UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DO NETARTOOLKIT: UM SISTEMA DISTRIBUÍDO DE REALIDADE AUMENTADA

LUCAS DE ARAÚJO OLIVEIRA

ORIENTADOR: PROF. DR. CLÁUDIO KIRNER

PIRACICABA, SP
2008

UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DO NETARTOOLKIT: UM SISTEMA DISTRIBUÍDO DE REALIDADE AUMENTADA

LUCAS DE ARAÚJO OLIVEIRA

ORIENTADOR: PROF. DR. CLÁUDIO KIRNER

Dissertação apresentada ao Mestrado em Ciência da Computação, da Faculdade de Ciências Exatas e da Natureza, da Universidade Metodista de Piracicaba – UNIMEP, como requisito para obtenção do Título de Mestre em Ciência da Computação.

PIRACICABA, SP
2008

DESENVOLVIMENTO DO NETARTOOLKIT: UM SISTEMA DISTRIBUÍDO DE REALIDADE AUMENTADA

AUTOR: LUCAS DE ARAÚJO OLIVEIRA

ORIENTADOR: CLÁUDIO KIRNER

Dissertação de Mestrado defendida em 27/02/08, pela Banca Examinadora constituída dos Professores:

_____

Prof. Dr. Cláudio Kirner (Orientador)
UNIMEP

_____

Prof. Dr. Alexandre Cardoso
UFU

_____

Profa. Dra. Regina Célia Coelho
UNIMEP

Aos

Meus pais João e Eunice

AGRADECIMENTOS

A Deus, criador da vida, pois sem Ele não estaria aqui.

Aos meus pais, pelo apoio e educação.

Ao Professor Nivaldi Calonego Junior, meu primeiro orientador e que me ensinou a trilhar por este caminho.

Ao Professor Cláudio Kirner, que deu continuidade ao trabalho.

A UNIMEP, pela oportunidade oferecida.

“No essencial, unidade; No não essencial, liberdade;
Em tudo, amor”

John Wesley
(1703 - 1791)

RESUMO

A colaboração sempre fez parte da vida das pessoas. Nesse sentido, o computador ligado em rede vem oferecendo meios de colaboração entre participantes remotos como *chats*, videoconferência, etc. No entanto, busca-se reproduzir e potencializar as condições presenciais com operações tangíveis. A realidade virtual e mais recentemente a realidade aumentada estão oferecendo essas condições. Nesse sentido, este trabalho mostra o desenvolvimento de um suporte colaborativo distribuído voltado para aplicações de realidade aumentada, envolvendo participantes remotos. O trabalho discute a infra-estrutura de rede, a alteração da biblioteca de realidade aumentada ARToolkit e apresenta alguns aspectos de avaliação de um protótipo.

PALAVRAS-CHAVE: Realidade Virtual, Realidade Aumentada, Sistema Distribuído, Ambiente Colaborativo.

ABSTRACT

Collaboration has always been part of people's lives. In this sense, the networked computer has been offering means of collaboration among remote participants such as in chats, videoconferences etc. However, it is sought to reproduce and to potentialize the presential conditions with tangible operations. Virtual reality and more recently augmented reality offer these conditions. Thus, this work shows the development of distributive collaborative support focused on augmented reality, involving remote participants. The work discusses the network infra-structure, the ARToolkit augmented reality library alteration, and presents some aspects of evaluating a prototype.

KEYWORDS: Virtual Reality, Augmented Reality, Distributed System, Collaborative Environment.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

1.1-CARACTERIZAÇÃO DO PROBLEMA

A história comprova que os primeiros experimentos matemáticos ocorreram através da música, isto é, um experimento artístico, pois "foi o próprio Pitágoras quem descobriu que as notas musicais se relacionavam com as razões do comprimento da corda do instrumento musical que as produziam ao vibrar" (MAOR, 1987 apud AZEVEDO; CONCI, 2003, p. 3).

Com a evolução dos conceitos matemáticos e paralelamente os tecnológicos, surgiu o conceito de computação gráfica, que segundo Azevedo, está relacionado com a matemática e a arte (AZEVEDO;CONCI, 2003, p.3). Segundo a *ISO – (International Organization for Standardization),* a definição de computação gráfica é: "um conjunto de ferramentas e técnicas para converter dados para ou de um dispositivo gráfico através do computador".

Estes objetos gráficos apresentados por meio do computador no início eram apenas realizados por meio de comandos de texto enviados pelo teclado, pois no início a interface utilizada era baseada em botões e alavancas. Em seguida, com o avanço tecnológico e o advento de um novo processo mais sofisticado, inicia-se o conceito de uma nova interface que é baseada no uso de mouse e menus (janelas), presentes nos dias atuais. Todas estas tecnologias de interfaces preconizavam que os usuários deveriam se adaptar e ajustar à nova tecnologia. Com o avanço do software, hardware e das telecomunicações, surgiram interfaces com voz, tangíveis, hápticas, etc, possibilitando aos usuários uma sensação de acesso como se estivessem no mundo real, isto é, falando, realizando gestos, etc. Toda esta evolução, não só no software, mas também nos computadores, circuitos, poder de processamento, computação gráfica, redes, Internet, contribuíram para o surgimento de poderosas interfaces 3D, que inclui a Realidade Virtual (RV) e a Realidade Aumentada (RA). RV e RA surgem como uma nova geração com o uso de recursos tridimensionais que se aproximam do mundo real e rompem

barreiras do monitor, possibilitando uma interação em tempo real com o usuário (KIRNER; SISCOUTTO, 2007a).

Interfaces baseada em realidade virtual têm como característica a representação do real e/ou o imaginário, possuindo como principais características a navegação, interação e a imersão no mundo virtual. É um ambiente sintético, gerado por computador, em que a interação homem-máquina é mediada por dispositivos multisensoriais e apresentam respostas que devem ser em tempo real (KIRNER, 2004a).

A modelagem desses ambientes virtuais pode ser elaborada por meio de uma linguagem chamada VRML (*Virtual Reality Modeling Language*) (KIRNER, 2004a). Esta linguagem permite que o usuário visualize ambientes, manipule objetos e outros componentes do cenário virtual, além de ter a característica da movimentação no mundo virtual e ser uma linguagem independente de plataforma.

Diferentemente da realidade virtual, existe um tipo de aplicação, na qual o espaço real é complementado com objetos virtuais. Este tipo de aplicação é chamada de realidade aumentada, onde são inseridos elementos virtuais com o objetivo de obter uma complementação das informações reais no ambiente virtual (AZUMA, 1997).

Sistemas de realidade virtual ou de realidade aumentada podem suportar uma pessoa, usando apenas um computador, ou até muitos usuários, usando um sistema distribuído. Sistemas distribuídos de realidade virtual permitem que usuários geograficamente distantes possam atuar em mundos virtuais que estão compartilhados por uma rede de computadores. O objetivo desses sistemas distribuídos é resolver problemas de maneira colaborativa.

Partindo deste princípio, isto é, da colaboração, da cooperação mútua, é que se propõe o desenvolvimento de um sistema distribuído com o uso da realidade aumentada, com o objetivo de exercitar as formas de comunicações e colher resultados pelo uso desta nova interface de comunicação.

1.2-MOTIVAÇÃO

A motivação para o desenvolvimento deste projeto se baseou na demanda de sistemas na área de realidade aumentada vinculada ao uso de sistemas distribuídos.

A aplicação deste sistema servirá para interação, colaboração, e estimulação de usuários, em que os sentidos serão exercitados.

1.3-OBJETIVOS

Este trabalho tem como objetivo geral o desenvolvimento de um sistema distribuído de realidade aumentada, utilizando a ferramenta NETARToolkit, biblioteca escrita na linguagem C++ de código livre que é responsável por fazer o rastreamento de marcadores pré-definidos pela ferramenta, e associar um elemento virtual ao marcador. Os usuários, remotamente, poderão trabalhar de forma colaborativa para atender objetivos em comum. A realidade aumentada será a interface usada entre o usuário e o computador, sendo uma interface intuitiva, isto é, os movimentos corporais são os próprios “dispositivos” de interação.

Como objetivos específicos, esta pesquisa visa testar as formas de comunicação entre os computadores na rede com RA, analisando-as e colhendo os resultados. Desenvolver-se-á exemplos de colaboração com Realidade Aumentada usando a rede. Dentro deste mesmo objetivo, será verificado o comportamento destes ambientes colaborativos de realidade aumentada, bem como a coleta de dados de análises referente às reações dos usuários.

1.4-ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido da seguinte forma:

No capítulo 1, há a Introdução ao tema em questão. Há uma apresentação do tema para situar o leitor, bem como mostrar a importância deste trabalho no contexto científico.

O capítulo 2 possui a fundamentação teórica, dos conceitos, definições, dispositivos e aplicações de realidade virtual e realidade aumentada, e as ferramentas utilizadas no projeto, VRML e ARToolkit.

O capítulo 3 apresenta os mecanismos de comunicação em rede de computadores.

O capítulo 4 apresenta as definições de sistemas distribuídos de realidade virtual e aumentada.

No capítulo 5, há a descrição do sistema, metodologia e materiais utilizados.

No capítulo 6 há a descrição dos testes realizados bem como a apresentação dos resultados e avaliações.

E por fim, há a conclusão com algumas contribuições e trabalhos futuros no capítulo 7.

2. REALIDADE VIRTUAL E AUMENTADA

2.1 CONCEITOS DE RV

O primeiro contato com realidade virtual registrado, foi em 1966, por Ivan Sutherland, considerado nos EUA como o pioneiro da Realidade Virtual, juntamente com seus colegas do Laboratório Lincoln, realizando experiências com diversos tipos de capacetes de visão. Suas pesquisas continuaram na Universidade de Utah e o primeiro dispositivo de imersão, bastante rudimentar, pôde funcionar em 1970. O usuário podia ver flutuar no ar um cubo de 10cm de aresta, representado por arestas de luz. Não obstante sua figuração simplista, esse cubo, que ficava numa posição estável independentemente dos movimentos da cabeça do usuário, parecia real (CADOZ, 1997).

As primeiras demonstrações públicas de experiências dessa natureza na Europa aconteceram por volta do final da década de 1980, com dispositivos de desempenho médio.

O conceito de realidade virtual veio se consolidando a partir de meados da década de 1990. Segundo Cadoz (1997), a realidade virtual estava caracterizada como um tipo de imersão dentro de uma imagem, isto é, pode-se ver, ouvir, tocar ou manipular objetos que não existem, percorrer espaços sem lugar, tendo a absoluta certeza da realidade e da presença destes objetos virtuais.

Hoje, o usuário munido de um capacete de visão e de uma luva ligados ao computador, está imerso num ambiente virtual em que os objetos estão diante de si, mas também atrás, acima, abaixo, basta dar meia-volta, levantar ou abaixar a cabeça para encontrá-los no mesmo lugar, permanentes.

2.1.1 DEFINIÇÕES

Segundo Burdea *and* Coiffet (2003), RV, em termos funcionais, é uma simulação nas quais gráficos de computadores são usados para criar uma aparência do mundo real. Além disso, o mundo sintético não é estático, mas responde às entradas do usuário (gestos, comandos verbais, etc). Esta

definição aponta as principais características desta interface, isto é, a resposta em tempo real e a interatividade, envolvendo o usuário por meio de múltiplos canais sensoriais. Tempo real, neste contexto, significa que o computador é habilitado para detectar as entradas do usuário e modificar instantaneamente o mundo virtual. Interatividade é um poder cativante que contribui para o sentimento de imersão, fazendo parte da ação sob a tela. Mas os impulsos da RV tentam cada vez mais e mais nivelar os canais sensoriais humanos. Certamente, usuários não só verão e manipularão objetos gráficos na tela, mas também poderão tocá-los e sentí-los. Pesquisadores estão estudando os sentidos de cheiro e tato, entretanto, nos dias atuais, estas modalidades sensoriais são pouco usadas (CADOZ, 1997).

Alguns pontos são relevantes, quando há a conceituação de RV, tais como:

- Interface do usuário;
- Navegação;
- Interação;
- Imersão;
- Ambiente sintético gerado por computador;
- Dispositivos multisensoriais;
- Tempo real.

Este tipo de interface, porém, requer treinamento e destreza, pois possui uma certa dificuldade em relação ao mundo real.

Pode-se dividir, durante a história, 4 gerações de interfaces:

1ª geração: interface por comandos de teclado;

2ª geração: interface por menus-mouse;

3ª geração: utilização de RV – interface 3D;

4ª geração: mundo real povoado com objetos virtuais.

Estamos caminhando para esta 4ª geração, na qual objetos virtuais farão parte da nossa vida cotidiana, seja ela em nosso trabalho, bancos, lazer, supermercados, etc, onde a forma de interação se dará de forma natural, usando nossos próprios movimentos corporais para manipulá-los.

2.1.2 APLICAÇÕES

A RV pode ser aplicada em diversas áreas. Aplicações que envolvem simulação, treinamento ou animações podem ser apoiadas pela RV. Dentre as diversas áreas do conhecimento, destacamos alguns exemplos de uso da RV.

2.1.2.1 Área Médica

Com o objetivo de ensinar a anatomia interna do sistema auditivo humano de uma maneira mais criativa e dinâmica, uma universidade americana[1], desenvolveu um sistema de RV de um ouvido no qual os alunos podem fazer um passeio virtual altamente realístico em três dimensões dentro dele, como ilustra a figura 1.

Fig. 1 – Ambiente Virtual de ouvido (SABBATINI, 1999)

Além desta aplicabilidade, existem outras, como por exemplo a possibilidade de um médico treinar uma cirurgia. Na figura 2, o médico está treinando uma cirurgia de vasos sanguíneos, usando equipamentos de RV que simulam o uso de tesouras, pinças e outros instrumentos médicos. Com o avanço das redes e sistemas distribuídos, esta cirurgia eventualmente poderá ocorrer a distância.

[1] http://www.informaticamedica.org.br/informaticamedica/n0202/sabbatini.htm

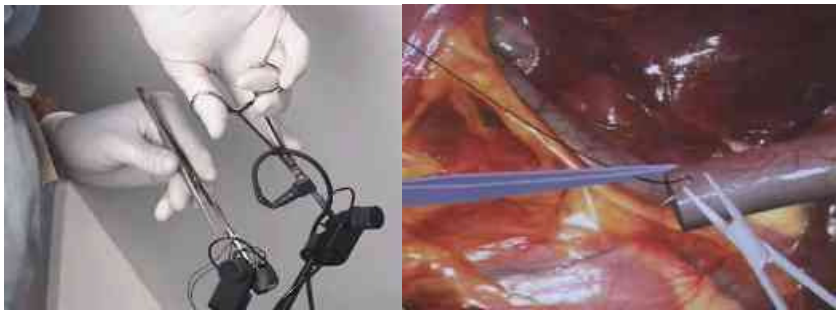

Fig. 2 – Simulação de uma cirurgia (SABBATINI, 1999)

2.1.2.2 Área Educacional

Pelo fato da RV ter a vantagem de possuir uma certa facilidade no que tange a área de simulação ou animação, o setor educacional é privilegiado também por esta tecnologia. Pode-se, com o uso da RV, realizar simulações de laboratórios, experiências, prover uma melhor condição de absorção, pois o aluno, apesar de não estar manipulando algo real, poderá realizar diversas vezes seu trabalho, lembrando que estes objetos virtuais possuem interação em 3D podendo ainda serem sentidos e manipulados.

A educação tem se beneficiado com o uso da RV por vários motivos, mas um se destaca, que é a experiência pessoal que o aluno tem quando ele mesmo faz a imersão, navegação e a exploração no mundo virtual, isto é, o aluno vive suas experiências e explora a informação como uma experiência diária (CARDOSO; LAMOUNIER, 2004b).

Bell, Foglerl (1995), Pinho (2000), Meiguins (1999) apud Cardoso e Lamounier (2004) apontam vantagens do uso da RV na educação:

- motivação dos estudantes baseada na experiência pessoal de exploração no mundo virtual;
- forte poder de ilustração de características e processos;
- possibilidade de visualização de detalhes do objeto;
- permite experiências virtuais;
- encoraja a criatividade, catalisando a experimentação.

2.2 REALIDADE AUMENTADA

A realidade aumentada está dentro de uma sub-área chamada realidade misturada. Esta área concentra-se entre o mundo real e o mundo

virtual. Além da realidade aumentada estar dentro desta realidade misturada, existe um outro conceito que é a virtualidade aumentada. A figura 3 ilustra estes conceitos.

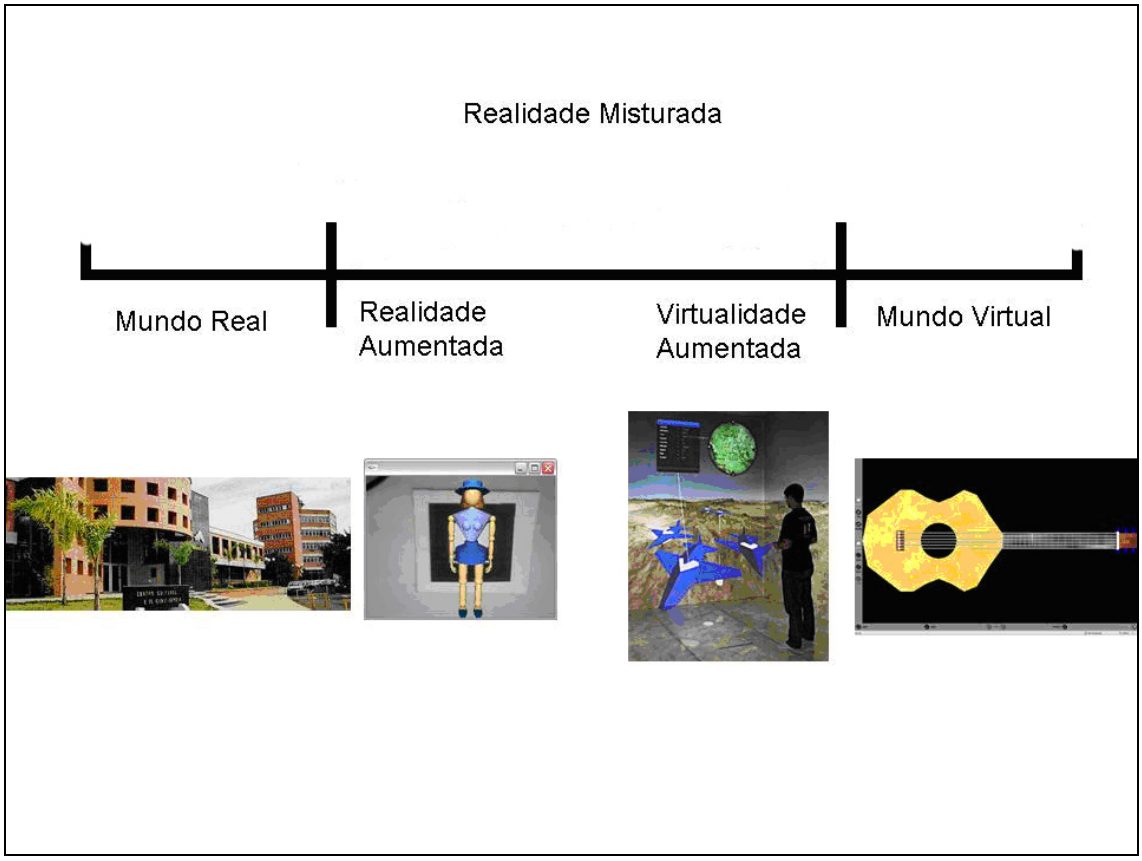

Fig. 3 – Realidade Misturada (adaptada de Milgram, 1994)

O que caracteriza realidade aumentada ou virtualidade aumentada é a forma de interação utilizada, isto é, se a interface utilizada se dá com meios reais ou virtuais. Segundo Kirner (2007a), a realidade aumentada "é a inserção de objetos virtuais no ambiente físico, mostrada ao usuário, em tempo real, com o apoio de algum dispositivo tecnológico, usando a interface do ambiente real, adaptada para visualizar e manipular os objetos reais e virtuais." [traduzido de KIRNER, C.; KIRNER, T.G. (2007a)]. A virtualidade aumentada é "a inserção de representações de elementos reais no mundo virtual, usando a interface que permite ao usuário interagir com o ambiente virtual". [traduzido de KIRNER, C.; KIRNER, T.G. (2007a)].

A realidade aumentada, portanto, é caracterizada pela inserção de elementos virtuais no espaço real. Essa modalidade de interface homem-máquina potencializa aplicações em que a complementação das informações

do real se faz necessária, isto é, existe a característica do enriquecimento do espaço físico real com informações, complementações ou objetos virtuais.

Esta modalidade de interface, diferentemente da RV, traz o ambiente virtual até o espaço do usuário, facilitando a navegação do mesmo. Devido às dificuldades encontradas com o manuseio de dispositivos (mouse, teclado, controles) na RV, surgiu-se o conceito de que os próprios movimentos do corpo serviriam como dispositivos de interação (mão, cabeça, perna), fazendo com que o manuseio neste mundo virtual fosse de uma maneira natural, dispensando treinamentos por se tratar de movimentos presentes no ser humano. Por estas e outras características a RA vem tomando o espaço da virtualidade aumentada.

A figura 4 ilustra um exemplo que apresenta o espaço físico potencializado e complementado com informações ou objetos virtuais.

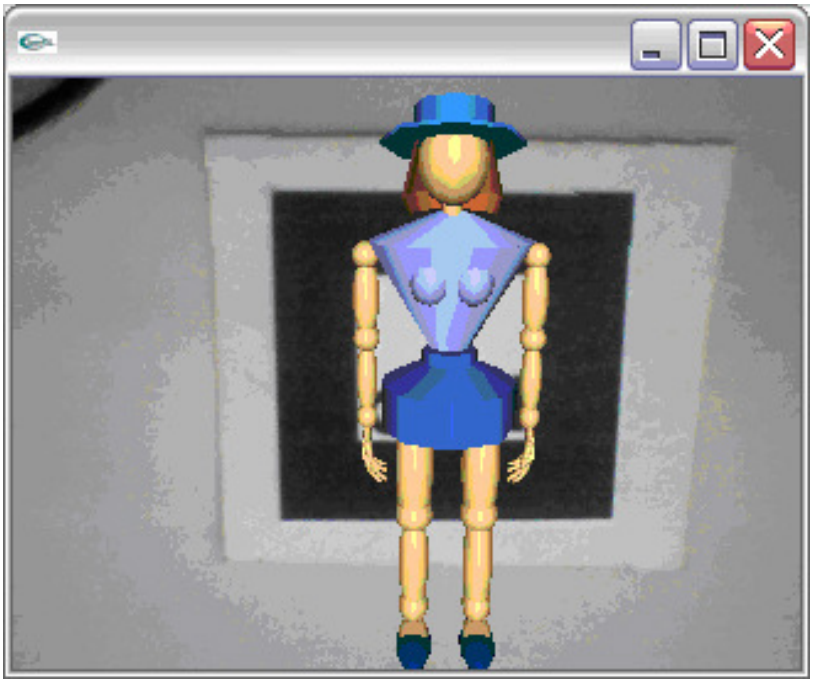

Fig. 4 – Objeto virtual carregado na cena real

2.2.1 COMPARAÇÕES

Esta seção objetiva comparar as interfaces virtualidade aumentada com realidade aumentada. Levando-se em conta a definição de ambas, pode-se concluir que uma mesma cena pode ser classificada como virtualidade aumentada ou realidade aumentada dependendo da interface de interação utilizada. Se tomarmos como exemplo a figura 4, isto é, um ambiente

real é complementado com informações virtuais com o uso de marcadores, tem-se a realidade aumentada. Porém, se este mesmo objeto fosse manipulado por uma luva (*dataglove*) que representa um interface virtual na cena, classificaríamos como virtualidade aumentada, pois minha cena virtual é potencializada com objetos reais. A tabela 1 ilustra esta comparação.

Tabela 1 – Comparação entre realidade aumentada e virtualidade aumentada

Interface	Objetos	Interação
Realidade Aumentada	Reais e virtuais	Ações Tangíveis
Realidade Virtual	Virtuais	Uso de dispositivos multisensoriais
Virtualidade Aumentada	Reais e virtuais	Uso de dispositivos multisensoriais

Seguindo esta linha de raciocínio, pode-se comparar as interfaces realidade virtual com virtualidade aumentada.

A RV, conforme definição anterior, é uma poderosa interface, onde o usuário pode navegar e interagir em tempo real com a cena virtual gerado pelo computador. Se um usuário possui um jogo de vídeo game, por exemplo, e a interação se dá apenas entre o usuário e a console, tem-se a RV. Se em algum momento, este usuário modificar o estilo do jogo para mostrar representações do mundo real, então esta cena será classificada como virtualidade aumentada, pois o mundo virtual será aumentado com a inserção de representações reais.

Para finalizar esta seção, será apresentado uma comparação entre a interface de realidade virtual e realidade aumentada (KIRNER, 2008).

- A RV trabalha somente com o mundo virtual; o usuário é transferido para dentro deste ambiente virtual; a característica de interação entre usuários é priorizada;
- A RA possui um mecanismo de combinar o mundo real com o mundo virtual; mantém a sensação de presença do usuário no mundo real; enfatiza a qualidade das imagens e a interação entre os usuários.

2.2.2 DEFINIÇÕES

Segundo Kirner (2007a), RA pode ser definida da seguinte maneira: Enriquecimento do mundo real com objetos virtuais, interagindo por meio de algum dispositivo tecnológico funcionando em tempo real.

Para a concretização destas características, surge um novo conceito, que é o do rastreamento. A importância do rastreamento se dá nas aplicações 3D para a interface do usuário dar informações sobre a localização do usuário ou do objeto no espaço como, por exemplo, o cálculo das informações sobre o movimento dos dedos da mão para que a mão virtual possa ser renderizada como a mão real.

Um ponto relevante a ser destacado é que na interação 3D em ambientes virtuais, o sistema pode oferecer a correspondência entre os ambientes físicos e virtuais. Portanto, ter um rastreamento correto é parte crucial para tornar as técnicas de interação úteis em aplicações de ambientes virtuais.

Existem três tipos de dispositivos de rastreamentos mais comuns:

- Rastreadores de movimentos;
- Rastreadores de olhos;
- Luvas de dados (*data gloves*).

O foco deste trabalho se concentrará em uma das tecnologias de rastreadores de movimentos, que é o rastreamento óptico.

O rastreamento óptico é fundamental no contexto das aplicações em realidade aumentada, dado que há a necessidade de captura das imagens reais às quais são inseridos os elementos virtuais.

Foi escolhido o software ARToolkit (HitLabNZ, 2007) para realização da captura da cena real e inserção dos elementos virtuais, por se tratar de um software livre e de código aberto, permitindo a alteração de funcionalidades. A seção 2.4.2 detalha as funcionalidades e características desta ferramenta.

2.2.3 TIPOS DE SISTEMAS DE REALIDADE AUMENTADA

Existem 4 tipos de sistemas de RA, sendo que esta classificação se dá pelo display utilizado (AZUMA, 1997 apud SILVA, 2006). São eles:

- Sistema de visão ótica direta;
- Sistema de visão direta por vídeo;
- Sistema de visão por vídeo baseado em monitor;
- Sistema de visão ótica por projeção.

O sistema de visão ótica direta utiliza óculos ou um capacete com a característica de receber a imagem do mundo real complementada com a cena virtual devidamente ajustada na cena. A figura 5 ilustra o funcionamento deste sistema.

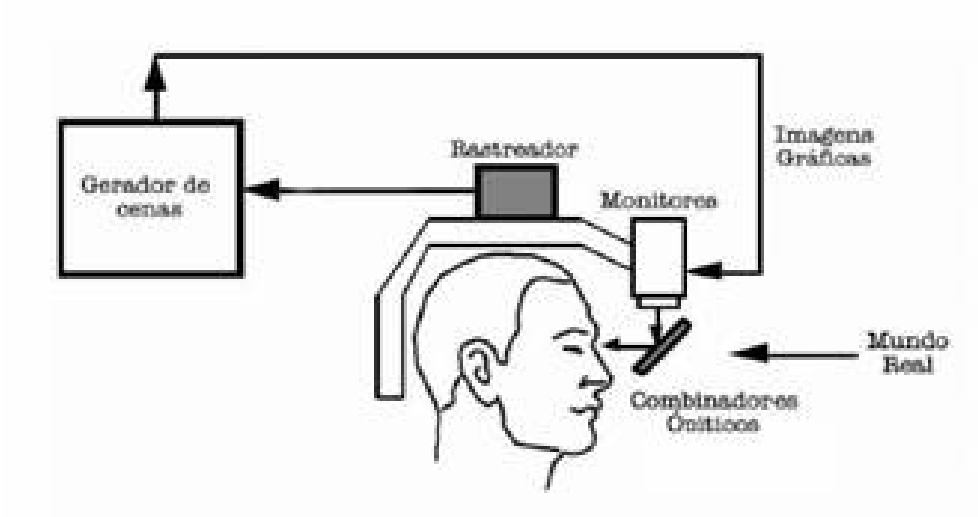

Fig. 5 - Sistema de visão ótica direta (Realidade Aumentada, 2007 apud AZUMA, 1997)

O sistema de visão direta por vídeo utiliza capacetes com câmeras de vídeos capturando o mundo real, que com a ajuda do computador é misturada com a cena virtual e apresentada ao usuário. A figura 6 apresenta este sistema.

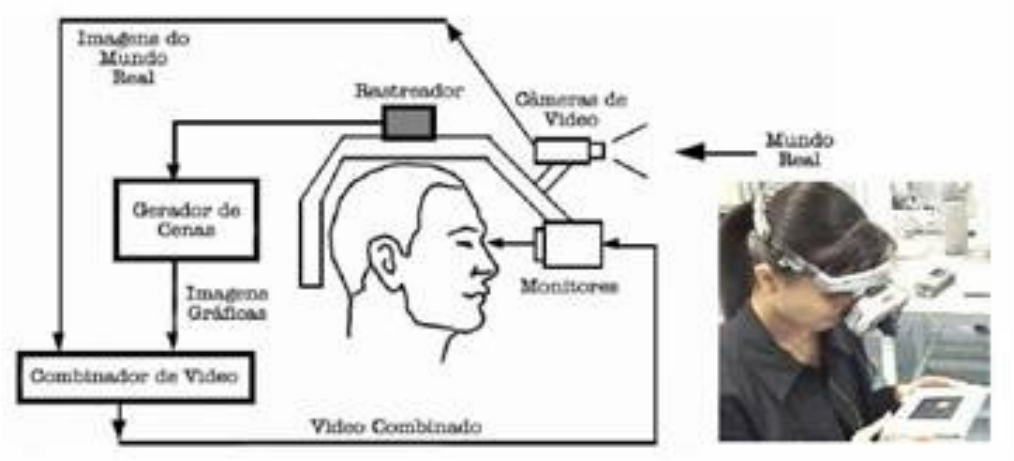

Fig. 6 - Sistema de visão direta por vídeo (Realidade Aumentada, 2007 apud AZUMA, 1997)

A principal diferença entre o sistema de visão ótica direta e o sistema de visão direta por vídeo é que, no primeiro sistema o usuário recebe informações diretas do mundo real e no outro sistema a imagem do mundo real é recebida por um monitor.

O sistema de visão por vídeo baseado em monitor utiliza uma *webcam* para capturar a cena real e após esta captura, o computador mescla os objetos virtuais e apresenta no monitor. A figura 7 mostra este sistema.

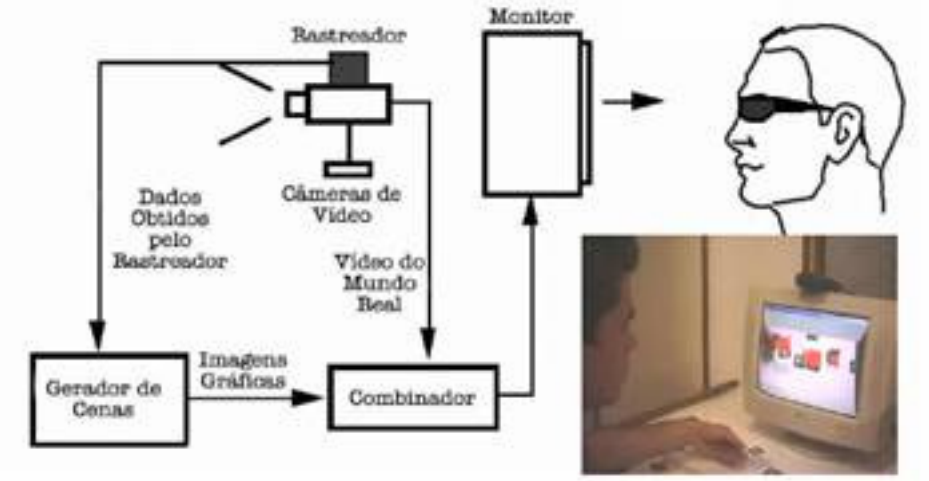

Fig. 7 - Sistema de visão por vídeo baseado em monitor (Realidade Aumentada, 2007 apud AZUMA, 1997)

O último sistema, o de visão ótica por projeção, projeta o mundo virtual em superfícies reais, não necessitando de nenhum equipamento para interagir neste cenário.

2.2.4 APLICAÇÕES

A aplicabilidade da RA estende-se à várias áreas. A título de ilustração, apresenta-se a utilização na RA na área médica e educacional.

2.2.3.1 Área Médica

A área médica pode ser favorecida com o uso da RA. Devido ao elevado número de informações e detalhes, a RA possibilita com o uso de algum sistema (apresentado acima), a complementação de informações virtuais para o médico que está analisando algum órgão ou algum paciente.

2.2.3.2 Área Educacional

Pelo fato da RA possuir características de modelagem muito poderosa de objetos, torna-se eficiente o uso da RA na educação, tendo como principal virtude a memorização, manipulação e a imersão dos estudantes.

Um exemplo é o projeto desenvolvido na UNIMEP, constituindo o livro "Sólidos Geométricos com realidade aumentada e ARToolKit" (KIRNER, 2007b), na qual os alunos podem visualizar os objetos geométricos de maneira tridimensional, como mostra a figura 8.

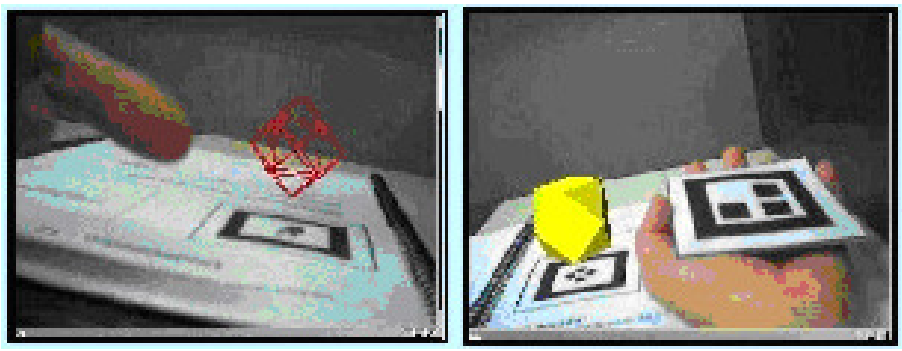

Fig. 8 - Livro interativo com realidade aumentada(KIRNER, 2007b)

2.3 FERRAMENTAS DE PROJETO

Esta seção tem como objetivo apresentar as ferramentas principais utilizadas no projeto. Neste caso, serão abordados a linguagem de modelagem VRML e a biblioteca ARToolkit.

2.3.1 VRML

Existe uma linguagem de modelagem 3D bastante usada para modelar objetos e disponibilizá-los no mundo virtual. Esta linguagem é VRML, *Virtual Reality Modeling Language* (HARTMAN, 1996 apud CALONEGO et. al, 2004). Enquanto que outras linguagens comuns na Internet, como por exemplo o HTML, disponibilizam textos, links, figuras entres outros, o VRML apresenta ao usuário mundos virtuais que podem ser representações do mundo real ou apenas cenas imaginárias.

Esta linguagem de modelagem de objetos e ambientes virtuais trabalha com objetos 3D no espaço baseado nas coordenadas X, Y e Z, sendo X o eixo horizontal, Y é o vertical e Z é o eixo de profundidade(TORI et al, 2006). Ela foi desenvolvida por Mark Pesce e Tony Parisi[2] em 1994, e construíram um protótipo de navegador 3D para a WWW.

Com o uso desta linguagem, é possível criar objetos tridimensionais podendo definir cor, transparência, brilho, textura. Os objetos podem ser de formas diversas, dependendo da expectativa do programador. É possível acrescentar interatividade a estes por meio de sensores, podendo assim deslocá-los de posição, acrescentar luz ou som.

Para criar um arquivo em VRML, precisa-se apenas de um editor de texto, e o arquivo deverá ser salvo com a extensão .wrl. Ao ler este arquivo, o navegador de web, por meio de um *plugin*, consegue interpretar os comandos e mostrar o objeto em seu navegador.

Alguns destes *plugins* são:

- Cortona - Parallel Graphics
- Blaxxun - Contact
- Octaga (X3D)
- WorldView
- VCom3D
- FreeWRL

[2] http://www.realidadevirtual.com.br/cmsimple-rv/?%26nbsp%3B_VRML:Conceitos_e_Instala%E7%E3o

A figura 9 ilustra um objeto carregado no cenário virtual com o uso do *plugin* Cortona[3]

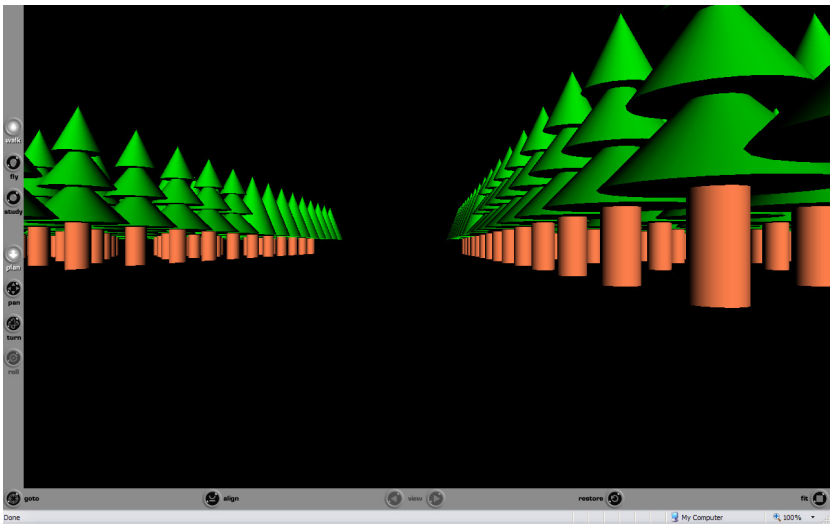

Fig. 9 – Mundo virtual no plugin cortona

A estrutura básica desta linguagem de modelagem possui 3 conceitos essências, que são:

- Internet;
- HTML e Java;
- Geometria 3D.

A Internet disponibiliza o mundo virtual ao usuário, podendo atuar junto com outros mundos, compartilhando informações e colaborando entre si, por meio de um sistema distribuído.

As linguagens HTML e Java atuam como um apoio e um suporte para o VRML. Tem como principais funcionalidades a criação de *scripts* e outras peculiaridades que o VRML não suporta.

E por fim, o VRML tem como essência a criação e a disponibilização de mundos virtuais 3D. É uma poderosa interface homem-máquina em que objetos mapeados ou apenas objetos criados por meio da imaginação de seu criador, são colocados neste mundo virtual, e, por meio de dispositivos de interação (mouse, luvas, etc), o usuário pode manipulá-los e analisá-los de diferentes ângulos, formas e jeitos.

Há 3 versões da linguagem VRML, que são:

- VRML 1.0 – primeira versão;

[3] http://www.parallelgraphics.com/products/cortona/

- VRML 2.0 - versão que foi modificada estruturalmente e permite animações;
- VRML97 (ISO/IEC) - VRML 2.0 padronizada.

A figura 10 ilustra o procedimento para a visualização de um arquivo VRML:

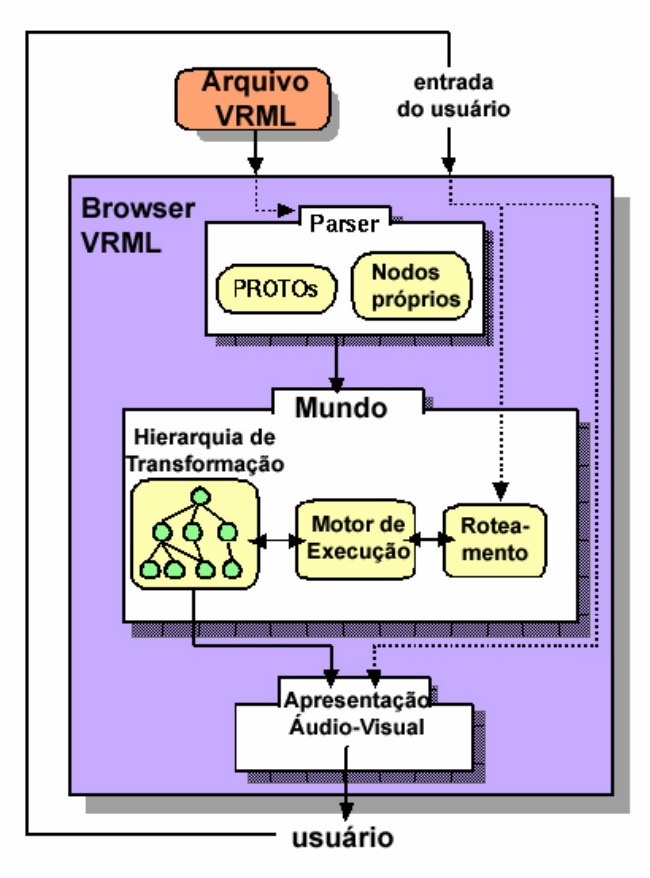

Fig. 10 – Arquitetura para visualização de um arquivo VRML (CONSULARO, 2006b)

2.3.2 ARTOOLKIT

O rastreamento óptico é fundamental no contexto das aplicações em realidade aumentada, dado que há a necessidade de captura das imagens reais às quais são inseridos os elementos virtuais. O ARToolkit (KATO, 1999) é uma biblioteca de código aberto, escrita em C++, que efetua o rastreamento de marcadores (descrito na figura 11) inseridos no mundo real. Atualmente, a biblioteca tem como principais funcionalidades: (i) a identificação dos marcadores; (ii) definição um sistema referencial cartesiano por marcador, que é dado em relação à posição de câmera; (iii) identificação de pose, que define a localização geográfica da marca em relação ao marcador. Portanto, o

sistema de rastreamento óptico implementado nessa biblioteca está associado a uma câmera conectada a um computador.

Fig. 11 – Marcador do ARToolkit

A figura 12 ilustra o funcionamento desta ferramenta.

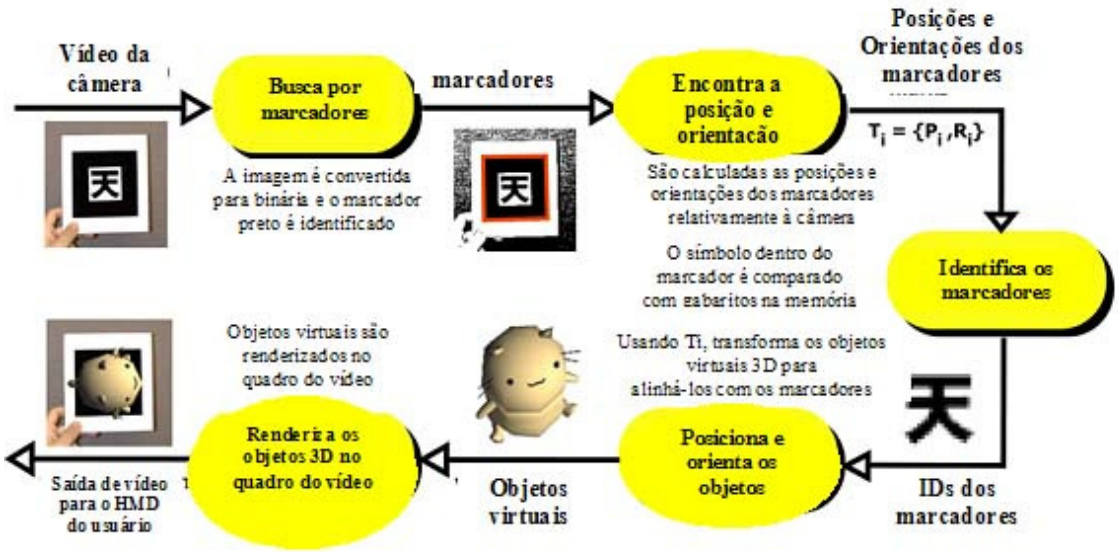

Fig. 12 – Funcionamento do ARToolkit (SINCLAIR, 2004)

Segundo Hartley e Zisserman (2003 apud Consularo et. al, 2006), uma das características mais trabalhosas na implementação de uma aplicação usando realidade aumentada é calcular precisamente o ponto de vista do usuário para que este veja as imagens virtuais alinhadas com as imagens do mundo real. Isto tudo deve ser feito em tempo real.

A biblioteca ARToolkit visa suprir esta dificuldade, pois esta biblioteca, com o uso de técnicas de visão computacional, calcula a posição real da câmera e sua orientação relativa aos marcadores, fazendo com que objetos virtuais sejam sobrepostos nestes marcadores. Segundo Consularo et.

al. (2006), este preciso e rápido rastreamento oferecido pelo ARToolkit possibilita um desenvolvimento rápido para diversas aplicações em RA.

Dentro desta biblioteca, existe uma aplicação chamada simpleVRML. Esta aplicação ilustra o desenvolvimento de software de realidade aumentada. Para ilustrar o problema pode-se considerar que essa aplicação implementa, resumidamente, o algoritmo representado na Figura 13.

início

3 Inicia as variáveis

3 Inicia a captura de vídeo

3 Define a função de gerenciamento de eventos

3 Define a função principal da renderização

Fim

Fig. 13 – Algoritmo da aplicação simpleVRML

As funções citadas nas linhas 3 e 4 do algoritmo são funções do tipo “call-back” e são chamadas por funções da biblioteca OpenGL[4], ao efetuar o percurso no grafo de cena. A função da linha 3 permite a entrada de dados via teclado.

A função descrita na linha 4 implementa os algoritmos de inserção de objetos virtuais no frame capturado. A inserção dos objetos depende dos elementos pré-definidos para cada um dos marcadores. Assim, durante o processo de inicialização, todas as cenas VRML cadastradas são transformadas em grafos de cena, usando as funções da biblioteca OpenVRML. Quando uma marca não pode ser identificada, ela é assinalada como invisível, não sendo renderizado esse grafo de cena. Portanto, a aplicação simpleVRML usa um vetor, em que cada elemento é um grafo de cena, mas nem todos esses grafos são renderizados. Ou melhor, somente um destes grafos é associado a um marcador pré-definido e renderizado. A figura 14 ilustra a execução desta aplicação.

[4] http://www.opengl.org/

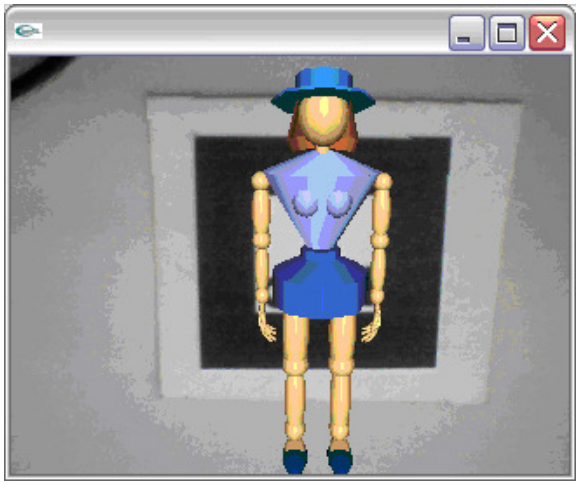

Fig. 14 – Execução do simpleVRML

3 MECANISMOS DE COMUNICAÇÃO

Sistemas de realidade virtual ou de realidade aumentada, além de outros sistemas computacionais, podem suportar uma pessoa, usando apenas um computador, ou até muitos usuários, usando um sistema distribuído. Sistemas distribuídos de realidade virtual permitem que usuários geograficamente distantes possam atuar em mundos virtuais que estão compartilhados através de rede de computadores. O objetivo desses sistemas distribuídos de realidade virtual é resolver problemas de maneira colaborativa.

Os sistemas distribuídos começaram a aparecer em meados da década de 60 com o objetivo de se implementar sistemas centralizados. Paralelo a este surgimento, as redes locais começaram apresentar um aumento de confiabilidade e desempenho. Segundo Kirner e Mendes (1988), nesta época já se tinha um domínio bom das técnicas de compartilhamento de recursos de comunicação, como canais e processadores de comunicação, enquanto que as técnicas de compartilhamento de recursos de computação, como hardware, programas, banco de dados, encontravam-se em fase inicial de desenvolvimento.

Muito se discutiu, através de encontros e workshops a definição de sistemas distribuídos no final da década de 70, e ficou definido que há vários elementos de um sistema que podem ser distribuídos, tais como processadores, dados, programas e o sistema operacional (PETERSON et all, 1979 apud KIRNER; MENDES, 1988).

Segundo Eckhouse (1978 apud KIRNER; MENDES, 1988) um sistema distribuído é “formado por um conjunto de módulos, compostos pelo menos por processador-memória, interligados frouxamente através de um subsistema de comunicação de topologia arbitrária.”

Segundo Coulouris et al. (2001, p.1), um sistema distribuído é um “sistema no qual computadores em rede se comunicam e coordenam suas ações, apenas pela passagem de mensagens”. Esta comunicação pode se dar independente de qualquer distância, isto é, podem estar em continentes separados, no mesmo prédio ou até mesmo na mesma sala. Através desta definição, tem-se às seguintes características de sistemas distribuídos:

- concorrência dos componentes: na qual a execução de programas concorrentes é a norma. Eu posso fazer meu trabalho em meu computador, enquanto você faz o seu em seu computador,
- falta de um clock global: cada processador tem seu tempo de execução;
- falhas independentes dos componentes: pois todos os computadores podem falhar, sendo a responsabilidade dos projetistas planejarem as conseqüências das possíveis falhas.

Podemos citar três exemplos de sistemas distribuídos:

- Internet;
- Intranet;
- Dispositivos móveis.

Para a construção de um sistema distribuído, deve-se levar em conta o compartilhamento dos recursos, cuja característica é a principal motivação para a construção do sistema (TANENBAUM, 1995).

Segundo Coulouris et al. (2001), existem alguns desafios para a construção desses sistemas. Entre eles estão a heterogeneidade dos componentes, flexibilidade, o qual permite componentes serem adicionados ou recolocados, segurança, escalabilidade – habilidade para se trabalhar bem quando o número de usuários aumenta - falhas humanas, concorrência, transparência e código móvel.

Redes de computadores estão em todos os lugares. A Internet é uma, no qual muitas redes fazem parte. Redes de telefones móveis, redes corporativas, redes de manufatura, redes em campus, redes em casa, redes em carro, todos eles, compartilham as características principais que fazem assuntos importantes de estudo no que diz respeito a sistemas distribuídos.

3.1 DESAFIOS

Um dos desafios para a implantação de um sistema distribuído é a heterogeneidade. A Internet é um exemplo de uma rede heterogênea, pois habilita ao usuário o acesso a serviços e a execução de aplicações. Heterogeneidade (isto é, variedade e diferenças) aplica-se às seguintes características (COULOURIS et al., 2001):

- Redes;
- Hardware de computador;
- Sistemas operacionais;
- Linguagem de programação;
- Implementações feitas por diferentes desenvolvedores.

Embora a Internet consista em vários tipos de redes, estas diferenças são mascaradas pelo fato de que todos os computadores unidos usam um protocolo de internet para se comunicarem uns com os outros.

Embora os sistemas operacionais de todos os computadores na Internet necessitem incluir uma implementação dos protocolos de Internet, não necessariamente precisam prover a mesma interface de programação da aplicação para estes protocolos.

Diferentes linguagens de programação usam representações diferentes para caracteres e estruturas de dados tais como vetores. Essas diferenças devem ser dirigidas, se os programas escritos em linguagens diferentes estiverem habilitados a se comunicarem uns com os outros.

Programas escritos por diferentes desenvolvedores não podem se comunicar uns com os outros, a não ser que eles usem padrões comuns, como por exemplo, para a comunicação de rede e a representação dos dados primitivos e as estruturas de dados nas mensagens. Para isto acontecer, padrões necessitam estar de acordos e adotados, assim como os protocolos de Internet.

Dentre os protocolos mais comuns, existem o UDP e o TCP. O protocolo UDP (*User Datagram Protocol*) é caracterizado pela transmissão de um processo emissor para um processo receptor, sem reconhecimentos e novas tentativas. Se uma falha ocorrer, a mensagem pode não chegar. Para enviar ou receber mensagens um processo primeiramente deve criar um *socket* para os endereços de internet de um servidor local e uma porta local.

Pode-se concluir que este protocolo é pouco confiável, pois existem técnicas para confirmar que os dados chegaram de maneira correta ao receptor, e também, é um protocolo não orientado para conexão.

O protocolo TCP (*Transmission Control Protocol*) foi originado do BSD 4.x UNIX e tem a característica de prover a abstração no fluxo de bytes

de dados, que podem ser escritos e de que dados podem ser lidos (COULOURIS et al., 2001). As seguintes características da rede são ocultadas pela abstração deste fluxo:

- Tamanho das mensagens;
- Mensagens perdidas;
- Controle de fluxo;
- Duplicação de mensagens e ordenação;
- Destino das mensagens.

A comunicação deste fluxo supõe que, quando pares de processos estabelecem uma conexão, um deles faz o papel de cliente, e o outro faz o papel de servidor. O cliente cria um *socket* em uma porta qualquer e então faz uma requisição de conexão com o servidor. Quando o servidor aceita a conexão, um novo *socket* de fluxo é criado para a comunicação do cliente com o servidor.

Quando uma aplicação encerra um *socket*, significa que não será mais gravado qualquer dado no fluxo de saída. Quando um processo falha, todos os *sockets* são fechados e qualquer tentativa de comunicação será descoberta.

O TCP usa um modelo para satisfazer a propriedade de integridade da comunicação segura, conhecido como Modelo de Falhas. Este modelo usa um *checksun* para detectar e rejeitar pacotes corruptos e números seqüenciais para detectar e rejeitar pacotes duplicados. Para a propriedade de validade, TCP usa *timeouts* e retransmissões de pacotes perdidos.

Quando uma conexão é quebrada, o processo será notificado se houver a tentativa de gravação ou leitura. Isto tem os seguintes efeitos:

- O processo em conexão não pode distinguir entre a falha da rede e a falha do processo;
- Os processos em comunicação não podem dizer se suas mensagens foram recebidas ou não.

Muitos dos serviços usam o protocolo TCP, alguns exemplos atuais são:

- HTTP;
- FTP;

- Telnet;
- SMTP.

Quando há padronização dos processos e troca de mensagens através destes protocolos, surge um novo conceito, conhecido como *Middleware.* Este termo se aplica à camada do software que provê a programação abstrata para mascarar a heterogeneidade das redes essenciais, hardware, sistemas operacionais e linguagens de programação (por exemplo: CORBA, Java RMI, RPC e *Sockets* são exemplos desta categoria).

Além de resolver problemas de heterogeneidade, o *middleware* fornece um modelo computacional único para ser usado pelos programadores de servidores e aplicações distribuídas. Esses modelos incluem: invocação remota de objetos, notificação remota de eventos, acesso remoto SQL e processamento de transações distribuídas.

O termo código móvel é usado para reportar ao código que pode ser enviado de um computador para outro e executá-lo em seu destino, como por exemplo Java applets.

A característica da flexibilidade de um computador determina se o sistema pode ser estendido e reimplementado de várias formas. A flexibilidade de um sistema distribuído é determinado primeiramente pelo grau no qual novos serviços de compartilhamentos podem ser adicionados e se tornarem úteis para uso pelos vários programas de clientes.

A flexibilidade não pode ser alcançada ao menos que a especificação e a documentação das interfaces principais do software dos componentes dos sistemas estejam disponíveis para os desenvolvedores do software. Este processo serve para padronização das interfaces.

Entretanto, este é um primeiro passo no que diz respeito aos sistemas distribuídos. O desafio dos projetistas é tentar resolver a complexidade dos sistemas distribuídos, que consistem nos muitos componentes feitos por diferentes pessoas.

Sistemas que são projetados para suportar o compartilhamento de recursos são chamados de sistemas distribuídos abertos para enfatizar o fato de que eles são extensíveis.

Para resumir:

- Sistemas abertos são caracterizados pelo fato de que as interfaces principais são publicadas;
- Sistemas distribuídos abertos são baseados na provisão de um mecanismo de comunicação uniforme e interfaces publicadas para se ter acesso aos recursos compartilhados;
- Sistemas distribuídos abertos podem ser construídos de software e hardware heterogêneos, advindos de diferentes fornecedores.

Muitas das informações que são disponíveis e mantidas nos sistemas distribuídos possuem um alto valor intrínseco para seus usuários. A segurança deles consequentemente se torna consideravelmente importante. A segurança das informações possui três componentes: confidencialidade, integridade e eficácia (COULOURIS et al., 2001).

Como a Internet permite que um programa em um computador se comunique com um outro programa em um outro computador, o risco de segurança está associado com a permissão do acesso livre a todos os recursos da Intranet. Embora um *firewall* possa ser usado para formar uma barreira na Intranet, restringindo o tráfego que entra e sai, isto não assegura o uso apropriado dos recursos pelos usuários na Intranet, ou o uso apropriado dos recursos da Internet, que não são protegidos pelo *firewall*.

Em sistemas distribuídos, clientes enviam requisições para acessar dados gerenciados por servidores, no qual envolve o envio de informações de mensagens através da rede.

O desafio é enviar a informação na rede de modo seguro. Mas a segurança não apenas está relacionada com envio seguro da mensagem, envolve também a identificação do processo emissor e o receptor. Esses desafios podem ser resolvidos através de técnicas de criptografia.

Um sistema é dito escalável se permanecer eficiente, quando há um aumento significativo do número de recursos ou usuários. A Internet é um exemplo claro de um sistema distribuído, pois possui um número significativo de computadores e um crescente aumento dos recursos.

Sistemas de computadores, de vez em quando, falham. Quando falhas ocorrem no hardware ou software, programas podem produzir

resultados incorretos ou podem parar antes que eles tenham completado a operação que foi estabelecida.

Falhas em sistemas distribuídos são parciais, isto é, um componente falha, enquanto os outros continuam trabalhando.

Algumas das técnicas para manipulação de falhas são (COULOURIS et al., 2001):

- *Detecção de falhas*: Algumas das falhas podem ser detectadas. O desafio é gerenciar, na presença da falha, o que não pode ser detectado, mas pode ser suspeito.
- *Mascarando falhas*: Algumas falhas que foram detectadas podem ser escondidas ou feitas menos graves.
- *Tolerância a falhas*: Muitos dos serviços na Internet apresentam falhas – não seria muito prático tentar detectar e esconder todas as falhas que ocorrerem em larga escala em uma rede de computadores. Os clientes podem aprender a tolerar falhas.
- *Recuperação das falhas:* A recuperação envolve o projeto de software, no qual dados permanentes podem ser recuperados, depois que um servidor sofreu algum problema.
- *Redundância:* Serviços podem ser tolerantes a falhas pelo uso da redundância dos componentes.

Os sistemas distribuídos possuem um alto grau de eficácia em relação às falhas de hardware. Quando um dos componentes falha, apenas o trabalho que estava realizando neste componente é afetado.

A presença de muitos usuários em um sistema distribuído é uma fonte de requisições concorrentes. Cada recurso deve ser planejado para ser seguro em um ambiente concorrente.

O objetivo é fazer certos aspectos dos sistemas distribuídos aparentarem ser invisíveis ao programador para que ele apenas fique concentrado em sua aplicação. Por exemplo, o programador não precisa estar concentrado na localização ou detalhes de como algumas operações são acessadas por outros componentes, ou se será replicada ou migrada.

Nos tópicos abaixo, serão explicadas duas formas de comunicação bastante usadas em sistemas distribuídos.

3.2 SOCKETS

Ambas as formas de comunicações (UDP e TCP) usam a abstração *sockets*, que tem a característica de prover um ponto final (*endpoint*) na comunicação entre os processos. O *socket* originou-se do BDS UNIX, mas hoje também está presente em alguns dos sistemas operacionais (COULOURIS et al., 2001). A comunicação de processos consiste na transmissão de uma mensagem entre um *socket* em um processo e um *socket* em um outro processo. Para um processo receber mensagens, o *socket* deve-se limitar à porta local e um dos endereços de Internet de um computador que fará a conexão. Mensagens enviadas para um endereço de Internet particular e um número de porta podem ser recebidos apenas pelo processo, no qual o *socket* está associado. Processos podem usar o mesmo *socket* para enviar e receber mensagens. Cada computador tem um grande número (2^{16}) de possíveis portas para uso dos processos locais para recebimento de mensagens. Qualquer processo pode fazer uso de múltiplas portas para receber mensagens, mas um processo não pode compartilhar portas com outros processos no mesmo computador. Entretanto, qualquer número de processos pode enviar mensagens para a mesma porta. Cada *socket* é associado com um protocolo particular – UDP ou TCP.

A figura 15 ilustra o funcionamento de uma conexão com *sockets*, onde o processo cliente possuiu um IP e é escolhida uma porta de comunicação com o processo servidor, para a mensagem ser passada.

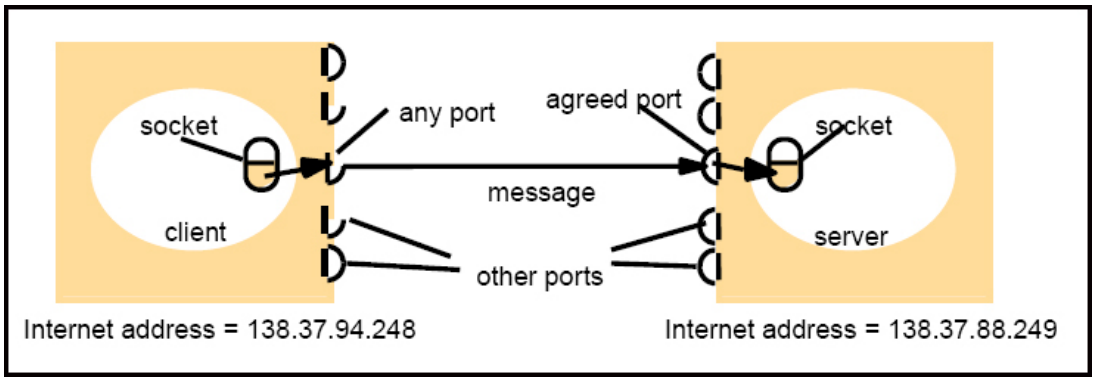

Fig. 15 – Portas e Sockets (COULOURIS et al., 2001, p. 129)

Como apresentado na figura 15, a conexão através de sockets entre cliente e servidor é única.

O *socket* utiliza a camada de transporte para a transmissão dos dados ou mensagens. Esta camada objetiva a transferência destes dados de forma segura e confiável, independente da rede que está sendo utilizada. É graças a esta camada que aplicativos podem ser desenvolvidos, através de alguns padrões, para funcionarem em uma variedade de redes, isto é, sem a preocupação com interfaces de sub-redes diferentes e uma possível transmissão não-confiável (TANEMBAUM, 2003 apud SILVA, 2006, p. 38).

Para permitir esta comunicação segura, é necessário que esta camada de transporte ofereça alguns serviços e operações, isto é, que haja uma interface de serviço para o transporte (SILVA, 2006). Esta interface é chamada de *Aplication Program* Interface – API.

Silva (2006) apresenta um exemplo genérico de um serviço de transporte. O objetivo é orientar a interface sobre quais conexões devem ser feitas.

Tabela 2. Primitivas básicas para um serviço de transporte simples(SILVA, 2006).

PRIMITIVA	SIGNIFICADO
LISTEN	Esperar um contato de um processo que queira se conectar.
CONNECT	Tentar ativamente estabelecer uma conexão.
SEND	Enviar informações/dados.
RECEIVE	Receber informações/dados
DISCONNECT	Pedir encerramento da conexão.

Para ilustrar o uso de um *socket*, é apresentado nas figuras 16 e 17, a implementação do *socket* na linguagem C++ e Java respectivamente.

```
1 try {
2     s = new SocketClient (addr, port);
3     s->SendBytes(comando);
4     printf(" Enviou: <<%s>>\n", comando);
5     s->Close();
6 }
7 catch (const char* s) {
8     std::cerr << s << endl;
```

```
9 }
10 catch (std::string s) {
11   std::cerr << s << endl;
12 }
13 catch (...) {
14   std::cerr << "unhandled exception\n";
15 }
```

Fig. 16 – Exemplo de uso de socket em C++

```
Socket s = new Socket(Proxy.NO_PROXY); will create a plain
socket ignoring any other proxy configuration.

Socket s = new Socket(new Proxy(Proxy.Type.SOCKS, new
InetSocketAddress("socks.mydom.com", 1080)));
```

Fig. 17 – Exemplo de uso de socket em Java

4. SISTEMAS DISTRIBUÍDOS DE REALIDADE VIRTUAL E AUMENTADA

Como já foi apresentado, ambientes de realidade virtual e aumentada podem ser analisadas sob um aspecto bem amplo, isto é, variar de apenas uma pessoa interagindo até multiusuários. Para isto, o uso de sistemas distribuídos se torna relevante. Podemos classificar este novo conceito como um sistema distribuído de realidade virtual ou aumentada, dependendo do caso.

Os sistemas distribuídos de Realidade Virtual permitem que usuários geograficamente distantes possam atuar em ambientes virtuais que estão compartilhados através de rede de computadores. O objetivo desses sistemas distribuídos de Realidade Virtual é resolver problemas de maneira colaborativa. Pessoas colaborando na consecução de tarefas podem fazer com que melhore o desempenho coletivo. Porém, uma característica que deve ser levada em conta é o tempo real das respostas, pois cada usuário deve ter a impressão de estar localizado em um mesmo lugar.

Além do trabalho colaborativo, os sistemas distribuídos de realidade virtual ou aumentada podem atuar no ramo dos negócios, entretenimento (jogos) e educação.

Segundo Rodello et al. (2007), na implementação de um projeto colaborativo, alguns requisitos devem ser analisados de acordo com a modalidade escolhida. Estas modalidades são: multiusuário, colaborativo, de grande escala e se a realidade virtual ou a aumentada é a interface de comunicação. Quando o ambiente é multiusuário, a infra-estrutura de uma rede deve ser usada para simular a interação em tempo real entre os usuários do sistema. Ambientes virtuais colaborativos preconizam que os participantes podem compartilhar um mesmo espaço virtual de trabalho, isto é, espaço, presença e tempo compartilhados. Ambientes virtuais de grande escala são aqueles de grandes proporções no requisito tamanho, quantidade de objetos presentes, quantidade de usuários e o nível de complexidade de seu gerenciamento.

Sistemas distribuídos de realidade virtual ou aumentada podem ser classificados de acordo com a quantidade de usuários que o sistema

suporta, isto é, se é monousuário ou multiusuários. No sistema monousuário, a existência de outros usuários pode ocorrer como espectadores, pois a navegação neste mundo virtual está disponível apenas para um participante. Já no ambiente multiusuário, a interação em tempo real entre os participantes se faz necessária, exigindo-se um alto poder de processamento e gerenciamento destes usuários (RODELLO et al, 2007).

Dependendo do tipo de sistema distribuído utilizado, o usuário é "transformado" em um objeto virtual, conhecido como avatar. Este avatar assume uma representação gráfica dentro do mundo virtual. Esta representação é relevante, pois os participantes devem ter a sensação e a percepção da presença de outros usuários no mundo virtual.

Tem-se, como exemplo claro, o uso do software SecondLife[5], que é um aplicativo de realidade virtual que simula o ambiente real. Agências de bancos já estão utilizando esta nova tecnologia como forma de realizarem reuniões, diminuindo custos de viagens e hospedagem dos participantes. A idéia é preparar o terreno para explorar, no futuro, as intermediações bancárias ligadas ao site, sobretudo as voltadas ao entretenimento. Existe uma forte tendência que grandes empresas explorem o uso deste sistema.

A comunicação nestes ambientes acontece por meio de gestos, textos ou falas.

Segundo Rodrigues et. al. (2004), um sistema distribuído de RV consiste em quatro componentes básicos:

- Displays gráficos;
- Dispositivos de comunicação e controle;
- Sistema de processamento;
- Rede de comunicação.

Os displays oferecem ao usuário a visão tridimensional do mundo virtual. Alguns exemplos são: monitores, HMDs e as CAVEs.

Os dispositivos de comunicação e controle possibilitam aos usuários manipularem objetos e se comunicarem com os demais participantes do mundo virtual. Estas tarefas são realizadas através de dispositivos de

[5] http://secondlife.com/

entrada, que podem ser: teclado, mouse, luva de dados (conhecida como *dataglove)* e microfones.

O sistema de processamento é responsável por receber os eventos destes dispositivos e processar os efeitos sobre o usuário e também nos objetos virtuais.

Por se tratar de um sistema distribuído, os participantes deste mundo virtual precisam estar em uma rede de comunicação para troca de informações. O objetivo da rede é proporcionar a sincronização do estado compartilhado neste mundo, levando-se em conta o tempo e visibilidade.

Rodello et. Al. (2007) apresentam alguns modelos de comunicação utilizados em ambientes colaborativos.

- Cliente/Servidor;
- Peer to Peer
- DIS
- VRTP
- DWTP
- Bluetooth

O modelo cliente/servidor implica na troca de mensagens direta entre o servidor e cliente e vice-versa. Torna-se impossível a comunicação entre os clientes nesta rede. Uma vantagem deste modelo é a segurança das informações e uma desvantagem pode ser uma possível falha no servidor, o que tornaria o sistema inoperante.

O modelo Peer to Peer implica na comunicação direta entre os clientes (chamados de *hosts)*, sem a necessidade da intervenção de um ou mais servidores. Um exemplo atual deste modelo são os compartilhadores de arquivos entre os usuários de Internet, como o Kazza, Emule e LimeWire.

O DIS (*Distributed Interactive Simulation*) surgiu da iniciativa de desenvolvimento de um sistema de treinamento de guerra com tanques, aeronaves, projéteis e outros que suportassem muitos computadores em rede (Macedônia, 1996 apud Rodello et. al 2007). Do desenvolvimento deste sistema surgiu um protocolo padrão para comunicação entre os usuários do ambiente.

O VRTP (*Virtual Reality Transfer Protocol*) é um protocolo onde os computadores conseguem executar várias tarefas ao mesmo tempo, isto é, ele pode ser um cliente, um *host* ou um *peer*. Os computadores têm direitos iguais.

O DWTP (*Distributed Worlds Transfer and Communication Protocol*) é um protocolo heterogêneo e independente da aplicação, isto é, dados de diferentes tipos podem ser trafegados.

E para finalizar, o bluetooth é uma rede sem fio (*wireless*) que permite a formação de um pequena rede com até 8 dispositivos. Estes dispositivos podem ser PDAs, Laptops, telefones móveis e computadores pessoais.

Tanto o ambiente colaborativo virtual como o aumentado objetivam a união de usuários geograficamente dispersos. Estes ambientes devem possuir um alto grau de realismo e que tenha um desempenho aceitável.

4.1 SISTEMAS DISTRIBUÍDOS DE REALIDADE VIRTUAL

Muito vem se pesquisando e desenvolvendo na área de sistemas distribuídos virtuais. Um exemplo é o DIVE[6], que é um ambiente desenvolvido pelo Swedish Institute of Computer Science. É um ambiente de realidade virtual com *n* usuários com o suporte de uma rede. Os participantes navegam em um cenário 3D e há uma interação com os demais participantes. A figura 18 ilustra este mundo virtual.

[6] http://www.sics.se/dive/

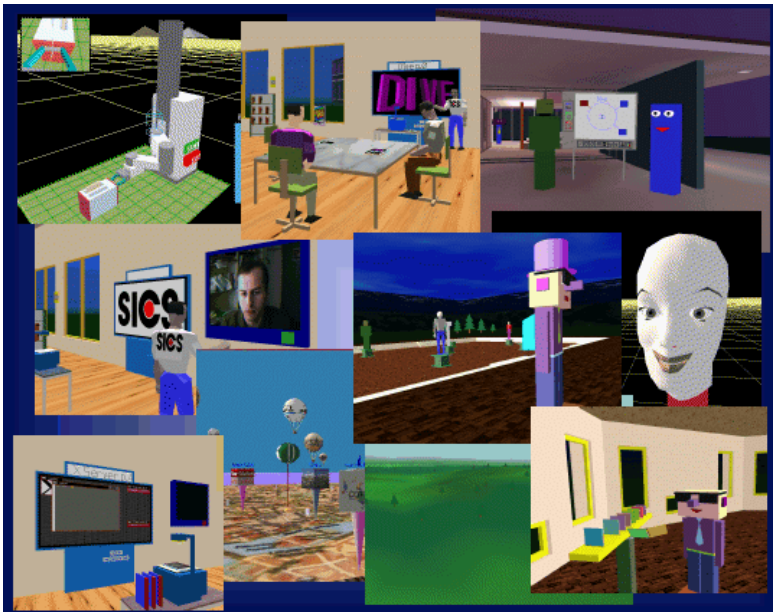

Fig. 18 – Sistema de RV (DIVE, 2007)

O Second Life, já citado anteriormente, está cada vez mais forte e sendo usado por grandes empresas e bancos. Este sistema simula o mundo real, e cada pessoa se torna um avatar. Este sistema foi desenvolvido como um jogo inicialmente, porém não há objetivos de finalização ou passagem de fases, e sim interagir com outras pessoas. A figura 19 apresenta uma visão deste ambiente.

Fig. 19 – Cenas do ambiente Second Life (Second Life, 2007)

Segundo Maia e Mattar (2007), o Second Life “é um ambiente colaborativo de realidade virtual, com interface 3D, em que é possível montar seu avatar, construir, comprar, vender objetos e que, por isso, possui sua própria moeda, o ‘lindendolar’ ”.

Este ambiente inovador traz consigo algumas potencialidades de uso como, por exemplo, em Educação a Distância, representando a sala de

aula presencial, porém com uma aula 3D, onde som, contato visual, slides e vídeos se fazem presentes.

Outro exemplo de aplicativo cujo uso se dá na educação é o projeto NICE, *Narrative Immersive Constructionist and Collaborative Environment.* Este projeto objetiva ensinar crianças a plantar e manter um jardim (JOHNSON et. al., 1998 apud KUBO et. al 2004). As figuras 20 e 21 ilustram o funcionamento do NICE.

Fig. 20 – Projeto NICE (NICE, 2007)

Fig. 21 – Projeto NICE (NICE, 2007)

4.1.1 COMPARAÇÕES

Esta seção apresenta uma tabela comparando os ambientes virtuais colaborativos apresentados anteriormente.

Tabela 3 – Comparação entre ambientes

Ambiente	Aplicabilidade	Modalidade	Modelo	Dispositivos
DIVE	Interação entre usuários através de uma interface 3D	Multiusuário	Cliente/Servidor	*Browser* com acesso a internet
NICE	Educação para crianças	Multiusuário	Cliente/Servidor	Aplicativo + HMD
Second Life	Interação entre usuários, marketing, troca e venda, jogo e educação.	Multiusuário	Cliente/Servidor	*Plugin* próprio deve ser baixado do site.

4.2 SISTEMAS DISTRIBUÍDOS DE REALIDADE AUMENTADA

Os sistemas distribuídos de RA não são desenvolvidos na mesma proporção dos de RV, pois são mais recentes e há dificuldades de gerenciamento dos elementos misturados.

Este tópico visa apresentar alguns sistemas colaborativos usando a realidade aumentada como interface.

4.2.1 SHAREDSPACE

Este projeto tem como característica a visualização de um *browser* de Internet tridimensional, possibilitando a colaboração multi-usuários. Estes, podem visualizar páginas flutuando no espaço. A escolha destas páginas são realizadas através da visualização independente das mãos. O usuário deve estar munido de um HMD e um sensor para rastrear sua posição (BILLINGHURST, 1996 apud SILVA, 2006 p. 34). As figuras 22 e 23 ilustram o SharedSpace.

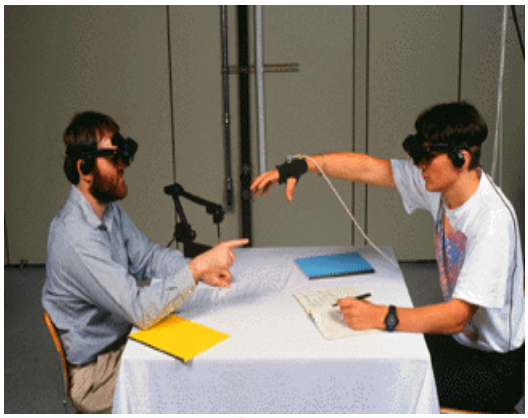

Fig. 22 - Usuários trabalhando (BILLINGHURST, 1996 apud SILVA, 2006)

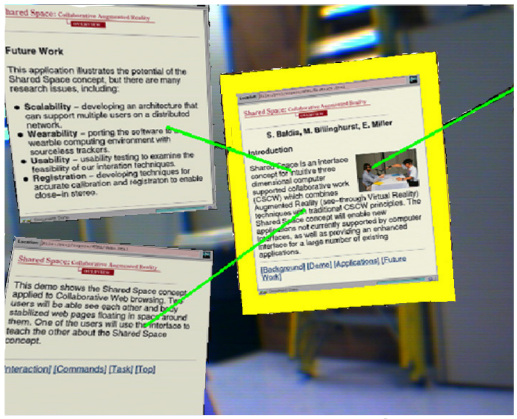

Fig. 23 - Visão do usuário (BILLINGHURST, 1996 apud SILVA, 2006)

4.2.2 CONSTRUCT3D

Este sistema foi projetado para ser aplicado na educação matemática geométrica. Para interação, é necessário um HMD e um PIP (*Personal Interaction Panel*), e é disponibilizado ao usuário opções de menus

que são selecionados por uma caneta especial, na qual é rastreada. A figura 24 apresenta este sistema (KAUFMANN, 2000 apud SILVA, 2006, p. 35).

Fig. 24 – Construct3D (KAUFMANN, 2004 apud SILVA, 2006)

4.2.3 ARTOOLKIT

O software ARToolkit foi desenvolvido inicialmente para rodar localmente. Através de modificações em código, é possível elaborar, dentre outros, sistemas colaborativos em rede. Um exemplo foi desenvolvido simulando um jogo da velha. Um marcador foi escolhido para ser associado com o tabuleiro virtual, e outros 2 marcadores para simbolizar os caracteres do jogo (KIRNER, 2007a). A figura 25 apresenta este sistema.

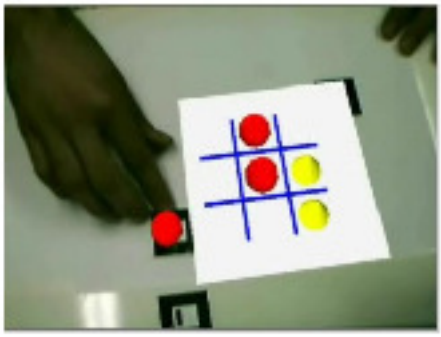

Fig. 25 – Jogo da velha em rede (KIRNER, 2007b)

4.2.4 MÃOS COLABORATIVAS

Kirner (2004a), em seu artigo: “Mãos Colaborativas em Ambientes de Realidade Misturada” mostra um exemplo de um sistema distribuído de RA “onde usuários podem realizar tarefas colaborativas com suas mãos, interagindo com objetos virtuais”, conforme apresentam as figuras 27 e 27.

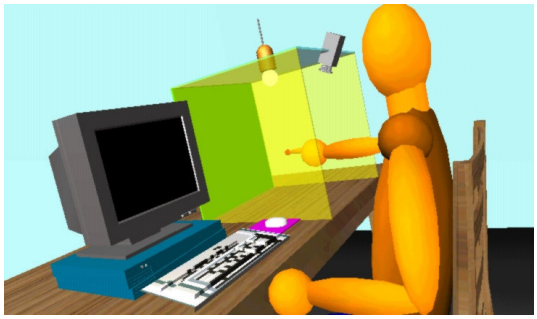

Fig. 26 – Projeto Mãos Colaborativas (KIRNER, 2004a)

Fig. 27 – Projeto Mãos Colaborativas (KIRNER, 2004a)

4.3 COMPARAÇÕES

Dentre os exemplos citados acima, pode-se concluir que há uma vasta gama de opções e aplicabilidades dos sistemas colaborativos usando a realidade aumentada como interface de comunicação.

A tabela 4 apresenta um comparativo entre os sistemas SharedSpace, Construct3D, ARToolkit e Mãos Colaborativas.

Tabela 4 – Comparação entre os sistemas colaborativos

Sistema	Aplicabilidade	Dispositivos
SharedSpace	Browser tridimensional	HMD
Construct3D	Educação matemática	HMD e PIP
Mãos Colaborativas	Colaboração	Monitor, web cam e marcadores
ARToolkit (jogo da velha)	Jogo	HMD ou web cam e marcadores

5. IMPLEMENTAÇÃO DO NETARTOOLKIT

Como explicado no tópico 2.4.1, a biblioteca ARToolkit oferece a possibilidade do desenvolvimento de aplicação de realidade virtual usando marcadores. Estes marcadores (disponíveis no pacote de instalação) servem como referência para o rastreamento óptico para a captura das imagens reais e sobre os quais são inseridos os elementos virtuais.

O NetArToolkit é um programa de realidade aumentada que usa a biblioteca ARToolkit e oferece suporte para aplicações em rede, permitindo a alteração de cenas escritas na linguagem VRML, atuando diretamente nas cenas pré-carregadas, usando as funcionalidades da biblioteca OpenVRML. Por exemplo, suponhamos que o usuário deseja apenas alterar a cor de um objeto já carregado. A aplicação simpleVRML não permite isso. A questão é que o programa não permite acesso ao grafo de cena, havendo a necessidade de remoção do objeto e da carga de um objeto semelhante e com a cor desejada.

A solução implementada é baseada no modelo cliente-servidor, em que o servidor é um subprograma da função principal de renderização, isto é, a cada laço do percurso do grafo de cena, o renderizador obtém um comando de uma fila de comando, interpreta e o executa, conforme ilustra a figura 28. Porém, há a necessidade de que a obtenção do comando para a respectiva inserção na fila seja um processo concorrente com o renderizador, dado que o processo de renderização não pode ficar parado à espera de entrada de dados e processo de leitura dos comandos nas interfaces de rede.

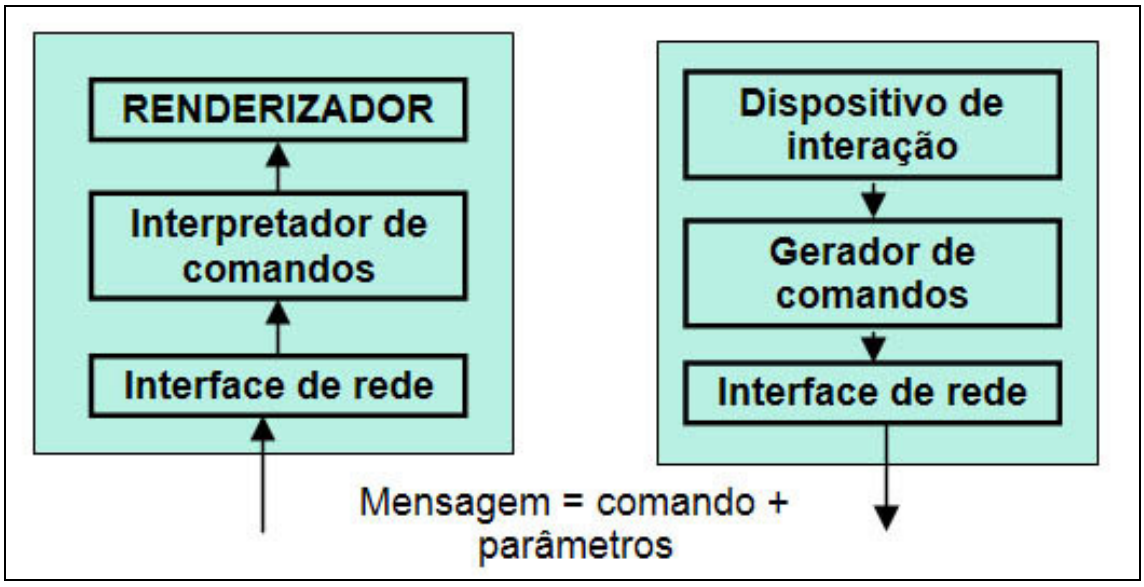

Fig. 28 – Modelo da Aplicação

O dispositivo de interação produz comandos, colocando-os na rede. A implementação inicial usa conexão baseada em *sockets*. Assim, os clientes podem ser implementados em quaisquer linguagens de programação, desde que a linguagem permita o uso de *sockets*. Estes comandos produzidos são transmitidos por uma interface de rede, através de um protocolo, havendo um interpretador de comandos, que lê os dados recebidos pela rede e atua na cena. O diagrama de blocos ilustrado na figura 29 apresenta o detalhamento desse modelo.

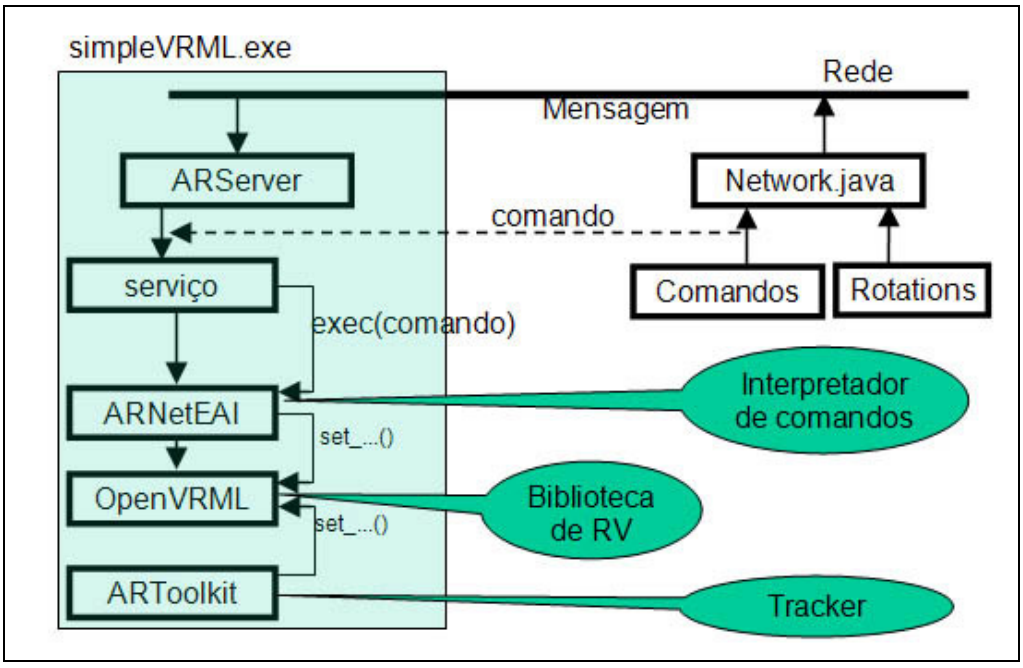

Fig. 29 – Módulos ativados pelo simpleVRML

Considerando-se o módulo “Interface de rede” da figura 28, fica definida a classe *ARServer*, que implementa o protocolo de *string* para mensagens trocadas entre cliente e servidor.

Os módulos *ARServer* e *ARNetEAI,* ilustrados na figura 29, são classes implementadas que servem para gerenciar a rede e interpretar os comandos dos clientes, respectivamente.

Esse novo modelo deve viabilizar alterações mínimas no algoritmo apresentado anteriormente na seção 2.4.2 página 35, gerando o algoritmo a seguir:

início

1. *Inicia as variáveis*
2. *Inicia a captura de vídeo*
3. *Inicia aplicações*

3.1. Define a função de gerenciamento de eventos
3.2. Define a função principal da renderização
3.3. Iniciar serviço de rede
3.4. Iniciar interpretador de comandos

fim

Fig. 30 – Algoritmo após a mudança

Quando um comando for recebido, através da classe *ARNetEAI*, será feito uma busca no grafo de cena que contém a representação dos arquivos VRML carregados e, em seguida, a cena será atualizada. Este código foi implementado conforme apresentado na figura 31:

```
1 VrmlScene *sc = sceneGraph->vrmlScene;
2 if (sc == NULL)return;
3 VrmlNamespace* ns = sc->scope();
4 if (ns==NULL)return;
5VrmlNodeMaterial* nd = (ns->findNode(node))->toMaterial();
6 if (nd==NULL)return;
7 VrmlSFFloat sh(value);
8 nd->setField("shininess", sh);
9 nd->render(sceneGraph);
```

Fig. 31 – Busca no grafo de cena e atualização da cena

Na linha 1 da figura 31, é feita a captura do grafo de cena através de um objeto. Em seguida, na linha 3, é capturado o escopo e na linha

5 é feita uma busca para se encontrar o nó. Na linha 9 é feita a renderização da cena virtual.

Testes foram realizados para ilustrar o funcionamento deste aplicativo, bem como verificar a possibilidade de se obter interação entre marcas, isto é, fazer com que um marcador possa ser utilizado como ferramenta para alteração da cena e não apenas como instrumento de navegação.

5.1 DESCRIÇÃO DO PROJETO

O compartilhamento em um sistema distribuído pode ocorrer localmente, ou seja, existe um determinado espaço físico, onde ocorrerá uma concorrência pela ocupação do espaço num dado momento. Como por exemplo, este espaço pode consistir em uma mesa, onde, ao redor desta mesa, são dispostos *n* usuários.

Existe também o compartilhamento entre usuários remotos, isto é, os usuários não precisam estar fisicamente no mesmo local. Visando atender e suprimir esta distância, as redes de computadores vêm preconizar e suprir esta distância.

Neste contexto de um espaço compartilhado remotamente, cada usuário pode estar conectado a um computador e possuir um *Head Mounted Display* (AZEVEDO, 2003). Os computadores estarão conectados entre si por intermédio de uma rede. Neste caso, cada um dos usuários poderá manipular o mesmo conjunto de marcadores e cada um terá a sua própria visão da cena em construção.

A colaboração se dá pela inserção, remoção ou alteração de cada uma das cenas associadas às marcas. Ao término da tarefa, é suficiente gerar o código VRML, fazendo-se o percurso no grafo de cena a partir do ponto de vista de um dos usuários. Portanto, considera-se que esses sejam os requisitos do ambiente colaborativo. Mas, espera-se que o espaço físico não seja restrito a um determinado local, isto é, espera-se que os usuários possam estar fisicamente distantes, considerando-se a capacidade de captura do HMD, e ainda assim, colaborarem na implementação de uma cena. Nesta situação, há marcas que não estão visíveis ou disponíveis para todos os usuários.

O desenvolvimento deste trabalho consiste em fazer uma extensão da biblioteca ARToolkit, de maneira tal que o espaço real seja observado em partes, conforme ilustra a figura 32. Cada uma destas partes será um recorte a ser compartilhado por um conjunto de usuários geograficamente distantes. A reconstrução das partes constitui o espaço da realidade aumentada em que todos os usuários colaboram. Nesse caso, técnicas de sistemas distribuídos serão experimentadas no desenvolvido do ambiente colaborativo.

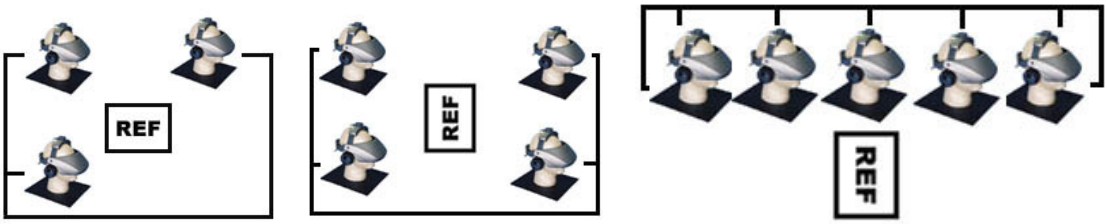

Fig. 32 – Visão de partes

Supondo que haja um conjunto de *n* marcadores por partes e que existam *m* partes*,* há um conjunto de *n*x*m* marcadores, constituindo o espaço virtual colaborativo. Neste caso, há dois problemas a serem tratados: (i) gerenciamento dos diversos pontos de vista dos usuários; (ii) gerenciamento do conjunto de marcas.

A biblioteca ARToolKit permite a determinação do ponto de vista da câmera, a partir de marcas fiduciais inseridas no espaço real. Dentre todos os marcadores possíveis, define-se que o marcador com o texto “REF” deve ser usado como referencial inercial para o cálculo de posicionamento das cenas.

O marcador “REF” permite que o espaço virtual seja composto a partir das partes, aplicando-se transformações homogêneas, caracterizando um espaço virtual único. O resultado do espaço, a partir da sobreposição virtual das marcas, está ilustrado na figura 33.

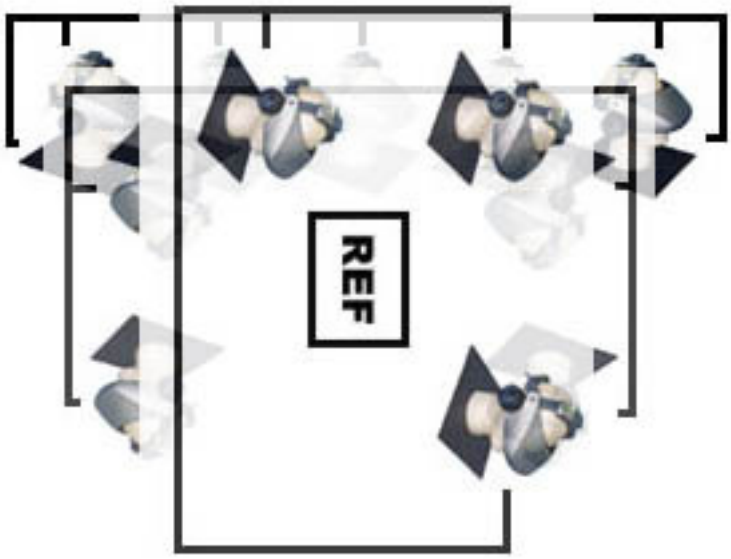

Fig. 33 – Junção das partes

A constituição do espaço global deve considerar que cada usuário tem o seu referencial e que recebe dos demais usuários as informações de seus respectivos referenciais, exigindo que sejam efetuadas as respectivas transformações de coordenadas para cada usuário. Por exemplo, para compor a visão global do usuário “Usr_1” da parte “P_2”, é necessário que ele receba atualização das informações de cada parte “P_1” e “P_3”. Assim, devem ser atualizados os atributos gráficos, a posição da marca de referência e posição de cada marcador no respectivo referencial, além de informação de visibilidade, inserção ou remoção de objetos novos não cadastrados na cena.

A aplicação dessa teoria matemática permite a determinação das coordenadas relativas entre as marcas, a partir de um referencial inercial. Esse problema foi resolvido por Kato (1999), que implementou a biblioteca de rastreamento de marcadores. Como resultado do rastreamento de marcas, obtém-se a posição de cada marcador em relação ao ponto de vista da câmera.

Neste caso, como o marcador “REF”, ilustrado na figura 32, fornece a visão da câmera de cada usuário, tem-se um referencial para cada subespaço, permitindo que se elaborem as respectivas transformações de coordenadas. Assim, é possível a obtenção da coordenada de cada um dos marcadores em relação ao referencial “REF”. Isto é necessário e suficiente

quando marcadores podem ter suas respectivas imagens capturadas a partir de uma única câmera, ou seja, estão suficientemente próximos. Quando os marcadores não atendem a esse requisito, é necessário que sejam analisados pontos de vista geometricamente distantes. Para isso, a organização lógica da obtenção das referências locais não deve ser alterada, mas há a necessidade de troca de informação entre os subsistemas de visualização, objetivando a construção da visão ilustrada na figura 33. Essa visão pode ser obtida se considerar que o conjunto de marcadores seja único e que cada subsistema é responsável pela atualização dos dados referentes a um subconjunto desse conjunto de marcadores.

Este gerenciamento e atualização dos marcadores deve ser estendido de maneira a permitir a identificação única dos elementos da cena. Localmente, os marcadores são cadastrados e registrados em um arquivo de configuração. Esse arquivo é lido quando a aplicação é iniciada. Como o cadastro é local, há a necessidade de um gerenciador para as marcas remotas. Cada marca possui um identificador local único. Assim, quando um usuário desejar iniciar suas atividades, sua aplicação local tratará de repassar para os demais usuários da rede sua identificação, composta de IP:*id*; onde IP seria o endereço Internet e *id* o identificador local das suas marcas.

Cada usuário deverá manter esta tabela de índice para efetuar a conversão da identificação do usuário remoto e o seu respectivo mapeamento no sistema de registro local, conforme ilustra a figura 34.

	Id de cena				
0	a	Identificadores locais			
1	b				
...					
15	c		Novo id	IP	Id Local
16	d	Identificadores remotos	16	10.1.192.1	0
17	e		17	192.10.1.1	0
18	f		18	10.1.192.1	1
19	g		19	192.10.1.1	1
20	h		20	10.1.192.1	2
21	i		21	165.10.1.1	0
22	j		22	165.10.1.1	1
23	k		23	165.10.1.1	2
24	l		24	10.1.192.1	3
25	m		25	10.1.192.1	4

Fig. 34 – Tabela de Índices

Neste exemplo, o marcador de índice 17 fornece a chave de acesso à cena “e”, mas o índice 17 deve ser obtido a partir do número IP “192.10.1.1” que possui identificador local “0”. Analogamente, o conjunto completo de identificadores pode ser construído usando-se uma tabela hash.

5.2 IMPLEMENTAÇÃO DOS COMANDOS

Para testar a comunicação em rede e verificar se o comando enviado alcança seu objetivo, envia-se pela rede um comando para atuar em outro ponto e verificar se o que foi esperado foi realizado.

A título de ilustração desse potencial, apresenta-se o uso da função set_rotation. Essa função permite a modificação da cena, conforme ilustram as figuras 36 e 37. A figura 36 ilustra o objeto na marca padrão, antes de qualquer modificação, e a figura 37 ilustra a alteração da cena, após a execução do comando “set_rotation Neck 0 1 0 1.57”. A função *set_rotation* tem como argumentos o identificador do node e 4 variáveis que correspondem ao quatérnio do campo *rotation* referente ao node *Transform* da linguagem VRML. A figura 35 ilustra apenas uma interface que chamamos Cliente onde

será feita a comunicação com o Servidor. Nesta interface, o endereço IP, a porta e a mensagem devem ser colocados.

Como citado anteriormente, a biblioteca ARToolkit foi desenvolvida na linguagem C++. Por meio da implementação do Cliente na linguagem Java, pode-se concluir que qualquer linguagem, desde que esta tenha suporte a *sockets,* pode fazer a comunicação com o ARToolkit. Este teste alcançou o objetivo esperado.

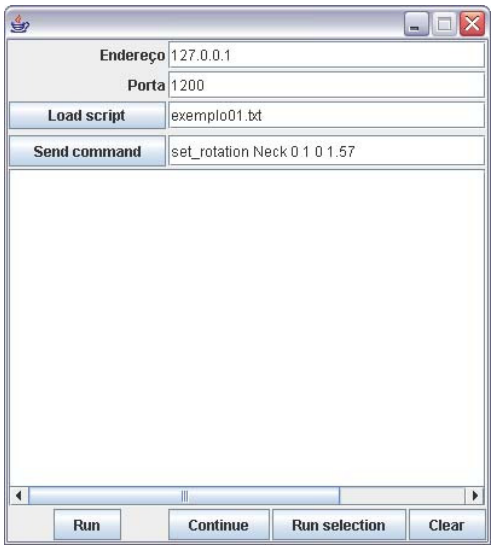

Fig. 35 – Cliente desenvolvido em Java

Fig. 36 – Objeto virtual carregado na cena

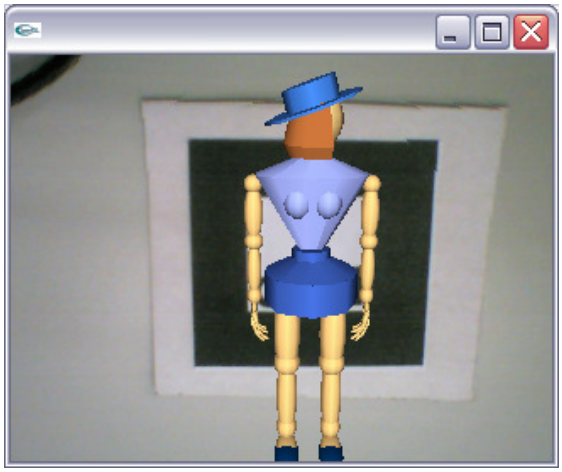

Fig. 37 – Objeto virtual após o envio do comando set_rotation

O envio do comando pela rede acontece em três etapas: Criação do socket (linha 4), envio do comando (linha 5) e fechamento da conexão (linha7) conforme ilustra a figura 38. Esta função está implementada na classe ARNetEAI, e recebe como parâmetros o endereço ao qual será enviado o comando, o número da porta e a mensagem (linha 1).

```
1 void ARNetEAI::send(char *addr, int port, char *msg)
2{
3   try {
4     s = new SocketClient (addr, port);
5     s->SendBytes(comando);
6     printf(" Enviou: <<%s>>\n", comando);
7     s->Close();
8   }
9   catch (const char* s) {
10      std::cerr << s << endl;
11   }
12   catch (std::string s) {
13      std::cerr << s << endl;
14   }
15   catch (…) {
16      std::cerr << "unhandled exception\n";
17   }
18}
```

Fig. 38 – Função send do NetARToolkit

Para cada comando da linguagem VRML, há uma função que decodifica este comando e atualiza o grafo de cena. No exemplo citado acima, a função set_rotation foi acionada para atualização do objeto. Esta função é descrita na figura 39.

```
1void ARNetEAI::set_rotation(const char* node, float x,
2float y, float z, float r){
3    VrmlScene *sc = sceneGraph->vrmlScene;
4    if (sc == NULL)     return;
5    VrmlNamespace* ns = sc->scope();
6    if (ns == NULL)     return;
7    VrmlNode* p_no = ns->findNode(node);
8    if (p_no==NULL) return;
9    VrmlNodeTransform* nd = p_no->toTransform();
10   if (nd == NULL) return;
11   VrmlSFRotation pos(x, y, z, r);
12   nd->setField("rotation", pos);
13   pos = nd->getRotation();
14   nd->render(sceneGraph);
15}
```

Fig. 39 – Função set_rotation

A seguir, é apresentado a lista de comandos da linguagem VRML que podem ser enviados pela rede, atuando diretamente no grafo de cena.

Tabela 5 – Lista de comandos da linguagem VRML

Set_marker
printNameSpace
Set_object
createFromString
Set_translation
Set_rotation
Set_scale
Set_diffuseColor
Set_transparency
Set_emissiveColor
Remove
Set_shininess
Set_diffuseColorAll

Set_emissiveColorAll
Set_transparencyAll
saveNameSpace
Set_InterTranslation

As figuras 40 e 41 apresentam o uso de dois comandos citados na tabela 5. Os demais comandos seguem a mesma estrutura de envio pela rede.

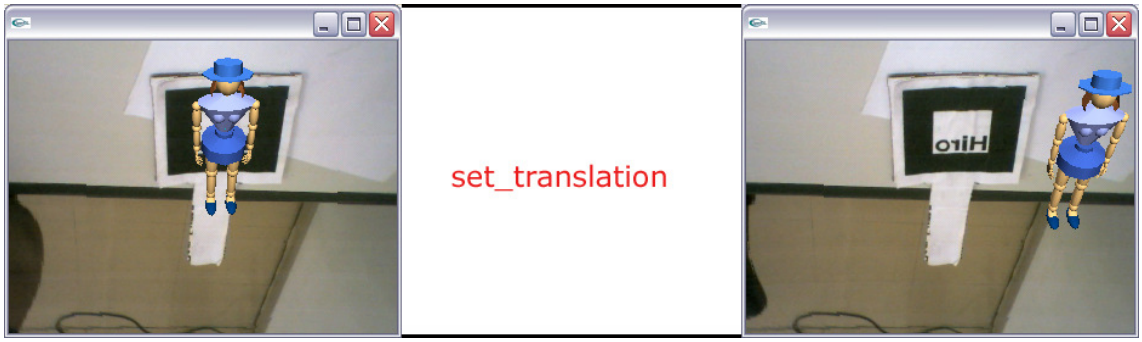

Fig. 40 – Execução do comando set_translation

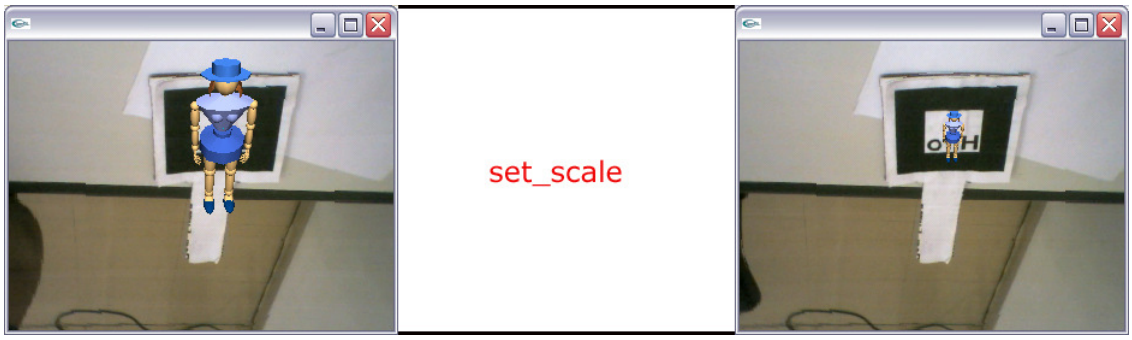

Fig. 41 – Execução do comando set_scale

Outra possibilidade da aplicação deste aplicativo é a interação entre marcadores, isto é, fazer com que um marcador possa ser utilizado como ferramenta para alteração da cena e não apenas como instrumento de navegação.

A posição das marcas em relação à câmera é armazenada na estrutura de dados *Object*. O valor padrão para o número de marcas define um vetor de 30 posições, que pode ser alterado, onde cada posição aponta para uma outra estrutura que guarda os dados de posição de cada marca. Portanto, cada marca possui uma matriz de transformação 3x4, denominada “trans”, que parte da estrutura *Object*, de acordo com o formato ilustrado na Tabela 6.

Tabela 6 – Matriz de Transformação 3x4

R00	R01	R02	Tx
R10	R11	R12	Ty
R20	R21	R22	Tz

Os elementos identificados por R00, R01, R02, R10, R11, R12, R20, R21 e R22 definem os argumentos da rotação da marca em relação ao referencial da câmera. Os elementos Tx, Ty e Tz representam o afastamento da marca em relação à câmera.

Um exemplo de aplicação desta matriz está descrito na seção 2.3.

A interação entre as marcas corresponde à aplicação de uma dada funcionalidade (associada à uma das marcas), aplicada à outra marca. Por exemplo, ao se fazer a atribuição "object [3]→trans[0][3] = object[0]→trans[0][3]", o objeto que está na posição 3 do vetor *object*, passará para as coordenadas definidas para a marca que está cadastrada na posição "0" do vetor *object*.

As marcas são detectadas por chamadas às funções da biblioteca ARToolkit na função principal. Assim, a interação com o uso de marcas não necessariamente gera mensagens.

Como exemplo de interação entre as marcas, implementou-se a rotação do pescoço do boneco quando ocorre a alteração da posição da marca que está associada ao cone. Quando a marca cadastrada no object[1] muda de posição, isto é, alguém atua sobre esta marca, o pescoço do boneco que esta cadastrado na marca object[0] sofre uma rotação com os valores da marca 1. Este exemplo poderia ser com qualquer outro comando da linguagem VRML, como por exemplo: translação, mudança de cor, etc.

Na figura 42, é apresentado o objeto virtual modificado com a atuação na marca, fazendo com que o pescoço do boneco sofra uma rotação a cada mudança da outra placa.

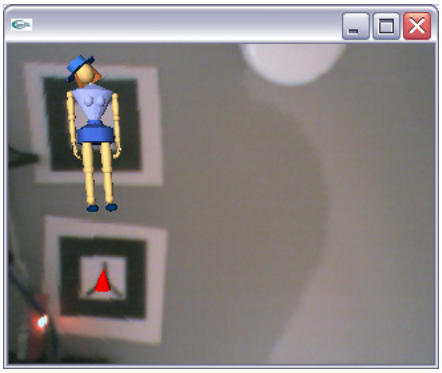

Fig.42a – Pescoço do boneco rotacionando de acordo com as posições da marca no object[1]

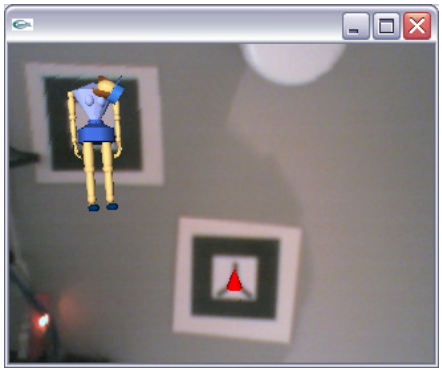

Fig. 42b – Pescoço do boneco rotacionado de acordo com as posições da marca no object[1]

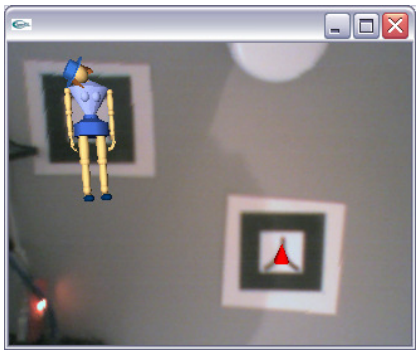

Fig. 42c – Pescoço do boneco rotacionando de acordo com as posições da marca no object[1]

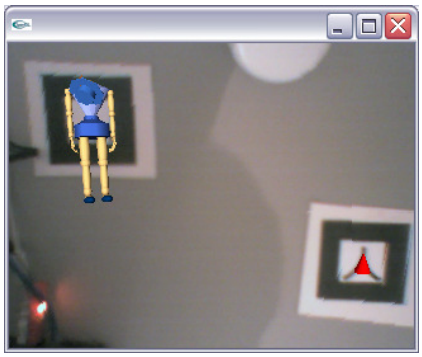

Fig. 42d – Pescoço do boneco rotacionando de acordo com as posições da marca no object[1]

Isso mostra que é possível interagir com diversas placas de diversas maneiras diferentes, utilizando as técnicas de sistemas distribuídos.

5.3 PERSISTÊNCIA

Uma importante funcionalidade implementada no aplicativo foi a introdução da Persistência. Esta tem por objetivo fixar o objeto virtual na cena sem a necessidade do marcador.

O processo é feito da seguinte forma:

- O marcador é inserido na cena;
- É acionada a persistência através de um botão;

- O marcador é retirado da cena e o objeto fica estacionado na última posição em que o marcador esteve visível.

Quando o aplicativo passa pela função “draw”, que é responsável pelo desenho da cena virtual, é feito uma condição para verificar se a cena está persistente ou não. Em caso afirmativo, a posição do marcador em relação à câmera é gravada em uma matriz e o objeto é desenhado nesta posição. A função de acionar a persistência e a função de verificação da condição na função “draw” estão descritas nas figuras 43 e 44, respectivamente.

```
1 static void   keyEvent( unsigned char key, int x, int y)
2 {
3 if( key == (char)'A' )
4    persistente = 1;
5 if( key == (char)'a' )
6    persistente = 0;
7 }
```

Fig. 43 – Atalho para acionar a persistência

```
1 for( i = 0; i < objectnum; i++ ) {
2if(object[i].visible==0&&object[i].persistente==0)continue
3    if(i>0 && object[i].persistente==1){
4         arUtilMatMul(matInv,object[i].trans,matDiff);
5         arUtilMatMul(object[0].trans,matDiff,matResult);
6         argConvGlpara(matResult, gl_para);
7         draw_object( &object[i], gl_para );
8    }
9    else {
10        argConvGlpara(object[i].trans, gl_para);
11        draw_object( &object[i], gl_para );
12   }
13 if(netObject[i].visible==0&&netObject[i].persistente==0)
14        continue;
```

```
15    argConvGlpara(netObject[i].trans, gl_para);
16    draw_object( &netObject[i], gl_para );
17    }
```

Fig. 44 – Código da função "draw"

5.4 METODOLOGIA, MATERIAIS E INFRA-ESTRUTURA

Através de revisões bibliográficas, estudos, orientações e testes no software ARToolkit, foi possível analisar e modificar códigos presentes na biblioteca.

O aprofundamento teórico se deu por meio de livros, artigos, sites e periódicos.

Os materiais e infra-estrutura física são pertencentes à UNIMEP, sendo que testes foram realizados no Laboratório de Realidade Virtual.

Foram coletados dados sobre o desempenho do ambiente colaborativo na rede, através do uso de *sockets*, que serão apresentados no capítulo 6.

Através do aprofundamento teórico e técnico sobre o NetARToolkit, experimentos como interação entre marcadores, compartilhamento de marcadores e interação entre eles, foram testados bem como o estudo da comunicação na rede e performance.

6. AVALIAÇÃO E TESTES

Este capítulo objetiva a apresentação de testes realizados no NetARToolkit, usando a rede como meio de comunicação. Estes testes foram relevantes para medir o desempenho na rede, comunicação, atraso, rotação e translação.

Para testar o desempenho na rede, foi elaborado a seguinte problematização: Medir quantos pacotes trafegavam na rede à cada laço de renderização da cena na função principal em um tempo de 10 segundos, movendo-se a placa a uma distância de 30 centímetros, conforme ilustra a figura 47. Este teste foi importante para fazer uma comparação entre o desempenho e a tolerância.

Estes pacotes equivalem a quantidade de vezes que o programa passa pela função mainLoop. Esta função pode ser representada pelo fluxograma apresentado na figura 45. Logo no início desta função, há uma condição de verificação do ponteiro da câmera, que é chamado de dataPtr. É verificado se este ponteiro está nulo, e, em caso afirmativo, há um retorno para o início da função. O objetivo é funcionar como um filtro, isto é, só mandar pacotes quanto este ponteiro não estiver nulo. A figura 46 ilustra este código.

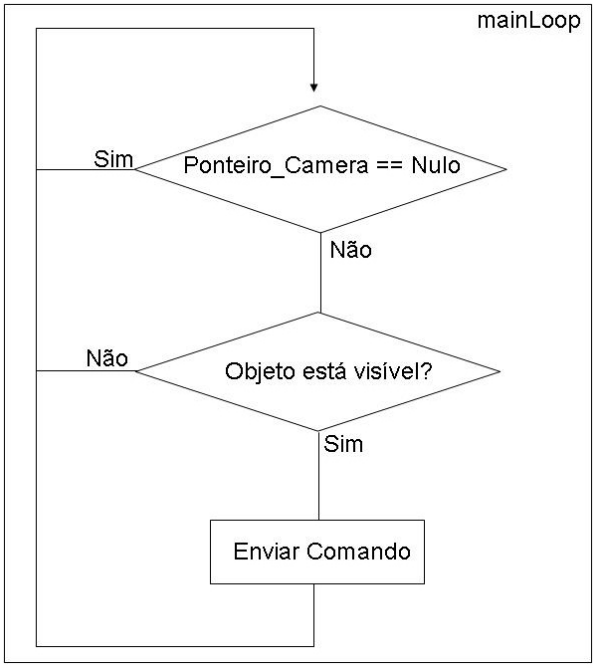

Fig. 45 – Fluxograma do mainLoop

```
1if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
2         arUtilSleep(2);
3         return;
4}
```

Fig. 46 – Filtro no início da função mainLoop

Um primeiro teste foi implementado colocando-se um contador, antes desta função, e um após e o aplicativo ficou rodando durante 1 minuto. Durante este tempo, o contador 1 que estava antes da função de verificação, contabilizou 4298 passagens pelo mainLoop. Já o contador 2, que estava após a função de verificação, contabilizou 662 passagens. Isto mostra um ganho de 85%. Esta função já é uma forma de ganho de pacotes na rede, pois o comando só é enviado pela rede quando um frame é capturado. Se não existisse esta função muitos pacotes se perderiam na rede, pois em alguns momentos não seria possível capturar o frame da câmera.

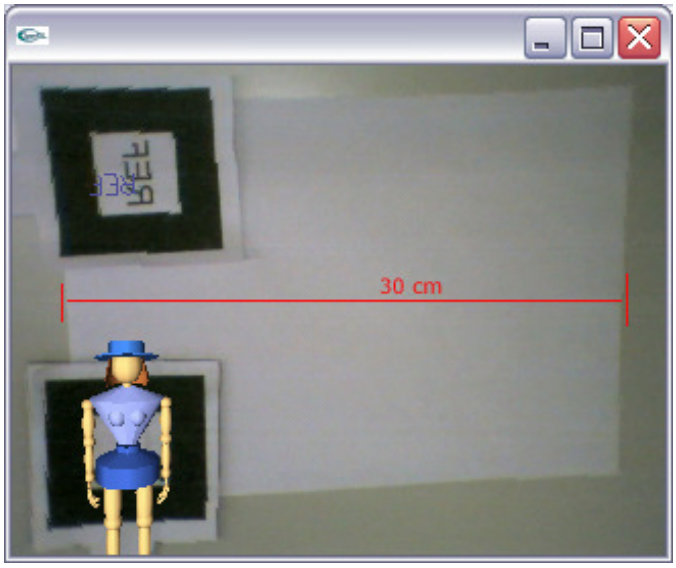

Fig. 47 – Espaço para realização do teste

Utilizando um movimento com o marcador durante o tempo de 10 segundos, foram enviados 100 pacotes. Este envio ocorrendo após a função de verificação contabilizou 690 realizando o mesmo movimento.

O resultado obtido, no teste de desempenho, é apresentado na tabela 7.

Tabela 7 – Pacotes enviados

Movimento	Distância	Envio do comando	Pacotes enviados
10s	30 cm	após a condição	100
10s	30cm	antes da condição	690

Com o elevado número de pacotes enviados pela rede, mesmo após a função de verificação, pode ocorrer um certo atraso na construção de uma cena virtual. Para minimizar o tráfego na rede, pensou-se em usar uma tolerância de movimentação, isto é, se o usuário mover o marcador em até x cm, não é enviado o pacote. A variável x pode representar 1mm,2mm,....nmm, sendo que se o movimento for maior que esta tolerância, o pacote é enviado.

A implementação desta função utilizou-se da seguinte lógica. É capturada a matriz do marcador, e esta é a cada laço verificada se a movimentação deste marcador é maior que a tolerância estipulada. Se for maior, o pacote é enviado e a nova posição é atribuída à matriz de comparação.

Após a implementação desta nova funcionalidade, o fluxograma da função mainLoop ficou representado, conforme ilustra a figura 48.

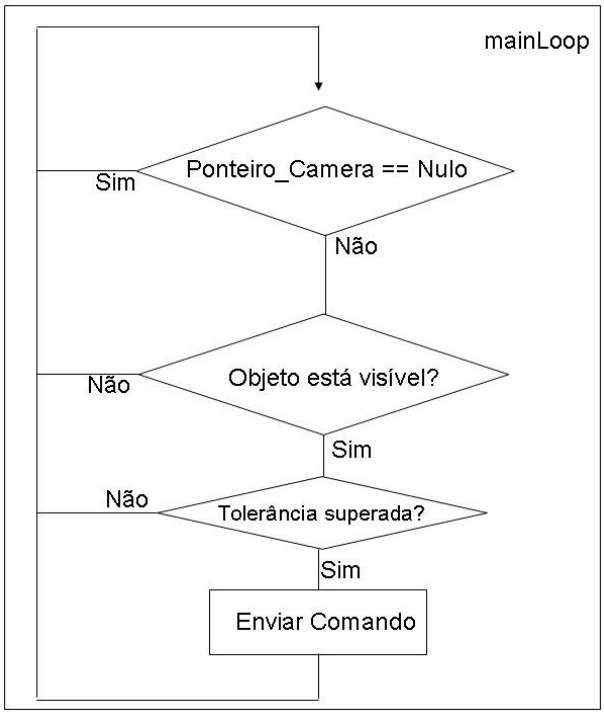

Fig. 48- Fluxograma do mainLoop após a modificação

A tabela 8 apresenta testes, utilizando o mesmo movimento na mesma distância, porém com tolerâncias diferentes.

Tabela 8 – Testes com tolerância

Distância	Tolerância	Pacotes enviados
30cm	2 cm	15
30cm	4 cm	7
30cm	8 cm	3
30cm	20 cm	1
30cm	40 cm	0

A comparação entre a tabela 7 e 8, apresenta a diferença entre o número de pacotes trafegados pela rede, concluindo que quanto maior a tolerância, menor o tráfego de rede. Neste teste, a variável tempo torna-se irrelevante, pois o foco principal é a distância percorrida do marcador de acordo com sua tolerância. Na tabela 7, numa distância percorrida de 30 cm, foram enviados pela rede 100 pacotes, independente se o marcador estava visível ou não ou sendo movimentado ou não. Quando há o cadastro de uma tolerância, o pacote só será enviado se atingir esta margem. Neste caso apenas 15 pacotes foram enviados tomando-se em conta a tolerância de 2 cm. Percebe-se um ganho de tráfego na rede de 85%.

Por se tratar de valores pequenos e levando-se em conta o rápido processamento das máquinas atuais, conclui-se que o trabalho colaborativo não será atrapalhado pelo valor de tolerância.

Esta tolerância foi implementada da seguinte forma:

```
1if(object[idPatt].visible){
2     arUtilMatInv(matLast, matInv);
3     arUtilMatMul(matInv, object[idPatt].trans, matResult);
4     refX=matResult[0][3];
5     refY=matResult[1][3];
6     refZ=matResult[2][3];
7     dist= refX*refX+refY*refY+refZ*refZ;
8     if(dist>(500)){
9          for(int i=0;i<3;i++){
```

```
10          for(int  j=0;j<4;j++)
11          matLast[i][j]=object[idPatt].trans[i][j];
12          }
13    return 1;
14}
```

Fig. 49- Implementação da tolerância

E por fim, foi calculado a taxa de *frames* por segundo. Este cálculo foi realizado como apresentado no código abaixo.

```
      fRcount/arUtilTimer();
```

A variável fRcount é do tipo inteiro e a cada percurso na função mainLoop é acrescido de 1. Para cálculo do número de *frames* por segundo, esta variável é dividida pela função do ARToolkit arUtilTimer, cujo objetivo é contar em segundos o tempo de execução do aplicativo.

O resultado manteve uma média de 10.95 frames por segundo, conforme apresentado na figura 50.

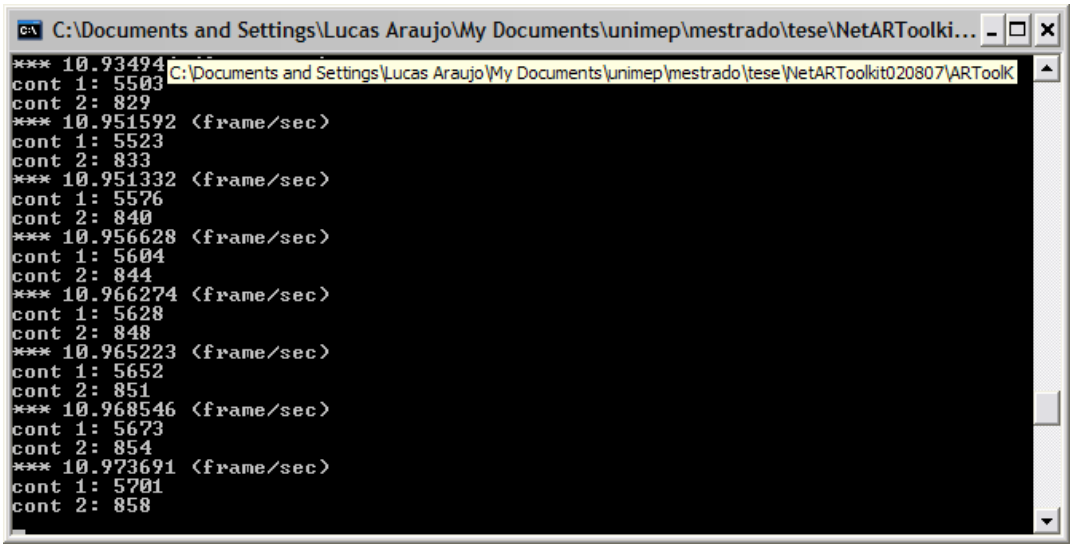

Fig. 50 – Frames/segundo na máquina 1

Este teste foi realizado em um computador cujo processador é de 3.33 GHz Pentium 4. Para comparação e verificação que a velocidade e potência de uma máquina é relevante, foi executado este aplicativo em duas máquinas diferentes cujos resultados são apresentados nas figuras 51 e 52.

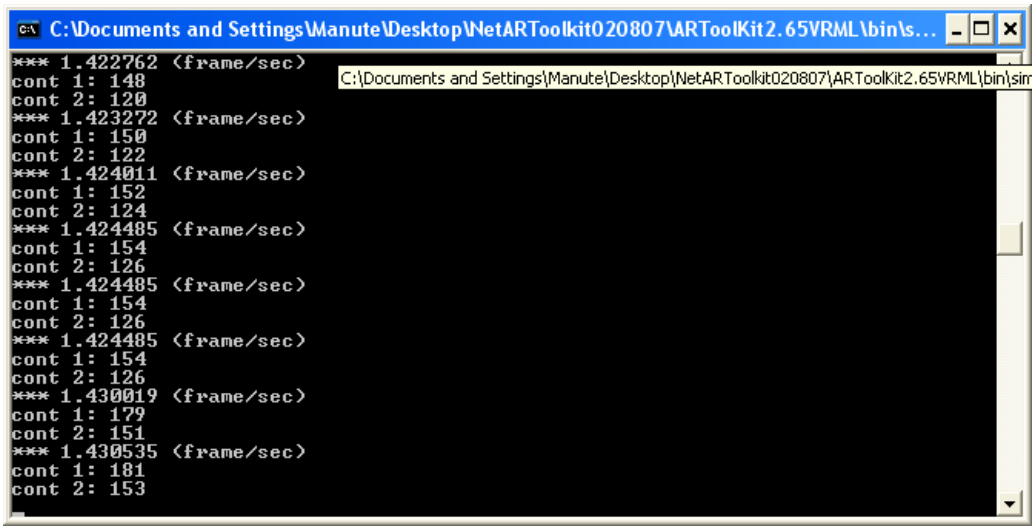

Fig. 51 – Computador 2 (Processador 2,66 GHz)

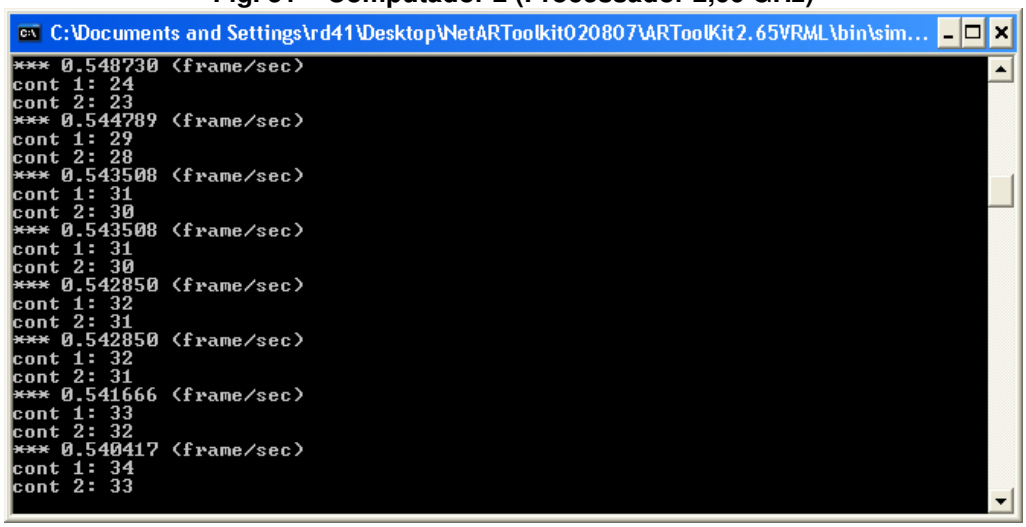

Fig. 52 – Computador 3 (Processador 1,8GHz)

A máquina que foi utilizada para realização dos testes apresentados na figura 51, foi um Pentium 4 – 2,66 GHz e o da figura 52 também foi um Pentium 4 porém com o processador de 1,8 GHz. O número baixo de pacotes pode ter sido em virtude de outros motivos como, a velocidade da interface USB para captura da câmera e da lentidão ocasionado pelo alto número de arquivos depositados nesta máquina.

A figura 53 apresenta um gráfico comparativo entre os computadores testados, em relação a taxa de *frames*/segundo.

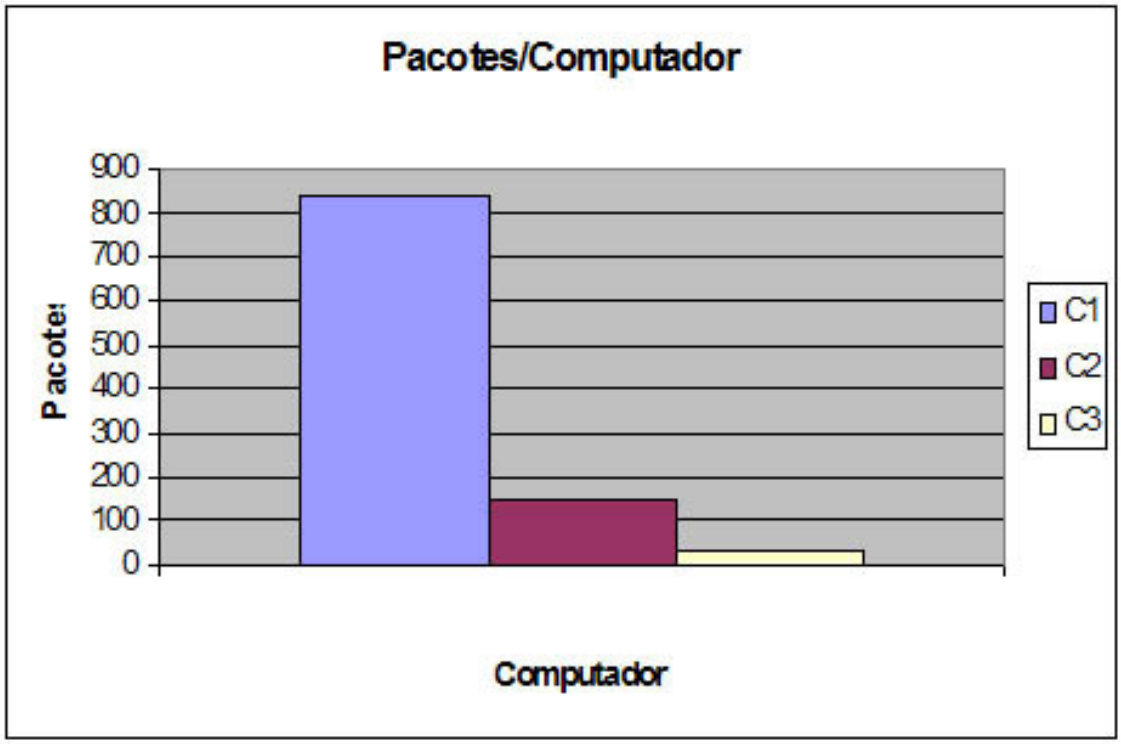

Fig. 53 – Gráfico dos pacotes enviados por máquina

Para implementação da função do marcador “REF”, isto é, realizar o compartilhamento entre as partes, foi elaborado uma função que tem por objetivo mudar as bases das matrizes realizando as transformações. Este cálculo no NETARToolkit foi realizado utilizando o comando arUtilMatMul. Esta função que realiza o cálculo está descrita na figura 54.

```
1 if(i>0 && object[i].persistente==1){
2     arUtilMatMul(matInv,object[i].trans,matDiff);
3     arUtilMatMul(object[0].trans,matDiff,matResult);
4     argConvGlpara(matResult, gl_para);
5     draw_object( &object[i], gl_para );
6}
```

Fig. 54 – Multiplicação de matrizes

Este comando de escolha *if* está dentro de um laço *for,* na qual a verificação feita é: se o marcador existe e se ele está capturado pelo *frame*. Em caso afirmativo, a mudança de base é feita e o objeto é desenhado na tela.

6.1 EXEMPLO DE USO DO NETARTOOLKIT

A título de ilustração do potencial de uso desta ferramenta, foi pensado em construir um cenário virtual colaborativo com os usuários distantes fisicamente, utilizando a rede como meio de comunicação.

Este cenário será uma floresta, contendo uma casa e uma rodovia. Cada marcador estará associado a um objeto.

Primeiramente, deve-se colocar o aplicativo no ar. Para isto, o primeiro passo a fazer é configurar o IP das máquinas que estarão interagindo.

Para configurar a rede de comunicação, acesse o arquivo r_hosts.txt disponível no diretório NetARToolkit020807\ARToolKit2.65VRML\bin.

Digite o IP das outras máquinas que farão parte da construção da cena colaborativa e salve o arquivo. O número 1202, como apresentado na figura 55, indica o número da porta que será criado o *socket*. Este número deve ser semelhante a todos da rede.

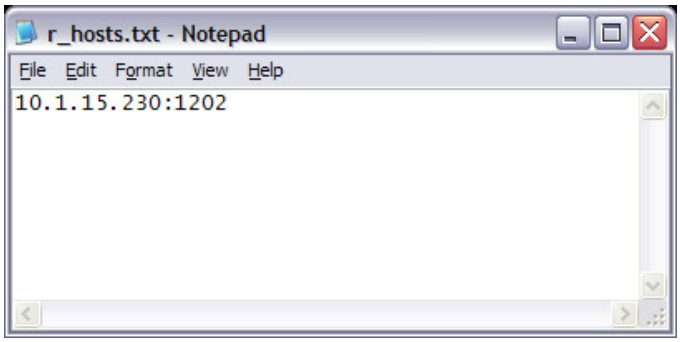

Fig. 55 – Arquivo r_hosts.txt

Finalmente, para rodar o aplicativo basta clicar no arquivo Rotations.bat disponível da raiz da estrutura. Este arquivo bat nada mais é que a chamada do arquivo simpleVRML do diretório NetARToolkit020807\ARToolKit2.65VRML\bin.

O cenário virtual foi configurado da seguinte forma: O marcador Kanji está associado à uma arvore, figura 56 e o marcador hiro está associado a uma casa, conforme ilustra a figura 57.

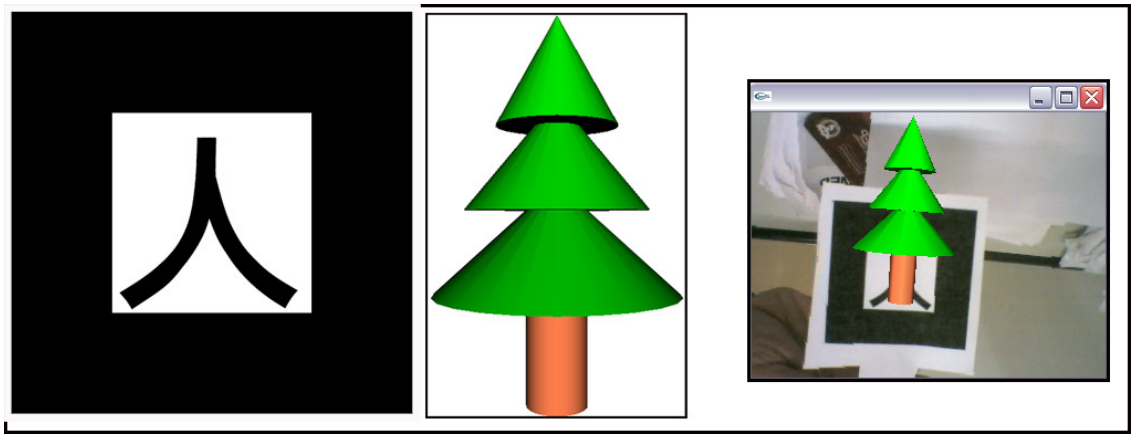

Fig. 56 – Marcador kanji associado ao objeto virtual

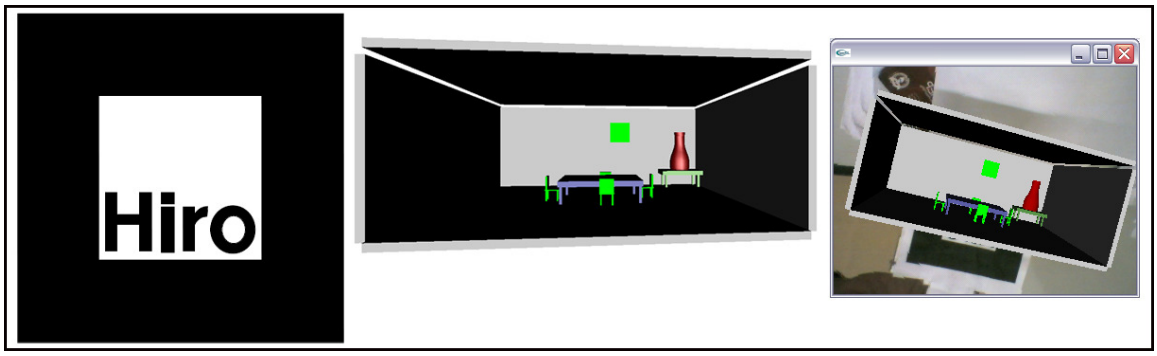

Fig. 57 – Marcador hito associado ao objeto virtual

Estando o aplicativo em execução, basta deixar visível o marcador “REF” e o marcador referente ao seu objeto virtual.

A figura 58 apresenta a cena vista do computador 1, onde o objeto virtual é a casa.

Fig. 58 – Usuário 1 disponibilizando a casa virtual

A figura 59 apresenta o objeto virtual local, neste caso a casa virtual, mais o objeto virtual vindo da rede, que é a árvore.

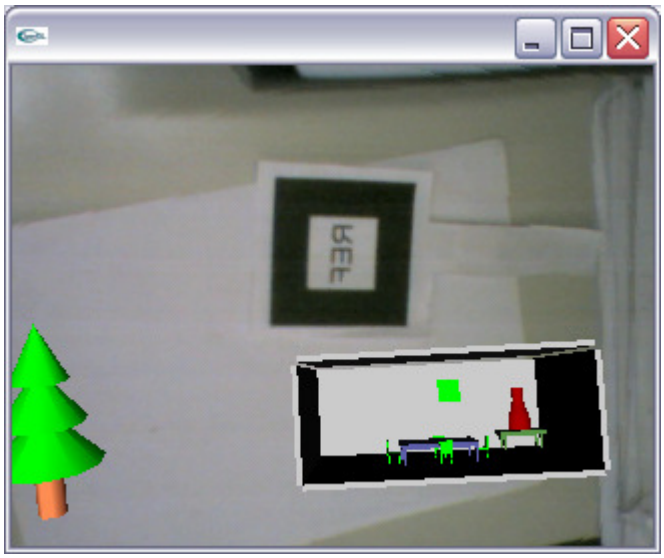

Fig. 59 – Visão do usuário 1 composta de seu objeto mais o da rede

Todos os objetos virtuais são alocados de acordo com o referencial. Caso o usuário deseje ver a cena de um ponto de vista diferente, basta movimentar o marcador “REF” que todo o cenário virtual será movimentado, conforme apresenta a figura 60.

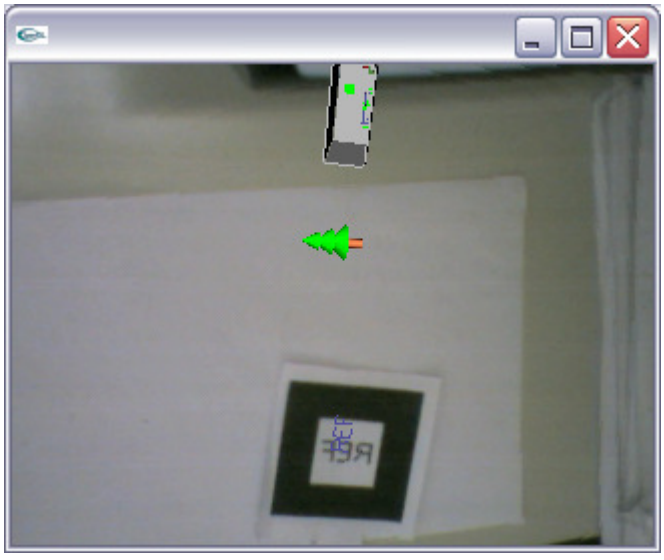

Fig. 60 – Cena virtual

7. CONCLUSÕES

A evolução tecnológica, característica cada vez mais atual nos dias de hoje, proporciona novos recursos e novas técnicas, possibilitando o uso de realidade virtual e aumentada, que permite o desenvolvimento de interfaces poderosas de navegação e interação ampliadas com o uso de sistemas distribuídos, através do compartilhamento de informações entre usuários e a colaboração mútua. Esta pesquisa teve como objetivo desenvolver um suporte de um ambiente colaborativo de realidade aumentada, na qual um espaço real possa ser observado em partes, de forma que cada parte seja um recorte a ser compartilhado por um conjunto de usuários. Visando a reconstrução destas partes, técnicas de sistemas distribuídos foram experimentadas no desenvolvimento deste ambiente colaborativo.

O suporte mostrou-se, através de testes realizados em rede e alterações em código, flexível e com bom potencial para ampliar o uso do ARToolkit para o desenvolvimento de aplicações de realidade aumentada em ambiente de rede.

Dentre as atividades realizadas no aplicativo, destacam-se as alterações no código do ARToolkit, diminuição do tráfego de rede com estratégias de sensibilidade à espaçamento, testes de tráfego e de quadros/segundo e testes de funcionamento.

A contribuição e a importância desta pesquisa mostraram a viabilidade do trabalho colaborativo entre usuários usando a RA. Devido ao fato do trabalho solitário tornar-se cansativo e desestimulante, esta nova pesquisa visa a interação virtual entre estes usuários, com a exploração no tridimensional em um ambiente 3D, com o objetivo da construção de uma idéia comum entre usuários.

Esta aplicação, como trabalho futuro, pode ser estendida com vistas a um melhor desempenho. Um trabalho a ser realizado refere-se ao gerenciamento do marcador que deve ser estendido de maneira a permitir a identificação única dos elementos da cena, de forma conjunta com o desenvolvimento de uma interface que colha os dados dos usuários. Isto quer dizer que quando um usuário desejar iniciar suas atividades, automaticamente

sua aplicação tratará de repassar aos demais usuários as respectivas identificações, como por exemplo seu IP e o número do identificador local dos marcadores.

Referências Bibliográficas

AZEVEDO, E.; CONCI, A. **Computação Gráfica:** Teoria e Prática. Editora Campus, Rio de Janeiro, 2003.

AZUMA, R. T. **A Survey of Augmented Reality**. Teleoperators and Virtual Environments, p. 355-385, 1997.

BURDEA, G.; COIFFET, P. **Virtual Reality:** Technology Second Edition. Editora Wiley Interscience, 2003.

CADOZ, C. **Realidade Virtual**. Editora ática, 1997.

CALONEGO, N. GARCIA, M. MEIGUINS, B. NETTO, A. CATERIANO, P. Modelagem e Programação de Ambientes Virtuais Interativos. In: KIRNER, C. TORI, R. **Realidade Virtual:** Conceitos e Tendências. São Paulo. Editora Mania de Livro, 2004. p. 95-108.

CARDOSO, A.; LAMOUNIER JÚNIOR, E. A Realidade Virtual na Educação e Treinamento. In: KIRNER, C. TORI, R. **Realidade Virtual:** Conceitos e Tendências. São Paulo. Editora Mania de Livro, 2004b. p. 259-264.

CARDOSO, A.; LAMOUNIER JÚNIOR, E. **Realidade Virtual**: uma abordagem prática. Editora Mania de Livro, 1ª. edição, São Paulo, 2004a.

CONSULARO, L. et al. **ARToolkit:** Aspectos Técnicos e Aplicações Educacionais. 2006a.

CONSULARO, L.: **VRML** – *Tutorial baseado no FloppyVRML*. Disciplina de Rastreamento de Realidade Virtual. Mestrado em Ciência da Computação – UNIMEP. 20/04/06b. Notas de Aula.

COULOURIS, G; DOLLIMORE, J.; KINDBERG, T. **Distributed Systems**: Concepts and Design. 3 ed. 2001.

HitLabNZ. ARToolkit. Disponível em < http://www.hitlabnz.org/wiki/Home >. Acesso em 05/10/07.

KATO, H. 1999. **Marker Tracking and HMD Callibration for a Video-based Augmented Reality Conferencing System**. Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality, pp. 85-94, Outubro 20-21, San Francisco, 1999.

KIRNER, C. "Mãos Colaborativas em Ambientes de Realidade Misturada". Anais do Workshop de Realidade Aumentada – 2004a. Piracicaba, SP, p. 1-4.

KIRNER, C. TORI, R. Introdução à Realidade Virtual, Realidade Misturada e Hiper Realidade. In:_____. **Realidade Virtual:** Conceitos e Tendências. Editora SBC. São Paulo – SP, 2004b, p. 3-20.

KIRNER, C. KIRNER, T. Virtual Reality and Augmented Reality Applied to Simulation Visualization. In: SHEIKH, A. AJEELI, A. ABU-TAIEH, E. **Simulation and Modeling:** Current Technologies and Applications. IGI Global, 2008.

KIRNER, C. MENDES, S. **Sistemas Operacionais Distribuídos:** Aspectos Gerais e Análise de sua Estrutura. Editora Campus. Rio de Janeiro-RJ, 1988.

KIRNER, C. SISCOUTTO, R. Fundamentos de Realidade Virtual e Aumentada. In: _____. **Realidade Virtual e Aumentada:** Conceitos, Projetos e Aplicações. Petrópoli-RJ: Editora SBC, 2007a. Cap. 1, p. 02 - 21.

KIRNER. Realidade Virtual. Disponível em <http://www.realidadevirtual.com.br> Acesso em 02/05/07b.

KUBO, M. MEIGUINS, B. GARCIA, M. OLIVEIRA, L. TORI, R. Aplicações de Ambientes Virtuais Colaborativos. In: KIRNER, C. TORI, R. **Realidade Virtual:** Conceitos e Tendências. São Paulo. Editora Mania de Livro, 2004. p. 311-320.

MAIA, C.; MATTAR, J. **ABC da EaD:** A educação a distância hoje. Editora Pearson Prentice Hall. São Paulo, 2007

MILGRAM, P. KISHINO, F. **A Taxonomy of Mixed Reality Visual Displays**. *IEICE Transactions on Information Systems* E77-D (12): 1321-1329, 1994.

NICE. The Narrative Immersive Constructionist/Collaborative Environments project. Disponível em <http://www.evl.uic.edu/tile/NICE/NICE/intro.html>. Acesso em 05/11/07.

OLIVEIRA, L. CALONEGO N. **Suporte para Autoria Colaborativa com Realidade Aumentada.** Mostra Acadêmica – UNIMEP, 2006, p. 1-4.

OLIVEIRA, L. CALONEGO N. **Uma Aplicação Cliente-Servidor usando ARToolkit.**. Workshop de Realidade Aumentada. Rio de Janeiro, 2006, p.1-4.

OMG. Object Management Group. Disponível em <http://www.omg.org> Acesso em 13/04/07.

POOLE, D. **Álgebra Linear**. Editora Thomson, 2004.

Realidade Aumentada. Disponível em: <http://www.realidadeaumentada.com.br> . Acesso em 20/02/07.

RODELLO, I. SEMENTILLE, A. BREGA, R. NUNES, F. Sistemas Distribuídos de Realidade Virtual e Aumentada. In: KIRNER, C. SICOUTTO, R. **Realidade Virtual e Aumentada:** Conceitos, Projetos e Aplicações. Petrópoli-RJ: Editora SBC, 2007. Cap. 7, p. 129 - 150.

RODRIGUES, L. KUBO, M. RODELLO, I. SEMENTILLE, A. TORI, R. BREGA, J. Ambientes Virtuais Distribuídos. In: KIRNER, C. TORI, R. **Realidade Virtual:** Conceitos e Tendências. São Paulo. Editora Mania de Livro, 2004. p. 43-60.

SABBATINI, R. Informática Médica – Volume 2 Disponível em <http://www.informaticamedica.org.br/informaticamedica/n0202/sabbatini.htm>. Acesso em 26/03/07

Second Life. Disponível em < http://secondlife.com/>. Acesso em 25/05/07.

SILVA, R. **Ambiente Colaborativo com Realidade Aumentada**. Piracicaba: UNIMEP, 2006. 94 f. Dissertação (Mestrado em Ciência da Computação). Universidade Metodista de Piracicaba, Piracicaba. 2006.

SINCLAIR, P. Disponível em <http://www.equator.ecs.soton.ac.uk/projects/artoolkit/> Acesso em 04/05/07.

SourceForge .Disponível em <http://www.eden.net.nz/phil/develop/artoolkit>. Acesso em 04/09/06

TANENBAUM, A.S. **Distributed Operating Systems**. 1. ed. New Jersey: Prentice-Hall. 1995.

TEICHRIEB. V. Realidade Virtual & Multimídia. Disponível em <http://www.di.ufpe.br/~if124/vrml/vrml.htm>. Acesso em 29/03/07

TORI R.; KIRNER C.;SISCOUTO R. **Fundamentos e Tecnologia de Realidade Virtual e Aumentada**. Editora SBC. Belém – PA, 2006.

VENTURELLI, S. Animação Computacional. Disponível em <http://www.arte.unb.br/museu/animac.htm>Acesso em 13/04/07.

ANEXO I – MANUAL DE INSTRUÇÕES

Manual de Orientação - NetARToolkit

Autor: Lucas de Araújo Oliveira

Piracicaba
2008

MANUAL DE INSTRUÇÕES

Este manual objetiva detalhar um passo-a-passo de como instalar e manipular o NETARToolkit, ferramenta que suporta o trabalho colaborativo em rede com o uso da realidade aumentada.

REQUISITOS DO SISTEMA

Para o acesso é necessário

- Computador com acesso à *Internet*;
- Web Cam.

INSTRUÇÕES PARA INSTALAÇÃO DO APLICATIVO

Para instalação do NETARToolkit, siga os seguintes passos:

1) Descompacte o arquivo ARToolkit2.65.zip em algum diretório de sua preferência. Após este processo, os arquivos serão distribuídos conforme ilustra a Figura 1.

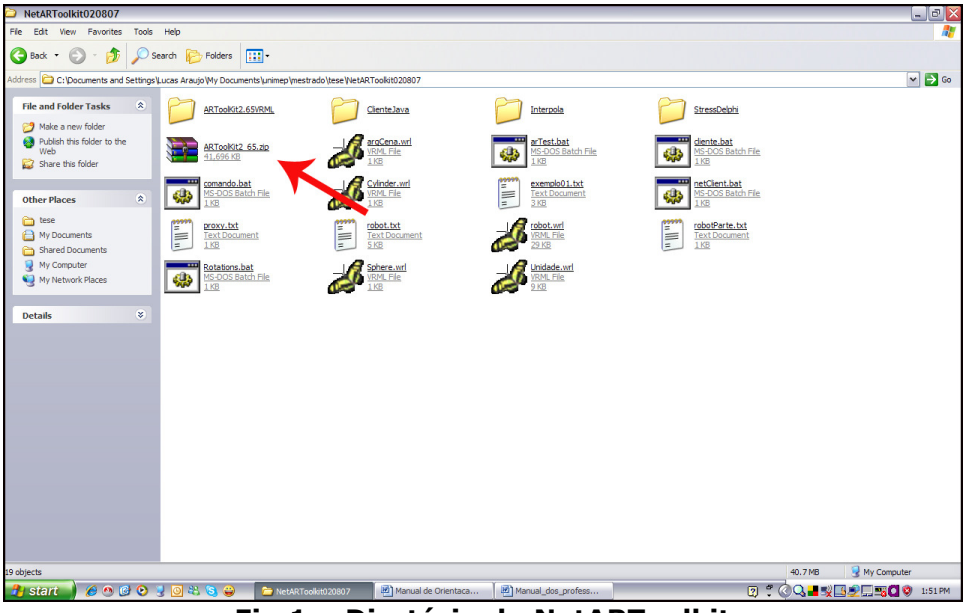

Fig.1 – Diretório do NetARToolkit

2) Para facilitar a execução do aplicativo, foi criado um arquivo "bat". A interface de comandos está nomeado como "comando.bat". Clicando neste arquivo, será aberto a interface Java denominada Cliente que envia os

comandos pela rede. Nesta interface, deve ser preenchido o número da porta, o IP e a mensagem.

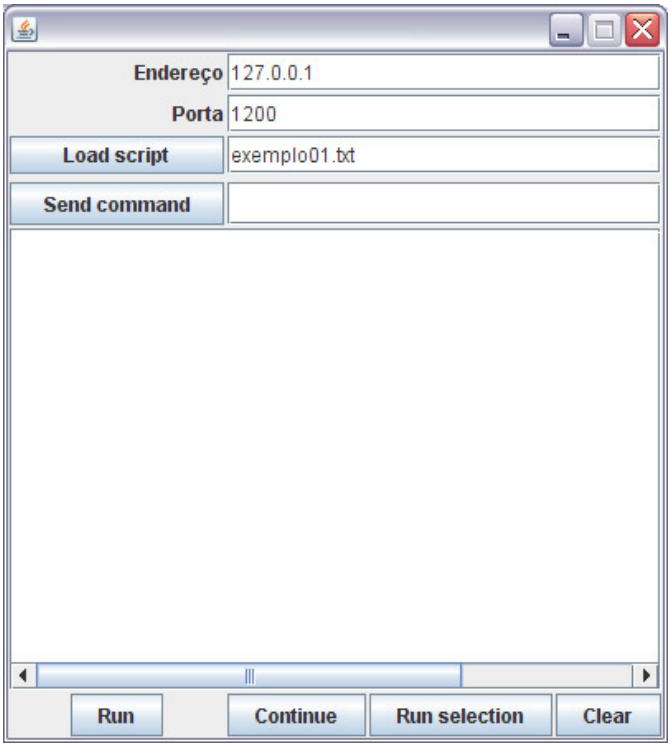

Fig. 2 – Interface do Cliente

3) Para executar a aplicação simpleVRML, também foi criado um arquivo bat para facilitar sua execução. O nome deste arquivo é: simple.bat e uma web cam deve estar conectada ao computador. Para visualização dos objetos virtuais, deverão ser impressos os marcadores, que estão disponíveis no diretório ARToolKit2.65VRML\patterns. A figura 3 apresenta este diretório e a figura 4 apresenta a execução do aplicativo simpleVRML.

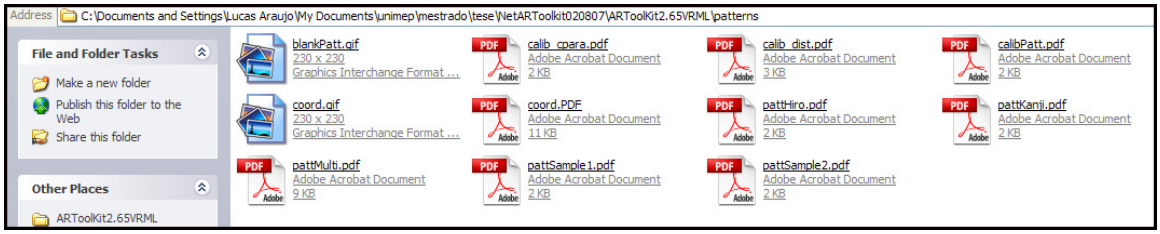

Fig. 3 – Diretório dos marcadores

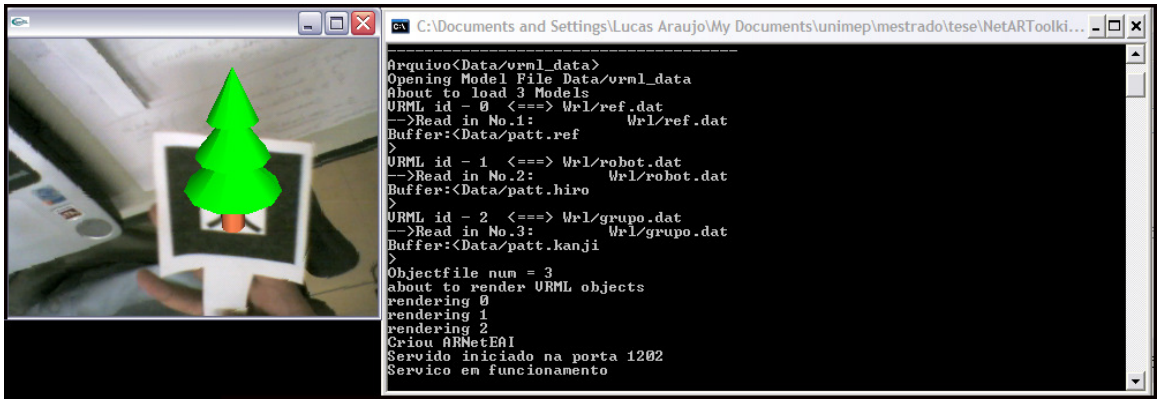

Fig. 4 – Execução do simpleVRML

4) Algumas configurações são feitas de maneira manual, como por exemplo a configuração das máquinas que farão parte do ambiente colaborativo. Para isto, o necessário é saber o IP de cada uma delas. Acesse o Prompt-Dos, digitando a palavra "cmd" na linha de execução do windows, isto é, clique em Iniciar->Executar, conforme ilustra a figura 5.

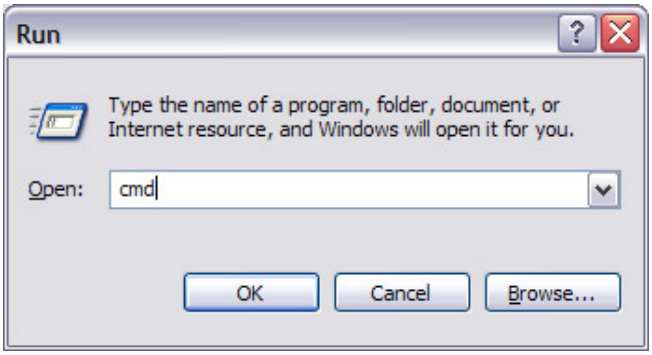

Fig. 5 – Linha de comandos do windows

Para saber seu IP, digite ipconfig no DOS. Em nosso exemplo, apresentado na figura 6, o IP da máquina é: 10.1.15.89.

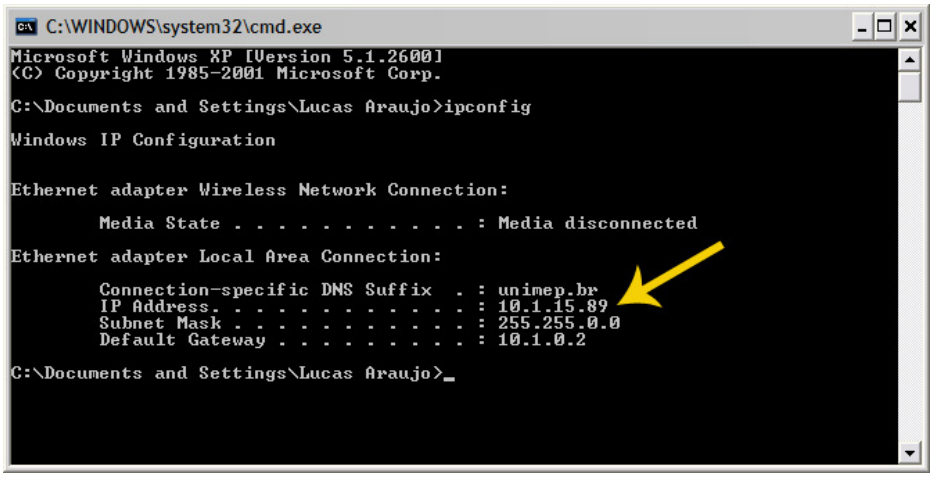

Fig. 6 – IP da máquina

5) Os IPs das máquinas que farão parte do ambiente deverão ser configuradas no arquivo r_hosts.txt disponível no diretório ARToolKit2.65VRML\bin. Deverá ser colocado o IP seguido de dois pontos(:) e o número da porta. Este número deve ser igual para todos. O IP da sua máquina não deverá ser preenchido neste arquivo.

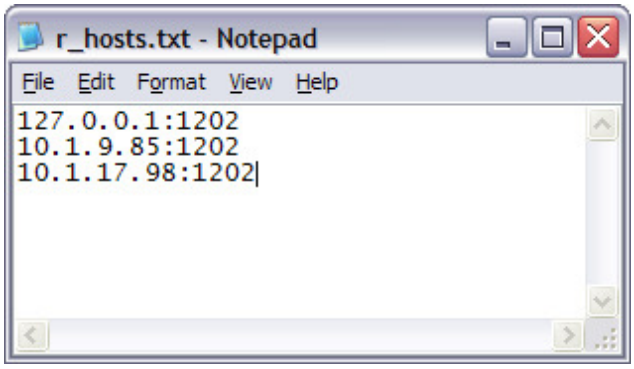

Fig. 7 – Arquivo r_hosts.txt

6) Por se tratar de um ambiente livre, todo o código está disponível no diretório ARToolKit2.65VRML\examples\simpleVRML_Server e o projeto principal está nomeado como simpleVRML.dsw, conforme apresenta a figura 8. Para modificações, deve-se ter o compilador do Microsoft Visual C++ 6.0.

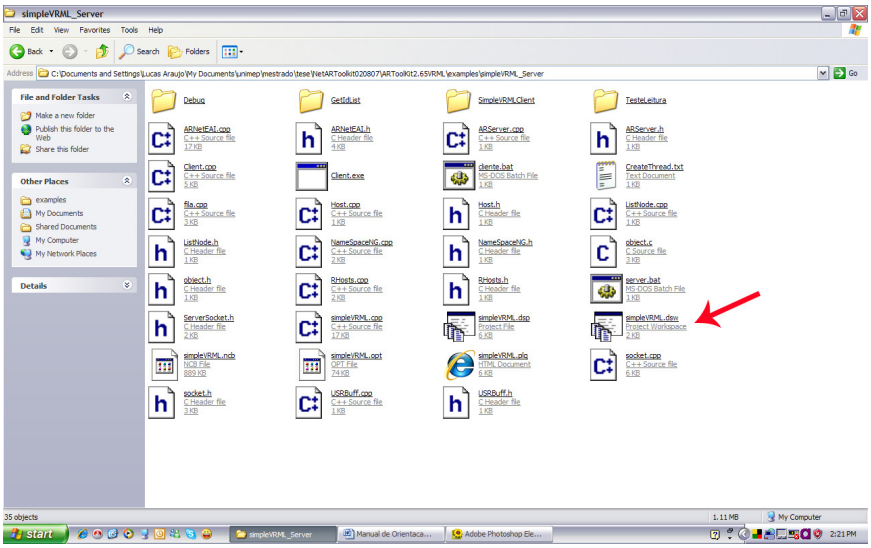

Fig. 8 – Diretório do projeto com os códigos

2.4.2.1 Execução e instalação

O ARToolkit está disponível para quase todas as plataformas. Para cada uma das plataformas existe uma versão diferente.

Após a instalação, disponível em http://www.eden.net.nz/phil/develop/artoolkit, será criado uma árvore de diretório em seu disco local, como ilustra a figura 17.

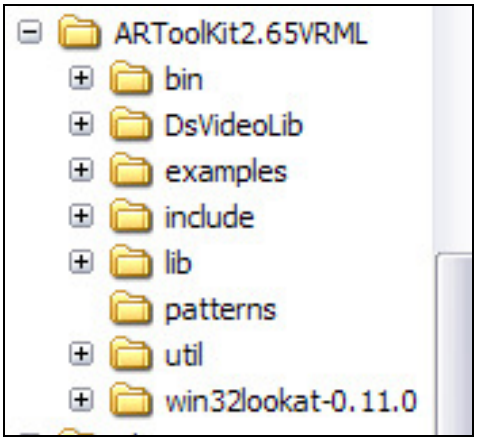

Fig. 17 – Diretório do ARToolkit

Os aplicativos executáveis estão localizados no diretório bin. O exemplo citado anteriormente sobre o simpleVRML, está localizado nesta pasta. Basta ter uma *web cam* conectado ao seu PC, imprimir um dos marcadores padrões e, executar o aplicativo. Estes marcadores estão disponíveis dentro do *path*: "patterns". Há vários marcadores pré-definidos, mas tem-se a opção de criar um marcador pessoal. A figura 18 apresenta um exemplo de um marcador pré-definido pelo ARToolkit.

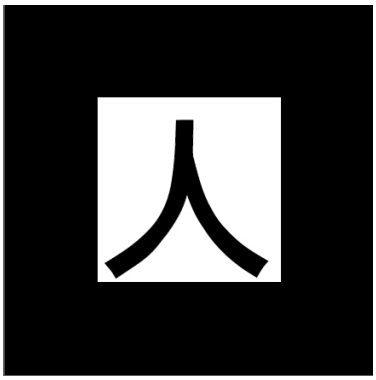

Fig. 18 – Marcador do ARToolkit

Depois de ter criado seu arquivo *WRL* (citado na seção 2.4.1) e o seu marcador, o aplicativo está pronto para ser testado. Apenas deve-se atentar para associar o seu arquivo WRL com o seu marcador. Para isto, abra o arquivo vrml_data localizado no "bin\Data". Faça a associação de seu arquivo com sua marca. A figura 19 ilustra este arquivo.

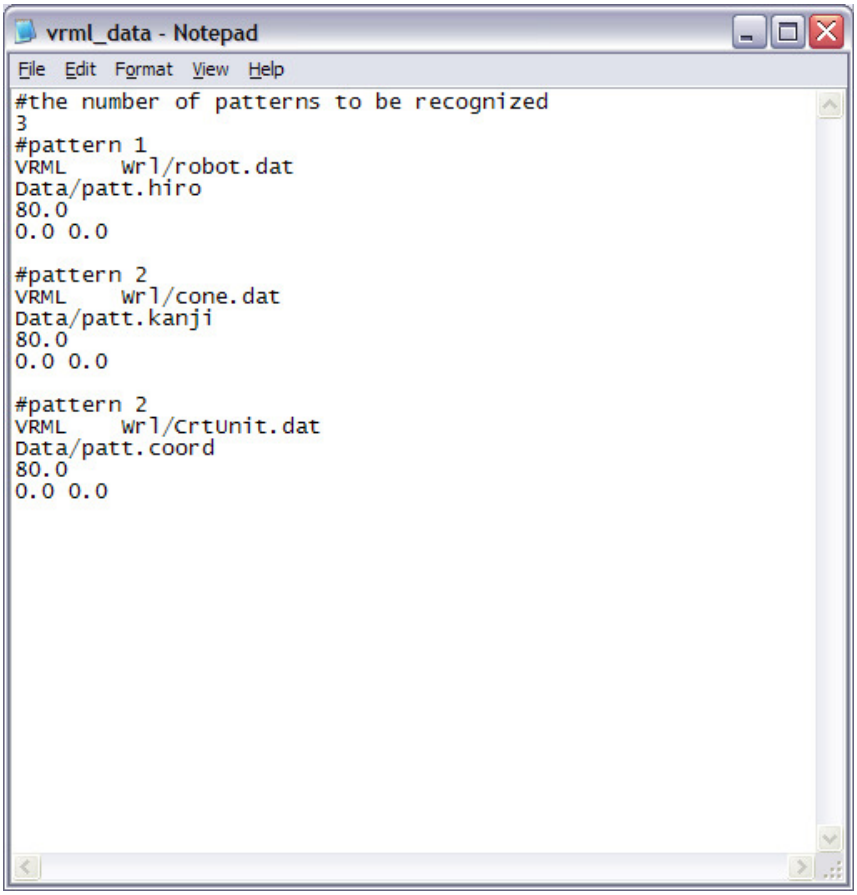

Fig. 19 – Arquivo vrml_data

ANEXO II – CLASSE ARNETEAI

```
/**Classe responsável pela criação do socket, envio do
comando e fechamento da conexão. Esta função recebe como
parâmetros o endereço que será enviado o comando, o número
da porta e a mensagem.
*/
#INCLUDE <MALLOC.H>
#INCLUDE "SOCKET.H"
#INCLUDE <STRING>
#INCLUDE "ARNETEAI.H"
#INCLUDE <STDIO.H>
#INCLUDE <IOSTREAM>
#INCLUDE "OBJECT.H"
#INCLUDE <AR/AR.H>


ARNETEAI::ARNETEAI(OBJECTDATA_T      *OBJ,  INT  SIZE,  OBJECTDATA_T
*NETOBJ){
      OBJECT = OBJ;
      NETOBJECT=NETOBJ;
      OBJECTNUM = SIZE;

      //SET_MARKER(0);// TRABALHA COM "HIRO" NA PRIMEIRA MARCA
      SET_MARKER(1);// TRABALHA COM "HIRO" NA PRIMEIRA MARCA
      RHOSTS = NEW RHOSTS("R_HOSTS.TXT");
/*
      FOR (INT I=0; I<RHOSTS->HOSTS.SIZE(); I++){
            HOST* AUX = RHOSTS->HOSTS[I];
            PRINTF("RHOSTS::RHOSTS  HOSTS[%D]  IP=%S  PORTA=%D\N",  I,
AUX->ADDR, AUX->PORT);
      }
*/
      PRINTF("CRIOU ARNETEAI\N");
}
FLOAT ARNETEAI::TOKENTOFLOAT(){
      FLOAT X;
      GETTOKEN();
      SSCANF(TOKENAUX, "%F", &X);
      RETURN X;
}

CONST CHAR* ARNETEAI::GETNAMESPACE(){
      CHAR* NS = (CHAR*) MALLOC(10);// VERIFICAR O VALOR DE RETORNO
      RETURN NS;
}

VOID  ARNETEAI::SET_TRANSLATION(CONST  CHAR*  NODE,  FLOAT  X,  FLOAT  Y,
FLOAT Z){
```

```
	VrmlScene *sc = sceneGraph->vrmlScene;
	if (sc == NULL){
		printf("minhaCena - Erro: sc == NULL\n");
		return;
	}
	VrmlNamespace* ns = sc->scope();
	VrmlNode* p_no = ns->findNode(node);
	if (p_no==NULL) return;
	VrmlNodeTransform* nd = p_no->toTransform();
	if (nd == NULL) return;

	//VrmlSFVec3f atual = (nd->toTransform())->getTranslation();
	VrmlSFVec3f pos(x, y, z);
	nd->setField("translation", pos);
	nd->render(sceneGraph);
}

void ARNetEAI::set_rotation(const char* node, float x, float y, float
z, float r){
	VrmlScene *sc = sceneGraph->vrmlScene;
	if (sc == NULL)  return;
	VrmlNamespace* ns = sc->scope();
	if (ns == NULL)  return;
	VrmlNode* p_no = ns->findNode(node);
	if (p_no==NULL) return;
	VrmlNodeTransform* nd = p_no->toTransform();
	if (nd == NULL) return;
	VrmlSFRotation pos(x, y, z, r);
	nd->setField("rotation", pos);
	pos = nd->getRotation();
	nd->render(sceneGraph);
}

void ARNetEAI::set_scale(const char* node, float x, float y, float z){
	// Obtem a cena do marcador
	VrmlScene *sc = sceneGraph->vrmlScene;
	if (sc == NULL){
		printf("minhaCena - Erro: sc == NULL\n");
		return;
	}
	VrmlNamespace* ns = sc->scope();
	if (ns == NULL) return;
	VrmlNode* p_no = ns->findNode(node);
	if (p_no==NULL) return;
	VrmlNodeTransform* nd = p_no->toTransform();
	if (nd == NULL) return;
	VrmlSFVec3f pos(x, y, z);
	nd->setField("scale", pos);
	nd->render(sceneGraph);
}
```

```
VOID ARNetEAI::SET_DIFFUSECOLOR(CONST CHAR* NODE, FLOAT R, FLOAT G,
FLOAT B){
      VRMLSCENE *SC = SCENEGRAPH->VRMLSCENE;
      IF (SC == NULL)RETURN;
      VRMLNAMESPACE* NS = SC->SCOPE();
      IF (NS==NULL)RETURN;
      VRMLNODE* P_NO = NS->FINDNODE(NODE);
      IF (P_NO==NULL) RETURN;
      VRMLNODEMATERIAL* ND = P_NO->TOMATERIAL();
      IF (ND==NULL)RETURN;
      VRMLSFCOLOR EM(R, G, B);
      ND->SETFIELD("DIFFUSECOLOR", EM);
      ND->RENDER(SCENEGRAPH);
}

VOID ARNETEAI::SET_TRANSPARENCY(CONST CHAR* NODE, FLOAT T){
      VRMLSCENE *SC = SCENEGRAPH->VRMLSCENE;
      IF (SC == NULL)RETURN;
      VRMLNAMESPACE* NS = SC->SCOPE();
      IF (NS==NULL)RETURN;
      VRMLNODE* P_NO = NS->FINDNODE(NODE);
      IF (P_NO==NULL) RETURN;
      VRMLNODEMATERIAL* ND = P_NO->TOMATERIAL();
      IF (ND==NULL)RETURN;
      VRMLSFFLOAT TRANSP(T);
      ND->SETFIELD("TRANSPARENCY", TRANSP);
      ND->RENDER(SCENEGRAPH);

}

VOID ARNETEAI::SET_EMISSIVECOLOR(CONST CHAR* NODE, FLOAT R, FLOAT G,
FLOAT B){
      VRMLSCENE *SC = SCENEGRAPH->VRMLSCENE;
      IF (SC == NULL)RETURN;
      VRMLNAMESPACE* NS = SC->SCOPE();
      IF (NS==NULL)RETURN;
      VRMLNODE* P_NO = NS->FINDNODE(NODE);
      IF (P_NO==NULL) RETURN;
      VRMLNODEMATERIAL* ND = P_NO->TOMATERIAL();
      IF (ND==NULL)RETURN;
      VRMLSFCOLOR EM(R, G, B);
      ND->SETFIELD("EMISSIVECOLOR", EM);
      ND->RENDER(SCENEGRAPH);
}
VOID ARNETEAI::SET_MARKER(INT M){
    IF (M<0 || M >=OBJECTNUM)
            RETURN; // NAO EXISTE O MARCADOR
      SCENEGRAPH = GETVIEWER(OBJECT[M].VRML_ID);
}
```

```
VrmlNode* ARNetEAI::getNodeByName(const char* node){
      // Obtem a cena do marcador
      VrmlScene* sc = sceneGraph->vrmlScene;
      if (sc == NULL)
            return NULL;
      VrmlNamespace* ns = sc->scope();
      if (ns == NULL)
            return NULL;
      VrmlNode* nd = ns->findNode(node);
      return nd;
}

void ARNetEAI::createFromString(const char* father, const char*
vrmlText){
      // Obtem a cena do marcador
      VrmlScene* sc = sceneGraph->vrmlScene;
      if (sc == NULL) return;
      VrmlNamespace* ns = sc->scope();
      if (ns == NULL) return;
      VrmlNode* nd = ns->findNode(father);
      if (nd == NULL) return;
      VrmlMFNode *nc = VrmlScene::readString (vrmlText, sc->scope());
      if (nc == NULL){
            printf("createFromString->VrmlScene::readString
Error\n");
            printf("%s", vrmlText);
            return;
      }
      VrmlNodeGroup *n = nd->toGroup();
      if (n == NULL) return;
      n->addChildren(*nc);
      n->render(sceneGraph);
      setNameSpace(vrmlText);
}

void ARNetEAI::setNameSpace(const char* cena){
      int i=0;
      int j;
      int cenaLen = strlen(cena);
      char no[256];
      while( i < cenaLen ){
            while (i < cenaLen && cena[i] != 'D') i++;
            if (i>=cenaLen) break;
            i++;
            if (cena[i]!='E') continue;
            i++;
            if (cena[i]!='F') continue;
            i++;
            if (cena[i]!=' ') continue;
```

```
            // Encontrou um DEF
            while (i < cenaLen && cena[i] == ' ') i++; // Elimina
espacos
            j=0;
            while (i < cenaLen && cena[i] != ' '){
                  no[j]=cena[i];
                  i++;
                  j++;
            }
            no[j]='\0';
            idNodes.add(no);
      }
}

void ARNetEAI::removeNameSpace(const char* node){
      char no[256];
      no[0] = '\0';
      idNodes.remove(node);
      strcpy(no, "Mat");
      strcat(no, node);
      idNodes.remove(no);
      idNodes.print();
}
void ARNetEAI::removeChildren(const char* father, const char* child){
//    printf("1 - remove child<%s> father<%s>\n",child, father);
      if (father == NULL || child==NULL)
            return;
      if (strlen(nodeId)==0 || strlen(tokenAux)==0)
            return;
      // Obtem a cena do marcador
      VrmlScene* sc = sceneGraph->vrmlScene;
      if (sc == NULL) return;
//    printf("2 - remove child<%s> father<%s>\n",child, father);

      VrmlNamespace* ns = sc->scope();
      if (ns == NULL) return;
//    printf("3 - achou nameSpace\n");

      VrmlNode* p_no = ns->findNode(father);
      if (p_no==NULL) return;
      VrmlNodeGroup* ng = p_no->toGroup();
      if (ng == NULL) return;
//    printf("4 -  Achou o group do %s\n", father);
      VrmlNode* nd = ns->findNode(child);
      if (nd == NULL) return;
      VrmlNodeChild* nc = nd->toChild();
      if (nc == NULL) return;
//    printf("5 - Achou %s \n",child);
      ng->removeChildren(nc);
 //   printf("6 - removeu \n");
```

```
      NG->RENDER(SCENEGRAPH);
//    PRINTF("7 - RENDERIZOU\N");
      REMOVENAMESPACE(CHILD);
//    PRINTF("8 - REMOVEU DO NAMESPACE -FIM");
}

VOID ARNETEAI::SET_SHININESS(CONST CHAR* NODE, FLOAT VALUE){
      VRMLSCENE *SC = SCENEGRAPH->VRMLSCENE;
      IF (SC == NULL)RETURN;
      VRMLNAMESPACE* NS = SC->SCOPE();
      IF (NS==NULL)RETURN;
      VRMLNODEMATERIAL* ND = (NS->FINDNODE(NODE))->TOMATERIAL();
      IF (ND==NULL)RETURN;
      VRMLSFFLOAT SH(VALUE);
      ND->SETFIELD("SHININESS", SH);
      ND->RENDER(SCENEGRAPH);
}


VOID ARNETEAI::SET_DIFFUSECOLORALL(CONST CHAR* NODE, FLOAT R, FLOAT G,
FLOAT B){
      VRMLSCENE *SC = SCENEGRAPH->VRMLSCENE;
      IF (SC == NULL)  RETURN;
      VRMLNAMESPACE* NS = SC->SCOPE();
      IDNODES.RESETNEXT();
    CHAR* NO = IDNODES.GETNEXT();
      VRMLSFCOLOR EM(R, G, B);
      WHILE (NO != NULL){
        IF (STRNCMP(NO, NODE, STRLEN(NODE))==0){
                  VRMLNODEMATERIAL*    ND    =    (NS->FINDNODE(NO))-
>TOMATERIAL();
                  IF (ND==NULL)RETURN;
                  ND->SETFIELD("DIFFUSECOLOR", EM);
                  ND->RENDER(SCENEGRAPH);
            }
           NO = IDNODES.GETNEXT();
      }
}

VOID ARNETEAI::SET_EMISSIVECOLORALL(CONST CHAR* NODE, FLOAT R, FLOAT
G, FLOAT B){
      VRMLSCENE *SC = SCENEGRAPH->VRMLSCENE;
      IF (SC == NULL)  RETURN;
      VRMLNAMESPACE* NS = SC->SCOPE();
      IDNODES.RESETNEXT();
    CHAR* NO = IDNODES.GETNEXT();
      VRMLSFCOLOR EM(R, G, B);
      WHILE (NO != NULL){
        IF (STRNCMP(NO, NODE, STRLEN(NODE))==0){
```

```
                    VrmlNodeMaterial*    nd    =    (ns->findNode(no))-
>toMaterial();
                    if (nd==NULL)return;
                    nd->setField("emissiveColor", em);
                    nd->render(sceneGraph);
              }
              no = idNodes.getNext();
        }
}

void ARNetEAI::set_transparencyAll(const char* node, float value){
      VrmlScene *sc = sceneGraph->vrmlScene;
      if (sc == NULL)  return;
      VrmlNamespace* ns = sc->scope();
      idNodes.resetNext();
    char* no = idNodes.getNext();
    VrmlSFFloat transp(value);
      while (no != NULL){
        if (strncmp(no, node, strlen(node))==0){
                    VrmlNodeMaterial*    nd    =    (ns->findNode(no))-
>toMaterial();
                    if (nd==NULL)return;
                    nd->setField("transparency", transp);
                    nd->render(sceneGraph);
              }
              no = idNodes.getNext();
        }

}

int ARNetEAI::getNode (){
      if (buffer == NULL)
            return -1;
      if (strlen(buffer) == 0)
            return -2;
      while(buffSize>buffPointer  &&  buffer[buffPointer]  !=  '\0'  &&
buffer[buffPointer] == ' ')
            buffPointer++;
      int i=0;
      while(buffSize>buffPointer  &&  buffer[buffPointer]  !=  '\0'  &&
buffer[buffPointer] != ' '){
            nodeId[i] = buffer[buffPointer];
            buffPointer++;
            i++;
    }
      nodeId[i] = '\0';
      return 0;
}

int ARNetEAI::getCmd (){
```

```
      IF (BUFFER == NULL)
            RETURN -1;
      IF (STRLEN(BUFFER) == 0)
            RETURN -2;
      WHILE(BUFFSIZE>BUFFPOINTER && BUFFER[BUFFPOINTER] != '\0' &&
BUFFER[BUFFPOINTER] == ' '){
      //    PRINTF("BUFFER1         %C         BUFFPOINTER         %I
\N",BUFFER[BUFFPOINTER],BUFFPOINTER);
            BUFFPOINTER++;

      }
      INT I=0;
      WHILE(BUFFSIZE>BUFFPOINTER && BUFFER[BUFFPOINTER] != '\0' &&
BUFFER[BUFFPOINTER] != ' '){
            //PRINTF("BUFFER2         %C         BUFFPOINTER         %I
\N",BUFFER[BUFFPOINTER],BUFFPOINTER);
            COMMAND[I] = BUFFER[BUFFPOINTER];
            BUFFPOINTER++;
            I++;
    }
      COMMAND[I] = '\0';
      RETURN 0;
}

INT ARNETEAI::GETTOKEN (){
      IF (BUFFER == NULL)
            RETURN -1;
      IF (STRLEN(BUFFER) == 0)
            RETURN -2;
      WHILE(BUFFSIZE>BUFFPOINTER && BUFFER[BUFFPOINTER] != '\0' &&
BUFFER[BUFFPOINTER] == ' ')
            BUFFPOINTER++;
      INT I=0;
      WHILE(BUFFSIZE>BUFFPOINTER && BUFFER[BUFFPOINTER] != '\0' &&
BUFFER[BUFFPOINTER] != ' ' ){
            TOKENAUX[I] = BUFFER[BUFFPOINTER];
            BUFFPOINTER++;
            I++;
    }
      TOKENAUX[I] = '\0';
      RETURN 0;
}

VOID ARNETEAI::PRINTNAMESPACE(VOID){
      IDNODES.PRINT();
}

VOID ARNETEAI::SET_INTERTRANSLATION(CONST CHAR* NODE, FLOAT X, FLOAT
Y, FLOAT Z){
```

```
      VrmlScene *sc = sceneGraph->vrmlScene;
      if (sc == NULL)  return;
      VrmlNamespace* ns = sc->scope();
      VrmlNode* p_no = ns->findNode(node);
      if (p_no==NULL) return;
      VrmlNodeTransform* nd = p_no->toTransform();
      if (nd == NULL) return;
      VrmlSFVec3f atual = (nd->toTransform())->getTranslation();
    float dx = (x-atual.x())/1000;
    while (abs(dx)< abs(x) -abs(atual.x())){
            atual.set(atual.x()+dx, y, z);
            nd->setField("translation", atual);
            nd->render(sceneGraph);
      }
}

int ARNetEAI::exec(const char* cmd, int sz){
      // Manter a ordem dos comandos
      buffer = cmd;
      buffSize = sz;
      buffPointer = 0;
      float x, y, z; // Posicao no espaco
      float aux; // Valor de campo - auxiliar
      // Quando ha troca de marca, deve redefinir sceneGraph.
      // Quando a mnarca nao eh reconhecida, os comandos
      // nao devem ser interpretados.
      //printf("Exec <%s>\n", cmd);

      //printf("Passou Exec size %i \n", sz);
      getCmd();
      if (strcmp(command, "set_marker")==0){
            int m;
          sscanf(&buffer[buffPointer], "%i", &m);
            set_marker(m-1);
            return 0;
      }

      if (sceneGraph == NULL){
            printf("ARNetEAI::exec Erro: cena nao definida sceneGraph
== NULL\n");
            return 0;
      }
      //printf("1 - Comando<%s>\n", cmd);
      if (strcmp(command, "printNameSpace")==0){
            printNameSpace();
            return 0;
      }

      getNode();
      if (strcmp(command, "set_object")==0){
```

```
            PRINTF("BUFFPOINTER %I \N",BUFFPOINTER);
            THIS->SETOBJECT(&OBJECT[ATOI(NODEID)],
&BUFFER[BUFFPOINTER]);
            RETURN 0;
      }

      IF (STRCMP(COMMAND, "CREATEFROMSTRING")==0){
            CREATEFROMSTRING(NODEID, &BUFFER[BUFFPOINTER]);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_TRANSLATION")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F%F%F", &X, &Y, &Z);
            SET_TRANSLATION(NODEID, X, Y, Z);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_ROTATION")==0){
            SSCANF(&BUFFER[BUFFPOINTER],  "%F%F%F%F",  &X,  &Y,  &Z,
&AUX);
            SET_ROTATION(NODEID, X, Y, Z, AUX);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_SCALE")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F%F%F", &X, &Y, &Z);
            SET_SCALE(NODEID, X, Y, Z);
            RETURN 0;
      }

      IF (STRCMP(COMMAND, "SET_DIFFUSECOLOR")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F%F%F", &X, &Y, &Z);
            SET_DIFFUSECOLOR(NODEID, X, Y, Z);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_TRANSPARENCY")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F", &X);
            SET_TRANSPARENCY(NODEID, X);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_EMISSIVECOLOR")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F%F%F", &X, &Y, &Z);
            SET_EMISSIVECOLOR(NODEID, X, Y, Z);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "REMOVE")==0){
            GETTOKEN();
//          PRINTF("PAI<%S> FILHO<%S>", NODEID, TOKENAUX );
            REMOVECHILDREN(NODEID, TOKENAUX);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_SHININESS")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F", &AUX);
```

```
            SET_TRANSPARENCY(NODEID, AUX);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_DIFFUSECOLORALL")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F%F%F", &X, &Y, &Z);
            SET_DIFFUSECOLORALL(NODEID, X, Y, Z);
            RETURN 0;
      }

      IF (STRCMP(COMMAND, "SET_EMISSIVECOLORALL")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F%F%F", &X, &Y, &Z);
            SET_EMISSIVECOLORALL(NODEID, X, Y, Z);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_TRANSPARENCYALL")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F", &AUX);
            SET_TRANSPARENCYALL(NODEID, AUX);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SAVENAMESPACE")==0){
            IDNODES.SAVE(NODEID);
            RETURN 0;
      }
      IF (STRCMP(COMMAND, "SET_INTERTRANSLATION")==0){
            SSCANF(&BUFFER[BUFFPOINTER], "%F%F%F", &X, &Y, &Z);
            SET_INTERTRANSLATION(NODEID, X, Y, Z);
            RETURN 0;
      }
      RETURN -1; // COMANDO NAO EXISTE.
}

//*************************************************************
**************************
VOID ARNETEAI::SEND(CHAR *ADDR, INT PORT, CHAR *MSG)
{
  STATIC SOCKETCLIENT* S;
  STATIC INT CMDSZ = 2500;
  STATIC CHAR* COMANDO=0;
  IF (COMANDO == 0)
        COMANDO = (CHAR*) MALLOC (CMDSZ);
  IF (CMDSZ < STRLEN(MSG)){
        FREE (COMANDO);
        CMDSZ = STRLEN(MSG) + 1;
        COMANDO = (CHAR*) MALLOC (CMDSZ);
  }
  COMANDO[0] = '\0';
  SPRINTF(COMANDO,"%I ", STRLEN(MSG));
//  FOR (INT I=STRLEN(COMANDO);I<10; I++)
//      STRCAT(COMANDO, " ");
```

```
  STRCAT(COMANDO, MSG);
  TRY {
      S = NEW SOCKETCLIENT (ADDR, PORT);
      S->SENDBYTES(COMANDO);
    PRINTF(" ENVIOU: <<%S>>\N", COMANDO);

      S->CLOSE();
  }
  CATCH (CONST CHAR* S) {
        STD::CERR << S << ENDL;
  }
  CATCH (STD::STRING S) {
        STD::CERR << S << ENDL;
  }
  CATCH (...) {
        STD::CERR << "UNHANDLED EXCEPTION\N";
  }
}

VOID ARNETEAI::SENDOBJS()
{
      CHAR* OBJ=0;
      STATIC CHAR CMD[2500];
      INT I=0;
      STATIC INT CONT=0;
      IF ( OBJECT == 0 )
            RETURN;
      //IF (CONT < 20){
      //    CONT++;
      //    RETURN;
      //}
    FOR( I = 1; I < OBJECTNUM; I++ ) {
            IF(OBJECT[I].VISIBLE==1){
                  OBJ = GETOBJECT(&OBJECT[I]);
                  SPRINTF(CMD, "SET_OBJECT %I %S", I, OBJ);
            // ENVIA PARA TODOS OS HOSTS
            FOR (INT NH=0; RHOSTS->HOSTS.SIZE() > NH; NH++){
                        HOST* AUX = RHOSTS->HOSTS[NH];
            //          PRINTF("SENDOBJECTS  %S  %D  LENGTH=%D\N",AUX-
>ADDR, AUX->PORT, STRLEN(CMD));
                        SEND(AUX->ADDR, AUX->PORT, CMD);
                  }
            }
      }
      //CONT = 0;
      //PRINTF ("\NNUMERO DE VEZES QUE PASSOU NO SEND: %I", CONT);
}
CHAR* ARNETEAI::GETOBJECT(OBJECTDATA_T* OBJ){
      INT I, J;
      STATIC CHAR BUFF[2000];
```

```
      BUFF[0] = '\0';
      STATIC DOUBLE MATINV[3][4];
      STATIC DOUBLE MATAUX[3][4];

      IF(OBJECT[0].VISIBLE==1){
            ARUTILMATINV(OBJECT[0].TRANS, MATINV);
            ARUTILMATMUL(MATINV, OBJ->TRANS,MATAUX);
      }

  SPRINTF(&BUFF[STRLEN(BUFF)], "%I %I %S %I %I ", OBJ->ID, OBJ-
>VISIBLE, OBJ->NAME,
        OBJ->VRML_ID, OBJ->PERSISTENTE);

  FOR (I=0; I<3; I++)
            FOR (J=0; J<4; J++)
                  SPRINTF(&BUFF[STRLEN(BUFF)],"%LF ", MATAUX[I][J]);


      RETURN BUFF;
}

VOID ARNETEAI::SETOBJECT(OBJECTDATA_T* OBJ, CONST CHAR* BUFF){
      INT I, J;
      INT ID;
      OBJECTDATA_T AUX;

      SSCANF(BUFF, "%I ",&(AUX.ID));

      IF(NETOBJECT == NULL)
            RETURN;

      ID=AUX.ID;

      PRINTF("BUFF %S \N",&BUFF[STRLEN(BUFF)]);

      SSCANF(BUFF,"%I %I %S %I %I %LF %LF %LF %LF %LF %LF %LF %LF %LF
%LF %LF %LF",&(NETOBJECT[ID].ID),

      &(NETOBJECT[ID].VISIBLE),NETOBJECT[ID].NAME,&(NETOBJECT[ID].VRM
L_ID), &(NETOBJECT[ID].PERSISTENTE), &NETOBJECT[ID].TRANS[0][0],

      &NETOBJECT[ID].TRANS[0][1],&NETOBJECT[ID].TRANS[0][2],&NETOBJ
ECT[ID].TRANS[0][3],&NETOBJECT[ID].TRANS[1][0],

      &NETOBJECT[ID].TRANS[1][1],&NETOBJECT[ID].TRANS[1][2],&NETOBJ
ECT[ID].TRANS[1][3],&NETOBJECT[ID].TRANS[2][0],

      &NETOBJECT[ID].TRANS[2][1],&NETOBJECT[ID].TRANS[2][2],&NETOBJ
ECT[ID].TRANS[2][3]);
```

```
}
#INCLUDE <MATH.H>
#INCLUDE <STRING.H>
#INCLUDE "ARSERVER.H"

#IFDEF _WIN32
#INCLUDE <WINDOWS.H>
#INCLUDE <STDIO.H>
#INCLUDE <MALLOC.H>
#INCLUDE <STDLIB.H>

#ENDIF
//#INCLUDE <STDIO.H>
//#INCLUDE <STDLIB.H>


#INCLUDE <GL/GL.H>
#INCLUDE <GL/GLUT.H>
#INCLUDE <AR/GSUB.H>
#INCLUDE <AR/PARAM.H>
#INCLUDE <AR/VIDEO.H>
#INCLUDE "ARNETEAI.H"
#INCLUDE "FILA.CPP"
// EXAMPLES:
//
// CHAR *VCONF = NULL;
// CHAR *VCONF = "SHOWDLG";
//       CHAR       *VCONF       =       "DEVICENAME=MICROSOFT       DV
CAMERA,VIDEOWIDTH=720,VIDEOHEIGHT=576";
//         CHAR                                          *VCONF          =
"IEEE1394ID=437D3B0201460008,VIDEOWIDTH=180,VIDEOHEIGHT=144";
// CHAR    *VCONF = "SHOWDLG";
//   CHAR             *VCONF   =   "SHOWDLG,DEVICENAME=MICROSOFT   DV
CAMERA,DEINTERLACESTATE=ON,DEINTERLACEMETHOD=BLEND,"
//
"PIXELFORMAT=PIXELFORMAT_RGB32,VIDEOWIDTH=320,VIDEOHEIGHT=240";

/*  PARAMETER FORMAT IS EITHER NULL OR A LIST OF TOKENS, SEPARATED BY
COMMAS ","
 *
 *   BINARY TOKENS:
 *   --------------
 *     FLIPH       : FLIP  IMAGE  HORIZONTALLY  (WARNING:  NON-OPTIMAL
PERFORMANCE)
 *     FLIPV       :  FLIP  IMAGE  VERTICALLY  (WARNING:  NON-OPTIMAL
PERFORMANCE)
 *   SHOWDLG  : DISPLAYS EITHER WDM CAPTURE FILTER'S PROPERTY PAGE OR
 *               MSDV DECODER FORMAT DIALOG (DEPENDING ON SOURCE MEDIA
TYPE).
 *
```

```
 *   PARAMETRIZED TOKENS:
 *   --------------------
 *   videoWidth=?  : preferred video width, EXAMPLE: "videoWidth=720"
 *         videoHeight=?   :   preferred   video   height,   EXAMPLE:
"videoHeight=576"
 *      deviceName=?    :  preferred  WDM  device  (WARNING:  locale-
dependent),
 *                      i.e. try to match substring <?> with "friendly
names" of
 *                  enumerated DirectShow WDM wrappers (ksproxy.ax).
 *                     EXAMPLE: "deviceName=Microsoft DV Camera" for
IEEE1394 DV devices
 *                                  "deviceName=QuickCam" for Logitech
QuickCam
 *    ieee1394id=?   : Unique 64-bit device identifier, as defined by
IEEE 1394.
 *                              Hexadecimal   value   expected,   i.e.
"ieee1394id=437d3b0201460008"
 *                         Use /bin/IEEE394_id.exe to determine your
camera's ID.
 *
 *   EXAMPLES:
 *
 *   arVideoOpen(NULL);
 *   arVideoOpen("showDlg");
 *   arVideoOpen("flipH,flipV,showDlg");
 *   arVideoOpen("videoWidth=640,flipH,videoHeight=480,showDlg");
 *                          arVideoOpen("deviceName=Microsoft       DV
Camera,videoWidth=720,videoHeight=480");
 *
arVideoOpen("deviceName=Logitech,videoWidth=320,videoHeight=240,flipV
");
 *                          arVideoOpen("deviceName=Microsoft       DV
Camera,ieee1394id=437d3b0201460008");
 *   ...
 */

/**************************************************************
******************/

/* external VRML files */
extern "C" {
  #include "object.h"
  #include <AR/vrml97.h>
}
#include <AR/vrml97int.h>
#include <time.h>


/* Object Data */
```

```
CHAR           *MODEL_NAME = "DATA/VRML_DATA";
OBJECTDATA_T    *OBJECT;
OBJECTDATA_T    *NETOBJECT;
INT            OBJECTNUM;

#IFDEF _WIN32
CHAR       *VCONF  =  "FLIPH,VIDEOWIDTH=160,VIDEOHEIGHT=140";  //  SEE
VIDEO.H FOR A LIST OF SUPPORTED PARAMETERS
#ELSE
CHAR    *VCONF = "";
#ENDIF

INT            XSIZE, YSIZE;
INT            THRESH = 100;
INT            FRCOUNT = 0;



CHAR          *CPARAM_NAME    = "DATA/CAMERA_PARA.DAT";
ARPARAM         CPARAM;

STATIC VOID        INIT(VOID);
STATIC VOID        CLEANUP(VOID);
STATIC VOID        KEYEVENT( UNSIGNED CHAR KEY, INT X, INT Y);
STATIC VOID        MAINLOOP(VOID);
INT                      DRAW( OBJECTDATA_T *OBJECT, INT OBJECTNUM );
STATIC INT         DRAW_OBJECT(     OBJECTDATA_T     *OBJECT,     DOUBLE
GL_PARA[16] );
STATIC VOID        INIT_LIGHTS(VOID);
VOID               GRAVADADOS(DOUBLE F,INT USER);

DOUBLE             MATINV[3][4];
DOUBLE          MATDIFF[3][4];

// ============= FUNCOES COMPLEMENTARES ================
UNSIGNED __STDCALL SERVICO(VOID* A);  // FUNCAO DE CALLBACK DO SERVER
INT PORTASERVICO; // PORTA EM QUE O SERVIDOR ESTA INSTALADO

CONST INT NUMEROTHREADS = 10;
INT CONTATHREADS=0;
INT CONTAACCEPT=0;
QUEUE FILABUFF;

CONST INT MINBUFFSZ = 1024;
ARNETEAI* NG; // INTERPRETADOR DE COMANDOS PARA ATUAR NA CENA
INT PERSISTENTE;

STATIC INT         CONT1=0,CONT2=0;
```

```
int verificaDistancia(ObjectData_T *object,int dist, int idPatt, int
flag );

int main(int argc, char* argv[])
{
      portaServico = 1202;
      persistente = 0;
      if (argc==2)
            portaServico = atoi(argv[1]);
      //initialize applications
    init();
      //start video capture
      arVideoCapStart();
      //start the main event loop
    argMainLoop( NULL, keyEvent, mainLoop );
      return 0;
}

unsigned __stdcall inicia(void* in){
  printf("Servico em funcionamento\n");
  while (1) {
    Socket* s=((SocketServer*)in)->Accept();
    unsigned ret;

      contaThreads++;
      contaAccept++;

    printf("-Iniciou Thread: %i   ContaAccept: %i \n",contaThreads,
contaAccept);
      _beginthreadex(0,0,servico,(void*) s,0,&ret);


  }
}
unsigned __stdcall servico(void* a) {
  static char* buff;
  int szBf;

  szBf = 1024;


  buff = (char*) malloc (minBuffSz);

  Socket* s = (Socket*) a;


  int status = s->ReceiveBytes(&buff, &szBf);

  //filaBuff.Insere(buff);
```

```
  //FREE(BUFF);
  IF (NG != 0){
        NG->EXEC(BUFF, STRLEN(BUFF));
  }
  //SLEEP(1000);
  S->CLOSE();
  PRINTF("   >>FINALIZOU THREAD: %I\N",CONTATHREADS);
  CONTATHREADS--;

  RETURN 0;
}

STATIC VOID   KEYEVENT( UNSIGNED CHAR KEY, INT X, INT Y)
{

    /* QUIT IF THE ESC KEY IS PRESSED */
    IF( KEY == 0X1B ) {
        PRINTF("***                %F                (FRAME/SEC)\N",
(DOUBLE)FRCOUNT/ARUTILTIMER());
            PRINTF("CONT 1: %I\NCONT 2: %I\N",CONT1,CONT2);
       //CLEANUP();
       // EXIT(0);
    }
      IF( KEY == (CHAR)'X' ){//LIMPA TELA
       CONTATHREADS=0;
         CONTAACCEPT=0;
       SYSTEM("CLS");
         PRINTF("LIMPO.\N");
         }
      IF( KEY == (CHAR)'A' )
       PERSISTENTE = 1;
      IF( KEY == (CHAR)'A' )
       PERSISTENTE = 0;
      IF  ( KEY == (CHAR) 'S'){
            ARVRML97VIEWER* CENATOTAL = GETVIEWER(OBJECT[0].VRML_ID);
            VRMLSCENE* SC = CENATOTAL->VRMLSCENE;
            SC->SAVE("..\\..\\ARQCENA.WRL");
      }

}

/* MAIN LOOP */
STATIC VOID MAINLOOP(VOID)
{
    ARUINT8         *DATAPTR;
    ARMARKERINFO    *MARKER_INFO;
    INT             MARKER_NUM;
    INT             I, J, K;
      DOUBLE                    FRAMES;
      INT                       FLAG;
```

```
    //CHAR* A;

      CONT1++;
    /* GRAB A VIDEO FRAME */
      IF( (DATAPTR = (ARUINT8 *)ARVIDEOGETIMAGE()) == NULL ) {
        ARUTILSLEEP(2);
        RETURN;
    }
      CONT2++;
/* INVERSÃO DA IMAGEM CAPTURADA PELA CAMERA
      ARUINT8         X;
      FOR (J=0; J<(YSIZE/2); J++)
            FOR (I=0; I<4*XSIZE; I++) {
                  X = DATAPTR[I + J*4*XSIZE];
                  DATAPTR[I  +  J*4*XSIZE]  =  DATAPTR[I+4*(YSIZE-J-
1)*XSIZE];
                  DATAPTR[I+4*(YSIZE-J-1)*XSIZE] = X;
            }
*/
    IF( FRCOUNT == 0 ) ARUTILTIMERRESET();
    FRCOUNT++;


    ARGDRAWMODE2D();
    ARGDISPIMAGE( DATAPTR, 0,0 );

      FRAMES = (DOUBLE)FRCOUNT/ARUTILTIMER();

//  MODIFICACOES LUCAS
//    PRINTF("%F  (F/S)   <--> %F TIME \N", FRAMES, ARUTILTIMER());



      //GRAVADADOS(FRAMES, NUSER);


      /* CAPTURE THE NEXT VIDEO FRAME */
      ARVIDEOCAPNEXT();

    /* DETECT THE MARKERS IN THE VIDEO FRAME */
            IF(    ARDETECTMARKER(DATAPTR,    THRESH,    &MARKER_INFO,
&MARKER_NUM) < 0 ) {
                  CLEANUP();
                  EXIT(0);
            }

      /* CHECK FOR OBJECT VISIBILITY */
    FOR( I = 0; I < OBJECTNUM; I++ ) {
        K = -1;
```

```
        FOR( J = 0; J < MARKER_NUM; J++ ) {
            IF( OBJECT[I].ID == MARKER_INFO[J].ID ) {
                IF( K == -1 ) K = J;
                ELSE {
                    IF( MARKER_INFO[K].CF < MARKER_INFO[J].CF ) K = J;
                }
            }
        }
        IF( K == -1 ) {
            OBJECT[I].VISIBLE = 0;
            CONTINUE;
        }

        IF( OBJECT[I].VISIBLE == 0 ) {
            ARGETTRANSMAT(&MARKER_INFO[K],
                          OBJECT[I].MARKER_CENTER,
OBJECT[I].MARKER_WIDTH,
                          OBJECT[I].TRANS);
                  FLAG=1;

                  IF(I==0)
                        ARUTILMATINV(OBJECT[0].TRANS,MATINV);

        }
        ELSE {
            ARGETTRANSMATCONT(&MARKER_INFO[K], OBJECT[I].TRANS,
                          OBJECT[I].MARKER_CENTER,
OBJECT[I].MARKER_WIDTH,
                          OBJECT[I].TRANS);
                  FLAG=0;


        }
        OBJECT[I].VISIBLE = 1;
             OBJECT[I].PERSISTENTE = PERSISTENTE;
    }


//    IF(VERIFICADISTANCIA(OBJECT,500,1,FLAG))
//          PRINTF("O MARCADOR SUPEROU O LIMITE ESTABELECIDO!!\N");
                        //IF (NG!=0)
                        //    NG->SENDOBJS();


      /* DRAW THE VIRTUAL OBJECTS ATTACHED TO THE CARDS */
    GLCLEARDEPTH( 1.0 );
    GLCLEAR(GL_DEPTH_BUFFER_BIT);
    GLDISABLE(GL_BLEND);
    DRAW( OBJECT, OBJECTNUM );
```

```
      //
==========================================================
      // ========= ENVIA OS DADOS DOS MARCADORES ===============
//    IF((VERIFICADISTANCIA(OBJECT,500,1,FLAG)) && (NG!=0)) {
//                      PRINTF("O    MARCADOR    SUPEROU    O    LIMITE
ESTABELECIDO!!\N");

      IF (NG!=0){
            IF(VERIFICADISTANCIA(OBJECT,500,1,FLAG)){
                  NG->SENDOBJS();  //COLOCADO  DENTRO  DA  FUNCAO  DE
VERIFICADISTANCIA - ACIMA
                  PRINTF ("\NOBJETO ATUALIZADO");
            }
      }

      /*
      A = (CHAR*) MALLOC (MINBUFFSZ);
      A = FILABUFF.REMOVE();
      IF (A!=NULL){
         //PRINTF("FILABUFF: <%S>\N",A);
         IF (NG != 0){
            NG->EXEC(A, STRLEN(A));
         }
      }
      FREE(A);
*/
      /* UPDATE THE VRML ANIMATION */
      ARVRML97TIMERUPDATE();
      /* GET READY TO GRAB A NEW VIDEO FRAME */
      ARGSWAPBUFFERS();

}

STATIC VOID INIT( VOID )
{
      INT I;
      ARPARAM  WPARAM;

      DOUBLE AUXMAT[3][4];
      /* OPEN THE VIDEO PATH */
    IF( ARVIDEOOPEN(VCONF) < 0 ) EXIT(0);
    /* FIND THE SIZE OF THE WINDOW */
    IF( ARVIDEOINQSIZE(&XSIZE, &YSIZE) < 0 ) EXIT(0);
    PRINTF("IMAGE SIZE (X,Y) = (%D, %D)\N", XSIZE, YSIZE);

    /* SET THE INITIAL CAMERA PARAMETERS */
    IF( ARPARAMLOAD(CPARAM_NAME, 1, &WPARAM) < 0 ) {
        PRINTF("CAMERA PARAMETER LOAD ERROR !!\N");
        EXIT(0);
    }
```

```
    arParamChangeSize( &wparam, xsize, ysize, &cparam );
    arInitCparam( &cparam );
    printf("*** Camera Parameter ***\n");
    arParamDisp( &cparam );

    /* open the graphics window */
    argInit( &cparam, 1.0, 0, 0, 0, 0 ); //cparam, zoom, fullflag,
xwin, ywin, hmd

    arFittingMode   = AR_FITTING_TO_IDEAL;
//    arFittingMode   = AR_FITTING_TO_INPUT;

    arImageProcMode = AR_IMAGE_PROC_IN_FULL;
//    arImageProcMode = AR_IMAGE_PROC_IN_HALF;

      argDrawMode     = AR_DRAW_BY_TEXTURE_MAPPING;
//    argDrawMode     = AR_DRAW_BY_GL_DRAW_PIXELS;

    argTexmapMode   = AR_DRAW_TEXTURE_FULL_IMAGE;
//    argTexmapMode   = AR_DRAW_TEXTURE_HALF_IMAGE;


      /* load in the object data - trained markers and associated bitmap
files */
      /* char *model_name = "Data/vrml_data";*/
    printf("Arquivo<%s> \n", model_name);

    if(  (object=read_VRMLdata(model_name,  &objectnum))  ==  NULL  )
exit(0);
    printf("Objectfile num = %d\n", objectnum);

    /* test render all the VRML objects */
    printf("about to render VRML objects \n");
    glEnable(GL_TEXTURE_2D);

    for ( i = 0; i < objectnum; i++ ) {
      printf("rendering %d \n",i);
      //renderVRMLObject(i);
        arVrml97Draw( object[i].vrml_id );
    }
    glDisable(GL_TEXTURE_2D);

      /* initialize lights */
    init_lights();

    netObject   =   (ObjectData_T   *)malloc(   sizeof(ObjectData_T)   *
objectnum );
      for ( i = 0; i < objectnum; i++ ) {
            netObject[i].id=object[i].id;
            netObject[i].visible=0;
```

```
            netObject[i].vrml_id=object[i].vrml_id;



    }

      // Cria o interpretador de comandos
    if (::ng==NULL) ng = new ARNetEAI(object, objectnum, netObject);
      /* ================================================ */
    ARServer s(portaServico,numeroThreads); /* Cria o servico de rede
*/
     /* Inicia o servico de rede */
    s.start();
    printf("Servido iniciado na porta %i\n", portaServico);
}

/* cleanup function called when program exits */
static void cleanup(void)
{
      arVideoCapStop();
    arVideoClose();
    argCleanup();
}

/* draw the the AR objects */
int draw( ObjectData_T *object, int objectnum )
{
    int     i;
    double  gl_para[16];
      double  mat_draw[3][4];
    double  matResult[3][4];


    /* calculate the viewing parameters - gl_para */


    for( i = 0; i < objectnum; i++ ) {
        if(  object[i].visible  ==  0  &&  object[i].persistente==0)
continue;

            if(i>0 && object[i].persistente==1){
                  arUtilMatMul(matInv,object[i].trans,matDiff);
                  arUtilMatMul(object[0].trans,matDiff,matResult);
                  argConvGlpara(matResult, gl_para);
                  draw_object( &object[i], gl_para );

            }
            else{
                  argConvGlpara(object[i].trans, gl_para);
                  draw_object( &object[i], gl_para );
```

```
            }
            IF(NETOBJECT[I].VISIBLE          ==          0          &&
NETOBJECT[I].PERSISTENTE==0) CONTINUE;

            ARGCONVGLPARA(NETOBJECT[I].TRANS, GL_PARA);
        DRAW_OBJECT( &NETOBJECT[I], GL_PARA );
    }

      FOR( I = 0; I < OBJECTNUM; I++ ) {
      IF(NETOBJECT[I].VISIBLE == 0) CONTINUE;

      ARUTILMATMUL(OBJECT[0].TRANS,NETOBJECT[I].TRANS,MAT_DRAW);

            ARGCONVGLPARA(MAT_DRAW, GL_PARA);
        DRAW_OBJECT( &NETOBJECT[I], GL_PARA );
    }
    RETURN(0);
}

/* DRAW THE USER OBJECT */
STATIC INT  DRAW_OBJECT( OBJECTDATA_T *OBJECT, DOUBLE GL_PARA[16] )
{
    ARGDRAWMODE3D();
    ARGDRAW3DCAMERA( 0, 0 );
    GLMATRIXMODE(GL_MODELVIEW);
    GLLOADMATRIXD( GL_PARA );
    IF( OBJECT->VRML_ID >= 0 ) {
        ARVRML97DRAW( OBJECT->VRML_ID );
       }
    ELSE {
      PRINTF("UNKNOWN OBJECT TYPE!!\N");
    }

    ARGDRAWMODE2D();

    RETURN 0;
}

/* INITIALIZE THE LIGHTS IN THE SCENE */
STATIC VOID INIT_LIGHTS()
{
    GLFLOAT LIGHT_POSITION[] = { 0, 0, -40.0,  0.0};
    GLFLOAT LIGHT_POSITION1[] = {1.0,1.0,-500.0,0.0};
    GLFLOAT LIGHT_POSITION2[] = {1.0,-500.0,0.0,0.0};
    GLFLOAT LIGHT_POSITION3[] = {-500.0,1.0,0.0,0.0};

    //GLFLOAT AMBI[] = {1, 1, 1, 0.1};
      GLFLOAT AMBI[] = {1, 1, 1, 1};
    GLFLOAT LIGHTZEROCOLOR[]    = {1,1,1, 1};
```

```
    /* LIGHT 0 */
    GLLIGHTFV(GL_LIGHT0, GL_POSITION, LIGHT_POSITION);
    GLLIGHTFV(GL_LIGHT0, GL_AMBIENT, AMBI);
    GLLIGHTFV(GL_LIGHT0, GL_DIFFUSE, LIGHTZEROCOLOR);  /* LIGHT 0
*/

    /* LIGHT 1 */
    GLLIGHTFV(GL_LIGHT1, GL_POSITION, LIGHT_POSITION1);
      GLLIGHTFV(GL_LIGHT1, GL_AMBIENT, AMBI);
      GLLIGHTFV(GL_LIGHT1, GL_DIFFUSE, LIGHTZEROCOLOR);

    /* LIGHT 2 */

    GLLIGHTFV(GL_LIGHT2, GL_POSITION, LIGHT_POSITION2);
      GLLIGHTFV(GL_LIGHT2, GL_AMBIENT, AMBI);
      GLLIGHTFV(GL_LIGHT2, GL_DIFFUSE, LIGHTZEROCOLOR);

    /* LIGHT 3 */
    GLLIGHTFV(GL_LIGHT3, GL_POSITION, LIGHT_POSITION3);
      GLLIGHTFV(GL_LIGHT3, GL_AMBIENT, AMBI);
      GLLIGHTFV(GL_LIGHT3, GL_DIFFUSE, LIGHTZEROCOLOR);


    /* LIGHT 4 */
    GLLIGHTFV(GL_LIGHT4, GL_POSITION, LIGHT_POSITION3);
    GLLIGHTFV(GL_LIGHT4, GL_AMBIENT, AMBI);
      GLLIGHTFV(GL_LIGHT4, GL_DIFFUSE, LIGHTZEROCOLOR);
      /* LIGHT 5 */

    GLLIGHTFV(GL_LIGHT5, GL_POSITION, LIGHT_POSITION3);
      GLLIGHTFV(GL_LIGHT5, GL_AMBIENT, AMBI);
      GLLIGHTFV(GL_LIGHT5, GL_DIFFUSE, LIGHTZEROCOLOR);
    /* LIGHT 6 */

    GLLIGHTFV(GL_LIGHT6, GL_POSITION, LIGHT_POSITION3);
      GLLIGHTFV(GL_LIGHT6, GL_AMBIENT, AMBI);
      GLLIGHTFV(GL_LIGHT6, GL_DIFFUSE, LIGHTZEROCOLOR);
      /* LIGHT 7 */

    GLLIGHTFV(GL_LIGHT7, GL_POSITION, LIGHT_POSITION3);
      GLLIGHTFV(GL_LIGHT7, GL_AMBIENT, AMBI);
      GLLIGHTFV(GL_LIGHT7, GL_DIFFUSE, LIGHTZEROCOLOR);

    GLENABLE(GL_LIGHTING);

    GLENABLE(GL_LIGHT2);
    GLENABLE(GL_LIGHT3);
    GLENABLE(GL_LIGHT4);
      GLENABLE(GL_LIGHT5);
      GLENABLE(GL_LIGHT6);
```

```
    //glDepthFunc(GL_GEQUAL);
    //glEnable(GL_DEPTH_TEST);
}

void  gravadados(double f, int user ){
  FILE *fptr;
  fptr = fopen("dados.txt","a+");
  fprintf(fptr,"%f (f/s)  <--> %i  \n", f, user);
  fclose(fptr);

}


int verificaDistancia(ObjectData_T *object,int dist, int idPatt, int
flag ){

      double matInv [3][4], matResult[3][4];
      static double matLast[3][4];
      double refX,refY,refZ;

      if(flag){
            if(object[idPatt].visible){
                  for(int i=0;i<3;i++){
                        for(int j=0;j<4;j++)

      matLast[i][j]=object[idPatt].trans[i][j];
                  }
            }
      }

      else{
            if(object[idPatt].visible){
                  arUtilMatInv(matLast, matInv);
                  arUtilMatMul(matInv,          object[idPatt].trans,
matResult);

                  refX=matResult[0][3];
                  refY=matResult[1][3];
                  refZ=matResult[2][3];

                  dist= refX*refX+refY*refY+refZ*refZ;
                  if(dist>(500)){
                        for(int i=0;i<3;i++){
                              for(int j=0;j<4;j++)

      matLast[i][j]=object[idPatt].trans[i][j];
                        }

                        return 1;
                  }
```

```
            }
        }

        RETURN 0;
}
```