UMA ARQUITETURA PARA A VISUALIZAÇÃO DISTRIBUÍDA DE AMBIENTES VIRTUAIS

ALEXANDRE GUIMARÃES FERREIRA

DISSERTAÇÃO DE MESTRADO

TeCGraf
DEPARTAMENTO DE INFORMÁTICA
PUC-RIO

Rio de Janeiro, 22 de dezembro de 1999

ALEXANDRE GUIMARÃES FERREIRA

UMA ARQUITETURA PARA A VISUALIZAÇÃO DISTRIBUÍDA DE AMBIENTES VIRTUAIS

Dissertação apresentada ao Departamento de Informática como parte dos requisitos necessários para a obtenção do título de Mestre em Ciências em Informática.

Orientador: Marcelo Gattass
Co-orientador: Waldemar Celes Filho

TeCGraf
DEPARTAMENTO DE INFORMÁTICA
PUC-RIO

Rio de Janeiro, 22 de dezembro de 1999

AGRADECIMENTOS

- A Marcelo Gattass, meu orientador, pelo seu apoio irrestrito e esforço continuado e conselhos adequados em todas as etapas de meu desenvolvimento.
- A Waldemar Celes, meu co-orientador, pelo esmero e atenção com que se dedicou em todas as fases deste trabalho, além das idéias e críticas que muito contribuíram para a sua realização.
- A meus pais, pelo apoio incondicional em todos os momentos.
- A minha noiva, pelo incentivo dado e pela compreensão nos diversos momentos em que precisei abdicar de estarmos juntos.
- A Renato Cerqueira, pelo auxílio no desenvolvimento e estruturação de parte deste trabalho.
- A Paulo Mattos, pelo desenvolvimento da biblioteca de múltiplos níveis de detalhe.
- Ao TecGraf e a todos os seus integrantes, pelo ambiente propício e positivo onde vivi parte de minha vida.
- Aos componentes da banca, por suas sugestões.
- Ao Exército Brasileiro e ao Instituto Militar de Engenharia (IME), pelo suporte financeiro durante o desenvolvimento deste trabalho.

RESUMO

Um grande número de aplicações requer um sistema de visualização que proporcione múltiplas visões de um ambiente virtual animado. Uma solução natural é a disposição de múltiplos dispositivos de visualização ao redor de um usuário, de forma a compor uma vista panorâmica da cena. Em um arranjo mais elaborado, vistas panorâmicas do ambiente virtual, obtidas a partir de diferentes pontos de vista, são fornecidas simultaneamente a diferentes usuários.

Cada cena panorâmica apresentada ao usuário é formada pela composição das imagens geradas em cada superfície de visualização. Desta forma, é possível imaginar cada dispositivo de visualização como sendo uma janela para o ambiente virtual.

O uso desta tecnologia visa realçar a percepção humana e aumentar o grau de imersão dos usuários. Isto é alcançado através do aumento quantitativo dos estímulos visuais fornecidos, decorrente do uso de múltiplas janelas.

Uma abordagem tradicional para o desenvolvimento de um sistema deste tipo é baseada em sistemas especializados, que utilizam *hardware* dedicado, responsável pelo controle e processamento centralizado de todos os dispositivos de visualização.

Este trabalho propõe uma arquitetura distribuída que proporcione maior escalabilidade, portabilidade e flexibilidade ao sistema de visualização utilizando uma rede heterogênea e estações gráficas de baixo custo.

A arquitetura proposta objetiva dar suporte ao desenvolvimento de aplicações que apresentem um resultado visual consistente do ambiente virtual. Para isso, são apresentadas técnicas para garantir o sincronismo e a integridade entre as diversas estações.

ABSTRACT

A number of applications require a visualization system that provides multiple visions of an animated virtual environment. The disposal of multiple visualization devices around a user consists in a natural solution to compose a panoramic view of the scene. Virtual environment panoramic views, simultaneously supplied to different users, can be obtained from different points of view, in a more sophisticated arrangement.

Each panoramic scene presented to a user is formed by the composition of images generated in each visualization surface. This way, it is possible to imagine each visualization device as a window to the virtual environment.

The use of this technology seeks to enhance human perception and to increase users' immersion. This is reached through a quantitative increase of the visual stimuli provided, due to the use of multiple windows.

A traditional approach for the development of such system is based on specialized systems that use dedicated hardware, which is responsible for the visualization devices' centralized processing and control.

This work proposes a distributed architecture to be used in heterogeneous networks with low-cost graphic workstations, seeking to provide greater scalability, portability and flexibility to the visualization system.

The proposed architecture seeks to support the development of applications that present a consistent visual outcome for the virtual environment. To accomplish that, techniques are presented to assure synchronism and integrity among several workstations.

ÍNDICE

ÍNDICE DE FIGURAS E TABELAS

CAPÍTULO 1 – INTRODUÇÃO

Há mais de duas décadas, a forma mais comum de interação entre os seres humanos e o computador é feita da seguinte forma: o usuário permanece sentado perante um monitor comandando um teclado e, mais recentemente, um *mouse*. Todavia, novas propostas e tecnologias vêm expandindo os limites dessa fronteira, permitindo ganhos e melhoramentos no processo de interação. Dentro desse escopo, busca-se por meio deste trabalho, apresentar uma solução cuja intenção é alargar o canal de recursos de interação no sentido máquina $\rightarrow$ homem.

No estágio atual do desenvolvimento tecnológico, um grande número de aplicações requer um sistema de visualização que proporcione múltiplas visões de um ambiente virtual animado. A Figura 1 ilustra a disposição de múltiplos dispositivos de visualização ao redor de um observador, possibilitando que este tenha uma visão panorâmica do ambiente virtual.

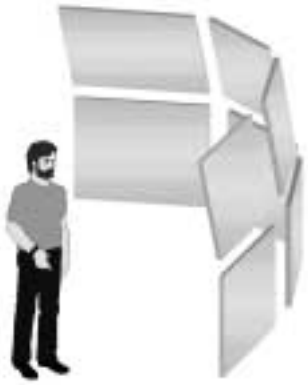

Figura 1 – Observador diante de múltiplos dispositivos de visualização

A imagem final apresentada ao observador é formada pela composição de imagens nos múltiplos dispositivos de visualização. Cada uma dessas imagens contém uma parte da imagem completa, obtida a partir de um processo de renderização simultânea do ambiente virtual sobre os diversos dispositivos de visualização. Desta forma, é possível imaginar cada dispositivo de visualização como sendo uma janela para o ambiente virtual. A utilização de tal configuração possibilita, entre outros aspectos, o aumento do grau de imersão e integração do usuário do sistema com o ambiente virtual apresentado.

Uma abordagem tradicional para se obter simultaneamente múltiplas vistas de um ambiente virtual é através do uso de sistemas especializados, que utilizam uma arquitetura de *hardware* dedicado para controlar os múltiplos dispositivos de visualização. Tal solução apresenta, geralmente, um custo elevado de implantação e escalabilidade limitada.

Este trabalho propõe uma arquitetura distribuída, que proporciona confiabilidade e flexibilidade ao sistema de visualização, utilizando uma rede heterogênea e estações gráficas de baixo custo. Essa arquitetura permite o desenvolvimento de aplicações que demandem altas taxas de quadros por segundo (*frame-rates)* e um resultado consistente da simulação ao usuário. Para isso, são traçados alguns requisitos e propostas soluções. Dentre os requisitos fundamentais para o funcionamento da arquitetura podem-se identificar a necessidade de sincronismo entre todos os eventos que ocorrem no mundo simulado e o resultado visual apresentado nas superfícies de visualização.

Esta dissertação apresenta a seguinte organização. Primeiro descrevem-se e analisam-se conceitos relacionados à visualização de ambientes virtuais, à sua inter-relação com a realidade virtual, focando-se em soluções distribuídas. A seguir apresenta-se uma arquitetura para a visualização distribuída que visa proporcionar escalabilidade, heterogeneidade e portabilidade na solução deste problema. Para avaliar a arquitetura proposta, é apresentada uma implementação do sistema, com a qual são realizados alguns testes. Com base no estudo da arquitetura e da sua implementação são traçadas algumas conclusões e fornecidas sugestões para trabalhos futuros.

O escopo deste trabalho não abrange o funcionamento dos processos de simulação que utilizam o sistema de visualização proposto. Eles são levados em consideração apenas no que diz respeito aos requisitos que induzem.

1.1 - Aplicações

Um grande conjunto de aplicações pode se beneficiar da utilização desta tecnologia. Áreas como educação, pesquisa científica, processo produtivo, aplicações militares, medicina e entretenimento são alguns dos exemplos abordados a seguir. [LOEFFLER94]

Na educação, a experiência visual e o grau de imersão proporcionados pela visualização de ambientes virtuais podem engajar os usuários, rompendo fatores como desinteresse e falta de estímulo, existentes ocasionalmente nos processos de ensino tradicional.

Na pesquisa científica, são vários os exemplos de estudos que se utilizam da inspeção e análise visual de dados e resultados de experimentos. Entre eles, pode-se citar a visualização de superfícies no campo da matemática, a análise de fluidos e tensões no campo da física, a pesquisa espacial, a modelagem molecular, etc. [ADAMS96].

Na medicina, observa-se uma gama de aplicações muito promissora, com ramificações em treinamento cirúrgico, ensino de anatomia, exames virtuais, tratamentos psicológicos (ex.: fobias, psicoses) e reabilitação [DURLACH95].

No processo produtivo, existem sistemas bastante complexos que exigem acompanhamento visual constante, como o roteamento de redes de companhias de telecomunicações, sistemas de acompanhamento e auxílio a decisões em bolsas de valores e no mercado financeiro de forma geral, entre outros. Existem também aplicações na indústria, no processo de *design* e projeto de máquinas e equipamentos, além de usos em engenharia e arquitetura, como a navegação por uma maquete virtual.

Na área militar, podem ser listadas diversas aplicações. No treinamento militar, um sistema desse tipo pode ser utilizado, por exemplo, na visualização em salas de jogos de guerra. Nesses jogos é simulado um teatro real de operações, sendo necessário um processo de visualização que proporcione a cada jogador uma visão dos elementos que o cercam no ambiente virtual. Outra aplicação seria o desenvolvimento de simuladores de diversos tipos de aeronaves e viaturas. Ela poderia também ser utilizada em sistemas de realidade aumentada. Por exemplo, um sistema de tiro de um carro de combate poderia ser composto por diversos visores onde seriam projetados elementos gráficos, representando alvos em potencial, obtidos a partir de sensores, que auxiliariam na calibragem correta dos sistemas de armas.

1.2 - Trabalhos relacionados

Os trabalhos mais relevantes em visualização distribuída são parte integrante de publicações e projetos que envolvem sistemas de realidade virtual e, mais especificamente, realidade virtual distribuída. O problema abordado neste trabalho encontra-se inserido nesse contexto. Isso decorre do fato de que a visualização de ambientes virtuais é um dos focos principais de estudo em sistemas de realidade virtual, que pode ser definida como "uma maneira do homem visualizar, manipular e interagir com computadores e dados extremamente complexos." [AUKSTAKALNIS92]

Destacam-se alguns sistemas de realidade virtual distribuída que devem ser levados em consideração pois apresentam visualização distribuída como parte integrante da solução por eles proposta:

AVIARY - É um Ambiente Virtual genérico, multi-usuário, de alto nível, projetado para explorar o paralelismo disponível em um sistema multi-processado ou em um sistema distribuído. Ele apresenta conceitos como "mundo virtual", que define um conjunto de atributos que serão processados por todas as entidades que o integram e especifica um conjunto de limites que governam o comportamento das entidades. Este sistema considera como "entidade" qualquer elemento que possa ser visto, ouvido, sentido ou experimentado de qualquer forma no mundo virtual [AVIARY].

DIVE - *Distributed Interactive Virtual Environment* - É um sistema de Realidade Virtual Distribuído Heterogêneo em que os usuários navegam em um espaço 3D e podem ver, encontrar e interagir com outros usuários e aplicações do ambiente. É baseado em UNIX e se utiliza de protocolos sobre TCP/IP para a comunicação em redes locais ou de longa distância. A consistência e o controle de concorrência dos dados comuns são alcançados por uma replicação ativa, protocolos de *multicast* confiáveis e métodos de bloqueio distribuído dos dados [DIVE].

The MR Toolkit - É um conjunto de ferramentas de *software* para a produção de sistemas de Realidade Virtual e outras formas de interfaces tridimensionais. Ele consiste de um conjunto de bibliotecas, acionadores de dispositivo, programas de apoio e uma linguagem de descrição de geometria e comportamento. Dentre suas características destaca-se a independência de dispositivos e a portabilidade para o desenvolvimento de aplicações de Realidade Virtual. Múltiplos usuários podem compartilhar o mesmo

ambiente virtual 3D. O MR Toollkit provê mecanismos eficientes para dividir uma aplicação em múltiplos processos e viabilizar a comunicação entre esses processos [MR].

NPSNET - É um pacote de simulação distribuída em tempo real de um ambiente virtual fornecido gratuitamente pela NPSNET – *Research Group at the Naval Postgraduate School* [NPSNET]. Ele faz uso de DIS (*Distributed Interactive Simulation*) [IEEE93] para simular articulações de braços humanos, veículos terrestres e aéreos, e embarcações que coexistam em um ambiente virtual. Atualmente ele permite a participação de cerca de 250 a 300 usuários utilizando-se de recursos computacionais e de rede compatíveis com os usualmente encontrados no mercado. Ele detém o título de primeiro ambiente virtual 3D que pode ser executado na Internet através de um *backbone multicast* (MBONE) [NPSNET]

PARADISE - O projeto PARADISE (*Performance Architecture for Advanced Distributed Interactive Simulation Environments*) propõe-se a produzir um ambiente de simulação 3D distribuído de larga escala que suporte sua utilização por múltiplos usuários interativamente, sendo executado em uma rede de longa distância [PARADISE].

Determinadas soluções tecnológicas empregadas nesses projetos foram consideradas e empregadas no desenvolvimento da arquitetura proposta neste trabalho. Dentre elas, destacam-se as técnicas de manutenção da consistência do estado dinâmico corrente, apresentadas na seção 2.3.1, a modelagem do ambiente virtual composto por entidades dotadas de diversos atributos e a decisão pela replicação parcial de dados dos elementos integrantes do ambiente virtual.

Uma vez que o enfoque apresentado neste trabalho está voltado para o problema da visualização, sua contribuição está relacionada com o estudo e a análise das técnicas empregadas. Como contribuição adicional, a arquitetura possibilita a aplicações já existentes a utilização de recursos de visualização por múltiplos dispositivos, sem a necessidade de que tais aplicações sejam reescritas nos moldes dos sistemas acima citados.

CAPÍTULO 2 – AMBIENTES VIRTUAIS

Conceituar ambientes virtuais, descrever os métodos de visualização, além de apresentar as técnicas utilizadas para sua visualização distribuída.

2.1 - Descrição

Existem várias possíveis formas de se descrever o que vem a ser um ambiente virtual. Uma interpretação seria a de que o ambiente virtual é um teatro no qual determinados aspectos do mundo real ou abstrato são simulados. Os mais diversos tipos de ambientes virtuais são simulados hoje em dia e, geralmente, cada um deles apresenta um método diferente de prospecção e análise dos dados que representam a simulação. Em alguns deles, é necessário que uma representação visual gráfica seja apresentada ao usuário como resultado da simulação. Encontram-se nesta categoria os ambientes virtuais de interesse deste estudo.

Serão denominados **entidades** todos os atores existentes neste teatro virtual. Uma entidade é, portanto, a representação virtual de algum elemento real que esteja sendo simulado. Os mais diversos atributos de uma entidade podem estar sendo simulados pelo ambiente virtual; todavia, os de maior interesse para este trabalho são aqueles relacionados com o resultado visual a ser apresentado ao usuário. Para exemplificar, uma simulação detalhada do lançamento de um míssil deveria dar atenção especial à potência, trajetória, inércia, ação de forças externas, entre outros fatores, enquanto os aspectos de interesse para a visualização levam em consideração fatores como posição, forma, textura da superfície, resposta à iluminação, etc.

2.1.1 - Atributos

No escopo do problema de visualização existem diversos atributos que devem ser considerados em um ambiente virtual. O conjunto desses atributos é essencial para alimentar o processo de renderização de cada entidade, sendo parcialmente responsável pelo grau de realismo das imagens geradas. O uso de modelos 3D detalhados e materiais ou texturas apropriadas geralmente produz bons resultados. A Figura 2 ilustra como o uso de diferentes tipos de atributos em uma mesma cena de um ambiente virtual influencia na qualidade visual.

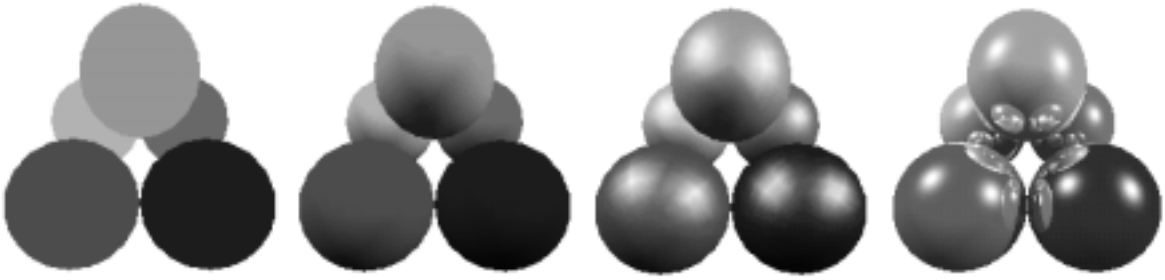

Figura 2 – Cena renderizada com diferentes níveis de atributos

Em primeiro lugar, são identificados os atributos relacionados com as entidades que integram um ambiente virtual. As entidades são geralmente compostas pelos atributos **descrição geométrica**, **aparência** e **comportamento**, como ilustrados na Figura 3.

Figura 3 – Atributos essenciais de uma entidade para visualização

Descrição Geométrica

A descrição geométrica é qualquer maneira de se descrever a forma da entidade que pode ser processada para se obter uma imagem dessa entidade. A modelagem geométrica de objetos é um campo de pesquisa vasto, existindo diversas abordagens possíveis. Um exemplo de um objeto geométrico simples é um ponto. Contudo, a descrição geométrica de uma entidade pode ser dada por uma equação implícita, por uma lista de polígonos, por superfícies de revolução, entre outros, ou até mesmo por uma combinação desses modelos.

A descrição geométrica pode também ser classificada como estática ou dinâmica no tempo. A descrição geométrica estática é aquela que não varia sua forma no tempo, enquanto a dinâmica é aquela que pode ter sua forma modificada por uma função relacionada ao tempo, bem como por ação de qualquer outro atributo da simulação, como um choque, o efeito de uma força, entre outros.

O grau de complexidade da descrição geométrica é, em geral, diretamente proporcional à qualidade visual, porém inversamente proporcional à velocidade com que a imagem é gerada. Essa premissa nem sempre é válida se for levada em consideração a excelente qualidade visual obtida quando se faz uso de texturas (atributo de aparência da entidade), mesmo sobre malhas pouco refinadas.

Definir qual a melhor descrição geométrica, qual o grau de complexidade da mesma e se é necessário ou não o uso de texturas é um problema complexo e bastante pesquisado. Em muitos casos, soluções baseadas em múltiplos níveis de detalhes (LOD – *Level of Detail*) são indicadas. Devido ao seu caráter adaptativo, o uso de múltiplos LODs na descrição de entidades (um tipo especial de descrição geométrica dinâmica) que compõem ambientes virtuais possibilita maiores taxas de renderização de quadros por segundo.

Aparência

O segundo atributo principal na descrição de entidades em um ambiente virtual é a sua aparência, que, assim como a descrição geométrica, interfere na qualidade da imagem final gerada. A aparência pode ser dada de diversas maneiras. Entre elas, aquelas que geralmente estão presentes nas aplicações de Realidade Virtual são:

- A simples ausência de aparência. Atributo útil para entidades presentes no ambiente virtual que não podem ser visualizadas.
- Cores opacas simples.

- A aparência baseada na definição do material, em que são levados em consideração fatores como sua componente de reflexão difusa, especular e ambiente, o coeficiente de brilho e a cor emitida pelo material.
- A aparência gerada pela aplicação de texturas. No seu formato mais simples, pode ser obtida a partir da simples deposição de uma imagem de textura sobre as primitivas geométricas. Aparências mais complexas podem ser obtidas a partir da composição por meio de modulação e *blending* de múltiplas imagens de texturas entre si ou sobre a primitiva geométrica.
- Texturas dinâmicas, em que os parâmetros que compõem a textura são funções do tempo, permitindo efeitos visuais como o de uma tela de televisão imersa no ambiente virtual exibindo um filme, a superfície do mar com ondas em ciclo ou o céu com nuvens em movimento.
- A combinação balanceada das aparências e dos atributos anteriores.

Comportamento

O terceiro atributo de grande interesse em ambientes virtuais é o comportamento da entidade. Podem existir ambientes virtuais onde todas as entidades sejam estáticas. Em outros casos, pode haver entidades que integram o ambiente virtual e que possuam algum atributo que varie no tempo. Deste modo, o comportamento de uma entidade pode ser definido como o conjunto de mudanças que qualquer um de seus atributos possa sofrer no decorrer do tempo. O contexto do comportamento não está relacionado apenas com mudanças na aparência e na descrição geométrica, mas principalmente com atributos como localização, velocidade e aceleração da entidade no espaço definido pelo ambiente virtual. É, portanto, através do controle do comportamento das entidades de um ambiente virtual que se torna possível a criação de um ambiente virtual animado.

O comportamento pode ser classificado em duas categorias distintas: determinístico e não-determinístico. Os comportamentos determinísticos são todos aqueles cujo estado pode ser definido como uma função do tempo. Este comportamento permite que se navegue para a frente e para trás no tempo, sabendo exatamente qual o estado de cada entidade em cada momento. Em contraposição, o comportamento não-determinístico é imprevisível, sendo geralmente reflexo de ações que não seguem a um padrão definido, como por exemplo certas ações humanas. Neste caso é impossível prever o estado de uma

determinada entidade que apresente esse comportamento no futuro e demasiadamente complicado restaurar seu estado no passado. [ROEHL95]

Roehl [ROEHL95] também classifica o comportamento em diferentes níveis, de acordo com a forma como os atributos das entidades são modificados no tempo:

- **Nível 0.** Estabelece a cada quadro o valor de determinado atributo. (Ex.: a posição da entidade Veículo neste momento é {*x*,*y*,*z*}.)
- **Nível 1.** Estabelece a forma como os atributos variam no tempo. (Ex.: a entidade Veículo esta posicionada em {*x*,*y*,*z*}, com velocidade 10 m/s e seguindo o rumo 250o NW.)
- **Nível 2.** Instrui a entidade a realizar uma tarefa, geralmente efetuada por um conjunto de comportamentos do nível 1. (Ex.: Siga com a entidade Veículo para a base mais próxima.)
- **Nível 3.** Instrui a entidade a tomar uma decisão de alto nível, geralmente executada pela escolha de comportamentos do nível 2. (Ex.: Solicite que a entidade Veículo decida entre atacar um inimigo ou recuar para base mais próxima.)

Ainda em relação ao comportamento, elemento melhor analisado adiante, a visualização de ambientes virtuais suporta geralmente os comportamentos de nível 0 e nível 1, uma vez que estes podem ser diretamente fornecidos pela simulação. Comportamentos de nível 2 e nível 3 requerem que o sistema de visualização empregado tenha um maior conhecimento do ambiente simulado, demandando, conseqüentemente, o uso de sistemas integrados.

Existem outros atributos que podem compor a definição de um ambiente virtual, sem que estes sejam tratados como entidades propriamente ditas. Dentre eles destaca-se um fator de extrema importância: a iluminação.

O uso de uma iluminação corretamente modelada auxilia na geração de imagens com um grau de realismo mais alto, quando a simulação em questão envolve elementos normalmente presentes no campo de visão humana. Para exemplificar, considere a importância da iluminação na simulação de um grupo de pessoas e objetos em uma sala, em comparação com uma simulação de reações químicas de moléculas. A primeira tende a exigir um modelo de iluminação mais complexo que reflita a realidade, enquanto a segunda requer um modelo de iluminação mais simples, uma vez que ela representa uma

abstração para fins de visualização sem a existência de parâmetros para comparação com a realidade.

Para que sejam alcançados bons resultados visuais, a descrição do ambiente virtual deve conter atributos para a definição de fontes de luz de diversos tipos, de acordo com o modelo de iluminação que venha a ser utilizado. Em particular, as interfaces de programação gráfica de aplicações (API gráfica) mais comuns implementam o modelo de iluminação de Phong [PHONG75], que contém os seguintes tipos de fontes de luz:

- **Ambiente:** componente de iluminação uniformemente aplicada a todas as primitivas da cena.
- **Posicional:** fonte de luz pontual com campo de influência em todo o espaço ao redor de sua posição no ambiente virtual, sendo os feixes de luz centrados em seu ponto base.
- **Direcional:** fonte de luz posicionada em um ponto no infinito em determinada direção, tendo como campo de influência todo o espaço, sendo ainda todos os feixes paralelos para fins de cálculos de iluminação.
- ***Spot*:** caso particular de uma fonte de luz posicional cujo campo de influência é limitado a um cone.

Dentre os atributos de cada tipo de fonte luminosa, podem ser destacados:

- **Intensidade da cor difusa:** qual a intensidade com que determinada fonte influencia na componente difusa do material de determinada primitiva.
- **Intensidade da cor especular:** idem para a componente especular.
- **Intensidade da cor ambiente:** idem para a componente ambiente.
- **Fator de decaimento da cor:** como decai a influência de uma fonte de luz em função de sua distância à primitiva que está sendo iluminada.

2.1.2 - Hierarquia

No ambiente virtual, as entidades estão geralmente organizadas segundo alguma estrutura inerente aos aspectos do mundo real sendo simulado. Esta estrutura representa a forma como as entidades estão relacionadas entre si. Podem existir ambientes virtuais nos quais as entidades que os habitam não apresentam qualquer relacionamento entre si,

coexistindo de forma independente. Contudo, é comum que determinados atributos de diferentes entidades estejam relacionados entre si.

Uma forma simples de relacionamento seria, por exemplo: a entidade B está fixa na entidade A. Em um relacionamento deste tipo, caso a entidade A sofra alguma modificação no seu atributo de posição, a entidade B deve acompanhar esta modificação. A simulação deve ser responsável pela manutenção e pelo controle da integridade dos relacionamentos.

Uma das estruturas encontradas na definição de ambientes virtuais é a hierárquica. Com ela, é possível criar entidades construídas a partir da composição de outras entidades sucessivamente. Mas os relacionamentos não se limitam ao posicionamento geométrico das entidades entre si; podem existir relacionamentos mais complexos envolvendo os diversos atributos citados na seção anterior.

Pode-se imaginar um ambiente virtual composto por alguns navios em deslocamento no mar. Cada navio, naturalmente, além de se deslocar com uma certa velocidade em uma determinada direção, pode apresentar uma oscilação em função das ondas do mar. Sobre um determinado navio podem haver entidades humanas se deslocando. E, ainda, em um determinado momento um helicóptero pode pousar sobre o passadiço de algum navio, e lá permanecer estacionado com seu rotor girando.

A simulação do cenário hipotético descrito acima requer uma estruturação elaborada, na qual seria útil, por exemplo, a possibilidade de se criar uma estrutura hierárquica comportamental entre as entidades presentes no ambiente virtual. Isso facilitaria, por exemplo, o cálculo da posição da pá do rotor do helicóptero em um determinado instante, levando em consideração sua posição em relação ao navio, bem como a oscilação, velocidade e direção do navio em relação aos outros navios.

Outro recurso importante é a possibilidade de, durante a simulação, mudar a posição e o relacionamento das entidades dentro da hierarquia. Isso permite que, ao pousar sobre o navio, o helicóptero passe a fazer parte da hierarquia do navio e herde seu comportamento, e que, ao decolar, volte a ter seu comportamento independente.

2.2 - Visualização

A visualização pode ser entendida como um processo computacional de geração, para o usuário, de estímulos sensoriais (visuais, auditivos, entre outros), de um ambiente virtual simulado [ISDALE93].

Como mencionado anteriormente, o objetivo deste trabalho é prover ao usuário meios de visualização de um ambiente virtual simulado. Surge então a necessidade de se conceituar como é feita a ligação entre o usuário e o ambiente virtual simulado.

O usuário de um sistema de visualização geralmente posiciona-se em frente a algum dispositivo de visualização, que pode ser um ou mais monitores de vídeo ou telas de projeção, ou os "veste", como no caso dos HMDs (*Head Mounted Displays*), que surgiram com o advento da realidade virtual.

Todavia, independentemente de qual o dispositivo utilizado, o usuário terá sempre um observador virtual a ele associado. Esse observador representa o mapeamento do usuário no ambiente virtual. Em certos ambientes virtuais, o observador pode estar ligado a uma determinada entidade e, com isso, pode herdar algumas de suas características, como comportamento, descrição geométrica ou aparência. Dessa forma ele passa a agir como uma entidade que pode ser vista por outros observadores dentro do ambiente virtual. Um exemplo simples é um observador posicionado ligado a uma entidade de um veículo. Uma vez que este veículo esteja em movimento, o observador herdará seu comportamento e poderá ser visto por outros possíveis observadores, que também poderão estar ligados a outras entidades imersas no mesmo ambiente virtual.

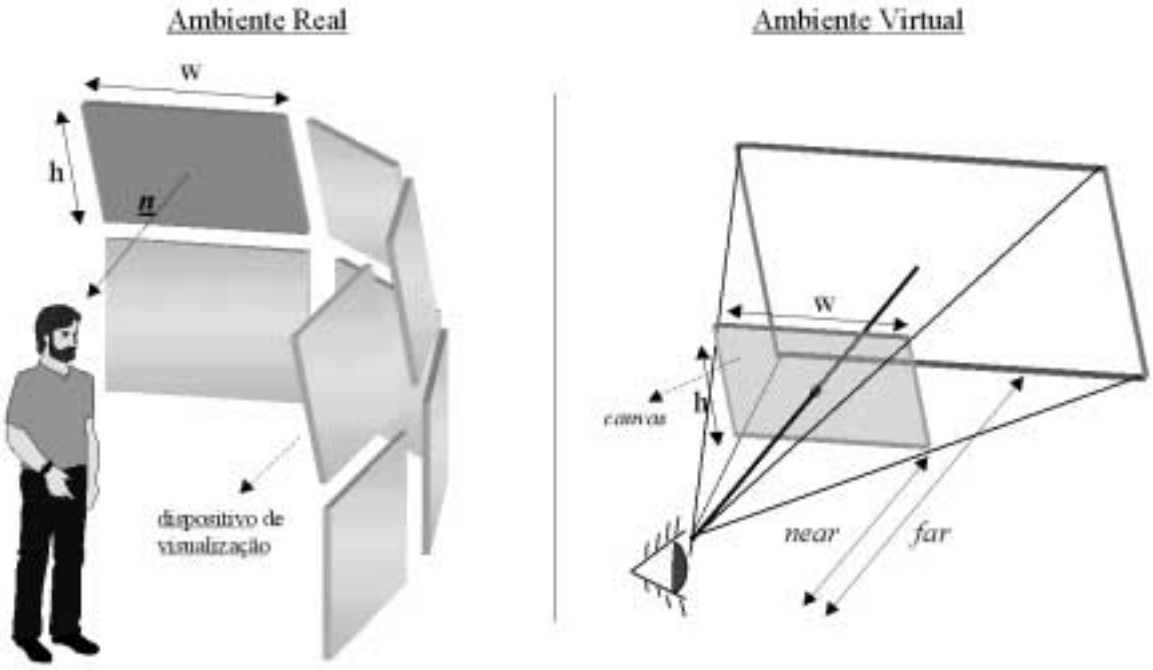

Figura 4 – Mapeamento de um dispositivo de visualização no *canvas*

Para cada observador existem uma ou mais superfícies de visualização virtuais que doravante serão denominadas *canvas*. Cada um desses *canvases* corresponde a um

dispositivo de visualização real posicionado ao redor do usuário. Dessa forma, cada dispositivo de visualização age como uma janela para o ambiente virtual. Diversos atributos, como tamanho, razão de aspecto, distância do usuário, entre outros, definem como é feito o mapeamento dos dispositivos de visualização sobre os *canvases*. A forma como os múltiplos dispositivos de visualização são dispostos fisicamente em relação ao usuário definem a forma como estão dispostos os *canvases* em relação ao observador. A Figura 4 ilustra o mapeamento de um dispositivo de visualização em um *canvas* e Figura 5 ilustra a importância do mapeamento da posição relativa entre os dispositivos de visualização e o observador.

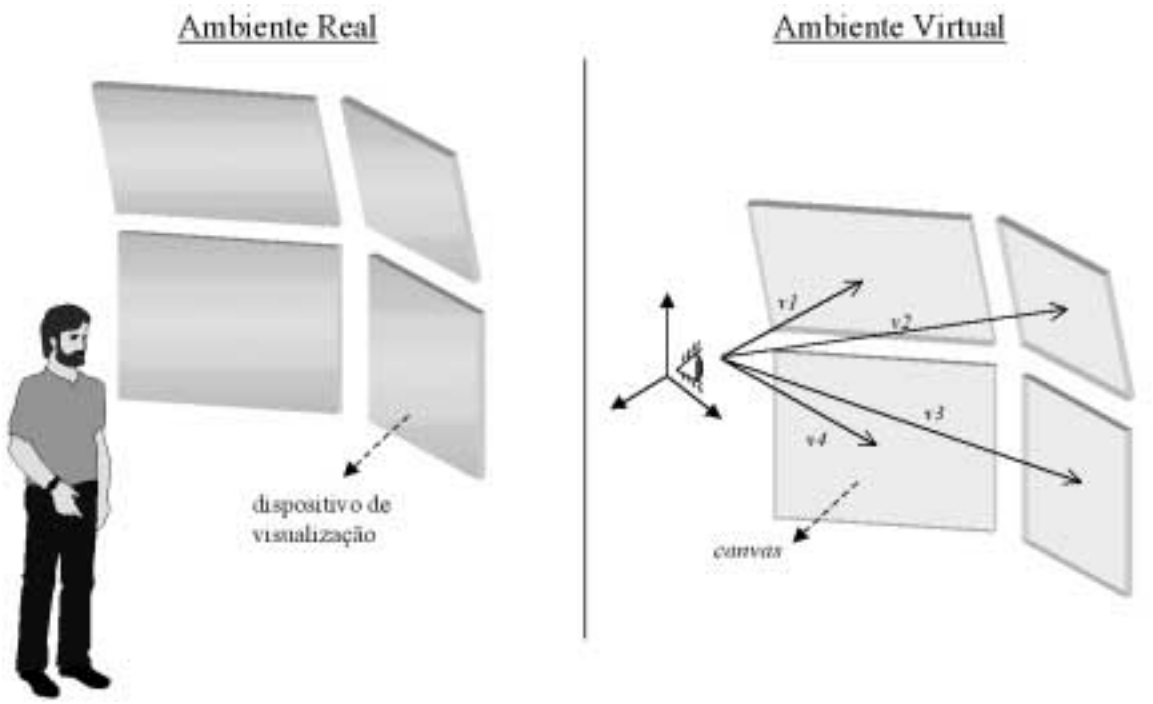

Figura 5 – Mapeamento de diversos dispositivos de visualização em relação ao observador

A tarefa de mapear cada dispositivo de visualização que cerca o usuário nos *canvases* que rodeiam o observador é muito importante, pois pequenos erros podem resultar na sobreposição ou ausência de pedaços visualizados no ambiente virtual, ou em objetos que apresentem distorções de escala e posicionamento, decorrentes de coeficientes incorretos na construção da matriz de projeção perspectiva. Conseqüentemente, resultados visuais corretos demandam o perfeito mapeamento dos dispositivos virtuais e reais.

Uma vez que os aspectos acima estejam solucionados, a preocupação seguinte é com o grau de imersão proporcionado aos usuários do sistema.

Deborah Barreau [BARREAU93] argumenta que um sistema que tenha por objetivo prover a um usuário a sensação de imersão deve induzi-lo a focar em determinado problema ou experiência sem distrações. A imersão ocorre quando a percepção do usuário é removida do mundo real e voltada para o mundo virtual. Geralmente, os ambientes de imersão são associados com HMDs, gráficos de alta definição, som realista e sensações táteis de retorno de movimento e retorno de força, usualmente utilizados para bloquear as sensações do mundo real e obrigar o usuário a focar exclusivamente no ambiente virtual. Ainda assim, podem ser criados ambientes de visualização imersivos sem muitos dos dispositivos acima mencionados, uma vez que uma simples projeção de vídeo, dependendo de seu conteúdo, pode engajar o usuário. Elementos como o uso de resoluções apropriadas e taxas de quadro por segundo suficientes para a visualização de determinado ambiente virtual animado podem torná-lo interessante de ser visto e inspirar a participação do usuário. Desta forma, para se criar um ambiente de visualização que seja imersivo, torna-se necessário que o projeto seja tecnicamente realista para convencer o usuário e que o conteúdo apresente relevância dentro do domínio de interesses do mesmo. Um ambiente imersivo para a visualização de dados sísmicos, por exemplo, possivelmente não apresentaria a mesma sensação de imersão para um piloto do que apresentaria para um geólogo, e vice-versa.

Conseqüentemente, o *design* de um ambiente de visualização imersivo deve ser capaz de integrar de forma eficaz os requisitos e a tecnologia disponível com os diversos fatores subjetivos que caracterizam a sensação humana.

2.3 - Visualização Distribuída

A proposta desta dissertação é apresentar uma arquitetura para o desenvolvimento de aplicações de visualização de ambientes virtuais adotando uma abordagem distribuída. A idéia central por trás de qualquer arquitetura distribuída é a ausência de um servidor central. O processamento é distribuído por diversas unidades que geralmente estão ligadas por uma rede (possivelmente inclusive pela Internet). A porção de interesse para o problema de visualização de cada entidade presente no ambiente virtual simulado será replicada em cada unidade de visualização, onde o processamento será localmente realizado de forma independente. Esse modelo de distribuição é conhecido como “base de dados homogeneamente replicada”.

No que diz respeito ao problema de visualização, uma abordagem distribuída introduz novas questões e requisitos que devem ser considerados: a **manutenção da consistência**, o **sincronismo**, a **heterogeneidade**, a **escalabilidade** e a **portabilidade**.

2.3.1 - Manutenção da consistência

O comportamento das diversas entidades presentes no ambiente virtual pode variar no tempo. O fator de fundamental importância é como garantir a consistência do estado dinâmico compartilhado nas diferentes unidades de visualização. O estado corrente dinâmico compartilhado pode ser definido como a variação do estado corrente (comportamento) que cada unidade de visualização deve manter de todas as entidades presentes no mesmo ambiente virtual. Ele pode ser composto, por exemplo, por informações como quais as entidades presentes, a posição relativa entre elas, seu comportamento, entre outras.

Levando em consideração que consistência é a medida de quão similar o ambiente virtual é visto por diferentes unidades de visualização, para se obter uma visualização consistente é fundamental que o estado dinâmico compartilhado seja mantido de forma precisa, de modo a imprimir no usuário a impressão de que ele está visualizando um ambiente virtual único, sem perceber que este está sendo independentemente gerado em diferentes máquinas. Este é um dos maiores desafios de um sistema de visualização distribuída, uma vez que deve-se encontrar um equilíbrio entre a carga da rede e o uso de recursos computacionais. Para resolver tal questão são geralmente adotadas as seguintes abordagens:

Propagação permanente do estado dinâmico corrente

Nesta abordagem, todos os atributos de cada entidade presente no ambiente virtual a ser visualizado são transmitidos (*broadcasted*) para todas as unidades de visualização do sistema toda vez que um novo quadro for ser renderizado. Neste caso, apenas instruções de comportamentos do nível 0 são enviadas pela rede.

Esta abordagem pode funcionar bem para pequenas redes dedicadas, porém existem alguns problemas relacionados com o uso deste modelo. O primeiro é que todas as máquinas devem estar na mesma sub-rede, premissa que nem sempre é válida. Em segundo lugar, todas as máquinas, estejam elas participando ou não da visualização, receberão os pacotes com novos atributos da simulação. Em terceiro lugar, transmitir todas as mudanças

de atributos a cada quadro de visualização potencialmente requer uma largura de banda alta, principalmente quando os ambientes virtuais têm um grande número de entidades. Isso também acarretaria em uma diminuição do número de quadros por segundo, uma vez que o processamento de um determinado quadro deve ficar aguardando a chegada pela rede de seus atributos. A freqüência com que os pacotes chegam às unidades de visualização também pode variar, acarretando variações na taxa de quadros por segundo devido à variação na latência da rede de um pacote para o outro (*jitter*).

A vantagem desta abordagem é a garantia de uma total consistência do estado dinâmico compartilhado em todas as unidades de visualização. Dessa forma, a imagem exposta em cada unidade de visualização reflete o estado exato do ambiente virtual a cada momento. Em contrapartida, uma solução desta natureza não é escalável, tornando inviável a sua utilização para a visualização de ambientes virtuais compostos por muitas entidades e que não estejam na mesma rede local.

Predição e convergência do estado dinâmico corrente

Para reduzir o tráfego e obter taxas de quadros por segundo maiores, de forma a permitir a visualização de um grande número de entidades, é necessário utilizar uma abordagem que abra mão da total consistência do estado dinâmico compartilhado.

A idéia por trás desta abordagem é transmitir pacotes apenas quando ocorrerem determinadas atualizações no estado dinâmico corrente. Enquanto o sistema não recebe uma nova atualização, cada unidade de visualização prevê os atributos baseando-se nos dados recebidos no último pacote de atualização. Quando um novo pacote chega, cada unidade de visualização atualiza seus atributos. Havendo disparidades entre o novo estado e o estado anterior, é feita uma convergência entre estes estados de forma corrigir a inconsistência. A Figura 6 ilustra essa técnica.

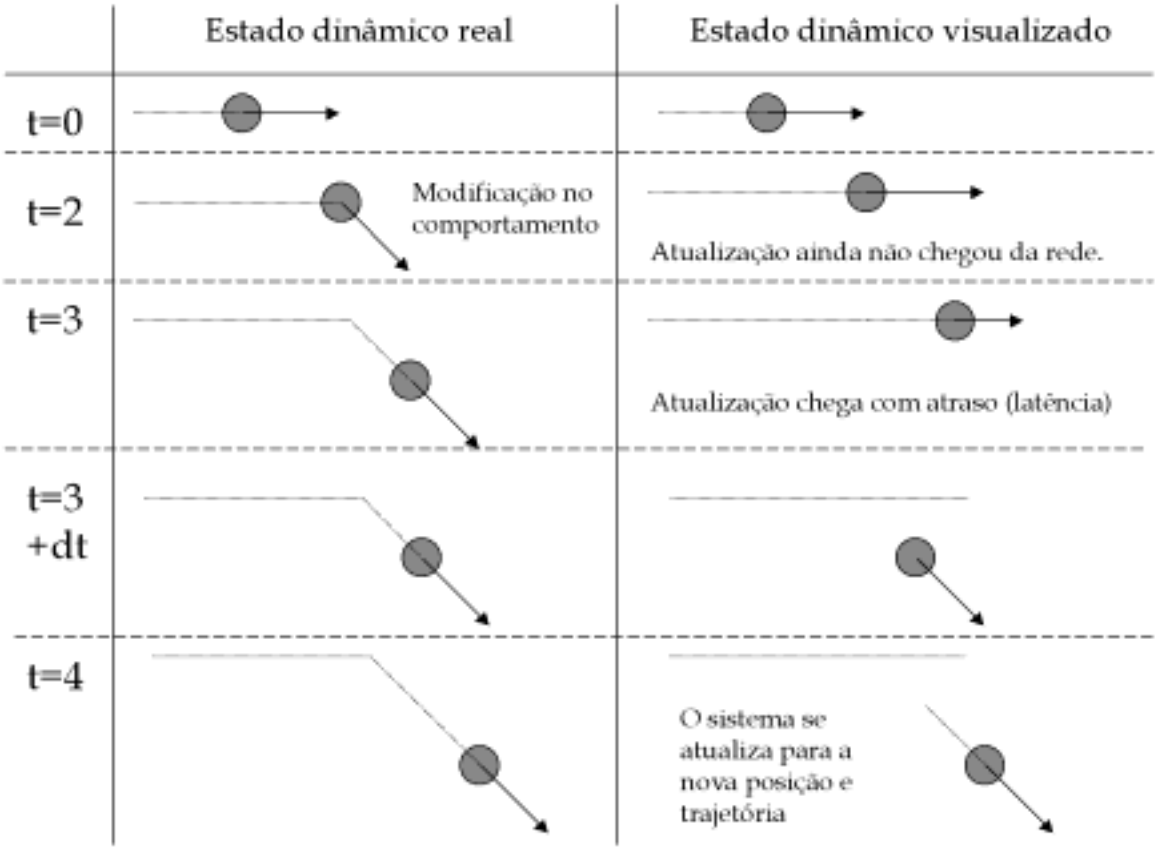

Figura 6 – Funcionamento da técnica *dead reckoning*

Esta técnica, conhecida como *dead reckoning*, é amplamente utilizada em diversos sistemas de realidade virtual distribuída, especialmente para simulações militares. Ela é a base do protocolo DIS (*Distributed Interactive Simulation*) [IEEE93], desenvolvido pela *Naval Postgraduate School* para uso no sistema NPSNET [NPSNET]**.**

A idéia central desta técnica é, ao invés de enviar instruções de comportamentos de nível 0 a cada quadro, descrevendo completamente todos os atributos de cada entidade, enviar instruções de comportamentos de nível 1, o que permite, de forma restrita, prever o estado das entidades em um futuro próximo. Uma das conseqüências do uso desta técnica é a necessidade de que cada unidade de visualização tenha conhecimento e compute, mesmo que parcialmente, o estado dinâmico de todas as entidades visualizadas.

A técnica *dead reckoning* pode ser dividida em duas fases: **predição** e **convergência**. Predição é a forma com que o estado dinâmico corrente é computado, baseado no estado recebido anteriormente, e convergência é a forma como são corrigidas eventuais diferenças entre os atributos preditos e os atributos recebidos.

As técnicas de predição mais utilizadas são baseadas no uso de equações diferenciais que definem o estado de cada entidade. Por exemplo, para um atributo de posição, em que

sua primeira derivada representa a velocidade, sua segunda derivada representa a aceleração e suas derivadas seguintes definem a forma como sua posição varia no tempo, se a equação que define esse comportamento for de conhecimento da simulação, basta que essa equação seja enviada para as unidades de visualização, capacitando-as a calcular localmente a posição da entidade em um intervalo de tempo no futuro. Dessa forma, são necessárias mensagens de atualização apenas quando houver uma alteração no comportamento, que neste contexto pode ser classificado como de nível 1.

Testes práticos mostram que os melhores resultados são obtidos modelando-se os comportamentos com polinômios de primeira e segunda ordem, uma vez que quanto maior o polinômio mais recursos computacionais são necessários, e o benefício advindo não compensa sua utilização. Além disso, é difícil obter uma informação precisa dos termos das derivadas de alta ordem de uma determinada entidade simulada, caso ela não apresente um comportamento comportado [SINGHAL99].

Uma variação da técnica de predição foi proposta por Singhal e Cheriton [SINGHAL95] para uso no sistema de realidade virtual distribuída PARADISE [PARADISE], no qual uma abordagem híbrida escolhe dinamicamente a ordem do polinômio de predição a ser utilizado. A diferença está no fato de que apenas um atributo de nível 0 é recebido, sendo a predição feita a partir do histórico armazenado de vários estados anteriores. Existem também outras técnicas de predição especializadas, que levam em consideração determinadas características específicas das entidades sendo visualizadas. Um exemplo é a técnica de predição para manobras de vôo de aeronaves, que leva em consideração restrições quanto a orientação e ângulo de ataque da mesma em relação à sua velocidade e trajetória [KATZ94].

A segunda fase da técnica *dead reckoning* é a convergência. Ela define como proceder quando do recebimento de uma mensagem contendo uma informação atualizada que difere da previsão feita. Um bom algoritmo de convergência permite que o estado seja rapidamente corrigido sem que os usuários percebam distorções visuais.

A forma mais simples de convergência é a de ordem zero, que implica em corrigir imediatamente o atributo sem qualquer preocupação com as possíveis distorções visuais, como ilustrado na Figura 7.

Figura 7 – Convergência de ordem zero.

Em outra técnica, com a qual se obtêm melhores resultados, ao invés de “pular” do estado corrente para o novo estado, uma convergência mais lenta é aplicada do estado corrente para um determinado ponto na trajetória da nova predição. Por exemplo, pode ser utilizada uma convergência linear através do estabelecimento de um ponto de convergência na nova trajetória prevista. Essa trajetória seria seguida por um determinado espaço de tempo até encontrar em algum momento a trajetória prevista, que seria seguida daí em diante até que uma nova mensagem atualizando o estado corrente chegasse. A Figura 8 ilustra esta técnica. O uso desta abordagem permite uma continuidade na trajetória seguida, evitando saltos de um estado para outro, porém não garante a suavidade da trajetória seguida, uma vez que a direção seguida pode sofrer mudanças bruscas.

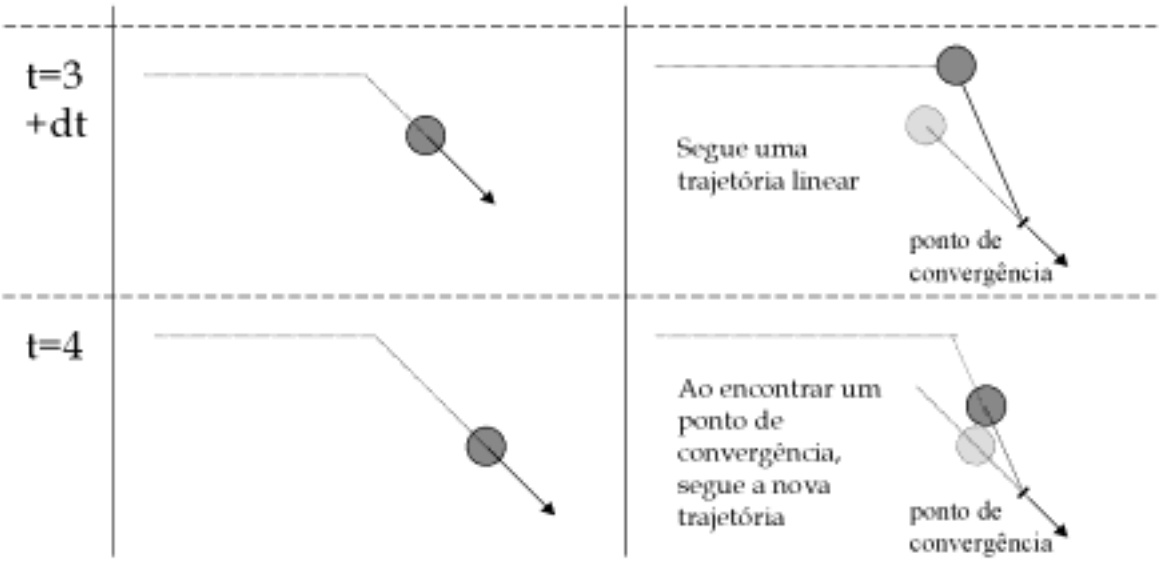

Figura 8 – Convergência linear.

Uma abordagem mais sofisticada implicaria no uso de técnicas de ajustes de curvas, onde uma curva quadrática permitiria casar suavemente a trajetória corrente com a nova trajetória prevista, como ilustrado na Figura 9. Entretanto, o uso de algoritmos complexos para a convergência dos estados de todas as entidades presentes em no ambiente virtual pode acarretar uma degradação no desempenho do sistema, além de, em muitos casos, ser desnecessário para determinadas entidades cuja influência no resultado visual seja pequena.

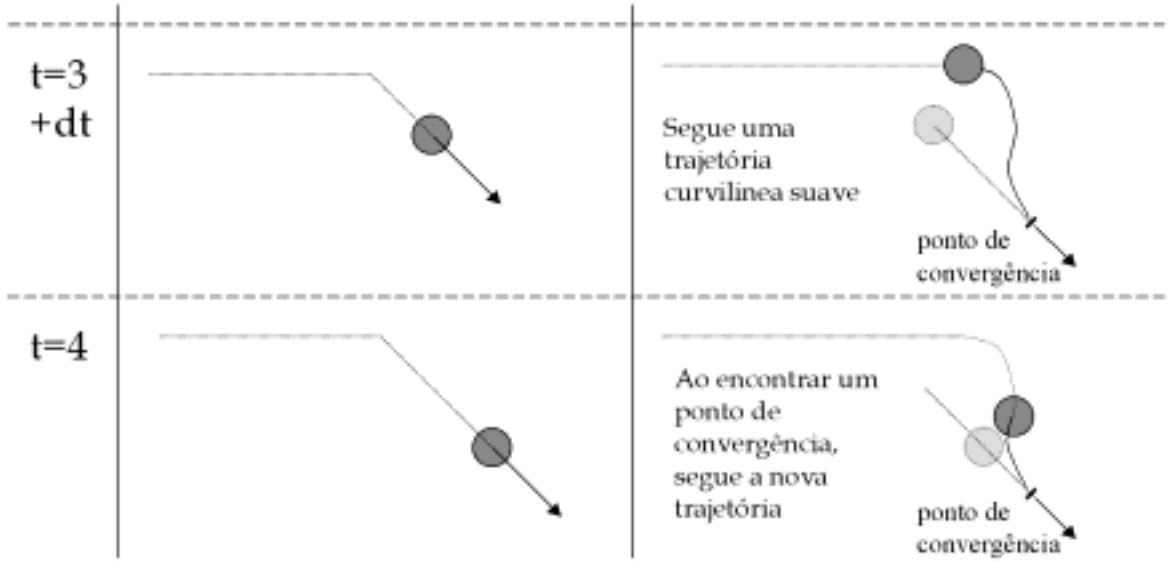

Figura 9 – Convergência por ajuste de curvas.

Resumindo, uma das vantagens do uso da técnica *dead reckoning* é a diminuição do tráfego gerado na rede, com um conseqüente aumento na taxa de quadros por segundo. Porém, para obter proveito de tais vantagens, a integridade do estado dinâmico compartilhado precisa ser quebrada, implicando na introdução de pequenos desvios e erros no estado dinâmico das entidades que estão sendo processadas independentemente em cada unidade de visualização. Conseqüentemente, com o uso desta técnica não é possível garantir a consistência instantânea de todas as réplicas do ambiente virtual.

A combinação das abordagens anteriores

Uma terceira abordagem busca balancear o uso das abordagens anteriores de forma a maximizar as qualidades de cada uma delas. Dessa forma, além das mensagens enviadas quando da mudança no estado de determinada entidade, são enviadas mensagens com certa freqüência atualizando o estado corrente de todas as entidades. Com isso, minimizam-se os problemas decorrentes da possível perda de pacotes e evitam-se grandes desvios na

trajetória. Esta técnica híbrida permite impor limites máximos de inconsistência no ambiente virtual. Todavia, seu uso deve ser moderado, a fim de evitar uma degradação no desempenho total do sistema. Além disso, nos casos em que haja garantia de chegada dos pacotes, tal procedimento pode ser desnecessário.

Como foi visto, a técnica *dead reckoning* é destinada basicamente a lidar com entidades que apresentem comportamentos determinísticos de nível 0 e 1. Comportamentos não-determinísticos podem implicar na necessidade de realizar atualizações freqüentes no estado dinâmico compartilhado, causando não apenas um aumento no tráfego de rede como também implicando em maiores desvios e erros na técnica de predição, aumentando assim as distorções na animação visualizada. Ainda assim, se as ações não-determinísticas tiverem um impacto lento no sistema, elas poderão ser aproximadas por comportamentos determinísticos durante certos intervalos de tempo.

2.3.2 - Sincronismo

Este é um dos requisitos de maior importância para o funcionamento de um sistema de visualização distribuída. Para que as técnicas de visualização distribuída apresentadas na seção anterior funcionem, é necessário que todas as unidades de visualização tenham seus relógios virtuais de simulação sincronizados entre si. Para isso, todas as mensagens contendo atualizações no estado corrente dinâmico, sejam elas determinísticas de nível 0 ou 1, contêm um campo onde é informado o momento exato em que a alteração ocorreu. A utilização deste campo permite que as alterações sejam feitas em relação ao momento correto, garantindo a consistência da simulação, independente dos atrasos na transmissão.

Considerando, por exemplo, que a simulação tenha alterado a direção e a velocidade de determinada entidade no tempo t_0, seriam enviadas mensagens para todas as máquinas integrantes do sistema de visualização. Certamente essas mensagens não chegarão a seus destinos no mesmo momento. Para uma determinada unidade de visualização, considerando que ela tenha recebido a atualização no momento t_1, usando a técnica de *dead reckoning* ela é capaz de convergir para o novo estado corrente da entidade sabendo que o mesmo já variou de $t_1 - t_0$ unidades de tempo referentes à latência da rede. Se o mesmo procedimento for adotado por todas as unidades de visualização, após a última atualização, realizada no momento t_n, pode-se garantir, assumindo que os relógios das entidades estejam sincronizados, que a consistência do ambiente virtual foi mantida, uma vez que cada unidade de visualização se atualizou, compensando seu respectivo atraso.

A precisão de sincronismo necessária varia em função da taxa de quadros por segundo desejada na visualização. Dessa forma, para uma taxa de quadros por segundo de 30 *fps*, a variação máxima da precisão nos relógios é da ordem de 33ms. Existem diversos fatores que podem influenciar na precisão dos relógios, tais como: o mecanismo de sincronização utilizado, a heterogeneidade do *hardware* (o cristal dos relógios internos das máquinas, uma vez que, mesmo sincronizados, eles podem divergir com o passar do tempo), o grau de dispersão das máquinas utilizadas (a dispersão física pode implicar em pequenas diferenças na freqüência da corrente alternada que alimenta as máquinas), entre outros. Caso não possa ser alcançado um sincronismo da ordem desejada entre as máquinas que compõem a visualização, isso não implica na impossibilidade de utilização do sistema. Em termos práticos, a velocidade com que o estado dinâmico das entidades varia também é um fator a ser considerado no cálculo do sincronismo necessário, uma vez que, para pequenas alterações no estado dinâmico, uma disparidade de cerca de 2 a 3 quadros entre as unidades de visualização pode não influir no resultado visual final.

Para atender aos requisitos acima expostos podem ser adotadas soluções baseadas no uso de protocolos de sincronização de computadores ligados em rede. Além disso, pode ser necessário que o processo de sincronização das unidades de visualização com o relógio da simulação seja executado periodicamente. Tal procedimento visa minimizar os efeitos decorrentes do fato de que os relógios das máquinas podem perder o sincronismo com o passar do tempo. Porém, não é possível estimar antecipadamente qual a freqüência com que isso deve ocorrer, uma vez que um grande número de variáveis pode interferir na precisão do relógio de uma máquina. Fatores como freqüência da rede elétrica, temperatura, umidade do ar, qualidade da fonte de energia do computador, entre outros, podem acarretar imprecisões da ordem de 1 segundo por dia, fazendo com que em casos extremos seja necessário realizar o re-sincronismo do sistema a cada 2 minutos.

Resumindo, o uso de protocolos de sincronização entre as máquinas em conjunto com as técnicas de manutenção de consistência representa a base para o desenvolvimento de um sistema de visualização distribuída.

2.3.3 - Heterogeneidade

Geralmente, os sistemas distribuídos são implantados utilizando um conjunto heterogêneo de equipamentos. Para tal, a arquitetura deve prover recursos a fim de suportar

as diferenças entre os diversos equipamentos utilizados, de forma a garantir a qualidade e a consistência da visualização gerada.

A heterogeneidade pode advir do uso de computadores dotados de diferentes capacidades de processamento, tanto numérico quanto gráfico, além de unidades de visualização com diferentes resoluções, tamanhos, entre outros.

A fim de suportar tais diferenças, um sistema pode adotar duas condutas distintas. A primeira é nivelar o uso dos recursos com relação aos equipamentos de menor capacidade presentes no sistema. Dessa forma, a heterogeneidade poderia ser mascarada pela redução no desempenho de todo o sistema a um mínimo denominador comum. A vantagem dessa abordagem está no fato de que não haveria grandes distorções caso uma unidade de visualização de alto desempenho fosse colocada ao lado de outra de menor capacidade. A desvantagem, no entanto, é que a simples inclusão de uma máquina mais fraca implicaria na degradação de todo o sistema.

A segunda alternativa seria utilizar ao máximo todos os recursos disponíveis em cada unidade de visualização. A escolha desta conduta implica em diferentes níveis de resultado visual apresentado nas diferentes unidades de visualização. Tal fato pode interferir na qualidade da visualização, uma vez que pode desviar a atenção do usuário e quebrar o processo de imersão.

Caso se adote a segunda conduta, faz-se necessário que o sistema seja configurável no que diz respeito a diversos fatores. Ele deve, por exemplo, dar suporte a dispositivos de visualização de diversos formatos, tamanhos e resoluções, e a otimizações que permitam o uso de diferentes LODs, a fim de que as diferenças possam ser equalizadas. Além disso, deve assegurar que a cena renderizada a cada momento em cada unidade de visualização será a mesma. Tal fato é melhor ilustrado na Figura 10, em que, por exemplo, duas unidades de visualização distintas estão mostrando o mesmo cenário virtual. Neste caso, independentemente da taxa de quadros por segundo de cada unidade, em um determinado momento *t*, caso ambas as unidades estejam aptas a renderizar uma imagem, esta deveria ser a mesma. A unidade mais rápida poderia continuar renderizando outras imagens referentes a momentos *t+1*, *t+2*, …, enquanto a unidade mais lenta estaria, em paralelo, renderizando a imagem referente ao tempo *t*.

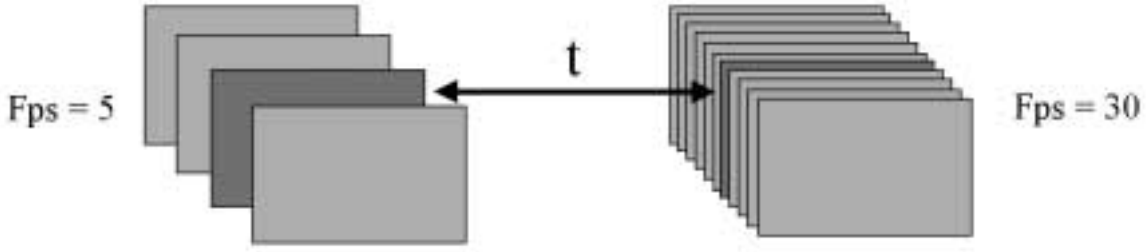

Figura 10 – Quadros renderizados em unidades de visualização heterogêneas

Como é exposto no próximo capítulo, a arquitetura proposta adota a segunda alternativa, por considerar que nivelar um sistema pelo mínimo denominador comum constitui um fator limitante prejudicial que restringe a sua escalabilidade.

2.3.4 - Escalabilidade

Este requisito constitui um desafio a ser resolvido em um sistema de visualização distribuída. Isto se deve ao fato de que, seja qual for o sistema, ele sempre contará com recursos limitados. A quantidade de memória de cada máquina, o poder de processamento numérico/gráfico e a velocidade da rede constituem alguns exemplos. Além disso, a curva que descreve a relação custo vs. incremento de recursos geralmente não é linear.

Em geral, é o equilíbrio entre dois parâmetros que delimita a escalabilidade do sistema. São eles: o **número de entidades presentes no ambiente virtual** e o **número de unidades de visualização utilizadas**. O aumento de um pode implicar na necessidade de diminuição do outro, e vice-versa.

Os fatores que limitam a escalabilidade, sejam eles no processamento ou na utilização da rede, podem ser originados em diversas condições:

- a presença de muitas entidades no ambiente virtual pode implicar em uma elevada utilização de tempo de processamento dedicado à atualização e predição dos comportamentos;
- o uso de geometria altamente detalhada também pode implicar em uma elevada utilização de tempo de processamento gráfico necessário para renderizar uma entidade dentro de uma cena;
- o uso excessivo de texturas pode comprometer o desempenho quando a memória destinada a elas for insuficiente, acarretando acessos indesejados à memória secundária;

- a existência de entidades com comportamentos não-determinísticos ou pouco comportados pode implicar na necessidade de atualizações excessivas, o que pode gerar tráfego excessivo na rede;
- o compartilhamento da rede com outras máquinas que não estejam participando da simulação pode degradar o sistema;
- a presença de roteadores e *switches* na interligação entre as redes conectadas às máquinas pode exigir a utilização de algoritmos mais complexos para o sincronismo e implicar em maiores retardos na rede.

Conseqüentemente, condicionantes desse tipo devem ser considerados nas fases de implementação e implantação da arquitetura proposta para o desenvolvimento de um sistema de visualização distribuída.

2.3.5 - Portabilidade

Esta característica não pode ser considerada propriamente um requisito, porém é recomendável. Ela consiste na possibilidade do sistema ser implantado e facilmente portado para diferentes plataformas de *hardware* e *software*. A principal indicação de portabilidade de um sistema é a possibilidade de que ele seja facilmente compilado, implantado e executado sobre outros sistemas operacionais. Todavia, outras características podem ser consideradas determinantes de portabilidade. Fatores como facilidade de instalação e configuração, amplo suporte a acionadores de dispositivos e a habilidade de abrir e converter uma variedade de formatos de entrada também atribuem uma maior ou menor portabilidade ao sistema.

CAPÍTULO 3 – ARQUITETURA PROPOSTA

Apresentar a arquitetura proposta, identificando as soluções tecnológicas adotadas, bem como apontar as motivações e implicações de seu uso. Descrever sua estruturação, apontando as dependências entre os módulos e a forma com que os dados fluem entre eles.

3.1 - Visão Geral

A arquitetura apresentada a seguir tem como objetivo servir de arcabouço para o desenvolvimento de um sistema de visualização distribuída. A Figura 11 a seguir apresenta uma visão geral da arquitetura.

A arquitetura do sistema é composta, basicamente, por cinco componentes**:** a **infra-estrutura de comunicação**, o **mundo externo**, o **mediador**, o **núcleo do sistema de visualização** e a **modelagem do ambiente virtual**.

Em linhas gerais, o mundo externo é qualquer sistema ou aplicação responsável pela simulação do ambiente virtual. O mediador funciona como interface entre o mundo externo e o núcleo do sistema, que é composto por um conjunto de unidades de visualização. Estas são responsáveis por renderizar cada porção da cena. A modelagem do ambiente virtual é feita através do uso de uma linguagem interpretada através da qual a estrutura da cena e o conjunto de atributos que a governam são declarados. A comunicação entre o mundo externo e o mediador, bem como a comunicação entre o mediador e o núcleo do sistema de visualização, é feita através da infra-estrutura de comunicação.

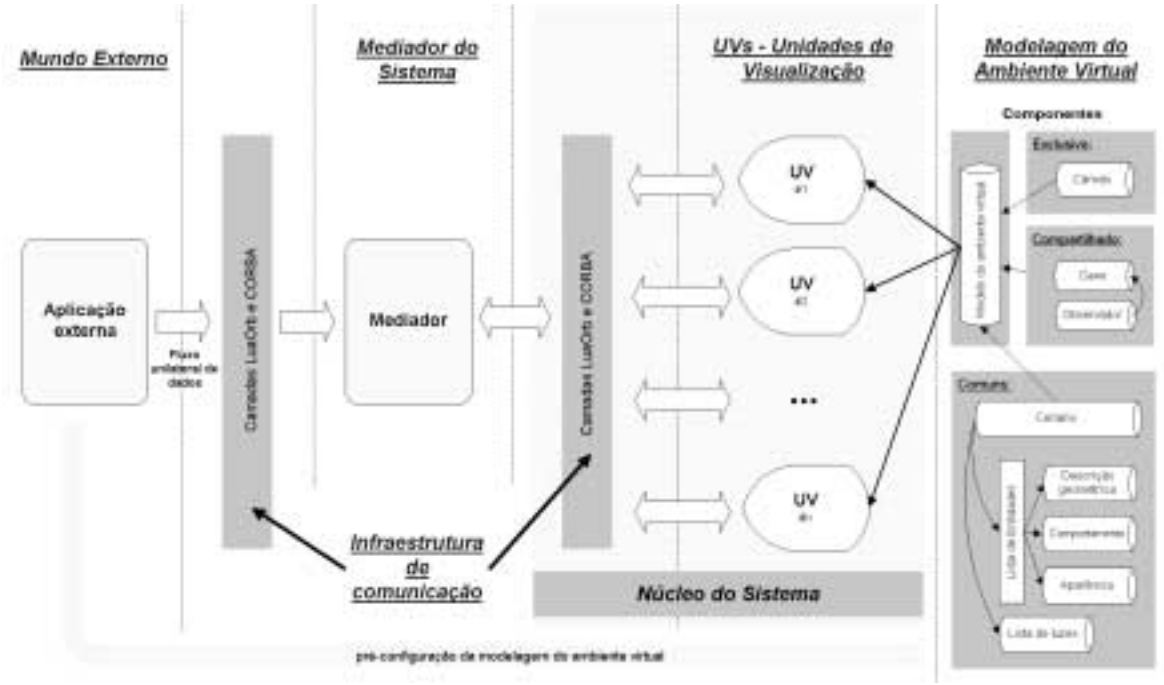

Figura 11 – Visão geral da arquitetura do sistema

O fluxo de dados segue das extremidades para o centro, representado pelo núcleo do sistema de visualização. O estado corrente do ambiente virtual, mantido pelo mundo externo, é enviado para o mediador, que, em seguida, o distribui para as unidades de visualização, que são inicialmente alimentadas com o estado inicial pré-modelado de acordo com a natureza do ambiente virtual a ser visualizado.

3.2 - Infra-estrutura de comunicação

No capítulo anterior foram apresentados diversos fatores relacionados com o problema da visualização distribuída. O processo de distribuição é geralmente implantado através da utilização de uma infra-estrutura de comunicação que deve apresentar características de confiabilidade e bom desempenho.

A seguir são indicados alguns conceitos fundamentais do funcionamento das redes de computadores e sua influência na determinação dos parâmetros da arquitetura distribuída proposta [SINGHAL99].

3.2.1 - Conceitos fundamentais

A **latência de rede** representa a quantidade de tempo que um pacote de informação leva para chegar da origem ao seu destino. Ela é decorrente de diversos fatores, dentre eles a limitação da velocidade com que a informação percorre os meios físicos (a presença de

roteadores, *hubs* e *switches* fazendo a ligação entre redes) e o tempo gasto com o empacotamento e desempacotamento da mensagem. Existem dois fatores importantes com relação à latência de rede: a sua duração (*lag*) e a variação de sua duração (*jitter*). Altos valores de *lag* e altas taxas de *jitter* podem comprometer a integridade do estado dinâmico corrente visualizado, problema cuja solução foi abordada no capítulo anterior.

A **largura de banda** representa a taxa limite de informações que a rede permite levar da origem ao destino. O limite da largura de banda é determinado pelo meio (rádio, fibra ótica, cabo coaxial, etc.) utilizado para a transmissão de dados e pelo *hardware* utilizado para a interface entre os diferentes meios e a origem/destino dos dados. A largura de banda é um dos fatores limitantes da escalabilidade de um sistema de visualização distribuída, uma vez que quanto maior for o número de entidades e unidades de visualização maior será o tráfego gerado. O impacto dessa limitação é maior em WANs (*Wide Area Network*) do que em LANs (*Local Area Network*), uma vez que as LANs podem atingir taxas de até 1 *Gbps*, enquanto as WANs estão geralmente limitadas a taxas de até 1.5 *Mbps*.

O **esquema de distribuição** representa a forma com que uma mensagem é propagada entre diversas unidades da rede. Ele pode ser de três tipos: *unicast*, *broadcast* e *multicast*, ilustrados na Figura 12. No esquema *unicast*, também conhecido como ponto-a-ponto, a comunicação é feita através do envio de um pacote de mensagem diretamente da origem para o destino. No esquema *broadcast*, possível apenas em redes locais, um pacote é propagado pela rede e recebido/processado por todas as unidades nela presentes, mesmo que elas não estejam participando da comunicação ou interessadas na mensagem. No esquema *multicast*, aplicado a WANs, um pacote de mensagem é progressivamente propagado da origem, através de roteadores, para as unidades registradas que desejam receber o pacote. A desvantagem do uso do esquema *multicast* é o fato de que não existem atualmente implementações eficientes que apresentem garantia de chegada da mensagem ao seu destino.

Com relação à visualização distribuída, a escolha do melhor esquema de distribuição depende de alguns fatores. Caso todas as unidades de visualização estejam localizadas na mesma rede local dedicada, a escolha natural é o esquema *broadcast*. Caso a rede local seja compartilhada com outras máquinas que não estão participando da visualização, a melhor escolha é o esquema *unicast*. E, caso as unidades de visualização estejam dispersas em uma WAN ou até mesmo na Internet, o melhor esquema é o *multicast*. Vale notar que

existem ainda algumas restrições tecnológicas relacionadas à necessidade de suporte por parte dos roteadores a protocolos de rede que implementem este esquema de distribuição.

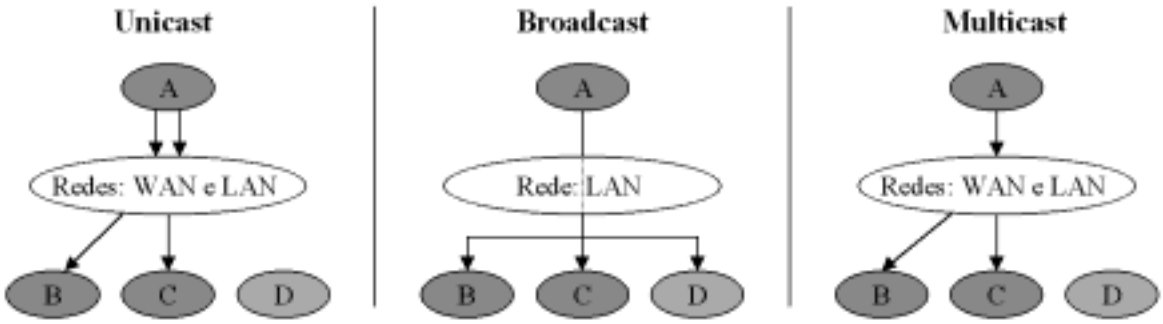

Figura 12 – Esquemas de distribuição no envio de um pacote da unidade A para B e C

A **confiabilidade da rede**, segundo Singhal [SINGHAL99], é baseada na medida de quanta informação é perdida durante o encaminhamento de um pacote de mensagem da origem ao seu destino. A perda de informações pode ser de dois tipos: a perda de um pacote, quando este simplesmente não chega ao seu destino, possivelmente por ter sido descartado pela rede, ou o corrompimento do pacote, quando o seu conteúdo, ao chegar no destino, difere do conteúdo original. A forma mais comum de perda de dados é a perda de pacotes, geralmente ocasionada por sobrecarga na capacidade de processamento dos roteadores. Isso ocorre, geralmente, quando muitos pacotes chegam ao mesmo tempo e não há memória suficiente na fila de entrada de pacotes. Em relação à quantidade de pacotes corrompidos, ela se torna considerável quando são usados meios de transmissão sem fio, uma vez que estes estão mais sujeitos à interferência externa.

Existem diversas técnicas para controlar a confiabilidade da rede. Contudo, qualquer que seja a técnica utilizada, ela implica em um aumento na quantidade de dados enviados entre a origem e o destino, o que acarreta uma maior demanda por largura de banda da rede.

Com relação à visualização distribuída, a fim de minimizar erros de integridade do estado corrente dinâmico, é aconselhável que a confiabilidade da rede seja alta, se possível através do uso de protocolos de rede que garantam o recebimento de todos os pacotes enviados.

O **protocolo de rede** consiste no conjunto de regras que duas aplicações localizadas em unidades diferentes adotam para comunicar-se entre si. Suas componentes básicas são: o formato do pacote, a semântica do pacote e a semântica de tratamento de erros.

O formato do pacote consiste na estrutura como os *bits* da mensagem estão organizados. Ele indica as regras para sua leitura e interpretação. A semântica é geralmente descrita como uma máquina de estados finita que representa o conjunto de transições e operações realizadas pelo protocolo. Como exemplo, a semântica adotada quando da abertura de um canal de comunicação pode ser: **A→B**: *OLA B,* **B→A**: *OLA A*. O tratamento de erros, um tipo especial de semântica, é o conjunto de regras que determina como cada extremidade da comunicação deve se comportar caso algum erro ocorra.

Existem diversas implementações de protocolos sendo utilizadas atualmente. Muitos protocolos são especializados para o cumprimento de determinadas tarefas específicas, como a difusão de áudio e vídeo em tempo real, a troca de informações de configuração entre roteadores ou a simples transferência de arquivos.

3.2.2 - Solução proposta

Os requisitos delineados nos capítulos anteriores implicam em restrições quanto à definição da melhor infra-estrutura de comunicação para o sistema de visualização em questão.

Uma abordagem conservadora adotaria uma interface de programação de aplicações (*API – Application Programming Interface*) baseada em *sockets* [STEVENS90] devido à sua alta portabilidade e interoperabilidade em um ambiente TCP/IP [POSTEL81a] [POSTEL81b]. Entretanto, esta API é baseada em um paradigma de programação com baixo nível de abstração.

O fator confiabilidade elimina, naturalmente, uma abordagem baseada no protocolo UDP [POSTEL80], uma vez que este não fornece garantias de que os dados enviados cheguem ao seu destino.

Uma vez que a arquitetura proposta indica a adoção de uma abordagem de programação orientada a objetos, é interessante que a infra-estrutura de comunicação também forneça melhor suporte à programação orientada a objetos.

A opção que se apresenta como a mais adequada é a utilização do padrão CORBA [OMG98][SIEGEL96]. Este padrão é a especificação de uma arquitetura que suporta aplicações distribuídas e que atende a todos os requisitos anteriormente apresentados. CORBA especifica interfaces de programação independentes de plataforma, linguagem e sistema operacional. Sua infra-estrutura aberta de objetos distribuídos é padronizada pela

OMG (*Object Management Group*) [OMG]. Tais características implicam em portabilidade.

Além disso, por trabalhar em um nível de abstração mais alto, CORBA torna transparentes os mecanismos de comunicação que estão sendo utilizados. As diversas implementações baseadas em CORBA dão suporte à execução sobre diversos protocolos e esquemas de distribuição, além de fornecer mecanismos de garantia da confiabilidade. Com isso, o desenvolvedor de aplicações distribuídas pode concentrar-se mais em outras tarefas mais importantes no contexto da aplicação do que em questões ligadas à programação em baixo nível da infra-estrutura de comunicação.

CORBA é fortemente baseado no conceito de interfaces de objetos. Uma interface de objeto é o conjunto de operações que um objeto provê. A descrição dessas interfaces de objetos é feita através de uma linguagem de definição da interface (IDL – *Interface Definition Language*). Todavia, a IDL não é uma linguagem de programação, o que implica que aplicações e objetos não podem ser implementados com ela. Seu único propósito é definir a interface de um objeto, independente da linguagem de programação na qual o objeto é implementado ou utilizado. Tal mecanismo permite que objetos de aplicações desenvolvidas em diferentes linguagens, rodando em máquinas diferentes, possam operar entre si como se estivessem operando localmente na mesma aplicação. Essa característica estabelece uma transparência na localização dos objetos que constituem as aplicações distribuídas.

A partir de uma definição em IDL, um compilador de IDL gera *stubs* que as aplicações clientes usam para acessar os objetos remotos. O *stub* gerado pode estar em uma linguagem de programação diferente da usada para implementar o objeto servidor. O *stub* cliente se comporta como um *proxy* de um objeto remoto, uma vez que delega para o objeto remoto qualquer operação realizada sobre ele. A Figura 13 ilustra a interação entre uma aplicação cliente e um objeto servidor.

A fim de simplificar ainda mais o acesso à infra-estrutura de comunicação, a arquitetura proposta prevê outra camada de abstração sobre CORBA. LuaOrb é uma interface entre a linguagem de extensão Lua e CORBA [CERQUEIRA99]. Ela possibilita que um programa escrito em Lua manipule objetos CORBA da mesma forma que manipula objetos locais. Isso une as facilidades de reuso de CORBA com a flexibilidade de uma linguagem interpretada. Utilizando-se LuaOrb, quando se deseja acessar um método

em um objeto CORBA, basta que a chamada seja escrita usando a sintaxe regular para chamadas de métodos em Lua. A interface LuaOrb intercepta a chamada, consulta a assinatura do método no repositório de interfaces, mapea dinamicamente os parâmetros de Lua para IDL, realiza efetivamente a chamada do método e mapea os resultados de volta para Lua.

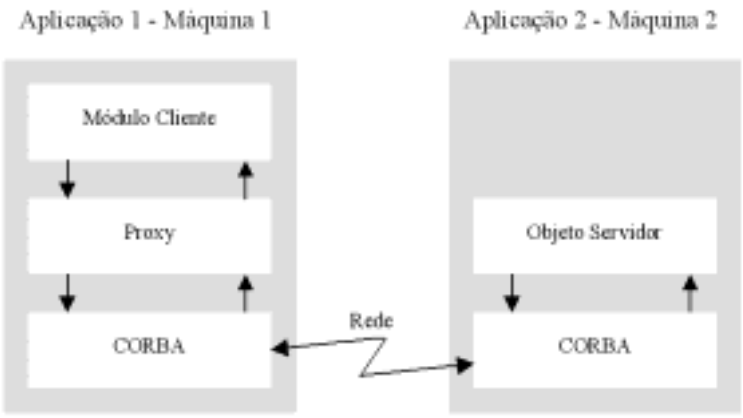

Figura 13 – Interação cliente-servidor entre objetos

Inicialmente, a decisão por utilizar a ferramenta LuaOrb se restringia à fase de elaboração do protótipo do sistema, devido às facilidades que a ferramenta oferece para esta etapa do desenvolvimento [CERQUEIRA97]. Posteriormente, as partes do sistema implementadas usando LuaOrb poderiam ser substituídas por *stubs* convencionais, para o qual se estimava obter um melhor desempenho. Entretanto, como o uso de LuaOrb demonstrou não comprometer o desempenho do sistema, optou-se por mantê-la e continuar tirando proveito de algumas facilidades oferecidas pela ferramenta, tais como uma maior flexibilidade para a evolução do sistema e uma separação explícita de todos os aspectos referentes ao suporte à distribuição do restante da aplicação.

A última decisão a ser tomada a respeito da infra-estrutura de comunicação é com relação a qual mecanismo utilizar para sincronizar os relógios de todas as unidades do sistema de visualização distribuída.

Uma solução é o uso do protocolo SNTP (*Simple Network Time Protocol*) [MILLS96], que é uma adaptação do protocolo NTP (*Network Time Protocol*) [MILLS92], usado para sincronizar o relógio de computadores ligados pela Internet.

O NTP é um sofisticado protocolo desenvolvido para sincronizar computadores ligados através de WANs e LANs, alcançando geralmente uma precisão cuja variação é da ordem de alguns milisegundos. Já o protocolo SNTP, uma simplificação do NTP, pode ser

usado quando todos os computadores estão na mesma rede local, alcançando também uma precisão cuja variação é da ordem de 1 a 10ms. Dessa forma, dependendo do ambiente no qual o sistema for implantado, WAN ou LAN, deve-se adotar uma solução baseada nos protocolos NTP ou SNTP, respectivamente.

A Figura 14 apresenta um esquema que resume o funcionamento da infra-estrutura de comunicação. Os quadros superiores representam duas aplicações distintas sendo executadas em máquinas diferentes. Cada objeto da aplicação 1, por exemplo, pode acessar e ser acessado, por meio de *proxys*, os objetos da aplicação 2. As diversas camadas, LuaOrb, CORBA, e os protocolos de nível mais baixo são responsáveis pelo mapeamento, pela identificação, pelo transporte correto dos dados e pelas chamadas de métodos entre a origem e o destino.

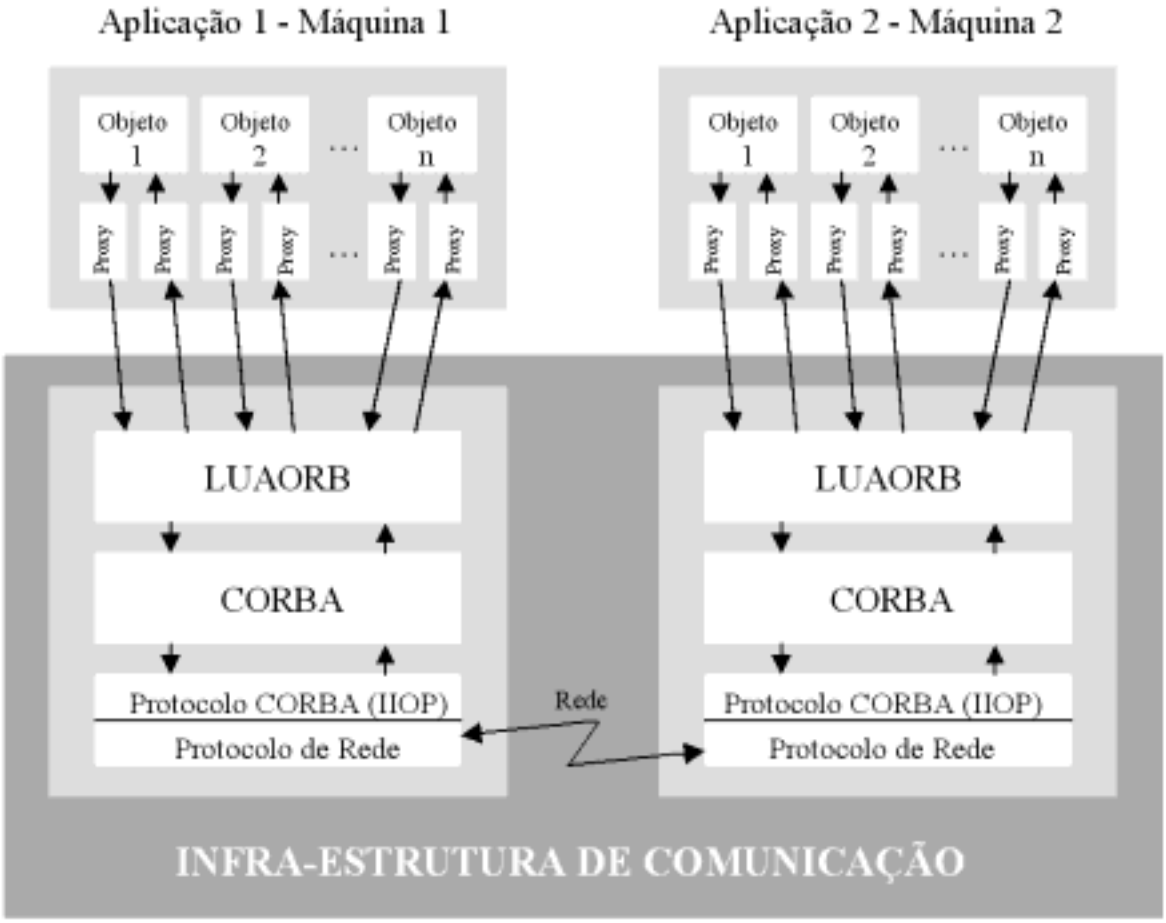

Figura 14 – Funcionamento da infra-estrutura de comunicação

3.3 - Mundo Externo

No contexto deste trabalho, o mundo externo é definido como a aplicação ou o conjunto de aplicações que mantêm o estado dinâmico do ambiente virtual e alimentam o processo de visualização distribuída.

A arquitetura não possui exigências a respeito da estrutura e do funcionamento do mundo externo. Ela apenas sugere um formato de interface para a sua ligação com o mediador do sistema.

O mundo externo pode ser constituído por um sistema de qualquer natureza, desde que este apresente como conteúdo de saída, dados que, ao serem processados, venham a estabelecer o estado corrente de algum ambiente virtual de interesse para a visualização. Dessa forma, a natureza do mundo externo pode assumir diversos papéis:

- interface de entrada direta de dados do usuário do sistema. Ex.: o mundo externo recebe comandos para a atualização da posição do observador em relação a um terreno e, possivelmente, retorna dados de posicionamento geográfico e orientação;
- interface de saída direta de dados possivelmente não visuais do sistema para o usuário. Ex.: o mundo externo gera um relatório contendo o roteiro das ações e modificações ocorridas com todas as entidades do ambiente virtual;
- roteiro de uma seqüência de animação realizada sobre um conjunto de entidades. Ex.: uma animação que reproduza a progressão do posicionamento e da movimentação de um grupo de pessoas e de veículos em um teatro de operações;
- interface de entrada de dados de diversos usuários do sistema em trabalho cooperativo. Ex.: o mundo externo recebe comandos para a atualização do estado corrente de determinadas entidades, que são enviados por usuários distintos observando o ambiente virtual a partir de diferentes pontos de vista;
- simulação. Ex.: o mundo externo é responsável por simular a trajetória de determinadas entidades sujeitas a variações nas suas condições de funcionamento e à ação de forças externas;
- sistema de monitoramento. Ex.: o mundo externo é composto por um sistema que monitora o estado de determinadas entidades do mundo real;

- sistema especializado que combina os papéis descritos acima, ou outros que porventura produzam uma saída adequada ao problema de visualização.

É interessante que o mundo externo se comunique com o mediador por meio da infra-estrutura de comunicação apresentada na seção anterior. Isto é, cada entidade (objeto) mantida pelo mundo externo faz acesso a métodos de entidades (objetos) equivalentes no mediador do sistema. Isso é indicado a fim de possibilitar um maior reuso das componentes do mediador.

Outro aspecto importante do mundo externo é o seu papel na definição de como o ambiente virtual será modelado. Isto é, em função da natureza do mundo externo, é necessário configurar as entidades que são inseridas no sistema de visualização. Nesta fase, é necessário escolher as descrições geométricas e as aparências adequadas, bem como o estado comportamental inicial de cada entidade. A seção 3.6 especifica a forma como essa modelagem inicial é feita.

3.4 - Mediador

Na arquitetura, o mediador é uma aplicação cuja função principal é a coordenação e o controle das unidades de visualização e seu acoplamento com o mundo externo. Essa aplicação é dividida em duas componentes principais: o **módulo de coordenação** e o **módulo de acoplamento**.

Uma abordagem mais simples para o problema de visualização seria a ligação direta do mundo externo com as unidades de visualização, eliminando a presença do mediador. Todavia, a utilização do mediador permite o desacoplamento das unidades de visualização do mundo externo, aumentando desta forma a capacidade de reuso do sistema de visualização.

Caso um novo ambiente virtual venha a ser visualizado pelo sistema, naturalmente faz-se necessária, em primeiro lugar, a mudança do mundo externo e a pré-modelagem do ambiente virtual. Em segundo lugar, são necessárias modificações no mediador que reflitam a nova interface de saída fornecida pelo mundo externo. Contudo, não são necessárias modificações no núcleo do sistema de visualização, que permanece inalterado. Isso só é possível graças à utilização do mediador.

A rotina de funcionamento da arquitetura distribuída prevê que o fluxo dos dados siga basicamente do mundo externo para o mediador e deste para as unidades de

visualização. Esse fluxo é unilateral do mundo externo para o mediador e bilateral do mediador para as unidades de visualização. A opção por fazer o fluxo de dados unilateral do mundo externo para o mediador é motivada pelo fato de que qualquer processo de visualização constitui exclusivamente um processo de saída de dados. Qualquer comunicação no sentido oposto pode ser considerada como entrada de dados, elemento não tratado diretamente pela arquitetura proposta. A comunicação bilateral entre o mediador e as unidades de visualização é explicada a seguir.

A implementação do mediador é baseada nos padrões adaptador e mediador propostos como modelo de desenvolvimento orientado a objetos [GAMMA95]. Estes padrões aumentam a modularidade e facilitam o reuso do sistema.

A Figura 15 ilustra a estrutura geral do mediador e do fluxo de dados entre suas componentes.

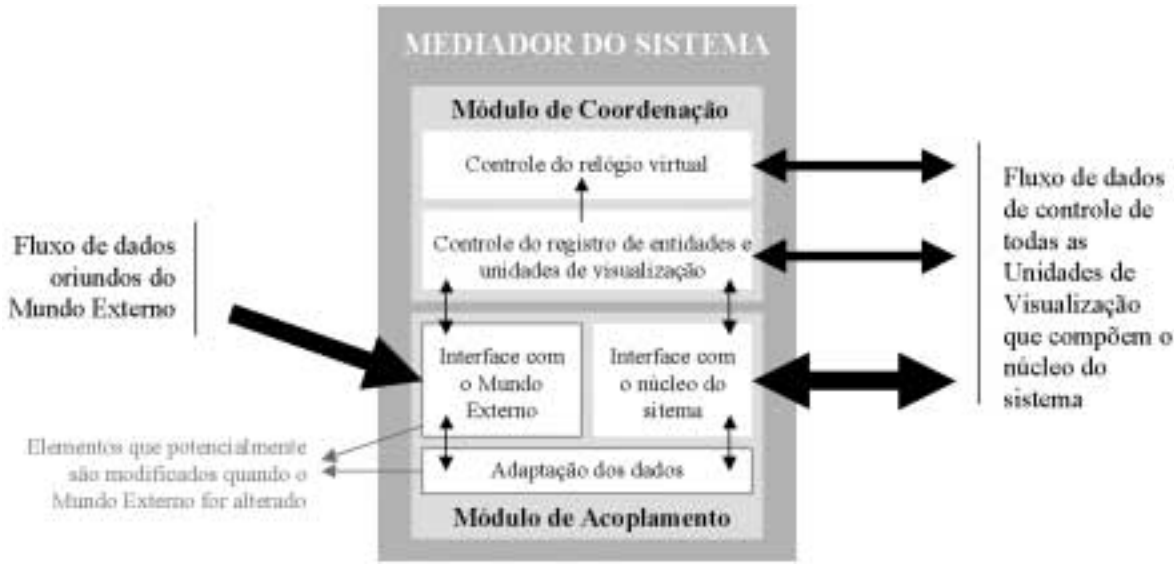

Figura 15 – Visão geral do mediador do sistema

3.4.1 - Módulo de coordenação

O módulo de coordenação do mediador é responsável pela manutenção de registros das unidades de visualização. Além disso, ele mantém registros de todas as entidades presentes no ambiente virtual. Essas informações são utilizadas para, quando chegar uma alteração de estado, serem distribuídas por todas as unidades de visualização.

O coordenador também é responsável por manter o relógio de simulação do ambiente virtual. Dessa forma, cada nova unidade de visualização, ao se registrar, tem seu relógio local sincronizado com o relógio de uma unidade de referência do sistema.

O processo de registro das unidades de visualização e das entidades presentes no ambiente virtual, além do processo de atualização dos relógios virtuais de simulação, justificam a necessidade de um canal de comunicação bilateral entre o núcleo do sistema de visualização e o mediador. A interface de comunicação entre este módulo e o núcleo do sistema de visualização é feita por intermédio da infra-estrutura de comunicação descrita na seção 3.2.

3.4.2 - Módulo de acoplamento

O módulo de acoplamento é responsável pela adaptação ou tradução de todas as mensagens relevantes oriundas do mundo externo para as unidades de visualização.

Sua interface com o mundo externo deve, preferencialmente, ser feita por intermédio da infra-estrutura de comunicação estabelecida na seção 3.2. No entanto isso não constitui uma limitação da arquitetura, uma vez que pode haver a necessidade de acoplamento do sistema de visualização com uma aplicação (mundo externo) previamente desenvolvida seguindo paradigmas diferentes. Neste caso, basta que a porção do módulo de acoplamento que faz a interface com o mundo externo seja modificada de forma a “conversar” com a aplicação externa, segundo algum modelo específico.

A outra interface do módulo de acoplamento é responsável pela comunicação com o núcleo do sistema de visualização. Ela deve seguir obrigatoriamente a infra-estrutura de comunicação estabelecida na seção 3.2, uma vez que essa porção do código não necessita ser modificada quando da alteração do mundo externo.

Esta interface é implementada como um conjunto de objetos mapeados sobre as entidades do ambiente virtual. Esses objetos são dotados de métodos para a modificação dos atributos das entidades. Todavia, em função da infra-estrutura de comunicação adotada, eles são, na realidade, apenas *proxies* dos objetos reais existentes em cada unidade de visualização.

Além das interfaces com os elementos externos, o papel principal do módulo de acoplamento é adaptar os dados e comandos oriundos do mundo externo para um formato compatível com a modelagem do ambiente virtual.

Pode-se imaginar um exemplo simples, em que a métrica utilizada pelo sistema de visualização seja diferente da métrica utilizada no mundo externo. Isto é, de um lado os atributos de velocidade e distância são medidos em nós e milhas, enquanto do outro lado os

atributos são medidos em metros/seg e metros. Cabe ao módulo de acoplamento fazer a tradução dos dados. Podem ainda ocorrer situações mais complexas, em que um comando simples oriundo do mundo externo é traduzido em diversos comandos a serem executados pelas unidades de visualização.

3.5 - Unidade de Visualização

As unidades de visualização (UV) são as aplicações executadas nos diversos computadores ligados em rede, que controlam, cada qual, os dispositivos de visualização a elas conectados. O objetivo de cada UV é renderizar individualmente, a partir de determinado ponto de vista, uma imagem do estado dinâmico corrente do ambiente virtual e exibi-la nos respectivos dispositivos de visualização.

A arquitetura do sistema é concebida de forma a não haver alguma comunicação entre as UVs. Isso significa que elas rodam independentemente, sem que haja qualquer troca de informações entre elas. A troca de informações é feita unicamente com o mediador do sistema.

Conforme mencionado anteriormente, o mediador é responsável pelo estabelecimento do sincronismo dos relógios das UVs. Isso é feito durante a inicialização do sistema. Durante o funcionamento do sistema, a manutenção do seu sincronismo é baseada na hipótese de que os relógios individuais de cada computador são suficientemente precisos. Essa consideração é razoável por períodos de tempo limitados. Em situações nas quais o sistema venha a ser utilizado por longos períodos de tempo, é facultada ao mediador a execução freqüente do processo de re-sincronismo dos relógios.

A Figura 16 ilustra o fluxograma de funcionamento de uma UV, que é basicamente dividido nas seguintes fases: **inicialização do sistema**, **recebimento de alterações de estado provenientes da rede**, **processamento do estado corrente e geração de um novo quadro**.

Com relação à implementação da UV, esta arquitetura propõe que ela seja desenvolvida seguindo um paradigma de orientação a objetos, uma vez que suas componentes (entidades, aparências, comportamentos, etc.) estão intimamente ligados com o conceito de objetos.

Figura 16 – Fluxograma de funcionamento da UV

3.5.1 - Inicialização

É nesta fase que todas as UVs são configuradas a fim de estarem em condições de serem controladas pelo mediador do sistema. Inicialmente é criada uma interface de comunicação, que em seguida é utilizada para realizar o registro no mediador do sistema. A unicidade do registro é baseada em um identificador construído a partir de parâmetros exclusivos da modelagem do ambiente virtual. A explicação destes e de outros parâmetros pode ser encontrada na seção 4.1.

Em seguida, é criado um objeto relógio. Seus métodos de configuração são exportados para o mediador através da infra-estrutura de comunicação, permitindo que a hora local seja estabelecida. É interessante que este objeto forneça meios de configuração do tempo, com recursos de aceleração, salto para um determinado momento no futuro e no passado, entre outros.

Em seguida, com base em parâmetros exclusivos de configuração de cada UV, é criada uma superfície de desenho (*canvas*), sobre a qual será renderizada a cena. Essa superfície de desenho deve ser preferencialmente constituída sobre uma interface de programação de aplicações gráficas (API – *Application Programming Interface*). A escolha da API a ser utilizada deve levar em consideração fatores como suporte de *hardware* para aceleração gráfica, portabilidade e estabilidade. Naturalmente, a API gráfica que apresenta todas essas características atualmente é o OpenGL [SEGAL97], sendo esta a API gráfica proposta nesta arquitetura.

A superfície de visualização é criada levando em consideração suas dimensões, a resolução utilizada e a sua distância ao observador. Tais parâmetros são utilizados na definição do volume de visão. Os parâmetros de posicionamento de cada *canvas* em relação ao observador definem transformações aplicadas ao volume de visão.

Uma vez que todas as fases acima tenham sido efetivadas, a UV estará pronta para que sua operação seja iniciada através da carga de um ambiente virtual, que é feita a partir de um comando emitido pelo mediador indicando a cena. O processo de carga é feito através do uso de uma linguagem interpretada, sobre a qual são apresentados maiores detalhes na seção 4.1. Durante esse processo, à medida que cada elemento do ambiente virtual (entidade, comportamento, aparência, etc.) for sendo criado, uma instância equivalente, nos moldes da infra-estrutura de comunicação, vai sendo gerada dinamicamente no mediador. Ao término deste processo, o mediador dispõe de uma interface para ter acesso a cada entidade pertencente a cada UV, bem como a todos os seus métodos públicos.

3.5.2 - Alteração do estado dinâmico pela rede

A alteração do estado dinâmico é feita por intermédio da infra-estrutura de comunicação, através dos *proxies* existentes no mediador, que permitem o acesso aos objetos criados durante a carga do ambiente virtual. Dessa forma, por exemplo, se a velocidade de determinada entidade for alterada no mundo externo, o mediador altera a mecânica referente a esta entidade em todas as UVs registradas, informando o momento exato em que a alteração acorreu.

Esse processo só é executado quando um evento de atualização proveniente do mediador chega à UV por meio de chamadas aos métodos locais dos objetos. Quando não há alterações pendentes na fila de eventos, todo o processamento é dedicado às outras etapas do funcionamento da UV, descritas a seguir.

3.5.3 - Manutenção do estado dinâmico corrente

A necessidade de atender aos requisitos de escalabilidade requer que as UVs estejam sempre tirando o máximo proveito dos recursos de *hardware* existentes. A cada instante o estado dinâmico corrente é processado em função do tempo presente, dado pelo relógio local de simulação de cada UV. A Figura 10, apresentada na seção 2.3.3, ilustra o comportamento de duas UVs heterogêneas.

O processamento realizado nesta fase é basicamente o indicado na seção 2.3.1, que trata das técnicas de predição e convergência do estado dinâmico corrente. Conseqüentemente, o seu papel é atualizar, em função do tempo e das últimas atualizações

recebidas, os parâmetros do estado dinâmico corrente que serão utilizados na fase de renderização de um quadro da imagem, conforme explicado a seguir.

3.5.4 - Processamento gráfico de um quadro

Esta é a fase que geralmente demanda a maior parcela do tempo de processamento de uma UV. Quanto mais depressa esta tarefa for realizada, maior será a taxa de quadros exibidos por segundo. Conseqüentemente, deve-se concentrar os esforços no seu desenvolvimento, a fim de se obter um resultado final adequado aos requisitos de qualidade e desempenho traçados inicialmente.

O processamento gráfico pode ser subdividido em algumas fases. Uma vez que a API gráfica do OpenGL esteja sendo utilizada, as fases iniciais consistem na configuração de seu estado corrente e na passagem de todos os atributos necessários para a renderização de um quadro. A fase final é feita pelo OpenGL, que renderiza a imagem a partir do conjunto de parâmetros que lhe foram passados na fase anterior. Procedimentos semelhantes são adotados caso outras APIs gráficas venham a ser utilizadas.

No decorrer da fase inicial, o primeiro passo é o estabelecimento da matriz que determina o posicionamento do observador em relação ao restante das entidades. A seção 4.2.2 apresenta maiores detalhes sobre as operações efetuadas nesse procedimento. Em seguida, são configurados atributos relacionados à iluminação e à presença de *fog* no ambiente.

O passo seguinte é o estabelecimento da aparência, seguido da passagem para o OpenGL das primitivas geométricas de cada entidade. Vale notar que simplesmente passar ao OpenGL todas a primitivas geométricas pertencentes a todas as entidades não é um procedimento eficiente. Pois, para se manter uma taxa superior a 20 quadros por segundo, em uma máquina que apresente um desempenho médio de 1.000.000 de polígonos por segundo, seria necessário que a descrição geométrica de todas as entidades presentes no ambiente virtual fosse limitada a cerca de 50.000 polígonos. Tal valor constitui um fator limitante da escalabilidade. Conseqüentemente, faz-se necessário o uso de algumas técnicas que reduzam a quantidade de polígonos enviados para o OpenGL sem reduzir a qualidade visual apresentada.

A eliminação das entidades que estejam completamente fora do volume de visão é uma abordagem possível. Um ponto de partida para este estudo é o trabalho desenvolvido por Jim Clark [CLARK76], em que o ambiente virtual é dividido em volumes espaciais

controlados por uma estrutura de árvore. Outra técnica é baseada no cálculo de conjuntos potencialmente visíveis (PVS – *Pontentially Visible Sets*) de polígonos [AIREY90][BROOKS86][MINE96]. Os polígonos são divididos em células, e apenas as células visíveis a partir da célula onde o observador se encontra são desenhadas. Esta técnica utiliza uma estrutura de árvore de partição espacial binária (BSP – *Binary Space Partitioning*) [FUCHS83][FUCHS80].

O uso das técnicas acima pode, ainda, ser insuficiente. Por isso, outra abordagem também pode ser enfocada. Em seu trabalho, Jim Clark [CLARK76] também discute a necessidade da adequação do nível de detalhe (LOD – *Level of Detail*) em relação à distância entre o objeto sendo desenhado e o observador. Isso decorre do fato de que, por exemplo, desenhar uma entidade dotada de 10.000 polígonos a uma grande distância do observador não acrescenta nenhum ganho efetivo de qualidade visual. Uma versão simplificada da entidade, contendo cerca de 500 polígonos, poderia ser desenhada caso o resultado final ocupasse apenas uma pequena porção da tela, alcançando o mesmo efeito visual. A Figura 17 ilustra a relação existente entre o volume de visão e o nível de detalhe.

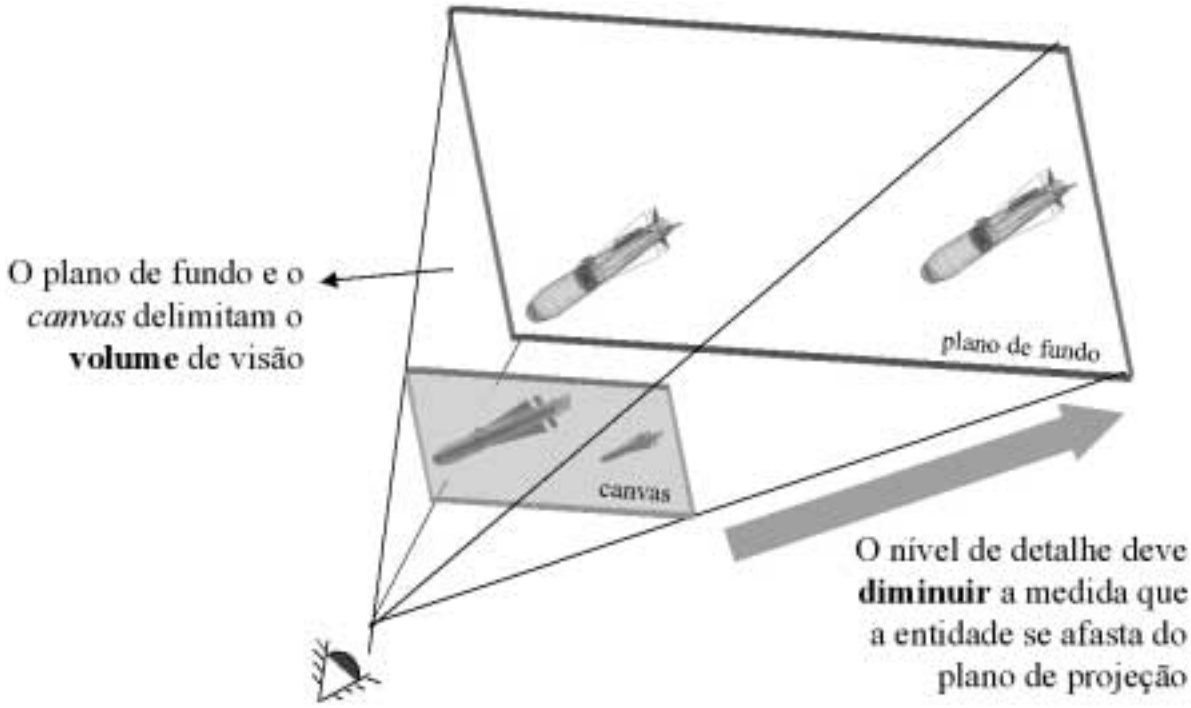

Figura 17 – Relação entre o LOD e a distância do objeto ao observador

Para implementar o tratamento de LOD, uma solução simples é o fornecimento de diversas descrições geométricas, em diferentes níveis de detalhe, de uma mesma entidade. A implementação dessa abordagem é simples e gera bons resultados em termos de

desempenho, porém ela ocupa muita memória para armazenar todas as diversas resoluções, além de produzir um artefato visual desagradável quando da transição entre duas resoluções vizinhas.

Neste caso, a solução mais adequada seria a utilização de técnicas para a simplificação de malhas em tempo real. Luekebke apresenta um resumo de diversas técnicas existentes [LUEKEBKE97].

Em uma das técnicas, é apresentada uma solução eficiente para calcular e armazenar essas simplificações conhecida como malhas progressivas (*progressive meshes*) [HOPPE96]. O seu uso permite, além de fornecer qualquer resolução desejada em tempo de execução, a realização de *morph* entre 2 resoluções, evitando assim o artefato visual causado pela transição.

O uso das técnicas de eliminação de entidades que estejam fora do volume de visão e da técnica de múltiplos níveis de detalhe exposto acima é indicado pela arquitetura. Isso implica em que, quando as descrições geométricas das entidades estiverem sendo carregadas, durante a fase de inicialização, sejam aplicados algoritmos de simplificação do modelo de forma a gerar as malhas progressivas que serão utilizadas durante a execução do programa. O uso dessas técnicas implica também na realização de testes durante o processamento gráfico, a fim de eliminar as entidades localizadas fora do volume de visão e determinar o nível de detalhe adequado a ser utilizado pelas entidades que não foram eliminadas.

Ainda com relação à questão do desempenho, no que diz respeito ao uso de aparências contendo texturas, indica-se a adoção da técnica de *mipmapping* [WILLIAMS83]. Seu princípio é o mesmo da técnica de LOD: a determinação de múltiplos níveis de detalhe. Da mesma forma, o processamento para a geração das texturas nas diversas resoluções deve ser feito durante a carga das aparências na fase de inicialização do sistema.

É interessante chamar a atenção dos seguintes fatos. As UVs rodam independentemente, e cada uma delas fornece, possivelmente, uma visão de uma porção diferente do ambiente virtual. Uma vez que as técnicas para aumento de desempenho indicadas são dependentes da posição das entidades em relação ao observador, conclui-se que cada UV vai dedicar sua capacidade de processamento aos elementos relevantes na

composição de sua porção da imagem. Tal característica reforça a decisão de tirar o máximo proveito da capacidade de processamento de cada máquina presente no sistema.

3.6 - Modelagem do ambiente virtual

Seja qual for o tipo de ambiente virtual que esteja sendo simulado, é necessária a definição de um modelo estruturado que o descreva. Esse modelo deve fornecer maneiras eficientes de modificação e recuperação de dados de suas componentes. Além disso, é interessante que ambientes virtuais diferentes possam ser modelados e que exista a possibilidade de reuso de seu código de representação.

Para atingir tais objetivos, optou-se por uma modelagem baseada no uso de Lua [IERUSALIMSCHY96][LUA]. Lua é uma linguagem interpretada que combina facilidades de descrição de dados e procedimentos convencionais usando uma sintaxe simples e clara, além de suportar todas as estruturas de controle convencionais, incluindo expressões, laços, declarações condicionais e chamadas de funções. O uso destas facilidades permite a construção de ambientes virtuais elaborados.

As componentes que descrevem o ambiente virtual são subdivididas em três grupos diferentes, de acordo com a sua aplicação na configuração das unidades de visualização: o grupo das **componentes exclusivas**, o das **componentes compartilhadas** e o das **componentes comuns**. A Figura 18 ilustra esta divisão.

Figura 18 – Modelagem das componentes do ambiente virtual

3.6.1 - Componentes exclusivas

No grupo de componentes exclusivas encontram-se as componentes que se aplicam a apenas uma única UV. A descrição do *canvas* se encontra neste grupo, uma vez que não faz sentido fornecer a mesma visão do ambiente virtual em duas UVs distintas, pois isso implicaria em que as duas UVs estivessem ocupando o mesmo lugar no espaço.

Os parâmetros que caracterizam o *canvas* são aqueles relacionados com seu tamanho e seu posicionamento em relação ao observador. A indicação do *canvas* a ser utilizado por uma determinada UV é feita durante a fase de inicialização dessa UV, antes que seu registro seja feito no mediador, uma vez que o identificador do registro é baseado nos atributos do *canvas*.

3.6.2 - Componentes compartilhadas

Neste grupo encontram-se as componentes comuns a determinadas UVs. Considerando que o sistema de visualização distribuída venha a ser utilizado por vários usuários simultaneamente, cada um observando o ambiente virtual de um diferente ponto de vista, a arquitetura permite que as UVs sejam estruturadas de forma que cada usuário disponha de mais de uma delas compondo um painel ao seu redor, como ilustrado Figura 19.

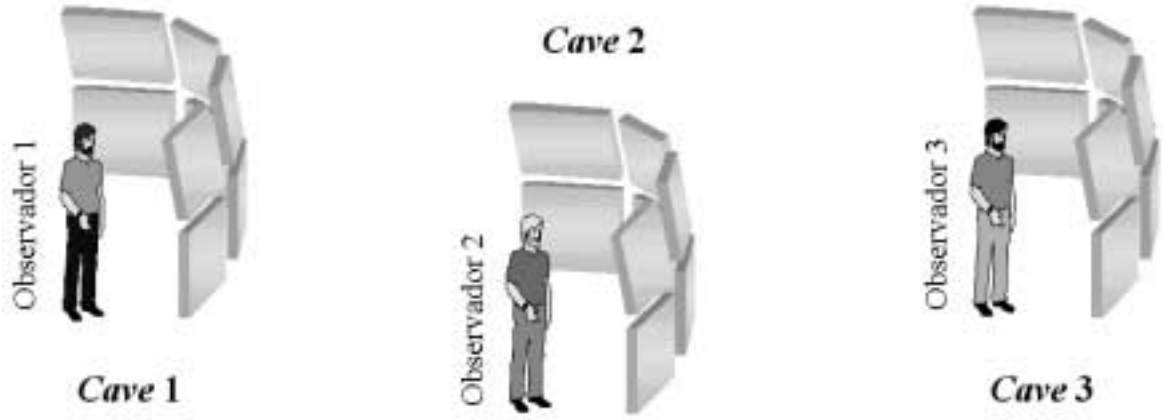

Figura 19 – Utilização do sistema por usuários em *caves*

A partir da análise da configuração apresentada na Figura 19, pode-se concluir que o observador e a *cave* (ambiente físico onde os dispositivos de visualização são dispostos) na qual ele se encontra constituem as componentes compartilhadas na modelagem do ambiente virtual. Isso se deve ao fato de que todas as UVs que compõem a *cave* 1 estão

renderizando imagens cujo ponto de vista depende do observador 1, condição que se repete para as *caves* 2, 3, etc.

Ao ser inicializada, cada UV também deve receber a indicação de a qual *cave* ela pertence. A *cave*, por sua vez, contém como atributo a descrição do observador que a utiliza. Este observador é, no momento da carga do ambiente virtual, ligado a uma entidade do mesmo, que serve de referência para seu posicionamento e controle.

3.6.3 - Componentes comuns

A este grupo pertencem todas as entidades e outros atributos do ambiente virtual. Durante a carga do sistema, os elementos que compõem este grupo são instanciados igualmente em todas as UVs. Tal restrição é indicada, uma vez que não faria sentido construir um ambiente virtual no qual determinadas entidades só existissem em determinadas UVs. Isso constituiria uma quebra da integridade, que conseqüentemente produziria inconsistências na visualização.

Compõem este grupo todos os atributos relacionados com o cenário, entre eles a iluminação e a presença de *fog*, bem como os atributos relacionados com as entidades, suas descrições geométricas, seus comportamentos e suas aparências.

CAPÍTULO 4 – IMPLEMENTAÇÃO DO SISTEMA

Apresentar uma implementação das principais características da arquitetura proposta, indicando os métodos e as técnicas utilizados para solucionar determinados problemas. Apresentar um conjunto de testes realizados.

4.1 - Modelagem do Ambiente Virtual

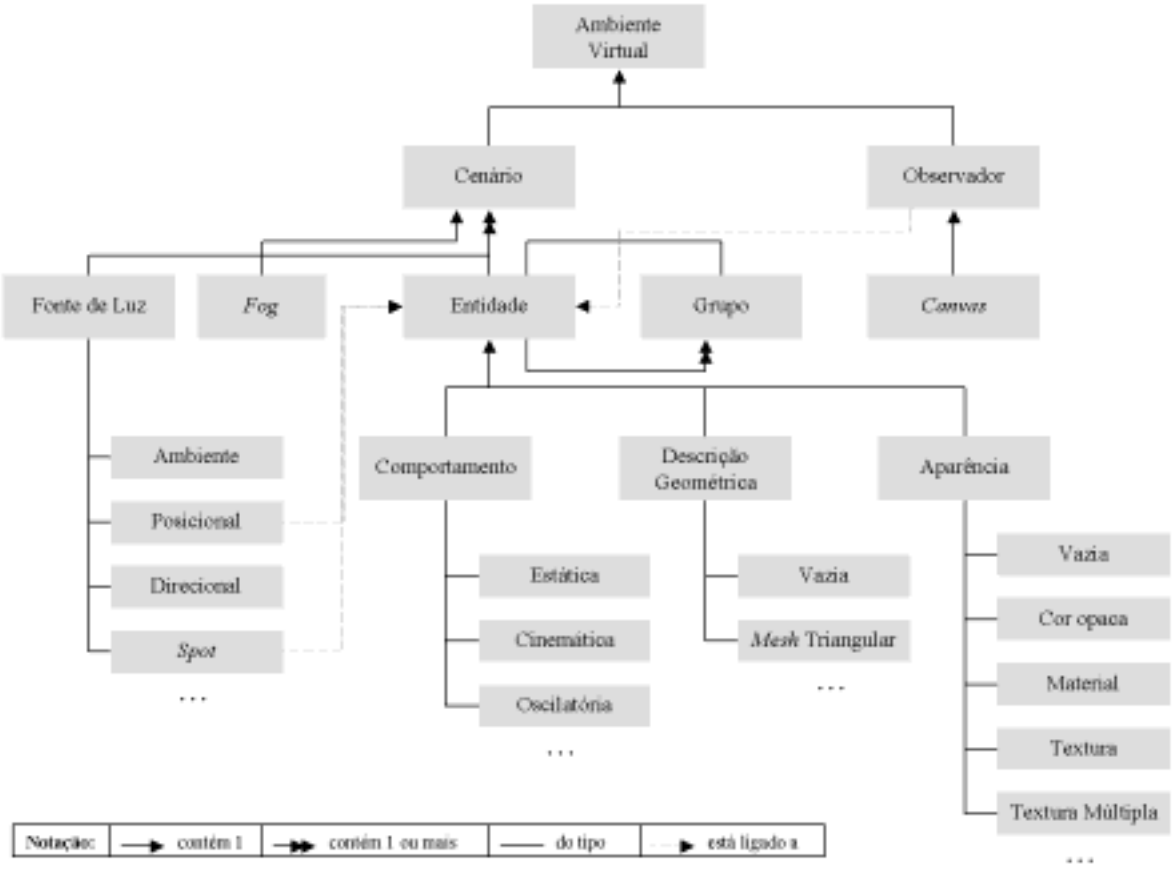

Figura 20 – Estrutura de um ambiente virtual

Os recursos disponibilizados para a modelagem do ambiente virtual constituem um fator importante na determinação da usabilidade do sistema de visualização. A modelagem deve ser genérica, extensível e clara.

A modelagem do ambiente virtual foi estruturada conforme ilustra a Figura 20. O conjunto das componentes apresentadas contêm apenas aquelas efetivamente implementadas. Contudo, a estrutura foi elaborada com o intuito de permitir extensões a este modelo, através da adição de novas componentes.

4.1.1 - Estrutura das componentes

Baseado na estrutura apresentada na Figura 20, o ambiente virtual contém obrigatoriamente um **cenário** e um **observador**.

O observador é um elemento compartilhado que é instanciado durante a inicialização de cada UV. Além de estar ligado a uma entidade do ambiente virtual, ele está associado a uma instância de *canvas*, elemento exclusivo também instanciado durante a inicialização de cada UV.

O cenário é composto por um conjunto de entidades, sendo cada uma dotada de uma instância de **comportamento**, **descrição geométrica** e **aparência**. Uma entidade também pode ser composta por um **grupo** de outras entidades, de forma recursiva. O comportamento de uma entidade pode ser uma mecânica do tipo **estática**, **cinemática** ou **oscilatória**. A descrição geométrica pode ser do tipo **vazia** ou **malha triangular**. Já a aparência pode ser do tipo **vazia**, **cor opaca**, **material**, **textura** ou **textura múltipla**. O cenário também contém instâncias de **fontes de luz** e opcionalmente uma instância de ***fog***. Cada fonte de luz pode ser do tipo **ambiente**, **posicional**, **direcional** ou ***spot***. A fonte de luz posicional e a fonte de luz *spot* podem estar ligadas a uma entidade. Neste caso, o posicionamento da fonte de luz é dado em função da posição da entidade.

4.1.2 - Descrição dos atributos

O espaço cartesiano foi escolhido como base para a modelagem do ambiente virtual, uma vez que ele é o sistema de coordenadas utilizado pela maioria das aplicações existentes (usuárias em potencial do sistema). Caso o espaço utilizado pela aplicação existente no mundo externo seja outro, cabe ao mediador a função de realizar a conversão entre os sistemas de coordenadas.

Com base no sistema de coordenadas escolhido, são definidos duas componentes auxiliares: o **vetor** e a **quádrupla**. O vetor é composto por três coordenadas nas direções *x*, *y* e *z*. A quádrupla consiste em um vetor de quatro coordenadas, sendo a primeira um escalar representando um ângulo de rotação e as três outras um vetor em torno do qual a rotação é feita. A utilização do vetor está relacionada com translações e posicionamento, enquanto a utilização da quádrupla está relacionada com rotações.

Internamente, as quádruplas são convertidas em quatérnios [HAMILTON66], que são utilizados para a composição de rotações. A escolha dessa abordagem foi motivada por uma série de vantagens decorrentes de seu uso frente a outras abordagens [BURCHFIEL90].

Para fins de identificação, todas as componentes do ambiente virtual contêm um atributo de nome, que consiste de um campo textual.

A seguir são apresentados os atributos que descrevem cada componente do ambiente virtual.

Ambiente virtual: é uma representação abstrata, que leva em conta apenas os atributos de interesse para o problema de visualização, controlada pelo mundo externo. Por esta razão, suas componentes são um cenário e um observador.

Observador: é um elemento compartilhado por um grupo de UVs que estão em uma mesma *cave*. Ele está sempre ligado a uma entidade do cenário, com vistas a herdar seu comportamento para fins de posicionamento. A posição relativa do observador em relação à entidade à qual ele está ligado é dada por um vetor e uma quádrupla denominados $\boldsymbol{T_o}$ e $\boldsymbol{Q_o}$, representando sua posição espacial linear e angular, respectivamente. Em cada UV, o observador está associado a uma instância de *canvas*, que é utilizado para o desenho.

Canvas*:** é uma componente exclusiva de cada UV. Sua função, definida na seção 2.2, é mapear o posicionamento de um dispositivo de visualização real em uma superfície virtual. Os atributos definidos a seguir levam em consideração um observador fixo olhando na direção do eixo ***z negativo e posicionado sobre a origem do sistema global. Esta é definida como a posição canônica do observador. Dessa forma, a Figura 21a ilustra a distância ***d*** do observador ao *canvas*, a largura ***w*** do *canvas*, a altura ***h*** do *canvas*, a resolução horizontal em *pixels* ***pw*** do *canvas* e a resolução vertical em *pixels* ***ph*** do *canvas*. Na Figura 21b são ilustrados dois *canvases*, o *canvas 1* encontrando-se perpendicular ao eixo ***z*** negativo e o *canvas 2* encontrando-se girado de um atributo ***teta1*** graus ao redor do

eixo ***y***. Neste caso as distâncias ***d1*** e ***d2*** de ambos ao observador são iguais. Na Figura 21c o *canvas 2* foi deslocado para uma posição coplanar ao *canvas 1*, que permaneceu parado. Para conseguir tal posicionamento, o atributo de distância ***d2*** foi incrementado e foi aplicada uma rotação de ***teta2*** graus em torno de um eixo ***y´*** cujo sistema local está localizado no centro do *canvas 2*, como ilustrado na figura.

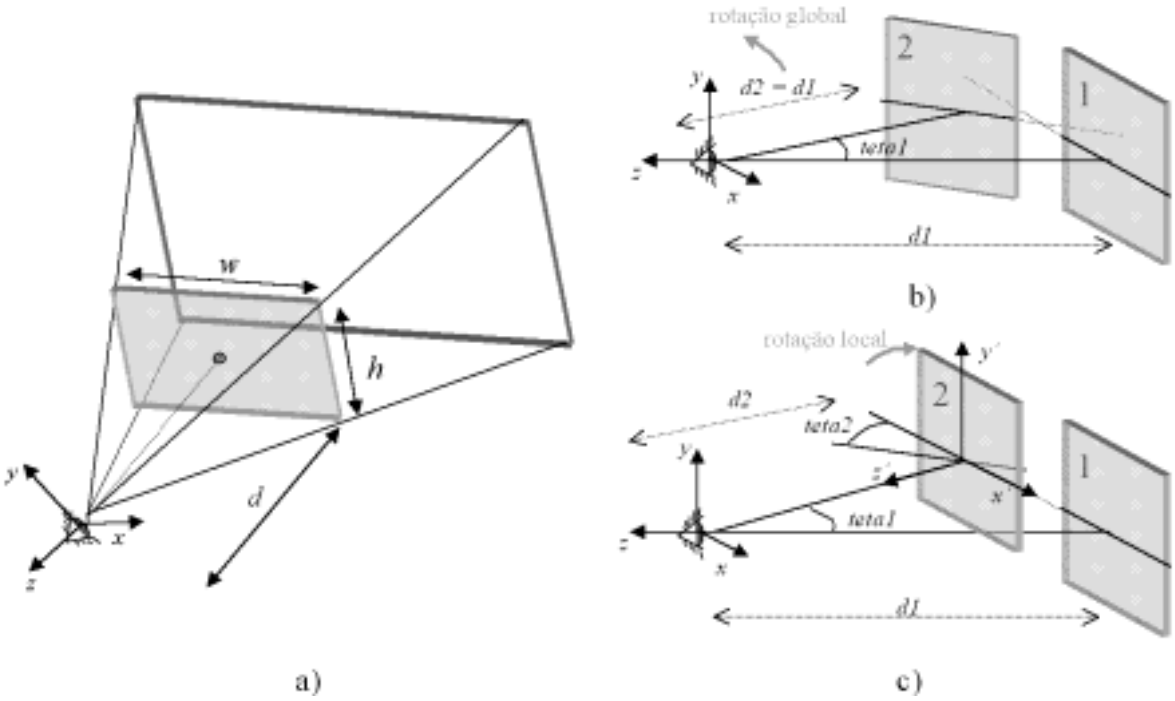

Figura 21 – Atributos de configuração de um *canvas*

De forma análoga, outros arranjos de posicionamento podem ser conseguidos através da composição de rotações aplicadas sobre o *canvas*. A rotação resultante em torno dos eixos ***x***, ***y***, e ***z*** é denominada $\boldsymbol{M_c}$. As rotações em torno do eixos locais ***x'*** e ***y'*** do *canvas* são dadas pelos atributos $\boldsymbol{teta_{x'}}$ e $\boldsymbol{teta_{y'}}$.

É possível, através desses atributos, mapear o posicionamento dos *canvases* em relação a diversos tipos e arranjos de dispositivos de visualização possíveis. Certamente o sistema não apresenta resultados válidos para situações impossíveis, como uma rotação em ***y*** de 90^o acompanhada de outra rotação em ***y'*** de -90^o, por exemplo.

Cenário: é um elemento que congrega os elementos comuns que modelam o ambiente virtual. Isso implica em que o cenário seja o mesmo em todas as UVs. Ele é composto basicamente por uma lista de entidades, uma lista de fontes de luz e uma instância de *fog*.

Entidades: são os principais habitantes do ambiente virtual. Elas podem ser simples ou compostas, sendo que as entidades compostas são aquelas formadas por um **grupo** de outras entidades (de forma recursiva). Essa estruturação hierárquica torna possível a elaboração de entidades dotadas de comportamentos mais elaborados.

Cada entidade possui uma instância de comportamento, aparência e descrição geométrica.

Comportamento: está associado ao estado da entidade em um determinado instante ***t*** no tempo. Foram implementados comportamentos ligados ao posicionamento das entidades no ambiente virtual. O posicionamento é, dessa forma, dado pelas funções $T_m(t)$ e $Q_m(t)$, que representam respectivamente a posição e a orientação da entidade no espaço do ambiente virtual.

O comportamento do tipo mecânica estática é utilizado para entidades que permanecem paradas. Isto é, as funções de posicionamento $T_m(t)$ e $Q_m(t)$ são constantes no tempo.

No comportamento do tipo mecânica cinemática, as funções de posicionamento $T_m(t)$ e $Q_m(t)$ seguem as leis físicas da cinemática. Para isso existem atributos que definem a velocidade linear (***vx***, ***vy*** e ***vz***) e a velocidade angular (***wx***, ***wy***, e ***wz***) da entidade.

No comportamento do tipo mecânica oscilatória, as funções de posicionamento $T_m(t)$ e $Q_m(t)$ acarretam oscilações tanto lineares quanto angulares no posicionamento das entidades. Para a configuração desse comportamento são utilizados diversos atributos, dentre os quais podem ser identificados um vetor contendo a direção de oscilação linear ***tx***, ***ty*** e ***tz*** e um vetor contendo o eixo ***rx***, ***ry*** e ***rz*** ao redor do qual ocorrerá a oscilação angular. Além disso existem os atributos escalares de período e comprimento angular da oscilação de rotação, bem como período e comprimento linear da oscilação de translação.

Descrição geométrica: pode ser vazia, tipo usualmente utilizado para a entidade ligada ao observador, ou pode ser do tipo malha triangular. Todas as descrições geométricas possuem três atributos em comum: uma translação, uma rotação e uma escala, dados respectivamente por um vetor T_s, uma quádrupla Q_s e um vetor S_s. Tais transformações são utilizadas para compatibilizar os modelos gerados por outros programas, que, geralmente, usam posicionamento, orientação e escala diferentes dos usados no espaço do ambiente virtual.

A malha triangular é um tipo de descrição geométrica simples obtida por intermédio da leitura de um arquivo textual contendo os seguintes atributos: o número de pontos, o número de faces triangulares, uma lista de pontos e uma lista de triângulos. Além do nome do arquivo contendo o modelo, existem ainda atributos que habilitam o desenho de uma caixa de contorno, que indicam a orientação dos planos usados para a geração automática de textura e o critério de simplificação utilizado pelo algoritmo de múltiplos níveis de detalhe.

Aparência: pode ser de diversos tipos. A aparência vazia não possui atributos. Ela pode ser aplicada, por exemplo, a uma entidade à qual o observador ou uma fonte de luz estejam ligados.

A aparência de cor opaca tem como atributo apenas um vetor contendo a intensidade de cor nas três componentes, R, G e B.

A aparência de material tem como atributo um vetor contendo a intensidade de cor difusa refletida, um vetor contendo a intensidade de cor especular refletida e um fator escalar de brilho.

A aparência de textura possui apenas dois atributos, um contendo o nome do arquivo de textura e outro que indica se deve ser usada a técnica de *mipmaping*.

A aparência de múltiplas texturas é dada pela aplicação sucessiva de um conjunto de texturas que permanecem aplicadas por um período constante. Dessa forma, os atributos que descrevem essa aparência são o nome do diretório que contém todas as texturas e o período que cada textura permanece aplicada.

Fontes de luz: os atributos de cada fonte de luz definem a forma como ela influencia a iluminação de todos os objetos da cena. Sejam quais forem, as fontes de luz têm em comum o atributo intensidade, que é dado pelas três componentes de cor, R, G e B, variando de 0 a 1.

A luz direcional é aquela que se encontra posicionada em uma determinada direção no infinito, fazendo com que seus raios sejam todos paralelos.

A luz posicional é pontual e seus raios partem de sua posição para todas as direções. Ela pode ter uma posição absoluta no espaço ou ser ligada a uma entidade. Para isso, possui um atributo de posicionamento que é dado por um vetor contendo a sua posição em relação ao sistema global ou local (caso esteja ligada a uma entidade).

***Fog*:** é uma componente opcional que permite inserir o efeito de neblina na cena. Para isso é necessário atribuir valores para a cor da neblina, a cor de fundo, a distância máxima em que se tem 100% de visibilidade e a distância a partir da qual se tem 0% de visibilidade.

4.1.3 - Formato de armazenamento

As componentes apresentadas acima são armazenadas em um arquivo codificado em Lua, que pode conter uma simples descrição ou um programa elaborado para descrever o ambiente virtual.

Abaixo, seguem extratos da descrição de um ambiente virtual. Cada extrato contém exemplos de como é feita a configuração de cada uma das componentes do ambiente virtual.

- Ambiente virtual:

```
Visualizer{ name = "Visualizer",
            scene = scen,
            observer = ent1, }
```

- Observador:

```
Observer{  name = "Cave1",
           axis_position = Vector {0.0,0.0,0.0},
           axis_rotation = Quadruple {0,0,1,0}, }
```

- *Canvas*:

```
Canvas{  name = "Esquerdo",
         distance = 0.07709,
         width = 0.0323,
         height = 0.023534,
         rotation = Quadruple {25.7699, 0, 1, 0},
         windowsize_x = 1146,
         windowsize_y = 835,
         local_rotation_x = 0,
         local_rotation_y = -25.7699, }
```

- Cenário / *Fog*:

```
scen = Scene{   name = "Scene",
                fog = TRUE,
                back_color = Vector{128/255, 183/255, 223/255},
                fog_color = Vector{128/255, 179/255, 223/255},
                fog_start = 150,
                fog_end = 1000;
                light0, light1,
                ent1, ent2, ent3, }
```

- Entidade:

```
ent1 = Entity{ name = "Hind 1",
               shape      = shp3,
               appearance = app2,
               mechanics  = sta1, }
```

- Comportamento:

```
sta5   = Static_Mechanic{   name = "Static 5",
                            position = Vector{-350.0, -5.0, -100.0},
                            rotation = Quadruple{0,0,1,0}, }

kin1   = Kinematic_Mechanic { name = "Kinematic 1",
                            position = Vector{7.0, 1.7, 700.0},
                            rotation = Quadruple{20.0, 0.0, 1.0, 0.0},
                            linear_speed = Vector{-0.045, 0.0, -0.15},
                            angular_speed = Vector{0.0, 10.0, 0.0},
                            actual_time = 0, }

osc2   = Oscillation_Mechanic{ name = "Oscillation 2",
                               position = Vector{0.0, 6.0, 650.0},
                               rotation = Quadruple {180,0,1,0},
                               actual_time = 0,
                               translation_period = 12,
                               translation_size = 30,
                               translation_direction = Vector{1,0,0},
                               rotation_period = 5,
                               rotation_size = 100,
                               rotation_direction = Vector{0,0,1}, }
```

- Descrição geométrica:

```
shp1 = MeshPly_Shape{   name = "Hint",
                        filename = ".\\data\\shapes\\hind.ply",
                        axis_position = Vector{0,0,0},
                        axis_rotation = Quadruple{180.0, 0.0, -1.0, -1.0},
                        scale = Vector{0.1,0.1,0.1},
                        auto_center = TRUE,
                        lod_criteria = EDGE_LENGTH + LOCAL_CURVATURE +
                                       PRESERVE_BOUNDARY,
                        bounding_box = FALSE, }
```

- Aparência:

```
app1 = Empty_Appearance{ name = "Transp" }


app2 = Color_Appearance{ name = "Blue",
                         color = Vector{34/255, 41/255, 65/255}, }

app3 = Material_Appearance{ name = "Steel",
                            diffuse = Vector{0.5, 0.5, 0.5},
                            specular = Vector{1.0, 1.0, 1.0},
                            shininess = 1, }

app4 = AutoTexture_Appearance{ name = "Sea",
                               filename = ".\\data\\textures\\mar4.tif",
                               mipmap = TRUE, }
app5 = Multi_AutoTexture_Appearance{ name = "Sea",
                                     directory =".\\data\\textures\\sea\\",
                                     mipmap = TRUE,
                                     cycle_interval = 0.2,
                                     actual_time = 0, }
```

- Fontes de luz:

```
light0 = Ambient_Light{ name = "Ambient 1",
                        intensity = Vector{0.1, 0.1, 0.1}, }

light1 = Positional_Light{ name = "Positional 1",
                           intensity = Vector{0.2, 0.2, 0.2},
                           position = Vector{0.0, -2.0, 100.0},
                           entity = ent1, }

light3 = Directional_Light{ name = "Directional 1",
                            intensity = Vector{0.05, 0.05, 0.05},
                            direction = Vector{0.0, 1.0, 0.0}, }
```

Pode-se observar que o uso da linguagem Lua para modelar o ambiente virtual permite que a modelagem seja feita de forma procedural, através da instanciação direta de objetos e chamadas de seus métodos. Isto é possível pois toda a interface de classes em C++ é exportada para Lua. A sintaxe total dos comandos não será apresentada, somente alguns deles para fins de ilustração.

```
app1 = autotexture:new(“Textura 1”, ".\\data\\textures\\mar4.tif", TRUE)

shp1 = meshply:new(“Estatica 1”, ".\\data\\shapes\\hind.ply",
                   Vector{0,0,0}, Quadruple{180.0, 0.0, -1.0, -1.0},
                   0.1, EDGE_LENGTH + LOCAL_CURVATURE, FALSE)

kin1 = kinematics:new(“Cinematica 1”, Vector{0,0,0},
                      Quadruple{90.0, 0.0, 0.0, 1.0})
kin1:sett(Vector{-0.045, 0.0, -0.15}, 0.0)
kin1:setr(Vector{0.0, 10.0, 0.0}, 0.0)

ent1 = entity:new(“Hind”)
ent1:set(app1)
ent1:set(sta1)
ent1:set(shp1)
…
```

É interessante mencionar que, uma vez que todos esses comandos são exportados da UV para Lua, eles também podem ser exportados de Lua para outra aplicação remota através da interface LuaOrb. É dessa forma que os comandos são enviados do mediador para as UVs. Com isso, o mediador pode configurar e alterar o estado dinâmico corrente de todas as componentes das UVs remotamente.

4.2 - Unidade de Visualização

Cada UV tem como requisito alcançar qualidade e desempenho compatíveis com os objetivos inicialmente traçados. No desenvolvimento do sistema foram tomadas diversas decisões tecnológicas. A Figura 22 apresenta uma imagem da aplicação desenvolvida sendo executada em três computadores ligados em rede.

Figura 22 – Visão do sistema de visualização em uso

A seguir são apresentados alguns detalhes de funcionamento e as decisões tecnológicas adotadas no desenvolvimento desta aplicação.

4.2.1 - Decisões tecnológicas

Devido à natureza dos elementos ligados ao problema de visualização, optou-se pela modelagem e implementação adotando uma abordagem orientada a objetos, sendo escolhida a linguagem de programação C++. Decidiu-se também pela adoção da STL (*Standard Template Library*) [MUSSER94], em função dos recursos que ela apresenta, que facilitam a estruturação interna da aplicação.

Seguindo a orientação da arquitetura, foi utilizada a API gráfica do OpenGL versão 1.1 como máquina de renderização do sistema [SEGAL97]. Também com relação ao problema gráfico, foi utilizada uma biblioteca de manipulação de níveis de detalhes em malhas poligonais com recursos de *morphing*, baseada em uma adaptação da técnica de malhas progressivas [MATTOS99][HOPPE96][MELAX98].

A interface desta aplicação é simples, uma vez que praticamente toda a entrada de dados é feita por intermédio de outra aplicação (o mediador do sistema). Neste caso, é necessária apenas a superfície de visualização. Além disso, optou-se por apresentar diálogos contendo informações de acompanhamento e registro (*log*) do sistema que são utilizadas para testes do sistema. Dentre as informações apresentadas no diálogo de acompanhamento estão a taxa de quadros por segundo, a quantidade total de polígonos sendo desenhados em um determinado instante e a quantidade de entidades presentes no ambiente virtual. A biblioteca IUP [LEVY96] foi escolhida como API para o desenvolvimento da interface em função de características como portabilidade para diversas plataformas de desenvolvimento.

A linguagem Lua versão 3.1 [LUA] foi utilizada no desenvolvimento da UV, seguindo a indicação feita pela arquitetura. Tal decisão mostrou-se muito favorável, pois os recursos obtidos com a utilização desta linguagem, aliados à facilidade de acoplamento (hospedagem) com a aplicação em C++, proporcionaram grande maleabilidade ao sistema, tanto em termos de desenvolvimento quanto de utilização.

Com respeito à infra-estrutura de comunicação, foi utilizada uma implementação da arquitetura CORBA denominada Orbacus, produzida pela Object-Oriented Concepts, Inc. A interface de CORBA com Lua foi feita utilizando a ferramenta LuaOrb versão 0.2 *Alpha* [LUAORB].

4.2.2 - Processamento gráfico

Os aspectos relevantes da fase de processamento gráfico a serem abordados nesta seção restringem-se à construção do modelo virtual de câmara e ao processo de posicionamento de todas as entidades a serem renderizadas. A escolha destes procedimentos visa tornar claro seu funcionamento, uma vez que eles representam, em conjunto com as técnicas de manutenção do estado corrente, a base de funcionamento das UVs.

Posicionamento da câmara virtual

O modelo de câmara é baseado nos atributos de inicialização e carga do ambiente virtual. Os atributos de inicialização, juntamente com os atributos de estado corrente, compõem o conjunto de transformações que constitui o posicionamento da câmara virtual. O processo de construção e posicionamento da câmara virtual consiste em estabelecer uma matriz de projeção e uma matriz de modelo que será usada pelo OpenGL no processo de renderização do cenário.

Inicialmente é feita a configuração do mapeamento das coordenadas do espaço normalizado do OpenGL nas coordenadas da tela (*viewport*), utilizando os atributos de descrição do *canvas* indicados na Figura 21a.

Em seguida, obtém-se a matriz de projeção através da composição das seguintes transformações.

Calcula-se uma matriz de projeção perspectiva ***P*** baseada nos atributos ilustrados na Figura 4, seção 2.2, que determinam o formato do volume de visão.

Caso o *canvas* apresente uma rotação local, como está ilustrado na Figura 21c, é feito o cálculo de uma matriz de cisalhamento (*shear*) ***SH***, indicada a seguir. Essa matriz, baseada nos parâmetros $\boldsymbol{teta_{x'}}$ e $\boldsymbol{teta_{y'}}$, é multiplicada pela matriz projeção ***P***. Isto é feito para distorcer o volume de visão da configuração ilustrada na Figura 23a para a configuração da Figura 23b. Dessa forma, é possível configurar o painel composto por dois *canvases* colocados lado a lado, como ilustrado na Figura 23c. Para isso, basta que a rotação local $\boldsymbol{teta_{y'}}$ seja igual ao ângulo de rotação do *canvas* em relação ao observador, como está ilustrado na Figura 23b.

$$SH = \begin{bmatrix} 1 & 0 & \tan(teta_{y'}) \\ 0 & 1 & \tan(teta_{x'}) \\ 0 & 0 & 1 \end{bmatrix}$$

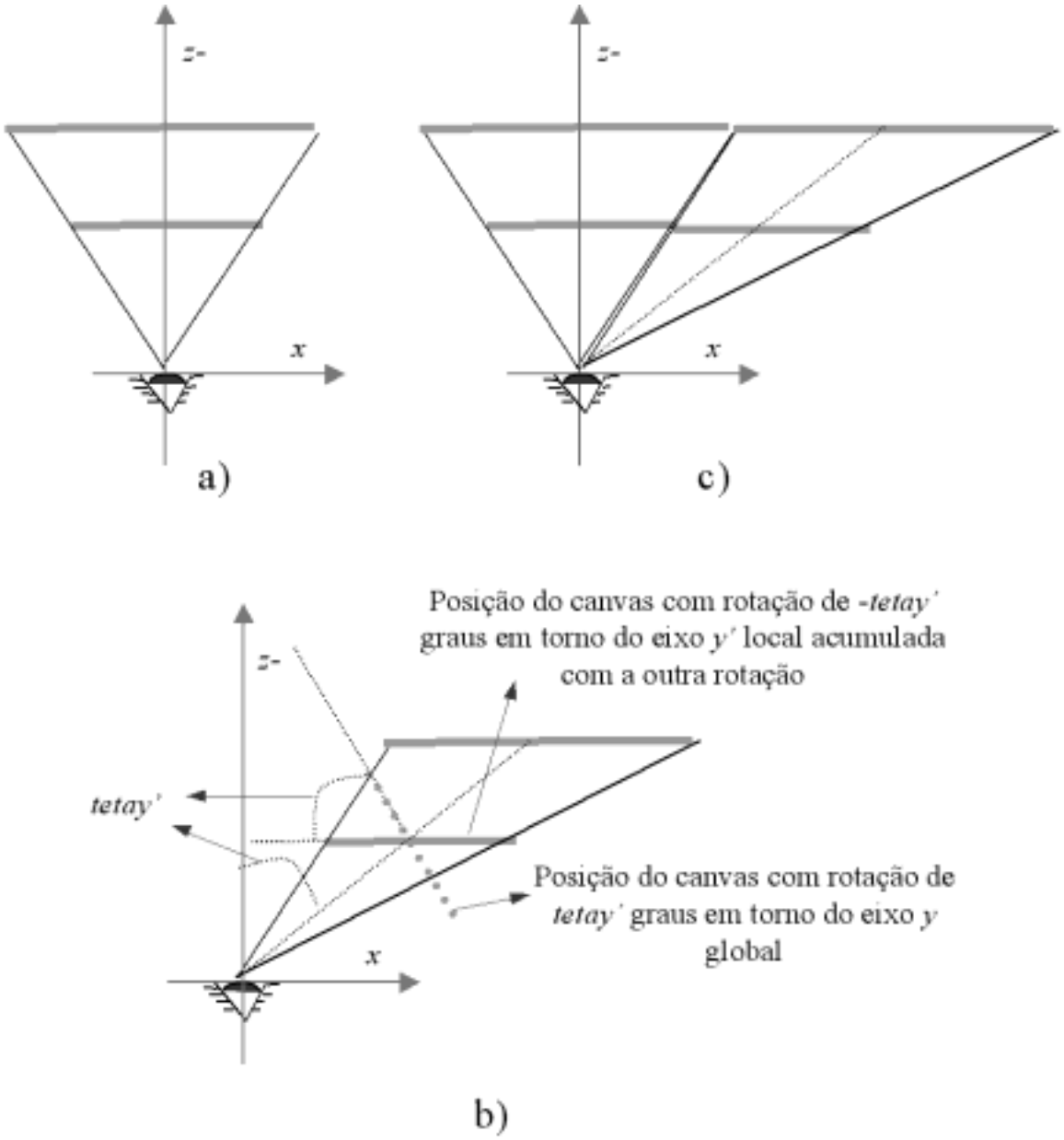

Figura 23 – Distorção do volume de visão para os *canvases* laterais

O próximo passo é a construção da matriz de modelo, que vai conter o conjunto de translações e rotações que levam a câmara virtual para a posição do observador. Essa matriz é obtida a partir da composição das seguintes transformações.

A matriz identidade inicialmente carregada é multiplicada pela matriz $\boldsymbol{M_c^{-1}}$, de forma a fazer com que o observador seja girado para a posição do *canvas.* A partir daí são

acumuladas as matrizes de posicionamento do observador (T_o e Q_o) e da mecânica ($T_{mo}(t)$ e $Q_{mo}(t)$) da entidade à qual ele está ligado.

O encadeamento resultante fica da seguinte forma:

- Matriz de projeção: $P\ SH$
- Matriz de modelo: $M_c^{-1}\ T_o^{-1}\ Q_o^{-1}\ T_{mo}(t)^{-1}\ Q_{mo}(t)^{-1}$

Posicionamento das entidades

Esta fase consiste basicamente em posicionar todas as entidades no cenário. Isso é feito por meio de um conjunto de matrizes que são acumuladas à matriz de modelo gerada no estágio anterior.

O posicionamento espacial de uma determinada entidade ***i*** é feito da seguinte forma: multiplica-se a matriz de modelo corrente pela translação e a rotação ($T_{mi}(t)$ e $Q_{mi}(t)$) decorrentes da mecânica da entidade. Em seguida, multiplica-se a matriz resultante pelas matrizes (T_{si}, Q_{si}, e S_{si}) de adaptação da descrição geométrica da entidade ao espaço do ambiente virtual.

Resumindo, o desenho de cada entidade ***i*** é feito pelo OpenGL através da matriz:

- Matriz de modelo: $(M_c^{-1}\ T_o^{-1}\ Q_o^{-1}\ T_{mo}(t)^{-1}\ Q_{mo}(t)^{-1})\ T_{mi}(t)\ Q_{mi}(t)\ S_{si}\ Q_{si}^{-1}\ T_{si}^{-1}$, sendo a porção $(M_c^{-1}\ T_o^{-1}\ Q_o^{-1}\ T_{mo}(t)^{-1}\ Q_{mo}(t)^{-1})$ comum a todas as entidades durante o processo de renderização do ambiente virtual em determinado instante ***t***.

4.3 - Mediador do sistema e Mundo externo

A implementação realizada para testes limitou-se ao módulo de coordenação e a implementação parcial do módulo de acoplamento, especificamente a interface do mediador com as UVs. A outra interface do módulo de acoplamento foi substituída, visando a realização de testes, por um módulo de interface com o usuário que cumpre o papel do mundo externo. Esse módulo consiste em um *console*, configurado para receber programas escritos em Lua. Sua utilização permite simular com facilidade diversos contextos de mundo externo na qual o sistema de visualização possa vir a ser acoplado.

O *console* implementado também é dotado de uma interface especial para a captura direta de dados provenientes do usuário. A interface especial é destinada ao controle do comportamento de determinada entidade à qual ela esteja associada. Sua utilização simula

uma aplicação externa que tenha como objetivo inspecionar o ambiente virtual por meio de interações de teclado. Considerando, por exemplo, que a tecla ↑ (seta para cima) do teclado seja pressionada, a interface especial emite um comando de alteração do comportamento da entidade, solicitando que ela se desloque de uma distância ***ds*** para a frente ou que sua velocidade para a frente seja incrementada de um valor ***dv***, dependendo do tipo de mecânica utilizada.

Para essa interface especial foram programadas as ***teclas ↑, ↓, ←, →, alt+↑, alt+↓, alt+←, alt+→, shift+↑, shift+↓, shift+← e shift+→,*** que significam, para o observador na posição canônica, transladar (na direção de) (***z***-), transladar: (***z***), rodar (na direção horária em torno de) (***y***-), rodar (***y***), rodar (***x***), rodar (-***x***), rodar(***z***-), rodar (***z***), transladar (***y***), transladar (***y***-), transladar (***x***-) e transladar (***x***), respectivamente.

Na programação desta interface são levados em consideração atributos que definem o incremento linear, o incremento angular, o incremento na velocidade linear e o incremento na velocidade angular, conforme ilustrado no extrato de código Lua usado para defini-la.

```
keyb = Keyboard_Input_Device{ name = "Keyboard 1",
                              active = 1,
                              entity = ent1,
                              linear_step = 1,
                              linear_speed_step = 1,
                              algular_step = 1,
                              algular_speed_step = 1, }
```

4.4 - Resultados e experimentos

Foram realizados alguns testes com a finalidade de julgar o comportamento do sistema em situações nas quais ele possa vir a ser utilizado.

O objetivo dos testes é determinar a adequação da implementação com os requisitos apresentados pela arquitetura. Para isso, foram elaboradas algumas questões a serem respondidas:

- O modelo de posicionamento utilizado para a definição do *canvas* produz resultados corretos?
- O sincronismo entre as máquinas é suficiente para evitar falhas no resultado visual apresentado?
- O uso da técnica de múltiplos níveis de detalhe implicou em ganhos efetivos em termos de desempenho da UV?
- A estrutura hierárquica permite a modelagem de comportamentos elaborados?

4.4.1 - Teste da modelagem do *canvas*

O teste da modelagem do *canvas* foi realizado visando verificar se a formulação para o seu posicionamento alcança os resultados desejados.

Para isso foi utilizado um único computador rodando todo o sistema. Isso não constitui nenhuma perda de generalidade, pois toda a infra-estrutura de comunicação foi usada, e o que se deseja verificar é se duas ou mais UVs com seus *canvases* deslocados fornecem o resultado desejado. A imagem produzida por cada UV é exibida em uma janela. Quando as janelas são posicionadas lado a lado, se consegue uma condição semelhante à de dois monitores lado a lado.

No primeiro teste feito buscou-se recriar uma condição semelhante à ilustrada na Figura 23c, em que dois *canvases* são configurados de forma a ficar lado a lado, e no segundo teste os *canvases* foram posicionados um sobre o outro. Para isso, três configurações de *canvas* foram criadas contendo os seguintes parâmetros:

```
Canvas{  name = "Canvas Central",
         distance = 0.2, width = 0.097, height = 0.073,
         rotation = Quadruple{0,0,1,0},
         windowsize_x = 400, windowsize_y = 300,
         teta_x = 0, teta_y = 0, }

Canvas{  name = "Canvas Esquerdo",
         distance = 0.22361955, width = 0.097, height = 0.073,
         rotation = Quadruple{ 26.865051,0,1,0},
         windowsize_x = 400, windowsize_y = 300,
         teta_x = 0, teta_y = -26.865051, }

Canvas{  name = "Canvas Inferior",
         distance = 0.21325103, width = 0.097, height = 0.073,
         rotation = Quadruple { -20.198876,1,0,0},
         windowsize_x = 400, windowsize_y = 300,
         teta_x = -20.198876, teta_y = 0, }
```

Foram tiradas cópias da tela do computador, com as quais foram geradas as imagens a seguir. A Figura 24 mostra a entidade utilizada, um retângulo texturizado com uma imagem de uma grade, sendo totalmente visualizada no Canvas Central. A Figura 25 ilustra a mesma entidade sendo parcialmente visualizada pelo Canvas Central e pelo Canvas Esquerdo. Finalmente, a Figura 26 ilustra a mesma entidade sendo visualizada pelo Canvas Central e pelo Canvas Inferior.

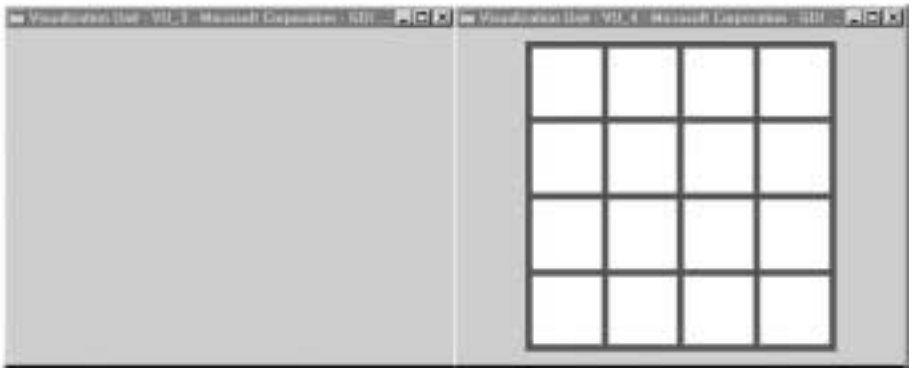

Figura 24 – Entidade totalmente contida no Canvas Central

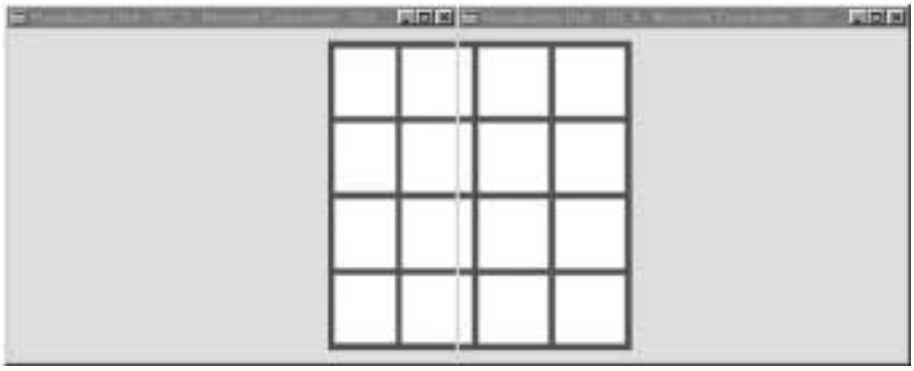

Figura 25 – Entidade dividida entre o Canvas Central e o Canvas Esquerdo

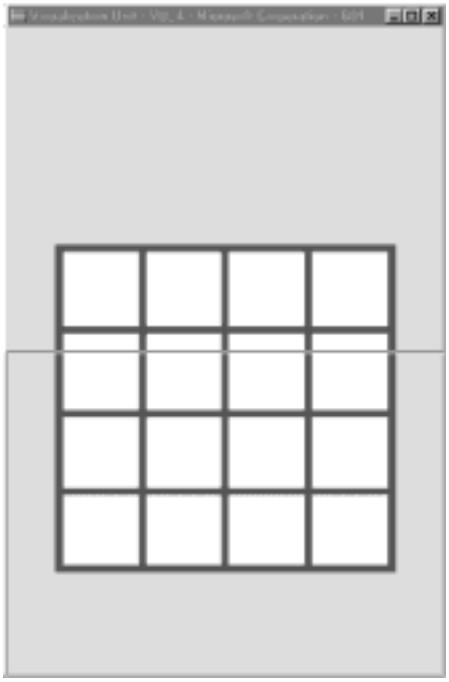

Figura 26 – Entidade dividida entre o Canvas Central e o Canvas Inferior

O resultado do teste mostrou-se satisfatório, uma vez que não houve distorções perceptíveis para os parâmetros utilizados.

4.4.2 - Teste do sincronismo das UVs

O teste do sincronismo das UVs foi realizado para verificar se a técnica de sincronismo adotada apresenta resultados satisfatórios e se é possível perceber distorções na continuidade da cena.

Uma vez que o principal parâmetro para a avaliação deste sistema é o resultado visual por ele fornecido, decidiu-se basear este teste na análise de resultados obtidos através de uma gravação em vídeo.

Para a confecção do vídeo foram utilizados dois computadores ligados em uma rede local. O primeiro, localizado à esquerda das imagens apresentadas a seguir é um Pentium III 400 Mhz, Windows NT, com placa de vídeo Viper 770 e recursos de aceleração gráfica para OpenGL em *hardware*. O segundo computador, localizado à direita, é um Pentium II 350 Mhz, Windows 98, com placa de vídeo Viper 550, que também possui recursos de aceleração gráfica para OpenGL em *hardware*. O primeiro computador foi utilizado exclusivamente como UV, e o segundo, além de funcionar como UV, foi responsável pela execução do mediador e da aplicação externa que alimenta o sistema.

O ambiente virtual sendo visualizado é composto por uma entidade cuja forma geométrica é um quadrado simples posicionado em frente ao observador no espaço 3D. Sua aparência é obtida pela aplicação sucessiva de texturas numeradas de 0 a 9, que são trocadas em intervalos de 0.2 segundos.

O vídeo foi gravado usando uma câmara VHS posicionada em frente aos monitores sobre um tripé, sendo em seguida digitalizado com uma taxa de 30 quadros por segundo. A Figura 27 apresenta um quadro do vídeo no qual a entidade está sendo inteiramente renderizada na UV da esquerda. A Figura 28 apresenta 48 quadros que correspondem a cerca de 1.5 segundos do vídeo nos quais a entidade está sendo renderizada pelas duas UVs, durante a sua transição de uma UV para outra. Já a Figura 29 apresenta o quadro no qual a entidade está sendo totalmente renderizada pela UV da direita. O teste consiste, basicamente, em verificar se a textura apresentada em cada UV é a mesma em um determinado instante no tempo.

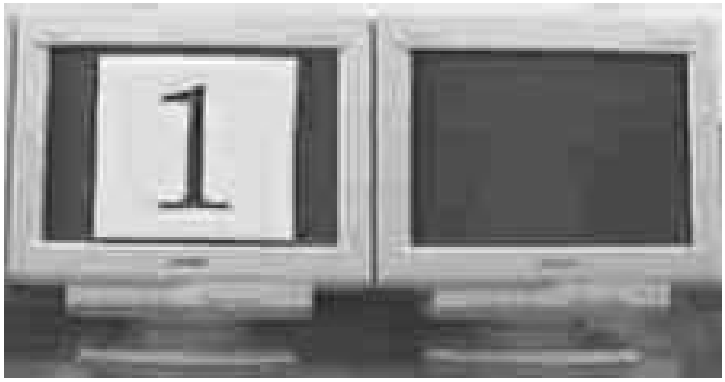

Figura 27 – Quadro que contém a entidade na UV da esquerda

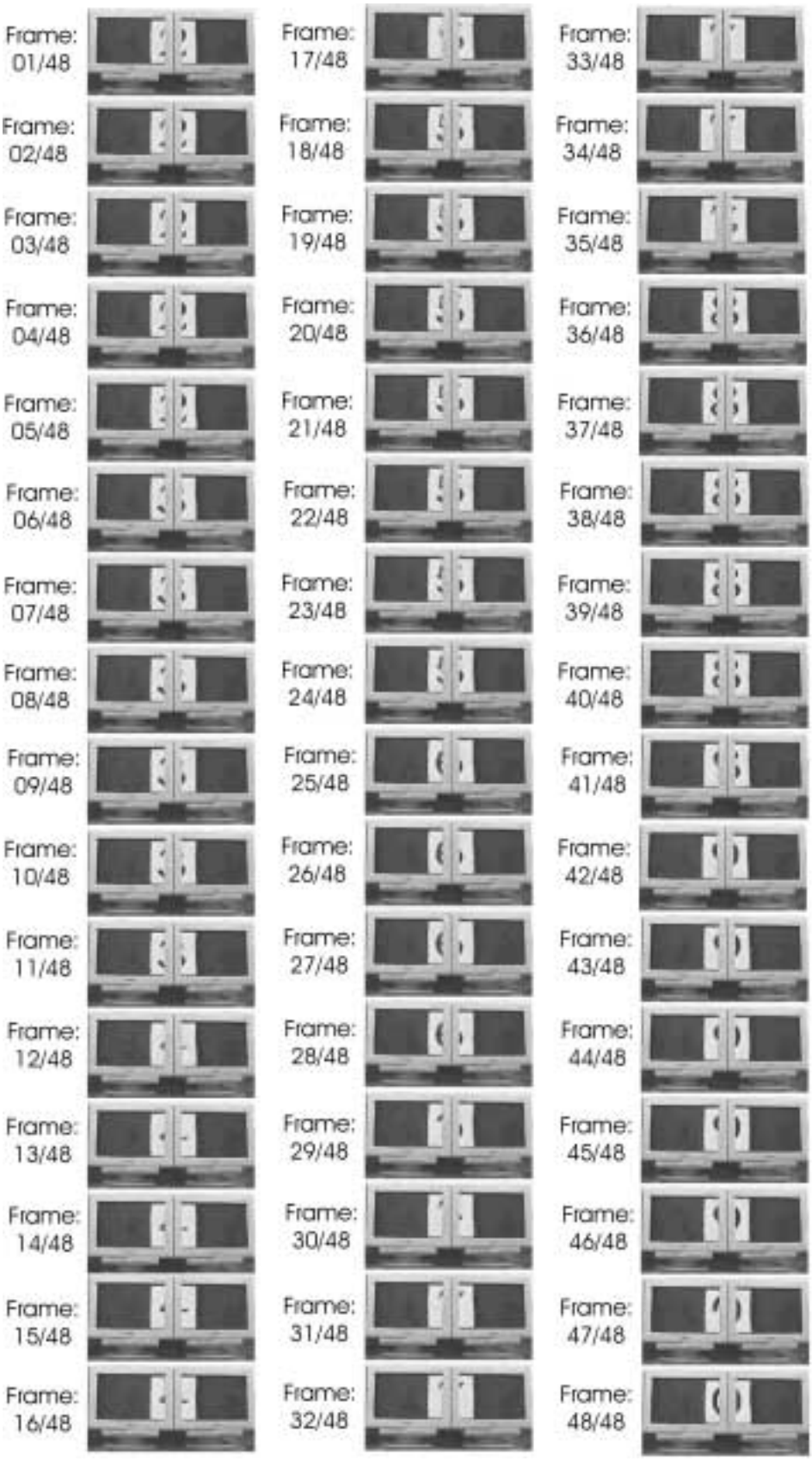

Figura 28 – Seqüência de 48 quadros do vídeo de teste de sincronismo

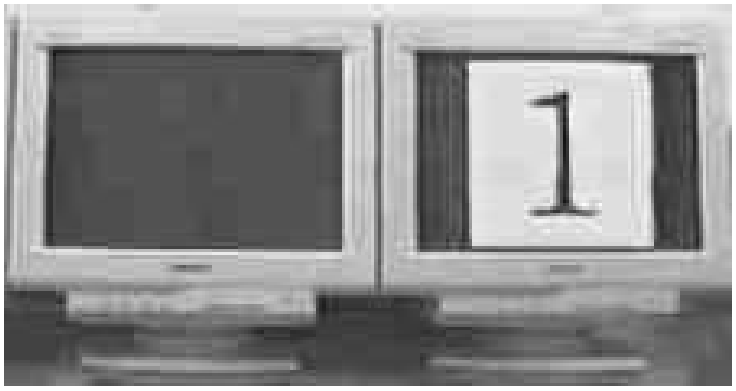

Figura 29 – Quadro que contém a entidade na UV da direita

O resultado que pode ser extraído da análise das figuras é que, mesmo sendo utilizadas máquinas com poder de processamento diferente e com sistemas operacionais diferentes, as pequenas variações encontradas nos quadros 17 e 41 são praticamente imperceptíveis, sendo possivelmente devido à da falta de sincronismo entre a câmara e a varredura dos monitores, fato que pode ser identificado nos quadros 29, 35 e 47, onde as imagens capturadas mostram quadros sendo parcialmente desenhados.

4.4.3 - Teste de desempenho

Este teste tem o objetivo de avaliar o ganho de desempenho decorrente do uso das técnicas de eliminação de entidades posicionadas fora do volume de visão e do uso de múltiplos níveis de detalhe na descrição geométrica das entidades.

O computador utilizado para o teste foi um Pentium II 350 Mhz, Windows 98, com placa de vídeo Viper 550, que dispõe de recursos de aceleração gráfica para OpenGL em *hardware*.

Foi usada a modelagem de um ambiente virtual habitado por um conjunto de entidades contendo descrições geométricas mais detalhadas. O ambiente virtual foi composto por dois navios, dois helicópteros, duas ilhas, o céu, o mar e um avião, contendo respectivamente 5977, 6448, 998, 8, 8 e 4428 polígonos cada entidade, totalizando 31290 polígonos no ambiente virtual.

Em seguida, o ambiente virtual foi visualizado a partir de dois pontos de vista diferentes, o primeiro em uma posição próxima às entidades e o segundo em uma posição afastada. Para ambos os pontos de vista foram geradas duas imagens, uma utilizando os recursos de simplificação por eliminação e por níveis de detalhe e a outra com estas opções desabilitadas. As imagens geradas são apresentadas na Figura 30 a seguir.

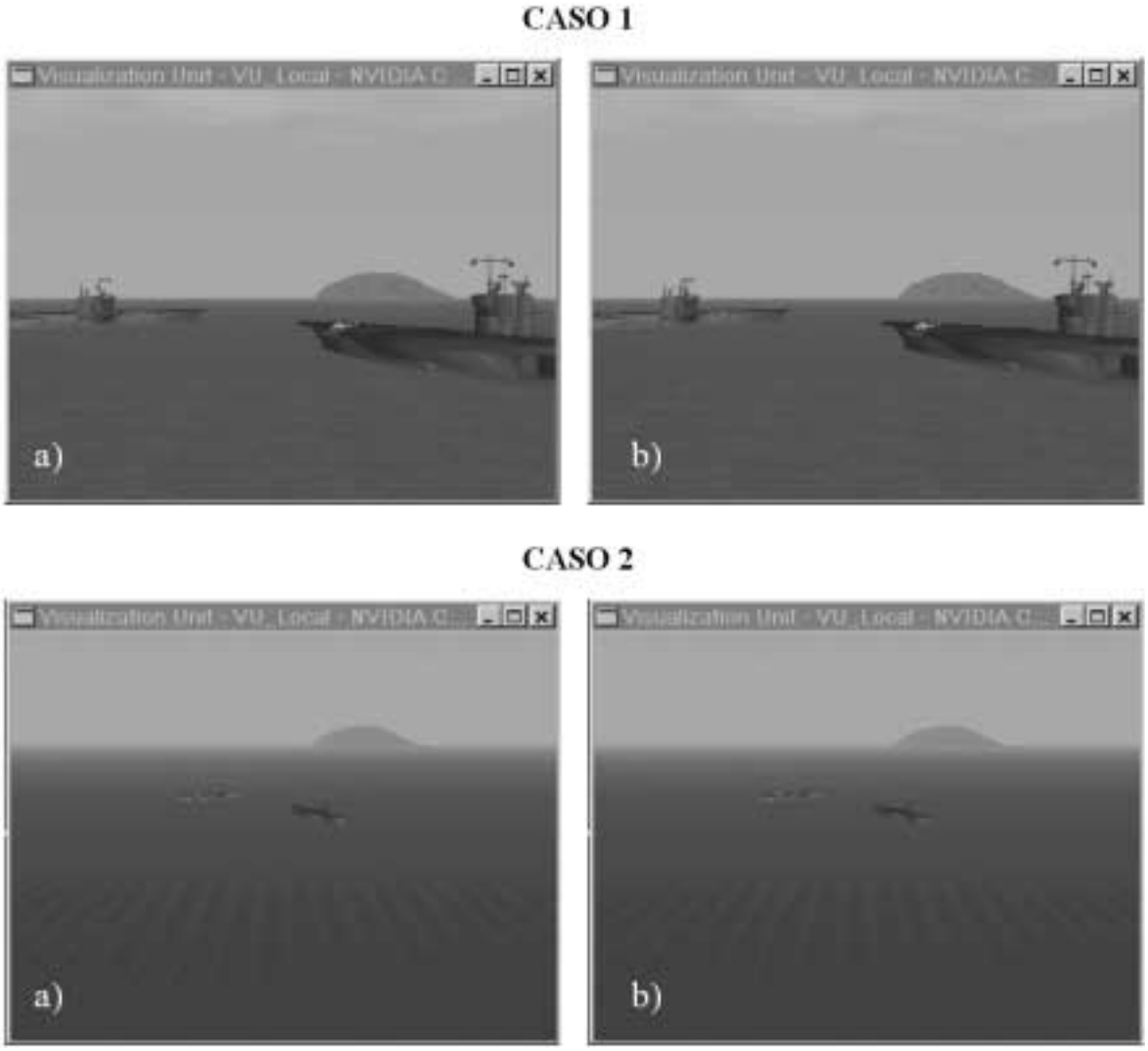

Figura 30 – Teste de desempenho e qualidade visual do sistema

Como resultado obteve-se:

Caso	Número de Polígonos	*Frame-rate*
1a	12197	8.18 fps
1b	31290	3.15 fps

Caso	Número de Polígonos	*Frame-rate*
2a	3744	26.92 fps
2b	31290	2.73 fps

Tabela 1 – Resultados obtidos nos testes de desempenho da UV

Comparando a qualidade visual das cenas renderizadas nos casos ilustrados na Figura 30, verifica-se que as diferenças de qualidade entre elas são pequenas em relação aos ganhos de desempenho alcançados, justificando dessa maneira a utilização dos recursos de aumento de desempenho indicados pela arquitetura.

4.4.4 - Teste de modelagem de comportamentos elaborados

Este teste consiste em mostrar como é possível fazer a modelagem de comportamentos elaborados utilizando a estruturação das componentes implementada.

Para isso será apresentado a seguir um extrato de código Lua utilizado na descrição de um cenário contendo um helicóptero estacionado sobre o convés de um navio. O navio possui um movimento de translação em relação ao mar, bem como um movimento de oscilação simulando o efeito de seu avanço sobre as ondas do mar. O rotor do helicóptero estacionado sobre o convés encontra-se em movimento circular.

```
app1 = Empty_Appearance{ name = "Transp" }

app1 = Material_Appearance{ name = "Steel", diffuse = Vector{0.5, 0.5, 0.5},
                            specular = Vector{1.0, 1.0, 1.0}, shininess = 1,}

shp0 = Empty_Shape{ name = "EmptyShape", }

shp1 = MeshPly_Shape{ name = "HelixShape",
                      filename = ".\\data\\shapes\\nff_to_ply\\a64rot.ply",
                      axis_position = Vector{0,0,0},
                      axis_rotation = Quadruple{-90.0, 1.0, 0.0, 0.0},
                      scale = 0.1, auto_center = true,
                      lod_criteria = EDGE_LENGTH + PRESERVE_BOUNDARY,
                      bounding_box = FALSE, test_clip = FALSE, }

shp2 = MeshPly_Shape{ name = "BodyShape",
                      filename = ".\\data\\shapes\\nff_to_ply\\a64body.ply",
                      axis_position = Vector{0,0,0},
                      axis_rotation = Quadruple{180.0, 0.0, 1.0, 0.0},
                      scale = 0.1, auto_center = true,
                      lod_criteria = EDGE_LENGTH + PRESERVE_BOUNDARY,
                      bounding_box = FALSE, test_clip = FALSE, }

shp3 = MeshPly_Shape{ name = "Ship",
                      filename = ".\\data\\shapes\\saratoga.ply",
                      axis_rotation = Quadruple{90.0, 1.0, 0.0, 0.0},
                      scale = 1.338, auto_center = TRUE,
                      lod_criteria = EDGE_LENGTH, bounding_box = FALSE, }

sta0 = Static_Mechanic{ name = "ShipMec",
                        position = Vector{0.0, 0.0, 0.0},
                        rotation = Quadruple{0,1,0,0}, }

sta1 = Static_Mechanic{ name = "BodyMec",
                        position = Vector{0.0, 0.0, 0.0},
                        rotation = Quadruple{0,1,0,0}, }

kin1 = Kinematic_Mechanic{ name = "HelixMec",
                           position = Vector{0.0, 3.0, -1.0},
                           rotation = Quadruple{0,1,0,0},
                           angular_speed = Vector{0.0, 0.0, 500.0}, }

kin2 = Kinematic_Mechanic { name = "A64Mec",
                            position = Vector{-3.2, 0.0, -3.2},
                            rotation = Quadruple{30.0, 0.0, 0.1, 0.0}, }

kin3 = Kinematic_Mechanic { name = "ShipKinematicMec",
                            position = Vector{7.0, 1.7, 700.0},
                            rotation = Quadruple{20.0, 0.0, 1.0, 0.0},
                            linear_speed = Vector{-0.045, 0.0, -0.15},
                            actual_time = 0, }

osc1 = Oscillation_Mechanic{ name = "ShipOscillationMec",
                             position = Vector{0.0, 0.0, 0.0},
                             rotation = Quadruple{0,0,1,0},
                             actual_time = 0, rotation_period = 10,
```

```
                              rotation_size = 1,
                              rotation_direction = Vector{1,0,0}, }

ent1 = Entity{ name = "Helix",
               shape = shp1, appearance = app1, mechanics = kin1, }

ent2 = Entity{ name = "Body",
               shape = shp2, appearance = app1, mechanics = sta1, }

ent3 = Entity_Group{ name = "A64",
                     shape = shp0, appearance = app0, mechanics = kin2;
                     ent1, ent2, }

ent4 = Entity{ name = "Ship",
               shape = shp3, appearance = app1, mechanics = sta0, }

ent5 = Entity_Group{ name = "Ship+A64",
                     shape = shp0, appearance = app0, mechanics = osc1;
                     ent4, ent3, }

ent6 = Entity_Group{ name = "Ship+A64(withOsc)",
                     shape = shp0, appearance = app0, mechanics = kin3;
                     ent5, }

light1 = Positional_Light{ name = "Positional 1",
                           intensity = Vector{0.5, 0.5, 0.5},
                           position = Vector{}, entity = ent1, }

scen = Scene{   name = "Scene",
                back_color = Vector{128/255, 183/255, 223/255},
                fog = FALSE;
                light1, ent6, }
```

Seria possível fazer com que o helicóptero decolasse do navio. Isso pode ser feito através das instruções indicadas abaixo. Inicialmente o helicóptero é retirado do grupo que o contém e é inserido diretamente na lista das entidades do cenário como uma entidade independente. Em seguida sua mecânica seria alterada de forma a indicar a rota de vôo.

```
ent5:remove(ent3)

scen:insert(ent3)

ent3:mechanics():sett(Vector{7.0,1.7,700.0}, Vector{-0.045, 10.0, -0.15}, 2.0)
ent3:mechanics():setr(Quadruple{20.0, 0.0, 1.0, 0.0}, Vector{0, 0, 0}, 2.0)
```

4.4.5 - Exemplos

A seguir são apresentados alguns exemplos de ambientes virtuais sendo visualizados pelo sistema de visualização proposto.

Figura 31 – Exemplo de terreno texturizado

Neste exemplo, ilustrado na Figura 31, um terreno sintético texturizado, contendo cerca de 2000 triângulos, está sendo visualizado em um arranjo formado por três UVs.

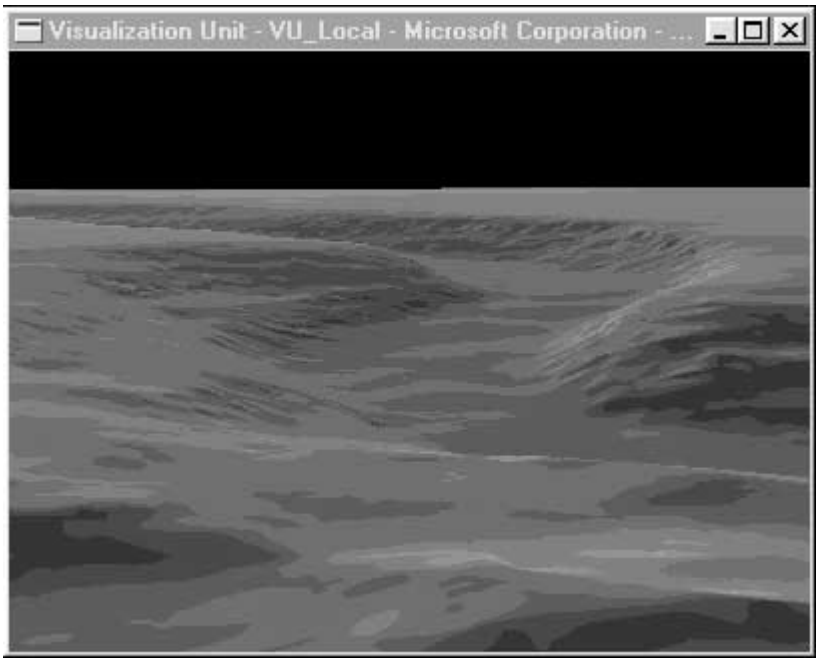

Figura 32 – Exemplo de um terreno texturizado de uma superfície no fundo do mar

Neste exemplo, ilustrado na Figura 32, outro terreno, modelado a partir de dados reais do fundo do mar e contendo cerca de 5000 triângulos é visualizado em uma UV.

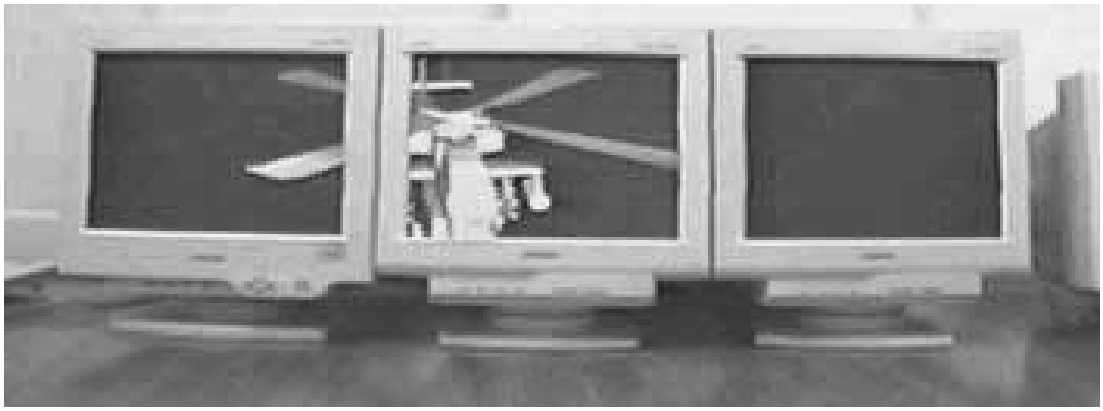

Figura 33 – Exemplo de um helicóptero animado

Neste exemplo, ilustrado na Figura 33, um helicóptero contendo cerca de 8000 triângulos é visualizado em um arranjo formado por três UVs. É interessante realçar que tal imagem é um quadro extraído de um vídeo no qual o rotor do helicóptero se encontrava em movimento circular. Dessa forma, mais uma vez, é ilustrada a precisão do sincronismo alcançado.

CAPÍTULO 5 – CONCLUSÕES E PROPOSTA DE TRABALHOS FUTUROS

Apresentar conclusões baseadas nos testes práticos realizados e sugerir possíveis linhas de pesquisa relacionadas ao tema estudado.

5.1 - Conclusões

Este trabalho apresentou uma proposta de arquitetura para o desenvolvimento de um sistema de visualização de ambientes virtuais por múltiplos dispositivos de visualização.

Discutiu-se inicialmente alguns conceitos relevantes com relação ao problema da visualização de ambientes virtuais, para o qual são traçados alguns requisitos quando da escolha de uma abordagem distribuída. Em seguida é apresentada a arquitetura proposta, sendo indicadas as soluções tecnológicas para os diversos problemas existentes. Para finalizar, é apresentada uma implementação do sistema, sobre a qual são aplicados alguns testes que permitem a validação do emprego da arquitetura em casos práticos.

Os estudos realizados e o desenvolvimento do trabalho mostram, com relação aos requisitos apresentados, que:

- o sincronismo atingido entre os relógios das máquinas integrantes do sistema é suficiente, uma vez que os erros não influenciam a qualidade visual apresentada e não se constituem artifícios visuais que desviem a atenção ou diminuam o grau de imersão do usuário;

- as técnicas de manutenção do estado dinâmico corrente são suficientes para manter a integridade do ambiente virtual visualizado, impedindo que UVs diferentes apresentem visões inconsistentes, fato que comprometeria a utilização do sistema de visualização proposto;
- o sistema funciona em redes heterogêneas, uma vez que podem ser utilizados computadores com níveis de desempenho diferentes, sendo extraído o máximo proveito em termos dos recursos computacionais de cada um;
- o uso de estações de trabalho *low-end* e a abordagem distribuída minimizam custos quando do aumento do número de dispositivos de visualização, em comparação com uma abordagem centralizada;
- a arquitetura apresenta características de portabilidade, uma vez que indica a utilização de bibliotecas e APIs que podem ser compiladas e utilizadas em diversas plataformas de *software* e *hardware*;
- a estrutura proposta pela arquitetura apresenta um bom nível de reuso uma vez que ela foi desenvolvida para lidar com diferentes ambientes virtuais sem a necessidade de alterações no núcleo do sistema de visualização do sistema.

Com relação à estruturação e à modelagem empregadas e às decisões tecnológicas adotadas, o desenvolvimento do trabalho e os testes mostram que:

- o uso da técnica de *dead-reackoning* é favorável, uma vez que minimiza o tráfego de mensagens enviadas pela rede;
- a estruturação hierárquica das entidades permite a modelagem de comportamentos elaborados;
- o modelo utilizado para a definição dos parâmetros de mapeamento dos dispositivos de visualização nos *canvases* fornece um bom grau de liberdade, permitindo uma grande variedade de configurações em relação ao seu posicionamento;
- a infra-estrutura de comunicação proposta possui um alto nível de abstração, o que facilita a implementação e a configuração do sistema, além de permitir sua implantação sobre diferentes esquemas de distribuição e protocolos de rede;

- a modelagem utilizada é genérica, extensível e clara, principalmente em decorrência da utilização de uma linguagem interpretada para a descrição e a manutenção dos diversos tipos de ambientes virtuais;
- a arquitetura proposta pode atingir níveis adequados de desempenho no processamento gráfico realizado em cada UV caso seja feito uso de algoritmos de eliminação de entidades e múltiplos níveis de detalhe.

Os fatores acima apresentados, em conjunto com a crescente demanda por sistemas de visualização desta natureza, implicam na existência de vários usos em potencial para essa tecnologia por parte de diversas aplicações reais existentes.

5.2 - Trabalhos Futuros

Uma vez que a arquitetura se destina a servir como arcabouço para o desenvolvimento de um sistema de visualização distribuída, existem diversas linhas de estudo que estão individualmente ligadas às diversas técnicas e soluções propostas.

Entretanto, é possível delinear algumas extensões possíveis e elementos que demandam uma análise mais aprofundada com relação à sua utilização.

Podem ser iniciados estudos relacionados à inclusão de recursos de entrada e manipulação direta de entidades do ambiente virtual, sem que estes estejam limitados ao escopo da aplicação existente no mundo externo.

Podem ser elaboradas novas componentes destinadas ao fornecimento de outros resultados sensoriais ao usuário. A inclusão de atributos relacionados ao estímulo auditivo representa um exemplo de extensão possível.

A arquitetura pode ser especializada para ser empregada no desenvolvimento de sistemas conhecidos como paredes de dados (DataWall), em que uma grande tela composta pela imagem gerada por diversos projetores é usada como área de trabalho por um grupo de pessoas trabalhando cooperativamente.

Podem ser estudadas soluções de *software* que permitam aplicar a arquitetura a configurações físicas em que projeções oriundas de diferentes UVs tenham um certo grau de sobreposição na tela de projeção, a fim de permitir um melhor grau de acoplamento e continuidade na imagem final gerada.

A estrutura de posicionamento, modelagem e renderização do cenário está centrada em torno da posição do observador. Isso implica que outras pessoas presentes no mesmo

espaço físico onde o sistema esteja sendo utilizado podem perceber a imagem gerada de forma deslocada ou distorcida. Seria então interessante o estudo de técnicas que permitam que as imagens sejam compreensíveis de diferentes pontos de vista.

Também podem ser realizados testes mais aprofundados com relação à escalabilidade do sistema. Para isso seria necessária uma maior quantidade de recursos, uma vez que testes desta natureza demandam a disponibilidade de grande quantidade de *hardware* adequado.

Finalmente, a implementação do sistema poderia ser estendida para tratar uma maior gama de formatos de descrição geométrica e de definições de aparência, bem como dar suporte a outros tipos de comportamentos.

BIBLIOGRAFIA

[ADAMS96] Adams, E. & Doherty, D. *Moving Worlds*. Prima Publishing, 1996.

[AIREY90] Airey, J. M.; Rohlf, J. H. & Brooks, F. P. "Towards image realism with interactive update rates in complex virtual buildings environments". *Computer Graphics* 24(2):41-50, March 1990.

[AUKSTAKALNIS92] Aukstakalnis, S. & Blatner, D. *Silicon Mirage: The Art and Science of Virtual Reality*. Peach Pit Press, 1992, ISBN 0-938151-92-7.

[BROOKS86] Brooks, F. P. "A dynamics graphics system for simulating virtual buildings". *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, 9-21, University of North Carolina at Chapel Hill, October 1986.

[BURCHFIEL90] Burchfiel, J. "The Advantages of Using Quaternions Instead of Euler Angles for Representing Orientation". *White Paper ASD-91-001*, Third Workshop on Standard for the Interoperability of Defense Simulations, Orlando FL., August 1990.

[CERQUEIRA97] Cerqueira, R.; Ierusalimschy, R. & Rodriguez, N. "A Dynamic Approach for composing CORBA Objects". *Techinical Report* 43/97, PUC-Rio, Rio de Janeiro, Brazil, 1997.

[CERQUEIRA99] Cerqueira, R.; Cassino, C. & Ierusalimschy, R. "Dynamic Componet Gluing Across Different Componentware Systems". *International Symposium on Distributed Objects and Applications (DOA'99)*, Edinburgh, Scotland, IEEE Press, 1999.

[CLARK76] Clark, J. “Hierarchical geometric models for visible surface algorithms”. *Communications of the ACM* 19(10):547-554, October 1976.

[DURLACH95] Durlach, N. I. & Mavor, A. S., eds. *Virtual Reality – Scientific and Technological Challenges*. National Academy Press 1995.

[FOLEY90] Foley, J. D.; van Dam, A.; Feiner, S. K. & Hughes, J. F. *Computer Graphics: Principles and Practice*. Addison-Wesley, New York, 2nd Edn, 216, 1990.

[FUCHS80] Fuchs, H.; Kedem, Z. & Naylor, B. “On visible surface generation by a priori tree structures”. *Proceedings of SIGGRAPH 1980*, 124-133, Summer 1980.

[FUCHS83] Fuchs, H.; Anram, G. & Grant, E. “Near real-time shaded display of rigid objects”. *Proceedings of SIGGRAPH 1983*, 65-72, Summer 1983.

[GAMMA95] Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[HAMILTON66] Hamilton, W. R. *Elements of Quaternions*. London: Longmans, Green, 1866.

[HOPPE96] Hoppe, H. “Progressive Meshes”. *Proceedings of SIGGRAPH 1996*, 99-108, Summer 1996.

[IEEE93] “IEEE standard for information technology-protocols for distributed simulation application: Entity information and interaction”. IEEE Standard 1278-1993. New York: IEEE Computer Society, 1993.

[IERUSALIMSCHY96] Ierusalimschy, R.; Figueiredo, L. H. & Celes, W. “Lua: an extensible extension language”. *Software: Practice & Experience*, 26 (6), pp. 635-652, 1996.

[KATZ94] Katz, A. & Graham, K. “Dead Reckoning for airplanes in coordinated flight”. *Proceedings of the Tenth Workshop on Standards for the Interoperability of Defese Simulations*, II:5-13. Orlando, FL, 1994.

[LEVY96] Levy, C. H.; Figueiredo, L. H.; Gattass, M.; Lucena, C. & Cowan, D. “IUP/LED: a portable user interface development tool”. *Software: Practice & Experience* 26 #7, 737-762, 1996.

[LOEFFLER94] Loeffler, C. E. & Anderson, T. “The Virtual Reality Casebook”. Van Nostrand Reinhold, 1994 - ISBN 0-442-01776-6.

[LUEKEBKE97] Luekebke, D. “A survey of Polygonal Simplification Algorithms”. *UNC Technical Report* TR97-045 (1997).

[MELAX98] Melax, S. “A Simple, Fast, and Effective Polygon Reduction Algorithm”. *Game Developer Magazine*, November 1998.

[MILLS92] Mills, D. “Network Time Protocol (Version 3): Specification, Implementation, and Analysis”. Internet RFC 1305, March 1992.

[MILLS96] Mills, D. “Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI”. Internet RFC 2030, October 1996.

[MINE96] Mine, M. & Weber, H. “Large models for virtual environments: A review of work by architectural walkthrough project at UNC”. *PRESENCE: Teleoperators and Virtual Environments* 5(1):136-145, Winter 1996.

[OMG98] “The Common Object Request Broker Architecture and Specification”. Revision 2.2, OMG, Framingham, MA, February 1998.

[PHONG75] Phong, B. “Illumination for Computer Generated Pictures”. *CACM*, 18(6), June 1975.

[POSTEL80] Postel, J. “User datagram protocol”. Internet RFC 768, Information Sciences Institute, Marina Del Rey, CA, October 1980.

[POSTEL81a] Postel, J. “Internet Protocol: DARPA Internet program protocol specification”. Internet RFC 791, Information Sciences Institute, Marina Del Rey, CA, September 1981.

[POSTEL81b] Postel, J. “Transmission control protocol: DARPA Internet program protocol specification”. Internet RFC 793, Information Sciences Institute, Marina Del Rey, CA, September 1981.

[SEGAL97] Segal, M. & Akeley, K. “The OpenGL Graphics System: A Specification (Version 1.1)”. Silicon Graphics Inc. March 1997.

[SIEGEL96] Siegel, J. *CORBA Fundamentals and Programming*. John Wiley & Sons, 1996.

[SINGHAL95] Singhal, S. & Cheriton, D. R. “Exploiting position history for efficient remote rendering in networked virtual reality”. *Presence: Teleoperators and Virtual Environments* 4(2):169-193, Spring 1995.

[SINGHAL99] Singhal, S. & Zyda, M. *Networked Virtual Environments*. ACM Press, 1999.

[STEVENS90] Stevens, W. *Unix Network Programming*. Prentice-Hall, 1990.

[WILLIAMS83] Williams, L. “Pyramidal Parametrics”, *Proceedings of SIGGRAPH 1983*, 1-11, Summer 1983.

REFERÊNCIAS NA INTERNET

[AVIARY] AVIARY Web site: http://www.crg.cs.nott.ac.uk/~dns/vr/aviary/

[BARREAU93] Barreau, D. & McGoff, K. “Immersion”, http://www.hitl.washington.edu/scivw/EVE/III.C.1.Immersion.html, Department of Computer Science, Human Interface Technology Laboratory, University of Maryland, 1993.

[DIVE] DIVE Web site: http://www.sics.se/dce/dive/dive.html

[ISDALE93] Isdale, J. “What is Virtual Reality – A Homebrew Introduction and Information Resource List”. http://www.cms.dmu.ac.uk/ ~cph/VR/whatisvr.html. 1993

[LUA] Lua Web Site: http://www.tecgraf.puc-rio.br/lua

[LUAORB] LuaOrb Web site: http://www.tecgraf.puc-rio.br/luaorb

[MATTOS99] MLOD Web site: http://www.tecgraf.puc-rio.br/~pmattos/mlod

[MR] MR Toolkit Web site: http://web.cs.ualberta.ca/~graphics/MRToolkit.html

[MUSSER94] Musser, D. "The Standard Template Library", Rensselaer Polytechnic Institute, 1994. http://www.cs.rpi.edu/~musser/stl.html

[NPSNET] NPSNET Web site: http://www.npsnet.nps.navy.mil/npsnet

[OMG] Object Management Group Web site: http://www.omg.org

[PARADISE] PARADISE Web site: http://www.dsg.stanford.edu/ paradise.html

[ROEHL95] Roehl, B. “Some Thoughts on Behavior in VR Systems”. http://ece.uwaterloo.ca/~broehl/behav.html, University of Waterloo, 1995.

[ROEHL95-2] Roehl, B. “Distributed Virtual Reality – An Overview”. http://ece.uwaterloo.ca/~broehl/distrib.html, University of Waterloo, 1995.