PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

REALIDADE AUMENTADA PARA INFORMAÇÕES GEOGRÁFICAS

por

Emanuel Motta Grohs
Patrick Renan Bernardes Maestri

Trabalho de Conclusão II
Curso de Bacharelado em Ciência da Computação

Prof. MSc. Márcio Serolli Pinho
Orientador

Porto Alegre, Junho de 2002

Agradecimentos

Gostaríamos de agradecer ao prof. Régis Alexandre Lahm da Faculdade de Filosofia e Ciências Humanas da PUCRS pelo apoio nas questões relativas à Geografia.

Agradecemos também ao prof. Jorge Alberto Villwock, diretor do Instituto do Meio Ambiente pelo empréstimo do equipamento GPS.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Hoje em dia, o alto grau de complexidade imposta por tarefas em diversas áreas da ciência está exigindo mais do homem do que seus sentidos naturais podem lhe oferecer. O emprego de técnicas de Realidade Aumentada pode auxiliar na melhora da percepção, interação e conseqüentemente a produtividade no dia a dia. Baseando-se nesta idéia, este trabalho tem como objetivo criar uma aplicação que forneça ao usuário informações a respeito de pontos específicos de uma cidade como prédios, igrejas, monumentos e outras construções em geral, fazendo uso da Realidade Aumentada para adicionar os dados informativos.

Através de uma câmera posicionada sobre a cabeça do usuário, as imagens são capturadas e manipuladas para adicionar as informações sobre as construções escolhidas. Depois de capturadas, as imagens são enviadas para um *notebook*, que o usuário estará portando, o qual fará a inserção dos dados caso necessário. Quando o usuário estiver próximo e olhando diretamente para um desses monumentos (previamente cadastrados no sistema), os dados sobre o mesmo serão exibidos em um HMD (*Head Mounted Display*). Este dispositivo é capaz de informar a orientação visual de um usuário através de sensores localizados junto a ele.

Para que se possa determinar a distância em relação aos objetos reais, faz-se uso de um equipamento de GPS (*Global Positioning System*) que também estará conectado ao *laptop* informando-o constantemente a posição do usuário. Ele também é utilizado no pré-cadastramento dos objetos sobre os quais serão exibidas as informações. Dessa forma será possível determinar quando as informações deverão ser exibidas ao usuário.

Palavras-chave: Realidade Aumentada, Sistema de Posicionamento Global (GPS), Head Mounted Display (HMD)

Abstract

Title: “Augmented Reality to geographical information”

Nowadays, the high complexity level imposed by tasks in diverse science areas requires more of man than his own native senses are able to offer. The techniques of augmented reality may help the increase of perception, interaction and, consequently, the productivity, day by day. Based on this idea, the goal of this work is to create an application that supply the user with information of specific points of a city as buildings, churches, monuments and other constructions in general, using the Augmented Reality to add the informative data.

Through a video camera located over the user´s head, the images will be captured and manipulated to add the information about the chosen constructions. After captured, the images are sent to a laptop that an user will be carrying, which will insert the data, if necessary. If the user is near and looking straight to one of those monuments (already registered in the system), its data will be exhibited in a HMD. This device is capable to inform the visual orientation of the user through head sensors attached to it.

To obtain the real distance between the real objects and the user, an equipment of GPS will also be connected to the laptop informing constantly the position of the user. It will also help in the registering process of the objects whose data will be exhibited. This will make it possible to determine when the information will be shown to the user.

Keywords: Augmented Reality, Global Positioning System (GPS), Head Mounted Display.

1. Introdução

Sistemas de realidade virtual desenvolvidos nos últimos anos trouxeram um ótimo realismo a ambientes totalmente artificiais sintetizados por computadores. Entretanto, sabe-se que a "realidade" implementada dessa forma está longe de ser comparada ao mundo real ao qual se conhece. Em certas aplicações, o uso de objetos e cenários reais pode ser de grande valia no processo interativo com um sistema computacional. Em face disso, a mescla de objetos reais e imagens virtuais surgiu como uma alternativa interessante a fim de se obter maior realismo em um cenário antes totalmente virtual. Esta técnica está sendo cada vez mais aplicada e é conhecida como Realidade Aumentada (RA).

É um tanto difícil precisar sua origem, entretanto sabe-se que os primeiros trabalhos na área, propostos por Sutherland e Sproull, datam da década de 60 [SCH01]. Nesta época surgiram os primeiros *see-through Head Mounted Displays* (stHMDs), “óculos” capazes de introduzir imagens de objetos virtuais em uma cena real vista pelo usuário. Nos anos seguintes, os avanços tecnológicos permitiram o desenvolvimento de novos dispositivos, cada vez mais leves e menores, motivando outras áreas a criarem aplicações contidas neste contexto. Hoje existem aplicações desta técnica em diversos campos da ciência, dentre elas, medicina, manutenção de *hardware*, construção civil e robótica.

A Figura 1.1 enquadra a RA em uma escala que vai desde o mundo real propriamente dito até os ambientes puramente virtuais sintetizados por computador.

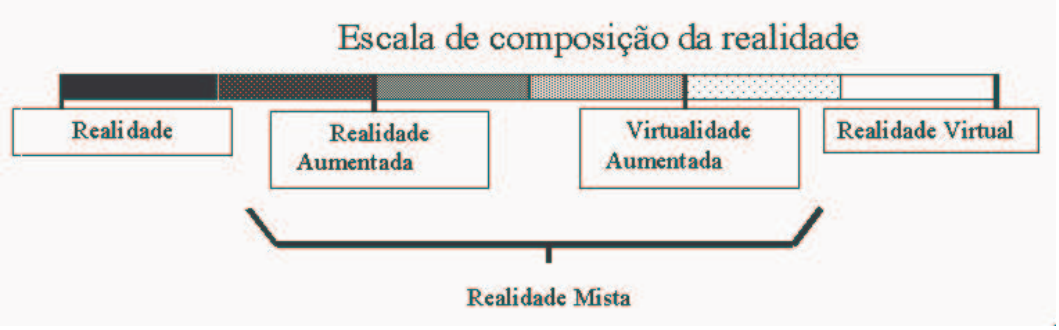

Figura 1.1 – Escala da realidade mista [DRA96]

Pode-se entender realidade aumentada (também chamada de realidade realçada) como uma combinação, em tempo real, da cena real vista por uma pessoa com a cena virtual gerada por computador. Pode-se defini-la ainda como um aumento da percepção humana através da adição de informação não detectada diretamente pelos sentidos naturais. Nela, objetos virtuais são inseridos e posicionados na cena real, em função da posição relativa do observador. Idealmente, a coexistência destes dois ambientes (real e virtual) deve ser harmônica a ponto de um usuário não distinguí-los. A Figura 1.2 mostra uma aplicação que combina imagens virtuais com objetos reais para auxiliar médicos em microcirurgias.

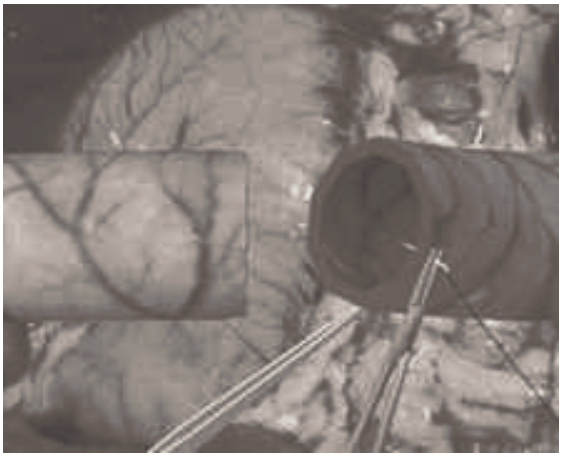

Figura 1.2 – Combinação de imagens virtuais no auxílio de cirurgias [SAB99]

O paradigma da RA, em seu estado da arte, pode introduzir benefícios de grande importância às atividades humanas. Sua utilização pode auxiliar no aumento da percepção, na melhoria da interação e, conseqüentemente, aumentar a produtividade na realização de tarefas do mundo real.

Uma área onde a RA pode ser de grande valia é no Turismo, visto que o usuário pode obter informações a respeito de um determinado local através da adição de informações virtuais, ou até mesmo sobrepor imagens virtuais (imagens tridimensionais) em construções que foram destruídas devido à influência da natureza (ventos, inundações, terremotos) ou a guerras acontecidas no passado, podendo assim, obter maior compreensão de como elas eram e mais informações a respeito das mesmas (com o uso de informações textuais).

Baseado nessa idéia, este trabalho descreve um aplicativo que permite ao usuário visualizar informações virtuais a respeito de pontos específicos de uma cidade, tais como prédios, igrejas, monumentos e outras construções, fazendo uso das técnicas de RA.

A utilização dessa ferramenta faz uso de quatro equipamentos: um receptor GPS (*Global Positioning System*), uma câmera digital, um *notebook* e um óculos de realidade virtual (HMD). O *notebook* serve para fazer todo o processamento de combinação das imagens (reais e virtuais), bem como para fazer a tradução dos dados obtidos do receptor GPS sobre a posição do usuário e das posições das construções das quais se deseja obter informações. Ele serve ainda para enviar a imagem resultante para o HMD, responsável pela exibição da mesma. Com o HMD também é possível saber para onde o usuário está olhando em um certo momento, já que ele possui um dispositivo de rastreamento de giro.

A utilização da câmera torna-se necessária para a obtenção das imagens reais que são combinadas com as imagens virtuais. Essa informação virtual somente será exibida quando o usuário estiver próximo e olhando diretamente para algum desses monumentos (que devem estar precadastrados no sistema).

Por fim, o uso de um receptor GPS torna-se necessário tanto para obter a posição do usuário, em tempo real, quanto para cadastrar as construções desejadas. Tendo a posição do usuário em relação a uma determinada edificação, pode-se saber quando deve ser exibida a imagem virtual correspondente a esta edificação.

As aplicações que estão rodando no *notebook* estão divididas em dois módulos separados. O módulo *GPS Register*, encarregado de gerar uma base de dados a respeito de todas as construções, cadastrando-as no sistema, e o módulo *GPS Viewer*, que é responsável pela execução do sistema e pelo carregamento da base de dados gerada pelo *GPS Register*.

1.1. Organização do Texto

No capítulo 2 são descritas diversas aplicações nas quais a RA pode auxiliar, por exemplo, em cirurgias ou em manutenção de aparelhos eletrodomésticos. No capítulo 3 são abordados os tipos de RA que podem ser desenvolvidos e suas limitações de acordo com o contexto na qual está inserida. No capítulo 4 está uma descrição detalhada a respeito do que trata esse trabalho, descrevendo o que foi desenvolvido e os recursos utilizados para tal.

O capítulo 5 contém informações a respeito dos problemas existentes na RA, descrevendo cada um deles e mostrando os motivos pelo qual eles devem ser tratados. O capítulo 6 faz referência sobre como obter a localização de uma pessoa no globo terrestre através do uso de um receptor GPS (*Global Positioning System*). O capítulo 7 descreve o protocolo de comunicação entre um receptor GPS e um computador, comentando sobre cada uma de suas camadas e as configurações que devem ser definidas no aparelho de GPS.

O capítulo 8 mostra como são feitas a captura e integração das imagens em tempo real, descrevendo como fazer uso das bibliotecas utilizadas. Nele também consta como foi feito o tratamento da oclusão e os ajustes necessários para que a câmera possa simular o funcionamento da visão. Por fim, o capítulo 9 contém informações de como foi desenvolvido a captura, manipulação e conversão dos dados enviados pelo aparelho de GPS ao *notebook*.

2. Aplicações

Técnicas de RA podem ser aplicadas em diversas áreas. A seguir, são apresentadas algumas delas.

2.1. Medicina

Existem alguns processos cirúrgicos que são feitos através de micro-incisões no corpo humano e que podem vir a prejudicar a vida de um paciente se não forem bem sucedidos. Apesar de beneficiadas pelo uso de sensores não invasivos como aparelhos de ressonância magnética, tomografia computadorizada ou aparelhos de ultra-som, estas cirurgias podem ainda apresentar um alto risco ao paciente.

Ao aplicar as técnicas de RA, o médico pode obter uma melhor visão do corpo humano, pois suas partes serão representadas através de modelos tridimensionais. Para geração destes modelos utilizam-se os aparelhos citados acima. A idéia é melhorar a visualização evitando que sejam feitas incisões no corpo e reduzindo o risco de prejudicar o paciente.

Na Universidade do Colorado do Norte (UNC) em Chapel Hill, um grupo de pesquisadores tem realizado mapeamentos do útero em mulheres grávidas através de sensores de ultra-som, gerando a representação 3D do feto, visualizando-o através de um HMD [BAJ92]. Seu objetivo é ajudar um médico no processo de visualização do feto movendo-se no útero (Figura 2.1). Posteriormente, esforços foram concentrados na costura de tecidos após a retirada de algum tumor no seio (Figura 2.2) [STA96b].

Outra aplicação consiste no treinamento de iniciantes na realização destas cirurgias, mostrando os passos que eles devem seguir, auxiliando-os na execução de suas tarefas.

Figura 2.1 – Representação 3D de um feto no útero [BAJ92]

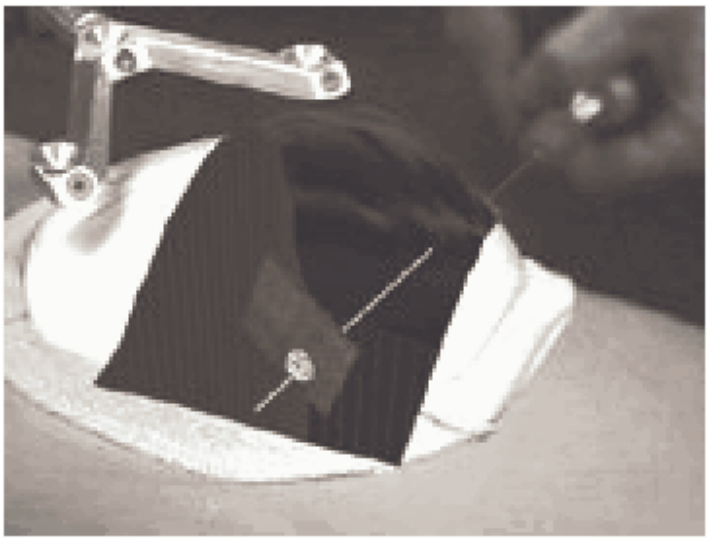

Figura 2.2 – Modelo de uma biopse de um tumor no seio [STA96b]

2.2. Processos de Construção e Manutenção

Um técnico realizando a manutenção de algum aparelho de alta complexidade pode ser beneficiado com a utilização de RA. Ao invés de se basear em longos e cansativos manuais, a adição de informação virtual sobre o equipamento real torna mais óbvia a realização de sua tarefa. Neste caso, o objetivo é facilitar os mecanismos de montagem e manutenção de aparelhos em geral, pois é muito mais intuitivo para um ser humano a compreensão através de

uma representação estereoscópica do que simplesmente uma leitura em um livro contendo ilustrações.

Pesquisadores do grupo *Steve Feiner*, da Universidade Colúmbia desenvolveram uma aplicação para manutenção de uma impressora laser. Na Figura 2.3 temos uma visão do usuário usando esta aplicação, que, neste caso, está vendo como remover a bandeja de papéis [FEI93a].

Este projeto utilizou rastreadores de posição para obter informações a fim de determinar a posição de uma parte desta impressora, na qual era necessário realizar alguma operação, bem como saber se existe ou não algum objeto obstruindo o caminho de uma determinada ação que, caso já estivesse concluída, não seria mais necessário mostrar o modelo virtual.

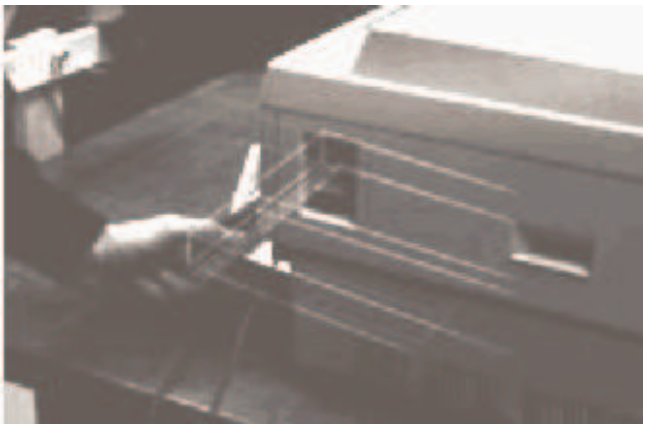

Figura 2.3 – Visão de um usuário demonstrando como remover a bandeja da impressora [FEI93a]

Outro exemplo de aplicação foi desenvolvido pelo grupo da Boeing [JAN93]. O projeto consiste na construção de uma ferramenta para auxiliar um mecânico na conexão de condutores elétricos que formam parte do sistema elétrico de aviões, economizando espaço e reduzindo custos (Figura 2.4).

Este sistema também ajuda os profissionais que não dominam completamente o funcionamento de uma determinada parte do equipamento ao qual se está trabalhando, reduzindo a margem de erro e aumentando a produtividade.

Figura 2.4 – Demonstração do protótipo de conexão dos condutores [JAN93]

2.3. Anotação e Visualização

Quando se trabalha na manutenção de um material ou deseja-se obter informação a respeito de algum componente do motor, por exemplo, pode-se utilizar técnica de apontamento sobre RA. Ela consiste na geração instantânea de uma caixa de diálogo (*pop-up*) contendo informações da parte apontada.

Um exemplo dessa aplicação foi desenvolvido na *European Computer-Industry Research Centre*, onde um usuário aponta para uma parte do objeto e o sistema indica o que está sendo apontado (Figura 2.5) [ROS95].

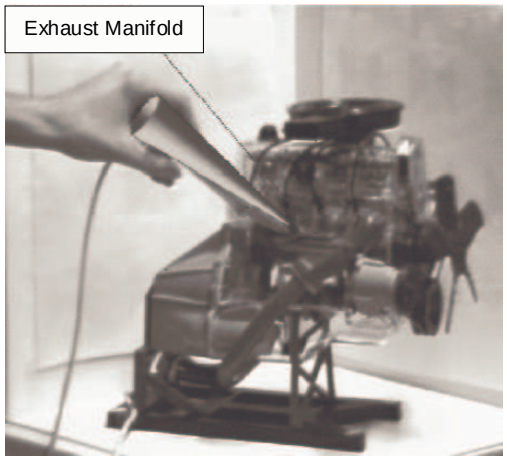

Figura 2.5 – Diálogo *pop-up* mostrando o nome da informação apontada [ROS95]

Outra aplicação semelhante foi criada por pesquisadores da Universidade da Colúmbia adotando a utilização de janelas informativas sobre

objetos específicos. Para funcionar corretamente, este *software* deve conhecer a posição inicial do objeto real, através de rastreamento prévio, e então posicionar o modelo virtual em relação a ele, podendo acompanhá-lo quando ele estiver em movimento (Figura 2.6) [FEI93b].

Na Figura 2.6 mostra a aplicação citada, onde há a adição de informações a respeito do usuário, do *notebook* e do teclado através do rastreamento de objetos reais.

Figura 2.6 – Janelas mostradas à frente de objetos reais específicos [FEI93b]

A RA pode auxiliar e facilitar também na visualização de determinadas tarefas. Por exemplo, ela pode dar a um arquiteto uma “Visão Raio-X” de suas construções, permitindo a visualização do sistema de canalização de água e a rede de fios elétricos. Utilizando esta idéia que pesquisadores da Universidade de Toronto desenvolveram um sistema chamado *Augmented Reality through Graphics Overlays on Stereovideo* (ARGOS) [DRA93a], que entre outras funções, serve para facilitar a compreensão de locais de difícil visualização. A

Figura 2.7 faz uso de RA para realçar contornos de um ambiente de baixa iluminação.

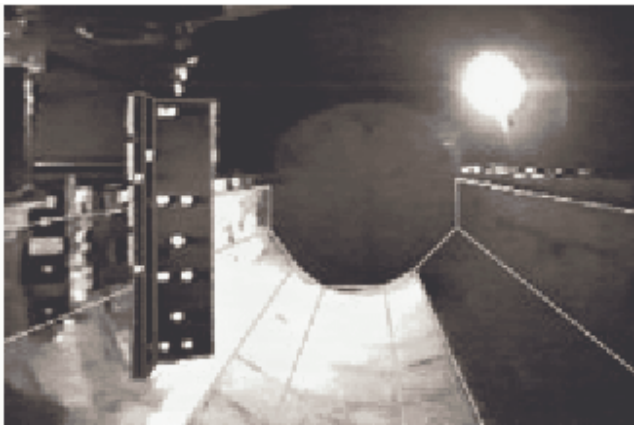

Figura 2.7 – Linhas virtuais facilitando a visualização [DRA93a]

2.4. Planejamento de caminho para robôs

Operar um robô, principalmente a uma grande distância, é uma tarefa que requer alguns cuidados. Atrasos gerados devido a problemas de comunicação prejudicariam o desempenho desta tarefa que requer tempo real a fim de não comprometer sua execução. Além disso, a execução dos movimentos do robô através de *joystick* pode ocasionar erros de precisão ao pegar um objeto e atrasos no cumprimento do serviço, requerendo do usuário grande concentração visual e coordenação motora.

Utilizando-se um objeto virtual sobreposto ao robô, poder-se-ia criar um plano para a realização dos movimentos do robô antes de executá-lo. Em outras palavras, para sua correta realização, o usuário montaria o plano, validá-lo-ia e simplesmente mandaria o robô executá-lo.

Desta forma, pode-se evitar atrasos devido à sua manipulação direta. Pensando nisso, o grupo que criou o sistema ARGOS (Figura 2.8), demonstrou que RA estereoscópica facilita e melhora a precisão para fazer o robô executar seu caminho.

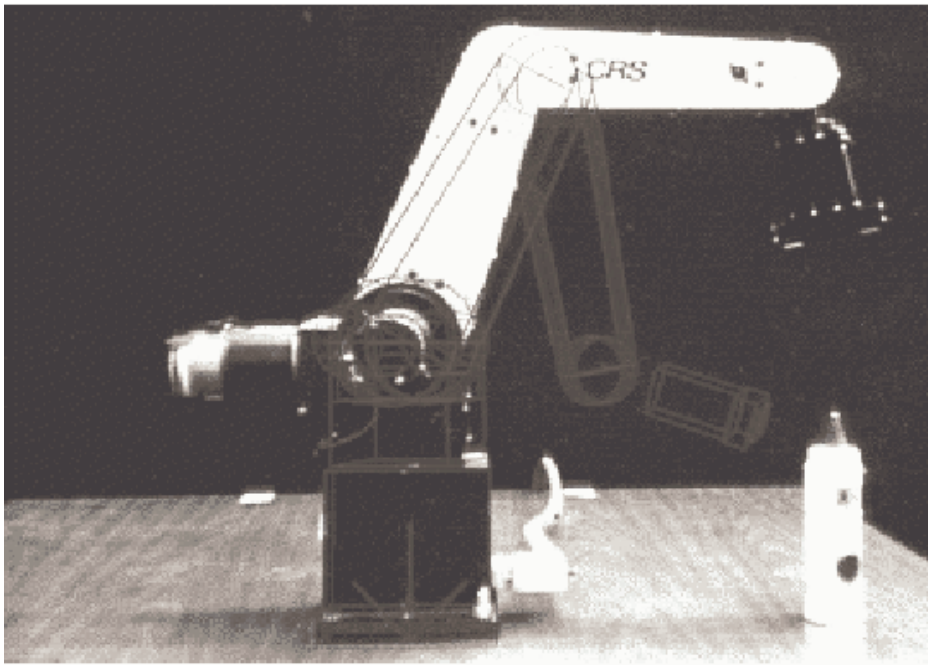

Figura 2.8 – Planejamento através da visualização um objeto virtual executado em tempo real. [KIM96]

2.5. Entretenimento

Na área da indústria cinematográfica, a RA está sendo usada como forma de baratear os custos dos filmes. Algumas cenas são previamente gravadas em salas que possuem uma cor de fundo. Depois de terminado este processo, as partes que continham essa cor são sobrepostas por um conjunto de imagens virtuais. A idéia é evitar a construção de todo o cenário para a gravação de filme, substituindo alguns objetos reais por seus equivalentes digitalizados.

Também existem diversas aplicações na área de jogos. Um grupo da universidade de Vienna montou um sistema onde mais de um usuário pode interagir durante a realização de uma partida de Mah-Jongg, um antigo e tradicional jogo chinês [SZA98]. Na Figura 2.9 está sendo realizada uma partida dele entre dois usuários. A idéia deste grupo é promover a realização de jogos em tempo real com multiusuários, além do entretenimento.

Nesta aplicação os usuários têm a liberdade de possuir seu próprio espaço privado. Privacidade é utilizada em jogos *multiplayer* para ocultar informações particulares de outros jogadores, ou seja, provém um espaço necessário para o usuário realizar suas estratégias.

Figura 2.9 – Ilustração mostrando o jogo Mah-jongg [SZA98]

2.6. Teleconferência e Tele-Imersão

Sistemas de teleconferências são promoções de reuniões entre pessoas que não se encontram em um mesmo local. Normalmente são realizadas apenas por intermédio de aparelhos telefônicos conectados a uma rede particular, porém tem-se um certo desconforto, pois um participante da teleconferência pode, algumas vezes, não ver outros participantes destas reuniões. Há, atualmente, várias abordagens tomadas em RA para tentar viabilizar este desconforto, dentre elas temos um grupo da Universidade de Washington que está desenvolvendo um projeto para conferências utilizando “computadores de vestir (*wearable*

computers)", na qual um usuário deste sistema pode ouvir e ver os outros participantes dela [BIL98].

O grupo da *Advanced Network & Service*, de Nova York, está implementando um sistema, chamado *National Tele-Imersion Initiative* [LAN98], para permitir que usuários que estão geograficamente distribuídos estivessem na mesma sala, todos reunidos virtualmente. Na Figura 2.10 temos 2 usuários interagindo no ambiente. Note que há imagens virtuais, como por exemplo, o fundo do cenário (Figura 2.10b). Neste sistema há também a manipulação de objetos virtuais como recurso didático a fim de melhorar o grau de compreensão dos outros participantes da conferência.

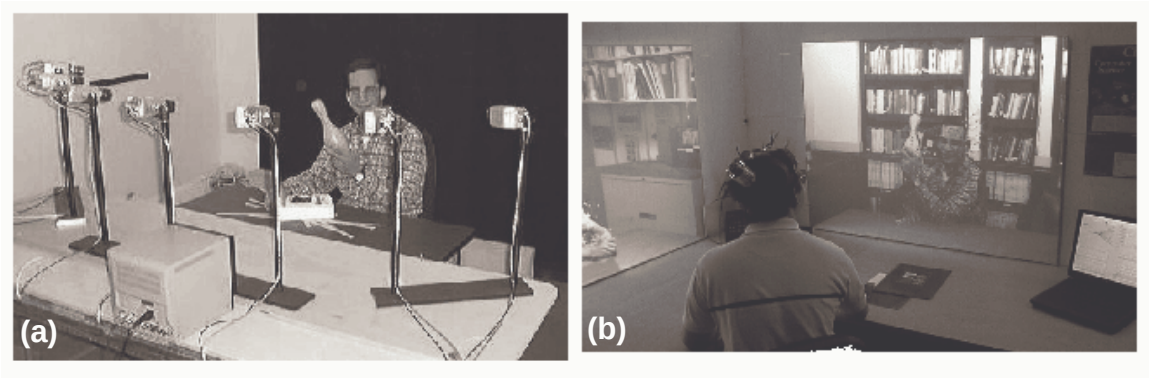

Figura 2.10 – (a)Usuário segurando um objeto real. (b) Outro participante ativo da teleconferência [LAN98]

2.7. Preservação de patrimônios históricos

Uma aplicação interessante de RA é o projeto *Augmented Reality-based Cultural Heritage On-site Guide* (*ARCHEOGUIDE*) [STR01a]. Nele é feita a sobreposição de edificações virtuais sobre as ruínas de um sítio arqueológico. Isso possibilita a sobreposição de objetos virtuais representando as construções como se fossem as originais, sobre as edificações destruídas, auxiliando aos visitantes e cientistas na melhor compreensão sobre o passado desses locais históricos, além de ajudar na educação cultural.

Esta aplicação utiliza uma arquitetura de cliente-servidor que possui três subsistemas: um servidor de dados (chamado *Site Information Server* (*SIS*)), unidades móveis (chamadas *Mobile Units* (*MU*)) e uma infraestrutura de rede. O SIS é um computador de alta qualidade responsável por armazenar todo o repositório de dados que serão utilizadas pelas MU's aplicando a RA para a adição de informações virtuais ao mundo real. Essa base de dados é composta por imagens bidimensionais, modelos tridimensionais, clipes de áudio e vídeo e textos a respeito dessas construções, sendo que elas estão organizadas numa hierarquia de acordo com a informação que representam. O SIS se comunica com as MU's via *WLAN* (*Wireless Local Area Network*).

As MU's são utilizadas pelos usuários que estão nos sítios arqueológicos e onde são utilizados *laptop*, *pen-tablet* e *palmtop computers* para rodar o sistema. Essas unidades fazem requisições para a base de dados do SIS baseados na posição do usuário e outros parâmetros calculados por um GPS (*Global Positioning System*), sendo que o sistema faz um cálculo para melhorar a precisão usando um sinal transmitido por um farol contendo um DGPS (*Differential GPS*) localizado em uma posição bem conhecida.

A Figura 2.11 mostra como foi feita a arquitetura do Archeoguide, que é composta pelos três subsistemas descritos anteriormente.

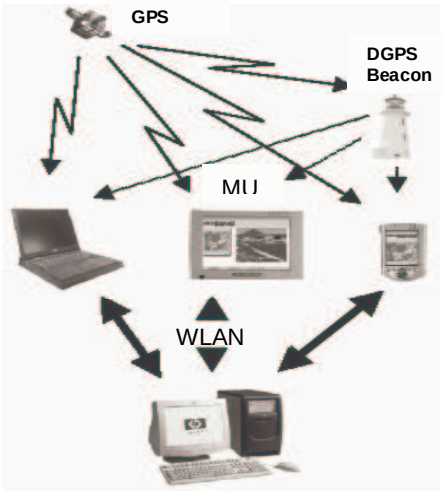

Figura 2.11 – Arquitetura de Comunicação do Archeoguide [STR01b].

O maior problema desse tipo de sistema é o desconforto devido ao excesso de aparelhos que devem ser carregados no corpo, pois os equipamentos de realidade virtual ainda são pesados apesar de já estarem sendo desenvolvidos outros dispositivos para evitar ou reduzir esses problemas. Contudo, o que tem de mais interessante nessa aplicação é no fato de se poderem preservar os patrimônios históricos, mantendo-os como se encontram originalmente e apenas adicionando informações virtuais sem precisar alterar o antigo.

Na Figura 2.12 mostra como essa aplicação funciona. Em (a) pode-se visualizar a construção real que já se encontra tombada. Em (b) tem-se a sobreposição da edificação grega como originalmente ela havia sido construída [STR01a].

Figura 2.12 - (a) construção real tombada pelo tempo. (b) resultado aplicando-se a RA nessas edificações [STR01a]

3. Tipos de Realidade Aumentada

Existem três métodos de construção de sistemas de realidade aumentada: RA Ótica, RA por vídeo e RA por monitores. Cada um possui suas vantagens e desvantagens, mas todos têm o mesmo objetivo: combinar o ambiente real com o virtual. Nas seções a seguir serão descritos estes três tipos de realidade aumentada, abordando como são configurados e seus pontos positivos e negativos.

3.1. Sistemas de RA óticos

Em sistemas de RA óticos o usuário tem a possibilidade de visualizar o ambiente real diretamente (a olho nu), sendo apenas geradas as imagens virtuais que serão visualizadas através de um dispositivo (óculos de realidade virtual) chamado *see-through HMD* (stHMD). Este dispositivo é um óculos semitransparente ou transparente que além de exibir as imagens virtuais, permite que o usuário enxergue *através* de suas lentes e assim veja também o mundo real.

A Figura 3.1 mostra o funcionamento deste dispositivo, que é composto por duas partes: um **gerador de cenas**, que receberá esses dados e fará os cálculos necessários, gerando imagens virtuais, enviando aos **monitores** do óculos que, por sua vez, reproduzirão as imagens virtuais sobrepostas ao mundo real visto através das lentes do HMD. Opcionalmente o sistema pode possuir um rastreador que captura os movimentos da cabeça do usuário.

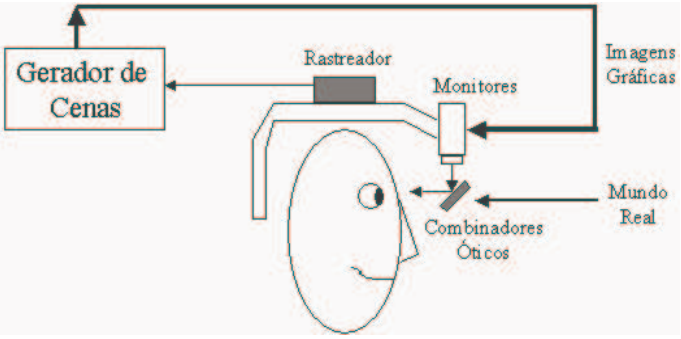

Figura 3.1 – Esquema da RA ótica [AZU97]

Uma limitação imposta por aplicações que utilizam dispositivos com combinadores óticos é o grau de luminosidade no cenário real. A maioria dos combinadores reduz a quantidade de luz vinda do mundo real. Além disso, em ambientes com muita iluminação, o grau de qualidade das imagens virtuais fica prejudicado, fazendo com que elas pareçam semitransparentes.

Apesar desses problemas, há duas grandes vantagens na utilização de sistemas de RA ótica que são o menor tempo de processamento e a maior resolução das imagens reais. O grau de processamento é menor que em outros sistemas de RA, pois somente é necessário consumir tempo de processamento com a geração das imagens virtuais para serem visualizadas, já que o mundo real é visto a olho nu. Quanto ao grau de resolução das imagens reais, este é o maior possível, pois a imagem é captada diretamente pelo olho do usuário.

3.2. Sistemas de RA por vídeo

Nos sistemas de RA por vídeo, o usuário visualiza o mundo real através de uma ou duas câmeras. Nestes sistemas, a câmera passa a desempenhar a função dos olhos do usuário.

A montagem de um sistema de RA por vídeo é bastante semelhante ao citado na seção anterior, contudo que o mundo real é capturado por meio de uma câmera e retransmitido para um combinador de vídeo, que reúne as cenas virtuais vindas do gerador de cenas com as imagens do mundo real, combinando-as de tal forma que pareçam um único ambiente. Por fim esse resultado é enviado

aos monitores (do *Head Mounted Display* (HMD)) que irão reproduzí-las. A Figura 3.2 mostra um modelo de como deve ser desenvolvido um sistema de RA por vídeo.

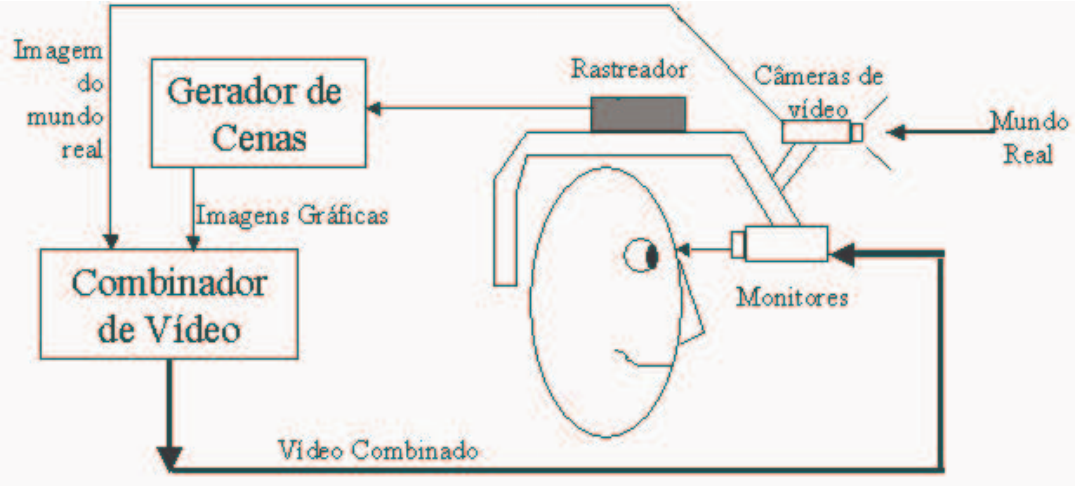

Figura 3.2 – Esquema de RA por vídeo [AZU97]

Para realização desta combinação, há várias técnicas, dentre elas o uso da *chroma-keying*, método ao qual se tem um fundo com uma cor específica, e tudo que contiver esta cor (na cena real) será substituído pelas imagens virtuais geradas por um computador. Algumas composições mais aperfeiçoadas utilizam a idéia de profundidade, onde cada *pixel* da imagem possui informação de sua profundidade facilitando a ordem de colocação dos objetos virtuais no cenário final. Com o emprego dessa metodologia tem-se a atualização do ambiente real e virtual de forma síncrona, visto que ambos os ambientes (real e virtual) serão processados por um único sistema de composição.

Uma das características interessantes na RA por vídeo é a possibilidade de ampliar ou reduzir o cenário real, característica essa conhecida como *zoom*. Esta característica permite ao usuário ampliar a imagem final ou ter uma visão panorâmica do ambiente real. Apesar dele oferecer a vantagem da ampliação do cenário em sistemas de RA por vídeo, esses sistemas possuem problemas na utilização de câmeras. O grau de qualidade da imagem real depende da resolução da câmera que captura as imagens do mundo real. Além disso, a utilização de somente uma câmera para desempenhar a função dos olhos

anula a sensação de estereoscopia, em face de que uma única imagem é vista pelos dois olhos. O ideal, então, é usar duas câmeras. Neste caso, porém, surge um outro problema, que é como acertar o posicionamento entre elas de forma a ter a mesma distância entre as pupilas de um usuário. Além disso, essa distância varia de acordo com o usuário, aumentando a complexidade ao utilizar este tipo de sistema.

3.3. Sistemas de RA utilizando Monitores

Algumas pessoas não possuem afinidade com dispositivos como HMD's. Neste caso, podem ser usados monitores de vídeo como televisões para visualizar as cenas geradas ao invés de utilizar óculos empregados nos outros sistemas de RA.

A montagem um sistema baseado em monitores é semelhante aos sistemas de RA por vídeo, com a diferença de que ao invés do combinador de cenas enviar o resultado da combinação das imagens (reais e virtuais) para um HMD, ele irá transmití-las para um monitor. A Figura 3.3 mostra um diagrama de um sistema baseado em monitores. A utilização de um *stereo glass* (óculos estéreo) é opcional, pois seu uso somente é necessário caso se queira ver imagens estereoscópica.

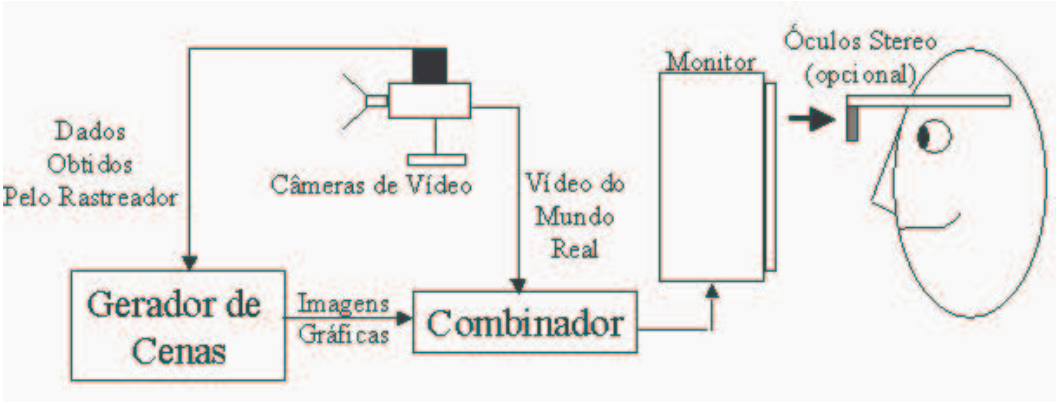

Figura 3.3 – Modelagem do sistema utilizando monitores [AZU97].

A aplicação deste modelo também é viável para sistemas óticos, possuindo um esquema similar ao da Figura 3.1, com a diferença de que os

monitores e os combinadores são posicionados em alguns locais específicos do ambiente real e o usuário pode visualizar o ambiente através deles.

A principal vantagem destes tipos de dispositivos é que o usuário não precisa vestir nenhum equipamento. Entretanto, o grande problema na utilização de monitores como televisão é o fato de ter que ficar parado em um único local, não permitindo locomover-se numa determinada área.

Uma proposta diferente foi desenvolvida pelo grupo *Sony Computer Science Labs Inc*, que criou o projeto NaviCAM [REK96], que na verdade substitui o monitor por um *palmtop* (um computador de mão) com uma câmera pendurada a ele, que tem a finalidade de capturar o ambiente real. Esta aplicação serve para a adicionar informações sobre determinados locais, através de diálogos *pop-up*, auxiliando os usuários na compreensão do que está sendo visualizado no momento (Figura 3.4).

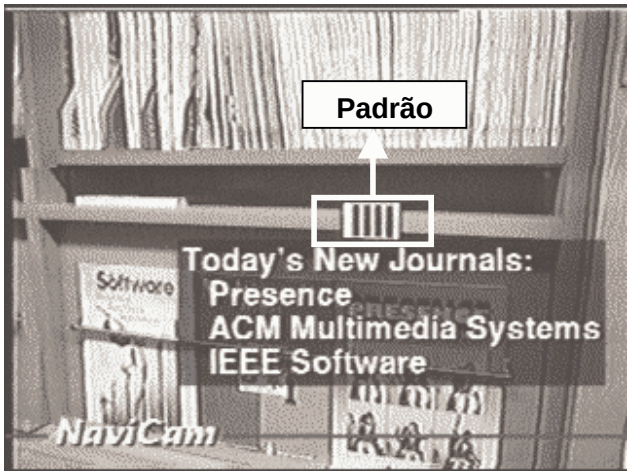

Figura 3.4 - Imagem capturada da execução do aplicativo no *palm* [REK96].

Para saber onde será posicionada a imagem virtual sobreposta a imagem real obtida pela câmera, essa aplicação funciona utilizando algoritmos de reconhecimento de padrões. Sobre cada objeto do qual o sistema possui uma informação, adiciona-se uma etiqueta contendo uma imagem. Esta imagem, uma vez captada pela câmera, é processada e “reconhecida”. Se for encontrado um dos padrões previamente cadastrados, então é desenhado sobre a imagem

capturada da câmera o texto que corresponde àquele determinado objeto. Dessa forma, o usuário não fica preso a uma cadeira e a um monitor.

4. Descrição do trabalho

Este trabalho tem como objetivo criar um aplicativo que forneça ao usuário informações a respeito de pontos específicos de uma cidade, tais como prédios, igrejas, monumentos e outras construções em geral, fazendo uso da Realidade Aumentada para adicionar os dados informativos. Este aplicativo recebe o nome de *GPS Viewer*. Como uma primeira etapa para a execução dessa tarefa, deve-se registrar a localização e as informações de todas as construções e monumentos desejados. Com o uso de um equipamento de GPS obtém-se a latitude e a longitude dos mesmos. Estes dados são armazenados em arquivo através do programa *GPS Register* para futura consulta durante a execução do *GPS Viewer*.

4.1. Recursos Utilizados

Para usufruir deste sistema, o usuário deve estar utilizando os seguintes equipamentos: Um computador portátil, um óculos de realidade virtual (HMD), um aparelho de GPS e uma câmera para capturar o cenário real. O computador serve para realizar todo o processamento de combinação das imagens (reais e virtuais), bem como a recepção e conversão dos dados obtidos pelo GPS a respeito da posição do usuário e das estruturas as quais se desejam obter informações e, por fim, enviar as imagens para o HMD, este responsável por sua exibição. Com o HMD também é possível saber para onde o usuário está olhando, já que este possui um dispositivo de rastreamento de giro da cabeça.

O computador portátil necessita de duas entradas (portas) seriais para a interconexão dos dispositivos (rastreador do HMD e o aparelho de GPS) e uma entrada (porta) USB, para a câmera digital. Maiores informações sobre o protocolo de comunicação entre o computador e o receptor GPS estão no capítulo 7.

A câmera serve para obter as imagens reais que são combinadas com as imagens virtuais. Depois de capturadas, elas são enviadas para o

computador. Quando o usuário estiver próximo e olhando diretamente para algum prédio ou monumento previamente cadastrado no sistema (isso se consegue através da leitura do *tracker* do HMD), as imagens virtuais são combinadas com as reais e exibidas no HMD (sabe-se que está olhando devido ao rastreador de giro do HMD). Caso o usuário não esteja vendo nenhum objeto cadastrado, somente será mostrada a imagem real obtida pela câmera. Maiores detalhes sobre o processo de obtenção e combinação das imagens estão apresentados no capítulo 8.

Por fim, o uso do aparelho de GPS serve tanto para obter a posição atual do usuário quanto para cadastrar as construções desejadas. Obtendo a posição do usuário, é possível saber sua distância em relação às edificações previamente cadastradas e, assim, saber quando se deve mostrar uma determinada construção. O processo de localização do usuário com o receptor GPS na Terra é descrito no capítulo 6. A Figura 4.1 ilustra um usuário utilizando os equipamentos que são necessários para a visualização desse trabalho.

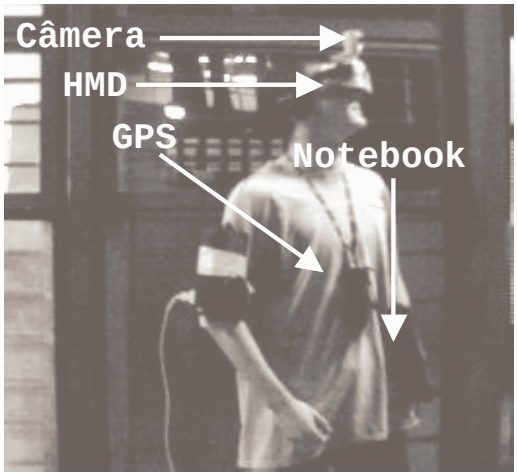

Figura 4.1 - Ilustração dos requisitos para a execução do programa [PIE02]

4.2. A arquitetura do sistema

A aplicação *GPS Viewer* informa os dados através de placas virtuais, que aparecem como objetos virtuais sobrepostos à cena visualizada. Estas

informações podem ser sobre fatos históricos ocorridos nestas construções, ano de construção e outros dados relevantes sobre a mesma (Figura 4.2).

Figura 4.2 - Ilustração da visão obtida através da RA utilizando um HMD.

Conforme já foi mencionado, o trabalho proposto está dividido em dois módulos: *GPS Register*, que é responsável pela geração dos arquivos que são pré-cadastrados no sistema; e o *GPS Viewer*, que é de fato a aplicação que gera as imagens "aumentadas" em tempo de execução. A Figura 4.3 ilustra a arquitetura desenvolvida neste trabalho.

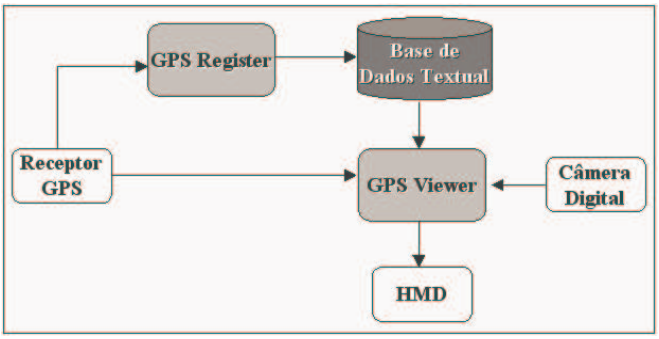

Figura 4.3 - A arquitetura do sistema

O funcionamento dessa arquitetura pode ser simplificado do seguinte modo: inicialmente deve ser utilizado o módulo *GPS Register* para gerar a base de dados textual. Quando for executado o *GPS Viewer*, ele carrega a base de dados,

recebe a imagem real obtida pela câmera digital e combina com as imagens virtuais, quando possível, enviando o resultado final para o HMD, este responsável pela sua exibição. O capítulo 5 esclarece melhor o porque de se gerar uma base de dados para o GPS Viewer.

Nas seções a seguir é abordada com mais profundidade a idéia de cada um dos módulos, bem como seu funcionamento.

4.2.1. O módulo GPS Register

O módulo *GPS Register* possui como objetivo a geração de uma base de dados em formato de arquivo texto que será utilizado pelo módulo *GPS Viewer*. A Figura 4.4 ilustra a interface gráfica inicial do módulo *GPS Register*.

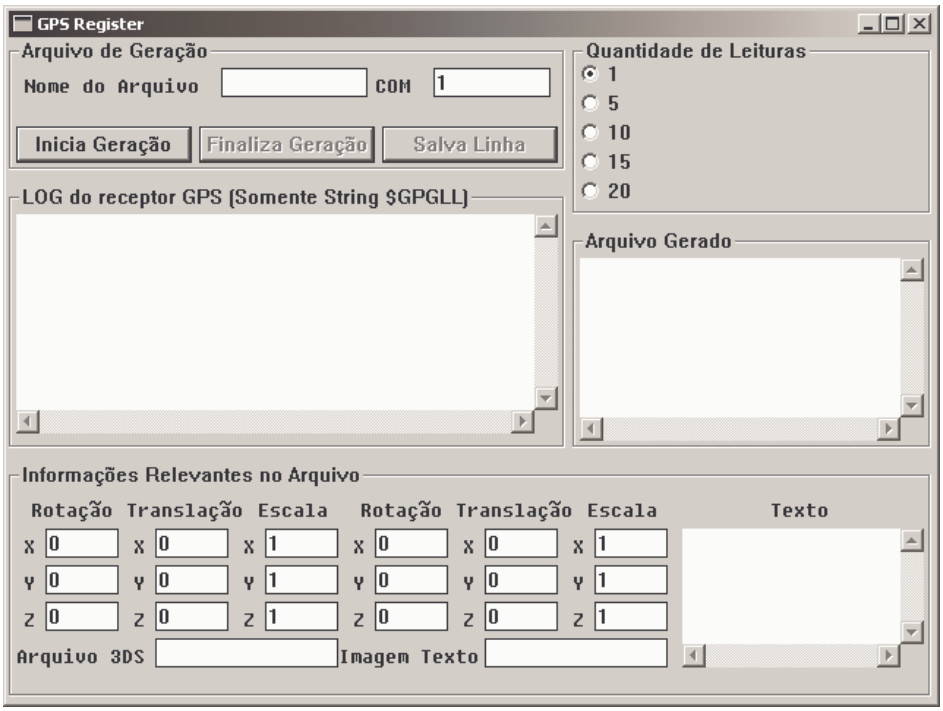

Figura 4.4 - Interface do módulo de registro dos objetos

Na base de dados são armazenadas informações sobre a posição geográfica de uma construção e um nome de um modelo tridimensional (arquivo no formato 3DS) semelhante à construção cadastrada com suas respectivas transformações geométricas, a fim de ajustá-lo às dimensões do objeto real. Além dessas informações, o arquivo gerado pelo *GPS Register* possui o texto que será

associado ao modelo 3DS. Esse texto pode ser uma imagem no formato TGA ou então um texto, também com suas devidas transformações geométricas. Caso o arquivo 3DS e a imagem virtual (texto) já possuam o tamanho ideal, podem ser mantidos os valores padrões das transformações geométricas.

Esses dados são necessários para que se possa saber onde estão localizadas as construções às quais se desejam adicionar as informações virtuais e qual a forma do objeto real (dada pelo arquivo 3DS que pode ser gerado por uma ferramenta CAD).

A Figura 4.5 mostra o formato utilizado na geração do arquivo. Esse mesmo formato pode ser visualizado no *log* de arquivo gerado na tela do *GPS Register*.

```
#FILE3DS <filename>
#SCALE <x> <y> <z>
#POSITION <x> <y> <z>
#ROTATION <x> <y> <z>
#IMAGE_TEXT <img.tga>   /* OU */    #TEXT <text>
#SCALE <x> <y> <z>
#POSITION <x> <y> <z>
#ROTATION <x> <y> <z>
```

Figura 4.5 - Formato do arquivo gerado pelo *GPS Register*

Inicialmente o *GPS Register* fica em modo de espera até que o usuário inicie a leitura dos dados enviados pelo receptor GPS. O início da leitura depende de um requisito imprescindível, que é o receptor GPS ter encontrado no mínimo quatro satélites, pois somente assim é possível obter a *string* enviada pelo aparelho de GPS contendo a posição do usuário (contendo os dados de latitude e longitude). Além dos quatro satélites, na aplicação deve ser preenchido o nome do arquivo onde são armazenados os dados, bem como deve ser informado o número da porta serial pela qual os dados do receptor GPS são recebidos. Detalhes a respeito da configuração do GPS para este trabalho estão na seção 7.4.

Depois de iniciada a leitura dos dados do GPS, o programa permanece recebendo os dados enviados pelo aparelho de GPS. Quando o usuário da aplicação clicar no botão `Salva Linha`, são gravadas no arquivo todas as informações necessárias e, logo após, continua a leitura dos dados

vindos do receptor GPS. O programa fica em *loop* até que seja finalizada a leitura ou até que seja fechada a janela. Maiores detalhes sobre a tradução das mensagens do aparelho de GPS podem ser encontrados na seção 9.4.

Uma funcionalidade interessante desse módulo é a possibilidade de se fazer uma média dos pontos lidos. Por exemplo, quando se deseja fazer uma média a cada cinco leituras, o programa faz a média aritmética das coordenadas. Isso pode ser interessante para aumentar a confiabilidade da posição informada quando o usuário estiver parado.

4.2.2. O módulo GPS Viewer

O módulo *GPS Viewer* é o módulo principal deste trabalho. Ele está encarregado de combinar as imagens reais com as virtuais, enviando-as para o HMD, este responsável por sua exibição.

Inicialmente ele carrega a base de dados textual que foi gerada pelo *GPS Register*. Carregadas e processadas as informações do arquivo, inicia-se a leitura dos dados enviados pelo receptor GPS. Essa informação serve para realizar o correto posicionamento do observador e, assim, compará-lo com as posições das construções que foram carregadas da base de dados.

O *GPS Viewer* também processa as informações sobre a rotação da cabeça do usuário (obtida pelo dispositivo HMD), obtendo, assim, a direção para onde ele está olhando. Sabendo a posição atual do usuário em relação a uma determinada construção e a direção em que olha, é possível determinar quando devem ser exibidas as imagens virtuais.

O processamento de combinação das imagens reais e virtuais ocorre, resumidamente, da seguinte forma: inicialmente o GPS Viewer recebe uma imagem real capturada pela câmera digital; quando o usuário estiver próximo a uma determinada construção pré-cadastrada e olhando para ela, a imagem virtual será combinada a imagem real e, por fim, esta é enviada para o HMD. Detalhes da composição das imagens são mostradas na seção 8.2.

4.3. Bibliotecas utilizadas

Neste trabalho, utilizam-se três bibliotecas: OpenGL [OGL03], MS Vision SDK [DOCMICROS] e SmallVR [SMALLVR], utilizadas somente no módulo *GPS Viewer*. Para fazer o *parsing* da *string* enviada pelo GPS, fez-se um estudo de uma biblioteca responsável pela leitura do equipamento de GPS. Utilizando-se funções da API do Windows, desenvolveu-se uma classe, em linguagem C++, para a interpretação das mensagens do receptor GPS, enquanto que as funções de conversão das coordenadas geográficas embutidas na *string,* foram obtidas de forma gratuíta no *site* do grupo *National Imagery and Mapping Agency* (NIMA). Ao longo dos próximos capítulos serão abordadas com mais clarezas cada uma dessas bibliotecas, mostrando o que está sendo feito com elas e como utilizá-las.

5. Problemas fundamentais de RA

Assim como qualquer aplicação de RA, este trabalho também está sujeito a alguns problemas clássicos da área: **registro** e **oclusão**. O problema do registro está relacionado ao fato de como obter as posições dos objetos reais em relação aos virtuais, enquanto que o problema da oclusão diz respeito a como serão ocultados os objetos virtuais em relação aos reais.

Estes dois problemas devem ser tratados de forma adequada a fim de evitar possíveis incoerências visuais da imagem final a ser exibida ao usuário. Nas seções a seguir são apresentados estes dois problemas, bem como algumas formas de abordá-los.

5.1. O problema do registro

Para o correto funcionamento de um sistema de Realidade Aumentada é imprescindível que objetos reais e virtuais estejam devidamente alinhados para que se tenha uma perfeita ilusão de coexistência dos dois ambientes. Esse alinhamento é chamado de *registro* e tem como objetivo informar o sistema sobre a posição e as dimensões de objetos reais que irão compor o cenário visualizado.

Os principais métodos de registro utilizados hoje são: Registro ótico por reconhecimento e registro por rastreamento. Cada uma dessas técnicas apresenta vantagens e desvantagens que devem ser consideradas dependendo da natureza da aplicação.

O **registro ótico por reconhecimento** é aquele que se vale da captura e processamento de imagens do ambiente real para a determinação da posição dos objetos reais que compõe a cena. Este método possui duas formas de atuação. A primeira baseia-se na captura da imagem do ambiente real, detecção dos contornos dos objetos e posterior identificação dos mesmos por meio de técnicas de reconhecimento de padrões. Essa técnica possui uma ótima eficiência e velocidade de processamento para ambientes de dimensões com

objetos simples, porém sofre uma alta degradação de qualidade a grandes distâncias, devido à necessidade de grande resolução requerida na captura das imagens e ainda se a variedade dos objetos é muito grande e/ou eles são muito complexos no que diz respeito à sua forma.

A segunda forma de registro por reconhecimento faz uso de **marcadores** previamente inseridos e posicionados nos objetos, para que possam ser reconhecidos em tempo de execução. Ao contrário da técnica de reconhecimento, essa técnica pode ser efetuada em ambientes maiores e mais complexos desde que os marcadores sejam bem escolhidos e posicionados.

A Figura 5.1 ilustra uma aplicação desenvolvida por um grupo da Universidade de Southampton utilizando uma biblioteca criada pelo grupo *Agents and Media Group* (AMG), chamada *Augmented Reality Toolkit* (*ARTOOLKIT*) [ART00]. Esta biblioteca serve para reconhecimento de padrões de imagens. Na Figura 5.1a tem-se o padrão utilizado para reconhecimento, enquanto nas Figura 5.1b e Figura 5.1c estão sendo desenhadas a imagem virtual. Na Figura 5.1c, quando se girou o padrão utilizado, a imagem virtual acompanhou o mesmo movimento aplicado ao objeto real.

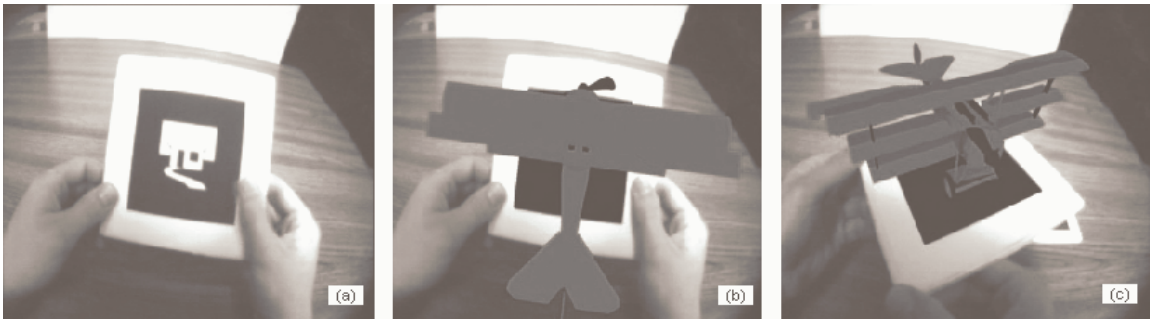

Figura 5.1 – Registro por reconhecimento com uso de marcadores [SINC02].

A técnica de **registro por rastreamento**, por sua vez, permite a utilização de vários métodos. Dentre eles pode-se destacar o acústico, o mecânico e o magnético. Atualmente o mais empregado é o magnético, por ser de simples utilização. Contudo ele torna-se impreciso e sujeito à falhas com o aumento de distância entre o transmissor e o objeto a ser rastreado. Além disso, esse modelo é facilmente perturbado por objetos metálicos ou outros campos magnéticos,

aumentando consideravelmente o erro no processo de rastreamento. Uma descrição mais detalhada sobre os vários dispositivos de rastreadores pode ser encontrada em http://www.inf.pucrs.br/~pinho/TCG/ApoioTCG.htm.

Quando é necessário se trabalhar com grandes áreas, tem-se como alternativa o **registro por rastreamento com equipamentos de GPS**. Com este equipamento pode-se obter tanto a posição de objetos reais como prédios ou veículos quanto fazer o registro do usuário do sistema. Este é o tipo de rastreamento usado no presente trabalho. Maiores detalhes sobre o equipamento de GPS são apresentados no capítulo 6.

5.2. O problema da oclusão

Oclusão refere-se ao ato de ocultar (mascarar) porções dos objetos reais ou virtuais na cena 3D tornando sua coexistência o mais real possível. Na Figura 5.2 pode-se observar uma cena real contendo dois objetos virtuais, um tabuleiro de xadrez virtual, que está sendo parcialmente oculto por um gabinete de computador real e uma mulher virtual que oculta parte do cenário real. Considerando que a geração da imagem final é feita inicialmente mostrando a imagem real e, logo após, sobrepostas a ela, as imagens virtuais, a imagem virtual apareceria sobre o gabinete nessa figura caso não fosse aplicada a oclusão em parte do tabuleiro, comprometendo, o grau de fidelidade da imagem resultante.

Figura 5.2 - Oclusão sendo aplicado ao tabuleiro [BAL00]

A Figura 5.3 ilustra melhor o problema que ocorre se não for aplicado algum método que resolva o problema da oclusão. Partindo-se de um cenário no qual um observador olha para dois objetos que representam prédios reais (a), o processo de geração da imagem final irá adicionar os objetos virtuais para compor a cena final. Para melhor ilustrar este passo, em (b) tem-se a visão aérea de onde está situado o observador em relação ao objeto real e em que direção ele está olhando. Obtida a imagem real, deve-se agora posicionar e desenhar sobre ela, todos os objetos virtuais que se deseja adicionar. Em (c) é ilustrada uma imagem, na visão superior, idêntica a de (b), só que com a imagem virtual adicionada. Nota-se que o objeto virtual está parcialmente dentro do objeto real (considerando-se a posição do observador). Entretanto, quando este for exibido sobre a imagem, o resultado gerado será incorreto, pois a imagem do objeto virtual aparece na frente do objeto real, o que não representa a imagem que era esperada (Figura 5.3d). Esse problema ocorreu devido ao fato de não ter sido feito o “registro” dos prédios, logo a sistema não tem a informação de sua existência.

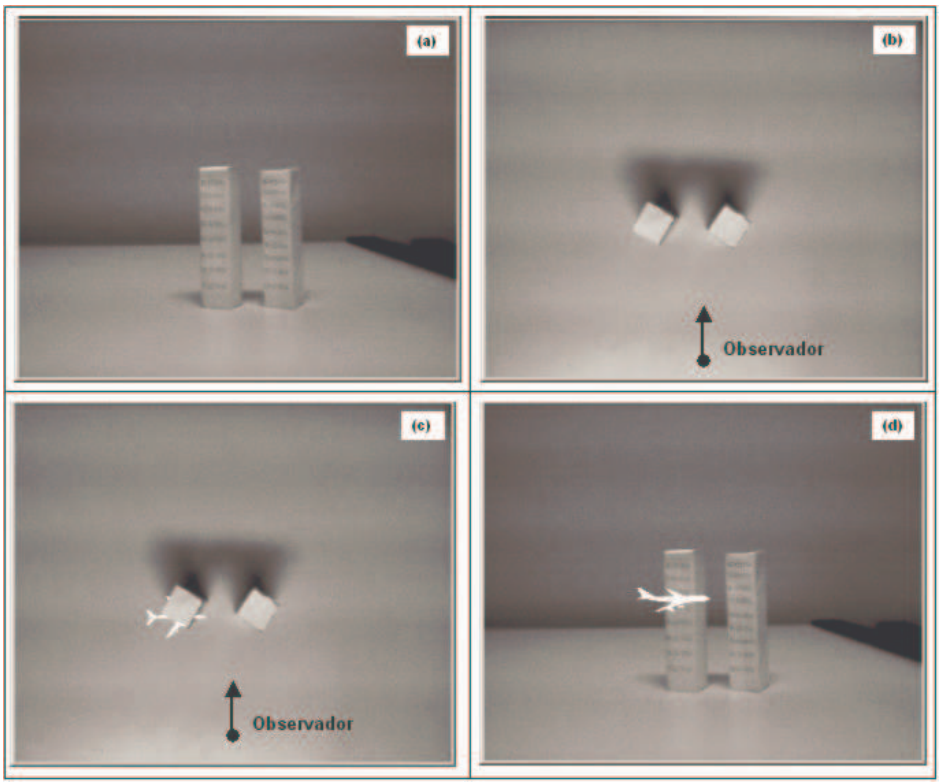

Figura 5.3 - Problema da oclusão

Para resolver esse conflito, deve-se deixar previamente cadastrados todos os objetos reais que deverão ocultar as informações virtuais, e criar com estes os objetos conhecidos como *fantasmas*. O ato de cadastrar (registrar) os objetos reais significa gerar uma imagem virtual (fantasma) que replique (em tamanho e forma) o objeto real podendo, assim, mascarar partes de uma outra imagem virtual que será adicionada à imagem real. Essas réplicas não aparecerão no resultado final (imagem combinada), e por isso são conhecidas como *fantasmas*. **A chave para a oclusão está em que esses objetos *fantasmas* irão ocultar partes dos objetos virtuais que deveriam ser ocultados pelo objeto real**. Por fim, basta unir a imagem virtual gerada (parcialmente oculta), com a imagem real, deixando a impressão de o objeto real estar obstruindo o virtual.

A Figura 5.4 ilustra o processo de mascaramento de partes das imagens virtuais e a correta oclusão do objeto virtual pelo real. Inicialmente, deve-se exibir todos os objetos virtuais *fantasmas* sobre a imagem da câmera, aplicando as transformações geométricas necessárias para seu correto

posicionamento em relação aos objetos reais, assim como é mostrada na imagem capturada pela câmera em (a). Tendo os objetos *fantasmas* já posicionados, aplica-se a técnica de mascaramento, que utilizará esses objetos para aplicar o processo de oclusão em qualquer outra imagem gráfica que passar sobre ela. Detalhes do processo de mascaramento podem ser apresentados na seção 8.3. Em (b), por exemplo, ao exibir-se o objeto virtual, parte dele não é mostrada na tela, pois a máscara (a) o está ocultando parcialmente. Por fim, o último passo é fazer a sobreposição da imagem virtual à real, que nesse exemplo é ilustrado pela união da imagem real (c) com a imagem virtual (b), resultando na imagem final (d). Nesse processo, em outras palavras o que está sendo feito é gerar imagens virtuais com partes delas já ocultas pelo modelo tridimensional do objeto real. Por isso é importante se gerar modelos tridimensionais tão parecidos quanto possível com os reais, evitando assim, incorreções no processo de oclusão.

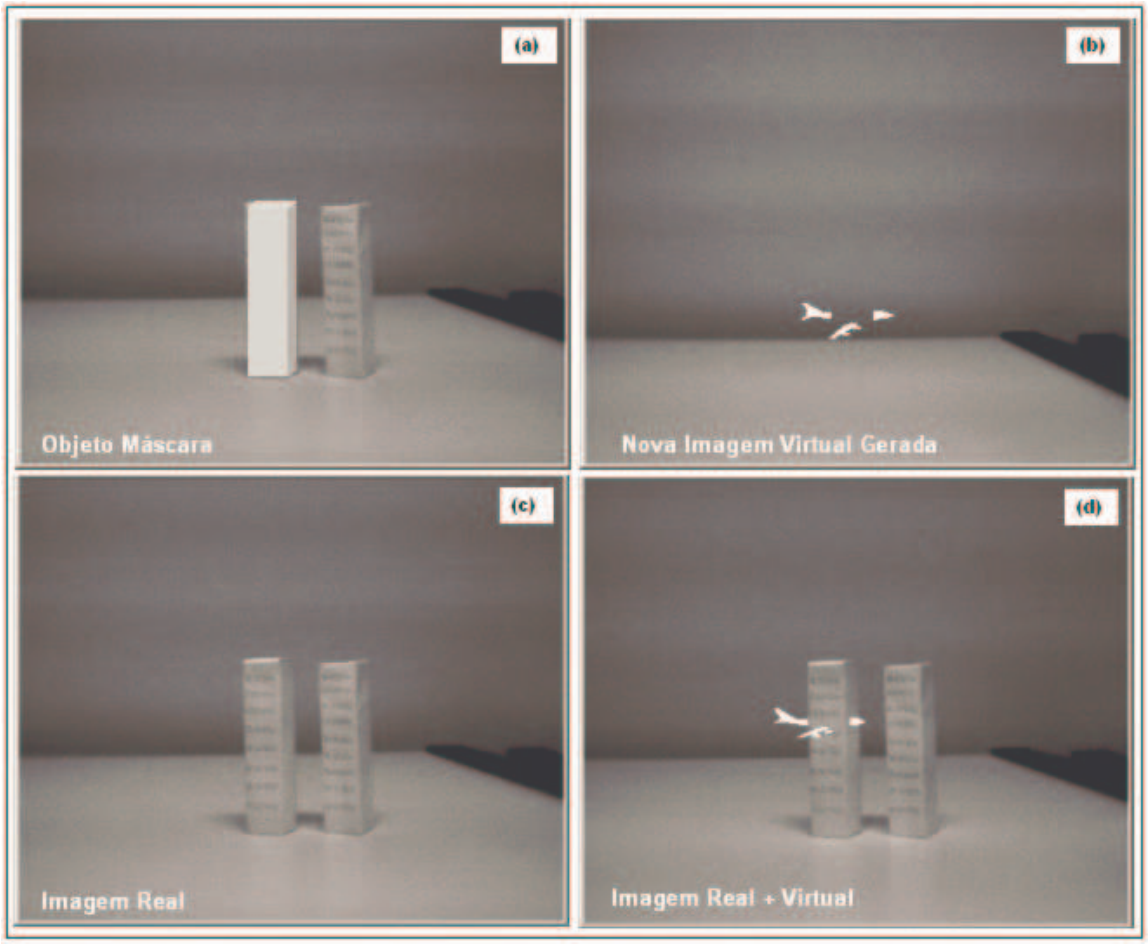

Figura 5.4 – Processo da máscara de objetos reais

5.3. Motivos para se tratar registro e oclusão neste trabalho

Em um sistema de informações geográficas com RA, não registrar alguns objetos reais que ocultam objetos virtuais pode fazer com que a aplicação mostre informações sobre alguma construção de forma incorreta. Neste caso, dependendo da posição em que se encontra o usuário da aplicação, o dado que pertencia a uma certa construção pode ser visualizado erroneamente sobre uma outra construção.

Por exemplo, supondo que um usuário encontre-se em um ponto de uma cidade próximo a um monumento sobre o qual serão exibidas informações.

Ao virar-se para este monumento, um objeto virtual informativo apareceria logo à frente da construção exibindo os dados sobre a mesma. Isso ocorre naturalmente, pois, sabendo a posição do usuário através do GPS e a direção em que ele está olhando, através do HMD, tem como se saber quando devem ser mostradas as imagens virtuais.

Entretanto, caso exista um outro objeto real cadastrado interpondo uma outra construção **também cadastrada** no sistema, deve-se determinar qual imagem virtual será mostrada. A Figura 5.5 ilustra essa situação, na qual as informações sobre o prédio do Centro Administrativo do Estado do Rio Grande do Sul são exibidas juntamente com as informações sobre o prédio da Companhia Procergs, ambos registrados no sistema. Na figura (a), tem-se a visualização da cena na qual a oclusão está com as placas mal posicionadas, fato que pode gerar confusão ao usuário. Por sua vez, a figura (b) demonstra a mesma cena com a placa devidamente posicionada, tendo-se o prédio da Procergs ocultando parte da placa do Centro Administrativo.

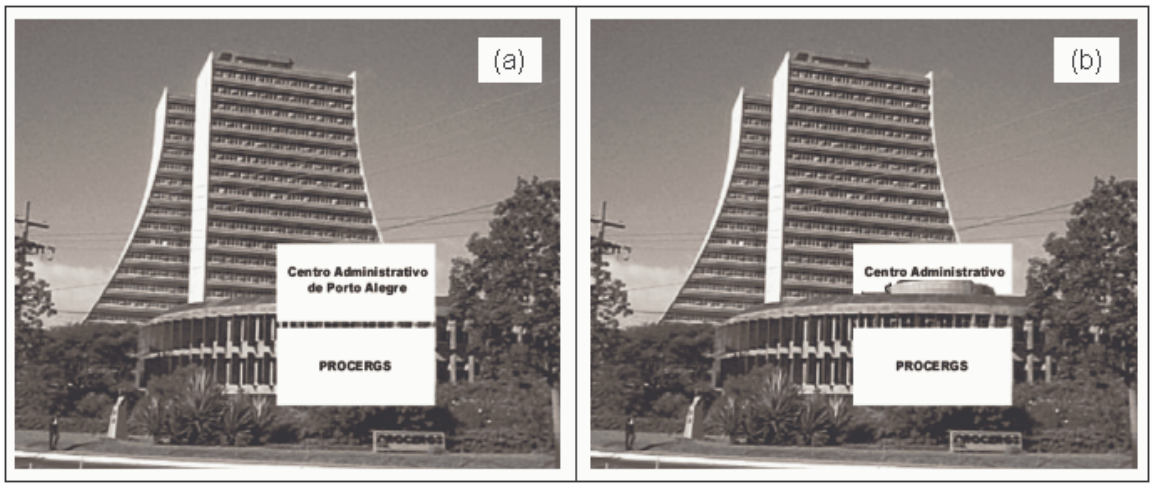

Figura 5.5 - Centro Administrativo de Porto Alegre (1º plano), Companhia Procergs (2º plano) [POA02].

O problema ocorre na disposição das imagens virtuais. Caso na imagem acima os textos estivessem lado a lado, seria difícil dizer qual pertence a uma determinada construção. A utilização de algoritmos de cálculo da distância

entre o observador e uma determinada construção pode resolver alguns casos, mas se duas construções estiverem muito próximas, a utilização do algoritmo da distância se torna inviável, pois não se consegue evitar o problema. Logo, o correto posicionamento dos textos virtuais deve ser tratado para que a aplicação funcione de forma correta.

6. Localização sobre o globo terrestre

Conforme mencionado anteriormente, este trabalho necessita obter a posição do usuário e dos objetos reais sobre os quais serão exibidas as informações. O objetivo disto é orientar o sistema na seleção das informações a serem exibidas de acordo com a posição atual do usuário em relação à posição dos objetos previamente cadastrados.

No passado, a determinação de posições era feita através do uso de equipamentos para rádio-navegação, entretanto eles não eram muito precisos. Mais tarde, com o desencadeamento da II Grande Guerra, desenvolveram-se sistemas mais precisos conhecidos como RADAR (*RAdio Detection And Ranging*) capazes de obter a posição de objetos através da emissão/recepção de ondas de rádio. Para se determinar a posição, mede-se o intervalo de tempo entre os sinais de rádio provenientes de locais conhecidos emitidos por transmissores ao mesmo tempo e com a mesma velocidade de propagação. Possuindo as exatas localizações dos transmissores, a velocidade das ondas de rádio e a diferença de tempo entre os dois sinais, pode-se obter a posição em uma dimensão, pois ele está numa linha reta entre os dois transmissores. A utilização de três transmissores permite obter uma posição bidimensional, em latitude e longitude [GOR01].

Baseados nesse mesmo princípio foram desenvolvidos os GPS's (*Global Positioning System*). O funcionamento desse sistema é idêntico ao do RADAR, com a diferença de que ao invés de se utilizar transmissores de rádio, são usados satélites que orbitam em torno da Terra e permitem saber a posição em três dimensões (latitude, longitude e altitude) de um receptor na superfície da Terra.

Os primeiros GPS's foram desenvolvidos com o intuito de auxiliar o exército americano na localização de alvos no mundo [TRI02]. Atualmente, estes instrumentos são empregados também quando se deseja determinar a posição e a velocidade de navios, carros e aviões comerciais, bem como sistema antifurto de

automóveis. Enfim, este dispositivo tem ajudado o cotidiano das pessoas no cumprimento de suas atividades.

Para que um usuário localizado em qualquer ponto da superfície terrestre possa fazer uso desta tecnologia, é necessário um grande conjunto de satélites sincronizados e distribuídos na estratosfera. O sistema é composto por vinte quatro satélites, distribuídos uniformemente em seis planos orbitais (Figura 6.1). Estes satélites são monitorados pelas chamadas *Ground Stations*, que verificam seu estado operacional e sua posição exata no espaço. As *Ground Stations* enviam informações sobre distâncias, posição e velocidade dos satélites que, por sua vez, atualizam seus dados e retransmitem aos receptores GPS.

Figura 6.1 - Planos orbitais [GARMIN]

O funcionamento de um GPS na determinação de uma posição é apresentado a seguir. Primeiramente é necessário que o receptor GPS conheça sua distância relativa aos satélites com os quais se comunica. Além disso, cada um dos satélites deve saber sua posição exata no espaço, a qual é informada pela sua respectiva *Ground Station*. Então, o cálculo que determina sua posição é realizado com o uso de quatro satélites. Sabendo a distância “d” do primeiro satélite até o receptor GPS, obtém-se uma esfera de raio “d”, centrada no satélite, cuja superfície remete às possíveis localizações do GPS. Aplicando o mesmo raciocínio para um segundo obtém-se outra esfera, que, ao ser interseccionada com a primeira, resulta em uma circunferência de possíveis localizações do

receptor (). Com o terceiro satélite obtém-se uma nova esfera que ao ser interseccionada com esta circunferência, reduz o número de possíveis localizações do receptor GPS à apenas dois pontos. Para saber sua localização exata, é necessária a presença de um quarto satélite que elimina o ponto mais distante da superfície terrestre. Pode-se também eliminar um dos pontos por lógica, visto que um deles tem uma coordenada que está fora dos limites da superfície da Terra. A Figura 6.2 demonstra as etapas deste procedimento.

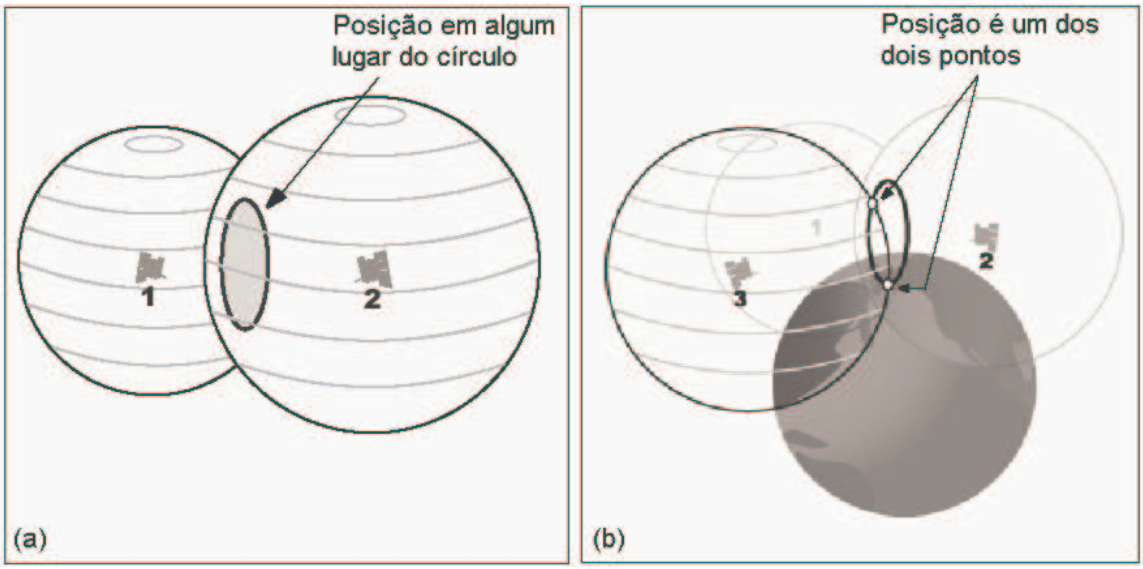

Figura 6.2 – Funcionamento utilizando 2 satélites (a) e 3 satélites (b) [GAR99].

Através do tempo de propagação de um sinal enviado da fonte ao destino, é possível calcular a distância entre um satélite e um ponto na Terra. Como a física demonstra, o produto entre o tempo de propagação do sinal com a velocidade do mesmo resulta na distância do satélite até um ponto com GPS. Porém esse cálculo não é trivial devido à influência do meio no tempo de propagação do sinal. Esse sinal é modulado e codificado sendo particular para cada satélite garantindo assim sua identificação no sistema geral.

Existem alguns problemas que dificultam a precisão dos GPS's. Interferências ionosféricas geram atrasos nos recebimento dos sinais; a geometria dos satélites em relação ao observador pode não ser propícia para que o receptor GPS consiga calcular a posição de forma precisa, sendo necessário que o receptor busque sinais provenientes de outras regiões (Norte, Sul, Leste e Oeste);

a interferência resultante da reflexão do sinal em algum objeto, aumenta a distância percorrida pelo sinal fazendo com que o receptor interprete que o satélite está mais longe que o normal.

A comunicação com o GPS está sujeita a problemas de *timing*, nos quais pequenos *overheads* no processamento dos dados no satélite geram grandes atrasos na transmissão do sinal. Isso ocorre devido aos baixos tempos de propagação da onda no meio, sendo necessário o uso de *clocks* de alta freqüência e muito precisos. Esses *clocks* são chamados de *clocks* atômicos. Seu nome deriva da constante oscilação eletromagnética dos átomos, que produz uma ressonância extremamente curta e precisa garantindo assim o sincronismo do satélite. Infelizmente o mesmo não ocorre com os receptores. GPS's utilizando *clocks* atômicos teriam um custo muito superior aos encontrados atualmente. Para se ter uma noção de valores, um GPS equipado com esse mecanismo estaria em torno de US$ 50K a US$ 100K, tornando inviável, em muitos casos, a compra do mesmo.

7. Protocolo de comunicação entre computador e GPS e configurações do receptor

A fim de que a aplicação tome conhecimento da posição atual do usuário e dos objetos a serem registrados, é necessário um meio de comunicação entre o receptor GPS e um computador executando a aplicação proposta nesse trabalho. Para tanto, far-se-á uso da interface serial de comunicação RS-232.

Essa interface de comunicação com o GPS suporta transferência bi-direcional de dados, permitindo o envio e recebimento de rotas, *track logs*, posições, entre outros. A interface está dividida em três camadas, que são a camada de aplicação, a camada de link e a camada física. Cada camada implementa diversos protocolos, que variam de acordo com a versão do equipamento de GPS. Nas próximas seções são descritas cada uma das camadas e suas respectivas funcionalidades.

7.1. O protocolo da camada física padrão

O funcionamento da camada física está baseado nas especificações da interface RS-232. Dentre os protocolos por ela implementados, o protocolo da camada física padrão está presente em todas as versões de GPS e é fundamental para o estabelecimento da *conexão*. Ele implementa as regras de transmissão dos dados em baixo nível, efetuando a comunicação do GPS com o *host* a ser *conectado*, fazendo uso de conectores DB-9 ou DB-25 para o estabelecimento da *conexão*. No modelo de GPS Garmin 12, utilizado neste trabalho, esta comunicação é do tipo *serial full duplex* e ocorre com uma taxa de 9600 *bauds*. A palavra de dados é de oito *bits* sem o *bit* de paridade, sendo que o ultimo *bit* é o *stop bit*.

7.2. O protocolo da camada de Link

Este protocolo é responsável pela transmissão dos dados. Sua função é facilitar a comunicação inicial entre o GPS e o *host*, fazendo este último saber com qual GPS está *conectado*.

Os pacotes de dados transmitidos são orientados a caractere, ou seja, possuem tamanho múltiplo de 8 bits. Esse protocolo é do tipo confirmado, ou seja, faz uso de pacotes do tipo *ACK* e *NAK* para indicar o estado da comunicação em nível de camada *link*, não verificando erros na camada de aplicação. Seu funcionamento é baseado na idéia de *stop and wait*, na qual todo pacote de dados recebido por um dos equipamentos deve ser sinalizado com o envio de um *ACK* ou *NAK*, antes do envio do próximo pacote. O recebimento de um pacote *ACK*, indica que o pacote previamente enviado foi recebido com sucesso e a comunicação pode continuar. Se um pacote *NAK* for recebido, o pacote anterior deve ser reenviado. A fim de evitar *deadlock*, um *NAK* também pode ser enviado, como mecanismo de *timeout*, quando o *host* demora a enviar os pacotes seguintes da comunicação (2-5 segs).

Cada pacote contém um cabeçalho (*header*) de três *bytes* (DLE, ID, e o tamanho). A Tabela 7.1 mostra como é o formato do pacote da camada de *Link*. O *byte* DLE serve para demarcar o início de um pacote do protocolo de *link*. Se existir algum *byte* no pacote com o mesmo valor do *byte* DLE, então outro *byte* DLE é inserido imediatamente após o mesmo indicando que não é o início de um novo pacote. Esse novo *byte* não é contabilizado no tamanho final do pacote nem utilizado no cálculo do *checksum* do mesmo.

Um pacote *ACK* possui o campo ID do pacote com valor 6 decimal, enquanto um pacote *NAK* possui o valor 21 no mesmo campo. Para indicar a qual pacote eles se referem, ambos carregam o identificador do pacote referido (8 bits) em seu campo de dados.

Tabela 7.1 – Formato do pacote de dados [GAR99]

Numero do Byte	Descrição do Byte	Notas
0	*Data Link Escape* (Delimitador DLE)	Caractere ASCII DLE (16 em decimal)
1	ID do pacote	Identifica o tipo do pacote
2	Tamanho dos dados do pacote	Numero de bytes da área de dados do pacote (bytes 3 até n-4)
De 3 a n-4	Dados do pacote	De 0 a 255 bytes
n-3	*Checksum*	Complemento de 2 da soma de todos os bytes (do 1 até n-4)
n-2	*Data Link Escape* (Delimitador DLE)	Caractere ASCII DLE (16 em decimal)
n-1	Fim de texto	Caractere ASCII ETX (3 em decimal)

O campo de *Dados do pacote* possui tamanho variável (de 0 a 255 bytes) e pode carregar qualquer informação referente a algum protocolo da camada de aplicação descrito a seguir.

7.3. Os protocolos da camada de aplicação

A camada de aplicação possui diversos protocolos para a transferência de dados entre o GPS e o computador. Estes protocolos implementam todas as funcionalidades de envio e recebimento de qualquer tipo de mensagem entre o *host* e o GPS (*waypoints*, rotas, posições, entre outros) sendo definidos como uma seqüência ordenada de pacotes contendo um sentido, ID do pacote e tipo de dados do pacote. A maioria destes protocolos de aplicação são simétricos, ou seja, tanto o GPS quanto o *host* podem assumir o papel de origem ou destino da comunicação. A seguir estão descritos os protocolos necessários para a implementação deste trabalho.

7.3.1. O Protocolo de requisição de versão

Esse protocolo deve ser implementado por qualquer aplicação que deseje estabelecer uma *conexão* com o aparelho de GPS. Ele é utilizado para

obter a versão de GPS em que o computador está conectado, a fim de determinar quais protocolos de transferência de dados são suportados pelo GPS. A Tabela 7.2 mostra as mensagens trocadas entre os dispositivos para a obtenção dos dados do produto.

Tabela 7.2 - Mensagens para obtenção de informações sobre o produto [GAR99].

Nro do Pacote	Sentido	ID do pacote	Tipo de dados do pacote
0	Host → GPS	Pid_Product_Rqst	Ignorado
1	Host ← GPS	Pid_Product_Data	Product_Data_Type

No pacote 0, o byte *Pid_Product_Rqst* indica uma requisição de versão enviada ao GPS. No pacote 1, o byte *Pid_Product_Data* indica que a parte do tipo de dados do pacote, contém uma estrutura de dados (struct *Product_Data_Type*) que possui os dados do GPS. A Figura 7.1 mostra a definição dessa estrutura. Os valores dos identificadores do pacote estão descritos no Anexo A-2.

Figura 7.1 - Estrutura de dados da versão do produto (Linguagem C).

```
typedef struct {
       int product_ID;
       int software_version;
} Product_Data_Type;
```

7.3.2. O Protocolo de obtenção de posições

Esse protocolo é usado para transferir a posição (latitude e longitude em radianos) entre o GPS e o computador. Ao fazer a requisição de uma posição, o computador deve gerar um pacote com a formatação descrita na Tabela 7.3.

Tabela 7.3 – Formato da mensagem para obtenção de posição [GAR99].

N	Sentido	ID do pacote	Tipo de dados do pacote
0	Host → GPS	Pid_Command_Data	Command_Id_Type

Os valores do identificador do pacote *Pid_Command_Data* e do flag de tipo de dados *Command_Id_Type* estão descritos no Anexo A-2 “L001 - Protocolo da camada de link 1” e “A010 - Protocolo de comandos do dispositivo de GPS 1”

respectivamente. O GPS ao receber essa requisição irá automaticamente respondê-la com um pacote com a formatação descrita na Tabela 7.4.

Tabela 7.4 - Estrutura de dados das coordenadas de um ponto

N	Sentido	ID do pacote	Tipo de dados do pacote
0	Host ← GPS	Pid_Position_Data	<D0>

O valor do identificador do pacote *Pid_Position_Data* está descrito no Anexo A-2 “A010 - Protocolo de comandos do dispositivo de GPS 1”. Esse pacote possui uma *string* que contém, dentre outras informações, as coordenadas geográficas do ponto atual. Maiores informações sobre o formato da *string* retornada pelo receptor GPS são encontradas na seção 9.4. A posição obtida é armazenada na estrutura da Figura 7.2, definida utilizando a linguagem C.

Figura 7.2 - Estrutura de dados das coordenadas de um ponto [GAR99].

```
typedef struct {
       double lat; //latitude em radianos
       double lon; //logitude em radianos
} Radian_Type;
typedef Radian_Type d700_Position_Type;
```

Os protocolos descritos nesse trabalho são apenas alguns dos protocolos implementados pelo modelo de GPS 12 da Garmin. Mais informações sobre os protocolos e tipos de dados podem ser encontradas no *site* da Garmin (http://www.garmin.com/support/pdf/iop_spec.pdf).

7.4. Configurações do receptor GPS neste trabalho

O correto funcionamento dos módulos desenvolvidos neste trabalho prescinde que se tenha a seguinte configuração no aparelho de GPS: o formato dos dados a respeito da posição deve ser hdddºmm.mmm’, onde hddd é 3 dígitos de grau; mm são 2 dígitos de minutos e mmm representa milésimos de minuto e o sistema de referência do aparelho de GPS utilizado é o WGS-84. Maiores informações sobre esse sistema de referência podem ser encontrados em http://br.groups.yahoo.com/group/parapenteportugal/message/1629 ou no artigo encontrado em http://www.cartografia.org.br/xixcbccd/artigos/c2/cII59/cII59-99.pdf.

A respeito da transmissão dos dados, a interface utilizada deve ser NMEA/NMEA, pois o *parser* das mensagens do receptor GPS é compatível com a especificação do padrão NMEA 0183 2.0. Por fim, a velocidade de transmissão dos dados pela serial deve ser de 9600 baud, velocidade máxima permitida pelo modelo de receptor utilizado.

8. Captura e integração de imagens em tempo de execução

Um dos principais requisitos para o desenvolvimento deste trabalho é a fusão entre imagens reais e virtuais. As imagens reais são obtidas através de uma câmera de vídeo e as virtuais são geradas através de alguma biblioteca gráfica.

Para desenvolvimento de uma aplicação que utilize um dispositivo de vídeo para captura de imagens e reproduza-as numa janela, foi utilizada a biblioteca MS Vision SDK (*Microsoft Vision Standard Development Kit*) [MSVISION], da Microsoft, que serve para manipulação de imagens. Para a geração das imagens virtuais, testou-se as bibliotecas GDI (*Graphics Device Interface*) [DOCMICROS] e OpenGL [OGL03].

A fusão das imagens reais e virtuais é feita a partir da conversão da imagem gerada pela câmera para um formato digital e pela posterior sobreposição dos desenhos virtuais sobre essas imagens.

Nas seções a seguir são apresentadas a biblioteca MS Vision SDK e as formas de compor as imagens quando se usa OpenGL e a GDI.

8.1. MS Vision SDK

A MS Vision é uma biblioteca de captura de imagens disponibilizada pela Microsoft de forma gratuita no *site* http://research.microsoft.com/projects/VisSDK/, que se baseia nos padrões VFW (*Video For Windows*) ou *DirectShow* para capturar uma imagem proveniente de uma câmera de vídeo e convertê-la para um formato digital representado por um mapa de bits (*bitmap*).

8.1.1. Funções de Captura da MS Vision SDK

Esta seção tem como objetivo mostrar os passos que devem ser realizados para adquirir as imagens *ao vivo* vindas da câmera. A MS Vision tem

como pressuposto a execução de três fases para essa aquisição: primeiramente tem-se a fase de tentar encontrar um *originador de imagens* (câmera), logo após é necessário fazer a *conexão* com o dispositivo de vídeo e por fim fazer a aquisição das imagens capturadas pela câmera.

A execução dessas fases é realizada através de um pacote de funções da MS Vision chamado *VisImSrc* (*Vision Image Source*), que permite escolher um dos digitalizadores instalados na máquina e também oferece suporte para aquisição das imagens da câmera.

Para saber qual é o originador das imagens (*Image Source*), a MS Vision possui um método chamado *VisFindImageSource*, que tem como função descobrir se há algum dispositivo de vídeo conectado à máquina. No caso de sucesso, essa função retorna um objeto chamado *CVisImageSource*, que será necessário para realizar a *conexão* com o dispositivo.

A segunda fase necessária para a obtenção das imagens é a *conexão* com o dispositivo de vídeo. A *conexão* é feita utilizando-se a classe *CVisSequence*, que é responsável também por reservar uma região de memória onde serão armazenadas as imagens capturadas pela câmera. Essa classe possui um método chamado *ConnectToSource* que é responsável pela *conexão* ao dispositivo, este previamente obtido pelo método *VisFindImageSource*.

Por fim, o último passo é a captura das imagens armazenadas em uma região de memória específica. O objeto *CVisSequence* possui um método que permite obter imagens que estão nesta região através dos métodos *Pop* e *PopFront*. Essa classe também permite que se controle a quantidade de imagens que serão armazenadas.

Nesse trabalho, a cada momento é necessário processar uma imagem, e não um conjunto de imagens. Para tanto, a classe *CVisSequence* possui o método *SetUseContinuousGrab* que, se receber por parâmetro o valor *false*, configura o objeto *CVisSequence* para armazenar somente uma imagem em memória.

8.1.2. Acesso a imagens

Uma imagem é formada por um *array* de *pixels*. A partir desse conceito que a biblioteca MS Vision implementa uma classe chamada *CVisImage*, que é responsável pela manipulação das imagens digitalizadas. Essa estrutura, similar a estrutura *Windows Bitmap Header* [DOCMICROS] da API (*Application Program Interface*) do Windows, possui uma variedade de propriedades a respeito de uma imagem, entre elas, um ponteiro para a área de memória usada para armazenar os dados da imagem, permitindo assim, o acesso aos *pixels* sem a necessidade de copiar a imagem para uma área da aplicação.

Usando o MS Vision, cada imagem deve ser modelada como uma instância da classe *CVisImage*, que é na verdade um *template* para a definição de um *array* de *pixels* de um certo tipo. Este tipo pode ser tanto *char, byte (unsigned char)*, *short*, *unsigned short*, *int*, *unsigned int*, *long*, *unsigned long*, *float* ou *double* quanto um dos tipos definidos na MS Vision, que são:

- *Pixels* em tons de cinza: armazena apenas um único componente de intensidade (tom de cinza). Estes pixels podem ser nomeados apenas com o tipo de dado (por exemplo, *CVisBytePixel*) ou com *Gray* + tipo de dado + *Pixel* (por exemplo, *CVisGrayBytePixel*). A *CVisBytePixel* é compatível com o formato de *bitmaps* de escalas de cinza do Windows;
- *Pixels* com cores RGBA: contém quatro componentes de intensidade de cor, que são o vermelho, verde, azul e *alfa* (medida de transparência – imagens capturadas da câmera não possuem este valor preenchido). São nomeados na forma *CVisRGBA* + tipo (por exemplo, *CVisRGBABytePixel*). Na MS Vision os dados deste tipo são armazenados na ordem B-G-R-A, tornando-os compatíveis com o formato usado nos *Bitmaps* coloridos do Windows;
- *Pixels* em cores YUVA: possui quatro componentes (luminescência, tonalidade, saturação e *alfa*). Nesse tipo, Y e A são *unsigned int*, enquanto que U e V são *signed int*. São nomeados sob a forma *CVisYUVA* + tipo (por exemplo, *CVisYUVABytePixel*).

Para cada um destes tipos de *pixels* definidos na biblioteca, existe um tipo de imagem correspondente a ele. Por exemplo, quando for utilizado o tipo de *pixel CVisYUVABytePixel*, deve-se usar a imagem *CVisYUVAByteImage*, que define uma imagem composta por um *array* de *YUVA bytes pixels*. Para esse trabalho será usado a imagem *CVisRGBAByteImage* (do tipo de pixel *CVisRGBABytePixel*) em face da sua maior compatibilidade com os formatos internos do ambiente Windows.

Na Figura 8.1 encontra-se um trecho de código que realiza os passos necessários para utilizar um dispositivo de vídeo e fazer a captura de imagens de uma câmera.

```
//FASE 1: TENTANDO ENCONTRAR UM ORIGINADOR DE IMAGENS:
HDC hdc;
/*Conecta usando o driver VFW (Video For Windows)*/
VisAddProviderRegEntryForVFW();
/*Tentando encontrar um Image Source*/
CVisImageSource imagesource = VisFindImageSource("");
/*Seqüência de imagens do tipo BYTE RGBA*/
CVisSequence<CVisRGBABytePixel> sequence;
//FASE 2: TENTANDO CONECTAR-SE AO DISPOSITIVO
/*Conectando a um dispositivo encontrado*/
sequence.ConnectToSource(imagesource, true, false);
/*Tratamento para caso não haja sucesso*/
if (!sequence.HasImageSource() || !sequence.ImageSource().IsValid())
   printf("Erro: Não pode-se conectar ao dispositivo ou ImSrc não é valido\n");
/*Seta a propriedade da seqüência de imagens para somente pegar 1 imagem*/
sequence.ImageSource().SetUseContinuousGrab(false);
//FASE 3: CAPTURANDO A IMAGEM
long timeout = 40000;
/*objeto para manipulação das imagens*/
CVisRGBAByteImage image;
/*pegando a imagem da seqüência. Caso passe o timeout, a imagem não é capturada*/
if (!sequence.Pop(image, timeout))
   printf("Erro: Não pode-se exibir a imagem");
/*Numa rotina que atualize o frame buffer*/
image.DisplayInHdc(hdc);
/*Ao término do programa, desconectar do dispositivo*/
sequence.DisconnectFromSource();
```

Figura 8.1 - Passos para trabalhar com imagens capturadas da câmera

8.2. Composição das imagens

Depois de obtida a imagem real, é necessária fundi-la com as imagens virtuais geradas por uma biblioteca gráfica e exibir o resultado final na tela.

Para manipular uma imagem, necessita-se um ponteiro para a região de memória onde a imagem se encontra. Para obter este ponteiro, a classe *CVisImage* dispõe do método *PbPixel*, que recebe por parâmetro um ponto da imagem em coordenadas de *pixel* e retorna um ponteiro do tipo *BYTE* (*unsigned char*) para a posição deste *pixel* dentro da imagem. No caso desse trabalho, necessita-se de um ponteiro para o primeiro ponto da imagem que pode ser obtido através da função *StartPoint*, que retorna o ponto mais a esquerda e acima da imagem. Caso queira utilizar algum outro ponto, deve-se passar um objeto do tipo *CPoint* com a posição requerida. Nas próximas seções são apresentados trechos de código que demonstram sua utilização.

8.2.1. Integrando OpenGL com o MS Vision SDK

Obtido o ponteiro para a imagem, é necessário realizar a fusão entre as imagens capturadas pelo dispositivo de vídeo com as imagens virtuais. A biblioteca OpenGL provê, entre outras funções, métodos que permitem a combinação de mapas de bits (*bitmaps*) com imagens geradas por suas rotinas de desenho bi ou tridimensionais.

A combinação das imagens torna-se possível através da rotina *glDrawPixels*, que escreve um bloco de *pixels* no *frame buffer*. Esta rotina recebe por parâmetros as dimensões do retângulo (largura e altura respectivamente) que será escrito no *frame buffer*, o formato em que está armazenada a imagem, o tipo de dado da imagem (definido por constantes OpenGL) e um ponteiro para o início da área (região de *pixels*) que será desenhada. Utilizou-se o formato GL_BGRA_EXT e o tipo de dado GL_UNSIGNED_BYTE, garantindo compatibilidade com o tipo de imagem *CVisRGBAByteImage* da biblioteca MS

Vision SDK, tanto na ordem de como estão armazenadas as cores como no tipo de dado que ela possui. O ponteiro foi obtido pelo método *PbPixel* da MS Vision.

Sabendo que para a MSVision a coordenada (0,0) das imagens se situa no extremo esquerdo e acima da janela e para a OpenGL encontra-se no extremo esquerdo e abaixo, é necessário invertê-las no eixo *Y* de forma a preservar sua correta orientação vertical. Para tanto, a biblioteca OpenGL possui a função *glPixelZoom*, função esta que especifica o fator de *zoom* do *pixel* e que deve receber os valores *1* e *–1*, respectivamente, pois assim conseguirá inverter a imagem de forma que ela apareça corretamente sem alterar seu tamanho original.

Após a inversão da imagem, também é necessário informar a posição a partir da qual será iniciado o desenho da imagem na janela. Para realizar isso, OpenGL fornece em seu conjunto de funções o método *glRasterPos2i*, que recebe por parâmetro a posição bidimensional, que informa a partir de onde começará a ser desenhado esta região de bits. Como a imagem está invertida, então deve-se iniciar, em y, pelo tamanho máximo da imagem - 1 (pois a imagem inicia em zero e não em um), enquanto que em x, o valor atribuído a ele é zero. Desenhada a imagem real, resta promover a integração das imagens virtuais geradas por computador através das rotinas fornecidas pela OpenGL.

Na Figura 8.2 segue um pequeno trecho de código que descreve como é realizada a fusão entre as imagens reais com as imagens virtuais criadas através de rotinas OpenGL.

```
/*Este trecho de código é uma função que deve ser criada para geração de textos*/
void drawString (void * font, char *s) {
	unsigned int i;
	for (i = 0; i < strlen (s); i++)
		glutBitmapCharacter (font, s[i]);
}
void texto() {
	glPushMatrix();
	glColor3ub(255,0,0);
	glRasterPos2f(50,50);
	drawString (GLUT_BITMAP_HELVETICA_18,"Mouse");
	glPopMatrix();
}
/*Este trecho de código deve ser colocado em uma rotina de redesenho (repaint)*/
/*Inversão da imagem verticalmente*/
glPixelZoom(1.0, -1.0);
/*ajuste do tamanho da imagem real*/
glRasterPos2i(0, image.Width()-1);
/*desenho a imagem real na tela*/
glDrawPixels(image.Height(),image.Width(),GL_BGRA_EXT,GL_UNSIGNED_BYTE,
		(unsigned char *)image.PbPixel(image.StartPoint()));
/*Desenho da imagem virtual*/
texto();
```

Figura 8.2 - Código mostrando os passos para combinar OpenGL e MS Vision

A Figura 8.3 ilustra como ocorre o processo de composição das imagens reais obtidas pela câmera com as imagens virtuais geradas por computador.

Figura 8.3 – Aplicação rodando com OpenGL.

8.2.2. Integrando a GDI com a MS Vision SDK

Uma outra forma de promover a integração de imagens geradas computacionalmente com imagens capturadas pelo dispositivo de vídeo é utilizar as funções da GDI do Windows. Como a MS Vision SDK utiliza como padrão alguns formatos fornecidos pela API do Windows e o modo como armazena as imagens em memória é idêntico ao padrão *Windows Bitmaps*, a integração é bastante simples.

Para conseguir usar uma imagem com a Windows GDI é necessário ter um ponteiro para o chamado contexto de dispositivo (*Handle Device Context* ou *HDC*). Para obter o contexto da imagem capturada pela MS Vision, a classe *CVisRGBAByteImage* possui o método chamado *Hdc*, o qual retorna o *HDC* da imagem. É nesse contexto que deverão ser também desenhadas as imagens geradas por computador.

Isso permite o uso dessa imagem capturada com as funções da GDI do Windows. Depois de construir o desenho virtual sobre a imagem da câmera, é preciso redesenhar a tela, exibindo a imagem. Isto pode ser feito de duas formas: a primeira é passar o *HDC* da imagem para a função *SwapBuffers* disponibilizada pela API do Windows. Esta função é responsável pela exibição do *frame buffer* na tela. Uma segunda maneira de exibir a imagem é através do método disponibilizado pela Vision SDK chamado *DisplayInHdc* do *swap*, com a diferença que essa rotina deve receber como parâmetros o contexto da janela onde a imagem será exibida. Para obter o contexto de dispositivo da janela, a API do Windows disponibiliza o método *GetDC*, que retorna o contexto da janela onde será repintada a imagem final.

A Figura 8.4 apresenta um pequeno trecho de código que serve para promover a integração entre MS Vision e a GDI do Windows. Ao final do código, há duas opções para atualizar o *frame buffer*, uma chamando rotinas da MS Vision e outra utilizando o método *SwapBuffers* da GDI. Há ainda um código que exibe um texto sobre a imagem.

A Figura 8.5 mostra a aplicação feita para testar o funcionamento utilizando as funcões da GDI do Windows, visualizando a imagem virtual o texto “Mouse”.

```
/*Variáveis globais*/
HDC hdc;
HWND hwnd; //handle para uma Window
CVisImageRGBAByteImage image;
/*Na função de callback (retorno de chamada)*/
case WM_CREATE:
hdc = GetDC(hwnd);
//USANDO-SE FUNÇÕES DA MS VISION SDK
/*insira aqui o trecho de código para desenhar imagens virtuais
  utilizando o HDC da janela como contexto*/
/*Imagem virtual*/
TextOut(hdc,80,250,”Mouse”,sizeof(“Mouse”));
image.DisplayInHdc(hdc);
//OU USANDO-SE FUNÇÕES DA GDI DO WINDOWS
/*insira aqui o trecho de código para desenhar imagens virtuais
  utilizando o HDC da imagem como contexto*/
/*Imagem virtual*/
TextOut(image.Hdc(),80,250,”Mouse”,sizeof(“Mouse”));
SwapBuffers(image.Hdc());
```

Figura 8.4 - Código mostrando os passos para combinar GDI e MS Vision

Figura 8.5 – aplicação rodando com a GDI.

8.3. Tratamento do problema da oclusão

Após a realização do processo de captura e composição das imagens, em alguns casos, é necessário efetuar a oclusão dos objetos virtuais pelos objetos reais pré-cadastrados no sistema, a fim de se obter a correta visualização do ambiente. Este processo deve ser preciso, tendo em vista o fato de que este trabalho manipula com objetos grandes e distantes.

Para tratar o problema de oclusão, utilizou-se a biblioteca gráfica OpenGL. Essa biblioteca possui vários recursos para criação, manipulação e eliminação de objetos gráficos. Fazendo-se uso do recurso chamado *Stencil Buffer*, é possível determinar áreas "protegidas" na imagem a ser exibida na tela [PIN02]. Essas áreas são demarcadas numericamente nesse *buffer* indicando porções reservadas da janela de desenho às quais nenhum objeto virtual pode sobrepor.

O *Stencil Buffer* é uma matriz bidimensional que possui o mesmo tamanho da janela de desenho usada para exibir os objetos do cenário a ser visualizado. Nesta matriz devem ser colocados valores que serão posteriormente testados para determinar se é possível, ou não, desenhar numa certa área da tela. A Figura 8.6 mostra como deve ser feita a inicialização do *Stencil* para seu futuro uso no programa.

```
/*Função da glut que reserva memória para a criação do Stencil Buffer*/
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH | GLUT_STENCIL);
glEnable(GL_STENCIL_TEST);   /*Habilita o uso do Stencil Buffer*/
glClearStencil(0);   /*Define que o valor 0 será utilizado para a
                       inicialização de todas as posições da matriz
                       do Stencil Buffer*/
glClear(GL_STENCIL_BUFFER_BIT);   /*Inicializa o Stencil Buffer*/
```

Figura 8.6 - Funções de inicialização do *Stencil Buffer*

Não há, entretanto, funções que permitam o acesso direto ao *Stencil Buffer*. O armazenamento de dados neste *buffer* é feito desenhando-se sobre a tela de exibição do OpenGL tendo-se previamente ativado funções de teste que irão colocar dados na área do *Stencil*, ou não, dependendo do resultado dos

mesmos. A partir disto, pode-se desenhar qualquer coisa na tela que o *Stencil Buffer* irá receber um novo valor nos pontos do *Stencil* correspondentes aos pontos ocupados pelo desenho na tela.

Para tanto, o OpenGL faz uso de duas funções, uma de “teste” e outra de “ação”. A primeira delas é a função de teste, *glStencilFunc*(Glenum *func*, GLint *ref*, Gluint *mask*), que define o tipo de teste, o valor de referência e a mascara para o teste do *Stencil* conforme ilustra a Tabela 8.1.

Tabela 8.1 - Parâmetros da função glStencilFunc [DOCMICROS]

Tipo de teste	**Significado**
GL_NEVER	Sempre retorna falso.
GL_LESS	Verdadeiro if (ref & mask) < (stencil & mask).
GL_LEQUAL	Verdadeiro if (ref & mask) ≤ (stencil & mask).
GL_GREATER	Verdadeiro if (ref & mask) > (stencil & mask).
GL_GEQUAL	Verdadeiro if (ref & mask) ≥ (stencil & mask).
GL_EQUAL	Verdadeiro if (ref & mask) == (stencil & mask).
GL_NOTEQUAL	Verdadeiro if (ref & mask) != (stencil & mask).

GL_ALWAYS

Sempre retorna verdadeiro.

Parâmetros:

func: Tipo de teste do *Stencil*.

ref: Valor de referência para o teste do stencil.

mask: Valor a ser escrito na posição atual do Stencil, com o qual o valor de referência e o valor da posição atual do Stencil são comparados.

A segunda é a função *glStencilOp*(Glenum *fail*, Glenum *zfail*, GLenum *zpass*), que define as ações a serem tomadas para cada um dos testes *fail*, *zfail* e *zpass*. A Tabela 8.2 mostra os *tokens* válidos para esses parâmetros.

Tabela 8.2 - Parâmetros da função *glStencilOp* [DOCMICROS]

Constante	**Ação**
GL_KEEP	Mantém o valor da posição atual do Stencil.
GL_ZERO	Escreve zero na posição atual Stencil.
GL_REPLACE	Escreve o valor de referência (especificado pela função glStencilFunc) na posição atual do Stencil.
GL_INCR	Incrementa o valor da posição atual do Stencil.

GL_DECR

Decrementa o valor da posição atual do Stencil.

GL_INVERT

Inverte (Bitwise) o valor da posição atual do Stencil.

Parâmetros:

fail: Ação a ser tomada caso o teste do Stencil (*glStencilFunc*) retorne falso.

zfail: Ação a ser tomada caso o teste do *Stencil* retorne verdadeiro, mas o teste do buffer de profundidade falhe.

zpass: Ação a ser tomada caso os dois testes retornem verdadeiro, ou caso o teste do *Stencil* retorne verdadeiro e o teste de profundidade (*zbuffer*) estiver desabilitado.

A Figura 8.7 demonstra o preenchimento do *Stencil* através do método *DefineStencil* implementado neste trabalho. A função de teste *glStencilFunc(tipoTeste, referência, novoValor)* (linha 7 da Figura 8.7) define qual será o teste feito com os valores da matriz do *Stencil* dependendo do parâmetro *tipoTeste*, os parâmetros *referência* e *novoValor*, retornando verdadeiro ou falso. Em caso verdadeiro, a função de operação *glStencilOp(Zmenor, Zigual, Zmaior)* (linha 8 da Figura 8.7) realiza **sempre** (parâmetro GL_ALWAYS, que sempre retorna verdadeiro) a **troca** (parâmetro GL_REPLACE) do parâmetro *referência* (valor atual no S*tencil*), pelo parâmetro *novoValor* para objetos virtuais com profundidade **menor**, **igual** ou **maior** (**primeiro**, **segundo** e **terceiro** parâmetros da função *glStencilOp*). A partir daí, qualquer tentativa de desenho sobre a tela OpenGL estará demarcando as regiões no *Stencil* ao invés. Conseqüentemente, a chamada do método *Render* (linha 11 da Figura 8.7) irá escrever novos valores no *Stencil Buffer*. Cabe ressaltar que os **objetos desenhados nesse momento, serão somente os *fantasmas* dos objetos reais previamente cadastrados em nosso sistema**, por isso a função S*etInStencil* (linha 10 da Figura 8.7) com parâmetro verdadeiro, fazendo com que somente sejam desenhados objetos que possuam o flag *inStencil* igual a *TRUE* (fantasmas, objetos a serem desenhados no *Stencil*).

```
1  void Stencil::DefineStencil(){
2         glEnable(GL_DEPTH_TEST); //Habilita o Depth Buffer
3         glDepthFunc(GL_LEQUAL);//Define o teste de profundidade
4         glEnable(GL_STENCIL_TEST);//Habilita o Stencil Buffer
5         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
6                 GL_STENCIL_BUFFER_BIT); //Limpa os Buffers
7         glStencilFunc(GL_ALWAYS,1,1);//Função de teste do Stencil
8         glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);//Operação
9         glPushMatrix();
10              rootObject->SetInStencil(TRUE);
11              rootObject->Render(rootObject);//Desenho dos objetos
12        glPopMatrix();
13 }
```

Figura 8.7 - Função para demarcar o *Stencil*

Após a demarcação do *Stencil*, é necessário desabilitá-lo temporariamente (linha 2 da Figura 8.8) para que se possa introduzir a imagem real capturada pela câmera. Por ser uma imagem bidimensional, também se deve desabilitar o uso de *buffer* de profundidade (linha 3 da Figura 8.8) bem como trocar o modo de visualização do OpenGL para 2D (linha 10 da Figura 8.8) a fim de obter sua correta exibição. Logo após sua exibição, deve-se retornar ao modo de visualização 3D (linhas 20 e 21 da Figura 8.8). A Figura 8.8 exibe esses procedimentos fazendo uso das funções de captura da câmera descritas na seção 8.2.1.

```
1   void Stencil::DrawBackgroundImageStencil(){
2         glDisable(GL_STENCIL_TEST);  //Desabilita o uso do Stencil Buffer
3         glDisable(GL_DEPTH_TEST);    //Desabilita o uso do Depth Buffer
```

```
4          glMatrixMode(GL_MODELVIEW);
5          glPushMatrix();              //Salva a matriz de desenho 3D
6                glLoadIdentity();
7                glMatrixMode(GL_PROJECTION);
8                glPushMatrix();      //Salva a matriz de projeção 3D
9                      glLoadIdentity();
10                     glOrtho(0,352,0,288,-1,1); /*Troca o modo de
11                                                visualização para 2D*/
12                     glStencilFunc(GL_ALWAYS,0,0);
13                     glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
14                     glPixelZoom(1.0,-1.0);  //Inverte a imagem
15                     glRasterPos2i(0, sizeOfCam.y-1); //Vincula os pixels
16                     //Desenha a imagem capturada pela câmera
17                      glDrawPixels(sizeOfCam.x,sizeOfCam.y,
18                                   GL_BGRA_EXT,GL_UNSIGNED_BYTE,
19                                   webCam->GetImage());
20               glPopMatrix();    //Retorna a matriz de projeção 3D
21               glMatrixMode(GL_MODELVIEW);
22         glPopMatrix();    //Retorna a matriz de desenho 3D
23   }
```

Figura 8.8 - Função de exibição das imagens reais.

Como última etapa do processo de composição da imagem final e tratamento do problema da oclusão, tem-se a introdução dos objetos virtuais no cenário final. A Figura 8.9 mostra a função responsável por esse procedimento. Primeiramente ocorre a reativação do *Depth Buffer* (linha 2 da Figura 8.9), pois novamente está se lidando com um contexto tridimensional. Por se tratar de objetos virtuais visíveis, também é necessária a mudança das funções de teste (linha 4 da Figura 8.9) e operação (linha 5 da Figura 8.9) do *Stencil*. O parâmetro GL_NOTEQUAL faz com que a função *glStencilFunc* retorne verdadeiro somente quando o valor atual do *Stencil* for **diferente** do parâmetro *referência*. Por sua vez, a função *glStencilOp* irá **manter** (parâmetro GL_KEEP) o valor do *Stencil* referente a região da tela analisada quando o teste anterior retornar verdadeiro. Essa lógica irá garantir que nada será desenhado sobre os *fantasmas* previamente cadastrados no *Stencil* através da função *DefineStencil*. Após essa consistência, uma nova chamada do método *Render* (linha 8 da Figura 8.9) irá então desenhar

somente os objetos virtuais visíveis ao usuário. Isso ocorre pela prévia chamada da função *SetInStencil* (linha 7 da Figura 8.9) agora com parâmetro falso.

```
1  void Stencil::DrawOutsideStencil(){
2          glEnable(GL_DEPTH_TEST);
3          glDepthFunc(GL_LEQUAL);
4          glStencilFunc(GL_NOTEQUAL,1,1);
5          glStencilOp(GL_KEEP,GL_KEEP,GL_KEEP);
6          glPushMatrix();
7                 rootObject->SetInStencil(FALSE);
8                 rootObject->Render(rootObject);
9          glPopMatrix();
10 }
```

Figura 8.9 - Função de desenho dos objetos virtuais

8.4. Ajuste dos parâmetros visuais do sistema

Para definir a posição de um usuário e o ângulo de visão do mesmo, a biblioteca de OpenGL disponibiliza as funções *gluLookAt* e *gluPerspective* respectivamente. Nas seções a seguir apresentam-se os métodos de obtenção de dados que servirão de parâmetro para estas duas funções.

8.4.1. Obtenção da posição do observador

Para que exista uma congruência entre a imagem da câmera e a imagem gerada pelo OpenGL, é necessário que se estabeleça uma relação posicional entre o observador virtual, definido pela função *gluLookAt(eyex, eyey, eyez, atx, aty, atz, 0, 1, 0)* da OpenGL, e a real posição do usuário no ambiente. Os três últimos parâmetros dessa função definem as coordenadas x, y e z do vetor de orientação vertical do observador respectivamente. Os três primeiros parâmetros, por sua vez, correspondem às coordenadas do observador virtual. Os parâmetros *eyex* e *eyez* são obtidos pelo dispositivo de GPS conforme descrito nas seções 4.2.2 e 9.5. O parâmetro *eyey* corresponde a altura da câmera em relação ao solo, e deve ser informado na inicialização do sistema para o correto posicionamento do observador no ambiente virtual.

8.4.2. Definição do aspecto visual

Após se definir a posição do observador no ambiente, é necessário saber para onde o usuário esta olhando, a fim de se estabelecer uma correspondência também desse fator. Para tanto, faz-se uso do dispositivo de rastreamento dos movimentos da cabeça (*head motion tracker*) presente no óculos de RV utilizado no trabalho. Este equipamento informa o ângulo de giro vertical e horizontal da cabeça, partindo de uma referência inicial. Os parâmetros atx, aty e atz da função *gluLookAt* são responsáveis por essa definição. Eles definem um ponto para o qual o observador virtual está olhando.

8.4.3. Obtenção da orientação da cabeça do observador

Para que se possa construir um cenário realista, evitando distorções visuais indesejadas, é necessário que o parâmetro *viewAngle* da função *gluPerspective* esteja bem definido. Ele deve corresponder ao ângulo de abertura horizontal da câmera, fazendo com que a composição da imagem real com a virtual fique perfeita. Para o cálculo desse ângulo construiu-se um cenário, sobre papel quadriculado, contendo duas colunas eqüidistantes da câmera, de forma a formarem um triângulo isósceles com a mesma (Figura 8.10).

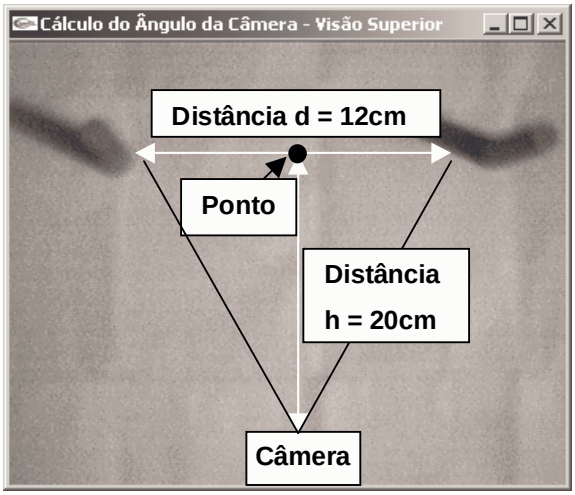

Figura 8.10 - Cálculo do Ângulo da Câmera (visão superior).

A partir disso, se aproxima ou se afasta a câmera a fim de fazer com que as colunas posicionem-se exatamente sobre as bordas da janela de

visualização da câmera. Esse processo deve ser efetuado com o cuidado de que a câmera e as duas colunas continuem sempre mantendo a formação de um triângulo isósceles. Também é importante garantir que a câmera esteja sempre mirando o ponto c. Deve-se agora medir a distância entre as colunas (d=12cm) e também a distância em que a câmera se encontra do ponto c (h=20cm). A partir daí, é possível obter o ângulo pela aplicação direta da seguinte fórmula:

viewAngle=2*tg(d/2*h), sendo d a distância entre as colunas e h a distância entre a câmera e o ponto c (Figura 8.10 – Vista superior).

8.4.4. Cálculo do campo de visão da câmera para a definição do ângulo de visão

Outro parâmetro a ser definido é o *aspectRatio*. Ele é responsável pela geração do aspecto visual, garantindo a devida proporção entre os objetos do cenário. Sua obtenção é um tanto trivial dependendo exclusivamente da resolução da câmera de captura das imagens. A câmera utilizada neste trabalho possui uma resolução de 352 pixels em largura por 288 pixels em altura. Logo o parâmetro *aspectRatio*, por ser uma relação entre as dimensões do plano de visualização, é obtido pela divisão 352/288.

9. Manipulação das mensagens do receptor GPS

Neste trabalho é necessário que se tenha um modo de fazer a recepção, tradução e conversão dos dados obtidos pelo aparelho de GPS. Esses passos são feitos através da utilização da classe GPS. Essa classe foi desenvolvida seguindo um modelo desenvolvido por Scuri, para parte de aquisição dos dados do GPS. Para conversão dos tipos de dados (coordenadas geodésicas e Universal Transversa de Mercator) foi usado um código distribuído de forma gratuita em http://164.214.2.59/GandG/geotrans/geotrans.html, pelo grupo *National Imagery and Mapping Agency* (NIMA). Ao longo das próximas seções são abordados alguns requisitos para obter a localização do usuário no globo terrestre.

9.1. Tipos de Coordenadas

Existem diversos tipos de coordenadas para a obtenção de uma posição na Terra. A classe GPS referida acima permite a manipulação de dois tipos de coordenadas, que são as coordenadas geodésicas e Universal Transversa de Mercator (UTM) [GOR01]. As coordenadas geodésicas (geográficas) são coordenadas que possuem informações sobre a latitude e longitude que definem pontos sobre a superfície da Terra, em relação ao elipsóide de referência adotado para representar a Terra. Já UTM é um sistema de coordenadas baseado em um sistema métrico. Ela divide a Terra em 60 fusos de 6º (zonas de seis graus), onde cada zona é dividida em uma projeção Transversa de Mercartor. A Figura 9.1 mostra como estão divididas as zonas no sistema de coordenadas UTM. Maiores informações sobre os tipos de coordenadas podem ser encontradas no *site* http://inside.uidaho.edu/tutorial/gis/glossary.asp, onde estão disponíveis uma série de definições a respeito disto. Informações sobre as coordenadas UTM podem ser acessadas pelo *site* http://www.cnr.colostate.edu/class_info/nr502/lg3/datums_coordinates/utm.html.

Nesse trabalho utiliza-se o sistema de coordenadas UTM tanto para cadastrar a posição dos prédios no módulo *GPS Register* como para atualizar a

posição do usuário no módulo *GPS Viewer*. A Figura 9.2 mostra as estruturas criadas para armazenar essas informações.

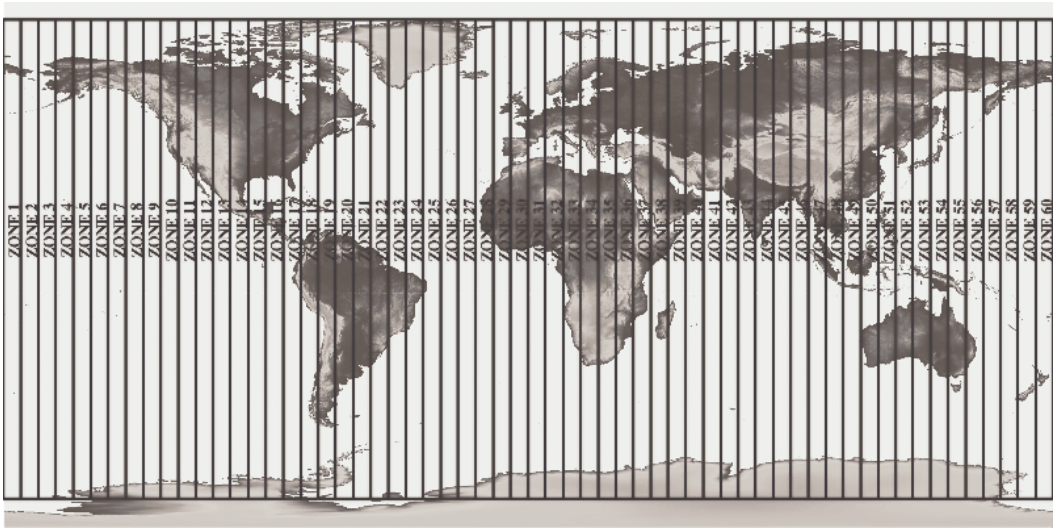

Figura 9.1 - Zonas usadas no sistema de coordenadas UTM

```
//Coordenadas Geodésicas
struct Geodetic_Projection {
       double longitude;
       double latitude;
};
//Coordenadas UTM
struct UTM_Projection {
       long zone;
       char hemisphere;
       double easting;
       double northing;
};
```

Figura 9.2 – Estruturas de armazenamento para coordenadas geodésicas e UTM

9.2. A classe GPS

A Tabela 9.1 apresenta os métodos da classe GPS. O uso dos métodos é descrito nas próximas seções.

Tabela 9.1 – A Classe GPS

Acesso	Método/Construtor/Destrutor	Descrição
Público	GPS	O construtor dessa classe recebe o nome da porta serial ao qual irá ser conectado ao GPS. Ele já abre uma conexão com o dispositivo. Neste construtor também são definidos alguns valores do sistema de referência WGS-84, que são o semi-eixo maior da elipsóide (6378137.000) e o achatamento do elipsóide (1/298.257223563). Neste trabalho colocou o valor zero, indicando sem zona de *override* (definição esta para utilizar na bilioteca Mgrs, da NIMA).
Público	~GPS	Destrutor da classe.
Privado	PrintError	Imprime um erro passado por parâmetros e fecha a conexão com a serial caso tenha dado algum problema com ela.
Público	GetUTMProjection	Retorna uma estrutura contendo os dados sobre a coordenada UTM, que são a zona (1..60), o hemisfério (Norte/Sul) e o quanto ao leste e norte o ponto está.
Público	SetCommPort	Abre a conexão com um dispositivo na serial. O nome da serial deve ser passada por parâmetros.
Público	ConvertGeodeticToUTM	Essa função converte coordenadas geodésicas para UTM (Universal Transversa de Mercator). Essa função faz chamada a duas funções da Mgrs, que são, respectivamente, Set_UTM_Parameters e Convert_Geodetic_To_UTM.
Público	StringToGeodetic	Método responsável pela conversão da *string* passada por parâmetros para coordenadas geodésicas. Essa *string* **deve** conter informações de posição.
Público	GetData	Este método é responsável pela leitura até que encontre o caracter delimitador de linha (‘\n’). Ela recebe por parâmetros se deseja imprimir erros na tela (show_err) o valor 1; e uma função de *callback*. Esta função permiti ao usuário a implementação de suas próprias rotinas de validação e processamento da *string* obtida do GPS.
Público	TestSentence	Método para validação dos dados. Retorna 1 se for válido, 0 caso contrário.
Público	ParseSentence	Este método faz a interpretação da *string* enviada pelo GPS. Retorna dados a respeito da *string* caso ela seja traduzida, caso contrário retorna NULL.

9.3. Leitura das mensagens

A leitura das mensagens é feita utilizando as funções de leitura da porta serial da API do Windows. Existem duas fases para fazer a leitura pela porta

serial: uma fase de “criação de uma conexão”, e a outra que é a leitura propriamente dita. O Anexo A-1 mostra os passos necessários para utilizar a API do Windows para ler dados da serial.

Para a leitura, a classe GPS possui um método chamado *GetData*, que faz a leitura dos dados advindos da serial e guarda o dado lido numa *string*. Após armazenada, essa *string* é enviada para uma função de *callback* passada por um dos parâmetros da função *GetData*. Essa função dá ao usuário a possibilidade de definir suas próprias rotinas de validação, interpretação e processamento das mensagens obtidas pela leitura do dispositivo.

9.4. Tradução e conversão dos dados

Após ter criado a conexão e feita a leitura de uma linha, resta agora fazer a tradução da *string* capturada, em coordenadas geodésicas e, por fim, transformar estas coordenadas em coordenadas UTM, utilizada no presente trabalho.

O receptor GPS é capaz de enviar vários tipos de informações, entretanto, a única informação utilizada neste trabalho é a de posição. Essa *string* contém informações sobre latitude e longitude. A Figura 9.3 mostra um exemplo da string enviada pelo receptor GPS, enquanto que a Tabela 9.2 ilustra o formato da mensagem utilizada neste trabalho. O formato das demais *strings* geradas pelo GPS podem ser obtidos em http://www.commlinx.com.au/tf30gps.pdf.

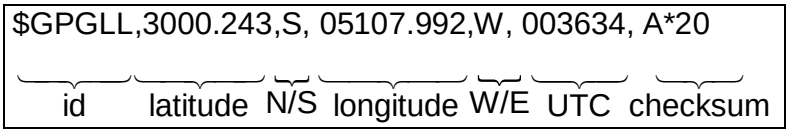

Figura 9.3 – Exemplo da string obtida pelo receptor GPS

Tabela 9.2 - Formato da mensagem de posição

Nome	Exemplo	Descrição
Id da mensagem	$GPGLL	Cabeçalho GLL (Global Latitude/Longitude)
Latitude	3000.243	hdddmm.mmm (degree and minutes notation). hd = grau opcional. d = grau obrigatório. m = minuto. No caso de latitude o hd não é preenchido.
Indicador N/S	S	N = North ou S = South
Longitude	05107.992	hdddmm.mmm (notação em graus e minutos). hd = grau opcional. d = grau obrigatório. m = minuto.
Indicador W/E	W	W = West ou E = East
Posição UTC	003634	Hhmmss. Esta informação não é utilizada neste trabalho.
Status	A	A = dado válido ou V = dado não válido
CheckSum	*20	
<Carriage Return><line feed>	'\r\n'	Término da mensagem

Essa mensagem possui como delimitador dos dados o caractere "*vírgula*". A quebra dessa *string* foi feita através da criação de uma classe chamada StringTokenizer, que tem como objetivo retornar *tokens* separado por um delemitador que, neste trabalho, foi o caractere vírgula. Através dessa classe, é possível fazer a tradução das mensagens enviadas pelo receptor GPS. Os dados a respeito da latitude e longitude são armazenados na estrutura de dados Geodetic_Projection, apresentado na Figura 9.2.

A partir da obtenção da coordenada geodésica, falta convertê-la para coordenadas UTM. Como mencionado no início deste capítulo, para realizar a conversão dos dados, foram utilizados alguns arquivos da biblioteca **Mgrs**, da *National Imagery and Mapping Agency* (NIMA). Neste trabalho, necessita-se apenas converter de geodésicas para UTM, portanto, utilizou-se apenas o código necessário para fazer esta conversão.

A classe GPS possui um método chamado *ConvertGeodeticToUTM*, que faz chamadas às funções da biblioteca Mgrs, que são, respectivamente, *Set_UTM_Parameters* e *Convert_Geodetic_To_UTM*. A primeira recebe por parâmetro informações a respeito do elipsóide que representa a Terra, enquanto a segunda serve para realizar a conversão de coordenadas geodésicas para UTM, baseada no elipsóide setado anteriormente.

A função *Set_UTM_Parameters* recebe por parâmetros o semi-eixo maior do elipsóide e o achatamento. Como citado na seção 7.4, o sistema de referência sobre o elipsóide é baseado no sistema **WGS-84**. Nele, o valor do semi-eixo maior equivale a 6378137.000 metros, enquanto que o achatamento do elipsóide equivale a 1/298.257223563.

Já a função *Convert_Geodetic_To_UTM* recebe por parâmetros as coordenadas geodésicas (latitude e longitude, respectivamente) e retorna os dados referentes a coordenada UTM, que são, respectivamente, zona, hemisfério, o quanto se está a leste e quanto se está ao norte. Estes dados são armazenados na estrutura de dados UTM_Projection, como mencionada na seção 9.1.

9.5. Validação dos dados

Algumas vezes os dados lidos da serial podem não vir corretamente pois o aparelho de GPS pode não estar recebendo o sinal de quatro satélites, ou ainda estar lendo algum dado incorreto. Para tanto, na classe GPS existem um método para garantir a validade dos dados, chamada TestSentence. Este método recebe por parâmetros a *string* capturada e o tamanho dessa *string*. Ele verifica se existe na primeira posição da string o caractere “$”, e se nas duas últimas posições existem os caracteres “\r\n”, que são, respectivamente, *carriage return* e *line feed*. Se não houver erro, esta função retorna o valor 1, senão retorna 0.

Existe ainda mais um método que também pode ser utilizado para consistência dos dados, desenvolvido na classe, que é o método ParseSentence. Essa função retorna NULL se não conseguir fazer a interpretação da sentença, caso contrário, ela retorna uma estrutura de dados contendo informações sobre a sentença. Dentre as informações nela contida, existe uma chamada Check, que serve para validar o checksum da *string*. Se Check contiver o valor diferente de -1, significa que possui *checksum* na *string*, caso contrário significa que há algum dado inválido no meio da *string*. Se Check tiver o valor 0, significa que o *checksum* é inválido, senão se tiver o valor 1, é validada a *string*.

10. Cenário gerado

A aplicação desenvolvida nesse trabalho permite que se crie um ambiente de RA através do cadastro de construções e suas respectivas informações pelo módulo *GPS Register* já descrito. A fim de validar e testar o desempenho dessa aplicação gerou-se um ambiente para a simulação da mesma.

O cenário é composto por três prédios da PUCRS de Porto Alegre, sobre os quais foram adicionadas placas informativas contendo seus respectivos números. Um usuário pode caminhar ao redor dos mesmos observando as placas numeradas quando seu ângulo de visão assim o permitir. A Figura 10.1 apresenta um mapa deste cenário contendo os prédios com suas respectivas placas virtuais e um ponto A indicando a localização do usuário no ambiente com uma seta demonstrando sua orientação visual na situação.

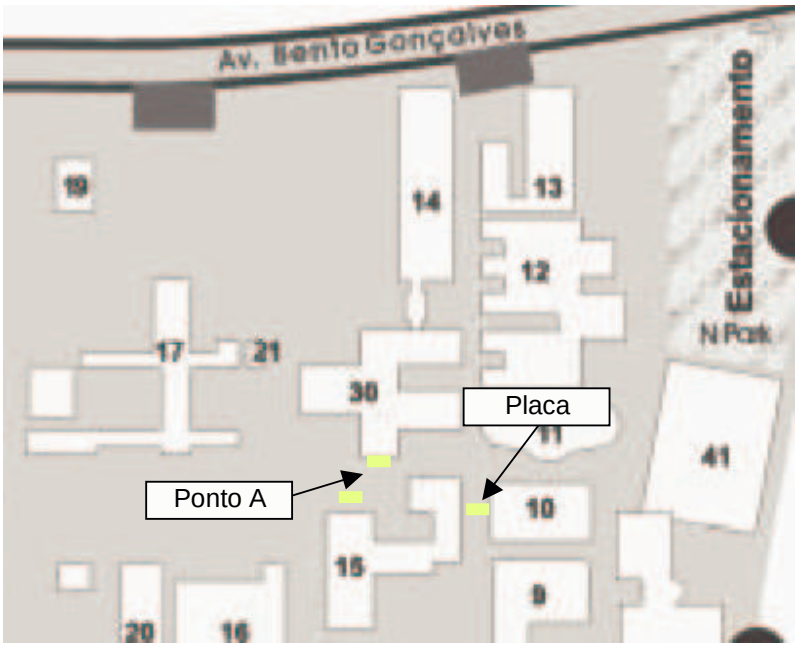

Figura 10.1 - Mapa do cenário vista superior (PUCRS).

Um usuário posicionado no ponto A consegue enxergar a placa do prédio 15 e uma parte da placa do prédio 10, devido ao processo de oclusão imposto pelo fantasma do prédio 15. Este usuário possui uma visualização ilustrada na Figura 10.2.

Figura 10.2 - Visão do usuário no ponto A.

Nesta cena é possível observar que a placa correspondente ao prédio 15 esta sofrendo oclusão por parte do mesmo e por isso aparece “cortada” na imagem. A placa relativa ao prédio 30 aparece à esquerda na frente da escadaria do mesmo.

11. Conclusões

Neste trabalho desenvolveu-se uma ferramenta que permite ao usuário visualizar informações virtuais a respeito de pontos específicos de uma cidade, tais como prédios, igrejas, monumentos e outras construções, fazendo uso das técnicas de RA. Com a utilização desse sistema, o usuário pode obter informações a respeito de um determinado local através da adição de placas virtuais.

Durante o desenvolvimento da mesma, ratificou-se a importância da aplicação de uma técnica de registro eficiente para o posicionamento dos objetos. Além disso, a modelagem das réplicas virtuais (fantasmas) deve ser a mais parecida possível com as construções sobre as quais se desejam adicionar as informações. No que tange aos modelos virtuais, procurou-se aproximar a arquitetura dos objetos reais através da modelagem de blocos dos mesmos. É importante destacar que, a utilização de modelos mais precisos melhorariam a qualidade do processo de oclusão sem alterações no projeto final.

Outro fator limitante na qualidade do registro dos objetos reais neste trabalho foi o baixo grau de precisão imposto pelo equipamento de GPS. Isto gerou oscilações indesejáveis da posição do usuário e no registro dos prédios cadastrados no sistema, causando uma ligeira incoerência visual na imagem aumentada. Da forma como foi concebido este projeto, a utilização de um equipamento de GPS mais preciso resolveria esse problema sem nenhuma alteração no código. No que diz respeito aos prédios cadastrados, um mapa contendo as coordenadas geodésicas dos mesmos, poderia ser facilmente incorporado ao sistema eliminando a necessidade da utilização do GPS para o registro das construções.

Outro fator relevante para o correto desenvolvimento de um sistema de RA é a correta obtenção dos parâmetros visuais da câmera utilizada para a captura das imagens reais. Isto se mostrou essencial para o correto posicionamento das imagens virtuais. O método empregado pra este fim mostrou-se suficientemente preciso. Além disso, em ambientes internos é

necessária uma outra forma de rastreamento, pois a utilização de receptores GPS necessita de uma área aberta para que o equipamento obtenha um mínimo de quatro satélites para a determinação da posição.

Devido às restrições citadas acima, a qualidade de interação com o sistema ficou um pouco prejudicada. Todavia, isto não invalidou o desenvolvimento do trabalho, permitindo a realização de testes e a avaliação de todos os aspectos envolvidos em um projeto deste gênero.

Anexo A - Leitura da serial usando a API Win32

Esta seção tem como objetivo mostrar os passos que devem ser realizados para adquirir os dados advindos do aparelho de GPS. A leitura das mensagens é feita utilizando as funções de leitura da API do Windows. Existem duas fases para fazer a leitura pela porta serial: uma fase de “criação de uma conexão” através da porta serial, e a outra que é a leitura propriamente dita.

Para fazer a conexão e leitura dos dados recebidos pela serial, é necessário ter um ponteiro para um dispositivo de comunicação (*HANDLE*). Este ponteiro serve para depois orientar em qual porta estão sendo recebidas as informações.

O acesso a uma determinada porta é feito através do método *CreateFile*, que retorna um *handle* para poder acessar um determinado objeto (que no caso é a porta serial). Apesar de o nome dessa função soar estranho, esta função pode ser utilizada para arquivos, *pipes*, *mailslots*, recursos de comunicação, dispositivos de discos (somente no Windows NT), *console* e diretórios somente para leitura.

Esta função recebe por parâmetros uma *string* que descreve o nome da porta na qual estão sendo recebidos os dados, o tipo de acesso ao objeto, seus modos de compartilhamento; um ponteiro contendo informações sobre atributos de segurança, o tipo de ação que deve ser tomado em objetos que já ou não existam; algumas *flags* para o objeto criado e, por fim, um ponteiro para um modelo de arquivo com atributos de somente leitura.

No caso de dispositivos de comunicação, os seguintes parâmetros devem ser descritos. O nome da porta que deve ser COMX, onde X é o número da porta correspondente à entrada dos dados enviados pelo aparelho de GPS. O tipo de acesso a porta que deve ser GENERIC READ | GENERIC_WRITE. O modo de compartilhamento, no qual deve-se passar o valor “Zero”, pois ele não é compartilhado. Atributos de segurança, especificado como NULL, pois não era necessário atributo algum. Com relação a ação que deve ser tomada, utilizou-se o

valor OPEN_EXISTING, pois caso não haja nenhum GPS conectado, a função *CreateFile* falha. Com relação as *flags* setadas, passou-se o valor “Zero”, pois não se pode sobrepor as flags para dispositivos de I/O. Por fim, o ultimo parâmetro foi setado o valor NULL, pois para dispositivos de comunicação este parâmetro deve ser nulo.

Obtido um ponteiro para o dispositivo de comunicação, é necessário definir os parâmetros de controle para o dispositivo de comunicação serial como por exemplo a taxa de transmissão dos dados e a configuração do *stop bit*. Para tanto, a API do Windows oferece uma estrutura de dados chamada DBC (*Device Control-Block*). A definição da configuração de controle da serial é feita através dos métodos *GetCommState* e *SetCommState* que, respectivamente servem para preencher no DBC os valores da porta definidos no aparelho que faz a comunicação dos dados; e setar os valores de controle para a serial.

Tendo o ponteiro para o dispositivo da serial e definidos os controles dela, falta agora é definir o *timeout* de leitura dos dados da serial. Na API do windows existe uma estrutura especial chamada COMMTIMEOUTS, que define os *timeouts* de leitura/escrita para dispositivos de comunicação. Assim como os métodos existentes na definição de controle, existem as funções *GetCommTimeout* e *SetCommTimeout*, que servem, respectivamente, para capturar os valores iniciais e setar o tempo de acesso para leitura/escrita no dispositivo.

Ao final desses passos são feitas mais duas chamadas para procedimentos de “limpeza” de erros e estados do dispositivo serial. Esses métodos são, respectivamente, *ClearCommErrors* e *PurgeComm*. Na primeira função passou-se no último parâmetro o valor NULL, pois não se deseja processar sobre as informações de erros. Em *PurgeComm*, estão sendo descartados todos os caracteres de entrada/saída do *buffer* do dispositivo de comunicação (caso tenha um) e são eliminadas todas as operações de escrita/leitura no dispositivo. A Figura A.1 mostra o código necessário para a criação de comunicação com o dispositivo na porta COM1.

```
HANDLE hCom;
DCB dcb;
COMMTIMEOUTS commtime;
DWORD dSize;
hCom = CreateFile(“COM1”, GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);
if (hCom == INVALID_HANDLE_VALUE) printf("erro em CreateFile");
GetCommState(hCom, &dcb);
dcb.DCBlength = sizeof(DCB);
dcb.BaudRate = CBR_9600;
dcb.fParity = FALSE;
dcb.Parity = NOPARITY;
dcb.ByteSize = 8;
dcb.StopBits = ONESTOPBIT;
dcb.fBinary = TRUE;
dcb.fAbortOnError = TRUE;
if (!SetCommState(hCom, &dcb)) printf("erro na função SetCommState");
GetCommTimeouts(hCom, &commtime);
//setando apenas timeout de leitura
commtime.ReadIntervalTimeout = MAXDWORD;
commtime.ReadTotalTimeoutMultiplier = MAXDWORD;
commtime.ReadTotalTimeoutConstant = 2000;  // 2 second time out
if (!SetCommTimeouts(hCom, &commtime)) printf("erro na função SetCommTimeouts");
ClearCommError(hCom, &dSize, NULL);
PurgeComm(hCom, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR | PURGE_RXCLEAR);
```

Figura A.1 – Criação de conexão com o dispositivo de comunicação.

Feita a conexão com o dispositivo de comunicação, o próximo passo é fazer a leitura dos dados advindo do GPS conectado a essa porta. A API do Windows possui o método ReadFile, que lê um dado de um certo contexto (no caso um ponteiro para um dispositivo). Maiores informações sobre este método podem ser encontrados em http://msdn.microsoft.com/library/default.asp.

Na classe GPS desenvolvida para este trabalho, desenvolveu-se uma lógica através do qual são lidos os dados advindos da serial até que se encontre uma quebra de linha (informação esta armazenada num *buffer*). A Figura A.2 mostra o método desenvolvido para fazer a leitura dos dados enviados pelo receptor GPS.

```
int GPS::GetData(int show_err, int (*callback_function)(char* buffer, int size)) {
	DWORD dSize;
	char buf[BUFFER_SIZE] = " ";
	char *buffer;
	ClearCommError(hCom, &dSize, NULL);
	PurgeComm(hCom, PURGE_TXABORT|PURGE_RXABORT|PURGE_TXCLEAR|PURGE_RXCLEAR);
	// Le dados até que a função de callback retorne 0
	buffer = buf+1; // Permite acesso ao buffer[c-1] quando c==0
	do {
		int ret, c = 0, err_count = 0;
		// Lê até que encontre uma quebra de linha
		do {
			dSize = 0;
			ret = ReadFile(hCom, buffer + c, 1, &dSize, NULL);
			// Trata problemas de comunicação
			if (!ret || dSize != 1) {
				ClearCommError(hCom, &dSize, NULL);
				PurgeComm(hCom, PURGE_TXABORT | PURGE_RXABORT |
					PURGE_TXCLEAR | PURGE_RXCLEAR);
				err_count++;
				if (err_count == 5) {
					printf(“limite de erro alcancado”);
					exit(0);
				}
				else if (show_err) {
					if (!ret) printf("Warn: read\n");
					else printf("Warn: size\n");
				}
			}
			else { err_count = 0;  c++; }
		} while(buffer[c-1] != LINE_FEED && c < BUFFER_SIZE);
		buffer[c] = 0; dSize = c;
	} while (callback_function(buffer, dSize) != 0);
	return 1;
}
```

Figura A.2 – Trecho de código do método de leitura da classe GPS

Anexo B - Definições sobre o protocolo do GPS

Os valores das flags podem ser associados com os seguinte valores mostrados a seguir:

Identificadores Básicos do Pacote

```
enum {
      Pid_Ack_Byte = 6,
      Pid_Nak_Byte = 21,
      Pid_Protocol_Array = 253, /* Pode não estar implementados em todos GPS´s */
      Pid_Product_Rqst = 254,
      Pid_Product_Data = 255
};
```

L001 – Protocolo da camada de link 1

Este protocolo de *Link* é usado pela maioria dos aparelhos de GPS. Este protocolo tem o mesmo conteúdo de L000 (*Basic Link Protocol*), exceto que ele possui os seguintes identificadores a mais (em negrito).

```
enum {
      Pid_Command_Data = 10,     /*Contém dado que indica um comando, provido por
                                 Command_Id_Type*/
      Pid_Xfer_Cmplt = 12,
      Pid_Date_Time_Data = 14,
      Pid_Position_Data = 17,    /*Contém o dado de posição, que é provido em
                                 específico produto de tipo de dados <D0>*/
      Pid_Prx_Wpt_Data = 19,
      Pid_Records = 27,
      Pid_Rte_Hdr = 29,
      Pid_Rte_Wpt_Data = 30,
      Pid_Almanac_Data = 31,
      Pid_Trk_Data = 34,
      Pid_Wpt_Data = 35,
      Pid_Pvt_Data = 51,
      Pid_Rte_Link_Data = 98,
      Pid_Trk_Hdr = 99
};
```

A010 – Protocolo de comandos do dispositivo de GPS 1 (Command_Id_Type)

Este protocolo é implementado pela maioria dos aparelhos de GPS. O Command_Id_Type é um inteiro que indica um comando particular. O valores associados à Command_Id_Type são mostrados abaixo:

```
typedef int Command_Id_Type;
enum {
      Cmnd_Abort_Transfer = 0,  /* abort current transfer */
      Cmnd_Transfer_Alm = 1,    /* transfer almanac */
      Cmnd_Transfer_Posn = 2,   /* transfer position */
      Cmnd_Transfer_Prx = 3,    /* transfer proximity waypoints */
      Cmnd_Transfer_Rte = 4,    /* transfer routes */
      Cmnd_Transfer_Time = 5,   /* transfer time */
      Cmnd_Transfer_Trk = 6,    /* transfer track log */
      Cmnd_Transfer_Wpt = 7,    /* transfer waypoints */
      Cmnd_Turn_Off_Pwr = 8,    /* turn off power */
      Cmnd_Start_Pvt_Data = 49, /* start transmitting PVT data */
      Cmnd_Stop_Pvt_Data = 50   /* stop transmitting PVT data */
};
```

Nota: O comando “Cmnd_Turn_Off_Pwr” pode não ser reconhecido por alguns aparelhos GPS.

Anexo C - Manual do usuário

Este manual tem como objetivo auxiliar um usuário a utilizar o sistema desenvolvido. O documento está dividido em três partes, que são respectivamente o módulo GPS Register, o módulo GPS Viewer e como usar esse sistema em outros locais, diferentes daqueles onde o sistema foi testado.

C.1 O módulo GPS Register

O módulo GPS Register tem como objetivo gerar um arquivo contendo informações sobre todos os prédios cadastrados, com seus respectivos textos (imagens virtuais) associados.

A execução desse módulo pode ser feita de duas formas: entre no diretório onde está o arquivo GPSRegister.exe, e dê um duplo clique no ícone do programa executável ou abra um *console* e digite GPSRegister e clique <ENTER>. Desse modo é aberta a janela inicial desse módulo. A Figura C.1 ilustra a tela inicial quando do programa.

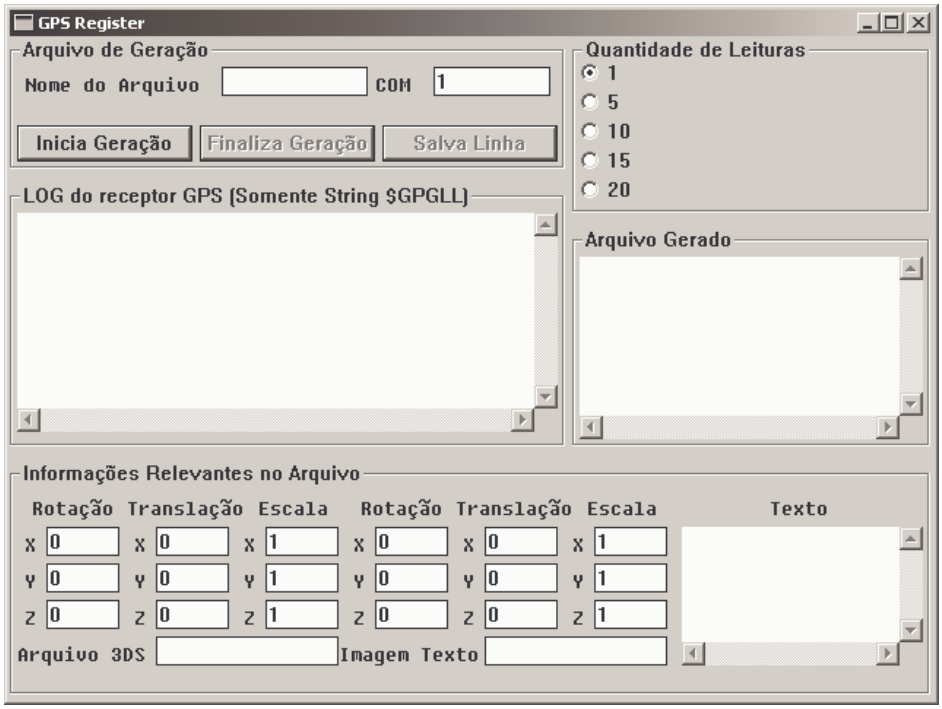

Figura C.1 – Tela Inicial do *GPSRegister*.

O GPSRegister somente inicia a leitura das mensagens do receptor GPS quando o usuário clicar no botão **`Inicia Geração`** e que tenham sido preenchido os campos com o nome do arquivo e a porta de comunicação em que está conectado o cabo ligado ao GPS (`COM1, COM2,...`). Caso algum desses campos não seja preenchido, uma mensagem de erro é mostrada e o programa não inicia a leitura dos dados vindos pela serial.

Caso os campos tenham sido preenchidos corretamente, o programa inicia lendo as mensagens do GPS, atualizando constantemente, na tela, a string que contém a informação de posição. A Figura C.2 mostra algumas das mensagens de posição obtidas após o início do programa, no campo `Log` do `receptor GPS` (Somente a string $GPGLL).

```
$GPGLL,3003.604,S,05110.414,W,003634,A*20
$GPGLL,3003.605,S,05110.414,W,003634,A*20
$GPGLL,3003.603,S,05110.414,W,003634,A*20
$GPGLL,3003.604,S,05110.413,W,003634,A*20
```

Figura C.2 – *GPSRegister* Lendo dados do GPS.

A gravação (**registro**) de uma determinada posição é feita quando for clicado no botão **`Salva Linha`**. Essa ação tem como pré-requisito que todos os campos referentes às transformações geométricas (`translação, escala e rotação`) estejam devidamente preenchidas, além de os campos de `Arquivo 3DS` e `Imagem Texto/Texto`. Além desses dados, o arquivo somente é gerado após o programa ter feito o número de leituras definida na opção `Quantidade de Leituras`. Ao salvar uma linha, o programa mostra um *preview* do arquivo gerado, dado este mostrado no campo `Arquivo Gerado`.

O número de leituras que pode ser alterado pelo usuário serve para que possa melhor determinar a posição enviada pelo GPS. Isso passa a ser interessante quando se deseja melhor registrar as construções virtuais (fantasmas) e suas respectivas placas que estão associadas aos prédios, visto que quando utilizados modelos de GPS com uma precisão não muito alta, como o usado neste trabalho, que tinha um erro de 7 a 10m. O término das leituras pode

ser realizado de duas formas que são, respectivamente, ao fechar a janela ou quando clicar no botão `Finaliza Leitura`.

C.2 O módulo GPS Viewer

O módulo GPS Viewer é o módulo principal do sistema, onde é responsável pela exibição das imagens resultadas da combinação das imagens virtuais com as reais. Esse módulo necessita que tenham sido instaladas as seguintes DLLs: MSVCRTD.dll, MSVCP60D.dll, VisCore.dll, VisCoreDB.dll, VisDisplay.dll, VisDisplayDB.dll, VisImSrc.dll, VisImSrcDB.dll, VisVFWCamera.dll, VisVFWCameraDB.dll (dlls do Visual C e da MSVision, respectivamente) e glut32.dll (dll da glut).

Para executar o módulo GPS Viewer, abrir um *console* e digitar: `gpsviewer <com_do_HMD> <com_do_GPS> <arquivo_do_register>`. É importante salientar que as portas seriais associadas a cada aparelho devem ser corretamente digitadas, visto que o programa inicia a leitura pelas portas mencionadas na linha de comando. Caso o arquivo informado não exista ou for digitado um nome inválido, o programa encerra sua execução.

É importante salientar que no momento da execução do sistema o usuário deve estar olhando para o norte, para logo após *resetar* o sistema de coordenadas que é atualizado pelos valores obtidos do HMD. Para reiniciar o sistema de coordenadas, basta pressionar a tecla ‘r’. Por fim, o programa fica em execução até que o usuário do sistema clique na tecla ‘ESC’.

C.3 Uso dos módulos em outros ambientes (fora da PUCRS)

Neste trabalho geraram-se testes somente dentro campus da PUCRS em Porto Alegre, entretanto, caso deseje-se utilizar esse sistema em uma outra região ou local, pode-se obter os dados a respeito de construções tanto através de algum órgão publico ou centro de geoprocessamento que possua os

dados a respeito da posição dos locais de interesse. Além disto, pode-se usar o GPS Register na rua para o registro das construções, necessitando apenas que esteja utilizando um *notebook* para geração do arquivo que será utilizado na execução do GPS Viewer. Por fim, também é interessante ter um programa para gerar os modelos tridimensionais das construções requeridas, modelos esses que são utilizados como fantasmas em nossas construções.

Referências Bibliográficas

[ART00] ARTOOLKIT – Augmented Reality Toolkit. Intelligence, Agents and Media Group - University of Southampton, 2000. http://www.equator.ecs.soton.ac.uk/projects/artoolkit/.

[AZU97] T. Azuma, Ronald. A Survey of Augmented Reality, UNC Chapel Hill, 1997.

[BAJ92] Bajura, Mike, Henry Fuchs, and Ryutarou Ohbuchi. Merging Virtual Reality with the Real World: Seeing Ultrasound Imagery Within the Patient. Proceedings of SIGGRAPH '92 (Chicago, IL, 26-31 July 1992). In Computer Graphics 26, 2 (July 1992), 203-210.

[BAL00] Selim Balcisoy, Marcelo Kallmann. A famework for rapid evaluation of prototypes with Augmented Reality. 2000.

[BIL98] Billinghurst, M., Bowskill, J., Jessop, M., Morphett, J. A wearable Spatial Conferecing Space. 1998.

[DOCMICROS] http://msdn.microsoft.com/library/default.asp. Maio de 2003.

[DRA96] Drascic, David, P. Milgran. Perceptual Issues in Augmented Reality. Feb,1996. http://vered.rose.utoronto.ca/people/david_dir/SPIE96/SPIE96.full.html

[DRA93A] Drascic, D., J.J. Grodski, P. Milgram, K. Ruffo, P. Wong, and S. Zhai. ARGOS: A Display System for Augmenting Reality. Video Proceedings of INTERCHI '93: Human Factors in Computing Systems (Amsterdam, the Netherlands, 24-29 April 1993). Also in ACM SIGGRAPH Technical Video Review, Volume 88. Extended abstract in Proceedings of INTERCHI '93, 521.

[FEI93a] Feiner, Steven, Blair MacIntyre, and Dorée Seligmann. Knowledge-based Augmented Reality. Communications of the ACM 36, 7 (July 1993), 52-62. http://www.cs.columbia.edu/graphics/projects/karma/karma.html

[FEI93b] Feiner, Steven, Blair MacIntyre, Marcus Haupt, and Eliot Solomon. Windows on the World: 2D Windows for 3D Augmented Reality. Proceedings of UIST '93 (Atlanta, GA, 3-5 November 1993), 145-155. http://www.cs.columbia.edu/graphics/publications/uist93.pdf

[GAR99] Garmin GPS interface protocol. December 6,1999. http://www.garmin.com.

[GARMIN] Garmin: What is Garmin. http://www.garmin.com/aboutGPS/. Maio 6, 2003.

[GOR01] Gorgulho, Miguel. GPS Trackmaker. GPS – O sistema de posicionamento global. Julho, 2001. http://gpstm.com/port/apostila_port.htm

[JAN93] Janin, Adam L., David W. Mizell, and Thomas P. Caudell. Calibration of Head-Mounted Displays for Augmented Reality Applications. Proceedings of IEEE VRAIS '93 (Seattle, WA, 18-22 September 1993), 246-255.

[KIM96] Kim, Won S. Virtual Reality Calibration and Preview / Predictive Displays for Telerobotics. Presence: Teleoperators and Virtual Environments 5, 2 (Spring 1996), 173-190.

[LAN98] Lanier, Jaron. A National Tele-Immersion Initiative. 1998. http://www.howstuffworks.com/framed.htm?parent=holographic-environment.htm&url=http://www.advanced.org/teleimmersion.html

[MIC02] Microvision. Augmented Reality. http://www.mvis.com. January, 2002.

[MSVISION] Microsoft Vision SDK 1.2. http://research.microsoft.com/projects/VisSDK. Maio 6, 2003.

[OGL03] OpenGL - The Industry's Foundation for High Performance Graphics. http://www.opengl.org/. Maio 6, 2003.

[PIE02] Piekarski, Waine, Dr Bruce Thomas. School of Computer and Information Science Advanced Computing Research Centre. University of South Australia. July, 2002. http://www.tinmith.net/wearable.htm

[PIN02] Márcio Serolli Pinho, www.inf.pucrs.br/~pinho/TCG/ApoioTCG.htm. PDFs 3,4 e 9; Mascaramento de Regiões Usando *Stencil Buffer*. http://www.inf.pucrs.br/~pinho/CG/Aulas/OpenGL/Mascaramento/Stencil.html

[POA02] Imagens de Porto Alegre. http://nutep.adm.ufrgs.br/fotospoa/fotospoa.htm. Porto Alegre. 2002.

[REK96] Jun Rekimoto. Augmented Interaction: The World Through the Computer. 1996. http://www.csl.co.jp/person/rekimoto/navi.html.

[ROS95] Rose, Eric, David Breen, Klaus Ahlers, Chris Crampton, Mihran Tuceryan, Ross Whitaker, and Douglas Greer. Annotating Real-World Objects Using Augmented Reality. Proceedings of Computer Graphics International '95 (Leeds, UK, 25-30 June 1995), 357-370. http://www.cs.iupui.edu/~tuceryan/research/AR/ECRC-94-41.pdf

[RS232] O Protocolo RS-232. http://www.ctips.com/rs232.html. 10 de Maio de 2003.

[SAB99] Sabbatini, Renato M. E. Informática Médica. Volume Número 2. Abril/maio-99. http://www.epub.org.br/informaticamedica/n0202/sabbatini.htm

[SCH01] Dieter Schmaltieg. An introduction to Augmented Reality. Vienna University of Tecnology. Austria. 2001. http://www.ims.tuwien.ac.at/teaching/vr/folien/ar_intro.pdf

[SINC02] Sinclair, Patrick. The Southampton Equator Project - Adaptive Hypermedia in Augmented Reality; (AR Makers demo). http://www.equator.ecs.soton.ac.uk/demos/. September, 2002.

[SMALLVR] SmallVR - A Simple toolkit for VR Application Development. http://www.smallvr.org/. Maio 6, 2003.

[STA96b] State, Andrei, Mark A. Livingston, Gentaro Hirota, William F.Garrett, Mary C. Whitton, Henry Fuchs and Etta D. Pisano. Techniques for Augmented-Reality Systems: Realizing Ultrasound-Guided Needle Biopsies. Proceedings of SIGGRAPH '96 (New Orleans, LA, 4-9 August 1996), 439-446.

[STR01a] Stricker, Didier. ARCHEOGUIDE: First results of an Augmented Reality, Mobile Computing System in Cultural Heritage Sites. , Virtual Reality, Archaeology, and Cultural Heritage International Symposium (VAST01), Glyfada, Nr Athens, Greece, 28-30 November 2001. http://archeoguide.intranet.gr/publications.htm

[STR01b] Stricker, Didier. Virtual Reality and Information Technology for Archaeological site promotion. November, 2001. http://archeoguide.intranet.gr/publications.htm

[SZA98] Zsolt Szalavári. Mah-Jongg – A Collaborative Gaming In Augmented Reality, 1998. http://www.cg.tuwien.ac.at/research/vr/gaming/mah-jongg/

[TRI02] Trimble Navigation Limited. 2002. http://www.trimble.com/gps/