ESCOLA POLITÉCNICA DE
PERNAMBUCO

Trabalho de Conclusão de Curso

Engenharia de Computação

Ferramenta Colaborativa para Prototipação de Ambientes Físicos Utilizando Realidade Aumentada

Autor: Carlos Henrique Barbosa da Cunha

Orientador: Prof. Sérgio Murilo Maciel Fernandes

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA DE PERNAMBUCO

CARLOS HENRIQUE BARBOSA DA CUNHA

FERRAMENTA COLABORATIVA PARA PROTOTIPAÇÃO DE AMBIENTES FÍSICOS UTILIZANDO REALIDADE AUMENTADA

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, junho de 2010.

Aos meus pais, por todo apoio e dedicação.

Agradecimentos

Agradeço aos meus pais Sr. Gilvan Floriano e Sra. Maria de Lourdes pela confiança e pelo apoio em todas as etapas da minha vida.

Agradeço à minha namorada Maria Eleonora pela compreensão e companheirismo em todos os momentos durante estes últimos 5 anos.

Agradeço ao prof. Sérgio Murilo por ter me orientado neste e em outros dois trabalhos sobre Realidade Aumentada com contribuições bastante significativas, mesmo não sendo esta a sua área de pesquisa.

Agradeço aos colegas da graduação pela cooperação nesta luta acadêmica que finalmente está sendo vencida, e que nos deixou mais preparados para uma próxima batalha.

Resumo

Este trabalho discute os recursos da Realidade Aumentada em favor da prototipação de ambientes físicos, uma etapa importante no desenvolvimento de projetos relacionados à arquitetura, engenharia civil, entre outros. Tradicionalmente, este tipo de prototipação é realizada através de ferramentas como lápis, maquetes ou softwares de modelagem. O propósito desta pesquisa é apresentar uma nova maneira de realizar prototipação de ambientes físicos, utilizando Realidade Aumentada, sem uso dos dispositivos de entrada convencionais, tais como *mouse* e teclado. Dessa maneira, foi construído um protótipo baseado em um novo framework chamado SudaRA.

Abstract

This document discusses the features of Augmented Reality in favor of prototyping physical environments, an important step in project development related to architecture, civil engineering, among others. Traditionally, this kind of prototyping is performed using tools such as pencils, physical models or modeling software. The purpose of this research is to present a new way to perform prototyping of physical environments, using Augmented Reality, without the use of conventional input devices such as mouse and keyboard. Thus, a software prototype was built based on a new framework called SudaRA.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

CAD - Computer Aided Design

RA - Realidade Aumentada

SDL - Simple DirectMedia Layer

RV - Realidade Virtual

HMD - Head Mounted Display

EVI - Easy Visualization In-Situ

API - Application Programming Interface

GPU - Graphics Processing Unit

CAVE - Cave Automatic Virtual Environment

OSG - OpenSceneGraph

OSGART - OpenSceneGraph for ARToolKit

GUI - Graphical User Interface

GLUT - Graphical Library Utility Toolkit

Cal3d - Character Animation Library

GLSL - OpenGL Shading Language

OpenAL - Open Audio Library

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

STL - Standard Template Library

GPL - General Public License

IDE - Integrated Development Environment

ODE - Open Dynamics Engine

VRML - Virtual Reality Modeling Language

Capítulo 1
Introdução

Nas etapas iniciais de um projeto, independentemente da área de atuação, o nível de prototipação é fundamental para servir como base para a etapa de desenvolvimento.

Na implementação de projetos de ambientes físicos como na construção civil ou na arquitetura, por exemplo, há uma necessidade de se visualizar o produto final ainda nos estágios iniciais do projeto.

Para a realização dessas atividades, o uso de lápis e papel são os mais práticos. Há também as ferramentas CAD (*Computer Aided Design*) convencionais, onde modelos 2d ou 3d são construídos e dispostos em um ambiente virtual com uso do mouse e teclado, porém a interação do usuário com tais ferramentas ainda é limitada a uma interface 2D.

A Realidade Aumentada (AZUMA, 1997) tem abordado vários ramos em suas aplicações devido a uma maneira simples e inovadora de interação com o usuário. Porém, projetos de auxílio ao desenvolvimento, como é o caso da prototipação, ainda são pouco explorados.

Com o aumento da complexidade dos projetos, o crescimento da globalização e da necessidade do homem atual estar em vários lugares em um curto espaço de tempo, há uma preocupação de se criar soluções colaborativas que façam uso de acesso remoto que ajudem na proximidade desses profissionais em um projeto comum.

Este trabalho visa apresentar uma alternativa às ferramentas tradicionais para prototipação de ambientes físicos como condomínios, rodovias, estabelecimentos comercias, etc, através de uma ferramenta colaborativa, que utiliza a Realidade Aumentada como principal recurso de interface com o usuário. Desta forma, o projetista terá uma maior praticidade para manipular os objetos do seu projeto além de poder contar com a participação de usuários remotos.

Para a construção do protótipo apresentado, foi necessário desenvolver um *framework* que abstrai a complexidade de uso de bibliotecas para RA, desenho 2d e 3d, rede, entre outras, facilitando tanto a criação quanto a manutenção deste projeto. Este *framework* foi batizado por SudaRA (**Su**porte ao **D**esenvolvimento de **A**plicações em **R**ealidade **A**umentada) e será detalhado mais adiante.

1.1. Metodologia e Estratégia de Ação

O trabalho realizado foi baseado em uma pesquisa de natureza aplicada, onde o projeto final resultou numa ferramenta que implementa funcionalidades básicas para prototipação de ambientes físicos. Seguem abaixo os passos para a realização deste trabalho:

1. Estudo dos fundamentos de Realidade Aumentada e revisão bibliográfica

Estudo aprofundado nos conceitos de RA, bem como nas ferramentas e técnicas já existentes.

2. Estudo da biblioteca ARToolKit (KATO e BILLINGHURST, 2007)

ARToolKit é uma biblioteca para reconhecimento de padrões usada para implementação de aplicações em RA. Esta foi estudada com o objetivo de entender suas principais funcionalidades.

3. Desenvolvimento do *framework* SudaRA

Nesta etapa, foi criado o *framework* SudaRA baseado em ARToolKit e SDL (*Simple DirectMedia Layer*) (LANTINGA, 2007).

Por meio do encapsulamento de todas as funcionalidades dessas bibliotecas, o objetivo deste *framework* foi estabelecer uma estrutura coerente e simplificada para facilitar o desenvolvimento de aplicações que utilizam recursos de Realidade Aumentada, como é o caso da solução principal.

4. Estudo sobre prototipação e modelagem com ferramentas CAD

Foi realizado um estudo sobre as principais funcionalidades de ferramentas de modelagem 3d, tais como Blender (BLENDER FOUNDATION, 2009) e SketchUp (GOOGLE, 2010).

5. Desenvolvimento de uma ferramenta de prototipação de ambientes físicos

Neste momento foi desenvolvida de fato a aplicação principal deste trabalho que funciona baseado na seleção de modelos para compor ambientes físicos através do posicionamento destes objetos 3d em um ambiente real, incluindo a possibilidade de colaboração pela da rede onde outras pessoas possam interagir remotamente.

Abaixo, segue uma lista das ferramentas usadas na implementação deste projeto:

- Visual C++ Express (MICROSOFT, 2009) para o ambiente de desenvolvimento;

- O *framework* SudaRA;

- Blender para a modelagem dos objetos 3d;

- GIMP (NATTERER et al., 2009) para a edição de texturas e elementos de interface gráfica;

- Dois computadores com *WebCam*.

1.2. Estrutura da Monografia

Este trabalho está dividido em cinco capítulos. O capítulo 2 mostra os fundamentos da Realidade Virtual e Aumentada, suas ferramentas de desenvolvimento (com ênfase em ARToolKit) e aplicações.

O capítulo 3 apresenta o *framework* SudaRA, criado com o objetivo de facilitar o desenvolvimento da aplicação principal além de aplicações futuras que utilizem recursos de Realidade Aumentada.

O capítulo 4 descreve o protótipo desenvolvido com o objetivo de demonstrar a capacidade da RA para prototipação de ambientes físicos.

Por fim, o capítulo 5 conclui este documento trazendo uma reflexão do que foi trabalhado e sugerindo melhorias futuras.

Capítulo 2
Realidade Virtual e Aumentada

Este capítulo aborda os principais fundamentos da Realidade Virtual (RV) e da Realidade Aumentada (RA), apresentando seus sistemas de visualização, equipamentos utilizados entre outros aspectos que acompanharam a evolução desta tecnologia.

Aplicações baseadas em RV e RA também são mostradas neste capítulo, assim como algumas das ferramentas de desenvolvimento mais utilizadas atualmente. Por fim, relacionamos alguns trabalhos os quais inspiraram a criação do protótipo proposto.

2.1. Fundamentos da Realidade Virtual e Aumentada

A concepção de sistemas tanto de Realidade Virtual quanto Aumentada reúnem vários conceitos e tecnologias da computação. Assim, técnicas de processamento de imagem, interação humano-computador, visão computacional, computação gráfica, representação 3d, entre outras compõem os recursos disponíveis para o desenvolvimento desses sistemas.

2.1.1. Realidade Virtual

Realidade Virtual é um termo que pode ser aplicado a ambientes de simulação em tempo real projetados por um computador, é uma maneira de visualizar, manipular e interagir com computadores e dados complexos (AUKSTAKALNIS, 1992).

O avanço tecnológico possibilitou o desenvolvimento de sistemas de RV que reproduzem ambientes estáticos e em movimentos com alta fidelidade, levando o usuário a interagir com situações reais ou fictícias. Um exemplo seria reproduzir um banco virtual, uma cidade, ou uma escola onde o usuário pudesse navegar e interagir com esses ambientes (TORI e KIRNER, 2006).

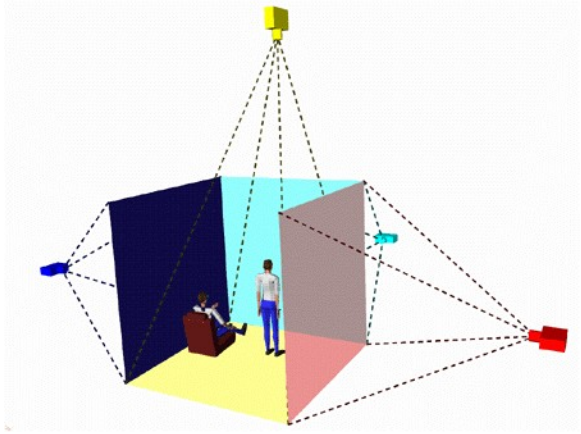
Um dos principais objetivos da Realidade Virtual é tornar a interação com o sistema de forma mais natural possível (TEICHRIEB e FIGUEIREDO, 2010), para tanto, recursos avançados de *hardware* e *software* vêm sendo desenvolvidos, aumentando o envolvimento entre os usuários e suas aplicações.

Outro fator importante na Realidade Virtual está relacionado com a possibilidade ou não de imersão. Sistemas imersivos baseiam-se no uso de dispositivos HMD (*Head Mounted Display)* ou em salas de projeção. Já os sistemas RV não-imersivos caracterizam-se pelo uso do monitor tradicional.

Dispositivos HMD são capacetes de visualização que possibilitam ao usuário uma melhor sensação de estar no ambiente virtual projetado. A Figura 1 mostra um exemplo desse dispositivo. Apesar do seu poder de imersão, os HMDs tem suas desvantagens como alto custo e desconforto por parte de quem usa.

Figura 1. Capacete de visualização para RV imersiva, conhecido por HMD

Salas de projeção, também conhecidas como CAVE (*Cave Automatic Virtual Environment*) (CRUZ-NEIRA et al., 1993) são estruturas onde as paredes, o teto e o chão servem como superfícies de projeção. O ambiente virtual mostrado nesse sistema muda de acordo com a posição e rotação do usuário, fazendo-o navegar e interagir com uso de dispositivos de entrada diversos. A Figura 2 apresenta o esquema de projeção de uma CAVE (2.a) e a mesma em execução (2.b).



(a)

(b)

Figura 2. Estrutura de projeção (a) e execução de uma CAVE (b)

Recursos como dispositivos de rastreamento e luvas eletrônicas são comuns em sistemas de Realidade Virtual (KIRNER e PINHO, 1997) e servem para manipular o ambiente virtual de acordo com a movimentação de uma ou várias partes do corpo.

Os rastreadores são divididos entre 6 categorias: mecânicos, ultrassônicos, magnéticos, óticos, inerciais e por extração de imagens.

- Rastreadores Mecânicos: usados onde é necessário alta velocidade e precisão do rastreamento. Basicamente trata-se de um braço mecânico articulado preso a uma parte do corpo, como por exemplo, a cabeça como mostra a Figura 3.

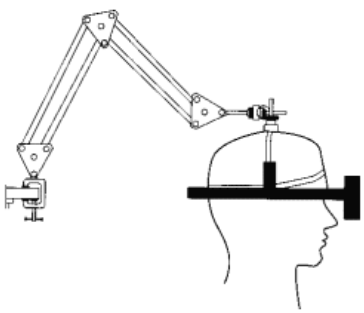


Figura 3. Rastreador mecânico preso à cabeça

- Rastreadores Ultrassônicos: com esse tipo de rastreador, é possível determinar uma posição através da captação do som emitido por esse dispositivo. O cálculo da posição é possível a partir do tempo decorrido desde a emissão até a recepção pelos captadores.

- Rastreadores Magnéticos: são rastreadores que produzem campos magnéticos, possuindo sensores para determinar o tamanho e a direção desses campos.

- Rastreadores por Extração de Imagens: são rastreadores baseados na filmagem de uma pessoa com *leds* em locais específicos do corpo. Uma vez que esses *leds* são capturados, o sistema processa esses dados e calcula a posição do usuário.

- Rastreadores Óticos: são o inverso dos rastreadores por extração de imagens. Câmeras são colocadas sobre a cabeça do usuário que filmam o teto composto por uma matriz de *leds* estáticos. A posição desse usuário é calculada a partir de um padrão de *leds* piscando enquanto o mesmo caminha pela sala.

- Rastreadores sem Referencial: são rastreadores que medem giros ou inclinações a partir de uma posição inicial.

As luvas eletrônicas, têm por objetivo obter informações referentes à movimentação das mãos e dos dedos e usá-las para a interação com a aplicação RV. Algumas luvas servem também como uma forma de retorno de sensação háptica, sendo assim estas apresentam uma forma de entrada e de saída em relação ao sistema em execução. A Figura 4 apresenta um exemplo de luva eletrônica interagindo com uma aplicação RV.

Figura 4. Exemplo de interação com uma luva CyberGlove da empresa Virtual Technologies

Além dos dispositivos citados vários outros podem ser encontrados abordando outras sensações do ser humano, como temperatura, equilíbrio, audição, etc (MACHADO, 2010).

2.1.2. Realidade Aumentada

A Realidade Aumentada é uma variação da Realidade Virtual onde é possível inserir elementos virtuais dentro do mundo real, ao contrário da RV onde todo o ambiente é virtual (AZUMA, 1997). Para que isso seja possível, um dispositivo é usado para capturar as imagens do mundo real e enviá-las para um software que faz um tratamento das informações obtidas através de técnicas de visão computacional para poder inserir e manipular os objetos virtuais. Todo esse processe é realizado em tempo real.

O exemplo abaixo (Figura 5) mostra uma sala contendo uma mesa com um abajur e um telefone em cima, além de duas cadeiras. Porém apenas a sala, a mesa e o telefone são reais, o abajur e as cadeiras foram adicionados virtualmente à cena.

Figura 5. Realidade Aumentada aplicada a uma sala

2.1.3. Aplicações

Uma vez que esses sistemas, tanto em RV quanto em RA, abordam diversas áreas de conhecimento, tecnologias e formas de interação, inúmeras possibilidades de aplicação podem ser exploradas.

Na medicina, por exemplo, com uso de pequenos marcadores retro-reflexivos posicionados no pé do paciente (Figura 6), é possível visualizar a sua estrutura óssea através do sistema EVI (Easy Visualization In-Situ) (NAVAB et al., 2007).

Figura 6. Sistema RA aplicado à visualização da estrutura óssea

Na área de treinamento, o uso de RA e RV também é muito explorado. A Embraer, empresa nacional atuante na construção e manutenção de aviões, possui uma sala equipada com sistemas de visualização onde pilotos e engenheiros são treinados com uso de aeronaves virtuais como mostra a Figura 7.

Figura 7. Treinamento de construção de aeronaves e simulação de voo com RV

Os jogos digitais são os representantes mais famosos da área de Realidade Virtual, suas tecnologias têm contribuído desde os anos 90 com o surgimento do jogo Doom, um dos pioneiros a transmitir a sensação de estar dentro do ambiente visualizado.

Em Realidade Aumentada, os jogos ainda não foram tão difundidos, porém há muita pesquisa nesta área. Um dos exemplos é o jogo Torre de Hanói que foi adaptado por Ezequiel Zorzal (ZORZAL et al., 2006) para ser jogado usando os recursos de Realidade Aumentada (Figura 8).

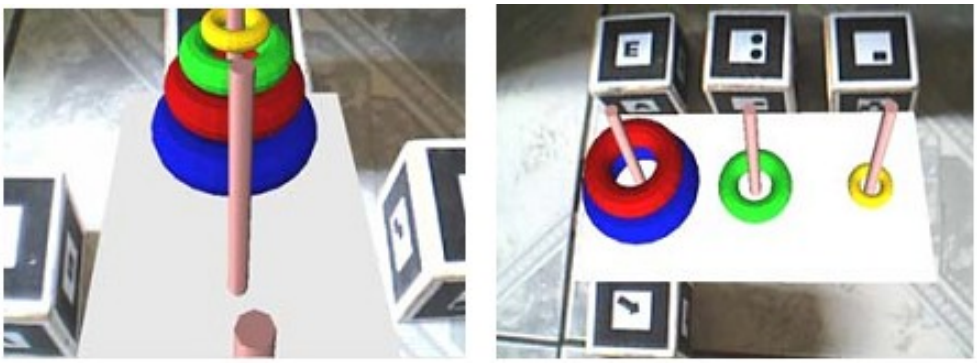


Figura 8. Torre de Hanói com Realidade Aumentada

2.2. Ferramentas de desenvolvimento

A construção de aplicações em RV e RA é possível graças ao constante avanço das suas ferramentas de desenvolvimento. APIs (*Application Programming Interface*) como OpenGL (SILICON GRAPHICS, 1992) e Direct3d (MICROSOFT, 1995) trouxeram recursos para o surgimento de diversas ferramentas que simplificam o desenvolvimento de aplicações gráficas. Entre as mais famosas, é possível destacar Ogre3d (TORUS KNOT SOFTWARE, 2000), OpenSceneGraph (OSFIELD e BURNS, 1998), ARToolKit e OSGART (HITLABNZ, 2006). Suas características mais comuns são: importação e manipulação de modelos 3d pelo grafo de cena (detalhes na seção 3.2.2.) e suporte a *shaders*.

Shaders são pequenos programas específicos para manipulação de vértices e pixels de uma aplicação gráfica, podendo alterar o aspecto visual dos modelos exibidos, gerando assim efeitos diversos. Diferentemente de um programa comum, os *shaders* são executados apenas no processador da placa de vídeo, conhecidos por GPU (*Graphics Processing Unit*).

Ogre3d é um motor gráfico multiplataforma feito em C++ para o desenvolvimento de aplicações gráficas. Baseado tanto em Direct3d quanto OpenGL. Entre as suas características encontram-se: suporte a modelos animados, *shaders,* sombras, além de fácil integração com bibliotecas de som, rede, entre outras. O exemplo na Figura 9 mostra a EGO Game Editor (SALGUEIRO, 2009), um editor de cenas baseado em Ogre3d.

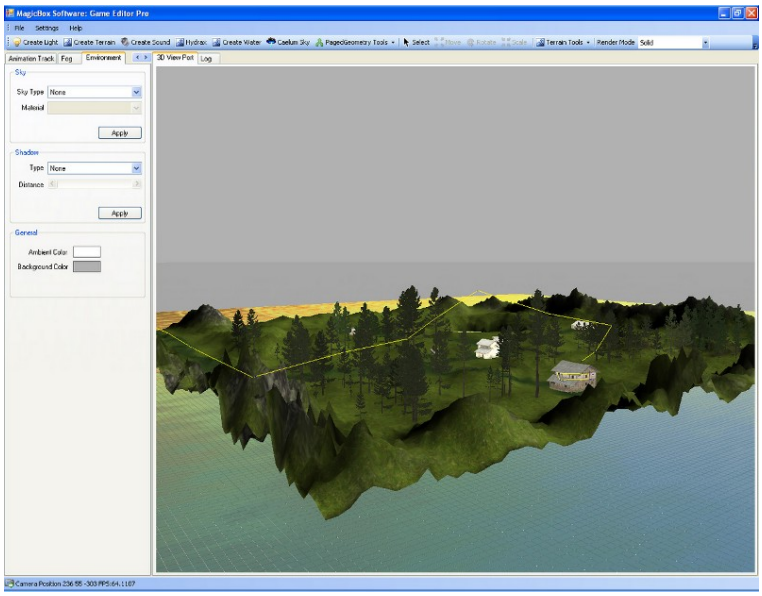


Figura 9. Editor de Cenas criado com Ogre3d.

OpenSceneGraph, também conhecido por OSG, é uma ferramenta gráfica multiplataforma de alta performance, baseada em C++ e OpenGL. Possui suporte para diversos formatos de modelos 3d incluindo COLLADA, OBJ, 3DS, entre outros. Tem suporte a *shader* e a outras linguagens como Java, Lua e Python.

ARToolKit é uma biblioteca para a construção de aplicações em Realidade Aumentada. Através de algoritmos de visão computacional, a ARToolKit consegue detectar marcadores com borda quadrada, colocando à disposição informações sobre a posição e rotação desses padrões. Com isso é possível incluir elementos virtuais dentro da cena real capturada. Esta ferramenta é multiplataforma, baseada em C e OpenGL. A Figura 10 mostra uma visão geral do funcionamento da ARToolKit.

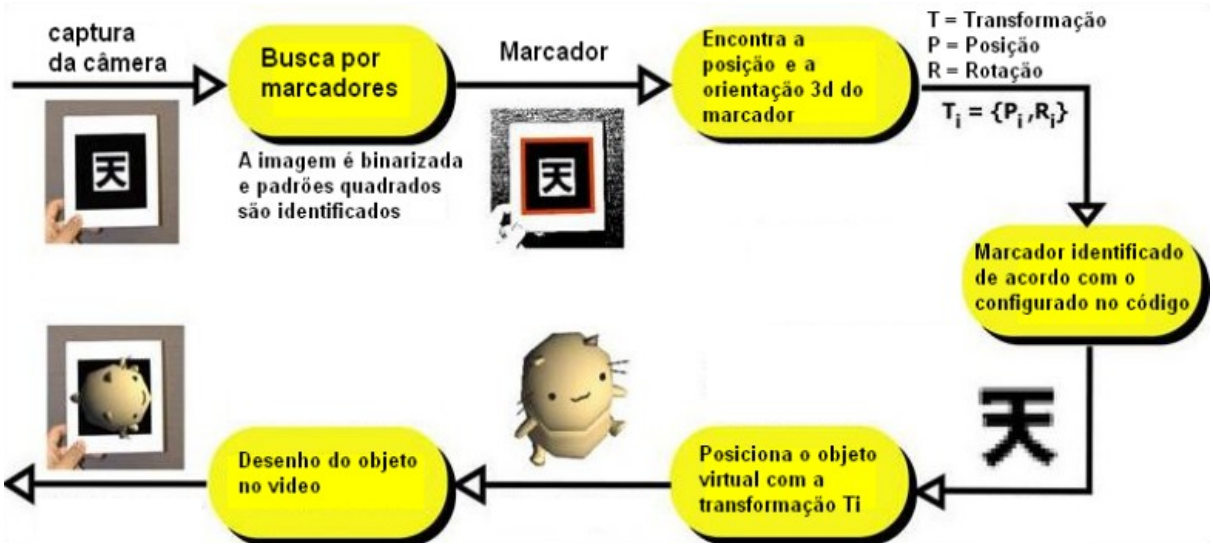


Figura 10. Funcionamento da ARToolKit

Por fim, a OSGART trata-se de uma biblioteca baseada na combinação entre OpenSceneGraph e ARToolKit, trazendo características como design orientado a objetos, suporte a vários modelos 3d, entre outras, agregadas aos recursos de detecção de padrões. Desta forma, é possível criar aplicações em Realidade Aumentada com as facilidades destas duas bibliotecas numa só ferramenta.

2.3. Colaboração em RV e RA

O trabalho colaborativo em Realidade Aumentada é um conceito recente que está relacionado com a atuação de múltiplos usuários em uma determinada aplicação, simultaneamente, de forma a atingir um objetivo comum (BILLINGHURST e KATO, 2002).

Normalmente, os usuários desse tipo de sistema estão separados geograficamente, cada um com seu equipamento (computador, câmera, etc), porém, interagindo na mesma aplicação em tempo real. Para que isso seja possível, as máquinas devem está conectadas entre si de alguma forma que possam trocar informações sobre as ações realizadas e os objetos virtuais manipulados.

O projeto HydroVR (LIDAL et al., 2007) é um sistema colaborativo para treinamento em exploração de óleo através de uma CAVE. A Figura 11 mostra a aplicação onde um profissional interage com um avatar controlado por um usuário remoto.

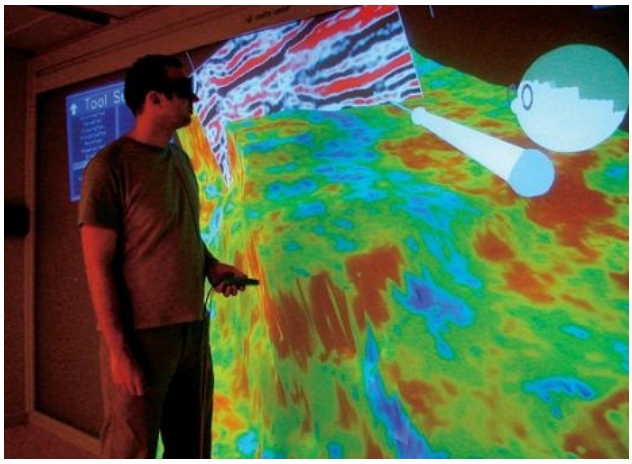


Figura 11. Projeto HydroVR em funcionamento

Outra aplicação interessante nesta área é o projeto NHE (TAVARES, 2009) pelo qual foi criado um ambiente colaborativo onde os usuários são auxiliados na visualização e resolução de trabalhos em equipe através da Realidade Aumentada. A Figura 12 apresenta a tela de diálogo do NHE. Neste caso, o computador da esquerda desenha um modelo virtual manipulado pelo usuário do computador da direita.

Figura 12. Tela de diálogo do projeto NHE

2.4. Trabalhos Relacionados

Entre os trabalhos relacionados com a ferramenta proposta, é possível destacar o 3DARModeler (DO e LEE, 2008) e o Projeto Tinmith (PIEKARSKI, 2004).

O 3DARModeler é uma versão simplificada de uma ferramenta de modelagem que possui as principais funções para construção de modelos 3d com recursos para aplicação de textura, animação, entre outros, combinando dispositivos de entrada tradicionais (teclado e mouse) com comandos realizados a partir de marcadores.

Durante a criação, objetos primitivos podem ser adicionados aos marcadores (Figura 13).

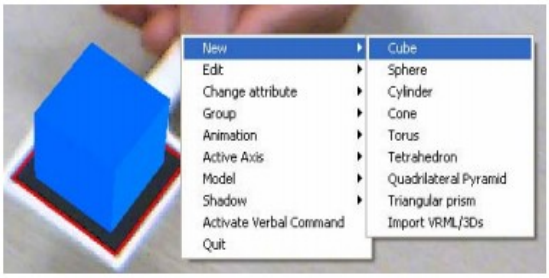


Figura 13. Escolha de modelos primitivos no 3DARModeler

Há também a possibilidade de importar objetos previamente modelados e salvos em um computador local (Figura 14).

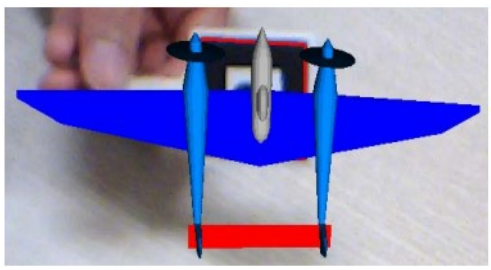


Figura 14. Modelo importado para o 3DARModeler

O projeto Tinmith trata de uma aplicação móvel voltada para modelagem 3d de grandes estruturas em ambientes externos. O projeto é composto por um hardware específico chamado de "*backpack*" (Figura 15.a) que envolve um computador executando seu software (Figura 15.b), um capacete com um HMD, luvas e dispositivos *wireless*.

(a) (b)

Figura 15. *Backpack* (a) e o software em execução (b) do projeto Tinmith

Capítulo 3
SudaRA

O desenvolvimento de aplicações gráficas é reconhecido como uma tarefa complexa em relação a outras categorias de aplicações da computação, por requerer conhecimentos em áreas diversas como, por exemplo, geometria e artes visuais. Dependendo do escopo, o desenvolvedor precisará, ainda, adquirir conhecimentos em física, anatomia, geografia, etc. Isto se agrava quando são utilizadas bibliotecas básicas como OpenGL ou Direct3d.

Nos últimos anos, esta realidade tem melhorado graças ao surgimento de plataformas que abstraem o uso dessas bibliotecas, fornecendo uma estrutura mais agradável para o desenvolvedor. Porém, ainda há poucas soluções semelhantes quando se necessita do uso de Realidade Aumentada.

Outra dificuldade é encontrada quando uma aplicação gráfica requer outros recursos como rede, som, etc. Neste caso, o desenvolvedor precisa buscar bibliotecas externas e agregá-las ao seu projeto para satisfazer suas necessidades.

SudaRA é um *framework* de suporte ao desenvolvimento de aplicações que utilizam Realidade Aumentada. Esta plataforma, desenvolvida em C++, tem a finalidade de unir algumas tecnologias disponíveis, em uma estrutura simplificada, para que o desenvolvedor tenha em mãos, recursos como importação de modelos 3d, elementos de GUI (*Graphical User Interface*), som, rede, além de outros, em uma só ferramenta.

3.1. Funções e Tecnologias Utilizadas

As funcionalidades da SudaRA estão relacionadas com o uso de algumas tecnologias já existentes, cada uma atuando de acordo com sua especificidade. Todas elas são multiplataforma e de código aberto, tornando a portabilidade da SudaRA mais eficaz e menos trabalhosa.

A principal biblioteca utilizada é a já citada ARToolKIt, que provê suporte para detecção de padrões, além de trazer consigo a OpenGL e a GLUT (*Graphical Library Utility Toolkit*) que é uma biblioteca que abstrai funcionalidades do Sistema Operacional em relação a criação e controle de janelas e tratamento de eventos (teclado e mouse).

A GLUT, porém, apresenta várias limitações, principalmente a falta de um suporte satisfatório com relação a imagens. Por isso, preferiu-se modificar internamente a ARToolKit para substituir a GLUT pela SDL que além de ter as funções básicas da GLUT, ainda possui suporte a operações com vários formatos de imagens, e recursos para concorrência, rede, *joystick*, e outros.

As demais tecnologias são apresentadas de acordo com as funcionalidades da SudaRA:

3.1.1. Detecção de padrões

A detecção de padrões é essencial para o desenvolvimento de uma aplicação RA, pois é a partir das informações obtidas que se faz o controle dos eventos e a manipulação dos elementos virtuais.

Para esta funcionalidade, SudaRA baseia-se na ARToolKit, e os padrões usados são marcadores com bordas quadradas como mostra a Figura 16.

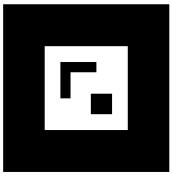


Figura 16. exemplo de marcador

O *framework* descrito suporta um ou mais marcadores na mesma aplicação, permitindo que o desenvolvedor tenha acesso às informações sobre posição, rotação e visibilidade dos mesmos.

3.1.2. Gráficos

Os elementos 2d na SudaRA são compostos por imagens, textos e botões. Por outro lado, modelos estáticos e animados, além de malhas personalizadas compõem os elementos 3d disponíveis pelo *framework*.

A SDL_image é a biblioteca responsável pelo o carregamento das texturas usadas tanto nas imagens 2d quanto nos modelos 3d. Além da compatibilidade com a SDL, a SDL_image traz como principal vantagem o suporte a diversos formatos de imagens, incluindo JPG e PNG. Já para a implementação de textos, a API responsável é a SDL_ttf.

Modelos estáticos são objetos 3d que não possuem animação associada. SudaRA importa modelos estáticos no formato OBJ. Já os modelos animados são carregados e manipulados com uso da biblioteca Cal3d (*Character Animation Library*) (HEIDELBERGER, 2006).

Toda parte visual, além da iluminação e transformações geométricas são comandadas pela OpenGL, uma API multiplataforma, utilizada há mais de 15 anos por diversas aplicações gráficas incluindo jogos famosos.

O suporte a programação com *shaders* fica por conta da GLSL (*OpenGL Shading Language*). Por enquanto esses efeitos estão agregados aos modelos 3d e podem ser facilmente carregados e ativados em qualquer momento durante a execução da aplicação.

3.1.3. Som

A importância do áudio em aplicações gráficas depende em grande parte do propósito da mesma. Em jogos, por exemplo, o som é fundamental para um maior envolvimento do usuário com o ambiente apresentado na tela.

SudaRA utiliza a OpenAL (*Open Audio Library*) (LOKI SOFTWARE, 2009) para reproduzir sons 2d (sons de botões, por exemplo) ou 3d (áudio referente a uma posição 3d no cenário da aplicação).

3.1.4. Rede

O uso de rede neste *framework* baseia-se no papel do cliente e do servidor. O desenvolvedor de uma determinada aplicação escolhe o protocolo (TCP ou UDP) a ser utilizado e faz chamadas a funções de envio e recebimento de mensagens. Desta forma é possível criar aplicações RA que envolva mais de um usuário em computadores diversos sem qualquer trabalho a nível de *sockets*.

Para esta funcionalidade, a biblioteca SDL_net foi escolhida pela compatibilidade com a SDL e pela facilidade de implementação dos recursos de rede necessários.

A Tabela 1 mostra um resumo das relações das funcionalidades com as tecnologias utilizadas pela SudaRA.

Tabela 1. Funções da SudaRA e suas tecnologias

Funcionalidades	Tecnologias
Controle de janela e eventos	SDL
Detecção de padrões	ARToolKit
Manipulação de imagens	SDL_image
Manipulação de textos	SDL_ttf
Animação	Cal3d
Gráficos 2d e 3d	OpenGL
Programação com *shaders*	GLSL
Som	OpenAL
Rede	SDL_net

3.2. Organização Interna

SudaRA está organizada em módulos, sendo cada um deles responsável por sua funcionalidade particular, e todos ligados a uma classe central que faz a interface com a aplicação do desenvolvedor.

As subseções a seguir mostram alguns detalhes de implementação dos principais conceitos para um melhor entendimento da sua arquitetura.

3.2.1. Entidades e cenários

Uma entidade significa qualquer elemento virtual mostrado na tela, como por exemplo: um modelo 3d, uma imagem ou um texto. A Figura 17 apresenta como está organizada a hierarquia das classes que representam estes componentes.

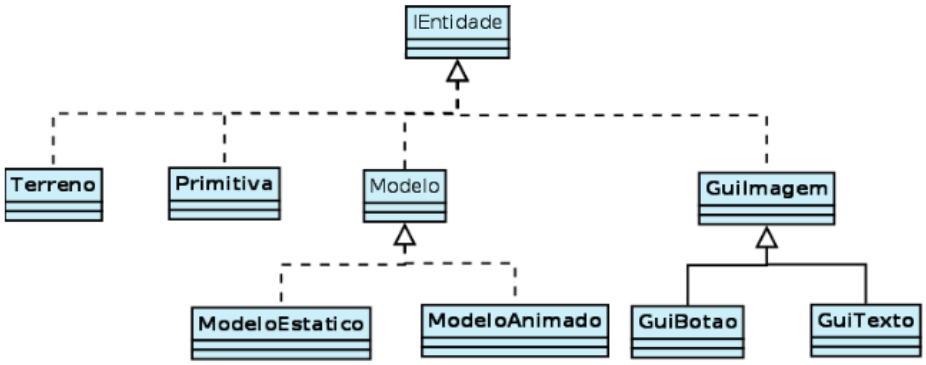


Figura 17. Hierarquia das classes que representam as entidades

Já um cenário é tratado como uma interpretação do marcador que será utilizado na aplicação desenvolvida, além de representar a base para o desenho das entidades inseridas no mesmo. Ou seja, um marcador do mundo real é na verdade um cenário em potencial.

Um cenário está relacionado com uma ou mais entidades como mostra a Figura 18. A estrutura "std::map" da STL (*Standard Template Library*) (STROUSTRUP, 1997) é usada como um mapeamento chave/valor, onde a chave é um identificador para uma entidade inserida no cenário.

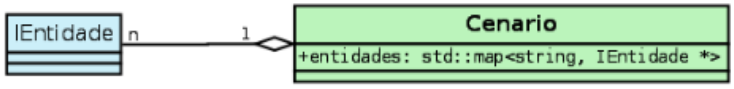
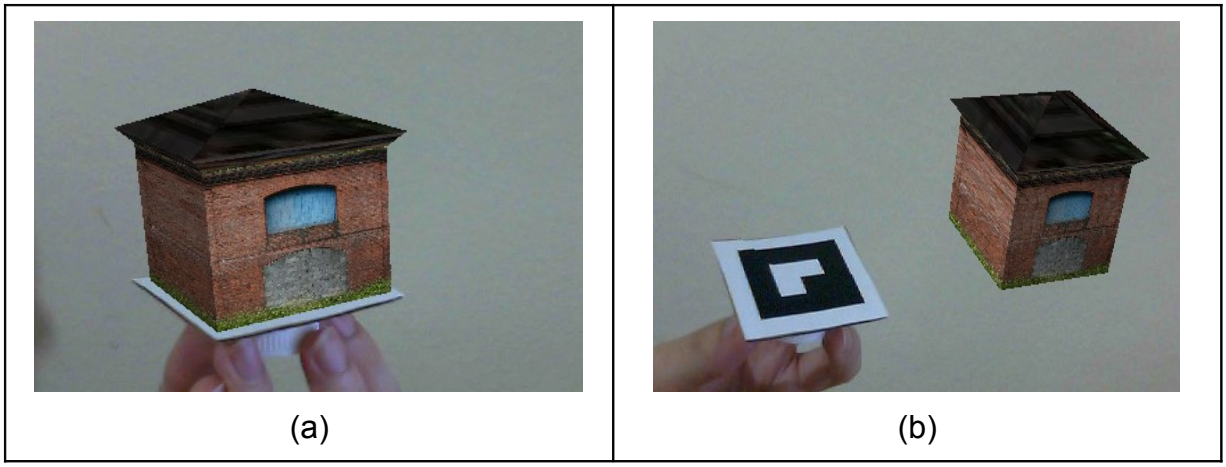


Figura 18. Relação do cenário com as entidades

Internamente, um cenário possui uma posição e uma rotação associadas. Por sua vez, cada entidade relacionada tem suas próprias transformações geométricas que são relativas ao cenário em questão. Isso faz com que todas as entidades acompanhem a posição e rotação do marcador associado, além de poder se deslocar para qualquer direção tendo como ponto de partida o cenário a que

pertencem. A Figura 19 mostra uma entidade acompanhando a posição de um marcador na sua origem (19.a) e com deslocamento (19.b).



(a) (b)

Figura 19. Exemplo de entidade situada na origem do cenário (a) e com deslocamento de 200 unidades para a direita (b)

3.2.2. Grafo de cena

Grafo de cena é um conceito utilizado em Realidade Virtual para a implementação de ambientes tridimensionais utilizando estruturas de dados (SILVA et al., 2004). Sua representação é feita em forma de árvore (Figura 20) formada por nós de cena, onde cada nó possui uma transformação geométrica que é influenciada pelos nós superiores.

Este conceito facilita, por exemplo, na representação de objetos que são compostos por uma entidade central e outras auxiliares. No caso de um veículo, as rodas seriam filhas da carcaça, que fariam sua rotação enquanto seguiam o deslocamento do seu "pai".

Baseado no exemplo abaixo, caso o CV2 (Componente Virtual 2) tenha um deslocamento de 10 unidades no eixo X, a posição dos componentes CV4, CV5 e CV6 são somadas com o deslocamento inicial do CV2.

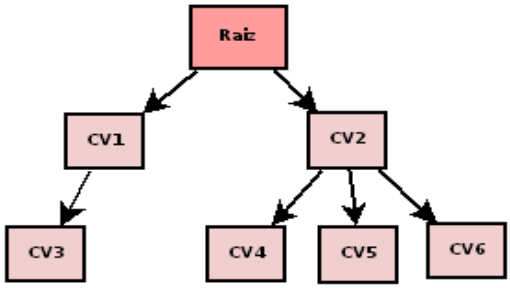


Figura 20. Demonstração de um grafo de cena. CV = Componente Virtual

Na SudaRA, um nó é representado por uma entidade a qual possui uma transformação, um "pai" e opcionalmente "filhos". Entidades sem filhos são na verdade folhas da árvore.

Há duas maneiras de se representar um ambiente 3d e uma em 2d no *framework* em questão: utilizando entidades globais, entidades associadas a um cenário ou associadas a uma interface 2d.

Entidades globais formam um grafo de cena independentemente de padrões detectados no mundo real. Sua raiz é a câmera de visualização, sendo o ponto de origem pra qualquer entidade inserida. A Figura 21.a mostra como foi idealizado o sistema de coordenadas para entidades globais, além de um exemplo de grafo de cena composto pelas mesmas (Figura 21.b).

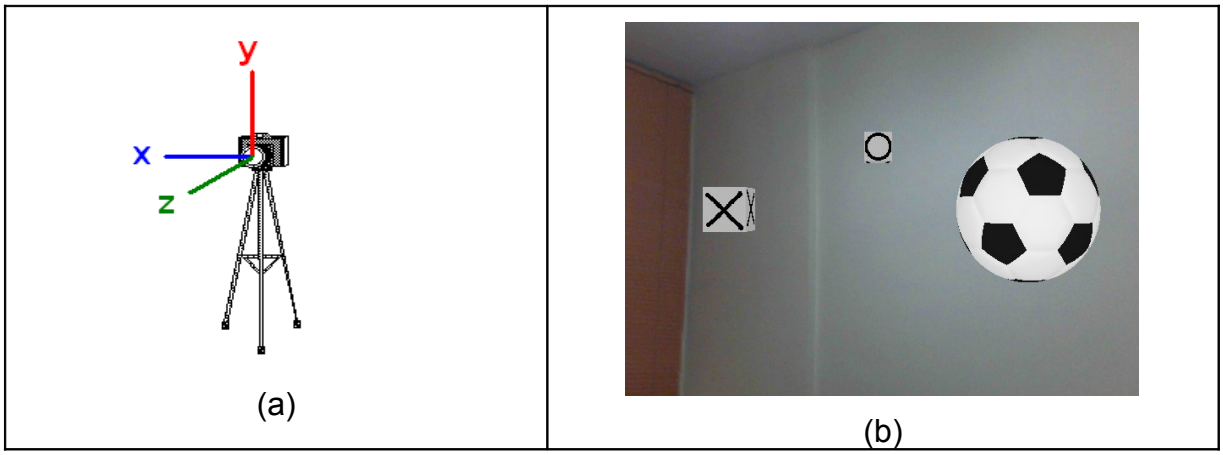


Figura 21. Sistema de coordenadas (a) e exemplo de grafo de cena para entidades globais

Entidades associadas a um cenário formam o modelo mais comum de grafo de cena encontrado nas aplicações em Realidade Aumentada, pois os elementos virtuais dependem da detecção de um marcador pra serem posicionados e desenhados.

O marcador neste caso é a raiz a qual marca a posição de origem para as entidades filhas. Sua posição e orientação em relação à câmera afetam todas as entidades associadas. A Figura 22 apresenta um exemplo junto com o sistema de coordenadas. Nota-se que neste caso, o eixo z é perpendicular à superfície do marcador.

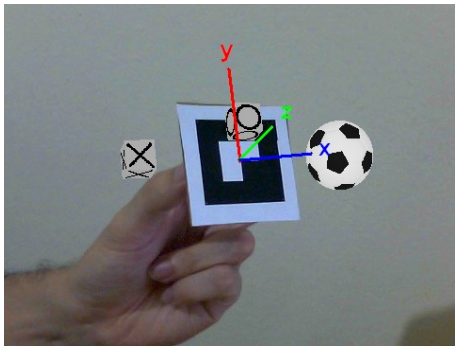


Figura 22. Cena formada por entidades associadas a um marcador e o seu sistema de coordenadas

A última forma de uso das entidades é a sua associação com a interface gráfica 2d. Nela é possível inserir tanto elementos 2d (mais comum) quanto 3d. A origem do sistema de coordenadas passa a ser o canto superior esquerdo da tela (Figura 23.a) com o "Y" crescendo pra baixo, mantendo o padrão da maioria dos motores de jogos 2d.

Assim como as entidades globais, os componentes da interface gráfica não dependem do aparecimento de algum marcador. A Figura 23.b mostra um exemplo de uma interface gráfica composta por objetos 2d e 3d.

Figura 23. Entidades associadas a uma interface gráfica. Sistema de coordenadas (a) e exemplo em execução (b)

3.2.3. Interface e Ciclo de execução

A interface da SudaRA com a aplicação do desenvolvedor é a parte mais importante deste *framework* pois permite a ele o acesso aos grafos de cena e aos eventos internos através de funções virtuais.

Este módulo agrega todos os cenários e entidades globais inseridas, além dos elementos de interface gráfica. A Figura 24 mostra uma representação do módulo de interface com alguns exemplos de funções virtuais disponíveis. O nome da classe de interface da SudaRA foi definido por "AplicacaoRA", o atributo "guiTela" é o objeto responsável pelo controle dos componentes da interface 2d.

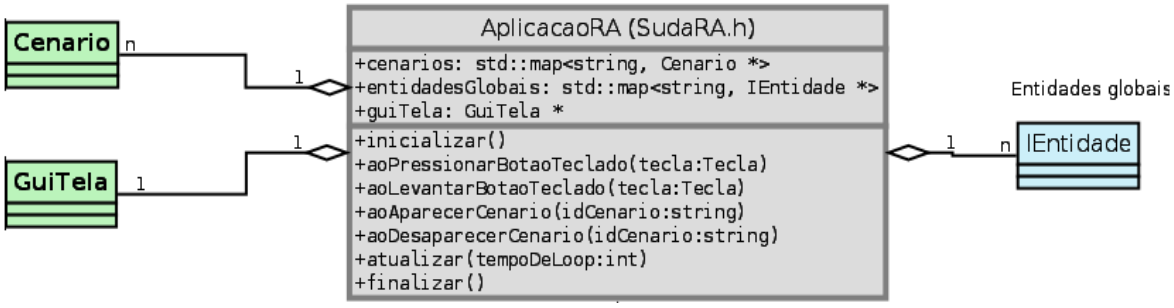


Figura 24. Módulo de interface da SudaRA

O ciclo de execução define-se pela ordem em que as principais funções são realizadas internamente. São divididas entre três etapas: gerenciamento de eventos, atualização e exibição.

O gerenciamento de eventos é responsável pela detecção dos eventos de entrada (tecla pressionada, movimentação do mouse, etc) e pelo direcionamento ao desenvolvedor o qual tomará a decisão do que fazer com os mesmos.

Atualização aplica-se à lógica do programa fazendo com que tarefas como cálculo de colisão, incremento de animação, entre outras sejam realizadas.

Por fim, a exibição está relacionada com os grafos de cena e o desenho de todas as entidades associadas. Nesta etapa, as entidades globais são inicialmente desenhadas, seguida pelas entidades associadas a cenários, e por último, as entidades da interface gráfica 2d.

A Figura 25 faz uma síntese do ciclo de execução mostrando o relacionamento entre as etapas.

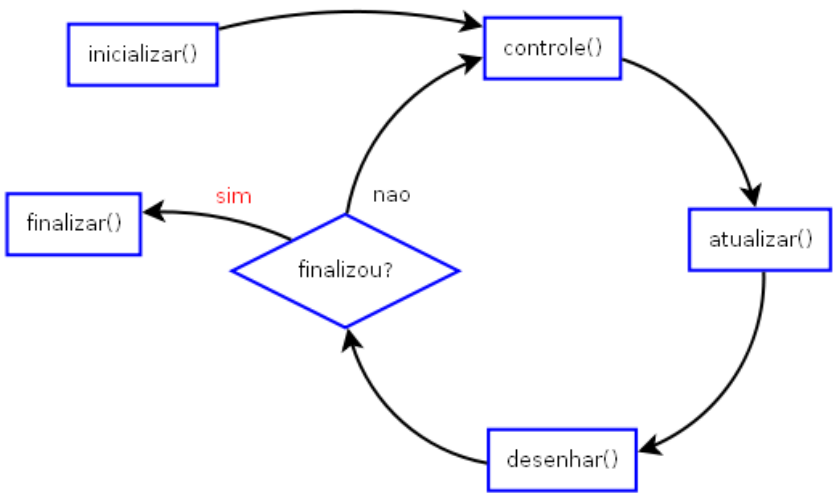


Figura 25. Etapas do ciclo de execução

3.3. Princípios de Utilização

A SudaRA está disponível no domínio da *SourceForge* (CUNHA, 2010) sob a licença GPL (FREE SOFTWARE FOUNDATION, 2007) em um pacote contendo os arquivos de cabeçalho (.h), bibliotecas estáticas (.lib) e dinâmicas (.dll). Para usá-la, basta incluir estes arquivos na IDE (*Integrated Development Environment*) de preferência do desenvolvedor.

O "programa hospedeiro" é definido como a aplicação que fará uso do *framework* SudaRA, que é de fato onde o desenvolvedor irá codificar. Para isto, é necessário criar uma classe herdando de AplicacaoRA. Esta classe terá 11 opções

de funções para sobrecarga (Tabela 2), porém apenas a função “inicializar” é obrigatória e necessária para se criar uma aplicação básica.

Tabela 2. Funções de sobrecarga disponíveis

Funções	Parâmetros	Descrição
inicializar	-	Função obrigatória. Executa configurações iniciais da aplicação
atualizar	tempoDeLoop	Chamada a cada ciclo antes de desenhar as entidades. O parâmetro “tempoDeLoop” guarda o tempo em milissegundos do último ciclo.
aoPressionarBotaoTeclado	tecla	Chamada toda vez que uma tecla é pressionada guardando a informação no parâmetro “tecla”.
aoLevantarBotaoTeclado	tecla	Chamada toda vez que uma tecla é levantada guardando a informação no parâmetro “tecla”.
aoMoverMouse	posRelativa	Chamada quando ocorre uma movimentação do mouse. O parâmetro “posRelativa” guarda a posição (“x” e “y”) relativa a posição anterior.
aoPressionarBotaoMouse	botao, posicao	Chamada toda vez que um botão do mouse é pressionado. O parâmetro “botao” referes-se ao botão pressionado e “posicao” à posição absoluta do cursor na tela.
aoLevantarBotaoMouse	botao, posicao	Chamada toda vez que um botão do mouse é levantado. O parâmetro “botao” referes-se ao botão pressionado e “posicao” à posição absoluta do cursor na tela.
aoPressionarGuiBotao	idBotao	Chamada quando um botão da GUI é pressionado. O parâmetro idBotao guarda o identificador do botão que foi pressionado.
aoAparecerCenario	idCenario	Chamada quando algum marcador é detectado. O parâmetro “idCenario” guarda o identificador do cenário relativo ao marcador.
aoDesaparecerCenario	idCenario	Chamada quando algum marcador deixa de ser detectado. O parâmetro “idCenario” guarda o identificador do cenário relativo ao marcador.
finalizar	-	Chamada quando a aplicação termina.

Abaixo (Figura 26.a) temos o exemplo mais simples de uso da SudaRA. Um modelo estático é carregado e inserido como entidade global, posicionado 200 unidades à frente da câmera resultando na aplicação mostrada na Figura 26.b. A chamada da função “vai” no “*main*” dá início ao ciclo de execução da SudaRA.

```
#include <SudaRA.h>

using namespace sudara;

class Exemplo : public AplicacaoRA
{
    public:
        Exemplo() : AplicacaoRA() {}

        void inicializar()
        {
            ModeloEstatico *arvore = new ModeloEstatico;
            arvore->carregar("arvore.obj");
            arvore->setPosicao(Vetor(0, 0, 200));

            this->adicionarEntidadeGlobal(arvore);
        }
};

int main(int argc, char **argv)
{
    Exemplo().vai("Exemplo basico com SudaRA");
    return (0);
}
```

(a)

(b)

Figura 26. Exemplo de código usando uma entidade global (a) e resultado do programa (b)

Antes de criar uma aplicação usando marcadores, o desenvolvedor precisa ter no disco rígido, os arquivos com os dados dos marcadores a serem utilizados. Atualmente, a SudaRA não faz este trabalho, porém a ferramenta mk_patt.exe da ARToolKit gera este arquivo de forma muito simples onde é necessário apenas posicionar o novo marcador na frente da câmera após a sua execução (KATO, 2010).

Durante o desenvolvimento, um objeto “cenário” é criado e a sua inicialização é feita referenciando o arquivo correspondente no disco rígido. A associação de uma entidade com este cenário é feita através da função “adicionarEntidade”. Por fim o cenário é adicionado internamente com a chamada “adicionarCenario” (Figura 27.a). O resultado é mostrado na Figura 27.b.

```
void inicializar()
{
    ModeloEstatico *arvore = new ModeloEstatico;
    arvore->carregar("arvore.obj");

    Cenario *cena = new Cenario;
    cena->carregar("marcador.ra");
    cena->adicionarEntidade(arvore);

    this->adicionarCenario(cena);

}
```

(a)

(b)

Figura 27. Exemplo de código usando uma entidade associada a um cenário (a) e resultado do programa (b)

A facilidade de uso desta plataforma é melhor observada quando compara-se a codificação desta com a da ARToolKit, por exemplo. Isso ocorre porque a SudaRA implementa as funcionalidades da ARToolKit internamente, disponibilizando para o desenvolvedor uma estrutura de mais alto nível. O Apêndice A apresenta um exemplo de código de uma aplicação semelhante ao mostrado na Figura 27, mudando apenas o modelo desenhado, neste caso, um cubo.

3.4. Limitações e Melhorias Futuras

Atualmente, a plataforma SudaRA está disponível apenas para o Windows (testes bem sucedidos foram realizados usando Windows XP e Windows 7), porém há um interesse num futuro próximo da sua portabilidade também para Linux, não apenas porque se trata de um Sistema Operacional economicamente mais acessível mas porque todas as tecnologias usadas funcionam em ambos os sistemas.

Recursos de física estão sendo pesquisados, porém, por enquanto, ainda não se encontram disponíveis. Estuda-se também a possibilidade de se agregar alguma biblioteca específica para esta funcionalidade como a ODE (*Open Dynamics Engine*) (SMITH, 2009), Bullet (COUMANS, 2010) ou PhysX (NVIDIA, 2010).

Como a detecção de padrões é completamente baseada na ARToolKit, existe uma forte dependência na questão do formato quadrado dos marcadores. Isto poderá ser resolvido buscando-se alternativas com APIs específicas para processamento de imagens como a OpenCV (INTEL, 2010), tornando os padrões buscados mais flexíveis.

O suporte aos formatos de modelos 3d também é um item importante a ser melhorado. Por enquanto, a SudaRA suporta apenas OBJ para objetos estáticos e Cal3d para objetos animados. De imediato, será incluído um módulo para importação de modelos criados com a tecnologia VRML (*Virtual Reality Modeling Language*) que já vem com o ARToolKit, posteriormente será estudada a possibilidade de incluir outros formatos.

Capítulo 4
Ferramenta Colaborativa para Prototipação de Ambientes Físicos

A criação de protótipos é uma atividade bastante relevante na fase de planejamento de um projeto, pois dá ao projetista uma noção, ainda que abstrata, do produto final a qual orientará este profissional no desenvolvimento do produto real.

Em áreas como a construção civil e arquitetura, a prototipação se dá através de desenhos a lápis, maquetes ou ferramentas de modelagem como AutoCad, SketchUp, etc. Estas ferramentas proporcionam uma maneira de criar modelos 2d e/ou 3d manipulando vértices, arestas e faces, aplicando materiais para definir cores, texturas e parâmetros de iluminação, entre outras funcionalidades. A Figura 28 mostra um exemplo de um modelo desenvolvido no SketchUp.

Figura 28. Modelo do bloco A da Escola Politécnica de Pernambuco criado no SketchUp (IAGOWS, 2008)

Em projetos envolvendo vários elementos 3d, como o de um condomínio ou de uma pequena cidade, é comum a criação de cada objeto individualmente, seguido pela junção destes em um só ambiente. É neste ponto que a solução de trabalho proposta se insere.

A ferramenta desenvolvida tem por objetivo apresentar uma alternativa interativa e de fácil manuseio para prototipação de ambientes físicos através da composição e manipulação de objetos 3d previamente criados (CUNHA e FERNANDES, 2010), além de acrescentar a participação de pessoas em lugares geograficamente separados atuando no mesmo protótipo.

4.1. Princípios de Funcionamento

O protótipo foi planejado para ser usado com um dispositivo HMD, para que o projetista fique com as duas mãos livres para manipular os marcadores que serão as suas ferramentas de prototipação.

Ao executar a ferramenta, o usuário encontrará uma tela onde deverá especificar a sua identificação na rede (o seu nome, por exemplo) e se o seu computador será cliente ou servidor. Caso seja cliente, será solicitado o endereço de IP no qual deseja se conectar. Caso seja servidor, o aplicativo atuará como servidor para si próprio e para os demais usuários.

O projetista pode também optar por não desenvolver seu projeto de forma colaborativa, construindo seu protótipo sem a necessidade de um segundo computador conectado.

Após esta etapa inicial, o aplicativo faz uma busca por todos os arquivos no formato OBJ a partir do diretório onde está localizado o executável. Estes modelos ficam então disponíveis para a composição do ambiente. As funcionalidades da ferramenta implementada baseiam-se na escolha desses modelos para posicioná-los no cenário podendo alterar seus parâmetros de rotação e escala.

Para a utilização do protótipo são necessários três marcadores: um para o cenário base, outro para o controle principal e outro para o controle auxiliar (Figura 29).

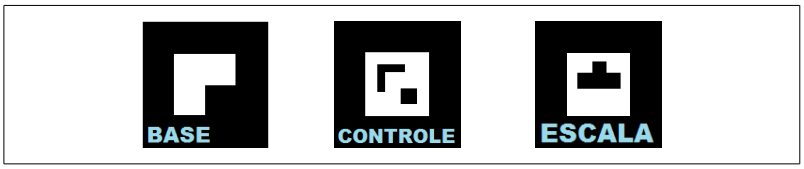


Figura 29. Marcadores utilizados no protótipo e suas funcionalidades

O marcador do cenário base fornece os dados sobre posição e orientação necessários aos modelos que estão na cena. A Figura 30 mostra um exemplo de ambiente atribuído a este marcador. Os rótulos em cima dos objetos indicam aqueles que estão sendo controlados por usuários remotos.

Figura 30. Exemplo de ambiente em desenvolvimento, com 2 usuários

O marcador de controle principal possui funcionalidades de acordo com a tela em execução. Na tela de escolha dos modelos, a seleção dos mesmos é feita girando este marcador em torno do seu eixo “z”. Os modelos são dispostos ao redor do marcador, e o modelo mais próximo da câmera fica destacado de verde, marcado como sendo o escolhido para ser inserido no cenário base (Figura 31). Já na tela principal de criação, este marcador serve para posicionar o objeto escolhido, compondo o protótipo do cenário visualizado.

O projetista pode optar por remover alguns objetos do cenário, neste caso ele tem a sua disposição uma borracha virtual que pode ser selecionada entre os modelos disponíveis. O uso da borracha consiste em encostá-la em algum modelo, depois basta esconder o marcador de controle (cobrindo-o com a mão, por exemplo) para confirmar a sua remoção.

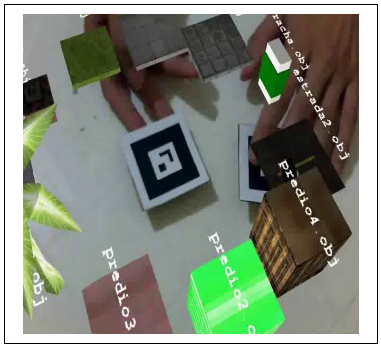


Figura 31. Tela de escolha dos modelos

O marcador de controle auxiliar traz uma funcionalidade extra ao projeto. Atuando apenas na tela de criação, este marcador é responsável pela escala do objeto controlado naquele momento. O eixo de escala é mudado quando o mesmo desaparece e aparece novamente para a câmera. A Figura 32 apresenta os estados possíveis no uso deste marcador. A escala é realizada de acordo com o eixo atual, posicionando o marcador com rotação inferior a -20 graus para decremento e superior a 20 graus para incremento.

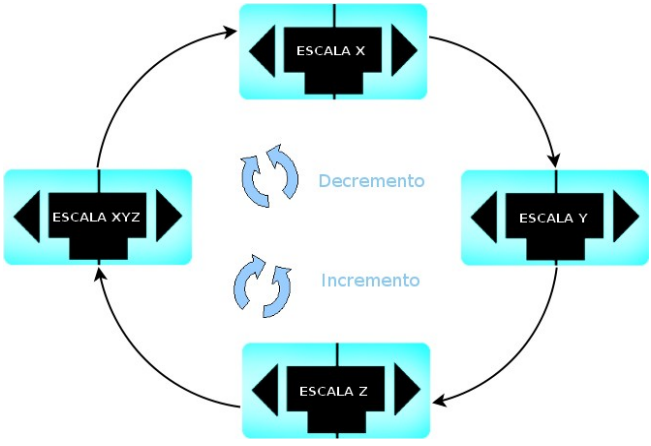
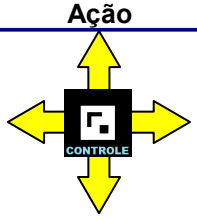


Figura 32. Diagrama dos possíveis eixos de escala

A Tabela 3 mostra um resumo dos comandos possíveis no protótipo em execução. Nota-se as várias possibilidades de interação dos marcadores com a aplicação, tornando os periféricos, como mouse e teclado, dispensáveis.

Tabela 3. Principais comandos a partir da ação dos marcadores

Ação	Comando na tela de menu	Comando na tela de cenário
 **Movimentar o marcador de controle principal**	---	Move o modelo atual nos eixos X e Y
 **Esconder o marcador do cenário**	---	Sai da tela de cenário e vai para a tela de menu
 **Mostrar o marcador do cenário**	Sai da tela de menu e vai para a tela de cenário	---
 **Esconder o marcador de controle principal**	---	Posiciona uma cópia do modelo atual no cenário
 **Mostrar o marcador de controle auxiliar**	---	Ativa a opção de escala para o modelo atual
 **Rotacionar o marcador de controle principal (eixo Z)**	Rotaciona o conjunto de modelos disponíveis para selecionar o mais próximo da câmera	Rotaciona o modelo atual
 **Rotacionar o marcador de controle auxiliar (eixo Z)**	---	Incrementa ou decrementa a escala do modelo atual

4.2. Implementação

O protótipo proposto foi desenvolvido com base no *framework* SudaRA, utilizando a linguagem C++. De acordo com a definição da sua arquitetura, a classe principal herda da classe de interface da SudaRA, sendo possível utilizar os métodos essenciais para o funcionamento do programa, além de poder fazer uso de funções

auxiliares como “aoAparecerCenario(std::string idCenario)” que é chamada toda vez que um marcador é detectado.

4.2.1. Arquitetura interna

A classe principal de interface com a SudaRA faz a ligação com o Gerenciador de Ambientes, o qual é responsável pelo controle do fluxo de execução dos ambientes. A tela de escolha dos modelos é representada pela classe AmbienteMenu e possui a estrutura de menus implementada pela classe Menu. Por outro lado, a Classe AmbientePrincipal gerencia o ambiente de composição do cenário como mostra a Figura 33.

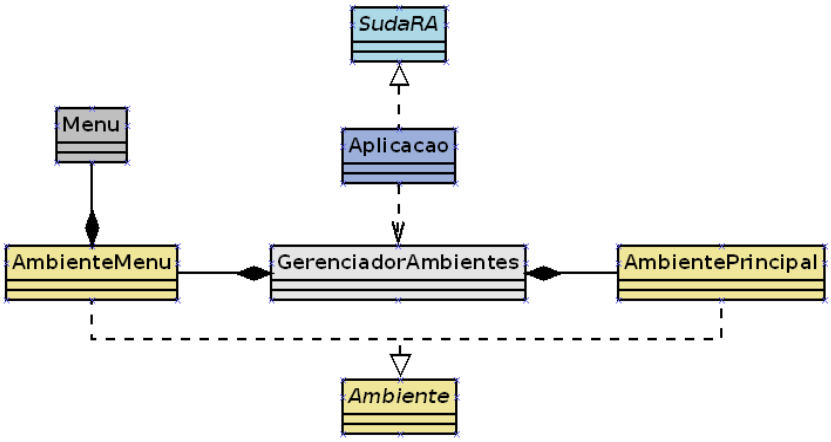


Figura 33. Arquitetura da aplicação

4.2.2. Estrutura de comunicação

O desenvolvimento da comunicação foi realizado com o suporte dos módulos “Servidor” e “Cliente” da SudaRA para a implementação das classes GerenciadorServidor e GerenciadorCliente respectivamente.

O GerenciadorServidor é usado quando a aplicação faz o papel de servidor, sua principal função é receber os dados de cada cliente e distribuí-los a todos os outros. Nesse caso, nenhum cliente pode ser destinatário dele mesmo (Figura 34).

```
para (clienteRemetente do primeiro_cliente até ultimo_cliente) faça
    para (clienteDestinatario do primeiro_cliente até ultimo_cliente) faça
        se (clienteRemetente != clienteDestinatario) entao
            servidor.enviarDados(clienteRemetente.dados, clienteDestinatario.endereco);
        fim-se
    fim-para
fim-para
```

Figura 34. Algoritmo do servidor para a distribuição dos dados

O GerenciadorCliente é usado tanto na aplicação do servidor quanto na dos clientes, uma vez que ele recebe e envia dados para todos os clientes através do

servidor. A Figura 35 mostra um um exemplo sobre a transmissão de dados de um cliente para os demais participantes.

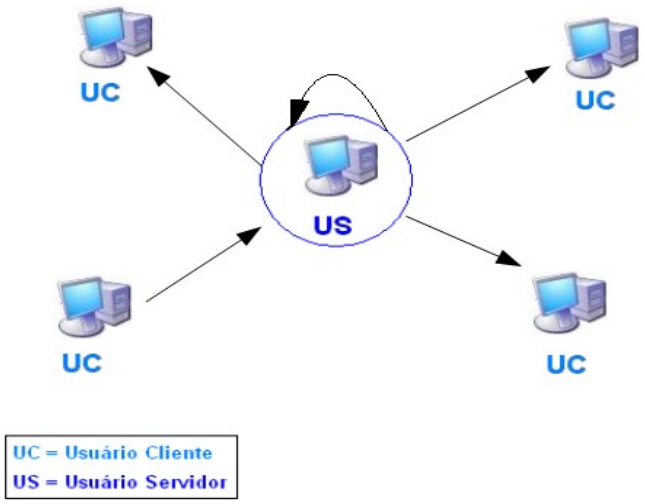


Figura 35. Exemplo de transmissão dos dados de um usuário para os demais

Os dados que trafegam entre os computadores são representações da posição, rotação e escala dos modelos que são controlados por cada cliente, fazendo com que estes sejam atualizados em tempo real. O comando de remoção feito pela borracha também é encaminhado para os outros clientes para que a remoção do modelo aconteça no cenário de cada usuário participante.

O protocolo utilizado foi o UDP (*User Datagram Protocol*) que não requer uma conexão entre o cliente e o servidor, assim o servidor pode atender a vários clientes com apenas um *socket*. Mesmo correndo o risco de perda de pacotes, o UDP é o mais indicado para este propósito, pois a aplicação, neste caso, faz uso de tráfego de dados em tempo real.

O módulo responsável pela comunicação externa está ligado diretamente com a execução do ambiente de criação, para que a mudança do cenário de um cliente reflita no cenário de todos os outros. Em um cliente conectado a um servidor remoto, o programa divide-se em 3 linhas de execução: uma *thread* primária e duas secundárias.

A *thread* primária executa as funcionalidades primordiais para o gerenciamento dos ambientes, dos eventos e de toda interação do usuário com a ferramenta. Está diretamente ligada ao ciclo de execução da SudaRA (seção 3.2.3.).

As *threads* secundárias estão divididas entre o envio das informações do cenário atual de um usuário para o servidor, e o recebimento dos dados do cenário dos clientes remotos.

Já para um usuário que também atua como servidor, 3 novas *threads* são criadas além das mencionadas. Uma funciona enviando os comandos de remoção de modelos e é chamada apenas quando um cliente usa a borracha, enquanto as

outras duas realizam o envio e o recebimento de dados fazendo a distribuição dos mesmos aos seus devidos destinatários.

O relacionamento da execução principal com as *threads* de comunicação se dá através de um sistema produtor/consumidor onde a *thread* primária consome as informações sobre os modelos adicionados por clientes remotos.

A Figura 36 apresenta uma visão geral sobre a relação do ambiente de criação (AmbientePrincipal) com o módulo de comunicação (36.a) e a atuação dos mesmos em tempo de execução (36.b) para os usuários cliente e servidor.

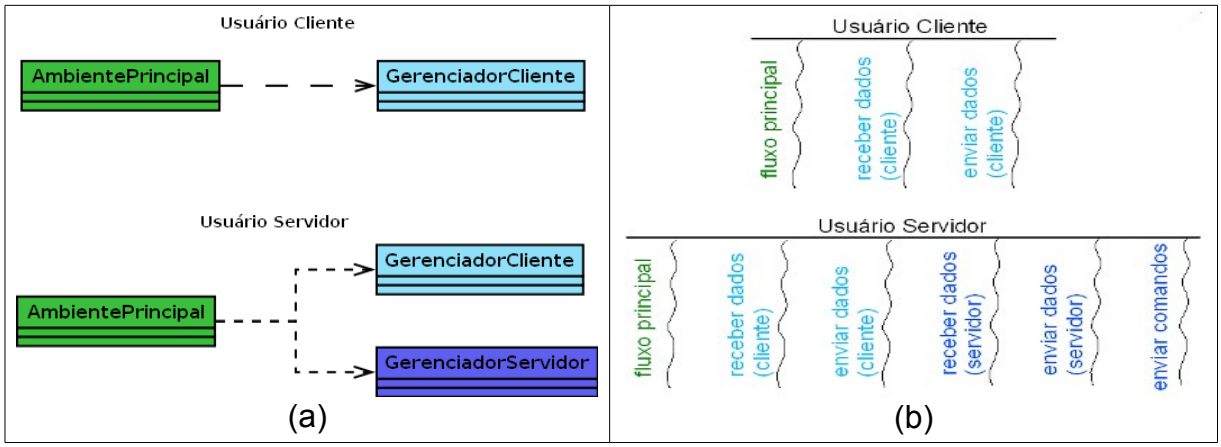


Figura 36. Relação entre o ambiente principal e o módulo de comunicação (a) e seus fluxos de execução (b)

4.3. Limitações e Dificuldades Encontradas

Para esta primeira etapa, o desenvolvimento do projeto foi limitado a utilizar objetos previamente modelados, ou seja, a criação de modelos a partir da manipulação de vértices, arestas e faces por enquanto está fora do escopo.

Na questão da configuração de hardware, por se tratar de uma aplicação que faz alto uso de processamento gráfico, há uma limitação para computadores com padrões técnicos não-atuais. Assim, é exigido no mínimo uma placa de vídeo com suporte a *shader*, além de um dispositivo de captura de vídeo. Os testes realizados foram baseados em um computador Intel Celeron 3Ghz, 1.5Gb de memória RAM, placa de vídeo ATI Radeon X1650 com 512Mb e uma webcam Microsoft LifeCam VX-2000.

Entre as dificuldades encontradas, destaca-se a implementação da manipulação do modelo controlado pelo marcador de controle principal, uma vez que a posição e a orientação deste objeto depende das transformações do marcador de controle e do cenário, que são obtidas através de alguns cálculos geométricos, não-triviais, envolvendo as suas matrizes de transformação. Esta dificuldade restringiu a movimentação do modelo somente aos eixos X e Y e sua rotação apenas ao eixo Z.

Capítulo 5
Conclusão e Trabalhos Futuros

É bastante notável o crescimento das ramificações de pesquisa em Realidade Aumentada. Assim como em qualquer outra área, é possível também utilizar esta tecnologia para o desenvolvimento de protótipos de ambientes físicos.

Com os recursos de RA, o software proposto trouxe uma maior liberdade de interação com os objetos do cenário e com a câmera de visualização, através dos 3 eixos de movimentação. Diferentemente das ferramentas CAD que são limitadas a uma interface 2d, e consequentemente há um trabalho a mais para posicionar a câmera usando dispositivos de entrada tradicionais.

A pesquisa também mostrou êxito na implementação da colaboração entre usuários em computadores distintos atuando sobre o mesmo projeto, possibilitando a criação em conjunto sem a necessidade de todos os projetistas estarem fisicamente presentes.

Como recursos para o experimento, foram usados modelos de casas e prédios, entre outras representações de ambiente urbano. No entanto, as aplicações podem estender-se a outros projetos de ambientes, como parques, casas, restaurantes, etc, bastando apenas o projetista possuir os objetos 3d necessários ao seu protótipo.

Vale ressaltar a importância da implementação do *framework* SudaRA que forneceu uma excelente estrutura para a construção deste protótipo de software, trazendo facilidades para a importação dos modelos 3d, rede, monitoramento dos marcadores, entre outras.

Porém, dada a variedade dos projetos, assim como a necessidade da criação de protótipos de ambientes físicos com mais fidelidade, melhorias como implementação de física, animação, iluminação, sombra, entre outros efeitos serão levados em consideração para melhorias futuras. A princípio, as limitações em relação à manipulação dos objetos em determinados eixos citadas na seção 4.3. serão priorizadas para a continuação deste trabalho.

Bibliografia

AUKSTAKALNIS,S. e BLATNER,D. **Silicon Mirage: The Art and Science of Virtual Reality**, Peatchpit Press, Berkeley, CA, 1992.

AZUMA, R. T. **A Survey of Augmented Reality**. In Presence: Teleoperators and Virtual Environments. Agosto de 1997. pp. 335-385.

BILLINGHURST, M. e KATO, H. **Collaborative Augmented Reality**. Communications of the ACM. Julho de 2002. pp 64-70.

BLENDER FOUNDATION. **Blender**. 2009 .http://www.blender.org> acesso em abril de 2010).

COUMANS, E. **Bullet Physics Library**. Versão 2.76. 2010. http://bulletphysics.org/wordpress (acesso em maio de 2010).

CRUZ-NEIRA, C., D. J., DEFANTI, T. A. **Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE**. ACM Computer Graphics. 1993. vol. 27, n. 2, pp 135-142.

CUNHA, C. H. B., **SudaRA (Suporte ao Desenvolvimento de Aplicações em Realidade Aumentada)**. 2010. http://sourceforge.net/projects/sudara (acesso em maio de 2010).

CUNHA, C. H. B. e FERNANDES, S. M. M. **Prototipação de Ambientes Físicos com Realidade Aumentada**. XII Brazilian Symposium on Virtual and Augmented Reality. Natal-RN: SBC. 2010.

DO, T. V., LEE, J. W. **3DARModeler: a 3D Modeling System in Augmented Reality Environment**, International Journal of Computer Systems Science and Engineering. 2008. v. 4, n. 2, pp. 145-154.

FREE SOFTWARE FOUNDATION, Inc. **GPL (General Public License)**. 2007. http://www.gnu.org/licenses/gpl.html (acesso em junho de 2010).

GOOGLE. **Google SketchUp**. 2010. http://sketchup.google.com (acesso em abril de 2010).

HEIDELBERGER, B. **Cal3d (Character Animation Library)**. 2006. http://home.gna.org/cal3d (acesso em maio de 2010).

HITLABNZ. **osgART (OpenSceneGraph for ARToolKit)**. 2006.http://www.artoolworks.com/community/osgart/index.html (acesso em maio de 2010).

IAGOWS. **Modelo da Escola Politécnica de PE - POLI - Bloco A. 2008**. http://sketchup.google.com/3dwarehouse/details?

mid=de8b5985efee81073fc6033c5bf4d7a2&ct=mdrm (acesso em junho de 2010).

INTEL RESEARCH. **OpenCV (Open Source Computer Vision Library)**. Versão 2.1.0. 2010. http://sourceforge.net/projects/opencvlibrary (acesso em junho de 2010).

KATO, H. **ARToolKit Documentation - Developng your First Application, Part 2**. http://www.hitl.washington.edu/artoolkit/documentation/devmulti.htm (acesso em junho de 2010).

KATO, H. e BILLINGHURST. M. **ARToolKit versão 2.72.1**. 2007. http://www.hitl.washington.edu/artoolkit (acesso em maio de 2010).

KIRNER, C. e PINHO M. S., **Introdução à Realidade Virtual**. Mini-curso do 1º Workshop de Realidade Virtual. São Carlos-SP. Novembro de 1997. p. 32-35.

LANTINGA, S. **SDL (Simple DirectMedia Layer)**. 2007. http://www.libsdl.org (acesso em maio de 2010).

LIDAL, M. E., LANGELAND, T., GIERTSEN, C., GRIMSGRAARD, J. e HELLAND, R. **A Decade of Increased Oil Recovery in Virtual Reality**. IEEE Computer Graphics and Applications. Novembro/Desembro de 2007. pp. 94-97.

LOKI SOFTWARE. **OpenAL (Open Audio Library)**. Versão 1.1. 2009. http://connect.creativelabs.com/openal/default.aspx (acesso em junho de 2010).

MACHADO, L. S. **Dispositivos Não-Convencionais para Interação e Imersão em Realidade Virtual e Aumentada**. XII Symposium of Virtual and Augmented Reality. p 23-33.

MICROSOFT CORP. **Direct3d**. http://msdn.microsoft.com/pt-br/directx/default(en-us).aspx (acesso em maio de 2010).

MICROSOFT CORP. **Microsoft Visual C++ 2008 Express Edition**. 2009. http://www.microsoft.com/express/Downloads/#2008-Visual-CPP (acesso em abril de 2010).

NATTERER, M. et al. **GIMP (The GNU Image Manipulation Program)**. 2009. http://www.gimp.org (acesso em abril de 2010).

NAVAB, N., TRAUB, J., SIELHORST, T., FEUERSTEIN, M. and BICHLMEIER, C. **Action- and Workflow-Driven Augmented Reality for Computer-Aided Medical Procedures**, IEEE Computer Graphics and Applications. Setember/October, 2007. pp. 10-14.

NVIDIA Corporation. **NVIDIA PhysX**. 2010. http://www.nvidia.com/object/physx_new.html (acesso em maio de 2010).

OSFIELD, R. and BURNS, D. **OpenSceneGraph**. 1998. http://www.openscenegraph.org (acesso em maio de 2010).

PIEKARSKI, W. **Interactive 3d Modelling in Outdoor Augmented Reality Worlds**. Research Thesis fot the Degree of Doctor of Philosophy. Adelaide, South Australia. Fevereiro de 2004.

SALGUEIRO, V. N. **Editor de Cenas 3D Baseado em Software Livre**. Trabalho de Conclusão de Curso - UPE. 2009.

SILICON GRAPHICS. **OpenGL**. http://www.opengl.org (acesso em maio de 2010).

SILVA, R. J. M., RAPOSO, A. B. e GATTASS, M. **Grafo de Cena e Realidade Virtual**. In Monografias em Ciência da Computação, n.11/04, Editado por C. J. P. de Lucena, Departamento de Informática, PUC-Rio, 2004.

SMITH, R. **ODE (Open Dynamics Engine)**. Versão 0.11.1. 2009. http://www.ode.org (acesso em maio de 2010).

STROUSTRUP, B. **The C++ Programming Language**. Addison-Wesley, Special edition, 1997, pp 480-489.

TAVARES, A. C. M. **Aplicação de um Ambiente Virtual Colaborativo Utilizando Realidade Aumentada**. Trabalho de Conclusão de Curso - UPE. 2009.

TEICHRIEB, V. e FIGUEIREDO, L. **Interação Natural**. XII Symposium of Virtual and Augmented Reality. Natal-RN: SBC. 2010. p 9-22.

TORI, R. e KIRNER, C. **Fundamentos de Realidade Virtual**. VII Symposium on Virtual Reality. Belém-PA: SBC. 2006. p. 2-21.

TORUS KNOT SOFTWARE Ltd. **Ogre3d (Object-Oriented Graphics Rendering Engine)**. 2000. http://www.ogre3d.org (acesso em maio de 2010).

ZORZAL, E. R. ; KIRNER, Claudio ; CARDOSO, Alexandre ; LAMOUNIER JÚNIOR, Edgard. **Viabilizando o Desenvolvimento de Jogos Espaciais com Realidade Aumentada**. In: SEMISH - XXXIII Seminário Integrado de Software e Hardware. 2006, Campo Grande - MS.

Apêndice A
Exemplo de um Cubo Sobre o Marcador Usando ARToolKit

Código de exemplo para a implementação de um cubo associado a um marcador utilizando ARToolKit.

```
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>

char    *vconf = "Data\\WDM_camera_flipV.xml";
int     xsize, ysize;
int     thresh = 100;
int     count = 0;

char           *cparam_name = "Data/camera_para.dat";
ARParam        cparam;

char           *patt_name = "Data/patt.hiro";
int            patt_id;
double         patt_width = 80.0;
double         patt_center[2] = {0.0, 0.0};
double         patt_trans[3][4];

int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   init();

   arVideoCapStart();
   argMainLoop( NULL, NULL, mainLoop );
   return (0);
}

static void init( void )
{
   ARParam  wparam;

   // abre o arquivo de configuração da câmera
   arVideoOpen( vconf );
   // encontra o tamanho da janela
   arVideoInqSize(&xsize, &ysize) < 0 );
```

```
    // configrura os paremtros iniciais da camera
    arParamLoad(cparam_name, 1, &wparam);
    arParamChangeSize( &wparam, xsize, ysize, &cparam );
    arInitCparam( &cparam );
    arParamDisp( &cparam );

        // carrega o arquivo referente ao marcador a ser utilizado
    patt_id = arLoadPatt(patt_name);

    // inicializa a janela
    argInit( &cparam, 1.0, 0, 0, 0, 0 );
}

static void mainLoop(void)
{
    ARUint8         *dataPtr;
    ARMarkerInfo    *marker_info;
    int             marker_num;
    int             j, k;

    // captura o quadro atual
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL )
    {
        arUtilSleep(2);
        return;
    }
    if( count == 0 ) arUtilTimerReset();
    count++;
    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );

    // detecta os marcadores neste quadro
    if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 )
    {
        cleanup();
        exit(0);
    }

    arVideoCapNext();

    // verifica a visibilidade
    k = -1;
    for( j = 0; j < marker_num; j++ )
    {
        if( patt_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 )
    {
        argSwapBuffers();
        return;
    }

        // obtém a transformação entre o marcador e a câmera
    arGetTransMat(&marker_info[k], patt_center, patt_width, patt_trans);
    draw();
    argSwapBuffers();
}

// finaliza os recursos
static void cleanup(void)
{
    arVideoCapStop();
    arVideoClose();
    argCleanup();
}

static void draw( void )
{
    double    gl_para[16];

    argDrawMode3D();
    argDraw3dCamera( 0, 0 );
    glClearDepth( 1.0 );
    glClear(GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);

    argConvGlpara(patt_trans, gl_para);
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd( gl_para );

    glMatrixMode(GL_MODELVIEW);

    // desenho do cubo
    glutSolidCube(50.0);

    glDisable( GL_DEPTH_TEST );

}
```