização de um protocolo específico para um problema pode representar um ganho efetivo para uma aplicação [DAN02, SIN99, VOG03]. O componente Protocolo foi projetado para possibilitar o uso de qualquer protocolo de comunicação pelas aplicações Glass.

A Figura 24 apresenta o Diagrama de classes Protocolo, cujo ponto chave é a classe `Protocolo`, que declara métodos, como `Setowner`, `listen` e `accept`, específicos para envio e recebimento de dados. As classes filhas `UDP` e `TCP` definem protocolos de transmissão de dados e implementam métodos abstratos da classe `Protocolo`. Esta define uma generalização de protocolos de transmissão e implementa operações relacionadas com as classes filhas. Assim, o protocolo UDP pode ser utilizado em ambientes onde a taxa de erros de transmissão é baixa; e o protocolo TCP em ambientes em que ocorre grandes retardos na transmissão dos dados, devido a alta taxa de erros. Para a inclusão de um novo protocolo na Glass, basta a criação da classe referente ao novo protocolo (classe filha), que contenha a implementação dos métodos abstratos da classe pai `Protocolo`.

Além do modelo de protocolos implementados na Glass, o Diagrama de classes Protocolo apresenta o serviço de depuração de pacotes. A classe principal desse serviço é a `PacketLog`, que oferece os serviços de armazenamento de todas as mensagens transmitidas entre os nós, independentemente do protocolo utilizado na transmissão. Essa classe é filha da classe `Log`, que define serviços como o `strlevel`, usado na especificação do nível de depuração. Este nível especifica quais dados serão armazenados no arquivo de depuração, como: criação de barreiras, conexão de nós, envio de mensagens e outros.

Figura 24 – Diagrama de classes Protocolo

Atualmente, a escolha do protocolo de transmissão a ser utilizado é estática, ou seja, cada funcionalidade utiliza um protocolo pré-fixado. Entretanto, pode-se estender essa funcionalidade da classe `Protocolo` a fim de realizar escolhas dinâmicas, em tempo de execução, conforme o tipo de dado a ser transmitido, confiabilidade, desempenho, número de destinatários, as características do protocolo (orientado, ou não, à conexão e outras facilidades adicionais) e considerando-se um determinado ambiente de execução.

A maioria das aplicações Glass implementadas utilizam o protocolo de comunicação TCP. Contudo, mesmo com a não-garantia de entrega dos pacotes do UDP, este projeto⁹ e outros o adotaram [HAR01], pois o problema de perda de pacotes em SANs é pequeno (*error free*), e o desempenho do UDP é maior do que do TCP. O desempenho é melhor porque o TCP exige mais memória e mais largura de

⁹ A abordagem para implementação das aplicações Glass, a de *Cálculos centralizado dos resultados e distribuição* (descrito nas seções 3.2.1.2 e 5.1), permite a recuperação da aplicação em tempo de execução caso ocorra perda de mensagens.

banda para garantir que a transmissão seja concluída [ARN97, DAN02], e ainda, a latência de rede desse protocolo é maior do que no UDP.

A incorporação de protocolos na Glass não acarreta problemas de compatibilidade, desde que os métodos abstratos da classe `Protocolo` implementados estejam funcionando de maneira correta. Dessa forma, além do TCP, UDP, MPI, PVM e CORBA, outros podem ser adicionados na solução Glass, por exemplo, o XTP (*Xpress Transport Protocol*), que é um protocolo de alto desempenho projetado para atender, com eficiência, um grande espectro de serviços, como: datagramas em tempo real, *multicasting* e transferência de grande quantidade de informações [DAN02].

Este componente, o Protocolo, mostra como a Glass é flexível na parte de comunicação, podendo incorporar as tecnologias que surgirem no decorrer do tempo. A busca por soluções que aumentam o desempenho na comunicação dos aglomerados tem sido constante, resultando em padrões de indústria, como, o VIA, que transfere a comunicação do nível do núcleo (*kernel level*) para o nível do usuário (*user level*), fornecendo às aplicações um acesso direto ao *hardware* de rede e vice-versa [MVI04]. Esse método implica em evitar os acessos desnecessários à memória e também a intervenção do sistema operacional e de protocolos adicionais, interagindo diretamente com o *hardware* (no caso as placas de rede), aumentando então o desempenho da comunicação [VOG03, PIT03]. Assim, espera-se que protocolos de alto desempenho sejam adicionados na Glass, melhorando então a comunicação das aplicações.

4.2.1.3 Componente Plugins

O arcabouço Glass é composto por diversos *plugins*. Os *plugins* fornecem recursos para o desenvolvimento de aplicações, como a sincronização (*plugin* Barreiras), o compartilhamento de dados (*plugin* Compartilhamento), a transmissão de Eventos (*plugin* Eventos) e a associação de funções (*plugin* Alias). Sem a modificação da estrutura interna da Glass, novos *plugins* podem ser adicionados, além disso, os atualmente disponíveis podem ser estendidos a fim de realizar tarefas específicas. Pode-se, por exemplo, como trabalho futuro, incluir *plugins* para a

realização do particionamento de dados gráficos entre os nós (*sort-first*, *sort-middle* ou *sort-last*).

4.2.1.3.1 *Plugin* Barreiras

Este *plugin* é de extrema importância para as aplicações de Realidade Virtual baseadas em aglomerados gráficos, pois suas funcionalidades garantem a sincronização de valores e/ou das tarefas. O funcionamento da sincronização entre os nós é simples: eles ficam em estado de espera em um determinado ponto do programa até que todos tenham chegado a este ponto, para então continuarem o processamento. Para isso, o nó Mestre aguarda uma mensagem de todos os nós Escravos contendo o aviso de que estão no ponto de sincronização (estado de espera). Após receber a mensagem de todos, o Mestre manda uma para cada nó Escravo avisando para todos continuarem o processamento. Esta mensagem pode conter também os novos valores dos dados compartilhados.

A Figura 25 exemplifica o processo de sincronização. Os nós Escravos avisam o Mestre que chegaram ao ponto de sincronização. Logo após, todos eles ficam esperando que todos os outros cheguem ao mesmo ponto, permanecendo no estado de espera; por fim, quando o Mestre recebe a mensagem de sincronização de todos, este distribui uma mensagem avisando para que todos os nós Escravos continuem o processamento.

Figura 25 – Processo de sincronização

Para melhor utilização dos recursos computacionais disponíveis (mais de um processador no nó), se possível, as aplicações devem ser divididas em diversos processos-leves. Para que ocorra sincronização correta entre as diversas tarefas, as barreiras de sincronização da Glass são numeradas e têm o mesmo escopo das variáveis comuns. Isso permite usar diversas barreiras diferentes simultaneamente, pois ocorre a distinção entre as diversas requisições de sincronização. É um ponto a ser ressaltado, pois a maioria das bibliotecas disponíveis, como a Chromium e DICElib, não consegue tratar as aplicações com processos-leves.

Além disso, a Glass é capaz de recuperar possíveis *deadlocks* (travamentos) devido à finalização de maneira anormal de uma aplicação. Por exemplo, se o Nó 2, da Figura 25 terminar a execução de maneira anormal, o servidor interno da Glass (Mestre), em tempo de execução, detecta que este nó não está mais ativo e o desconsidera. Dessa maneira, o nó Mestre não ficará aguardando a mensagem de sincronização deste nó.

A Figura 26 exemplifica o uso das barreiras de sincronização. Inicialmente, incluiu-se o código da Glass (`#include "libglass.h"`). Na função `main` declarou-se uma barreira denominada `b` (`Barrier b = Barrier(1);`), passando como parâmetro o número desta barreira, e em seguida é realizada a sincronização (`b->sync();`).

```
// inclusão da Glass
#include "libglass.h"

int main () {

    . . .
    // declaração da barreira
    Barrier b = Barrier(1);

    // sincronização da barreira b
    b->sync();

    . . .

    return 0;
}
```

Figura 26 – Barreira de sincronização

4.2.1.3.2 *Plugin* Compartilhamento

O *plugin* Compartilhamento fornece recursos para a troca e compartilhamento de informações entre os nós, para tanto, possibilita a criação de variáveis assíncronas. Essas variáveis podem ser dos tipos comuns de dados (*floats*, inteiros, *string* e outros), ou podem ser criadas a partir de estruturas de dados ou de dados complexos.

Para obter um novo tipo de dado a ser compartilhado o desenvolvedor deve criar dois métodos: um que converte o seu dado para o formato *string* (empacota) e outro que transforma do formato *string* para o novo tipo criado (desempacota). Após isso, na declaração da variável compartilhada deve-se informar o nome dos métodos.

Para resolver o problema de dois processos modificarem o valor da mesma variável, um sistema de hierarquia foi desenvolvido. Cada aplicação Glass é associada a um número durante sua conexão com o servidor Glass (Mestre). Os números são em ordem crescente, e quanto menor seu valor maior sua posição hierárquica. Isto permite desenvolver facilmente um sistema de prioridade para a atualização das variáveis. Desta maneira, o processo superior, hierarquicamente, terá prioridade na atribuição dos valores das variáveis compartilhadas. E, para melhorar o desempenho, somente os nós que declararam a variável, ou seja, os interessados, recebem as mensagens de atualização.

A Figura 27 mostra um exemplo de aplicação Glass com uma variável compartilhada. Inicialmente, incluiu-se a Glass (`#include "libglass.h"`). Na função `main` realizaram-se a declaração e instanciação de uma variável compartilhada denominada `teste` do tipo *float* (`Shared <float> *teste=new Shared <float> ("teste")`). A seguir, o valor “1234” é atribuído a `teste` e é enviado (`teste->sendUpdate()`) para o Mestre. Por fim, o novo valor é obtido do Servidor Glass e impresso (`cerr << "new value = " << teste->getData() << endl;`).

```
#include "libglass.h"
. . .

int main() {
    . . .

    // declaração da variável sharedint
    Shared<float>*teste=new Shared<float>("teste");
    . . .

    // atribuição do valor 1234 para a variável teste
    *teste = 1234;
    . . .

    // envio da atualização da variável
    // compartilhada
    teste->sendUpdate();
    . . .

    // atualização da variável
    teste->getUpdate();
    . . .

    // Imprime a variável
    cerr << "new value  = " << teste->getData() << endl;
    . . .

    return 0;
}
```

Figura 27 – Variável compartilhada

A Figura 28 demonstra o compartilhamento de uma variável do tipo criado pelo desenvolvedor, no caso, o `BigFix`. Para isso, inicialmente, incluiu-se a `Glass` (`#include "libglass.h"`). Em seguida, criou-se o método `packf`, que recebe como parâmetro um valor do tipo `BigFix` e retorna uma *string*; e o método `unpackf`, que recebe uma *string* e o converte em `BigFix`. No método *main*, declarou-se a variável `posX`, informando o tipo, o método de empacotamento e de desempacotamento (`Shared <BigFix, packf, unpackf> *posX; posX = newShared <BigFix, packf, unpackf> ("posX");`). A seguir, ocorrem a atualização do valor local de `posX` (`*posX = appSim->getObserver().getPosition();`) e o envio do seu novo valor para a o Mestre (`posX->sendUpdate();`). Por fim, ocorre a atualização de `posX` (`posX->getUpdate();`).

```
#include "libglass.h"
...
string packf(const BigFix &t) {
    string s;
    // converte t (BigFix) para s (string)
    return s;
}
void unpackf(string &s, BigFix &t) {
    // converte s para t
}
int main(int argc, char* argv[])
{
    // declaração de uma variável compartilhada do tipo BigFix
    Shared<BigFix,packf,unpackf> *posX;
    posX= new Shared<BigFix,packf,unpackf>("posX");
    ...
    // alteração do valor de posX
    *posX= appSim->getObserver().getPosition();
    // atualização de posX
    posX->sendUpdate();
    ...
    posX->getUpdate();
    ...
    return 0;
}
```

Figura 28 – Compartilhamento de variável de tipo criado

4.2.1.3.3 *Plugin Alias*

O *plugin Alias* tem como finalidade permitir a associação de métodos a nós do aglomerado em tempo de execução. Isto permite, por exemplo, que a responsabilidade de um nó de gerar a imagem da tela direita, de um sistema de multiprojeção, seja mudada para a da tela esquerda (em tempo de execução). Desta maneira, antes da inicialização das aplicações, não há a necessidade de realizar o mapeamento de cada nó para uma determinada tela no comutador de vídeo, o que facilita o gerenciamento e inicialização das aplicações.

A Figura 29 apresenta um exemplo de utilização das funcionalidades deste *plugin* em uma CAVERNA^{TM} Digital. Neste exemplo cada nó deve gerar o ponto de vista do observador de acordo com a sua responsabilidade: um nó gera o ponto de vista da direita, outro da esquerda, e assim por diante. Então, inicialmente, incluiu-se a Glass (#include "libglass.h"). A seguir, definiu-se os métodos frente, esquerda, direita, fundo e chao (void esquerda(int a) e outros). Cada um destes métodos calcula o ponto de visão respectivo. Por exemplo, a função frente não modifica a posição ou orientação, enquanto que a função esquerda rotaciona o

observador em 90° em torno do eixo vertical (`glRotated(90, 0, -1, 0);`), e assim sucessivamente. No `main`, ocorrem a declaração e a instanciação de uma variável do tipo *alias* nomeada `assoc`, em seguida, tem-se os métodos que poderão ser associados (`assoc->addAlias((const char *)"frente", frente);` e outros), e por fim, associa-se cada nó a um determinado método (`assoc->associate(firstId+1, " frente ");` e outros).

```
#include "libglass.h"

void frente(int a) // frente
{
}

void esquerda(int a)
{glRotated(90, 0, -1, 0);}

void direita(int a) {
{glRotated(90, 0, 1, 0);}
. . .

int main(int argc, char **argv) {
    // declaração e cria uma instância no novo alias
    Alias<void (*)(int)> *assoc
    assoc = new Alias<void (*)(int)>("frente", frente);
    // adição do método no alias
    assoc->addAlias((const char *)"frente", frente);
    assoc->addAlias((const char *)" esquerda ", esquerda);
    assoc->addAlias((const char *)" direita ", direita);
    // associação do nó ao método
    assoc->associate(firstId+1, " frente ");
    assoc->associate(firstId+2, "esquerda");
    assoc->associate(firstId+3, "direita");
    . . .
    return 0;
}
```

Figura 29 – Associação de métodos

Esta funcionalidade também pode ser utilizada para a implementação de um sistema de migração de tarefas entre os nós ou para implementar um sistema de tolerância a falhas, que permite um nó do aglomerado assumir a tarefa que algum outro nó parou de realizar.

4.2.1.3.4 *Plugin* Eventos

Os dispositivos na Realidade Virtual proporcionam a sensação de imersão, através da conexão do usuário com o ambiente virtual. Para tanto esses dispositivos atuam de duas formas [PIN01]:

- Lendo os movimentos realizados pelo usuário (pelas várias partes de seu corpo);
- Impressionando os sentidos do usuário a fim de simular sensações. A Realidade Virtual atua em geral sobre a visão, a audição e o tato.

A sensação de imersão gerada pelo uso dos dispositivos é obtida por intermédio do tratamento imediato das ações dos usuários [SIN99]. É através do rastreamento que se pode capturar e compreender os movimentos e comandos dos usuários e, com isso, criar ambientes virtuais interativos. Quanto melhor for a precisão, a velocidade no rastreamento e o tratamento das interações, mais real será a simulação [CES01].

O tratamento imediato das ações dos usuários depende da combinação do tempo de processamento das entradas e o número de dispositivos no ambiente. O tratamento pode ser feito por um nó específico ou pode ser implementado de tal forma que seja tratado por diversos nós do aglomerado gráfico, o que dependerá da arquitetura implementada no sistema.

Os nós que manuseiam os dados de entrada dos dispositivos não precisam estar executando as aplicações, contudo eles devem receber as entradas, processar e enviar o resultado para os nós que estão executando a aplicação. Os eventos devem ser recebidos por todos os nós interessados, caso isso não aconteça, incoerências no ambiente podem ocorrer. Também não é preciso que todos os nós responsáveis pelo tratamento dos dados de entrada dos dispositivos recebam todos os tipos de dados de entradas, por exemplo, a aplicação pode ser projetada de tal forma que um nó seja responsável pelos dados de entradas do teclado, enquanto um outro é responsável pelos dados de entrada de um dispositivo de rastreamento (*tracker*).

O tratamento correto dos dados de entrada dos dispositivos é de extrema importância para as aplicações de multiprojeção, pois o ponto de vista de cada imagem deve ser preciso para que não ocorra incoerência na imagem. Quando a aplicação recebe o dado de entrada, esta deve executar a rotina de tratamento apropriada, caso contrário, as imagens não serão geradas corretamente.

O *plugin* Eventos visa fornecer recursos que garantam a distribuição correta dos dados gerados por dispositivos de entrada (PDAs, teclado, *mouse* ou outros) para

os nós responsáveis pelo tratamento. As funcionalidades deste *plugin* são disponibilizadas pelas variáveis do tipo *Event*. As instâncias das aplicações que declaram variáveis desse tipo recebem todos os dados inseridos nela. Este tipo de variável implementa uma fila (FIFO - *first in first out*). Assim, quando há dados na fila, o nó pode retirar o dado da fila e realizar o processamento referente. Como as variáveis *Event* são assíncronas, então, não há necessidade de sincronização dos dados para a inserção ou remoção de um dado na fila, o que possibilita então o uso de dispositivos com baixa capacidade de processamento, quando comparados com computadores de mesa, como os PDAs para a execução de algumas tarefas, como o gerenciamento da navegação nas aplicações.

A Figura 30 exemplifica o funcionamento da utilização das variáveis do tipo *Event*. A Figura 30(a) mostra a sequência em que o usuário interage com um dispositivo, no caso uma luva, que está acoplada ao nó 1, e o envio do dados (*Event*) para os outros nós (linha pontilhada), e no final, os nós que tratam esses dados e enviam (linha contínua) as imagens para o CAVERNA^{TM} Digital. A Figura 30(b) mostra a sequência em que o nó 2 gera um dado (*Event*) de retorno e este é enviado (linha pontilhada) para o nó n, onde está acoplado um dispositivo de *force-feedback*. Essa figura mostra também que os eventos podem ser enviados para nós específicos do aglomerado gráfico.

Figura 30 – Entrada/Saída da variável *Event*

A Figura 31 demonstra o uso de uma variável do tipo *Event*. Inicialmente, incluiu-se a Glass (`#include "libglass.h"`). Na função `main` realizaram-se a

declaração e instanciação de uma variável *Event* denominada *evento* do tipo inteira (`Event<int> *evento = new Event<int>("integer");`). A seguir, gerou-se um evento com o valor “1234” (`evento->enqueueEvent(1234);`). Por fim, o programa testa se algum evento chegou (`if ( !evento->isEmpty())`); se sim, então imprime o valor do evento (`cerr << evento->getEvent() << endl;`). Assim, todas as instâncias que declararam a variável *evento* irão receber o valor enviado.

```
#include "libglass.h"
...
int main(int argc, char **argv) {
    ...
    Event<int> *evento = new Event<int>("integer");
    ...
    evento->enqueueEvent(1234);
    ...
    if ( !evento->isEmpty())
        cerr << evento->getEvent() << endl;
    ...
    return 0;
}
```

Figura 31 – Variável *Event*

Apesar de esse *plugin* ter sido criado com o propósito de tratar dos dados de dispositivos, ele pode ser utilizado para o envio de qualquer tipo de dado. Por exemplo, pode ser utilizado para avisar que todos os nós devem finalizar a execução da aplicação corrente ou para a execução de uma determinada função quando uma interação ocorrer.

A seção seguinte apresenta as aplicações que podem ser construídas utilizando-se a Glass e algumas aplicações que auxiliam o desenvolvimento de tais aplicações.

4.2.2 Aplicações

A Glass possui dois grupos de aplicações. O primeiro são as Aplicações Glass criadas pelos desenvolvedores e os programas exemplos presentes na biblioteca. O segundo são as Aplicações de Suporte que auxiliam o desenvolvimento de aplicações. Neste grupo, tem-se o GTracer, que possibilita a visualização gráfica de todas as mensagens transmitidas entre os nós; o GEditor, que permite a geração automática de aplicações para PDAs; e o GVoicer, que auxilia o desenvolvimento automático do controle via voz das aplicações. As próximas seções apresentam estes grupos em detalhes.

4.2.2.1 Aplicações Glass

A Glass oferece abstração dos dados e independência de outras bibliotecas. Deste modo, os tipos comuns de dados e outros podem ser criados e manipulados; além disso, as soluções já disponíveis podem ser reutilizadas. A reutilização de outras bibliotecas é uma opção dos desenvolvedores, que podem, por exemplo, escolher a biblioteca gráfica OpenGL [OPE03a] ou a OpenGL Performer [OPE03b] para a implementação das aplicações, e podem utilizar a biblioteca de som OpenAL [OPE03c]. Ainda, as aplicações podem utilizar diversos dispositivos de multiprojeção, como: CAVERNAs^{TM} Digitais, capacetes de visualização e monitores; e vários dispositivos de interação, como: *mouse*, rastreadores e dispositivos de sensação de toque e podem ser executadas em Linux, MAC OS X, Irix ou Windows.

Esta pesquisa é voltada principalmente para aglomerados gráficos, que podem ter nós heterogêneos ou não, e que estão ligados por uma rede local (SANs), devido à alta latência de transmissão das redes de longa distância. Mas, como projeto futuro, preveu-se estender a Glass para aplicações heterogêneas remotas (interligadas por redes de longa distância). Para tanto, anteviu-se na arquitetura a possibilidade de uma conexão ponto-a-ponto entre os nós Mestres para manter a sincronia e transmissão de dados. Mais ainda, previu-se a possibilidade da troca de dados heterogêneos entre aglomerados, permitindo então a utilização do padrão XML (*Extensible Markup Language*) para a transferência de dados. Este padrão pode ser incorporado nas próprias variáveis do tipo Compartilhada (*Shared*) ou Evento (*Event*), sendo apenas necessária a construção de métodos de empacotamento e desempacotamento dos dados das variáveis. Além de facilitar a integração de dados de diferentes fontes e incompatíveis, XML permite a compressão, e é suportado pela maioria dos produtos disponíveis e possibilita que diferentes nós visualizem os mesmos dados de forma diferente [ARC02]. Assim, um dado pode ser visualizado de uma maneira no PDA e de outra maneira em um monitor de um computador de mesa.

A proposta de utilização de XML como padrão de integração de aglomerados gráficos remotos é um dos resultados desta pesquisa e foi apresentada no congresso

denominado Extensible Modeling and Simulation Framework Technical Challenges Workshop [ZUF02b], com o objetivo de direcionar as novas pesquisas da área.

A Figura 32 exemplifica a integração de dois aglomerados remotos, em que os nós podem ser heterogêneos. Nesta arquitetura, os nós Clientes recebem as interações dos usuários; os nós Escravos executam as tarefas vindas do Mestre; e os nós Mestres além de receber os dados dos nós Clientes e enviar os comandos para os nós Escravos, comunicam-se entre si, utilizando o padrão XML, para manter a sincronia entre os aglomerados gráficos.

Figura 32 – Utilização do padrão XML para integração de aglomerados remotos

O padrão XML não está restrito apenas à integração de aglomerados, pode também ser utilizado como base para a apresentação de documentos 3D via rede, como proposto pelo consórcio Web 3D [WEB04]. Por isso, diversas pesquisas têm utilizado tal padrão para representar os mundos virtuais, a fim simplificar a complexidade da escrita e compilação do código [GRI02, SOA04a]. Sendo, então, fundamental a necessidade de compatibilidade deste padrão com o uso de aglomerados gráficos.

Além disso, a possibilidade de uso do padrão XML é importante para que a Glass suporte, no futuro, aplicações de Computação em Grade (*Grid Computing*). A Computação em Grade visa permitir o compartilhamento, seleção, agregação de

recursos independentes e geograficamente distribuídos, onde partes com interesses comuns, como centros de pesquisa nas áreas da saúde, biologia, astronomia, física, empresas e universidades compartilham seus recursos através da interligação de seus laboratórios e centros de pesquisa [FOS04]. Atualmente, as pesquisas na área de Computação em Grade estão buscando resolver, principalmente, os três seguintes aspectos: heterogeneidade (multiplicidade de recursos, grandes distâncias geográficas); escalabilidade (capacidade de expansão e tolerância a falhas); e adaptabilidade (variação na disponibilidade de recursos e serviços). No caso das aplicações de Realidade Virtual, devem-se também adicionar aspectos como o suporte a interações dos usuários em tempo real [MAR04]. Como a Computação em Grade vai além dos recursos distribuídos localmente (aglomerados), esta se torna uma extensão dos aglomerados.

A possibilidade da união de aglomerados gráficos para a execução da mesma aplicação de Realidade Virtual, sem restrições de tempo e disposição geográfica, motiva a pesquisa da Computação em Grade. A Computação em Grade, junto com a Realidade Virtual, representa um novo e desafiador horizonte para os projetistas de linguagens de programação, compiladores e desenvolvedores de sistemas. A junção dessas áreas requer grandes investimentos financeiros e de pesquisas. Diversas dificuldades de implantação devem ser superadas para a realização de uma infraestrutura de *middleware* capaz de suportar completamente o proposto. Entretanto, a realização deste objetivo é essencial para o crescimento da ciência e da engenharia [FOS04].

4.2.2.2 Aplicações de Suporte

A seguir são apresentadas as ferramentas que auxiliam o desenvolvimento de aplicações Glass.

4.2.2.2.1 GTracer

O componente Protocolo, que faz parte do arcabouço Glass, permite o armazenamento de todos os pacotes que são enviados/recebidos pelos nós. Para facilitar a visualização desses pacotes, criou-se uma aplicação em Java, o GTracer. Esta aplicação auxilia os desenvolvedores, em tempo de execução ou não, a

verificarem se as transmissões de pacotes estão sendo realizadas corretamente, podendo, então, descobrir erros de implementação, principalmente durante o processo de desenvolvimento de novos *plugins*.

Se a opção de depuração for habilitada durante a compilação, cada aplicação Glass cria um arquivo com todos os dados de transmissão/recebimento. Se não houver interesse na depuração, então esta deve estar desabilitada, pois ela consome tempo de gravação em disco, o que afeta o desempenho da aplicação. A Figura 33 apresenta a interface do GTracer, que permite o desenvolvedor escolher o arquivo a ser depurado. O nome do arquivo é formado pela cadeia de caracter “tcp” seguida do número do processo, e com a extensão “log”. Por exemplo: “tcp2802.log”

Figura 33 – Escolha da aplicação no GTracer

Após a escolha da aplicação a ser depurada, é inicializada a interface apresentada na Figura 34. Esta interface é composta por botões que permitem a execução, finalização e reinicialização do processo de depuração em tempo execução, ou a visualização das mensagens depuradas no formato texto. Como existem diferentes tipos de mensagens, por exemplo, de sincronização ou de envio de atualização de variáveis, esta ferramenta permite aplicar filtros para a visualização tipos específicos de mensagens.

Figura 34 – Depuração no GTracer

4.2.2.2 GEditor

Tradicionalmente, as aplicações de Realidade Virtual utilizam dispositivos de interface especializados de alto custo, como luvas, *3-D joysticks* e capacetes de visualização, que, muitas vezes, limitam a liberdade dos usuários e são de difícil manipulação por usuários leigos.

Nos últimos anos, os computadores PDAs têm sido utilizados largamente. Como consequência, um grande número de pessoas estão familiarizadas com ele e o preço tem diminuído. Além disso, a sua qualidade tem melhorado, como a velocidade dos processadores e a capacidade de armazenamento, o que tem possibilitado utilizá-los para tarefas complexas, como visualização de imagens gráficas, execução de vídeos e aplicações de áudio. Outro ponto importante é que eles são compatíveis com cartões de rede sem fio IEEE 802.11 e com outras interfaces, como *Bluetooth*, Serial e Infra-Vermelho [THE03b].

A facilidade da utilização de PDAs em ambientes de multiprojeção já foi estudada em diversos projetos, como Tweek [HAR02], JAIVE [HIL00] e pelo Virtual Heritage [BEN02]. Para facilitar o uso de PDAs nas aplicações Glass, desenvolveu-se o GEditor, que, além de facilitar o uso desses dispositivos nas aplicações, gera, de maneira simples, a interface gráfica que é executada no PDA. As

interfaces geradas são simples e intuitivas, podendo ser utilizadas por usuários que não possuam treinamento para interagir com aplicações de Realidade Virtual. As interações ocorridas nas aplicações dos PDAs são transmitidas para um nó do aglomerado gráfico; assim, elas são tratadas, e as imagens do ambiente são geradas de maneira sincronizada [GUI03b, GUI04].

O GEditor é uma ferramenta para a fácil geração da interface gráfica para PDAs, que não exige conhecimento de programação dos desenvolvedores. Ela foi escrita em Java e permite definir os *hot spots* (retângulos e elipses) de uma imagem (gif, jpeg) de maneira gráfica, associando-os a eventos. Um *hot spot* é uma área de uma imagem que, ao ser pressionada, dispara a transmissão de um evento associado para um nó específico do aglomerado gráfico, o qual é responsável pelo tratamento dos eventos dos PDAs. A interface gráfica visível no PDA é a imagem, sem os *hot spots*, que foi criada com o objetivo de representar o sistema de controle da aplicação. Tendo os *hot spots* de uma imagem, essa ferramenta gera um código de configuração para o aplicativo do PDA, e o trecho de código para o tratamento dos eventos recebidos nos nós responsáveis por seu processamento. A mesma aplicação pode ter várias interfaces (*skins*), cada interface com os seus próprios *hot spots*. As interfaces da aplicação podem ser mudadas dinamicamente em tempo de execução.

A Figura 35 mostra a interface do GEditor, que é constituída por botões, menus e por uma área de desenho, que mostra os *hot spots*, no qual cada um está associado a um evento. Os botões têm como objetivo permitir selecionar, criar e apagar *hot spots*. Ainda, eles acionam a leitura da imagem e a geração do código que será executado no PDA, bem como a geração do trecho de código que será executado no nó que tratará os eventos vindos do PDA. A área da imagem apresentada é 320 x 240, pois está configurado para um iPAQ H3970; entretanto, pode ser alterado de acordo com o PDA utilizado. A interface apresentada nesta figura é referente ao programa de navegação na Galáxia, o Celestia [CEL04], que foi portado para a CAVERNA^{TM} Digital utilizando-se a Glass. Para esta aplicação, foram criadas diversas interfaces, como a de visualização de planetas, a de configuração e a de visualização de estrelas. Em cada uma delas, adicionaram-se os seus próprios *hot spots*, por exemplo, na interface planetas (*Planets*) adicionou-se a cada planeta um *hot spot*, que, quando

disparado, envia ao aglomerado gráfico o comando de navegação referente ao planeta escolhido (visualização do planeta escolhido).

Figura 35 – GEditor

Cada *hot spot* é associado a um evento, o qual possui um número (gerado automaticamente), nome, cor e tipo. Os valores do nome, cor e tipo podem ser modificados pelos desenvolvedores durante o processo de criação da interface. Um evento pode ter mais do que um *hot spot* associado, além disso, cada interface pode ter um conjunto diferente de *hot spots*. Existem quatro tipos de eventos:

- **Eventos simples (*Simple*):** quando disparado, o número do evento associado é enviado para o nó do aglomerado gráfico responsável pelo recebimento e processamento do evento;
- **Eventos de saída (*Exit*):** quando disparado, a aplicação do PDA será encerrada, e os outros nós do aglomerado gráfico serão notificados para encerrar a aplicação que está sendo executada;
- **Eventos de mudança de interface (*Skin*):** quando disparado, a interface corrente será trocada pela associada ao evento. Este tipo de evento não é enviado para os nós do aglomerado gráfico, pois ele é tratado localmente; e
- **Eventos de posição (*Position*):** quando disparado, a coordenada x,y da tela que recebeu o evento é enviada para o nó do aglomerado gráfico responsável pelo recebimento e processamento do evento.

A Figura 36 apresenta a janela do GEditor que permite o desenvolvedor escolher o tipo do *hot spot*. Além da escolha do tipo, esta janela permite apagar um *hot spot*, modificar suas características (cor e nome), adicioná-lo e apagá-lo de todas as interfaces, e ainda, possibilita, criar, apagar e nomear uma interface.

Figura 36 – Tipos de *hot spot*

Após a criação da interface para o PDA, com os respectivos *hot spots*, o desenvolvedor deve salvar o projeto. O arquivo salvo é utilizado no PDA para a inicialização e execução da aplicação. Para isso, desenvolveu-se um leitor em Java deste arquivo, que apresenta a interface da aplicação no PDA sem os *hot spots*. Além disso, esse leitor obtém as interações do usuário na interface e envia os eventos associados aos *hot spots* para o nó do aglomerado gráfico responsável pelo recebimento dos eventos. A Figura 37 mostra exemplos de aplicações sendo executadas em PDAs. Nota-se que os *hot spots* não estão visíveis.

Figura 37 – Aplicações com execução no PDA

A Figura 38 mostra as camadas de tecnologias envolvidas na construção das aplicações para PDAs. A aplicação (camada superior - Leitor) é em Java, entretanto poderia ter sido escrita em C/C++, o que exigiria a utilização de bibliotecas de construção de interfaces gráficas, como a GLUT [GLU04] ou a QT [TRO04]. Na camada intermediária, utilizou-se o pacote JNI [JAV04], da linguagem Java, para realizar a integração de a aplicação do PDA (Java) com a Glass (C/C++). A camada inferior mostra as vantagens da aplicação (Leitor) ter sido escrito em Java, que é a possibilidade de ser executado em diversos sistemas operacionais (Windows, Linux, Windows CE, IRIX) e em arquiteturas de computadores (PC, PDAs, Silicon), sem modificações ou compilações.

Figura 38 – Camadas de *software*

Para o tratamento das interações vindas dos PDAs, há a necessidade da construção de uma rotina específica, que deve ser executada em um nó do aglomerado gráfico. Para facilitar o desenvolvimento dela, o GEditor gera, de maneira automática, um código base na linguagem C.

A Figura 39 mostra um exemplo de código base de tratamento de eventos gerado. O código refere-se a uma aplicação de navegação no Celestia (Figura 35). Quando o usuário pressionar a área do *hot spot* referente ao planeta Mercúrio (Mercury), por exemplo, esta rotina irá receber o valor associado e executará (switch) a rotina apropriada.

```
enum evento = {Mercury=1, Pluto=2, Venus=3 . . . };
switch (evento) {
    case Mercury:
        // rotina de visualização do planeta Mercúrio
        break;
    case Pluto:
        // rotina de visualização do planeta Pluto
        break;
    case Venus:
        // rotina de visualização do planeta Venus
        break;
    . . .
}
```

Figura 39 – Código base para o tratamento de eventos

A Figura 40 mostra a arquitetura do ambiente de execução de uma aplicação que utiliza PDAs como dispositivo de interação. No centro da figura está o dispositivo de multiprojeção (CAVERNA^{TM} Digital), que é controlado pelos nós do aglomerado. O PDA é considerado como um outro nó do aglomerado, portanto oferece comunicação bidirecional. A mesma aplicação pode executar em todos o PDAs. Como os eventos são assíncronos, os outros nós não dependem do desempenho do PDA. Qualquer aplicação, tanto a que está sendo executada no PDA como a dos outros nós, pode ser inicializada ou finalizada em qualquer momento, que as outras aplicações continuam a execução. Dessa maneira, é possível a variação de quantidade de nós em tempo de execução, o que permite, por exemplo, novos usuários serem incluídos na simulação em tempo de execução.

Figura 40 – O PDA é um nó do aglomerado

O tratamento remoto dos eventos do PDA pelos outros nós do aglomerado gráfico facilita o desenvolvimento da interface, exige pouca capacidade de processamento deste dispositivo. Além disso, torna fáceis mudanças nas interfaces, pois não são necessárias mudanças ou recompilação da aplicação, apenas mudanças no arquivo de configuração (sem conhecimento de programação). Como trabalho futuro, planeja-se incluir nas interfaces o suporte para dados multimídia, como animações e vídeos.

4.2.2.2.3 GVoicer

Com o objetivo de agregar mais um mecanismo de acesso e controle às aplicações Glass, desenvolveu-se, utilizando-se a linguagem Java, o GVoicer, que possibilita a criação do controle das aplicações via comandos de voz, onde cada comando é associado a um evento; e um leitor que recebe e trata os comandos de voz e que transmite os eventos associados para um nó do aglomerado gráfico, o qual receberá e executará a rotina apropriada. Como eles foram desenvolvidos em Java, então são suportados por diferentes sistemas operacionais (Windows, Linux, Windows CE, IRIX) e arquiteturas de computadores (PC, PDAs, Silicon). Entretanto, o computador que executa o leitor deve possuir um microfone.

A Figura 41 detalha o funcionamento de uma aplicação Glass controlada por comandos de voz. Inicialmente, os comandos são obtidos em um microfone acoplado a um computador. Em seguida, o leitor faz o reconhecimento do comando, utilizando a biblioteca Sphinx-4 [SPH04], e gera o evento associado ao comando para a Glass, utilizando JNI. No final, o evento é obtido em um nó do aglomerado gráfico e tratado.

Figura 41 – Arquitetura do GVoicer

Assim como é necessário o tratamento das interações geradas pelos PDAs, há também a necessidade da construção de uma rotina de tratamento dos comandos via voz. Para tanto, o desenvolvedor aciona o processo de geração do trecho de código do auxílio no GVoicer, que serve de base para a criação da rotina que receberá os comandos de voz nos nós do aglomerado gráfico. O código é semelhante ao apresentado na Figura 39, entretanto refere-se aos comandos de voz. Além dele, é gerada a gramática que será utilizada pelo Sphinx-4. Ela especifica os possíveis comandos a serem reconhecidos e é gerada de forma a eliminar possíveis ambigüidades. A Figura 42 mostra um exemplo de gramática gerada.

```
#JSGF V1.0;

grammar sentence;

public < sentence> = (left | right | up | down | exit );
```

Figura 42 – Gramática

A Figura 43 mostra a interface do GVoicer. Esta interface é constituída por botões, menus e por uma tabela de edição de comandos. Os menus têm como finalidade criar novos projetos, abrir os existentes, salvar projetos ou finalizar o GVoicer. Os botões têm como objetivo permitir adicionar e remover os comandos que os usuários poderão emitir, e, além disso, possibilitam a geração do trecho de código que auxiliará o desenvolvimento da rotina de tratamentos dos comandos enviados para o aglomerado gráfico e a geração da gramática a ser utilizada pelo sistema reconhecedor dos comandos de voz (Sphinx-4).

Figura 43 – GVoicer

A interface apresentada na Figura 43 é referente a uma aplicação que possui os seguintes comandos de navegação associados a um número de evento: *left* (esquerda) associado a 1; *right* (direita) associado a 2; *up* (para cima) associado a 3; *down* (para baixo) associado a 4; e *exit* (sair) associado a 5. Assim, quando algum usuário falar algum destes comandos de voz no microfone, o número do evento associado é enviado para o nó do aglomerado gráfico responsável pelo tratamento das interações de voz.

4.3 Testes

Foram feitos diversos testes com os *plugins* da Glass a fim de avaliar o funcionamento e desempenho em diferentes situações de carga, isto é, para um número variável de nós e de tamanho de mensagens. O principal objetivo foi o de obter valores quantitativos que caracterizassem e confirmassem a viabilidade desta biblioteca para o desenvolvimento de aplicações de Realidade Virtual, principalmente as de multiprojeção.

Todos os testes foram executados no aglomerado gráfico Perfect Cluster, descrito na seção 3.3. O sistema operacional utilizado foi o Linux Red Hat 9.0, com a arquitetura interna Cliente/Servidor e o protocolo de transmissão TCP/IP. Os *plugins* testados foram os seguintes: Eventos, Barreiras e o de Compartilhamento. O Alias, não foi testado, pois utiliza pouca banda de rede e não demanda respostas rápidas.

4.3.1 Barreiras

Nos testes do *plugin* Barreiras, apresentado na Figura 44, mediu-se o tempo de sincronização de uma variável barreira (*Barrier*) em cada nó 1000 vezes, variando o número de nós de 1 até 5. Nota-se que mesmo utilizando-se um protocolo *unicast*, cujo desempenho é inferior ao de um protocolo *broadcast*, como o UDP, ou *multicast*, como o XTP, os resultados obtidos mostram que a sincronização é suficiente para as aplicações que requerem sincronizações constantes. Com 5 nós, foi possível fazer 2400 sincronizações por segundo – o requerido pelas aplicações varia de 30 a 120 sincronizações por segundo.

Figura 44 – Barreiras (1000 sincronizações)

4.3.2 Compartilhamento

Nos testes do *plugin* Compartilhamento, mediu-se o tempo de sincronização de uma variável compartilhada (*Shared*) em cada nó 1000 vezes, variando o número de nós de 1 até 5, e o tamanho das mensagens de 1 kb até 4096 kb. Para melhorar a visualização, os resultados são apresentados na Figura 45 (1 kb até 64 kb) e na Figura 46 (128 kb até 4096 kb). Assim como ocorreu no teste do *plugin* Barreiras, no *plugin* Compartilhamento, o crescimento obtido foi linear, devido às propriedades do protocolo TCP/IP.

Figura 45 – Compartilhamento (1000 sincronizações) – de 1 byte até 64 kb

Figura 46 – Compartilhamento (1000 sincronizações) – de 128 kb a 4096 kb

4.3.3 Eventos

No teste do *plugin* Eventos, mediu-se o tempo de envio de 1000 mensagens do tipo Evento (*Event*), variando o número de nós de 1 até 5 e o tamanho das mensagens de 1 kb até 4096 kb. Os eventos foram transmitidos por um nó mestre e recebidos por até 4 nós escravos. Para melhorar a visualização, os resultados são apresentados na Figura 47 (1 kb até 64 kb) e na Figura 46 (128 kb até 4096 kb). Assim como ocorreu no teste do *plugin* Barreiras e no Compartilhamento, no Eventos o crescimento obtido foi linear, devido às propriedades do protocolo TCP/IP. Isso mostra que o gerenciamento da transmissão de dados na Glass tem pouca interferência no processo de envio, sendo o limite a capacidade de transmissão da rede.

Figura 47 – Eventos (1000 sincronizações) – de 1 byte até 64 kb

Figura 48 – Eventos (1000 sincronizações) – de 128 kb a 4096 kb

4.4 Conclusão

Este capítulo apresentou a Glass, que é uma biblioteca para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos. Esta biblioteca é composta por dois componentes: o arcabouço Glass que fornece as funcionalidades básicas para as aplicações, como a comunicação e sincronização; e Aplicações Glass, que são as aplicações de Realidade Virtual e Aplicações de Suporte, que auxiliam o desenvolvimento de aplicações ou de funcionalidades.

O arcabouço Glass foi obtido a partir da idéia de um projeto com pontos adaptáveis (*plugins*). Utilizando esta idéia, diversas funcionalidades foram criadas obtendo-se então, a flexibilidade necessária para o desenvolvimento de aplicações. As Aplicações de Suporte foram construídas a partir da necessidade de se depurar as mensagens enviadas entre os nós do aglomerados, e a partir da possibilidade de utilização de PDAs e de comandos de voz como mecanismo de interação com as aplicações. Assim, utilizando-se as funcionalidades do arcabouço Glass e das ferramentas de Suporte, as aplicações de Realidade Virtual são criadas independentes

da arquitetura utilizada (Com Replicação ou a Sem Replicação), tendo a comunicação e sincronização entre os nós totalmente transparentes para os usuários.

Dessa forma, desenvolveu-se a Glass, na qual os desenvolvedores contam com fatores como: a facilidade de uso, a confiabilidade, a legibilidade, a robustez, a compatibilidade e a reusabilidade; e na qual as aplicações contam com as vantagens de um sistema fracamente acoplado e de baixo custo, que são os aglomerados gráficos.

Capítulo 5

---

5. DESENVOLVIMENTO DE APLICAÇÕES

O capítulo anterior apresentou a biblioteca Glass e as suas características funcionais, que permitem o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos.

Este capítulo mostra, inicialmente, as estratégias de desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos. Nele é detalhada outra contribuição dessa pesquisa, que é a estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída*, específica para aplicações de Realidade Virtual baseadas em aglomerados gráficos. Ao final, exemplifica algumas aplicações que foram desenvolvidas utilizando-se a Glass.

5.1 Estratégias de desenvolvimento

Para permitir uma definição clara das estratégias de desenvolvimento, várias aplicações foram desenvolvidas no âmbito dessa tese. Praticamente existem duas estratégias para o desenvolvimento de aplicações de Realidade Virtual baseadas em aglomerados gráficos. A primeira é a de *Desenvolvimento Automático*, que isenta o programador da tarefa de criar a aplicação. Propõe a utilização de um mecanismo, como um compilador, por exemplo, que, a partir de um código fonte de uma aplicação centralizada a transforma em uma aplicação de Realidade Virtual baseada em aglomerados gráficos. O ambiente Chromium [HUM02] é um exemplo de implementação dessa estratégia. Ele transforma comandos OpenGL de uma aplicação centralizada em fluxos de comandos que são enviados para os nós dos aglomerados gráficos gerarem as imagens [HUM01].

A Figura 49 apresenta os passos da estratégia de *Desenvolvimento Automático*. Inicialmente, o Analista de Sistemas faz a modelagem do sistema a partir de seus requisitos, utilizando técnicas de modelagem como a UML (*Unified Modeling Language*) [FOW97]. No segundo passo, ele faz a implementação da aplicação utilizando as bibliotecas gráficas e de som. Por fim, no terceiro passo, utiliza uma

biblioteca para programação distribuída capaz de receber os dados de uma aplicação e distribuir entre os nós do aglomerado gráfico ou um compilador capaz de transformar uma aplicação centralizada em uma aplicação de Realidade Virtual baseada em aglomerado gráfico.

Figura 49 – Desenvolvimento Automático

Esta solução é simples para o desenvolvedor, pois não há necessidade de prever, durante a modelagem e implementação, os problemas referentes a ser um sistema distribuído; entretanto, a construção de uma biblioteca de *Desenvolvimento Automático* eficiente é difícil. Além disso, essa estratégia limita as aplicações aos recursos fornecidos pela *Biblioteca de Desenvolvimento Automático*.

Outra estratégia é a de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída*. Tal abordagem tem gerado resultados melhores do que a anterior. A idéia básica é adição de um código que possibilita a troca de informações entre os nós do aglomerado gráfico. A desvantagem dessa proposta em relação a anterior é que durante a modelagem e implementação, o código de comunicação, sincronização e compartilhamento dos dados entre os nós do aglomerado gráfico deve ser incluído. Exemplos de bibliotecas que podem utilizar esta estratégia incluem a DICElib, Glass, Net Juggler e OpenSG.

A Figura 50 apresenta os passos da estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída*. Inicialmente, o Analista de Sistemas faz a modelagem do sistema a partir de seus requisitos, utilizando técnicas de modelagem como a UML. A modelagem é realizada prevendo a comunicação, sincronização e compartilhamento de dados entre os nós do aglomerado gráfico. No segundo passo, o Analista de Sistemas faz a implementação da aplicação utilizando as bibliotecas gráficas, de som e a distribuída.

Figura 50 – Desenvolvimento Utilizando Biblioteca para Programação Distribuída

Nesta estratégia, algumas decisões de projeto devem ser tomadas desde o início. No primeiro passo, no qual o sistema é especificado, há a necessidade de determinar qual a abordagem de sincronização de dados será utilizada, que pode ser: a de *Distribuição de estímulos* ou a de *Cálculo centralizado dos resultados e distribuição*. A primeira costuma ser muito simples, consistindo na adição de comandos que distribuam os eventos de Entrada/Saída (teclado, *mouse*, etc) para os outros nós. Apesar de simples, essa alternativa pode trazer alguns problemas, já que não existe nenhuma sincronia de dados. Isso significa que, se por algum motivo um dos nós sair de sincronia (perdendo um evento, processando-o incorretamente ou fora de ordem), o aplicativo não se recuperará.

A segunda alternativa é uma solução mais robusta, e em geral pouco mais difícil de se implementar. O passo primordial é a identificação dos dados a serem compartilhados ou sincronizados: além da posição e orientação do observador, é necessário verificar se outros dados precisam ser sincronizados. Isso pode incluir estados internos do aplicativo, opções de visualização (esqueleto - *wireframe* - ou sólido, por exemplo), entre outros. De forma geral, adiciona-se o código referente à distribuição de dados nos métodos de processamento de entrada de dados e o código referente à sincronia nos métodos de geração de imagem. Como apresentado na seção 3.2.1.2, existe também a abordagem de *Cálculo centralizado dos dados gráficos e distribuição*, na qual o nó Mestre utiliza o algoritmo *sort-first*, *sort-middle* ou o *sort-last* para fazer o particionamento dos dados entre os Escravos. Entretanto, essa técnica não será abordada, pois está fora do escopo deste trabalho, sendo então proposta como trabalho futuro.

O ideal é que qualquer que seja a estratégia de desenvolvimento escolhida, a sincronia dos dados (*datalock*) e a sincronia dos quadros (*swaplock*) devem ser feitas da mesma maneira. A sincronia dos dados deve ser realizada antes do processamento dos dados e a dos quadros deve ser feita no momento da troca dos acumuladores de imagem. A escolha da estratégia não interfere na implementação do som, que é dependente do equipamento disponível. A solução mais simples é usar um nó para a geração do som, o que simplifica a implementação. Soluções mais avançadas podem criar imersão sonora, usando-se diversas caixas acústicas para gerar som espacialmente localizado.

Após as considerações técnicas sobre as estratégias de desenvolvimento, apresenta-se, na seção a seguir, um exemplo de porte de aplicação utilizando-se estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída*.

5.2 Estratégia de Desenvolvimento Utilizando Bibliotecas para Programação Distribuída

Atualmente existem diversas aplicações de Realidade Virtual que foram desenvolvidas para serem executadas em sistemas fortemente acoplados. Esta seção mostra que, além de possibilitar o desenvolvimento de novas aplicações, a estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída* pode ser utilizada para portar aplicações que foram desenvolvidas para sistemas fortemente acoplados. A seguir, tem-se o porte de uma aplicação gráfica simples para um sistema de multiprojeção baseado em aglomerados gráficos. Utilizou-se a biblioteca Glass e a biblioteca gráfica, a GLUT [GLU04].

A Figura 51 apresenta a imagem gráfica da aplicação a ser portada utilizada neste exemplo. O aplicativo consiste em uma série de esferas que giram em torno do observador. O objetivo é fazer com que esta aplicação seja executada em quatro telas de um sistema de multiprojeção.

Figura 51 – Imagem da aplicação em um único nó

A Figura 52 apresenta o código fonte dessa aplicação de forma resumida, que é composto pelas seguintes funções:

- `glutDisplay`: é chamada sempre que a janela precisar ser redesenhada. É nesta função que são chamadas as funções OpenGL.
- `glutKeyboard`: é chamada cada vez que uma tecla é pressionada (por exemplo: a, b, A, b, 1, 2).
- `main`: são chamadas as rotinas de criação de janela, inicializações e registros de *callbacks*.
- `glutIdle`: é executada durante os momentos ociosos (sem interação do usuário).

```
#include <stdio.h>
#include <GL/glut.h> /* contém os protótipos das funções utilizadas
pelo programa. Ele também inclui os headers gl.h e glu.h que
definem, respectivamente, as bibliotecas de funções OpenGL e a
GLUT*/
void glutDisplay (void) {
    . . .
    // aplica-se a rotação
    glRotated((deg+j*120)%360, 0, 1, 0);
    . . .
    // redesenha a esfera
    glutSolidSphere(8.0, 10.0, 10.0);
    . . .
    for (i = 60; i >= -60; i -= 20 )
        for ( j = 0; j < 3; j++ ) {
            glPushMatrix();
            glRotated((deg +j*120)%360, 0, 1, 0);
            . . .
        }
    . . .
    // apresenta as imagens
    glutSwapBuffers();
}
. . .

void glutKeyboard (unsigned char key, int x, int y) {
    . . . // processa eventos de teclado
}

void glutIdle(void) {
    . . . // atualiza a animação
}

int main (int argc, char **argv) {
    // inicializa a aplicação
    . . .
    glutCreateWindow("Around - Glass Application");
    // registra as funções
    glutIdleFunc(glutIdle);
    . . .
    // inicializa a "máquina de estados" e processa as mensagens
    // específicas do sistema operacional
    glutMainLoop();
}
```

Figura 52 – Código resumido da aplicação a ser portada

A Figura 53 mostra as imagens geradas (multiprojeção). São apresentadas quatro instâncias da aplicação, cada uma gerando um ponto de vista. As duas instâncias localizadas na parte superior da figura mostram o momento em que uma série de esferas se move da janela localizada à direita para a janela à esquerda, apresentando uma sincronia correta.

Figura 53 – Aplicação portada

Esta aplicação foi portada com a utilização da abordagem de *Cálculo centralizado dos resultados e distribuição*. O nó responsável pelo cálculo dos resultados, a nova posição da esfera, é o nó de maior prioridade. A prioridade de cada nó das aplicações Glass obedece à ordem de inicialização. Dessa maneira, o primeiro nó a ser inicializado recebe a prioridade, seguindo do segundo, e assim por diante. O nó responsável pela distribuição dos dados é o servidor interno da Glass. As tarefas a serem feitas para o desenvolvimento/porte da maioria das aplicações de Realidade Virtual são as seguintes:

- **Inserção das barreiras:** para a implementação da sincronia de dados, criam-se pontos de sincronização, nos quais os nós ficam parados até que todos eles tenham chegado a este ponto. Nesse momento são todos desbloqueados, e executam a troca dos acumuladores de imagem. Para isso, criou-se, neste exemplo, uma barreira que serve como ponto de sincronização de dados.
- **Criação de métodos referentes a cada ponto de vista:** cada nó deve gerar o ponto de vista da imagem de acordo com a sua tela: um nó deve gerar o ponto

de vista da direita, outro da esquerda, e assim por diante. Então, definiram-se as funções `frente`, `esquerda`, `direita` e `porta`. Cada uma delas calcula o ponto de visão respectivo. Por exemplo, a função `frente` não modifica a posição ou orientação, enquanto que a função `esquerda` rotaciona o observador em 90° em torno do eixo vertical, e assim sucessivamente.

- **Criação de variáveis compartilhadas:** para manter a coerência, todos os nós fazem a geração das imagens tendo como base o mesmo ponto de vista. Assim definiu-se uma variável compartilhada, que tem como objetivo armazenar a posição atual dos objetos.
- **Associação do nó à instância:** inicialmente a cada nó associa-se um ponto de vista, permitindo a multiprojeção.

Realizando essas tarefas, a maioria das aplicações de Realidade Virtual pode ser desenvolvida/portada para ambientes multiprojeção baseados em aglomerados gráficos com relativa facilidade. Pois, geralmente, possuem geometria estática, requerendo apenas a sincronia dos dados de controle, e da posição e orientação do observador.

Existem várias alternativas de bibliotecas para computação distribuída que podem ser utilizadas; entretanto, a escolha deve ser realizada analisando-se os requisitos mínimos, por exemplo, se a latência de transmissão de dados é suficiente para esse tipo de aplicação. Entretanto, cabe ressaltar que a biblioteca Glass traz diversas vantagens para tal tipo de aplicação, pois foi projetada tendo como base os seus requisitos. A maneira simples de implementação das funcionalidades (sincronização, compartilhamento de dados e associação), como apresentada no capítulo anterior, facilita o processo de desenvolvimento ou de porte de aplicações.

A Figura 54 mostra o código resumido da implementação.

```
// cabeçalhos

void frente (void)
{ }

void esquerda (void)
{glRotated(90, 0, -1, 0); }

void direita (void)
{      glRotated(90, 0, 1, 0);}

void porta (void)
{glRotated(180, 0, -1, 0);}

. . .

void glutDisplay (void) {
    . . .
    // sincroniza os dados
    . . .
    // corrige o ponto de vista
    . . .
    // renderiza os objetos
    . . .
    // barreira para troca dos acumuladores
    glutSwapBuffers();
}

. . .

void glutKeyboard (unsigned char key, int x, int y) {
    // trata os eventos
    . . .
}

int main (int argc, char **argv) {
    . . .
    // inicializa biblioteca
    . . .
    // cria barreira
    . . .
    // cria variáveis compartilhadas
    . . .
    // associa os pontos de vista
    . . .
    glutMainLoop();
}
```

Figura 54 – Código da aplicação portada

Após as considerações técnicas sobre a implementação de aplicações, apresentam-se, na seção a seguir, algumas aplicações que foram portadas utilizando-se a Glass.

5.3 Exemplos de aplicações

As aplicações de Realidade Virtual são, por natureza, altamente interativas, utilizam diversos dispositivos de *hardware* e possuem diferentes requisitos para o seu desenvolvimento, quando comparadas com as aplicações convencionais da computação. Esta seção mostra exemplos de aplicações de Realidade Virtual que foram portadas utilizando-se a Glass, e demonstra a viabilidade da utilização de aglomerados gráficos.

Nestas implementações foram testados os pontos-chave desse tipo de aplicação, como a comunicação e sincronização, mostrando então que podem ser atendidos. Também, avaliaram-se funcionalidades como a reinicialização das aplicações em tempo de execução. Por exemplo, finalizou-se uma aplicação em um nó e reinicializou-se outra novamente, verificando-se se houve perda de sincronização. Outro teste realizado foi o de tratamento de erros, por exemplo, a simulação da queda da rede de um nó. Esses testes foram suficientes para os problemas apresentados, ou seja, as aplicações não perderam a sincronização e não travaram.

Utilizando-se o GEditor, tornou-se possível testar o uso de PDA como dispositivo de interação em ambientes de Realidade Virtual imersivos, e utilizando-se o GVoicer foi possível valer-se de comandos de voz para navegar nas aplicações (foi empregado um microfone sem fio).

As aplicações apresentadas a seguir foram executadas no aglomerado gráfico Perfect Cluster, apresentado na seção 3.3, tendo como biblioteca gráfica a OpenGL. Todas as aplicações estão sendo apresentadas na CAVERNA^{TM} Digital. Utilizou-se um PDA IPAQ H3970 com tela 240 x 320 e com o sistema operacional Linux, sendo o padrão IEEE 802.11b a tecnologia de comunicação sem fio usada. A taxa de transmissão (11 Mb/s) dessa tecnologia foi suficiente para a interação em tempo real. O desempenho das aplicações teve como limite a capacidade de processamento da placa gráfica utilizada.

A Figura 55 mostra um exemplo de aplicação desenvolvida utilizando-se GEditor e sendo executada no PDA. Esta aplicação possui um conjunto de *skins* (Planetas, Estrelas, Luas, Cometas e outras) que podem ser trocadas em tempo de

execução. Por exemplo, se o usuário deseja navegar nos planetas, basta que ele selecione a *skin Planets*, tendo então como opção a navegação nos planetas pertencentes à Via Láctea (Marte, Terra, Plutão e outros).

Figura 55 – Aplicação executada no PDA

Notou-se que os PDAs, além de serem utilizados para facilitar a interação com os ambiente imersivos, principalmente de usuários inexperientes, podem controlar o próprio ambiente, executando aplicações, controlando luzes e projetores. Os testes mostraram que mesmo os usuários não familiarizados com tais dispositivos podem utilizá-los de maneira fácil. Outra vantagem: são um meio eficiente de apresentação de grandes volumes de dados, pois os usuários estão acostumados com a sua interface. Além disso, podem-se apresentar respostas visuais da aplicação e adaptar a interface da aplicação, que está em execução, de acordo com as características do usuário em tempo de execução.

A desvantagem apresentada na utilização dos PDAs refere-se à necessidade de visualização da sua tela para a interação, o que o torna não recomendado para algumas simulações, como, as de combates aéreo, pois neste tipo de simulação, os usuários devem estar constantemente visualizando o ambiente, caso contrário, eles não conseguem gerar as interações em tempo suficiente. Entretanto, se o tempo gasto na visualização da tela não prejudicá-la, como na aplicação Celestia [CEL04] e na Cathedral, que serão apresentadas a seguir, os PDAs podem ser utilizados.

A Figura 56 mostra usuários utilizando um PDA para navegar na aplicação Celestia, que é uma aplicação de astronomia, no qual o código fonte foi modificado

para suportar multiprojeção e estereoscopia. Este exemplo mostra que aplicações complexas (aquelas em que os atributos dos objetos são modificados no decorrer do tempo, mesmo sem interação do usuário) podem ser portadas para serem executadas em aglomerados gráficos, e que podem ter como dispositivo de interação PDAs.

Figura 56 – Usuários navegando na aplicação Celestia

Para o porte desta aplicação e das aplicações a seguir, utilizou-se a estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída* com a abordagem de *Cálculo dos resultados dos dados centralizados e distribuição* apresentada na seção 5.2. Como essa abordagem permite a recuperação dos valores compartilhados no decorrer da execução e a rede do Perfect Cluster é *error free* (pouca ou nenhuma perda de pacotes), então implementaram-se algumas aplicações, como o sistema visualizador de esferas, apresentado na seção 5.2, utilizando-se o protocolo UDP, no qual a entrega dos pacotes não é confiável, mesmo assim, não foi possível notar visualmente nenhuma perda de sincronia das imagens. As aplicações a seguir utilizaram TCP, pois o desempenho da Glass com este protocolo foi suficiente.

Utilizando-se o GVoicer e um microfone sem fio acoplado em um nó do aglomerado gráfico, implementou-se também o controle via voz na aplicação Celestia, o que possibilitou interagir com ela, por meio de comandos de voz, como: “Earth”, “Pluto”, “Mars” e outros. Quando o usuário fala algum desses comandos, o

sistema apresenta o planeta correspondente. A Figura 57 mostra o usuário interagindo com o Celestia utilizando comandos de voz.

Figura 57 – Celestia controlado por comandos de voz

A combinação do reconhecimento de voz com a mobilidade do microfone sem fio tornou o sistema flexível, e permitiu a movimentação do usuário durante sua interação. Além disso, como a voz é inerente aos seres humanos, criou um ambiente natural de interação com a aplicação.

Entretanto, o sistema reconhecedor de voz utilizado pelo GVoicer, o Sphinx-4, gera erros, que podem ser divididos em três categorias: as de rejeição (o reconhecedor não reconhece os comandos); a de reconhecimento incorreto (o reconhecedor entende uma palavra diferente da que foi pronunciada); e a de alarme falso (o usuário não fala, mas o reconhecedor produz um resultado). Durante os testes, notou-se que diversos fatores influenciaram na elevação dos erros de reconhecimento do sistema, por exemplo: maneira de pronunciar as palavras, o dispositivo de captura de som (microfone) e som do ambiente. Atualmente, o GVoicer não suporta o treinamento do reconhecedor de voz – acredita-se que melhores resultados podem ser obtidos se houver o treinamento.

A tecnologia de reconhecimento de voz teve um grande progresso nos últimos anos, mas ainda apresenta limitações quando aplicada a interações entre os usuários e computadores. Existem diversos desafios a serem superados; dentre eles: a enorme diversidade de palavras, a existência de palavras com a mesma pronúncia e até mesmo as características que diferenciam um usuário do outro. O sinal de voz

depende de características físicas do sistema vocal, que por sua vez dependem de fatores como sexo, idade e sotaque. Além disso, o clima, o ambiente e a formalidade da situação podem influenciar o tom de voz e a precisão da fala, o que dificulta o reconhecimento [MEI03].

A Figura 59 apresenta outras imagens do Celestia sendo executada na CAVERNA^{TM} Digital.

Figura 58 – Imagens do Celestia

A Figura 59 mostra a Cathedral, que é outro exemplo de porte de aplicação realizado e utilização do PDA. O modelo dessa aplicação foi cedido por Marko Dabrovic (mdabrov@rna.hr).

Figura 59 – Usuário navegando na aplicação Cathedral

A Figura 60 mostra o simulador de avião o GL-117 portado [GL104]. A interação é realizada utilizando-se luvas. Nessa aplicação, além da posição dos aviões, os mísseis também estão sendo sincronizados, o que mostra que a Glass pode ser utilizada para aplicações com alta taxa de interação [KAN04]. Ainda, demonstra a flexibilidade dessa biblioteca ao incorporar um novo dispositivo de interação. Notou-se que o sistema teve um bom tempo de resposta às interações dos usuários.

Figura 60 – Simulador de avião GL-117

A Figura 61 mostra a flexibilidade da Glass, ao possibilitar o porte de uma aplicação de visualização de veículos que foi desenvolvida utilizando-se a biblioteca OpenSceneGraph [OPE03d], e, ainda, ao suportar um dispositivo de interação do tipo rastreador (*Flock of Birds*).

Figura 61 – Visualizador baseado no OpenSceneGraph

5.4 Conclusão

A construção de aplicações de Realidade Virtual baseadas em aglomerados gráficos depende do conhecimento de várias áreas da ciência da computação, como da Interação Homem-Máquina, de Redes e Comunicações, de Sistemas Operacionais, de Hipermissão e Multimídia. Utilizando os conceitos e tecnologias destas áreas, este capítulo focou as estratégias de desenvolvimento de aplicações, o uso da estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída* e apresentou exemplos.

Foram abordadas as estratégias de desenvolvimento, que podem ser a de *Desenvolvimento Automático*, em que o desenvolvedor utiliza uma ferramenta para converter uma aplicação centralizada em uma aplicação baseada em aglomerados gráficos; ou a de uso da estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída*, em que o desenvolvedor adiciona o código na aplicação para comunicação e sincronização dos dados.

Esta última estratégia, que foi a contribuição deste capítulo, a de uso de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída* com a abordagem de *Cálculo centralizado dos resultados e distribuição*, possibilitou o porte das diversas aplicações apresentadas neste capítulo. O porte foi realizado executando-se as seguintes tarefas: *Inserção de barreiras*, que adicionou os pontos sincronização de dados; *Criação de métodos referentes a cada ponto de vista*, em que se criaram os métodos referentes ao ponto de vista de cada tela do dispositivo de multiprojeção; *Criação de variáveis compartilhadas*, no qual criaram-se os dados a serem sincronizados; e por último, a tarefa de *Associação do nó a instância*, em que se associou cada nó a um ponto de vista.

Capítulo 6

---

6. CONSIDERAÇÕES FINAIS

Os benefícios pertinentes ao uso dos aglomerados gráficos para as aplicações de Realidade Virtual são essenciais em uma sociedade globalizada onde cada vez mais a competitividade e a produtividade serão fatores determinantes para o sucesso das empresas. O uso de aglomerados gráficos para execução de aplicações de Realidade Virtual não é somente uma tendência para os próximos anos, é uma necessidade. A partir do momento em que este reúne em seu escopo características como escalabilidade, diminuição de custos, tolerância a falhas, desempenho e outras, torna-se uma representação de uma forma adequada e produtiva de execução de tais aplicações, sendo inclusive a que exige o menor valor financeiro de investimento.

Entretanto, as aplicações de Realidade Virtual baseadas em aglomerados gráficos demandam um suporte que possua desde a infra-estrutura de comunicação e desenvolvimento até a capacidade de manipular mensagens e acessar dados compartilhados presentes em nós diferentes. Esse trabalho constituiu um esforço no sentido de mostrar a viabilidade dessa solução; para isto, apresentou um ambiente de desenvolvimento de aplicações de Realidade Virtual que se constituiu do aglomerado gráfico Perfec Cluster, da biblioteca Glass, de uma estratégia de desenvolvimento de aplicações e de diversos exemplos.

Nele foi dada ênfase às aplicações de multiprojeção, como as que são executadas nos dispositivos do tipo CAVERNA^{TM} Digital, com o compartilhamento e sincronização dos dados entre os diversos nós do aglomerado gráfico. Pelos resultados obtidos e apresentados nos capítulos 4 e 5, chegou-se à conclusão de que, utilizando-se uma biblioteca adequada, como a Glass, e uma estratégia de desenvolvimento, como a apresentada, os aglomerados gráficos podem ser utilizados como ambiente de execução de aplicações de Realidade Virtual de multiprojeção.

Em seguida, são mostradas as características presentes no suporte implementado e feitas algumas considerações a respeito da utilização deste.

6.1 Os aglomerados gráficos na Realidade Virtual

A crescente utilização de aglomerados gráficos tem causado um forte impacto no mercado de sistemas fortemente acoplados (supercomputadores). Nos últimos anos, as vendas de máquinas paralelas dedicadas sofreram algumas baixas, em virtude das conquistas atingidas com os aglomerados. A disponibilização de diferentes tecnologias de comunicação, o envolvimento de um número cada vez maior de pesquisadores, o conhecimento adquirido e a solução de alguns problemas pertinentes a este tipo de arquitetura resultaram em plataformas de testes e ambientes de programação e de gerenciamento, que de forma geral são confiáveis, eficientes e são capazes de prover a maioria das funcionalidades requeridas pelas aplicações tradicionalmente executadas neste ambiente.

No caso específico das aplicações de Realidade Virtual, o uso de aglomerados gráficos é viável, desde que a biblioteca de comunicação e sincronização utilizada seja capaz de atender os requisitos mínimos de comunicação e sincronização. Funcionalidades adicionais, como as disponíveis na Glass, como a depuração de mensagens, suporte a processos-leves e a facilidade de uso, são necessárias para a criação de aplicações cada vez mais complexas. Apesar de o desenvolvimento de aplicações para aglomerados gráficos ser difícil por natureza, pois este não possui memória compartilhada, esta pesquisa mostra que a maioria das aplicações de Realidade Virtual com multiprojeção pode ser desenvolvida facilmente, pois os objetos são geralmente estáticos, sendo necessário apenas o compartilhamento dos pontos de vista do observador e das interações dos usuários entre os diversos nós.

6.2 A biblioteca Glass

A biblioteca Glass, que permite a construção de aplicações de Realidade Virtual baseadas em aglomerados gráficos, foi criada a partir do estudo do desenvolvimento de diversas aplicações no Laboratório de Sistemas Integráveis da Universidade de São Paulo. Esse estudo identificou requisitos a serem atendidos e originou uma arquitetura flexível, capaz de incorporar novas funcionalidades sem a alteração de sua estrutura básica.

Além da arquitetura flexível, cabe destacar que a Glass atingiu os requisitos propostos:

- Todas as funcionalidades são usadas utilizando-se os princípios básicos de programação, como, a sintaxe na declaração das variáveis, o que facilita o seu uso;
- A utilização desta biblioteca não impede a utilização de outras;
- Ela suporta recursos como processos-leves, que geralmente não são totalmente suportadas pelas bibliotecas que comunicação;
- Pode-se utilizar qualquer ferramenta para a depuração das aplicações;
- É independente da biblioteca gráfica;
- Os mecanismos de comunicação e sincronização são suficientes para esse tipo de problema.

Espera-se, então, que os desenvolvedores de aplicações Glass utilizem as funcionalidades desta biblioteca, de modo que a manutenção e o tempo de desenvolvimento das aplicações sejam reduzidos, que a compatibilidade entre as aplicações seja aumentada e que seja possível desenvolver aplicações mais complexas.

6.3 A estratégia de desenvolvimento

As aplicações projetadas e implementadas durante esse projeto utilizaram a estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída*, o que acarretou uma experiência realista a respeito da complexidade da construção destas aplicações. A implementação de operações, como a comunicação de dados e sincronização, tornou ainda mais clara a necessidade de uma ferramenta flexível como a Glass, onde novos protocolos de comunicação podem ser incorporados conforme a necessidade e/ou outras bibliotecas podem ser utilizadas. Acredita-se que no futuro novos trabalhos poderão ser desenvolvidos, contribuindo em termos de complementação e de evolução do suporte atual.

A estratégia de *Desenvolvimento Utilizando Bibliotecas para Programação Distribuída* não altera metodologias já consagradas como a UML, apenas adiciona

algumas tarefas a mais, como o desenvolvimento de métodos específicos para a geração de cada ponto de vista do sistema de multiprojeção e o uso de variáveis compartilhadas para o compartilhamento de dados. Entretanto, utilizando-se a Glass, essas tarefas podem ser implementadas de maneira simples.

Espera-se, então, que os desenvolvedores concentrem os esforços de desenvolvimento no conteúdo das aplicações, que são os problemas de visualização e manipulação dos dados, e não nos detalhes complexos de programação dos aglomerados gráficos.

6.4 Trabalhos futuros

Diversos trabalhos futuros podem ser propostos por esta pesquisa. Dentre eles:

- A implementação de novos protocolos na Glass, como o XTP e o VIA;
- O projeto e implementação de uma Aplicação de Suporte (interface gráfica) capaz de gerenciar todas as Aplicações Glass (inicialização, finalização, reinício);
- O desenvolvimento de funcionalidades de acordo com os princípios da Computação em Grade, visando um ambiente flexível, seguro, com recursos coordenados e compartilhados;
- Adição de recursos de multimídia, incluindo-se animações e sons nas aplicações geradas pelo GEditor;
- Adição de funcionalidades para a depuração dos valores das variáveis no GTracer;
- Melhoria do sistema de reconhecimento de voz do GVoicer;
- Utilização da Glass para o desenvolvimento de aplicações tradicionais de processamento paralelo;
- Implementação de *plugins* na Glass para a geração de imagem paralela utilizando as técnicas de *sort-first* e *sort-last*;
- A integração e avaliação de aplicações geograficamente distantes.

6.5 Publicações

A seguir, as diversas publicações na área que consolidaram essa pesquisa.

• Artigos internacionais

- GUIMARÃES, M. P. & GNECCO, B. & ZUFFO, M. **Graphical Interaction Devices for Distributed Virtual Reality Systems**. Virtual-Reality Continuum and its Applications in Industry –VRCAI.Singapura. Jun. 2004.
- SOARES,L. & CABRAL,M. & DULLEY,L. & GUIMARÃES,M. & LOPES, R. & ZUFFO, M. **Virtual Hang-gliding over Rio de Janeiro**. Workshop VR for Public Consumption. IEEE VR 2004. Mar. 2004.
- GUIMARÃES, M. P. & GNECCO, B. & CABRAL, M. & SOARES, L. & ZUFFO, M. **Synchronization and data sharing libray for PC clusters**. Workshop on Commodity Clusters for Virtual Reality - VR-CLuster'03. Los Angeles. USA. Marc. 22th-26th.2003.
- ZUFFO, M. & GUIMARÃES, M. P. **XML Opportunities in Real Time Immersive Simulation & Visualization Based on Clusters of Commodity Computers**. Extensible Modeling and Simulation Framework (XMSF) Technical Challenges Workshop. Aug.19-20. Naval Postgraduate School. Monterey. California. 2002.

• Artigos nacionais

- GUIMARÃES, M. P. & GNECCO, B. & ZUFFO, M. **Ferramenta de Geração de Interfaces Gráficas para PDAs**. Simpósio de Realidade Virtual. Ribeirão Preto. 2003.
- GUIMARÃES, M. P & BRESSAN, P. ZUFFO, M. **Frame Lock Synchronization for Multiprojection Immersive Environments Based on PC Graphics Clusters**. SRV – Simpósio de Realidade Virtual. Fortaleza. 2002.
- GUIMARÃES, M. P. & ZUFFO, M. **Estudo Comparativo de Comunicação em Aplicações Java 3D**. Universidade de Passo Fundo. III Simpósio de Informática do Planalto Médio. Universidade de Passo Fundo - Passo Fundo (RS).21 a 24 de Maio de 2002.

● **Posters nacionais**

- MARTINS,S.I & GUIMARÃES,M.P. & GNECCO, B.B. & ZUFFO,M.K.
Simulações de ambientes virtuais geograficamente distantes. VII Symposium on Virtual Reality. oct. 19-22th. São Paulo. Brasil. 2004.
- KANESHIRO, T. Y. & GNECCO, B. B. & GUIMARÃES, M. P. & ZUFFO, M. K.
Ambientes Imersivos de Multiprojeção de Tempo Real. VII Symposium on Virtual Reality. Oct. 19-22th. São Paulo. Brasil. 2004.
- GNECCO,B. B & GUIMARÃES, M. P & ZUFFO, M.K. **Um Framework Flexível e Transparente para Computação Distribuída de Alto Desempenho.** Simpósio de Realidade Virtual. Ribeirão Preto. 2003.

● **Livros**

- CARDOSO,A. **Ambientes Virtuais: projeto e implementação.** Porto Alegre. 2003.
Capítulos 4 e 5.

● **Revista nacional**

- GUIMARÃES, M. P & ZUFFO, M. **Caverna Digital: ambiente de imersão e interação.** Revista Acta Científica. São Paulo.2002. (Artigo aceito).

● **Tutoriais**

- GUIMARÃES, M. P & GNECCO,B. B. & BRESSAN, P & ZUFFO, M.
Programação Distribuída Aplicada a Realidade Virtual Tutorial. Simpósio de Realidade Virtual. Ribeirão Preto. 2003.
- BRESSAN, P & GUIMARÃES, M. P & SOARES, L.P. CABRAL, M.C & KURASHIMA,C & TODESCO,. & ZUFFO, M. **Visualização de Alto desempenho: utilizando componentes Convencionais.** SRV – Simpósio de Realidade Virtual. Ribeirão Preto. 2003.
- ZUFFO, M. & SOARES,L. P. & BRESSAN, P. GUIMARÃES, M. P. **Commodity Cluster for Immersive Projection Environments.** Tutorial. SRV – Symposium on Virtual Reality. Fortaleza. 2002.

7. REFERÊNCIAS BIBLIOGRÁFICAS

- [AHM01]AHMAD,I. & MAJUMDAR, S. **Achieving high performance in CORBA-based systems with limited heterogeneity.** Object-Oriented Real-Time Distributed Computing. 2001.
- [ALL02a]ALLARD,J. & GOURATION,V. & LECOINTRE,L. & MELIN,E. & RAFFIN,B. **Net Juggler: Running VR Juggler with Multiple Displays on a Commodity Componet Cluster.** IEEE VR 2002.Orlando. Florida. Disponível em: <http://netjuggler.sourceforge.net/Publications.php>. Acesso em: 12 dez. 2002.
- [ALL02b]ALLARD,J. & GOURATION,V. & LECOINTRE,L. & MELIN,E. & RAFFIN,B. **Net Juggler and SoftGenLock: Running VR Juggler with Active Stereo and Multiple Displays on a Commodity Component Cluster.** Proceedings of the IEEE Virtual Reality 20002 (VR'02). 2002. Disponível em: <http://netjuggler.sourceforge.net/SoftGenLock.php>. Acesso em: 03 out. 2002.
- [ALL02c]ALLARD,J. & GOURATION,V. & LECOINTRE,L. & MELIN,E. & RAFFIN,B. **Parallelizing Pre-rendering Computations on a Net Juggler PC.** Immersive Projection Technology Symposium. Orlando. Florida. USA. .2002. Disponível em: <http://netjuggler.sourceforge.net/Publications.php>. Acesso em: 22 dez. 2002.
- [ALV02]ALVES,M. J. P. **Construindo Supercomputadores com Linux.** Editora Brasport. Rio de Janeiro. 2002.
- [ARA97]ARAUJO, R. M. & DIAS, M. S. & BORGES, M. R. S. **A Framework for the Classification of Computer Supported Collaborative Design Approaches.** III CYTED-RITOS International Workshop on Groupware. Madrid Spain. oct.1997
- [ARC01]ARCARI,G.M. **Realidade Virtual e Odontologia.** Workshop de Informática aplicada à Saúde – CBComp. Univali. Itajaí. SC. 2001. Disponível em: <http://www.cbcomp.univali.br/anais/indice.htm>. Acesso em: 15 jun. 2002.

- [ARC02]ARCINIEGAS,F. **C++ XML**.Tradução: Flavia Barktevicus Cruz. Person Education do Brasil Ltda. 2002.
- [ARN97]ARNETT,M. F. **Desvendando o TCP/IP**. Tradução de ARX Publicações. Rio de Janeiro. Editora Campus. 1997.
- [BEN02]BENINI, L. & BONFIGLI, M.E. & CALORI, L. & FARELLA,E & RICCÒ, B. **Palmtop Computers for managing Interaction with Immersive Virtual Heritage**. Proceedings of EUROMEDIA 2002, pp. 183-189.2002.
- [BRE01]BRESSAN, P.A. & ZUFFO, M.K. **Reducing latency to volume visualization on PC cluster**. Brazilian Symposium on Computer Graphics and Image Processing. pp. 374. 2001.
- [BIE00]BIERBAUM,A.D. **VR Juggler: A virtual platform for virtual reality application development**. Master of Science. Iowa State University. Ames. Iowa. 2000.
- [BUR94]BURDEA,G & COIFFET,P. **Virtual Reality Techonology**. Wiley-Interscience Publication. jun. 1994.
- [BUY99]BUYYA,R. **High performance cluster computing. Programming and Applications**. Volume 2. Prentice Hall PTR.1999.
- [CAM96]CAMPO, M. R & PRICE, R. T. **Um Framework Reflexivo para Ferramentas de Visualização de Software**. Anais do X - Simpósio Brasileiro de Engenharia de Software. Brasil. out.1996.
- [CAR03]CARDOSO,A. **Ambientes Virtuais: projeto e implementação**. Porto Alegre.. Capítulos 4 e 5. 2003.
- [CAP99]CAPIN,T. K. & PANDZIC,I.S. & THALMANN,D. **Avatars in Networked Virtual Environments**. John Wiley & Sons. 1999.
- [CAR97]CARNEIRO,T.G.S. **Balanceamento de Carga em Ambientes de Realidade Virtual Distribuídos**. I Semana de Pós-Graduação em Ciência da Computação. UFMG. 1997. Disponível em: <http://www.dcc.ufmg.br/pos/html/spg97/anais/sexta.html>. Acesso em: 19 set. 2002.
- [CEL04]CELESTIA. Disponível em: <http://www.shatters.net/celestia/>. Acessos em : 01 de jan. 2004.

- [CES01]CESAR,A.K.& LANGIE,L.C. & PINHO,M.S & ANDRADE,A.C. **Um Dispositivo de Rastreamento Baseado em Cabos**. SBC 4th Symposium on Virtual Reality – SVR. 2001.
- [CHE01a]CHEN,Y & CHEN,H. & CLARK,D.W. & LIU, Z. & WALLACE,G & LI,K.**Software Environments for Cluster-based Display Systems**. The First IEEE/ACM International Symposium on Cluster Computing and the Grid - CCGrid 2001. Brisbane. Australia. 15-18 may 2001 Disponível em: <http://www.cs.princeton.edu/omnimedia/papers.html>. Acesso em: 17 jan. 2004.
- [CHE01b]CHEN,Y & CHEN,H. & FINKELSTEIN,A. FUNKHOUSER,T. & LI,K & LIU, Z. & SAMANTA,R. & WALLACE,G. **Data Distribution Strategies for High Resolution Display**. Computers and Graphics Vol. 25. PP. 811-818.2001.
- [CLU04]**CLUSTERING**. Disponível em: <http://www.oliver-abert.de/opensg/Clustering.html>. Acesso em: 17 aug. 2004.
- [COS00]COSTA, A. & SILVA,C.G & BRAGA,F.D & FANTINATO,M. **Realidade Virtual**. Disciplina IA368F - Tópicos em Engenharia de Computação V. DCA/FEEC/UNICAMP. 2000. Disponível em: <http://www.dca.fee.unicamp.br/courses/IA368F/>. Acesso em: 12 abr. 2001.
- [CRU92]CRUZ-NEIRA,C & SANDIN,D& DEFANTI,T. **The CAVE, Audio Visual Experience Automatic Virtual Environment**. Communications of the ACM. pp. 64-72. jun. 1992.
- [CYB04]**CYBersickness**. Hong Kong University of Science & Tecnology. Disponível em: <http://www.cybersickness.org/>. Acesso em: 22 set.04.
- [DAN02]DANTAS,M.**Tecnologias de Redes de Comunicação e Computadores**. Excel Books do Brasil Editora Ltda. 2002.
- [DIE01]DIEHL,S.**Distributed Virtual Worlds**. Foundations and Implementation Techniques Using VRML, Java, and Corba. Springer.2001

- [ESQ01]JES-QALLI, T. & FLEURY, E. & GUYARD, J.& BHIRI, S. **Evaluating the performance of CORBA for distributed and grid computing applications.** IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001). Brisbane. Australia. may 2001.
- [FER99]FERREIRA,A.G.**Uma Arquitetura de Visualização Distribuída de Ambientes Virtuais.** Dissertação de Mestrado. Departamento de Informática PUC-RIO. dez. 1999.
- [FOR97]FORSBERG, A. & JUNIOR, J. J. L. & MARKOSIAN, L. & ZELEZNIK, R. **Seamless interaction in Virtual Reality.** IEEE Computer Graphics and Applications. Vol. 17. No. 6.pp. 6 – 9. 1997.
- [FOS04]FOSTER,I. & KESSELMAN,C. **The Grid 2: Blueprint for a New Computing Infrastructure.** Elsevier. 2004.
- [FOW97]FOWLER, M & SCOTT, K .**UML Distilled - Applying the Standard Object Modeling Language.** Addison-Wesley. 1977.
- [GL104]GL-117 **Flight simulator.** Disponível em <http://home.t-online.de/home/Primetime./gl-117/gl-117.html>. Acessado em 20 jun. 2004.
- [GLU04]GLUT - **OpenGL Utility Toolkit.** Disponível em: <http://www.sgi.com/software/opengl/glut.html>. Acesso em: 2 maio 2004.
- [GNE01]GNECCO, B. B. & BRESSAN, P. A.& LOPES, R. D.& ZUFFO, M. K. **DICELib: A Real Time Synchronization Libray for Multi-Projection Virtual Reality Distributed Environments.** 4th SBC Symposium on Virtual Reality. Proceedings of SRV 2001. pp.338-343. out. 2001.
- [GNE03]GNECCO,B. B & GUIMARÃES, M. P & ZUFFO, M.K. **Um Framework Flexível e Transparente para Computação Distribuída de Alto Desempenho.** Simpósio de Realidade Virtual. Ribeirão Preto. 2003.
- [GRI94]GRIGORE,B & COIFFET,P. **Virtual Reality Technology.** A Wiley-Interscience Publication. New York. NY.USA. 1994.
- [GRI02]GRIEPP,T. & CRUZ-NEIRA,C. **XJL – an XML Schema for the Rapid Development of Advanced Synthetic Environment.** Immersive Projection Technology - IPT Symposium 2002. Orlando. FL. mar. 2002.

- [GUI02a]GUIMARÃES, M. P & ZUFFO, M. **Caverna Digital: ambiente de imersão e interação.** Revista Acta Científica. São Paulo. 2002.(artigo aceito).
- [GUI02b]GUIMARÃES, M. P & BRESSAN, P. ZUFFO, M. **Frame Lock Synchronization for Multiprojection Immersive Environments Based on PC Graphics Clusters.** SRV – Simpósio de Realidade Virtual. Fortaleza. 2002.
- [GUI02c]GUIMARÃES, M. P. & ZUFFO, M. **Estudo Comparativo de Comunicação em Aplicações Java 3D.** Universidade de Passo Fundo. III Simpósio de Informática do Planalto Médio. Universidade de Passo Fundo - Passo Fundo (RS).21 a 24 de maio de 2002.
- [GUI03a]GUIMARÃES, M. P. & GNECCO, B. & CABRAL, M. & SOARES, L. & ZUFFO, M. **Synchronization and data sharing libray for PC clusters.** Workshop on Commodity Clusters for Virtual Reality - VR-CLuster'03. Los Angeles. USA. marc. 22th-26th.2003.
- [GUI03b]GUIMARÃES, M. P. & GNECCO, B. & ZUFFO, M. **Ferramenta de Geração de Interfaces Gráficas para PDAs.** Simpósio de Realidade Virtual. Ribeirão Preto. 2003.
- [GUI04]GUIMARÃES, M. P. & GNECCO, B. & ZUFFO, M **Graphical Interaction Devices for Distributed Virtual Reality Systems.** Virtual-Reality Continuum and its Applications in Industry –VRCAI.Singapura. jun. 2004.
- [HAN01]HANNENA,D.L.**Interaction in Virtual Reality.** Master thesis. University of Amsterdam. Amsterdam. The Netherlands. sep. 2001.
- [HAR01]HARTILING, P. & JUST,C. & CRUZ-NEIRA,C. **Distributed Virtual Reality Using Octopus.**Virtual Reality 2001 Conference.. Yokohama. Japan. mar. 2001.
- [HAR02]HARTLING, P. & BIERBAUM, A. & CRUZ-NEIRA, C. **Tweek: Merging 2D and 3D Interaction in Immersive Environments.** 6th World Multiconference on Systemics, Cybernetics. and Informatics. Orlando. Florida. jul. 2002.

- [HIG04]**HIGH Resolution Tiled Display Wall - Room 139 Computer Building.**
Disponível em: <http://gears.aset.psu.edu/viz/facilities/displaywall/>.
Acesso em: 10 novembro 2004.
- [HIL00]HILL, L. & CRUZ-NEIRA, C. **Palmtop interaction methods for immersive projection technology systems.** Fourth International Immersive Projection Technology Workshop (IPT 2000).2000.
- [HUM99]HUMPHREYS,G. & HANRAHAN,P. **A Distributed Graphics System for Large Tiled Displays.** Proceedings of IEEE Visualization '99.
Disponível em: <http://www.cs.virginia.edu/~humper/>. Acesso em: 27 maio 2002.
- [HUM00]HUMPHREYS,G. & BUCK,I. & ELDRIDGE,M. & HANRAHAN,P. **Distributed Rendering for Scalable Display.** Proceedings of IEEE Supercomputing 2000. Disponível em: <http://www.cs.virginia.edu/~humper/>. Acesso em: 25 maio 2002.
- [HUM01]HUMPHREYS,G. & ELDRIDGE,M. & BUCK,I. & STOLL,G. & EVERETT,M. & HANRAHAN,P.**WireGL: A Scalable Graphics System for Clusters.** Preceedings of SIGGRAPH 2001. Disponível em: <http://www.cs.virginia.edu/~humper/>. Acesso em: 19 jan. 2003.
- [HUM02]HUMPHREYS,G. & HOUSTON,M. & NG,R. & FRANK,R. & AHERN,S. & KIRCHNER,P. & KLOSOWSKI,J.T.**Chromium: A Stream Processing Framework for Interactive Graphics on Clusters.** Proceedings of SIGGRAPH 2002. Disponível em: <http://www.cs.virginia.edu/~humper/>. Acesso em: 24 maio 2002.
- [HWA93]HWANG.K. **Advanced Computer Architecture: Paralelism, Scalability, Prprogrammability.** McGraw Hill.1993.
- [INT03]**INTRODUÇÃO** ao MPI. Centro Nacional de Processamento de Alto Desempenho em São Paulo. Disponível em: <http://www.cenapad.unicamp.br/servicos/treinamentos/mpl.shtml>. Acesso em : 10 jan. 2003.
- [JAV04]**JAVA** Native Interface. Disponível em: <http://java.sun.com/j2se/1.3/docs/guide/jni/>. Acesso em: 01 maio 2004.

- [KAN04]KANESHIRO, T. Y. & GNECCO, B. B. & GUIMARÃES, M. P. & ZUFFO, M. K. **Ambientes Imersivos de Multiprojeção de Tempo Real**. VII Symposium on Virtual Reality. Oct. 19-22th. São Paulo. Brasil 2004.
- [KIR01]KIRNER,C. **Realidade Virtual**. Grupo de Pesquisa em Realidade Virtual. Departamento de Computação. Universidade Federal de São Carlos – UFSCar. Disponível em: <http://www.dc.ufscar.br/~grv/tutrv/tutrv.htm>. Acesso em: 03 mar. 2001.
- [KIR03]KIRNER, C. **Realidade virtual: Dispositivos e Aplicações**. Disponível em: http://www.realidadevirtual.com.br/publicacoes/apostila_rv_disp_aplicacoes/apostila_rv.html. Acesso em: 20 fev. 2003.
- [KIW01]KIWELEKAR,A.W. & SINHA,P.H. **Evaluating CORBA as a Cluster middleware for Heterogeneous Cluster**. ADCOM. 9th International Conference on Advanced Computing and Communications. Bhubaneshwar. India. December. 2001. Disponível em: <http://www.csse.monash.edu.au/~rajkumar/ADCOM/>. Acesso em: 11 mar. 2002.
- [LEC03]LECHENER,T. **Stream Process Unit for CAVE**. Disponível em: http://pubweb.northwestern.edu/~tel768/CAVE_SPU.zip. Acesso em: 16 mar. 2003.
- [LEV99]LEVERAGING **Object – Oriented Frameworks**. <http://www.ibm.com/java/education/ooleveraging/>. Acesso em: 27 out. 1999.
- [LIN02]LIN,P & PAN,Z &YANG,J. & SHI,J. **Implementation of a Low-Cost CAVE system Based on Networked PC**. Virtual Environment on a PC Cluster Workshop 2002. Protvino. Russia. September. 2002. Disponível em: <http://viswiz.gmd.de/VEonPC/2002/about.html>. Acesso em: 04 nov. 2002.
- [MAC92]MACHADO,F.B. & MAIA, L.P. **Introdução à arquitetura de sistemas Operacionais**. 3. ed. Rio de Janeiro. Livros Técnicos e Científicos.1992.

- [MAR04] MARTINS, S.I & GUIMARÃES, M.P. & GNECCO, B.B. & ZUFFO, M.K. **Simulações de ambientes virtuais geograficamente distantes**. VII Symposium on Virtual Reality. oct. 19-22th. São Paulo. Brazil. 2004.
- [MEI99] MEIGUINS, B.S. **Uso de Realidade Virtual em Ensino a Distância Mediado por Computador**. Pontifícia Universidade Católica de Campinas. Dissertação de mestrado. Campinas. 1999.
- [MEI03] MEIGUINS, B. S. & GONÇALVES, A.S. & SANTOS, D.N.A. & GARCIA, M.B. & JUNIOR R.D.S. **Interação em Ambientes Virtuais Tridimensionais Utilizando Comando de Voz**. VI Symposium on Virtual Reality. Ribeirão Preto. oct. 15-18th 2003.
- [MVI04] **M-VIA: A High Performance Modular VIA for Linux**. Disponível em: <http://www.nersc.gov/research/FTG/via/>. Acesso em: 20 maio 2004.
- [NET02] NETTO, A & MACHADO, L. & OLIVEIRA, M. **Realidade Virtual - Definições, Dispositivos e Aplicações**. Sociedade Brasileira de Computação. Revista Eletônica de Iniciação Científica. Tutorial. Simpósio de Realidade Virtual. Brasil. Ano II. Volume II. Número I. mar. 2002.
- [NET04] **NET Juggler : a single image VR juggler cluster**. Disponível em: <http://netjuggler.sourceforge.net/NetJuggler.php>. Acesso em: 20 set. 2004.
- [OLS02] OLSON, E. C. **Cluster Juggler – PC Cluster Virtual Reality**. Master thesis. Computer Engineering. Iowa State University. Ames. Iowa. 2002.
- [OPE03a] **OPENGL**. The Industry's Foundation for High Performance Graphics. Disponível em: <http://www.opengl.org/>. Acesso em: 21 jan. 2003.
- [OPE03b] **OPENGL Performer**. Disponível em: <http://www.sgi.com/software/performer/>. Acesso em: 21 jan. 2003.
- [OPE03c] **OPENAL**. Disponível em <http://www.openal.org/home/>. Acesso em: 15 jan. 2003.
- [OPE03d] **OPEN Scence Graph**. Disponível em: <http://www.openscenegraph.org/>. Acesso em: 16 jan. 2003.
- [ORF94] ORFALI, R. & HARKEY, D. & EDWARDS, J. **Instant Corba**. Published by Wiley. 313 pages. 1994.

- [ORF96]ORFALI, R. & HARKEY,D. & EDWARDS, J. **The Essential Distributed Objects Survival Guide**. Published by Wiley. 604 pages.1996.
- [OVR03]**OVERVIEW: Virtools™ VR Pack**. Disponível em: http://www.virtools.com/solutions/products/virtools_vrpack.asp. Acesso em: 21 jan. 2003.
- [PAI98]PAIVA, A. C & RODRIGUEZ, N. & GATTAS, M. **Técnicas Paralelas de Rendering**. Monographs. Computer Science Department, PUC-RioInf. MCC34-98. September 1998. Disponível em: <http://www.tecgraf.puc-rio.br/~gattass/monographs.html>. Acesso em: 15 jul. 2001.
- [PAR03]**PARALLEL Virtual Machine**. Disponível em: http://www.csm.ornl.gov/pvm/pvm_home.html. Acesso em: 10 jan. 2003.
- [PIN96]PINHO, M. S. **Realidade Virtual como Ferramenta de Informática na Educação**. SBIE. 1996. Disponível em: <http://grv.inf.pucrs.br/Pagina/Educa/educa.htm>. Acesso em: 12 dez. 2002.
- [PIN01]PINHO, M. S. & KIRNER, C. **Uma Introdução à Realidade Virtual**. Instituto de Informática – PUCRS. 2001.
- [PIT03]PITANGA, M.**Computação em Cluster: o estado da arte**. Brasport. Rio de Janeiro. 2003.
- [PIZ98]PIZZOL, A. M. **Um Framework para implementação na WWW de aplicações hipermídia modeladas com OOHDM**. Dissertação de Mestrado. Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro. 10 dez. 1998.
- [REI02a]REINERS,D. & BEHR,J. & VOSS,G. **OpenSG Starter Guide**. Version 1.0. Disponível em: <http://www.opensg.org/documentation.EN.html>. Acesso em: 17 ago. 2002.
- [REI02b]REINERS,D.**Scene Graph Rendering**. Open Source Virtual Reality Course at IEEE VR 2002. Orlando. Florida. USA. 2002. Disponível em: <http://www.vrjuggler.org/pub/scenegraph-rendering.ieeevr2002.pdf>. Acesso em: 03 dez. 2002.
- [REI02c]REINERS,D. & VOSS,G. & BEHR,J. **OpenSG: Basic Concepts**. 1º OpenSG Symposium. 29 Jan. 2002. Disponível em:

- <http://www.opensg.org/OpenSGPLUS/symposium/Papers2002/>. Acesso em: 28 abr. 2002.
- [REV04]**REVOLUTIONIZING Advanced Visualization**. Disponível em: http://www.nvidia.com/page/quadrofx_3000g.html. Acesso em: 20 jun. 2004.
- [SAN01]SANTOS, E. T. & ZUFFO, M. K. & NETTO, M. L. & LOPES,R. D. **Computação Gráfica - Estado da Arte e Pesquisa na USP**. GRAPHICA 2001 - 15. International Conference on Graphics Engineering for Arts and Design. São Paulo. Brazil. November. 2001. Disponível em: <http://www.lsi.usp.br/~lobonett/>. Acesso em: 23 jan. 2003.
- [SCH00]SCHAEFFER,B. **A Software System for Inexpensive VR Via Graphics Clusters**. 2000. Disponível em: <http://www.isl.uiuc.edu/ClusteredVR/paper/DGDooverview.htm>. Acesso em: 12 nov. 2002.
- [SCH02a]SCHIMDT,D. & VINOSKI, S. **Object Interconnections: Real-time CORBA, Part 1: Motivation and Overview**. C/C++ Users Journal. Disponível em: <http://www.cuj.com/experts/1912/vinoski.htm>. Acesso em: 01 maio 2002.
- [SCH02b]SCHAEFFER,B. **Networking and Management Frameworks for Cluster-Based Graphics**. Proceedings of Virtual Environments on PC Workshop. Protvino. Russia. September. 2002. Disponível em: <http://www.isl.uiuc.edu/Publications/publications.htm>. Acesso em: 13 nov. 2003.
- [SCH02c]SCHAEFFER,B. & GOUDESEUNE.C. **Syzygy: Native PC Cluster VR**. IEEE Virtual Reality Conference. Aug. 2002. Disponível em: <http://www.isl.uiuc.edu/syzygy.htm>. Acesso em: 20 set. 2004.
- [SGI01]SGI Graphics Cluster^{TM}: **The Cluster Architecture Challenges, the SGI^{TM} Solution**. Visualization Solutions Development Group. White Paper. 2001. Disponível em: http://www.sgi.com/products/legacy/vis_systems.html. Acesso em: 14 nov. 2001.

- [SIL01]SILBERCHATZ, A & GAGNE, G & GALVIN, P. B. **Sistemas Operacionais: Conceitos e Aplicações**. Editora Campus. 2001.
- [SIN99]SINGHAL, S. & ZYDA, M. **Networked Virtual Environments: design and Implementation**. Addison Wesley Professional. SIGGRAPH series.1999.
- [SOA04a]SOARES,L. & CABRAL,M. & DULLEY,L. & GUIMARÃES,M. & LOPES, R. & ZUFFO, M. **Virtual Hang-gliding over Rio de Janeiro**. Workshop VR for Public Consumption. IEEE VR 2004. mar. 2004.
- [SOA04b]SOARES, L. & ZUFFO, M. **JINX: an X3D browser for VR immersive simulation based on clusters of commodity computers**. Proceedings of the ninth international conference on 3D Web technology. Monterey. California. EUA. abr. 2004.
- [SOU96]SOUZA,M. **Avaliação das Rotinas de Comunicação Ponto-a-Ponto do MPI**. Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos da Universidade de São Paulo.dez. 1996.
- [SOU98]SOUZA, C.**Um Framework para Editores de Diagramas Cooperativos baseados em Anotações**. Dissertação de Mestrado. Universidade Estadual de Campinas. 12 de out. de 1998.
- [SPH04]SPHINX-4. Disponível em: <http://cmusphinx.sourceforge.net/sphinx4/>. Acesso em 03 maio de 2004.
- [STA03]STAADT,O.G & WALKER,J. & NUBER,C. HAMMANN,B. **A Survey and Performance Analysis of Software Platforms for Interactive Cluster-Based Multi-Screen Rendering**. Proceedings of IPT/EGVE 2003. ACM Press. Zurich. pp. 261-270. May 22-23. 2003.
- [TAN95]TANENBAUM, A. **Distributed Operating Systems**. Prentice Hall. 1995.
- [THE03a]THE Message Passing Interface (MPI) standard. Disponível em: <http://www-unix.mcs.anl.gov/mpi/>. Acesso em: 15 jan. 2003.
- [THE03b]THE Official Bluetooth Web site. Disponível em <http://www.bluetooth.com/>. Acesso em: 27 maio 2003.
- [TRO04]TROLLTECH. Disponível em <http://www.trolltech.com/>. Acesso em 22 jan. 2003.

- [USI02]**USING Wildcat's GenLock and Multiview Option in Visual Computing Applications.** Disponível em: <http://www.3dlabs.com/support/developer>. Acesso em: 08 jan. 2002.
- [VIR03]**VIRTUAL Realities.**Global Distributor of Quality Virtual Reality Products. Disponível em: <http://www.vrealities.com/>. Acesso em: 27 mar. 2003.
- [VOG03]VOGELS,W.H.P. **Scalable Cluster Technologies for Mission-Critical Enterprise Computing.** Thesis.Vrije Universiteit.11 nov. 2003.
- [WEB04]**WEB 3D Consortium.** Disponível em: <http://www.web3d.org/>. Acesso em 01 maio 2004.
- [WLO95]WLOKA, M. **Lag in Multiprocessor VR.** Presence: Teleoperators and Virtual Environments 4.1. Winter. pp. 50-63. 1995. Disponível em: www.vrac.iastate.edu/research_archive/hfactors/crcd/references.html. Acesso em: 07 set. 2002.
- [ZUF01]ZUFFO,J.A & SOARES,L.P & ZUFFO,M.K. & LOPES, R.D. **CAVERNA Digital – Sistema de Multiprojeção Estereoscópico Baseado em Aglomerados de PC's para Aplicações Imersivas em Realidade Virtual.** In Proceedings of the 4th Brazilian Symposium on Virtual Reality - SVR'01. pp. 139–147. Florianopolis. SC. Brazil. Oct. 16-19. 2001.
- [ZUF02a]ZUFFO, M. & SOARES,L. P. & BRESSAN, P. & GUIMARÃES, M. P. **Commodity Cluster for Immersive Projection Environments.** Tutorial.SRV – Symposium on Virtual Reality. Fortaleza. 2002.
- [ZUF02b]ZUFFO, M. & GUIMARÃES, M. P. **XML Opportunities in Real Time Immersive Simulation & Visualization Based on Clusters of Commodity Computers.** Extensible Modeling and Simulation Framework (XMSF) Technical Challenges Workshop. Naval Postgraduate School. Monterey. California. Aug.19-20. 2002.