SANCHES, Silvio Ricardo Rodrigues

A Utilização da Técnica de Chromakey para Composição de Cenas em Ambientes de Realidade Misturada / Silvio Ricardo Rodrigues Sanches; Orientador: Antonio Carlos Sementille. Marília, SP, 2007.

150 f.

Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília - Fundação de Ensino Eurípides Soares da Rocha.

1.Realidade Misturada 2.ARToolkit 3.*Chromakey* 4.Oclusão Mútua

CDD: 006

FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" - UNIVEM
PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SILVIO RICARDO RODRIGUES SANCHES

A UTILIZAÇÃO DA TÉCNICA DE CHROMAKEY PARA COMPOSIÇÃO DE CENAS EM AMBIENTES DE REALIDADE MISTURADA

Marília
2007

SILVIO RICARDO RODRIGUES SANCHES

A UTILIZAÇÃO DA TÉCNICA DE CHROMAKEY PARA COMPOSIÇÃO DE CENAS EM AMBIENTES DE REALIDADE MISTURADA

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação. (Área de Concentração: Realidade Virtual).

Orientador:

Prof. Dr. Antonio Carlos Sementille

Marília
2007

SILVIO RICARDO RODRIGUES SANCHES

A UTILIZAÇÃO DA TÉCNICA DE CHROMAKEY PARA COMPOSIÇÃO DE CENAS EM AMBIENTES DE REALIDADE MISTURADA

Banca examinadora da dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação. Área de Concentração: Realidade Virtual.

Resultado: APROVADO

Orientador: Prof. Dr. Antonio Carlos Sementille

1º EXAMINADOR: Prof. Dr. José Remo Ferreira Brega

2º EXAMINADOR: Prof. Dr. Romero Tori

Marília, 19 de outubro de 2007.

AGRADECIMENTOS

Primeiramente agradeço a Deus, por estar sempre presente.

Aos meus amigos, por me tolerarem.

À minha família, por me tolerar e me ajudar.

À minha namorada, por me tolerar, me ajudar e me compreender.

Ao meu orientador, por todos os ensinamentos.

À CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, pelo apoio financeiro.

Não sabendo que era impossível, ele foi lá e fez.

Jean Cocteau

SANCHES, Silvio Ricardo Rodrigues. **A utilização da técnica de chroma-key para composição de cenas em ambientes de realidade misturada**. 2007. 150 f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

RESUMO

O alinhamento tridimensional de objetos virtuais com o ambiente real, denominado "problema do registro", pode ser considerado um dos maiores desafios no desenvolvimento de sistemas de Realidade Misturada. A maioria das ferramentas de suporte e bibliotecas de *software* existentes, embora estimem satisfatoriamente a posição e a orientação em que os objetos virtuais devam ser gerados, normalmente não permitem que elementos reais sobreponham objetos virtuais. Isso provoca, em determinadas situações, uma incoerente relação de profundidade entre os elementos visualizados na cena, o que caracteriza um erro de registro. Objetiva-se o presente trabalho a organizar e exibir elementos reais e virtuais de forma coerente, em ambientes de Realidade Misturada, utilizando-se da técnica do *chromakey* e de funções para manipulação do *framebuffer* do OpenGL.

Palavras-chave: Realidade Misturada. Chromakey. ARToolkit. Oclusão Mútua.

SANCHES, Silvio Ricardo Rodrigues. **A utilização da técnica de chroma-key para composição de cenas em ambientes de realidade misturada**. 2007. 150 f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

ABSTRACT

The three-dimensional alignment between the virtual object and the real environment, also known as "registration problem", can be considered one of the great challenges in the development of Mixed Reality systems. Most available support tools and software libraries, in spite of satisfactorily estimating the position and orientation in which virtual objects are supposed to be generated, do not usually allow that real elements superimpose on virtual ones. In certain situations, the problem may cause an incoherent relation of depth between the virtual objects and the real elements of the scene — that is, it defines the so called registration error. The present work proposes both the organization and the exhibition in a coherent way of real ans virtual elements in Mixed Reality environments, by using the chromakey technique and the OpenGL framebuffer manipulation functions.

Keywords: Mixed Reality. Chromakey. ARToolkit. Mutual Occlusion.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

3D: Tridimensional

API: *Application Programming Interface*

APT: *Advanced Packaging Tool*

AVI: *Audio Video Interleave*

BBC: *British Broadcasting Corporation*

BMP: *Bit Mapped Picture*

CMYK: *Cyan*, *Magenta*, *Yellow* e *BlacK*

DART: *Designer's Augmented Reality Toolkit*

GNU: *General Public License*

HDTV: *Higth-Defnition Television*

HITLab: *Human Interface Technology Laboratory*

HITLabNZ: *Human Interface Technology New Zealand*

HMD: *Head Mounted Display*

HLS: *Hue*, *Lightness* e *Saturation*

HSR: *Hidden Surface Removal*

HSV: *Hue*, *Saturation* e *Value*

IDE: *Hue*, *Integrated Development Environment*

JNI: *Java Native Interface*

JPEG: *Joint Photographic Experts Group*

LCD: *Liquid Crystal Display*

LED: *Light Emitting Diode*

MPEG: *Moving Picture Experts Group*

NHK: *Nippon Hoso Kyokai*

OpenCV: *Open Computer Vision Library*

OpenGL: *Open Graphics Library*

OpenVRML: *Open Virtual Reality Modeling*

OSGART: *ARToolkit for OpenSceneGraph*

PDA: *Personal Digital Assistant*

RA: Realidade Aumentada

RAM: *Random Access Memory*

RGB: *Red*, *Green* e *Blue*

RGBA: *Red*, *Green*, *Blue* e *Alpha*

RGBD: *Red*, *Green*, *Blue* e *Depth*

RGBα: *Red*, *Green*, *Blue* e *Alpha*

RM: Realidade Misturada

RV: Realidade Virtual

SAR: *Spatial Augmented Reality*

VA: Virtualidade Aumentada

XML: *eXtensible Markup Language*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Em se tratando do desenvolvimento de sistemas de Realidade Misturada, as maiores dificuldades encontram-se freqüentemente associadas a problemas de registro, uma vez que se faz imprescindível um alinhamento correto dos objetos virtuais em relação ao ambiente real, para que se obtenha a ilusão de coexistência entre eles.

Os níveis de exigência para precisão no registro variam de acordo com a natureza da aplicação a ser desenvolvida. Sistemas médicos, por exemplo, mesmo que utilizados para treinamento, obviamente devem ser mais precisos que os voltados ao ensino ou ao entretenimento.

Grande parte das ferramentas para desenvolvimento de sistemas de Realidade Misturada oferece funções para estimativas de posicionamento e orientação para geração de objetos virtuais no ambiente misturado; no entanto, tais ferramentas não estimam localizações de objetos reais.

Esta limitação faz com que a cena real seja exibida como plano de fundo durante todo o tempo de execução da aplicação, e que, mesmo posicionado mais ao fundo, o objeto virtual em nenhum momento seja obstruído

por algo real no processo de visualização.

Segundo Kiyokawa *et al.* (2000), é desejável que, em ambientes misturados, objetos reais e virtuais mutuamente se ocultem (oclusão mútua), reproduzindo noções de profundidade essenciais para obtenção de maior realismo da cena e evitando problemas relacionados ao registro.

Considerando o contexto exposto, apresenta-se, neste trabalho, um método para composição de cenas tridimensionais em sistemas de Realidade Misturada, baseados na biblioteca ARToolkit (ARToolkit, 2006), que utiliza a técnica do *chromakey* para extração dos elementos reais.

O método, implementado como um componente para o ARToolkit, utiliza uma "máscara" da camada de imagem, que deve ser reproduzida em primeiro plano e realiza operações na imagem no *framebuffer* OpenGL (Segal e Akeley, 2007), anteriormente à exibição na tela.

A utilização das funções desenvolvidas permite a composição de cenas misturadas com consideração de profundidade, possibilitando que elementos reais possam ser exibidos mais próximos do observador e, desse modo, obstruam os objetos virtuais dispostos na cena, se necessário.

Para que favoreça seu pleno entendimento, este trabalho está organizado da forma que segue:

O Capítulo 2 mostra os conceitos e definições de Realidade Misturada, as principais aplicações, as diferenças em relação à Realidade Virtual e suas principais características.

Apresentam-se, inclusive, os componentes essenciais de um sistema de Realidade Misturada, as principais tecnologias de visualização e algumas das mais conhecidas ferramentas, comerciais ou acadêmicas, de auxílio ao desenvolvimento.

No Capítulo 3 abordam-se as técnicas de extração de chave e composição de imagens mais comuns na indústria da televisão e do cinema. São

analisados seu processo evolutivo e suas abordagens mais utilizadas. Uma ênfase maior é dada à técnica do *chromakey*, apresentando-se suas características, conceitos, aperfeiçoamentos, limitações e *hardware* dedicado.

Em relação à algoritmos, apresenta-se uma abordagem da técnica por meio da análise de alguns sistemas de cores utilizados na implementação, além de métodos para isolamento de cores e representação de transparência de *pixels*.

O Capítulo 4 trata da importância da noção de profundidade na composição de cenas misturadas, a necessidade de precisão no cálculo do registro, e a importância da oclusão mútua para obtenção de maior naturalidade na cena. Também se apresentam algumas tecnologias deste tipo, baseadas em *hardware* e utilizadas principalmente na indústria televisiva.

Ademais, descrevem-se os principais algoritmos de remoção de partes não visíveis de objetos virtuais. Tais algoritmos são responsáveis por cálculos de profundidade entre partes de um mesmo objeto ou entre vários objetos, o que permite tornar visíveis as superfícies mais próximas do observador.

Faz-se uma análise detalhada de cada *buffer* do conjunto de *buffers* (*framebuffer*) do OpenGL. Os testes e operações realizadas que afetam cada fragmento de *pixel* ou o *buffer* no todo são descritos no mesmo Capítulo.

No Capítulo 5, descreve-se a proposta para o desenvolvimento do projeto, os detalhes da implementação, as definições dos algoritmos e a utilização das funções para composição de cenas com consideração de profundidade. Exibem-se, inclusive, os testes e a análise de desempenho dos protótipos desenvolvidos, bem como detalhes do ambiente experimental utilizado.

Finalmente, reservou-se o Capítulo 6 para as conclusões do presente estudo, e para apresentar as perspectivas de trabalhos futuros.

Capítulo 2

REALIDADE MISTURADA

2.1 Conceitos básicos

O termo Realidade Misturada (RM) foi introduzido no mundo da computação por Milgram *et al.* (1994) representando uma variação Realidade Virtual (RV). Pesquisas nesta área têm como objetivo básico aumentar a percepção do usuário e a interação com o mundo real pelo suplemento da realidade com objetos virtuais tridimensionais que pareçam coexistir no mesmo espaço (Azuma, 2004). A utopia é a criação de um ambiente em que o usuário não consiga distingüir o mundo real do virtualmente aumentado.

Na proposta de Milgram *et al.* (1994), que se baseia em um "*Continuum* de Virtualidade", imagina-se um ambiente real, em que objetos virtuais são inseridos, para que se obtenha uma cena combinada, mas com predominância da realidade. Esse é o ponto em que se encontra a Realidade Aumentada (RA) (Figura 2.1).

Esta combinação também pode ser feita de maneira similar inserindo-

se objetos reais em ambientes sintéticos. Havendo predominância do virtual, caracteriza-se o que Milgram *et al.* (1994) definiram como Virtualidade Aumentada (VA). A RM representa todo espaço de transição entre as "realidades", feita de modo intuitivo.

Torna-se evidente a impossibilidade de identificação de limites rígidos entre a RA e a VA e, por esse motivo, as transições são demonstradas sugerindo a idéia de continuidade.

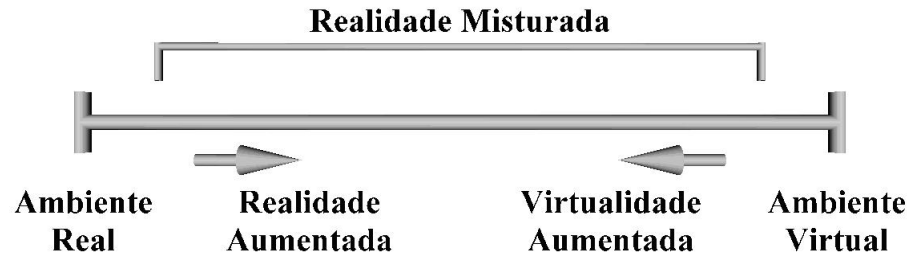


Figura 2.1: "*Continuum*" de Virtualidade de Milgram (Milgram *et al.*, 1994)

Na Figura 2.1, observa-se à esquerda um ambiente totalmente real que pode ser representado pelos Sistemas de Telepresença (Schloerb, 1995). O ambiente virtual está posicionado à direita na mesma Figura. Fazem parte desse outro grupo todos os sistemas que imergem o usuário em um ambiente totalmente sintético. Um exemplo clássico é o modelo de imersão das "Cavernas Digitais" (Buxton e Fitzmaurice, 1998). A RA e a VA ocupam toda parte central do "*Continuum* de Milgram".

Esta definição ressalta uma característica considerada fundamental na diferenciação entre sistemas de RM e sistemas de RV. No segundo, o usuário é imerso em um ambiente gerado por computador, o que o impede de enxergar o mundo real à sua volta, enquanto na RM permite-se ao usuário uma visão do mundo real e os objetos virtuais apenas o complementam (Simsarian e Akesson, 1997).

Realidade Melhorada é outra terminologia encontrada para sistemas com características semelhantes. O conceito fundamental para esse termo é a idéia de "anotação visual" (Bowskill e Downie, 1995).

Informações dos objetos em cena, tais como tamanho, distância, entre outras características, podem ser descritas na tela como forma de melhoria de seu entendimento. Além de melhoria de conteúdo, o uso do processamento da imagem para gerar um aumento na sua qualidade pode ser considerado Realidade Melhorada.

Definições mais antigas vinculam sistemas de RA, e conseqüentemente de RM, à utilização de *Head Mounted Displays* (HMDs); no entanto, esta é uma definição bastante restritiva, e boa parte dos sistemas existentes não estariam aí enquadrados (Azuma, 1997).

Azuma (1997) apresentou uma idéia bem mais abrangente para definilos. A proposta baseia-se na coexistência de três características essenciais para que um sistema seja considerado um sistema de RM (Azuma, 1997). São elas:

- combinação de algo real e virtual;
- interação em tempo real; e
- alinhamento e sincronização precisos dos objetos virtuais tridimensionais com o ambiente real (registro).

Desse modo, a caracterização dos sistemas de RM não os vincula a nenhum tipo de dispositivo específico.

As áreas de aplicação da RM estão em constante expansão. Podem ser destacados projetos na medicina, na manutenção de equipamentos, na engenharia biomédica e no treinamento militar, além de aplicações colaborativas de vídeo-conferência, ou voltadas ao entretenimento (Azuma, 2004).

2.2 Componentes de um sistema de RM

Obter-se um "modelo" de sistema para RM torna-se tarefa bem complicada, em decorrência da infinidade de dispositivos e aplicações. Não obstante, uma análise acurada de suas características possibilita a identificação de tarefas essenciais que podem ser organizadas e classificadas como sistemas menores (subsistemas).

A proposta apresentada por Vallino (2005) e complementada por Braz e Pereira (2005) sugere pelo menos cinco subsistemas considerados indispensáveis a qualquer sistema de RM.

Nesses subsistemas, são divididas as tarefas de captação da imagem real, a obtenção da posição e orientação do subsistema de captura da imagem real, a geração de objetos virtuais, a mistura da imagem e a exibição da imagem de saída.

Estendendo um pouco mais esse conceito para considerar a existência de objetos reais em movimento na cena, fato comum em muitas aplicações de RM, Braz e Pereira (2005) propuseram o acréscimo de três outros subsistemas:

- um subsistema de manipulação real de objetos capacitado para pode lidar tanto com os objetos reais quanto com os virtuais;
- um subsistema para o acesso à localização do subsistema anterior; e
- um subsistema para o acesso à localização dos objetos reais que se movimentam no ambiente aumentado.

O primeiro subsistema adicional, segundo Braz e Pereira (2005), deve ser incorporado aos apresentados por Vallino (2005), e os dois últimos, mesclados e transformados em um único, para localização global. Na Figura 2.2

exibe-se a arquitetura de um sistema de RM típico, de acordo com o modelo proposto por Braz e Pereira (2005).

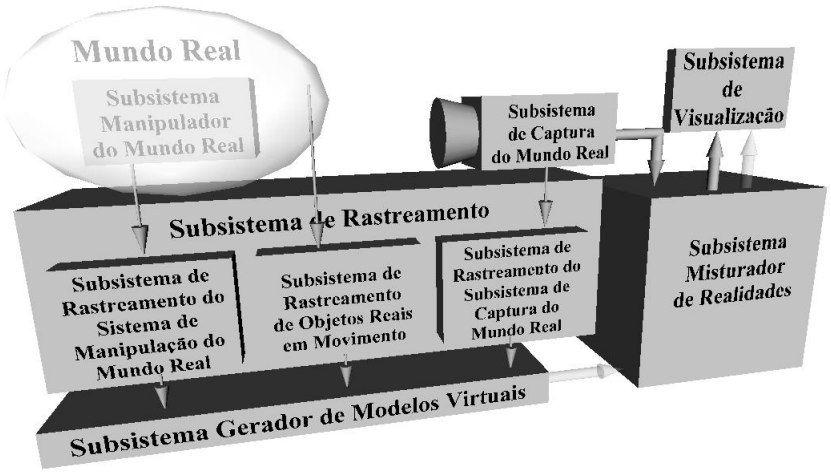


Figura 2.2: Componentes de um Sistema de RM (Braz e Pereira, 2005)

Um sistema de RM pode, desse modo, ser caracterizado pelos seis subsistemas mostrados na Figura 2.2. De maneira simplificada, pode-se definir o Subsistema de Captura do Mundo Real como o sistema que estimula os respectivos sentidos do ser humano, com sinais originados do mundo real.

O Subsistema Gerador de Modelos Virtuais é responsável pela construção da parte gráfica. Por meio desse subsistema, pode-se classificar os modelos gráficos gerados, analisando-se o grau de realismo, o modo como são construídos, os níveis de interação com o usuário e o tipo de tecnologia utilizada.

O Subsistema Misturador de Realidades realiza a combinação do real e do virtual e permite a identificação das tecnologias de mistura, que podem ser ópticas ou eletrônicas.

O Subsistema Manipulador do Mundo Real possui características completamente diferentes para cada tipo de tecnologia utilizada com essa finalidade. É responsável pela interação háptica com o mundo real ou virtual.

O Subsistema de Rastreamento é o responsável pelo acesso à localiza-

ção do observador e dos objetos reais. Esse subsistema pode ser caracterizado pelo grau de complexidade, tipo de rastreamento (passivo e baseado em marcadores ou ativo) e por algumas outras características técnicas. Sua implementação é, sem dúvida, o maior obstáculo no desenvolvimento de sistemas de RM.

O Subsistema de Visualização relaciona-se com a tecnologia dos dispositivos utilizados para essa finalidade. É responsável pela identificação do ponto de vista do usuário, que pode estar imerso, pelo uso do HMD, ou enxergando o mundo por meio de uma janela de monitor de vídeo convencional. A quantidade de cores suportadas pelos dispositivos e as taxas de quadros exibidas podem ser obtidas por meio desse subsistema.

Os dispositivos de visualização, que representam esse subsistema, são discutidos detalhadamente na Seção 2.3. Esses dispositivos, baseados em tecnologias bastante diferenciadas, são normalmente utilizados como critério para classificação de sistemas de RM.

2.3 Tecnologias de visualização

Um modo muito comum de se classificar sistemas de RM é diferenciá-los considerando-se o dispositivo de visualização utilizado. Não é difícil imaginar que a escolha desse critério deve-se ao fato de esta tecnologia ter influência direta na qualidade da imersão proporcionada ao usuário (Braz e Pereira, 2005).

Um dispositivo de exibição (*Display*) pode ser considerado um sistema de formação de imagens, que utiliza um conjunto de componentes ópticos, eletrônicos e mecânicos para gerar imagens em algum lugar, entre os olhos do observador e o objeto físico a ser sobreposto (Bimber e Raskar, 2005).

Ao utilizar-se do termo "formação de imagens", considera-se que alguns dispositivos projetem essa imagem em algum tipo de superfície, e não as exibam em telas próprias.

Os mais comuns dispositivos de exibição dos sistemas de RM são os HMDs com visualização por meio de vídeo (*video see-through*), ou com visualização direta do mundo real (*optical see-through*).

Esses dispositivos, que são discutidos de maneira mais detalhada na Subseção 2.3.1, ainda possuem algumas limitações, inclusive ergonômicas, o que impede sua utilização em um maior número de aplicações.

Novos conceitos de dispositivos de visualização, considerados uma variação da tecnologia tradicional, apresentados por Bimber e Raskar (2005), foram denominados "*Spatial Augmented Reality*" (SAR). Esses dispositivos são caracterizados, sobretudo, pelo fato de estarem separados do corpo do usuário e não acoplados à cabeça, como os tradicionais.

Na Figura 2.3 são mostrados três grupos de *Displays* de RM, e identificadas as diferentes possibilidades de formação de imagens, o posicionamento dos dispositivos e o tipo de imagem produzida. A identificação do tipo de imagem produzida refere-se ao fato de alguns métodos, baseados em projetores, produzirem imagens em superfícies não planas, diferentemente da maioria das abordagens que formam imagens planas.

Segundo Bimber e Raskar (2005), os diferentes tipos de dispositivos podem ser organizados em três categorias, descritas nas próximas Subseções.

2.3.1 Dispositivos acoplados à cabeça

Estes tipos de dispositivos fazem parte de sistemas de exibição que são presos à cabeça do usuário. Classificados de acordo com a tecnologia de ge-

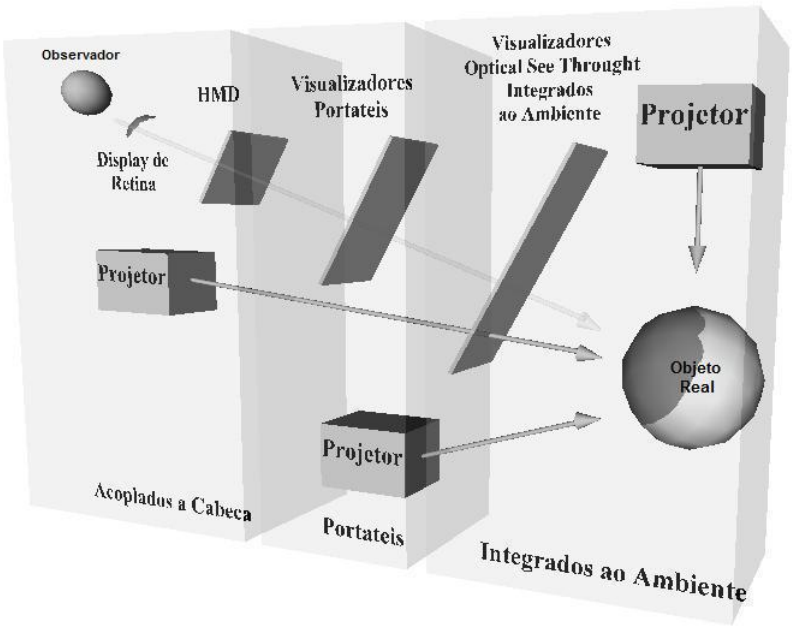


Figura 2.3: Tipos de *displays* de RM(Bimber e Raskar, 2005)

ração de imagem, três destes dispositivos podem ser considerados principais: os "*Displays* de retina",os HMDs e os capacetes projetores.

Os "*Displays* de retina" aplicam laser de baixa freqüência diretamente sobre a retina do usuário, em vez de utilizar telas posicionadas à frente de seus olhos. Isso produz imagens mais claras e de alta resolução, além de atingir um campo de visão maior que os dispositivos baseados em visualização por meio de telas. No entanto, podem causar incômodo e desconfiança em relação a problemas que esse laser possa causar à visão.

Os HMDs são dispositivos muito utilizados em aplicações de RM e baseiam-se em dois tipos de tecnologias. Nos HMDs do tipo *video see-through*, as imagens são capturadas por uma câmera de vídeo, misturadas e exibidas ao observador. Nesse caso, o mundo real é visto apenas pela tela posicionada à frente de seus olhos. Nos HMDs do tipo *optical see-through*, o mundo real é visto através de *displays* transparentes, onde são projetados os objetos virtuais.

Esse tipo de tecnologia de exibição, muitas vezes, é fator limitante no desenvolvimento de sistemas de RM, pelo fato de esses dispositivos não possuírem brilho, resolução, campo de visão e contraste suficientes para uma combinação "sem costuras" nos limites entre o objeto virtual e o mundo real (Azuma *et al.*, 2001).

Outro tipo de *display* denominado "capacetes projetores" pode ser dividido em dois tipos. Os *head-mounted projective displays* utilizam um sistema que redireciona a projeção, assim que as imagens são emitidas sobre um plano retro-reflexivo, localizado à frente dos olhos do observador.

Essa superfície retro-reflexiva possui centenas de micro-cubos com propriedades ópticas diferentes, que emitem luz na direção contrária à direção incidente. As imagens produzidas nesse tipo de superfície possuem mais brilho que as imagens refletidas em superfícies normais, que difundem a luz. Na Figura 2.4, apresenta-se a diferença entre superfícies reflexiva, difusa e a retro-reflexiva utilizada.

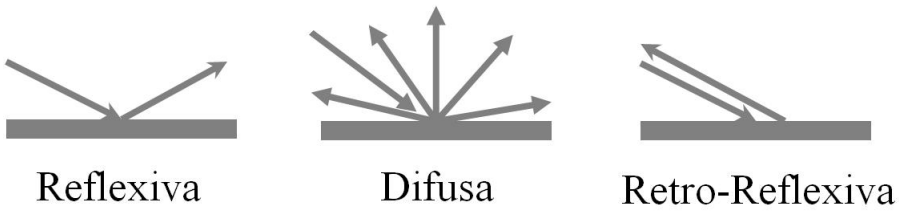


Figura 2.4: Tipos de superfícies de reflexão (Bimber e Raskar, 2005)

Os *projective head-mounted displays* emitem raios de luz sobre imagens criadas em superfícies regulares, colocadas à frente do observador. Utilizam-se dois espelhos especiais para integrar a imagem, projetada no campo de visão do observador, assim que os parâmetros dos projetores são ajustados.

2.3.2 Dispositivos de exibição portáteis

Encaixam-se nesta categoria de dispositivos, os *Tablets*, assistentes digitais pessoais (PDAs) e aparelhos celulares. Esses aparelhos possuem processador, memória, visualizadores e uma série de outros recursos que permitem sua perfeita utilização em sistemas de RM. A tecnologia de *video see-through* é normalmente utilizada nesses dispositivos, embora exista *optical see-through* aplicada a aparelhos portáteis.

Uma das vantagens da utilização desses dispositivos é que *software* e *hardware* muito similares aos de sistemas tradicionais podem ser utilizados. Computadores pessoais e HMDs são dispositivos muitas vezes inadequados em determinadas situações, principalmente devido ao peso e ao espaço físico que ocupam (Wagner e Schmalstieg, 2003).

O crescente interesse em *displays* menores vem acelerando o desenvolvimento de sistemas RM voltados a esses aparelhos, cuja mobilidade é o principal fator a se considerar.

2.3.3 Dispositivos separados do usuário

Este tipo de tecnologia, ao contrário das mostradas anteriormente, utiliza dispositivos separados do corpo do observador, e posicionados em algum lugar no ambiente delimitado pelo sistema. Conhecidos como (*Spatial Displays*), são baseados em três diferentes abordagens, diferenciadas pelo modo de inserção dos objetos virtuais no ambiente (Bimber e Raskar, 2005).

Os *Screen-Based Video See-Through Displays* são, muitas vezes, con-

siderados uma janela do mundo virtual. Nesta categoria, encontram-se os monitores convencionais onde são exibidas as imagens misturadas. Embora forneçam um baixo grau de imersão, devido ao campo de visão estar limitado ao tamanho do monitor, aplicações que utilizam esse tipo de dispositivo são bastante comuns e estão dentro do contexto de RM.

Os *Spatial Optical See-Through Displays*, contrariamente aos dispositivos presos à cabeça ou portáteis, geram imagens que são ajustadas dentro do ambiente real. Esse tipo de combinação óptica é representado pelas telas transparentes, hologramas ópticos, entre outras tecnologias de exibição de imagens similares, que possuem recursos de visualização não acoplados ao observador.

Nos *Projector-Based Spatial Displays*, as imagens não se formam em seus próprios *displays*, mas por meio de uma projeção direcionada para construí-las diretamente sobre a superfície de objetos físicos no campo de visão do observador. Existem aplicações que suportam projetores estáticos ou dirigidos e, em alguns casos, múltiplos projetores são utilizados para aumentar a área passível de projeção.

2.4 Principais dificuldades no desenvolvimento de sistemas de RM

Muitas dificuldades encontradas no desenvolvimento de sistemas de RM são familiares a grande parte dos pesquisadores da área (MacIntyre *et al.*, 2004). Entre esses problemas, podem ser destacados: o registro, a portabilidade, o cálculo do campo de visão, a interação com o usuário, as limitações e a complexidade das ferramentas de apoio existentes.

O registro pode ser considerado o maior desafio entre as dificuldades

citadas. Problemas de registro podem estar relacionados a distorções ópticas, erros no sistema de rastreamento, desalinhamentos mecânicos, parâmetros incorretos ou erros resultantes de atrasos no sistema (Romão, 2007). Os objetos reais e virtuais devem se alinhar de maneira precisa, para que a ilusão de coexistência não seja comprometida (Azuma, 1997).

Torna-se evidente que determinadas aplicações exigem mais precisão no registro do que outras. Uma aplicação médica obviamente deve ser mais precisa que aplicações voltadas ao entretenimento.

Problemas de registro também ocorrem em ambientes de RV; contudo são mais difíceis de serem detectados do que em ambientes de RM pelo fato de os usuários apenas visualizarem objetos virtuais (Azuma, 1997).

Erros de registro são de difícil controle devido, principalmente, às inúmeras origens às quais podem estar relacionados. Pode-se dividi-los em dois tipos. O primeiro relaciona-se a erros estáticos que ocorrem mesmo quando o ponto de visão do usuário e os objetos do ambiente permanecem imóveis.

O segundo tipo de erro são os dinâmicos, que só podem ser detectados quando há uma mudança no ponto de visão, ou quando se inicia algum movimento de objetos virtuais da cena. Erros dinâmicos ocorrem com maior freqüência, mas erros estáticos não podem ser desconsiderados (Azuma, 1997).

Problemas de alinhamento tridimensional e sincronização não estão restritos a RM ou RV. Esses problemas também devem ser contornados na criação dos efeitos especiais que misturam objetos virtuais e atores em composições de cinema e televisão.

No entanto, nessas circunstâncias podem se fazer ajustes a cada quadro de vídeo, no caso dos filmes, e atores podem ser dirigidos cuidadosamente durante o processo de gravação, mesmo em transmissões de televisão ao vivo. Em sistemas de RM, o sistema não pode controlar os movimentos do usuário,

mas deve responder rapidamente às suas ações (AZUMA, 1997).

Sistemas RM também devem possibilitar que elementos reais sejam exibidos, quando necessário, à frente de elementos virtuais e vice-versa. Em grande parte das aplicações, o elemento virtual é exibido sempre em primeiro plano (KIYOKAWA *et al.*, 2000). Essa limitação pode ocasionar problemas de registro em determinadas situações.

A portabilidade é uma característica muitas vezes desejável em sistemas de RM. Algumas aplicações exigem que o usuário se movimente livremente em um determinado espaço físico (ROMÃO, 2007). A maioria dos dispositivos de RM existentes dificultam a implementação de sistemas com essa característica, devido a questões relacionadas a tamanho e peso dos equipamentos.

Encontrar o campo de visão do usuário também é uma tarefa essencial em sistema de RM. Para tanto, faz-se necessário um método de difícil implementação, para obter sua posição e orientação a cada *frame* (ROMÃO, 2007).

Outro problema a ser contornado por desenvolvedores de sistemas de RM é a interação do usuário. O modo de interação, normalmente, está diretamente relacionado com o dispositivo utilizado. Dessa forma, grande parte das rotinas se torna específica da aplicação, ou necessita de muitas adaptações (MACINTYRE *et al.*, 2004).

As limitações e a complexidade das ferramentas de apoio também pode ser considerado um problema no desenvolvimento de sistemas RM. Muitas dessas ferramentas obrigam o programador a trabalhar em baixo nível de programação, ou não separam o alto do baixo nível, tornando a aplicação dependente da tecnologia utilizada. MacIntyre *et al.* (2004) definem tais suportes e bibliotecas como "desenvolvidas por cientistas da computação para cientistas da computação" devido à inexistência de *toolkits* que ofereçam in-

terfaces mais amigáveis.

2.5 Ferramentas para desenvolvimento de aplicações de RM

Nesta Seção, apresentam-se algumas das principais ferramentas para desenvolvimento de aplicações RM, acompanhadas de suas respectivas funcionalidades, que procuram minimizar algumas das tradicionais dificuldades impostas pela programação e criação de ambientes misturados, citados na Seção 2.4. Entre essas ferramentas, são destacados os projetos do Studierstube, DART, Tinmith, OpenCV e ARToolkit.

2.5.1 Studierstube

O Studierstube é uma ferramenta de apoio ao desenvolvimento de aplicações de RMs móveis, colaborativas e ubíquas. Construído para a ferramenta *Open Inventor* (SGI, 2007), como um conjunto de classes C++, consiste em um sistema de gerenciamento de interfaces de usuário para aplicações de RM complexas.

Seu desenvolvimento começou na Universidade de Tecnologia de Viena, em 1996 e, a partir de 2004, as principais pesquisas e atividades de desenvolvimentos são realizadas na Universidade de Tecnologia Graz (STUDIERSTUBE, 2007).

O objetivo principal é proporcionar uma interface de usuário utilizável para aplicações que requeiram manipulação de informações tridimensionais

complexas a todo o momento. A essência do projeto procura encontrar uma metáfora para interfaces tridimensionais de usuário equivalente ao que as estações de trabalho (*desktop*) representam para as interfaces bidimensionais.

O ambiente de trabalho do Studierstube é controlado por um painel de interações, por dispositivos manuais e interfaces do tipo *pen-and-pad*, que proporcionam versatilidade na interação com o ambiente virtual, além de utilizar recursos convencionais como multitarefa e utilização simultânea de várias janelas (SCHMALSTIEG *et al.*, 2002).

O projeto Studierstube possui uma excelente documentação e está na base de muitos sistemas de RM, principalmente móveis e colaborativos. Na Figura 2.5, exemplifica-se a aplicação desenvolvida utilizando o Studierstube.

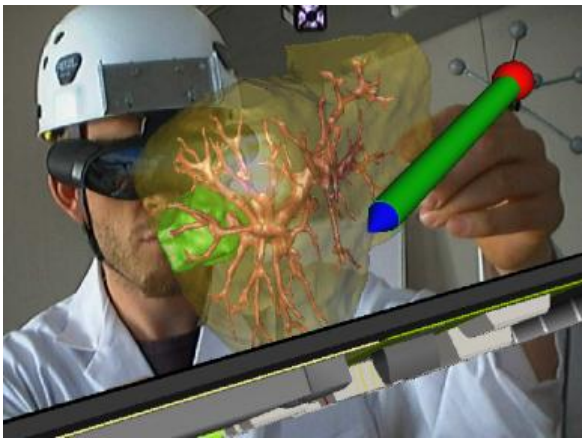


Figura 2.5: Aplicação baseada no Studierstube (STUDIERSTUBE, 2007)

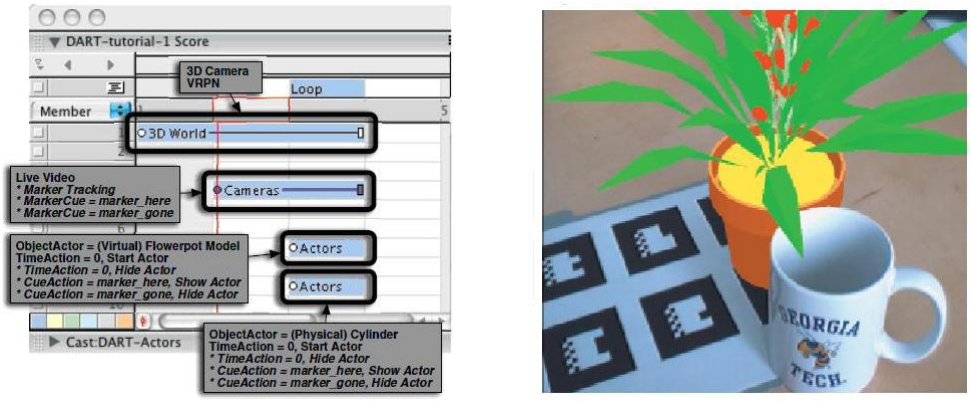
2.5.2 DART (*Designer's Augmented Reality Toolkit*)

O DART é um conjunto de ferramentas de *software* desenvolvido para ser utilizado de forma integrada à ferramenta multimídia profissional *Macromedia Director* (ADOBE SYSTEMS, 2007). Suas funcionalidades auxiliam

o desenvolvimento de aplicações e outras experiências de RM (Figura 2.6). O objetivo é facilitar todo processo de concepção e implementação, além de oferecer ferramentas de testes para o produto final (Geogia Tech, 2006).

A maior parte do DART é implementada em forma de *scripts* e pode ser modificada pelos desenvolvedores conforme suas necessidades. Esse conjunto de ferramentas permite aos desenvolvedores especificar relacionamentos complexos entre o mundo físico e o virtual, e possibilita que etapas do projeto sejam freqüentemente testadas.

Diferentemente de outras ferramentas de RM, que requerem a utilização de Linguagens de Programação como C ou C++ (as quais exigem um bom conhecimento de códigos fonte), o DART é mais voltado para desenvolvedores familiarizados com ferramentas visuais de desenvolvimento como o Photoshop, Flash e Maya. A linguagem de programação embutida no *Macromedia Director* é uma linguagem orientada a objetos chamada Lingo.



(a) Interface do Dart **(b)** Aplicação em Execução

Figura 2.6: Sistema DART (Geogia Tech, 2006)

2.5.3 Sistema Tinmith

O sistema Tinmith, desenvolvido na Universidade do Sul da Austrália, serve como base para aplicações de RM, que permitem ao usuário construir modelos simples de estruturas externas. A tecnologia de interface utiliza uma luva de dados baseada em um sistema de menus e técnicas de interação tridimensional (Piekarski e Thomas, 2002).

A ferramenta permite a construção eficiente de formas de geometrias sólidas do mundo em tempo real, sem um conhecimento prévio do ambiente e, diferentemente da maioria dos trabalhos que focalizam a seleção e manipulação de objetos virtuais sobre o mundo, o Tinmith permite a criação dos objetos partindo de um mundo vazio (Figura 2.7).

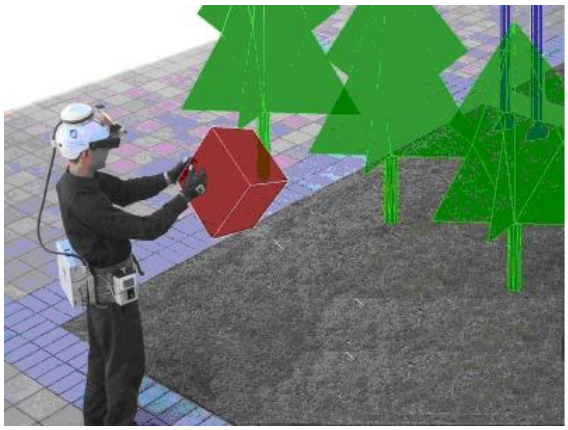


Figura 2.7: Contrução de modelos virtuais utilizando Tinmith (Piekarski e Thomas, 2002)

2.5.4 OpenCV (*Open Computer Vision Library*)

A biblioteca OpenCV (OpenCV, 2007) consiste de um conjunto de

funções com implementações de algoritmos de processamento de imagens e visão computacional, que podem ser utilizados para o desenvolvimento de sistemas de RM. Suas funcionalidades são consideradas uma alternativa interessante, principalmente em relação a interface (Bernardes *et al.*, 2007).

Técnicas de visão computacional podem ser utilizadas para captar entradas de usuários com um custo menor, em relação à utilização de sensores (por exemplo, magnéticos). É possível, por meio da OpenCV, separar partes de objetos (elemento de interesse) inseridos no ambiente misturado, além de extrair outras características da imagem.

A OpenCV tem suporte oficial para as plataformas Windows, Linux e MacOs X, embora seja possível uma adaptação a novos sistemas operacionais ou novas arquiteturas com relativa simplicidade (Bernardes *et al.*, 2007). Na Figura 2.8, exibe-se uma aplicação desenvolvida utilizando a OpenCV.

Figura 2.8: Aplicação desenvolvida utilizando a OpenCV (Paula *et al.*, 2007)

2.5.5 ARToolkit

A biblioteca de software ARToolkit (ARToolkit, 2006), que utiliza

a linguagem C e C++, oferece um conjunto de funções que realizam algumas das tarefas mais complexas das aplicações de RM. Sua implementação é baseada em algoritmos de visão computacional para resolver o problema da estimativa do campo de visão do usuário (KATO *et al.*, 1999).

Esse cálculo, como já foi descrito na Seção 2.4, é uma das dificuldades clássicas no desenvolvimento de aplicações desse tipo, e uma solução precisa é fundamental para que se possa construir o objeto virtual na posição e na orientação correta.

O projeto da biblioteca está sob responsabilidade da Universidade de Osaka, no Japão, apoiada pelo *Human Interface Technology Laboratory* (HITLab) da Universidade de Washington, nos Estados Unidos, e pelo HITLab NZ na Universidade de Canterbury, na Nova Zelândia (HIT LAB, 2006).

O ARToolkit é distribuído livremente para fins não comerciais sob a licença GNU, é suportado por várias plataformas, possui uma excelente documentação e é utilizado em conjunto com alguns *softwares* gráficos.

A renderização[1] normalmente é realizada pela biblioteca OpenGL (*Open Graphics Library*) ou OpenVRML (*Open Virtual Reality Modeling Language*), e a visualização pode ser feita por meio de dispositivos do tipo *Optical* ou *Video See-Through* (KATO *et al.*, 1999).

Bibliotecas de alto nível de acesso (HALLER *et al.*, 2002) também podem ser utilizadas para geração de objetos virtuais, como mostrado em Loosea *et al.* (2007) e em Sementille *et al.* (2004).

Entre as principais funcionalidades do ARToolkit, destacam-se o rastreamento da posição e da orientação da câmera, a utilização de simples quadrados marcadores para localização dessas coordenadas, a facilidade de configuração da câmera e uma execução rápida o suficiente para aplicações de tempo real. Na Figura 2.9, exibe-se uma aplicação do ARToolkit em

[1]Processo de geração dos objetos virtuais da imagem, feito por um computador

execução.

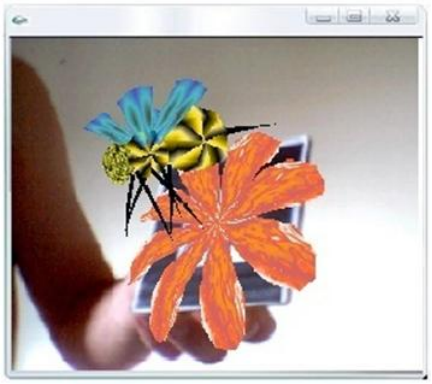


Figura 2.9: Aplicação do ARToolkit em execução

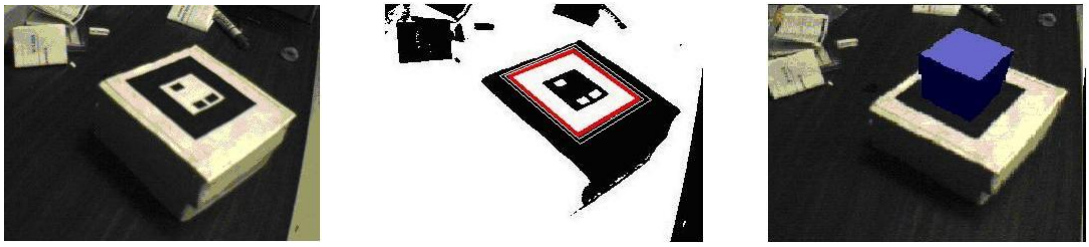
Uma alternativa para acessar as funções do ARToolkit, utilizando a linguagem Java, Lingo ou qualquer linguagem de script, é a JARToolkit (Geiger *et al.*, 2002). Baseada na JNI (*Java Native Interface*), possibilita, inclusive, trabalhar-se com bibliotecas de alto nível de acesso como JAVA 3D, ou de baixo nível de acesso como GL4JAVA, para renderização de objetos virtuais.

Outro projeto baseado na biblioteca ARToolkit é o OSGART (Loosea *et al.*, 2007). A biblioteca OSGART combina as funções de detecção e rastreamento de marcadores do ARTookit com as funções para construção de modelos virtuais da biblioteca OpenSceneGraph (Universidad Politecnica de Valencia, 2007).

2.5.5.1 Aspectos gerais do funcionamento

As técnicas de visão computacional utilizadas pelo ARToolkit permitem o cálculo do ponto de visão da câmera de vídeo em relação a um marcador real colocado na cena. Esse processo inicia-se com a captura da

imagem (Figura 2.10a), que é convertida em imagem binária obedecendo a um valor limite (*thresholding*) estabelecido, para transformar as cores da imagem em preta ou branca (Figura 2.10b) (KATO *et al.*, 1999).



(a) Marcador real

(b) Região identificada

(c) Renderização do modelo virtual

Figura 2.10: Processo de reconhecimento (KATO *et al.*, 1999)

Esse valor limite é fixado, mas pode ser alterado manualmente em tempo de execução provocando uma interrupção no programa. A variação desse valor muitas vezes é necessária principalmente em ambientes em que a iluminação não é constante. Um *thresholding* adaptativo para a biblioteca ARToolkit foi desenvolvido por Pintaric (2003), na tentativa de automatizar esse ajuste.

Na imagem binária são identificadas regiões quadradas que passam por um processo de comparação com modelos de marcadores previamente inseridos no sistema. Havendo sucesso na comparação, identifica-se um marcador.

O fato de as dimensões do marcador estarem armazenadas no sistema, assim como a orientação correta do identificador do usuário em seu interior, permite ao ARToolkit calcular a posição da câmera de vídeo em relação a esse marcador físico.

Esses valores são utilizados para encontrar as coordenadas para projeção dos objetos virtuais na tela, que deve ser feita precisamente sobre o marcador real (Figura 2.10c) por meio de algum *software* gráfico (KATO *et*

al., 1999).

Os princípios matemáticos envolvidos nas estimativas de posicionamento e orientação dos marcadores baseiam-se em cálculos que geram valores para matrizes de transformação. As coordenadas dos marcadores são convertidas entre os vários quadros de coordenadas do ARToolkit (Figura 2.11) para que possam servir de parâmetros aos objetos virtuais renderizados sobre esses marcadores na tela.

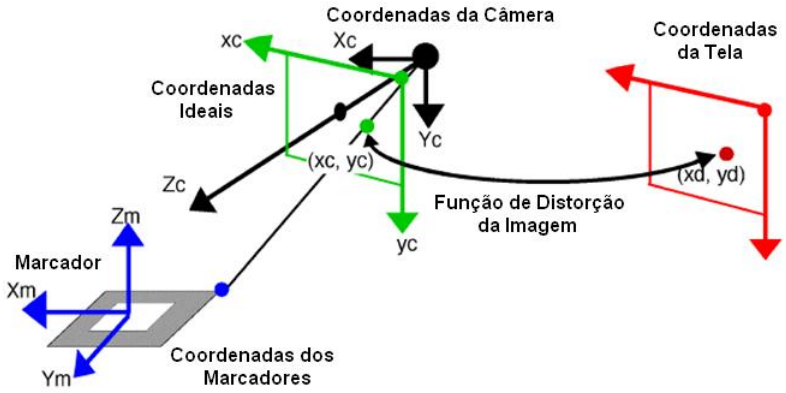


Figura 2.11: Quadro de coordenadas do ARToolkit (Kato e Billinghurst, 1999)

2.5.5.2 Precisão do processo de detecção

As funções do ARToolkit, que realizam os cálculos para definir as coordenadas onde o objeto virtual deve ser gerado, fornecem resultados satisfatórios, mantendo os objetos virtuais sempre alinhados ao marcador real.

Para analisar, de modo mais preciso, as estimativas de posição e orientação dos marcadores, Piekarski *et al.* (2004) definiram uma função que calcula as áreas em que os erros são menores. Na experiência, a câmera foi

fixada de modo que a altura da lente e o centro do marcador ficassem no mesmo nível. O ângulo em torno dos eixos X e Z permaneceram fixos em 0º e variações foram medidas em relação ao eixo Y.

A função de precisão ilustrada na Figura 2.12 mostra os quatro intervalos citados e identifica as áreas em que a estimativa é mais precisa. As áreas em tons de cinza mais escuros representam os locais onde o ARToolkit é mais preciso, e a área preta são os locais onde os marcadores não foram reconhecidos.

Os resultados da experiência mostram que os erros do ARToolkit, em relação à distância do marcador na faixa entre 20 e 70 cm são baixos e o desvio padrão entre 20 e 50 cm é pequeno. Em relação aos ângulos de rotação, os erros são menores na faixa entre 30º e 40º, e o desvio padrão é pequeno na faixa entre 40º e 85º.

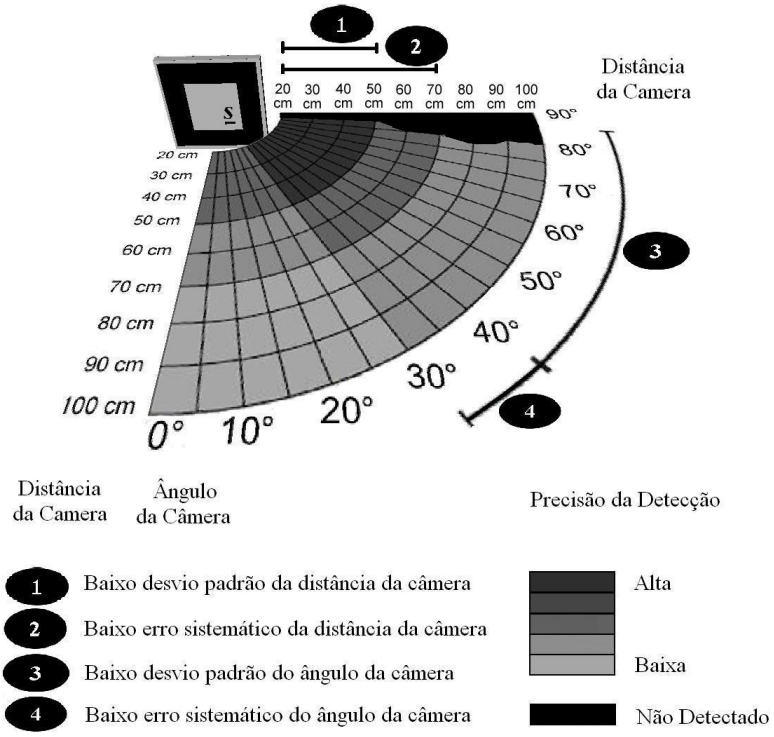


Figura 2.12: Análise da precisão do ARToolkit (Piekarski *et al.*, 2004)

2.5.5.3 Confiabilidade x desempenho

A resolução da imagem do marcador inserida no sistema, com informações de identificadores, tem influência direta no seu reconhecimento. Quanto maior a resolução, maior é número de *pixels* analisados e mais preciso é o processo de identificação (Figura 2.13).

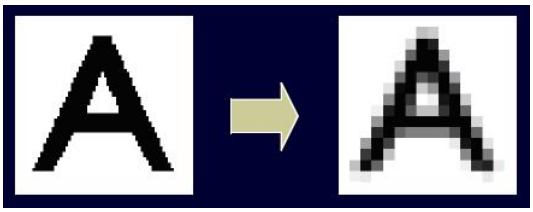


Figura 2.13: Exemplos de diferentes resoluções de imagens (KATO, 2007)

Obviamente, essa precisão provoca algum prejuízo ao desempenho do sistema. O problema fica mais evidente quando existe a necessidade de inserção de muitos marcadores na cena. Nesse caso, uma resolução muito alta certamente provocará atrasos significativos (KATO, 2007).

A principal função para detecção de marcadores do ARToolkit armazena valores em um histórico para maior confiabilidade em suas detecções. Existe uma função de detecção mais simples que ignora esses valores para diminuir o tempo de execução, o que torna o processo menos confiável (KATO, 2007).

2.5.5.4 **Desafios do ARToolkit**

Muitas das exigências dos sistemas mais complexos de RM ainda não são exploradas pelo ARToolkit. Efeitos como sombras de objetos virtuais e possibilidade de controle da profundidade de toda a cena são características importantes ainda não implementadas na biblioteca.

Várias propostas têm sido apresentadas em relação a algumas dessas limitações. No entanto, nenhuma delas foi incorporada à biblioteca. Kont-tinen *et al.* (2007) utilizam o ARToolkit para calcular o ponto de visão da câmera e renderizar objetos virtuais em um ambiente que produz sombras e luzes sobre objetos reais.

Loosea *et al.* (2007) ressaltaram a importância da geração de sombras em sistemas de RA para uma cena coesa e de como seus efeitos melhoram a percepção de profundidade do usuário em relação aos objetos da cena. O sistema proposto utiliza o ARtoolkit como base, e o método apresentado para geração de sombras também necessita de inserção manual das medidas de profundidade dos objetos reais do ambiente.

Uma limitação implícita da biblioteca ARToolkit, bastante prejudicial à naturalidade dos ambientes produzidos, é o fato de o mundo real estar sempre no contexto do plano de fundo. Os objetos virtuais renderizados em nenhum momento são combinados de modo que sejam sobrepostos por atores ou outros objetos reais, ocasionando problemas de registro em algumas situações.

Para que se produza esse tipo de efeito, alguns problemas devem ser contornados. Um desses problemas é o fato de a oclusão do marcador, por algo real, provocar a perda de seu reconhecimento, o que o impede de ser visualizado.

No momento da composição da imagem, é necessário o conhecimento

das medidas de profundidades de todos os objetos reais e virtuais que fazem parte da cena, para que possam ser corretamente combinados. Um outro problema, associado à implementação desse tipo de efeito, é a necessidade de um método de recorte da parte de interesse do ator, ou do objeto real, que deve aparecer à frente de objetos virtuais.

2.6 Considerações Finais

Neste Capítulo, apresentou-se uma visão geral a respeito de sistemas de RM, detalhando seus principais componentes, as tecnologias de visualização mais utilizadas e como esses dispositivos podem ser classificados.

Foram apresentados também algumas das ferramentas de auxílio ao desenvolvimento de sistemas de RM produzidas pela comunidade científica, as facilidades proporcionadas por essas ferramentas e suas limitações.

Uma ênfase maior foi dada à biblioteca ARToolkit. Detalhes de seu funcionamento foram descritos, incluindo a precisão do registro em suas aplicações, a confiabilidade no processo de detecção e os problemas potenciais, comuns aos desenvolvedores de sistemas RM, ainda não resolvidos pela biblioteca.

Entre esses problemas, destacam-se a adição de sombras de objetos virtuais no ambiente misturado e a sobreposição de objetos virtuais por elementos reais.

Capítulo 3

EXTRAÇÃO DE CHAVE E COMPOSIÇÃO DE IMAGENS

As imagens exibidas no vídeo, em transmissões de televisão ou em cenas de filmes, podem ser imaginadas como uma composição de duas camadas, uma contendo os objetos de primeiro plano (*foreground*) e outra contendo o plano de fundo (*background*).

Tanto a indústria da televisão (Gibbs *et al.*, 1998) quanto a indústria do cinema dispõem de recursos para utilizar uma camada de primeiro plano isolada, criando a necessidade de retirá-la de seu contexto original.

Esta separação, conhecida como extração de chave, tem como objetivo uma futura combinação com camadas de planos de fundo originadas de diversas fontes, que podem ser filmagens em outros locais ou imagens produzidas em computadores (Joshi *et al.*, 2006).

Uma definição simplificada da técnica, apresentada por Smith e Blinn (1996), a considera o problema da separação de um objeto de primeiro plano, normalmente não retangular, de um plano de fundo quase sempre retangular.

Muitos trabalhos relacionados à extração de chave são protegidos por

patentes (VAN DEN BERGH E LALIOTI, 1999), e alguns deles, descritos em Vlahos (1963) e em Vlahos (1971), são considerados pioneiros na utilização da técnica.

3.1 Processo evolutivo

Os primeiros métodos de extração de chave foram criados pela indústria cinematográfica, e seus princípios ainda são aplicados em processos modernos de composição de imagens digitais. Uma breve descrição sobre as origens e a evolução das técnicas que serviram de base para os modelos de composições atuais é apresentada por Chuang (2004). A evolução destas técnicas, segundo Chuang (2004), pode ser dividida segundo o exposto nas Subseções seguintes.

3.1.1 Composições ópticas

O desenvolvimento da técnica de composição de imagens foi direcionado principalmente para suprir as necessidades da indústria cinematográfica de combinar, de forma convincente, imagens filmadas em momentos ou locais diferentes, em uma única faixa de filme.

Uma das primeiras soluções propostas foi a técnica da "revelação dupla", na qual partes do filme eram bloqueadas no momento da primeira captura, para que, em um novo processo de filmagem, novos elementos fossem incluídos nessas áreas preservadas. No início do século passado, essas combinações eram realizadas por meio de impressoras ópticas que foram especialmente construídas para impressão em filmes.

Nascia, então, a composição óptica, gerada a partir de três pedaços de filme. O primeiro contendo a imagem de primeiro plano a ser sobreposta sobre um diferente plano de fundo; o segundo, monocromático e denominado *matte*, que determinava as áreas transparentes do filme de primeiro plano, e o terceiro contendo o novo plano de fundo.

A combinação desses filmes permitia, no momento da projeção, que a passagem da luz fosse obstruída em algumas áreas e permitida apenas nas regiões da imagem que se desejava exibir.

O termo *hold matte* representava exatamente o oposto do *matte* e cujo objetivo era preservar a região de interesse da imagem, tornando transparentes as demais regiões (Smith e Blinn, 1996). A máscara construída pelo *hold matte* tem correspondência direta com o recurso do canal alfa, utilizado nas técnicas digitais de composição, o que será analisado na Subseção 3.4.2.

Um exemplo do processo de separação dos elementos e combinação com o novo plano de fundo é mostrado na Figura 3.1, onde em (a), mostra-se a imagem do novo plano de fundo, que pode ser um filme. Obtém-se a imagem do elemento de primeiro plano (atriz), estando esta sempre à frente de um plano de fundo de cor constante (Figura 3.1b).

O *matte* monocromático pode ser criado utilizando filtros especiais que transformam as regiões azuis em brancas e todas as outras cores em regiões pretas (Figura 3.1c), ou vice-versa (Figura 3.1d), possibilitando a identificação da silhueta do objeto.

Estas máscaras definem as regiões do filme que serão exibidas na imagem final e as regiões em que o plano de fundo será visualizado. O filme, que agora possui quatro partes (os dois originais e as duas máscaras), é criado a partir da combinação dessas partes em camadas.

Primeiramente, o novo plano de fundo é sobreposto pela silhueta da atriz. Aplica-se o primeiro processo de gravação, cujo resultado é a imagem

(a) Novo plano de fundo

(b) Atriz filmada em frente ao fundo azul

(c) *Hold matte*

(d) *Matte*

(e) Resultado da primeira gravação

(f) *Matte* sobre a imagem da Figura(b)

(g) Resultado da segunda gravação

Figura 3.1: Processo Óptico de Composição de Imagens (Brain, 2007)

mostrada na Figura 3.1(e). As regiões preservadas do filme ("buraco" deixado pela máscara) devem, então, ser preenchidas.

A imagem da atriz mostrada na Figura 3.1(b) é sobreposta pela camada do filme mostrado na Figura 3.1(d), resultando na imagem da Figura 3.1(f). Esta imagem , então, é colocada sobre a parte do filme resultante do primeiro processo de gravação.

A máscara contrária (*Matte*) define as áreas que são preservadas da imagem da atriz à frente do plano de fundo azul. Em um segundo processo de gravação, obtém-se a composição mostrada na Figura 3.1(g) (Brain, 2007).

A maioria das técnicas ópticas de composições envolvia objetos ou atores fotografados em frente a cenários de cor uniforme, como ilustrado no exemplo, para isolá-los mais facilmente.

3.1.2 Composições digitais

A composição óptica começou a ser substituída no início da década de oitenta. O avanço da tecnologia digital tornou possível a utilização de computadores e programas especializados nas composições cinematográficas e televisivas.

Os conceitos e técnicas das composições ópticas continuam sendo aplicados diretamente em composições digitais. A técnica do "*matting* digital" surgiu para combinações de imagens digitais como um correspondente direto das técnicas ópticas.

Os equipamentos digitais auxiliaram o surgimento de melhorias nos processos de extração de chaves e combinação de imagens além de proporcionarem o desenvolvimento de novos métodos.

3.2 Principais abordagens para extração de chaves

As principais abordagens das técnicas de extração de chave para composição de imagens são diferenciadas principalmente por características relacionadas ao plano de fundo. Entre essas abordagens podem-se destacar o caso em que o plano de fundo é conhecido e o caso em que o plano de fundo é completamente arbitrário.

3.2.1 Plano de fundo conhecido

Aplica-se a técnica mais praticada de extração de chave sobre um

ambiente totalmente controlado. O princípio adotado consiste em fotografar o ator à frente de um plano de fundo de cor constante e extrair a imagem de primeiro plano a cada quadro de vídeo (CHUANG *et al.*, 2001).

Normalmente, a cor azul é adotada para esse plano de fundo (Figura 3.2). Essa escolha não é aleatória: sua utilização proporciona um melhor contraste, uma vez que grande parte das aplicações envolve atores em primeiro plano. O tom azul, independentemente de etnias, não é encontrado na pele humana, o que facilita a identificação do elemento de primeiro plano (VAN DEN BERGH E LALIOTI, 1999).

Esta característica tornou a abordagem conhecida como "*matting* de tela azul", embora outra denominação, mais genérica, "*matting* de cor constante" também seja aceita, pelo fato de existirem muitas implementações que utilizam outras constantes de cores, como o amarelo e, principalmente, o verde.

Figura 3.2: Técnica do plano de fundo conhecido (THINKQUEST, 2007)

Na indústria do cinema, é comum a utilização do termo *matting* ou *matte*, ao passo que os termos *key* ou *keying* são mais restritos à indústria da televisão. Isso se deve ao significado original do termo *matte* descrito na Subseção 3.1.1.

Essa diferenciação não se restringe simplesmente à variação na nomenclatura. Existem algumas diferenças fundamentais na forma como a técnica é utilizada na televisão e no cinema. Pode-se destacar o fato de que, no

cinema, os efeitos são aplicados geralmente na fase de pós-produção, em contraste com a televisão, em que todo o processo, na maioria das vezes, é aplicado em tempo real para transmissões ao vivo (THOMAS, 2006).

Na indústria televisiva, a técnica do *chromakey* é considerada clássica, e o processo é realizado por dispositivos de *hardware*, denominados *chromakeyer*, de alta qualidade e normalmente de alto custo (VAN DEN BERGH E LALIOTI, 1999).

3.2.2 Plano de fundo arbitrário

Uma abordagem mais recente para extração de chave de imagens é a aplicação de técnicas que agem sobre planos de fundo arbitrários. Esse processo é conhecido como "*matting* de imagem natural".

Extrair as camadas da imagem em um ambiente não controlado é um problema que tem sido alvo de muitas pesquisas e tem apresentado consideráveis evoluções. No entanto, a maioria das soluções existentes requer a intervenção de usuários, possuem normalmente um elevado tempo de execução e apresentam problemas potenciais com imagens mais detalhadas (JOSHI *et al.*, 2006).

Destaca-se, entre as principais pesquisas, o algoritmo proposto por Mitsunaga *et al.* (1995), para extração de uma determinada cor chave de um plano de fundo arbitrário. O método apenas reduz a tarefa manual de operadores, mas proporciona uma interação mais intuitiva.

O algoritmo proposto por Zongker *et al.* (1999) realiza a extração de objetos de primeiro plano com informações de características que permitem uma composição com efeitos de suavização de bordas, refração e reflexão de objetos, mas possui elevado tempo de execução, o que o torna inviável para

utilização em aplicações de tempo real.

Para realizar o *matting* de imagem natural, outro método que apresenta resultados satisfatórios em imagens complexas, utiliza a equação matemática de Poisson com um campo para o *matte*. O algoritmo proposto por Sun *et al.* (2004) necessita de entradas de usuários para ajustes e refinamentos, e seu tempo de execução por quadro de imagem pode atingir a ordem de minutos.

McGuire *et al.* (2005) apresentaram uma abordagem completamente automática para extrair elementos em imagens com planos de fundo arbitrários. O método utiliza três tomadas de vídeos sincronizadas que compartilham um mesmo ponto de projeção, mas são diferentes em seus planos de foco. O processo, baseado na minimização de erros dessas tomadas de vídeos, possui muitas limitações, dentre as quais deve ser destacada o tempo de processamento fora dos parâmetros mínimos para execução em tempo real.

A solução para o *matting* de imagem natural, apresentada por Joshi *et al.* (2006), utiliza um conjunto de câmeras focadas no objeto de primeiro plano, as quais captam as altas freqüências de cenas naturais, reduzindo o contraste dos *pixels* do objeto de primeiro plano e aumentando o contraste dos *pixels* do plano de fundo (Figura 3.3).

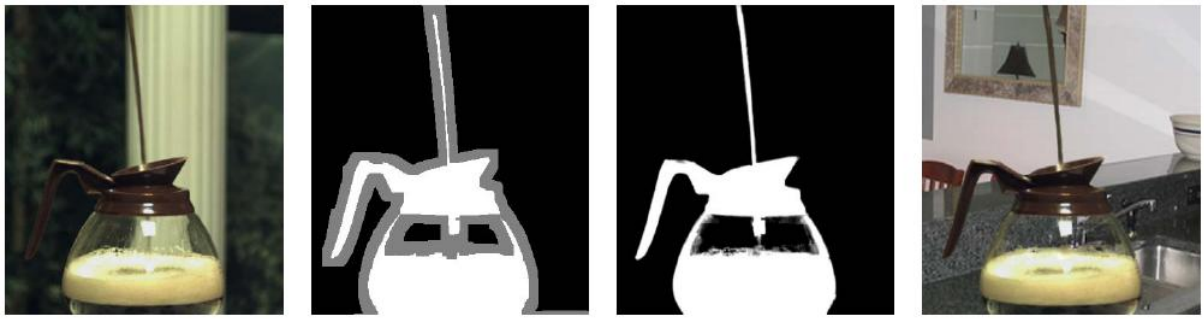


Figura 3.3: Técnica do plano de fundo arbitrário (Joshi *et al.*, 2006)

O processo é totalmente automático e seu desempenho é proporcional

ao número de câmeras do conjunto. Esse sistema pode ser considerado o primeiro a realizar composições de imagens de alta qualidade em ambientes arbitrários, com velocidades próximas as aceitáveis em sistemas de tempo real.

Como se pode observar, a maioria das soluções apresentadas para extração de chave de ambientes arbitrários utiliza-se de técnicas quase manuais para encontrar a máscara do objeto de primeiro plano. Nenhuma das abordagens existentes, para essa situação, apresenta resultados robustos ou são efetivas, e são raros os dispositivos comerciais que permitem algumas operações desse tipo (THOMAS, 2006).

Uma abordagem simples que pode ser adotada, sem utilização de fundos de cor única, é a comparação da imagem atual com uma imagem estática da cena, antes de elementos de primeiro plano serem inseridos no ambiente.

A idéia é obter-se uma imagem binária que funcione como o *matte*, que, multiplicado pela imagem da câmera, resulte em uma imagem do elemento de interesse sem o fundo original. Funções para utilização desse artifício são oferecidas pela biblioteca OpenCV, bem como métodos de eliminação de ruídos na máscara gerada (BERNARDES *et al.*, 2007).

3.3 *Chromakey*

A técnica do *chromakey*, há algum tempo, é bastante utilizada em produções de televisão, inclusive para transmissões ao vivo. Seu princípio baseia-se na identificação de partes da imagem, em que se encontra uma determinada cor, denominada "cor chave". Uma máscara é então gerada para selecionar quais *pixels* devem permanecer e quais devem ser substituídos por *pixels* de outra imagem, utilizada como novo plano de fundo (VAN DEN

Bergh e Lalioti, 1999).

A complexidade do problema é melhor entendida quando, por exemplo, todo o processo deve ser realizado em tempo real, quando a superfície do objeto a ser extraído é transparente, se existirem sombras a serem preservadas ou retiradas, ou, quando, na silhueta do objeto em primeiro plano, houver elementos como cabelos e pêlos.

Escolhe-se freqüentemente o azul para compor o plano de fundo não apenas por haver melhor adaptação em relação aos tons de cores da pele humana, como descrito na Subseção 3.2.1, mas também pelo fato de, historicamente, câmeras e filmes serem mais sensíveis a luzes azuis.

A tecnologia atual, no entanto, torna essa constatação menos verdadeira, porquanto canais de verde são mais reflexivos que canais de azul e podem ser identificados mais facilmente (Seanet Corporation, 2007).

Conceitualmente, a operação *chromakey* é simples, mas apenas a comparação de *pixels* do objeto de primeiro plano com a cor chave, selecionada automaticamente ou por meio de um operador, pode conduzir a composições grosseiras e pouco naturais.

Abordagens muito simplificadas produzem uma série de imperfeições e, por esse motivo, sistemas modernos que utilizam a técnica aplicam algoritmos para preservar detalhes do objeto a ser isolado (Gibbs *et al.*, 1998). Alguns problemas potenciais são comuns à técnica, caso nenhum controle adicional seja implementado. Podem-se observar imperfeições nas bordas ao redor dos objetos de primeiro plano extraídos, sombras sobre o plano de fundo podem ser entendidas como elemento de primeiro plano, e a câmera de vídeo deve permanecer imóvel durante o processo, para que se mantenha a da coerência entre as camadas de primeiro e segundo planos.

Uma movimentação lateral fará com que o ator ou objeto de primeiro plano pareça deslizar em cena para um dos lados, enquanto uma aproximação

ou afastamento fará com que esse aumente ou diminua de tamanho (GIBBS *et al.*, 1998).

O fato de o ator possuir, em seu vestuário, alguma peça da mesma cor do plano de fundo, ou o objeto a ser extraído possuir partes dessa mesma cor constitui uma limitação implícita dessa abordagem. Nesse caso, a parte coincidente é extraída juntamente com o plano de fundo no decorrer do processo.

Na tentativa de contornar esse problema, Yamashita *et al.* (2002) propuseram um método em que o ator é fotografado à frente de um plano de fundo composto por listras de duas cores diferentes.

No momento da extração do ator ou objeto de primeiro plano em que a cor chave é localizada, para que esta seja considerada como pertencente ao plano de fundo, são verificados também os *pixels* vizinhos ao *pixel* analisado, na tentativa de localização de uma segunda cor chave. Caso esta condição de adjacência não seja satisfeita, o *pixel* será considerado como pertencente ao primeiro plano.

O método apresenta resultados satisfatórios e extrai o objeto de primeiro plano com correção, independentemente da sua cor, embora ainda possua limitações.

O elevado tempo de processamento o torna inviável para aplicações de tempo real, e o controle da iluminação e das dimensões dos padrões de listras deve ser bastante rígido. O problema mais grave, embora de rara ocorrência, é a possibilidade de o objeto ou o ator de primeiro plano possuir partes com padrões de listras semelhantes ao padrão do fundo. Nesse caso, a parte coincidente é extraída juntamente com o plano de fundo.

Outra solução apresentada, que se refere a ambientes controlados, propõe fotografar o mesmo objeto de primeiro plano em frente a diferentes planos de fundo, mas o método, embora propicie excelentes resultados na

composição final, ainda não é adequado para produções ao vivo e necessita de um controle do ambiente ainda mais rígido que o *chromakey* convencional (Smith e Blinn, 1996).

Uma dificuldade a ser contornada, e que não está restrita a planos de fundo constantes, é o *Color Spill*. Esse é o problema da contaminação do objeto de primeiro plano com as cores do plano de fundo.

O *Collor Spill*, mais visível em fundos verdes devido a estes serem mais reflexivos, ocorre nas bordas dos objetos de primeiro plano, onde os *pixels* acumulam iluminação de ambas as camadas, ou, em casos de objetos transparentes de primeiro plano em que luzes refletidas os atravessam e retornam à câmera.

Esta contaminação provoca a necessidade de remoção dessa impureza, antes da combinação do objeto de primeiro plano com o novo plano de fundo (Gvili *et al.*, 2003).

3.3.1 **Ultimatte**

Um modo bastante sofisticado de realizar *chromakey* é por meio do equipamento Ultimatte (Figura 3.4). Rápido o bastante para ser executado em transmissões ao vivo, o Ultimate é amplamente utilizado em produções de vídeos há algum tempo. Esse equipamento possibilita a criação de composições com elementos como fumaça, objetos transparentes, diferentes tons de azul e sombras.

Ultimatte é uma marca registrada da Ultimatte Corporation, da cidade de Chatsworth na Califórnia. Fundada por Petro Vlahos em 1960 para o *Motion Picture Research Council*, com o objetivo de ter o melhor sistema para composições de imagens em movimento, a companhia americana também

Figura 3.4: Equipamento Ultimatte (Ultimatte Corporation, 2007)

fabrica *softwares* que trabalham em conjunto com outros programas para criação de composições digitais.

É mais correto imaginar o funcionamento do Ultimatte como um processo completo e não apenas de extração de chave, embora sua principal funcionalidade esteja relacionada ao problema da separação. O equipamento utiliza a intensidade e a pureza do canal de azul como uma função para determinar a quantidade da mistura entre o primeiro e o segundo plano.

Os modelos mais modernos permitem um ajuste independente das cores para as camadas de primeiro plano e plano de fundo, e esses ajustes são feitos por meio de botões distribuídos em um painel. Os modelos digitais utilizam um dispositivo com uma tela e dispõem de recursos bastante sofisticados.

Outra característica importante do Ultimatte é uma função de correção de tela. Esse recurso permite a um operador a criação de composições de alta qualidade a partir de planos de fundo imperfeitos. Existem também funções para retenção de sombras sobre o plano de fundo, ao invés de tentar criá-las, como acontece na maioria das abordagens.

Um problema comum a métodos de tela azul, presente no Ultimatte, são as demais superfícies azuis do cenário. Invariavelmente estas possuem diferentes tons da cor azul porque a luz reflete sobre elas de diferentes ângulos.

Esse efeito, em alguns casos, pode ser removido por meio de um filtro de polarização.

Outra restrição importante do equipamento é a impossibilidade da utilização de dispositivos que regulam a intensidade da luz que ilumina o plano de fundo. Baixando a intensidade da luz com um dispositivo desse tipo, baixa-se também sua temperatura, tornando-a mais alaranjada, e, em conseqüência, afetando o plano de fundo, de modo a torná-lo mais distante do azul inicialmente configurado.

Existem vários modelos do equipamento Ultimatte, com características diversificadas. Os preços variam em uma faixa entre U$ 9,500.00 do modelo U-400 até os U$ 85,000.00 do modelo U-HD (Ultimatte Corporation, 2007).

3.3.2 **Estúdios virtuais**

As técnicas convencionais de *chromakey* necessitam de um controle bastante rígido do ambiente para que se obtenha resultados satisfatórios.

Estas exigências, somadas às suas várias limitações, resultaram no desenvolvimento de muitos métodos para estender a utilização do *chromakey* e atender à exigente indústria da televisão (Gibbs *et al.*, 1998). Tais métodos permitem que, por meio de imagens geradas pela computação gráfica, estúdios virtuais possam ser utilizados facilmente para a produção de programas.

Muitos produtos comerciais, baseados em supercomputadores, envolvem um conjunto de recursos bastante sofisticados para alcançar esse objetivo e torna possível a realização de perfeitas composições, inclusive em tempo real, de vídeos exibidos ao vivo com imagens sintéticas ou mesmo

imagens naturais.

Os modelos gráficos gerados por computador podem ser renderizados na tela a cada quadro de vídeo por meio de *hardware* específico de alta qualidade. Sistemas de controle permitem que a câmera possa ser movimentada, fornecem um controle rígido de iluminação e possibilitam que objetos de primeiro plano ou planos de fundo virtuais possam ser redesenhados em uma taxa de 50 a 60 vezes por segundo, para acompanhar o ponto de visão da câmera (THOMAS, 2006).

O conjunto envolvendo *hardware* e *software* necessários para realizar essa composição, permitindo essa liberdade de movimentação, possibilitou que modelos virtuais fossem utilizados em maior escala. O termo "Estúdio Virtual" foi criado para distinguir esses novos sistemas das técnicas mais comuns de *chromakey* (THOMAS, 2006).

Os Sistemas de Estúdios Virtuais, apesar de haver abordagens alternativas, normalmente iniciam a partir de protótipos experimentais que são extensões do *chromakey*; portanto, são definidos como técnicas avançadas do método tradicional. Alguns exemplos de Estúdios Virtuais, mostrados em (Gibbs *et al.*, 1998), são descritos nas próximas Subseções.

3.3.2.1 O modelo do Synthevision

Um precursor da idéia de Estúdios Virtuais foi o método de realizar *chromakey* desenvolvido por pesquisadores da companhia japonesa NHK. O Synthevision, que atualmente é um produto, apareceu pela primeira vez durante as Olimpíadas de 1988, em Seul, na Coréia do Sul.

O método idealizado pela NHK, que trabalha com sensores presos à cabeça, permite simular os movimentos da câmera sobre o plano de fundo,

produzindo efeitos de mudança de ponto de visão, inclinação, e aproximações.

Todos os movimentos da câmera são, dessa forma, reproduzidos em tempo real, provocando uma alteração de perspectiva do plano de fundo para acompanhar o objeto de primeiro plano. A combinação é realizada por meio do *chromakey* tradicional nas camadas modificadas de primeiro e segundo plano, como mostrado na Figura 3.5.

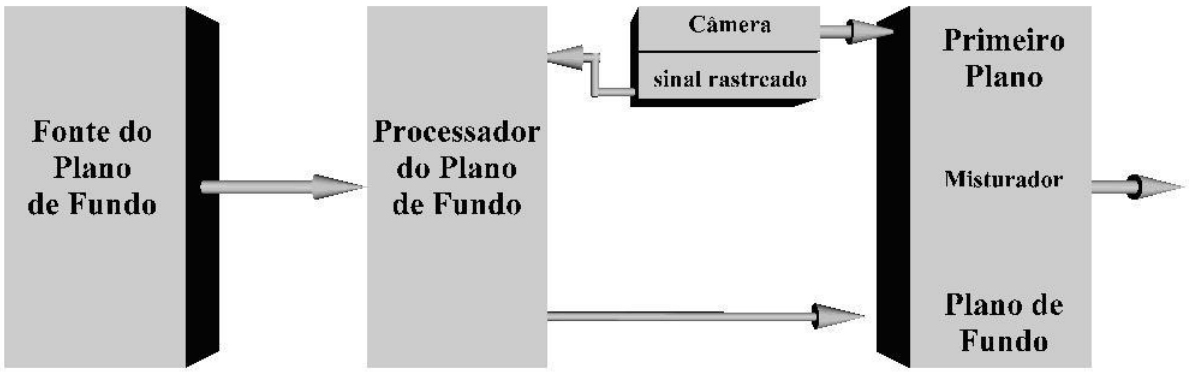


Figura 3.5: Modelo do Synthevision (GIBBS *et al.*, 1998)

Para conseguir esse tipo de efeito, o Synthevision utiliza como base para o plano de fundo uma HDTV (*Hight-Definition Television*), e o tamanho desse dispositivo é que determina a área de imagem adaptada a suas modificações de perspectivas (GIBBS *et al.*, 1998).

3.3.2.2 Agrupamento de câmeras

Sugerindo a idéia de estúdio virtual, outro método para realizar composições é o "agrupamento de câmeras". Nesse método, utilizam-se duas câmeras reais de forma a obedecerem a um relacionamento do tipo "mestre e escravo".

Normalmente, a câmera responsável pela geração do primeiro plano opera como câmera mestra e toda sua movimentação é acompanhada pela câmera responsável pelo plano de fundo, como mostrado na Figura 3.6.

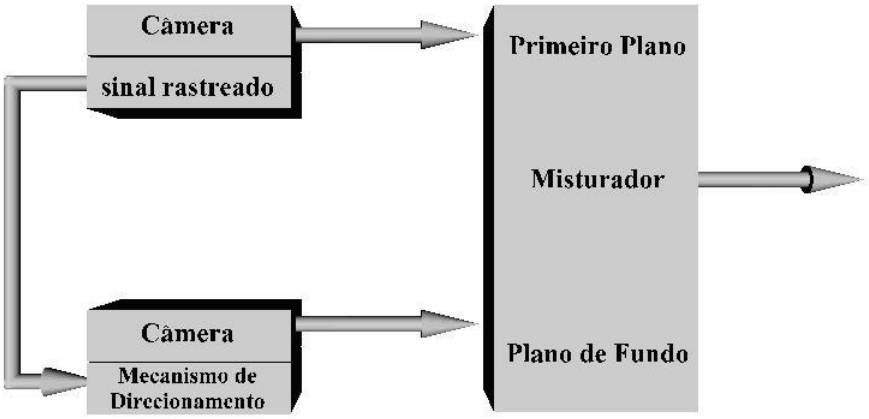


Figura 3.6: Modelo do Agrupamento de Câmeras (GIBBS *et al.*, 1998)

Esta técnica é comum na produção de efeitos especiais e proporciona ótimos resultados, principalmente quando as duas camadas da imagem combinada exibem vídeo ao vivo, ao invés de computação gráfica.

São várias as maneiras de controlar e sincronizar o movimento das câmeras para aplicação desse método. Uma das formas é armazenar os dados de movimentação da câmera mestra e utilizá-los para direcionar a câmera escrava.

Outra forma é conectá-las fisicamente, de modo que a câmera escrava acompanhe os movimentos da câmera mestra pela movimentação de um único operador. Esta sincronização também pode ser realizada por algum tipo de controle externo, como um computador, responsável pela movimentação do conjunto.

3.3.2.3 Cenário virtual pré-computado

Uma outra técnica utilizada em Estúdios Virtuais é bastante útil, quando se conhece previamente toda movimentação da câmera e a composição do cenário. Dessa maneira, é possível construir antecipadamente todo o conjunto virtual e executá-lo com as entradas da câmera por meio de *chromakey*.

Esta técnica pode ser executada rastreando a movimentação da câmera durante o processo de gravação dos movimentos do objeto, ou analisando, quadro a quadro, essa movimentação, não necessariamente em tempo real (Figura 3.7).

Posteriormente, os dados obtidos são utilizados para posicionar a câmera que renderiza o cenário virtual. Nesse método, ambas as camadas da imagem são combinadas depois de compostas.

Isso também pode ser feito, controlando a movimentação da câmera por meio de programas de computador próprios para animação, que armazenam toda movimentação da câmera responsável pela geração do plano de fundo. A produção do objeto de primeiro plano, nesse caso, deve obedecer à seqüência pré-definida do plano de fundo, o que permite inclusive composições com vídeo ao vivo.

3.4 Algoritmos para implementação do Chromakey

Nos Estúdios Virtuais, o processo *chromakey* é feito por meio de dispositivos de *hardware*, normalmente de alto custo, tornando inviável sua aplicação em determinadas situações, principalmente nas em que as exigên-

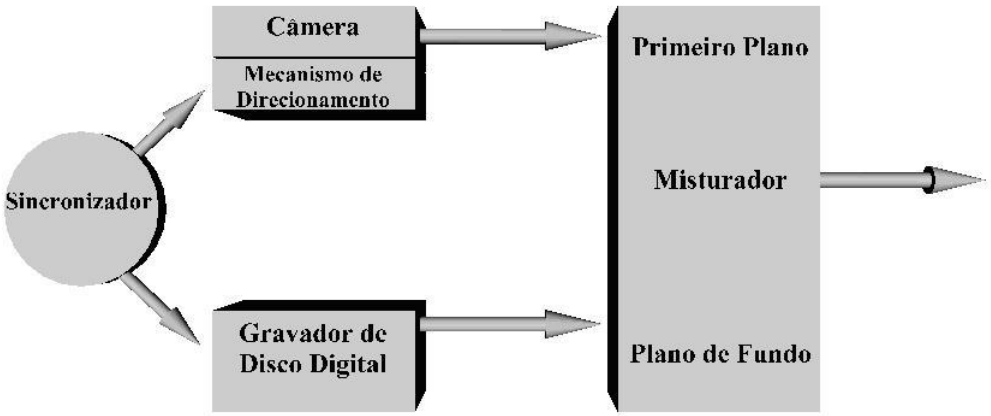


Figura 3.7: Modelo do Cenário Virtual Pré-Computado (GIBBS *et al.*, 1998)

cias não são tão rígidas (VAN DEN BERGH E LALIOTI, 1999). A evolução dos computadores pessoais utilizados com dispositivos gráficos torna perfeitamente viável uma implementação em *software*.

A tarefa mais importante de um algoritmo *chromakey* é classificar um *pixel* como sendo da cor chave ou não (VAN DEN BERGH E LALIOTI, 1999). Por esse motivo, a compatibilidade do modo de representação do espaço de cores entre a implementação e o dispositivo utilizado é determinante, especialmente em aplicações de tempo real, em que conversões têm impacto significativo no tempo de execução.

3.4.1 Sistemas de cores

As diferentes representações, implementadas em dispositivos, obrigam o sistema a identificar, de alguma forma, sob qual espaço de cores a imagem foi construída.

Esta tarefa fica a cargo de alguma ferramenta embutida no Sistema

Operacional, que normalmente fornece um tipo de representação visual do modelo de cores, organizada em um espaço tridimensional, e algum modo de modificar parâmetros nesse espaço (Douglas e Kirkpatrick, 1999). Alguns desses modelos, propostos pela comunidade da Computação Gráfica e suportados por dispositivos de visualização, podem ser destacados.

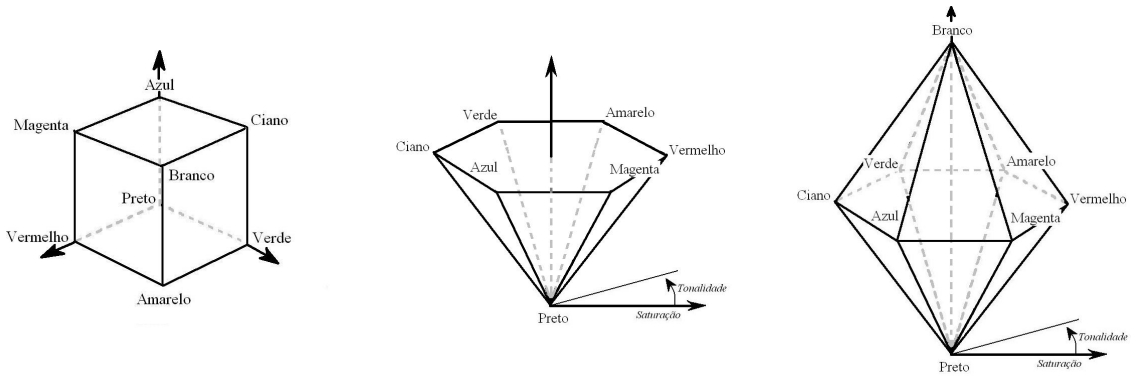
O modelo mais comumente encontrado, o RGB (*Red*, *Green* e *Blue*), possui como cores primárias o vermelho, o verde e o azul. Esse modelo utiliza um sistema de coordenadas cartesianas R, G, B, cujo espaço de representação é um cubo (Figura 3.8a).

Baseado na sensibilidade do olho humano, o modelo não consegue reproduzir algumas cores pela sobreposição das três cores primárias. Isso significa que algumas cores na natureza não podem ser mostradas nesse sistema de representação (Azevedo e Conci, 2003).

O modelo HSV (*Hue*, *Saturation* e *Value*) foi baseado na maneira pela qual um artista descreve e mistura as cores. O sistema de coordenadas é representado por um cilindro e o subconjunto, dentro do qual o modelo é definido, é um hexágono na base de um cone (Figura 3.8b). A tonalidade, a saturação e a luminância são os parâmetros de cor utilizados nesse sistema.

As várias tonalidades estão representadas na parte superior e exibem-se pelo ângulo ao redor do eixo vertical, sendo o vermelho igual a 0^o, o verde igual a 120^o e, assim, sucessivamente. A saturação é medida ao longo do eixo horizontal, e a luminância ao longo do eixo vertical que passa pelo centro do cone (Azevedo e Conci, 2003).

Um modelo alternativo para o HSV, bastante popular entre programadores, é o HLS (*Hue*, *Lightness* e *Saturation*). Esse modelo, baseado na tonalidade, brilho e saturação, faz-se representar por um hexágono duplo na base de dois cones, como mostrado na Figura 3.8(c) (Azevedo e Conci, 2003).



(a) Modelo RGB **(b)** Modelo HSV **(c)** Modelo HLS

Figura 3.8: Representações de cores (Azevedo e Conci, 2003)

Além dos modelos de cores mostrados, também podem ser citados o modelo CMYK (Ciano, Magenta, Amarelo e Preto) e o YIQ — utilizado em transmissões de televisão em cores nos Estados Unidos (Azevedo e Conci, 2003).

Muitos autores classificam essas representações considerando que algumas sejam mais "naturais" que outras e, conseqüentemente, são de mais fácil utilização. O modelo HSV, por exemplo, é considerado um modelo mais próximo da percepção humana. O RGB, por sua vez, é classificado como orientado à máquina (Douglas e Kirkpatrick, 1999). De fato, cores próximas no espaço RGB, muitas vezes, são bastante diferentes na percepção visual humana.

As principais abordagens da técnica do *chromakey* têm seu desenvolvimento baseadas no espaço RGB, provavelmente devido a maioria dos dispositivos de visualização utilizar essa representação do espaço de cores. Apesar disso, algumas abordagens baseadas em representações mais perceptivas foram propostas, apoiando-se em uma maior facilidade de isolamento de faixa de cores.

3.4.2 Canal alfa

Na representação da cor de um ponto na tela, armazena-se normalmente apenas a informação dos valores que produzem a cor. Quando se sobrepõe uma camada de imagem, contendo um objeto ou um ator, à outra camada, contendo um plano de fundo completamente arbitrário, um novo elemento, denominado canal alfa (α), pode ter sua relevância.

Esse artifício é utilizado para representar níveis de transparência de *pixels* da imagem de primeiro plano e torna possível a separação de elementos de interesse, a produção de efeitos de suavização nos pontos de junção entre duas camadas de imagem e a preservação da transparência de objetos. Em termos matemáticos, permite que cada *pixel* possa controlar a interpolação linear entre as cores das duas camadas (PORTER E DUFF, 1984).

O canal alfa, responsável pela informação de transparência desses *pixels*, contém valores que variam em um canal em escala de cinza (PORTER E DUFF, 1984). Aos *pixels* que não devem ser exibidos, deve ser atribuído um valor alfa igual a zero, ao passo que um valor máximo deve ser atribuído aos *pixels* a serem totalmente preservados. Os *pixels* parcialmente visíveis devem conter valores de alfa intermediários.

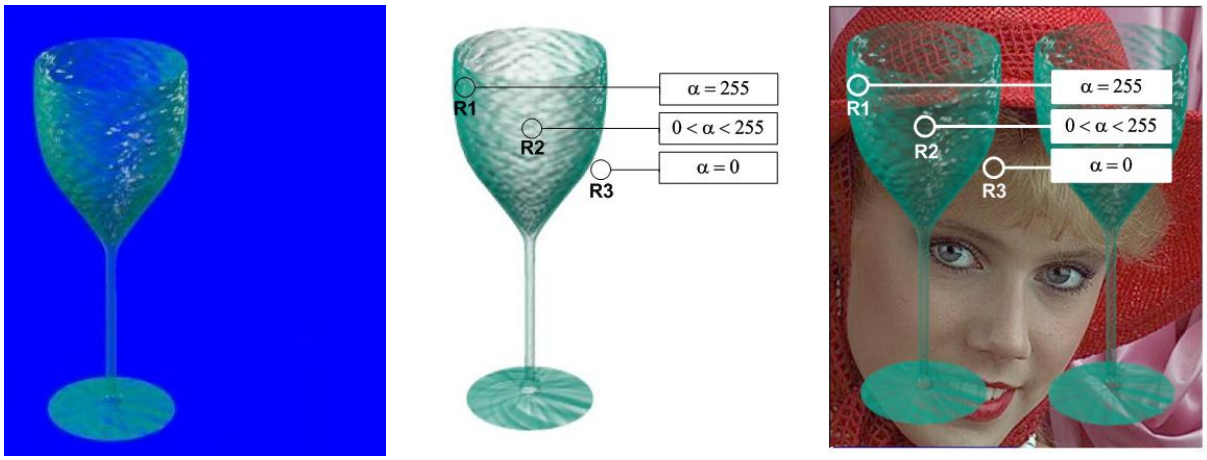
Sua utilização possibilita a criação de imagens digitais complexas e, por esse motivo, a maioria dos métodos de composição é baseada na "Equação da Composição", que faz uso do canal alfa (Equação 3.1).

$$C = \alpha F + (1 - \alpha)B \tag{3.1}$$

onde: C é a cor resultante mostrada na composição;
F é a cor do elemento de primeiro plano (*foreground*);
B é a cor do novo plano de fundo (*background*); e

α é componente de transparência do *pixel*.

Na Figura 3.9, demonstra-se como o canal alfa pode ser utilizado para compor uma imagem complexa com qualidade. Os valores de alfa do elemento de primeiro plano, nesse exemplo, variam entre 0 e 255. No entanto, em muitas implementações, utilizam-se valores que variam entre 0 e 1, como ocorre também com os valores RGB convencionais.



(a) Objeto a ser extraído **(b)** Extração do objeto **(c)** Composição final

Figura 3.9: Valores de alfa no elemento de primeiro plano (Smith e Blinn, 1996)

Na Figura 3.9(a), exibe-se o objeto de interesse a ser extraído do seu contexto original, colocado à frente de um plano de fundo azul. Em seguida, o objeto é extraído e apenas as regiões de interesse são preservadas (Figura 3.9b). Os valores de alfa dos *pixels* da região (R1) permanecem 255, por que esses *pixels* devem ser totalmente preservados. Esses valores passam a ser 0 nos *pixels* destacados na região (R3), que não serão exibidos na imagem final. Valores de alfa intermediários são atribuídos aos *pixels* da região (R2), cujas tonalidades terão influência em sua cor final pelos *pixels* correspondentes (da mesma posição) do novo plano de fundo.

Na Figura 3.9(c), a composição final pode ser observada. A região (R1) permanece com a cor do elemento de primeiro plano, a região (R3) é

totalmente ocupada pelo novo plano de fundo, e a região (R2) é composta por *pixels* semitransparentes. Nessa região, tanto o objeto de primeiro plano, quanto o novo plano de fundo são visíveis.

Uma maneira conveniente de agregar o valor alfa no espaço de cores RGB é incluí-lo como uma extensão dos valores convencionais $Cor = [RGB\alpha]$ e previamente multiplicá-lo por cada um dos canais de cor (Smith e Blinn, 1996).

3.4.3 **Isolamento do fundo azul**

Na busca de soluções para a eliminação do plano de fundo, Smith e Blinn (1996) provaram que, utilizando a abordagem descrita na Subseção 3.4.2, não seria possível uma solução geral para a técnica de extração de chave com plano de fundo de cor única. No entanto, uma solução seria viável: caso se houvesse uma restrição ao objeto de primeiro plano quanto à existência de componentes da mesma cor do plano de fundo. Por exemplo, o azul.

Embora aparentemente óbvia, a restrição de todos os tons de azul pode conduzir a resultados indesejados. A conseqüência desse procedimento, aplicado a cores na representação RGB, é a exclusão também de todos os tons de cinza, exceto o preto. Isso representa dois terços da tonalidade toda, além de se considerar que outras tonalidades, como o branco, possuam azul.

Outro problema incide na dificuldade de um controle rígido a um ambiente afetado pelas condições de iluminação. Qualquer impureza sobre o plano de fundo torna-o não totalmente azul. Para minimizar esse problema, os algoritmos utilizados fazem uma restrição a um espaço de cor, e não apenas a uma constante única para cada canal de cor (Smith e Blinn, 1996).

Baseados nessas considerações, Smith e Blinn (1996) sugeriram que,

para C_o (cor do objeto) não contendo azul, a Equação 3.2 significa a solução, a que denominaram "problema do *matting*".

$$C_o = [R_f \;\; G_f \;\; 0 \;\; 1 - \frac{B_f}{B_k}] \tag{3.2}$$

onde: R_f é a canal vermelho do elemento de primeiro plano;
G_f é o canal verde do elemento de primeiro plano;
B_f é o canal azul do elemento de primeiro plano; e
B_k é o canal azul do plano de fundo.

Outra solução proposta é restringir o objeto de primeiro plano a tons de cinza. Nesse caso, a solução seria a Equação 3.3.

$$C_o = [R_f \quad G_f \quad B_f - B_k + \alpha_o B_k \quad \frac{G_f - (B_f - B_k)}{B_k}] \tag{3.3}$$

onde: R_f é a canal vermelho do elemento de primeiro plano;
G_f é o canal verde do elemento de primeiro plano;
B_f é o canal azul do elemento de primeiro plano;
B_k é o canal azul do plano de fundo; e
α_0 é o componente alfa do objeto;

Como descrito na Seção 3.3, Smith e Blinn (1996) ainda apresentaram uma solução geral para a técnica, mas o método requer que o objeto de primeiro plano seja fotografado à frente de dois planos de fundo de cores diferentes. Apesar de o resultado final ser bastante satisfatório, esse procedimento torna o método inviável para aplicações em tempo real.

Um dos métodos de isolamento de cores proposto por van den Bergh e Lalioti (1999) é baseado no espaço de cores HLS. A representação geométrica do HLS, como discutida na Subseção 3.4.1, é um hexágono na base de dois cones que visto de cima tem a forma exibida na Figura 3.10.

Faz-se a representação da cor pelo ângulo ao redor da borda, que inicia do vermelho 0°, enquanto a posição do azul está em 240°. Para eliminar canais de azul, basta isolar o seu espaço de cores, que está localizado entre 230° e 250°.

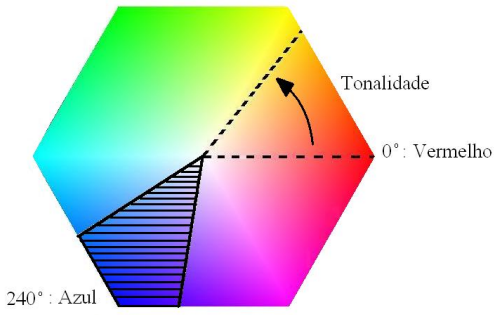


Figura 3.10: Hexágono HLS (VAN DEN BERGH E LALIOTI, 1999)

A escolha dos ângulos é arbitrária e deve ser feita de modo a melhor adequar-se à tonalidade do fundo azul da aplicação.

Esta abordagem é bastante eficiente quando aplicada a imagens no modelo de cores HLS. Do contrário, a demorada conversão para o padrão RGB tem impacto significativo no tempo de processamento. Uma maneira de reduzir esse atraso é implementá-lo de modo a converter apenas os canais de azul (VAN DEN BERGH E LALIOTI, 1999).

Uma abordagem adequada ao espaço de cores RGB, também proposta por van den Bergh e Lalioti (1999), requer um máximo de cinco operações por *pixel*. Os critérios a serem obedecidos estabelecem que $B > R$ e $B < G$, pois, em *pixels* azuis, o canal de azul deve ser predominante.

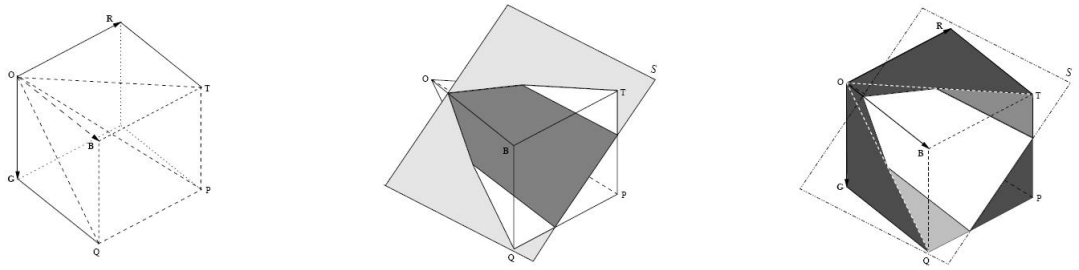
Esta simples restrição ainda não se mostrou suficiente, uma vez que, quando se tem $B = x$, $R = x - 1$ e $G = x - 1$, observa-se uma tonalidade de cinza classificada como azul, segundo o critério estabelecido.

A maneira adotada para melhorar o critério foi estendê-lo para representar uma distância (d_{max}) entre um canal de cor e um plano, de modo a

assegurar que o ponto reconhecido seja realmente azul. A representação desta distância é mostrada na Equação 3.4. Uma simplificação pode ser realizada para evitar o elevado tempo de processamento das operações com raízes e expoentes.

$$d = \sqrt{(B-R)^2 + (B-G)^2} > d_{max} \qquad d = 2 \times B - R - G \quad (3.4)$$

Na Figura 3.11(a) exibe-se um cubo RGB contendo uma pirâmide inclinada $OBTPQ$ nas linhas pontilhadas. Esta pirâmide define o volume do cubo em que $B \geq R$ e $B \geq G$. Na Figura 3.11(b) é mostrada a pirâmide inclinada $OBTPQ$ cortada pelo plano S. Esse plano, paralelo à diagonal principal OP, representa a superfície da constante d. A Equação 3.4 representa os pontos no interior da pirâmide com predominância do azul.



(a) Pirâmide $OBTPQ$ **(b)** Corte do plano S **(c)** Cubo sem o volume acima do plano

Figura 3.11: Método de Bergh e Lalioti (VAN DEN BERGH E LALIOTI, 1999)

O polígono cinza escuro é a superfície de intersecção da pirâmide inclinada com o plano S. Os pontos dentro do polígono representam as cores do espaço RGB com um valor específico de d. A diagonal OP são as tonalidades de cinza do cubo e, como todas as cores nesta linha, têm componentes RGB equivalentes. O plano S é paralelo à diagonal OP, representando que todos os pontos do polígono cinza são igualmente azuis.

A Equação d, de uma determinada cor dentro da pirâmide inclinada, pode ser utilizado para classificá-la como azul ou não. Esse valor de distância não é uma distancia métrica, mas uma aproximação computacional.

Na Figura 3.11(c), exibe-se o cubo sem o volume acima do plano S. Os triângulos cinza mais claros são os lados da pirâmide visíveis acima do plano. O plano S é a posição exata do valor d_{max} e funciona como um valor limite. Calcula-se o valor d em cada ponto, e a máscara é gerada de acordo com os resultados destas comparações.

A máscara gerada não é binária, o que permite a utilização do canal alfa para os efeitos de transparência nas composições. Utiliza-se distância d calculada como entrada para a função alfa que retorna um valor de transparência entre 0 e 255, baseada na distância do ponto da diagonal OP no espaço RGB. Na Figura 3.12, é apresentado o laço principal do algoritmo implementado em C++.

```
unsigned char *rp=s,*gp=s+1,*bp=s+2,*ap=s+3;
    for (int i = 0; i < imgsize; i++) {
        if (*bp > *rp && *bp > *gp )
            *ap = alpha_map[(*bp<<1) - *rp - *gp];
        else
            *ap = 255;
            rp += 4; bp += 4; gp += 4; ap += 4;
    }
```

Figura 3.12: Código C++ da implementação do algoritmo (VAN DEN BERGH E LALIOTI, 1999)

3.5 Considerações Finais

No presente Capítulo, procurou-se apresentar as técnicas de extração de elementos de primeiro plano para composições digitais, utilizadas na in-

dústria de cinema e da televisão.

Além de abordar a evolução dos métodos, aqui se discutiram os detalhes das composições ópticas e a mudança da tecnologia óptica para a digital. Relativamente ao tipo de plano de fundo, analisaram-se as principais abordagens para composições digitais, com ênfase nas técnicas que utilizam planos de fundo de cor única.

A técnica de *chromakey* foi apresentada por meio de análise rica em detalhes, em que se incluíram os principais equipamentos utilizados — particularmente em Estúdios Virtuais —, alguns exemplos dos quais foram demonstrados, destacando-se os métodos utilizados para contornar as restrições impostas pela técnica tradicional do *chromakey*.

Avaliaram-se os principais algoritmos para implementação do *chromakey*, propostos pela comunidade científica. Para tanto, procurou-se expor os sistemas de espaços de cores em que se baseiam.

Ademais, discutiu-se o recurso do canal alfa — que interfere no sistema de cores RGB —, e sua importância no processo de isolamento do fundo da cena.

Capítulo 4

PROFUNDIDADE EM AMBIENTES MISTURADOS E O OPENGL *FRAMEBUFFER*

No processo de combinação, a representação correta da profundidade dos elementos da cena é essencial à obtenção de níveis elevados de realismo.

Elementos reais e virtuais devem ser exibidos de forma coerente, respeitando suas distâncias em relação ao observador. Para isso, são necessários métodos que possibilitem a esses elementos obstruírem-se mutuamente durante a execução da aplicação.

4.1 Oclusão mútua

A maioria das ferramentas utilizadas no desenvolvimento de aplicações de RM resolve de maneira aceitável o problema do posicionamento do objeto virtual, mas "ignoram" a localização dos objetos reais em cena.

Esta limitação faz com que os objetos virtuais, em muitas aplicações, sejam renderizados à frente da imagem real, que, por sua vez, é considerada

como plano de fundo durante todo o tempo de execução do programa (Figura 4.1).

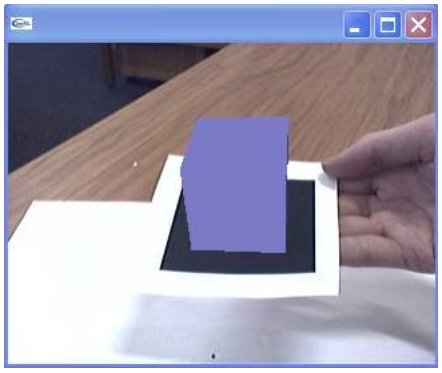


Figura 4.1: Objeto virtual (cubo) gerado sobre o marcador real (HIT LAB, 2006)

Denomina-se "oclusão mútua" o equacionamento a esse problema. A oclusão mútua se dá entre os objetos reais e virtuais dispostos na cena, o que aumenta ao usuário a sensação de que os objetos virtuais realmente existem no mundo real (KIYOKAWA *et al.*, 2000).

Para que se realize esse tipo de efeito em tempo real, evidencia-se a necessidade de algum método de estimativa de profundidade, não só de objetos virtuais mas também de objetos reais em cena.

Soluções para esse tipo de combinação, embora raras, têm surgido principalmente para produção de alguns efeitos em transmissões de televisão. Nessas produções, muitas vezes é desejável a construção de objetos virtuais atrás de um apresentador, mas à frente do plano de fundo real da cena (BBC RESEARCH, 2007).

Algumas experiências de RM também utilizam métodos de estimativas de profundidade de objetos reais e compõem imagens respeitando a oclusão mútua, baseadas nesses valores (KIYOKAWA *et al.*, 2001).

A oclusão mútua está diretamente relacionada ao problema do registro e pode evitar incoerências na imagem exibida. Os detalhes dessa relação serão discutidos no Capítulo 5.

Algumas soluções que realizam combinações de imagens com consideração de profundidade são mostradas nas Subseções seguintes.

4.1.1 *Display* ELMO

Displays convencionais do tipo *optical see-through* normalmente não podem representar oclusão mútua entre objetos reais e virtuais. Esse tipo de tecnologia constrói objetos sintéticos translúcidos, flutuando sobre a cena real e causando um efeito denominado "conflito de oclusão" (KIYOKAWA *et al.*, 2001).

O *Display* ELMO (Figura 4.2), uma forma abreviada do termo "*enhanced optical see-through display using an LCD for mutual occlusion*", possui ambas as características necessárias para solucionar esse problema: um seletivo mecanismo bloqueador de luzes e um mecanismo de percepção de profundidade em tempo real.

(a) Mistura Normal **(b)** Bloqueio da Imagem Real **(c)** Consideração de Profundidade **(d)** Resultado Final

Figura 4.2: Funcionamento do *Display* Elmo (KIYOKAWA *et al.*, 2001)

Por meio da comparação da profundidade medida com os valores de profundidade da cena virtual, armazenada em um *buffer*, a oclusão correta pode ser exibida, em tempo real, inclusive em ambientes não controlados e

dinâmicos. Os resultados são aceitáveis, embora o recorte do elemento real seja grosseiro e visível na imagem final.

4.1.2 **Zcam**

Em programas de televisão, muitas vezes é desejável a construção de objetos virtuais atrás de um apresentador, mas à frente do plano de fundo real da cena. Em determinadas situações, existe a impossibilidade da utilização de um plano de fundo de cor constante, tornando-se inviável empregar-se técnicas de *chromakey* para solucionar o problema.

A solução desenvolvida pela empresa 3DV Systems para composições de imagens naturais realiza a extração do objeto de primeiro plano, a partir de um plano de fundo arbitrário, utilizando a distância relativa à câmera de vídeo. Trata-se da câmera de vídeo com estimativa de profundidade Zcam (Gvili *et al.*, 2003).

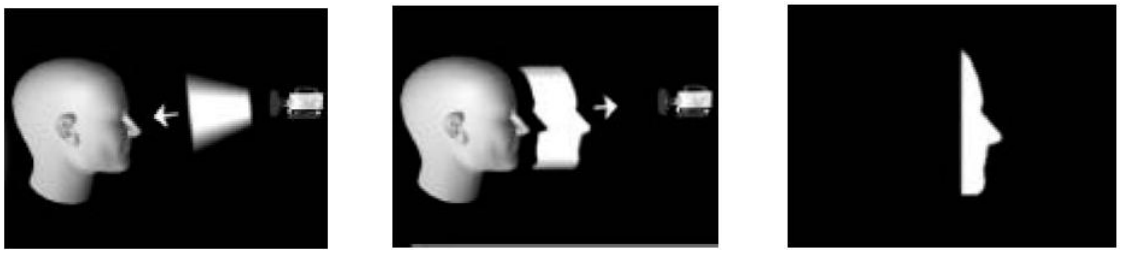
A câmera produz sinais do tipo RGBD, em que D se encarrega do armazenamento da distância de cada ponto da imagem. O sinal RGBD possibilita a criação de composições de múltiplas camadas de imagens (Figura 4.3).

Figura 4.3: ZCam em imagens naturais (Gvili *et al.*, 2003)

A Zcam, que é compatível com a maioria dos formatos, pode simultaneamente capturar informações de vídeo anexas às informações de profundidade de cada ponto da imagem em tempo real.

De forma simplificada, o processo consiste em produzir um "muro de luz" que se move de acordo com o campo de visão. Esse sinal é gerado por um pulso de curta duração, que atinge a mesma área do campo de visão em um limite de dez metros.

A luz emitida não é visível, para que não interfira na imagem da câmera. Quando refletido, esse sinal retorna à câmera, após passar por um processo de simplificação — o que consiste no bloqueio de algumas luzes de retorno para obter maior precisão nas informações de posicionamento em relação à câmera (Gvili *et al.*, 2003). Os passos do processo são mostrados na Figura 4.4.



(a) Incidência do Muro do Luz

(b) Retorno do sinal à Câmera

(c) Simplificação do Sinal

Figura 4.4: Funcionamento do Sistema Zcam (Gvili *et al.*, 2003)

4.1.3 FlashKey

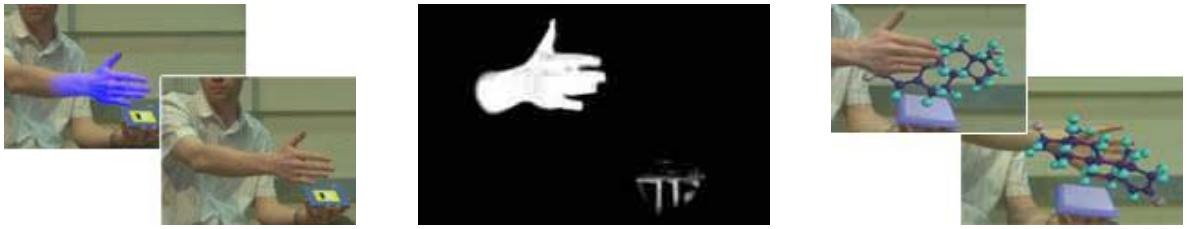
Um método denominado FlashKey (BBC Research, 2007), desenvolvido por Thomas e Russel (2005) para produções de televisão da BBC

(*British Broadcasting Corporation*), de Londres, também pode exibir objetos virtuais com estimativas de profundidade.

A pessoa, ou objeto real, que se deseja posicionar em frente ao objeto virtual, é iluminado por uma luz azul intermitente, e a câmera de vídeo reconhece a ação da luz em determinadas áreas da imagem. Para produzir essas luzes, utilizam-se LEDs acionados por um componente especialmente desenvolvido para o controle de sinais do vídeo.

Processa-se, então, a imagem da câmera e gera-se um sinal nas áreas do azul brilhante, por meio de um programa de tempo real executando em um computador pessoal convencional. Simultaneamente, executa-se um procedimento para eliminar a luz azul da imagem, utilizando-se um filtro com características de cor e de tempo.

Na Figura 4.5, são mostradas as etapas do funcionamento do FlashKey.



(a) Imagens sucessivas Captadas pela Câmera

(b) Detecção da Luz Azul

(c) Resultado Final

Figura 4.5: Fases do processo FlashKey (BBC Research, 2007)

4.1.4 Sistema 3DK

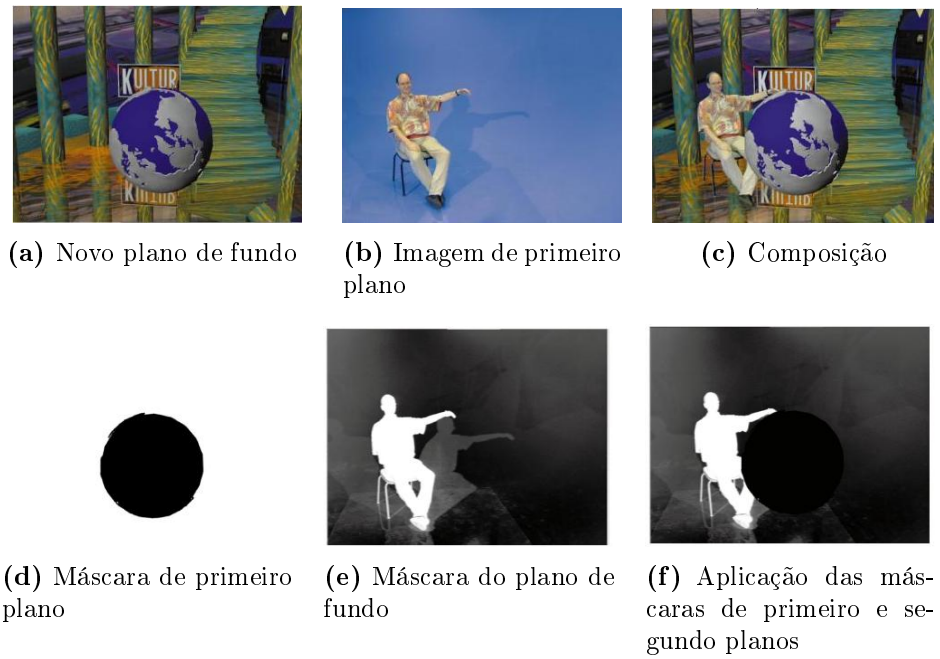
O 3DK é um típico Sistema de Estúdio Virtual, descrito na Subseção 3.3.2. Desenvolvido no *German National Research Center for Information Technology*, consiste em um conjunto de câmeras equipadas com sistemas ras-

treadores, que fornecem informações a respeito da movimentação da câmera (Gibbs *et al.*, 1998).

As câmeras produzem sinais de vídeo relativos ao elemento de primeiro plano, enquanto um sistema gerador de imagens, normalmente um supercomputador gráfico, produz sinais que correspondem ao plano de fundo, e, opcionalmente, sinais de máscaras, que podem ser utilizados para realizar a oclusão mútua.

As imagens de primeiro e segundo planos são combinadas, normalmente por um *chromakeyer*, e a composição resultante fica disponível a outros componentes, que utilizam as máscaras geradas para produzirem outros efeitos, inclusive a consideração da profundidade para compor a cena.

Cada sinal pode também ser utilizado como *feedback* para o alinhamento da câmera, auxiliando o operador a posicioná-la corretamente. Na Figura 4.6, exibe-se o funcionamento do 3DK.



(a) Novo plano de fundo

(b) Imagem de primeiro plano

(c) Composição

(d) Máscara de primeiro plano

(e) Máscara do plano de fundo

(f) Aplicação das máscaras de primeiro e segundo planos

Figura 4.6: Sinais de vídeo utilizados no Sistema de Estúdio Virtual 3DK (Gibbs *et al.*, 1998)

4.2 Remoção de superfícies ocultas de objetos virtuais

O objetivo básico no processo de geração de modelos virtuais, tanto em cenas aumentadas quanto em ambientes totalmente sintéticos, é fazer com que não só os objetos se assemelhem o máximo possível com objetos reais, mas a cena toda seja entendida como uma cena real.

Emprega-se o termo "realismo virtual" para definir as técnicas de tratamento computacional aplicadas aos objetos virtuais gerados, buscando criar uma imagem sintética o mais semelhante possível com a realidade que se obteria, caso eles fossem construídos e filmados (Azevedo e Conci, 2003).

Uma das fases do processo de realismo virtual é o tratamento de partes escondidas dos objetos virtuais. Para identificar quais as partes dos objetos estão visíveis no momento, é preciso considerar a posição relativa entre eles e o observador a cada mudança no ponto de observação, ou na forma de projeção.

Valores relacionados à profundidade de cada ponto são necessários para que se possa renderizar as superfícies visíveis do objeto e remover as suas partes ocultas por ele próprio, ou por outros objetos da cena.

As eliminações de faces inteiras são feitas normalmente utilizando coordenadas tridimensionais do próprio objeto, ao passo que a maioria das eliminações de partes de objetos se faz por meio das coordenadas do dispositivo de visualização. Portanto, em uma fase posterior (Azevedo e Conci, 2003).

Técnicas desse tipo fazem-se necessárias para níveis básicos de realismo. Para obtenção de níveis mais elevados, são necessárias técnicas mais sofisticadas, que envolvem iluminação da cena, efeitos de sombras, reflexos e

transparência de objetos.

Desenvolveram-se vários algoritmos para remoção de partes ocultas ao observador (HSR - *Hidden Surface Removal*), na tentativa de se resolver o problema de visualização entre objetos virtuais. Considerando-se esses algoritmos, observa-se que existem soluções mais simples demandando uma grande quantidade de memória, outras que utilizam menos memória, mas com tempo elevado de execução e soluções simples e rápidas, mas de pouca aplicação prática.

Vários algoritmos HSR são discutidos por Azevedo e Conci (2003), entre os quais merecem destaque o Algoritmo de Visibilidade por Prioridade, o Algoritmo de Eliminação de Faces Ocultas pela Normal, o *Ray Tracing* e o *Z-Buffer*.

4.2.1 **Algoritmo de visibilidade por prioridade**

Voltado para fins didáticos e também chamado de "Algoritmo do Pintor", baseia-se na idéia de que técnica o pintor aplicaria para tratar a visibilidade de uma tela pintada a óleo. Os detalhes mais próximos a ele são pintados cobrindo-se objetos mais afastados (Azevedo e Conci, 2003).

Se um objeto "A" bloqueia a visão de um objeto "B", e ambos os objetos se encontram na mesma linha de visão do observador, então o objeto "B" está mais distante do observador que o objeto "A". É possível criar um algoritmo que calcule a distância dos objetos em relação ao observador, e que dê prioridade à visualização dos objetos mais próximos (Azevedo e Conci, 2003).

De forma simplificada, o algoritmo calcula a distância de todas as faces poligonais da cena em relação ao observador, ordena os polígonos, resolve

problemas de distâncias iguais e desenha os polígonos iniciando pelo mais distante.

Como a visibilidade por prioridade se baseia na suposição de que uma superfície qualquer sempre domina o seu plano de visibilidade (se "A" bloqueia "B", então "B" não pode bloquear "A"), podem ocorrer algumas incorreções quando formas poligonais com contornos não usuais ou polígonos não-convexos[1] são analisadas.

4.2.2 Algoritmo de eliminação de faces ocultas pela normal

Este algoritmo utiliza o ângulo que a sua normal[2] faz com a direção de observação para determinar sua visibilidade.

Não se pode considerá-la uma técnica suficientemente completa, por que sua aplicação resume-se a poliedros convexos fechados. No entanto, sua utilização como uma espécie de "pré-filtro" pode trazer ganho considerável em processos de criação de cenas em tempo real.

Independentemente da distância do observador, é possível visualizar apenas a face do plano da frente de qualquer superfície plana. Rotacionando a superfície de modo a visualizar a outra face, em determinado momento, esta superfície será visível apenas como um segmento de reta (Azevedo e Conci, 2003).

Nessa posição, sua normal estará perpendicular (90º) à direção dos olhos do observador. Pode-se deduzir que uma superfície apenas se faz visível se o ângulo de sua normal com o observador estiver entre -90º e +90º.

[1] Se um segmento de reta unindo pontos de um polígono estiver sempre no seu interior, o polígono será convexo, caso contrário será não-convexo (Azevedo e Conci, 2003).

[2] Vetor perpendicular a uma determinada superfície (Azevedo e Conci, 2003).

Partindo desse princípio, é possível identificar as faces ocultas de objetos.

4.2.3 *Ray Tracing*

Uma técnica também utilizada como algoritmo de determinação de visibilidade de elementos em cena é o *Ray Tracing*, embora sua maior aplicabilidade na Computação Gráfica implique efeitos de sombras, refração, reflexão e texturas (Azevedo e Conci, 2003).

O *Ray Tracing* tem por princípio simular a geometria óptica envolvida no trajeto de alguns raios de luz que viajam pelo espaço da cena. Por motivos computacionais, o modelo utilizado é o contrário do que realmente acontece na visualização de uma cena. Nessa técnica, supõe-se que um raio originário de nossos olhos chegue até o objeto que se quer renderizar. Pelas leis da óptica, essa reversão não provoca alteração alguma na geometria envolvida.

O objetivo da técnica é simular a propagação da luz no ambiente, avaliando a sua interação com os objetos que o compõem e considerando a interação da luz com as suas superfícies.

4.2.4 Algoritmo Z-Buffer

A técnica mais utilizada para a remoção de partes ocultas é a do *Z-Buffer*. O barateamento da tecnologia tem possibilitado a armazenagem de grandes quantidades de informação a baixo custo, além de impulsionar a utilização de métodos baseados no uso intensivo de memória.

Os algoritmos utilizados para renderização em tempo real são embutidos ou em alguma estrutura gráfica como bibliotecas 3D (OpenGL), ou em

hardwares específicos. A organização dessas estruturas, apesar de algumas propostas de melhorias (JOHNSON *et al.*, 2005), estranhamente tem sofrido poucas alterações nas últimas décadas, e o *z-Buffer* tem prevalecido.

Também conhecido por *Z-Buffering*, possui uma implementação simples para determinar visibilidade de superfícies. No entanto, apresenta alto custo computacional. Embora possa ser implementado utilizando-se uma única matriz ou *buffer*, usualmente até duas matrizes com iguais dimensões da tela de apresentação são alocadas.

A utilização dessas matrizes é feita da seguinte forma: a primeira matriz, também chamada de *Buffer* de Imagem, é opcional e funciona como uma espécie de rascunho, durante os cálculos de visibilidade dos objetos. A segunda matriz, que dá nome ao método, denomina-se *Buffer* de Profundidade, ou *Z-Buffer*, e armazena a distância de cada ponto da tela, da imagem de rascunho ao plano de projeção.

Z-Buffers são normalmente implementados em *hardware* com inteiros de 16 a 32 bits, e com pontos flutuantes, quando implementados em *software* para minimizar os problemas dos recortes das bordas de objetos.

A inicialização do *Buffer* de Profundidade é feita com os valores mais altos possíveis antes do cálculo e da projeção dos objetos. Isso significa que esses objetos estão colocados no plano de projeção em uma distância máxima. O *Buffer* de Imagem é inicializado com os valores das cores de fundo da imagem.

Para cada novo ponto dos objetos projetados na tela de rascunho, realiza-se o cálculo de distância em relação ao plano de projeção. Os valores de cor do ponto substituem os valores armazenados na tela de rascunho naquela coordenada, caso a distância seja inferior à distância armazenada no *z-Buffer*. Caso contrário, o ponto é considerado bloqueado por outro já calculado, que está mais próximo do plano de projeção (Figura 4.7).

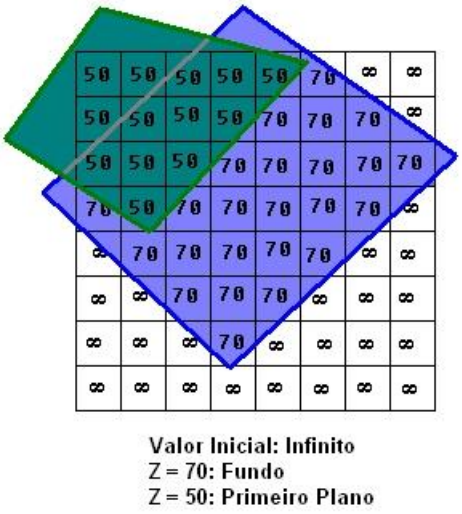


Figura 4.7: Representação gráfica do *Z-Buffer* (Azevedo e Conci, 2003)

O algoritmo baseia-se nas coordenadas dos pontos da tela, e as coordenadas relacionadas à profundidade (valor do eixo z), são armazenadas em um *array*. Os pontos colocados na tela, no mesmo local, são comparados por meio dos valores desse *array* para verificação da distância com o observador.

O *Z-Buffer* é de fácil implementação, mas ocasiona uma freqüente alteração no valor dos pontos da tela, obrigando um novo cálculo da cor. É difícil contornar os problemas de "serrilhamento" pelo fato de apenas um valor de intensidade e profundidade ser armazenado e, como já foi enfatizado, a utilização de memória é intensa.

4.3 O OpenGL *framebuffer*

O OpenGL é uma biblioteca de *software*, desenvolvida para *hardware*

gráficos, constituída de centenas de procedimentos e funções para manipular objetos e operações sobre esses objetos em imagens tridimensionais de alta qualidade (Segal e Akeley, 2007).

Bibliotecas de *software* gráficos, como o ARToolkit, utilizam as funcionalidades do OpenGL para construir e movimentar os objetos virtuais em cena.

À medida que se desenham formas geométricas, textos ou imagens na tela, cálculos de rotação, translação, escala, iluminação e projeção de perspectiva são processados. Deve-se compreender bem o modo como os *pixels* da janela são afetados para que se possam determinar as cores com que esses *pixels* devam ser desenhados (Woo *et al.*, 1997).

A construção de objetos gráficos: linhas, pontos, polígonos e figuras do tipo *bitmap*, muitas vezes necessitam de alguns efeitos, como suavização de bordas e mapeamento de texturas. Algumas dessas tarefas são custosas em termos de processamento, e sua execução, via OpenGL, é dependente da existência de *buffers* (Segal e Akeley, 2007).

O *framebuffer* consiste de um conjunto de pontos organizados em uma matriz bidimensional, de modo que cada um de seus pontos pode ser entendido como um conjunto de *bits*. O número de *bits* por *pixel* é variável entre as implementações da API (Segal e Akeley, 2007).

Bits correspondentes a cada *pixel*, no *framebuffer*, são agrupados em um *bitplane*. Cada *bitplane* contém um único *bit* para cada *pixel*. Esses *bitplanes* agrupam-se em alguns *buffers* lógicos.

Com algumas variações entre suas implementações, esses *buffers* são responsáveis pela cor (*Color Buffer*), pelo armazenamento de partes da imagem (*Stencil Buffer*), pelo armazenamento de valores relacionados à profundidade (*Depth Buffer* ou *z-Buffer*) e de informações de várias imagens (*Accumulation Buffer*) (Woo *et al.*, 1997).

O *Color Buffer* consiste de um conjunto de *buffers* (*front left*, *front right*, *back left*, *back right* e alguns *buffers* auxiliares) que contém informações no formato RGB, podendo suportar outro formato e conter valores alfa. Normalmente, visualiza-se o conteúdo do *front left* e do *front right*, ao passo que os demais trabalham em segundo plano.

Implementações OpenGL que suportam visão estereoscópica utilizam o *front left* e o *front right buffer*. Sistemas monoscópicos utilizam apenas o *front lefth buffer* para visualização (Segal e Akeley, 2007).

Os valores de profundidade de cada ponto da tela são armazenados na implementação OpenGL do algoritmo do *z-Buffer* (*Depth Buffer*). Mede-se normalmente essa profundidade em termos de distância dos olhos, fazendo que os pontos com menor profundidade sobrescrevam os pontos com valores maiores, como exibido na Subseção 4.2.4.

Em aplicações onde partes do conteúdo da imagem devem permanecer fixas, o *Stencil Buffer* pode ser utilizado. Exemplos típicos da facilidade desse recurso são os simuladores, em que os instrumentos internos do automóvel (ou avião) podem ser carregados uma única vez e apenas os objetos externos em movimento são processados.

O *Accumulation Buffer* contém somente dados de cores no formato RGBA e pode ser utilizado quando o *buffer* de cor funciona nesse modo. Esse *buffer* é normalmente utilizado para armazenar um conjunto de imagens utilizadas para composições digitais e permite, inclusive, operações de suavização de bordas por meio de cálculos da média de amostras retiradas da imagem, com a finalidade de produzir valores finais da cor do *pixel* que proporcionem melhor qualidade no resultado (Woo *et al.*, 1997).

4.3.1 Operações de fragmentos

O fragmento produzido pelo processo de rasterização[3] modifica o ponto com localização correspondente no *framebuffer* baseado em alguns parâmetros e condições, conforme se demonstra no diagrama da Figura 4.8. Esses testes e modificações são descritos de acordo com sua ordem de execução.

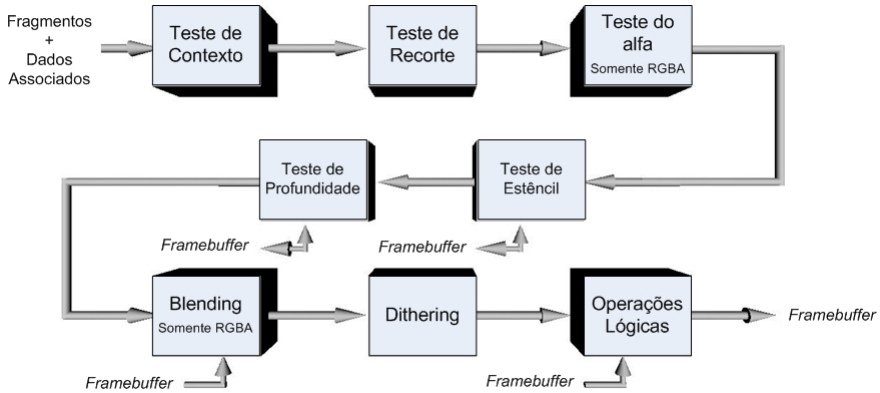


Figura 4.8: Diagrama OpenGL das operações de fragmentos (Segal e Akeley, 2007)

O primeiro teste realizado é o de contexto (*Ownership Test*), que serve para determinar se o *pixel* da posição está, no momento, fazendo parte do contexto da biblioteca gráfica. Caso não esteja, o sistema de janelas decide o que fazer com o fragmento, podendo, inclusive, descartá-lo.

O teste de recorte (*Scissor Test*) determina se o fragmento de entrada, correspondente a um *pixel*, está localizado dentro de uma determinada porção retangular definida para restringir a área do desenho.

O teste do alfa (*Alpha Test*) é aplicado apenas para o modo RGBA e permite aceitar ou rejeitar um fragmento baseado no seu valor alfa. Se

[3]Processo de conversão da representação vetorial para a representação matricial da imagem tornando-a possível de ser armazenada na memória de dispositivos matriciais (Azevedo e Conci, 2003)

habilitado, o teste compara o valor alfa do fragmento de entrada com um determinado valor de referência. Sua utilização está mais relacionada a algoritmos de transparência e texturas mapeadas, transparentes em determinadas partes.

O teste de estêncil (*Stencil Test*) descarta um fragmento baseado no resultado da comparação entre o valor no *Stencil Buffer* na localização do fragmento de entrada e um valor de referência. O valor armazenado é modificado dependendo do resultado desse teste, e o fragmento é sempre aceito quando não existe um *buffer* desse tipo.

A fase seguinte é o teste responsável pela resolução do problema da profundidade. O OpenGL oferece diversos tipos de comparações para o teste de profundidade (*Depth Test*). O parâmetro padrão permite que o fragmento de entrada passe no teste, caso o valor de profundidade seja menor que o armazenado. Isso significa que o objeto correspondente está mais próximo de ponto de visão.

Uma vez realizados todos esses testes, e, caso o fragmento de entrada ter passado por todos eles, pode-se, então, combiná-lo com os valores contidos no *buffer* de cor. O modo mais simples de realizar esta operação é sobrescrever os valores existentes. Para produzir efeitos de transparência e suavização de bordas utilizando-se o modo RGBA, pode-se calcular a média dos valores contidos no *buffer* e do fragmento de entrada (*Blending*) para encontrar o valor do *pixel*.

A modificação dos valores de cores para produzir outras tonalidades também é possível em sistemas com baixo número de cores disponíveis. Esse processo, conhecido como *Dithering*, ocasiona perda na resolução da imagem.

A operação final sobre o fragmento é a operação lógica (*Logical Operation*), que pode ser aplicada nos valores do fragmento de entrada, nos valores do *buffer* de cor, ou em ambos.

Essas operações de fragmentos são especialmente úteis em máquinas em que as operações gráficas primárias copiam retângulos de dados de um lugar da janela para outro, da janela para a memória do processador, ou da memória para a janela.

As transferências de dados não são feitas diretamente na memória, o que torna possível executar uma operação lógica entre os dados de entrada e os dados atuais, para, posteriormente, sobrescrever os dados existentes com o resultado da operação.

4.3.2 Operações no *framebuffer*

As operações descritas na subseção 4.3.1 ocorrem nos fragmentos individuais que estão sendo enviados ao *framebuffer*. Existem outras operações que manipulam ou afetam o *framebuffer* como um todo.

Para selecionar um *buffer* para escrita, a primeira operação é controlar os *buffers* do *Color Buffer* em que cada uma das cores do fragmento é escrita. Os *buffers* são selecionados por meio de constantes passadas como argumentos de funções. Além de operações de escrita e leitura, blocos de *pixels* podem ser copiados de uma região para outra do *framebuffer*.

Existem algumas operações que permitem um controle mais refinado de atualizações no *buffer*, utilizadas para mascarar a escrita dos *bits* em cada um dos *framebuffers* lógicos, depois que todas as operações de fragmentos são executadas.

Operações de limpeza são normalmente as mais custosas para o desempenho de um programa gráfico. Quando existe a necessidade de limpeza de outros *buffers* além do *buffer* de cor, o tempo necessário para sua execução é ainda maior (Woo *et al.*, 1997).

Algumas arquiteturas são equipadas com *hardwares* capazes de limpar mais de um *buffer* ao mesmo tempo. Podem -se especificar valores para serem escritos nos *buffers* em que se deseja aplicar o processo de limpeza. Caso o *hardware* suporte operações simultâneas, todo processo de limpeza ocorre de uma única vez; caso contrário, são executados em cada *buffer* seqüencialmente (WOO *et al.*, 1997).

As operações no *Accumulation Buffer* não escrevem diretamente nele. Normalmente geram-se várias imagens em um dos *buffers* de cores padrões e, então, são acumuladas, uma por vez. Finalizada a acumulação, o resultado é copiado no *buffer* de cor para visualização.

4.4 Considerações Finais

Discutiu-se neste Capítulo, a importância da consideração da profundidade dos elementos para construção da cena em ambientes misturados.

Problemas que envolvem estimativas de profundidade foram analisados e observou-se a necessidade de os elementos reais e virtuais obstruírem-se mutuamente no decorrer da aplicação, de acordo com sua localização na cena.

Algumas soluções, baseadas em *hardware*, que permitem a construção de ambientes misturados, com consideração de profundidade foram apresentadas, e foram discutidos os detalhes de seu funcionamento.

Analisaram-se os algoritmos mais conhecidos, que resolvem o problema das superfícies de objetos virtuais que devem estar visíveis, baseado em sua distância em relação ao observador. Entre eles, destacou-se o algoritmo do *Z-Buffer*.

O conjunto de *Buffers* do OpenGL (*framebuffer*) foi detalhado, bem

como o funcionamento de cada *Buffer* que o compõe, enfatizando-se as funções por meio das quais são manipulados.

Capítulo 5

A UTILIZAÇÃO DA TÉCNICA DE CHROMAKEY PARA COMPOSIÇÃO DE CENAS EM AMBIENTES DE REALIDADE MISTURADA

Como descrito na Seção 2.4, as maiores dificuldades encontradas no desenvolvimento de sistemas de RM estão freqüentemente associadas a problemas de registro. Um alinhamento correto dos objetos virtuais em relação ao ambiente real é fundamental para que se obtenha ilusão de coexistência entre eles.

Neste contexto, são fatores importantes a serem considerados a noção de profundidade entre elementos reais e virtuais e métodos que permitam representá-los corretamente no ambiente misturado, ponderando-se sua distância em relação ao observador (câmera).

Na Seção 4.1, demonstraram-se algumas tecnologias que permitem realizar composições, em tempo real, cujos elementos reais podem ser exibidos tanto em primeiro plano quanto em plano de fundo (oclusão mútua). Esse tipo de recurso pode minimizar problemas de registro, em aplicações de RM, e permitir uma visualização coerente da cena.

No entanto, as soluções encontradas — baseadas em *hardware* e utilizadas principalmente na indústria televisiva — ainda são raras e custosas, inviabilizando seu uso na maior parte das aplicações de RM. Nesses casos, uma solução em *software* seria mais adequada.

5.1 Definições do projeto

5.1.1 Motivação

Diversas bibliotecas de apoio ao desenvolvimento de sistemas de RM — entre elas o ARToolkit — fornecem funções que resolvem satisfatoriamente o problema da estimativa do posicionamento e da orientação em que o objeto virtual deve ser gerado dentro do ambiente misturado. No entanto, tais objetos são exibidos apenas em primeiro plano, durante todo o tempo de execução do programa.

Esta limitação provoca uma incoerente relação de profundidade nas situações em que o elemento real está mais próximo do observador que o objeto virtual. Nesse caso, este elemento real deveria ser exibido em primeiro plano.

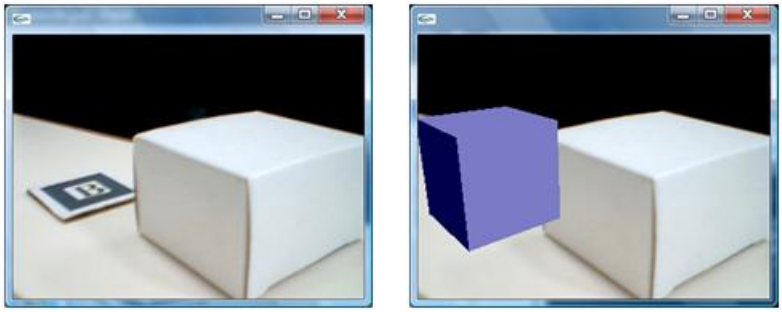
Isso ocorre devido ao projeto do ARToolkit e de outras ferramentas baseadas em reconhecimento de marcadores, "desprezarem" o posicionamento de elementos reais e não possibilitarem, em nenhuma circunstância, que essa sobreposição ocorra.

Os objetos virtuais são gerados sobre marcadores passivos, e a obstrução da visibilidade dessas marcas — mesmo que parcial — por algo real

posicionado à sua frente, impede seu reconhecimento e, conseqüentemente, a renderização do elemento virtual.

Não obstante, existem determinadas situações nas quais os marcadores estão visíveis à câmera, mesmo que estes estejam posicionados mais distantes do observador que um elemento real. E, no momento da geração do objeto virtual, a cena é montada de forma incoerente.

Uma das referidas situações pode ser observada na Figura 5.1. O marcador, a que o objeto virtual está vinculado, foi posicionado mais distante do observador (câmera) do que o objeto real (Figura 5.1a). No entanto, o objeto virtual renderizado sobre esse marcador (caixa azul) é visualizado à frente do elemento real (caixa branca), obstruindo parte desse elemento e provocando um erro de registro (Figura 5.1b).



(a) Marcador real posicionado ao fundo

(b) Objeto virtual gerado em primeiro plano

Figura 5.1: Erro de registro

Analisando o exemplo citado, pode se observar a relação direta da oclusão mútua com o problema do registro. Torna-se evidente que o registro vai além da estimativa do posicionamento e da orientação do objeto virtual em relação ao ambiente real. Existe também a necessidade de que a cena seja visualizada de forma coerente, respeitando-se a profundidade dos elementos, reais e virtuais, que a compõe.

O ARToolkit não possui um método de recorte de silhueta de objetos

reais (elementos de interesse) dispostos em cena, como os discutidos na Seção 3.2. Tal método, de difícil implementação nesta situação, faz-se necessário para que a oclusão mútua possa se realizar.

A importância da extração do elemento real também pode ser observada, na medida em que algumas experiências com o ARToolkit permitam a utilização de um conjunto de câmeras (Dias Junior, 2005), ou redundância de marcadores (Kato e Billinghurst, 1999), minimizando o problema da oclusão dos mesmos, descrito na Subseção 2.5.5.4.

5.1.2 Objetivos e estrutura

O propósito desta pesquisa foi o estudo e a implementação de um componente para a biblioteca ARToolkit que permita a construção de cenas tridimensionais, cujos elementos do ambiente misturado possam ser exibidos em níveis coerentes de profundidade.

Sua utilização permite o desenvolvimento de aplicações em que elementos reais obstruam elementos virtuais, minimizando problemas de registro e estendendo as funcionalidades do ARToolkit a um número maior de aplicações.

A estrutura do projeto, mostrada na Figura 5.2, permite a identificação de dois módulos destacados em tom mais claro de cinza, implementados e utilizados em conjunto com as bibliotecas OpenGL, glut (biblioteca de utilidades para programas OpenGL) e ARToolkit.

O módulo "Chromakey" consiste de um conjunto de funções responsável pela extração do elemento de interesse, inclusão do novo plano de fundo, identificação da profundidade dos objetos (reais e virtuais), entre outras funcionalidades necessárias à composição da cena. "Aplicações" é o módulo no

qual se desenvolveram os protótipos utilizados nos testes.

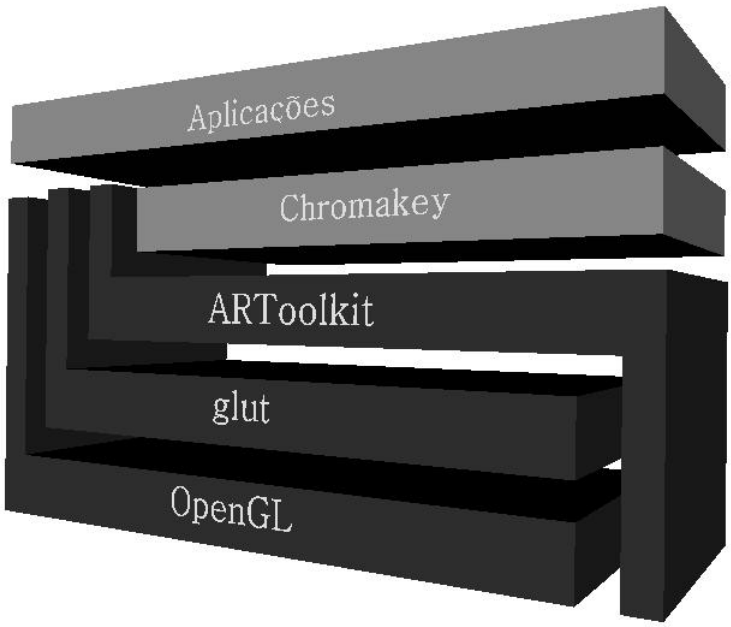


Figura 5.2: Estrutura do projeto

5.1.3 Metodologia

Para a elaboração da metodologia utilizada no desenvolvimento do projeto, é necessário o entendimento da forma com que o ARTookit gerencia e exibe a imagem recebida do dispositivo de captura e os objetos virtuais, construídos por meio de alguma biblioteca gráfica.

A API OpenGL, responsável pelo gerenciamento da janela na qual a cena é visualizada, utiliza um conjunto de *buffers* (*framebuffer*) para construir e exibir a imagem, como descrito na Seção 4.3. O ARToolkit, que é baseado no OpenGL, transfere diretamente ao *Color Buffer* a imagem capturada pela câmera, como mostrado na Figura 5.3.

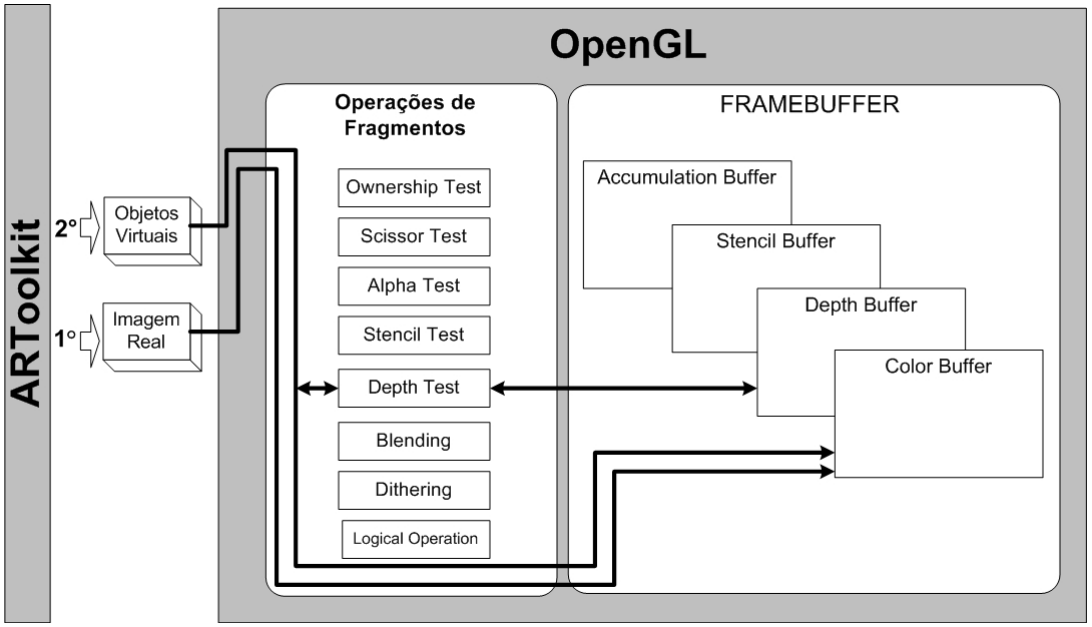


Figura 5.3: Formação da imagem no ARToolkit

Em relação aos objetos virtuais, gerados a partir de coordenadas tridimensionais, pode-se utilizar suas coordenadas "z" para controlar quais partes estarão visíveis na cena. Para isso, o *Depth Test* (utilização do *buffer* de profundidade do *framebuffer*) deve ser ativado (Figura 5.3).

Por meio do algoritmo de eliminação de superfícies ocultas *Z-Buffer*, descrito na Subseção 4.2.4 e implementado no OpenGL, os objetos virtuais são organizados, em relação às suas respectivas coordenadas "z", pelo *Depth Buffer* e, em seguida, copiados para o *Color Buffer* sobre a imagem real anteriormente carregada. Montada no *Color Buffer*, a imagem resultante é, então, exibida pelo dispositivo de visualização.

Desta forma, mesmo com a movimentação dos marcadores (e, por conseqüência, dos objetos virtuais), faz-se o cálculo da profundidade a cada *frame*, exibindo-os corretamente. No entanto, a imagem real, capturada pela câmera e imediatamente transferida ao *Color Buffer*, permanece sempre ao fundo.

Como discutido na Seção 3.2, vários são os métodos de recorte de regiões de interesse, tanto em ambientes controlados quanto em ambientes arbitrários. E, a maioria desses métodos aplicados a imagens arbitrárias não são suficientemente rápidos para realizar essa tarefa em tempo real.

Entre as técnicas de recorte aplicáveis em ambientes controlados, o *chromakey* permite o isolamento de objetos de primeiro plano colocados sobre fundo de cor constante, em tempo real, desde que tal objeto não contenha partes na mesma cor desse plano de fundo.

Considerando-se a possibilidade da utilização de planos de fundo constantes, propõe-se um ambiente formado por um elemento real, dois ou mais marcadores (um deles fixo ao elemento real) e um plano de fundo de cor única (azul ou verde), como exibido na Figura 5.4.

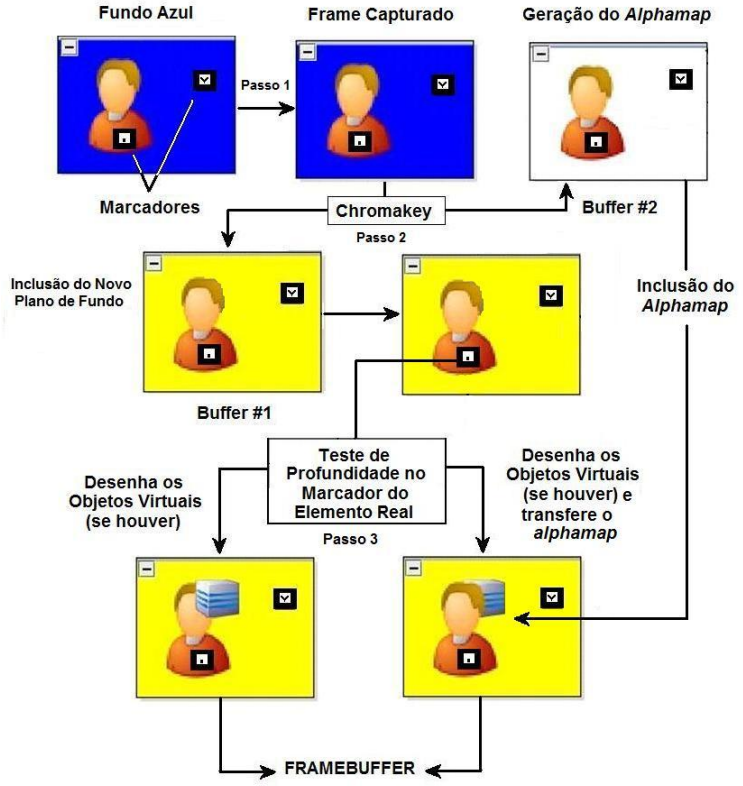


Figura 5.4: Abordagem proposta

As funções de captura de vídeo da biblioteca ARToolkit, o que é demonstrado no passo 1 da Figura 5.4, armazenam cada *frame* obtido em sua representação no *framebuffer* OpenGL, incluindo o valor do componente alfa, discutido na Subseção 3.4.2.

O *alphamap* (mapa de transparência) do elemento real (elemento de interesse) pode ser extraído por meio da técnica do *chromakey* e armazenado em um *buffer* (*Buffer#1*). O novo plano de fundo pode ser construído substituindo-se os *pixels* azuis da imagem original por uma imagem estática, por uma seqüência de *frames* (filme), ou por modelos virtuais, como mostrado no passo 2, da Figura 5.4.

Resultantes desse processo, obtêm-se a imagem do elemento de interesse à frente do novo plano de fundo e o *alphamap* armazenados no *Buffer#1* e no *Buffer#2* respectivamente.

Para a obtenção das coordenadas dos elementos reais — necessárias à composição da cena — a solução utilizada neste trabalho foi a fixação de um marcador no elemento real, que é considerado o elemento de interesse. Desse modo, sua profundidade (coordenada "z") pode ser obtida pelas funções de rastreamento do ARToolkit e utilizada para construção da cena no *framebuffer* do OpenGL antes da sua exibição.

No passo 3, os marcadores são rastreados e, se não existirem objetos virtuais cuja coordenada "z" possua valor maior (mais distante) que a coordenada "z" do marcador fixado no elemento real, a imagem armazenada é transferida diretamente ao *Color Buffer*.

Caso existam objetos virtuais posicionados mais distantes do observador que o elemento real, estes objetos devem ser gerados. O *alphamap* (elemento de interesse), armazenado no *Buffer#2*, é então copiado sobre a imagem, de modo que o elemento real sobreponha os objetos virtuais da cena.

Finalmente, os objetos mais próximos do observador (caso existam)

são renderizados, e a imagem resultante deve, então, transferir-se ao *Color Buffer* para ser exibida.

5.1.4 Limitações impostas pelo *chromakey*

Como solução para o problema do registro, a proposta da utilização do *chromakey* impõe algumas restrições às aplicações desenvolvidas. A mais importante delas relaciona-se à exigência de um plano de fundo de cor única.

Grande parte das aplicações baseadas no ARTookit é executada em ambientes internos e não muito amplos devido a dificuldades de reconhecimento de marcadores distantes da câmera. Estúdios virtuais, ambientes colaborativos de RM, sistemas de ensino com RM são exemplos de aplicações nas quais planos de fundo de cor única são aceitáveis.

O fato de o fundo original ser substituído por uma imagem ou um vídeo não possibilita, obviamente, a interação do usuário com elementos do plano de fundo. No entanto, torna possível, em espaços fechados, a construção de ambientes misturados formados por imagens de cenas externas.

O *chromakey* necessita de um ambiente com iluminação controlada para que se mantenha a uniformidade da cor do plano. Entretanto, esta também é uma exigência das aplicações de RM baseadas no ARToolkit, para que o reconhecimento dos marcadores não seja prejudicado.

Outra restrição imposta pela utilização do *chromakey* é que os elementos reais não devem conter partes com a mesma cor do plano de fundo. Por outro lado, a utilização da técnica possibilita a invisibilidade dos marcadores na cena, como será demonstrado na Subseção 5.2.4. Isso não seria possível em aplicações convencionais do ARToolkit, em que o plano de fundo é a imagem capturada em tempo real pela câmera.

A maior vantagem da utilização do *chromakey* para construção de ambientes de RM é a possibilidade da sobreposição de elementos virtuais por elementos reais, o que possibilita a implementação da consideração da profundidade, proporcionando uma visualização coerente dos elementos da cena, além de evitar problemas de registro semelhantes ao exemplificado na Subseção 5.1.1.

Na Seção 5.2, demonstra-se detalhadamente a utilização da técnica do *chromakey* como solução para os problemas citados.

5.2 Implementação

Uma vez definidas a estrutura e a metodologia adotadas no projeto passa-se à fase seguinte: a da implementação das funções responsáveis pelas tarefas descritas na Subseção 5.1.3. Detalhes da elaboração do módulo "Chromakey", como a utilização de suas funções, a escolha do algoritmo para isolamento do fundo e os tipos de planos de fundo que podem ser utilizados, são descritos nesta Seção.

A elaboração de marcadores especiais, que não são visualizados no ambiente misturado, e a utilização das funções do módulo para composição de cenas com consideração de profundidade são também discutidos.

5.2.1 Módulo Chromakey

Dos módulos mostrados na Figura 5.2, o "Chromakey" é onde se implementa o conjunto de funções que operam como um componente para a biblioteca ARToolkit. Todas essas funções estão localizadas no arquivo

"chromakey.c", acessado por meio do arquivo de cabeçalhos "chromakey.h". A descrição das funcionalidades do módulo, as estruturas de dados e os algoritmos utilizados são apresentados nas Subseções seguintes.

5.2.1.1 **Estruturas de dados e constantes**

Muitas vezes, nas aplicações de RM, é desejável que vários objetos virtuais sejam inseridos na cena. No entanto, a quantidade de objetos processados tem influência direta no desempenho do sistema. Por esse motivo, no arquivo de cabeçalhos "chromakey.h", em que estão definidos os protótipos de funções, constantes e, tipos de dados, pode se delimitar o número máximo de objetos virtuais a serem carregados na cena ($OBJECT\_MAX$).

Também deve ser definido um tamanho limite para resolução de tela, na constante $SCREEN\_MAX$, e um tipo enumerado, *FormatType*, contendo os formatos de *pixels* suportados pelo módulo e passados como parâmetros para algumas de suas funções.

A estrutura de dados criada para o armazenamento dos *pixels* da imagem capturada pela câmera e das imagens intermediárias, montadas e utilizadas no decorrer do processo de composição, também está definida no arquivo "chromakey.h".

Cada *pixel* da imagem é composto por quatro canais de cor (BGRA), onde cada canal é do tipo *ARUint8*. O *ARUint8* é um tipo definido pelo ARToolkit que corresponde ao *unsigned char*, da Linguagem C. Na Figura 5.5, exibem-se as definições citadas.

A estrutura que armazena as informações dos objetos virtuais, preenchida pelas funções do ARtoolkit, também está definida no arquivo "chromakey.h". Nessa estrutura, cabe destacar o atributo que recebe a coordenada

```
#define   OBJECT_MAX 30
#define SCREEN_MAX 1024000
enum FormatType{_BGRA, _RGBA, _BGR, _RGB};
enum BackColor{_GREEN, _BLUE};

typedef struct{ ARUint8 pixel3[3];}pixel3;
typedef struct{ pixel3 buffer[SCREEN_MAX];} rec3;
typedef struct{ ARUint8 pixel[4]; }pixel;
typedef struct{ pixel buffer[SCREEN_MAX];} rec;
```

Figura 5.5: Estruturas de dados e constantes do arquivo "chromakey.h"

"z" (z_coord) do marcador. Esse atributo é utilizado para o cálculo da profundidade dos objetos (Figura 5.6).

```
typedef struct {
    char       name[256];
    int        id;
    int        visible;
    int        collide;
    double     marker_coord[4][2];
    double     trans[3][4];
    double     marker_width;
    double     marker_center[2];
    double     z_coord;
} crObjectData_T;
```

Figura 5.6: Estrutura com informações do objeto virtual.

5.2.1.2 **Funções**

As funções implementadas no módulo "Chromakey", cujas funcionalidades permitem a construção de cenas misturadas com consideração de profundidade, como já foi mencionado, estão localizadas no arquivo "chromakey.c". Além da manipulação de imagens, funções relacionadas ao acesso a objetos virtuais também foram definidas nesse arquivo.

A função *chLoadFile()* é responsável pela abertura e carregamento do arquivo (imagem ou vídeo). Esse arquivo contém o plano de fundo que substitui o fundo original. Inicialmente, apenas arquivos binários, previamente capturados pela câmera, são suportados. No entanto, pretende-se estender a funcionalidade desta função para suportar outros formatos de imagens e vídeos.

A função *chReadFile()* faz a leitura do arquivo indicado pela função *chLoadFile()*. Faz-se a leitura de um *frame* por vez durante a execução do *loop* principal do programa, e pode ser aplicada tanto a uma imagem estática (único *frame*) quanto a um arquivo de vídeo (seqüencia de *frames*). Caso o final do arquivo tenha sido encontrado, a leitura recomeça do início.

A troca do plano de fundo de cor única pela imagem do arquivo é feita pela função *chSwapBackgroundBlue()* ou pela função *chSwapBackground-Green()*, conforme a cor (azul ou verde) do plano de fundo. Nessas funções, também é gerado o *alphamap*, armazenado em um *buffer* para posterior utilização. Um trecho do código fonte da função *chSwapBackgroundBlue()*, utilizada para fundos azuis, é mostrado na Figura 5.7.

```
for(j=0;j<image_size;j++){
    k = (image_cam[i]<<1) - image_cam[i+2] - image_cam[i+1];
      if (k>threshold){
        alphamap->buffer[j].pixel[3] = 0;
        image_cam[i] = new_back->buffer[j].pixel[0];
        image_cam[i+1] = new_back->buffer[j].pixel[1];
        image_cam[i+2]= new_back->buffer[j].pixel[2];
      } else {
        alphamap->buffer[j].pixel[0] = image_cam[i];
        alphamap->buffer[j].pixel[1] = image_cam[i+1];
        alphamap->buffer[j].pixel[2]=  image_cam[i+2];
        alphamap->buffer[j].pixel[3] = 255;
      } i+=4;
}
```

Figura 5.7: Trecho do código fonte da função *chSwapBackgroundBlue()*

Na função *chVisibilityandDepth()* realiza-se toda rotina para verifi-

cação da visibilidade dos marcadores e obtenção da profundidade dos objetos. O teste de visibilidade pode ser implementado no protótipo; no entanto, a função citada pode simplificar esta tarefa.

Para que elementos reais possam sobrescrever objetos virtuais na cena exibida, o *alphamap*, gerado durante a execução da função *chSwapBackgroundBlue()*, deve ser colocado sobre a imagem armazenada no *framebuffer*, depois da geração e transferência dos objetos virtuais ao *Color Buffer*. Nesse momento, tais objetos já estão em coordenadas de tela (bidimensionais) e são parte do plano de fundo.

Para isso, faz-se necessário copiar a imagem em questão do *Color Buffer*, inserir o *alphamap* sobre esta imagem e transferí-la novamente ao *Color Buffer*. Todo esse processo é realizado pela chamada da função *chDrawAlphamap()*, cujo trecho do código fonte é mostrado na Figura 5.8.

```
glReadBuffer(GL_BACK_LEFT);
glReadPixels(0,0,image_width,
image_height, GL_BGRA, GL_UNSIGNED_BYTE, image_cam); i=0;
   for(j=0;j<aux;j++){
        if (alphamap->buffer[j].pixel[3] == 255){
          image_cam[i] = alphamap->buffer[j].pixel[0];
          image_cam[i+1] = alphamap->buffer[j].pixel[1];
          image_cam[i+2] = alphamap->buffer[j].pixel[2];
          image_cam[i+3] = alphamap->buffer[j].pixel[3];
        } i+=4;
   }
glClear(GL_COLOR_BUFFER_BIT); glDrawBuffer(GL_BACK_LEFT);
glDrawPixels(image_width,image_height,GL_BGRA,GL_UNSIGNED_BYTE,image_cam);
```

Figura 5.8: Trecho do código fonte da função *chDrawAlphamap()*

A função *chCaptureZ()* retorna a coordenada "z" do objeto virtual. O valor do *threshold* para *chromakey*, calculado pelo algoritmo de van den Bergh e Lalioti (1999), pode ser obtido pela chamada da função *chGetThresh()*.

A função *read_ ObjData()*, utilizada em alguns modelos de aplicações do ARTookit, foi incorporada ao módulo "chromakey.c". Esta função permite

que marcadores especificados em um arquivo de configuração do usuário possam ser vinculados aos objetos virtuais.

Todas as funções descritas nesta Subseção, a descrição de seus parâmetros e de seu tipo de retorno exibem-se na Tabela 5.1.

Tabela 5.1: Funções do módulo "Chromakey"

Função	**Tipo**	**Parâmetros**	
chLoadFile()	void	new_back	Endereço do arquivo que contém o novo plano de fundo
		file_name	Nome do arquivo a ser carregado
chReadFile()	void	new_back	Endereço do arquivo que contém a imagem de fundo
		new_back_buffer	Endereço da estrutura que receberá o *frame* do novo plano de fundo
chSwapBackgroundBlue() chSwapBackgroundGreen()	void	image_size	Tamanho da imagem
		image_cam	*Pixel* a ser analisado
		threshold	Valor limite para o algoritmo que realiza o *chromakey* identificar um *pixel* como azul ou não
		new_back	*Buffer* que contém o novo plano de fundo
		alphamap	*Buffer* no qual o *alphamap* será armazenado
		pixel_format	Formato do *pixel* (restritos aos tipos definidos em *FormatType*)
chVisibilityandDepth()	void	objectnum	Número de objetos virtuais, obtidos de um arquivo de configuração do usuário
		marker_num	Número de marcadores detectados
		markers	Informações dos marcadores
		object	Informações dos objetos virtuais
chDrawAlphamap()	void	image_width	Largura da imagem
		image_height	Altura da imagem
		image_cam	*Pixel* a ser analisado
		alphamap	Mapa de transparência do elemento de primeiro plano
		pixel_format	Formato do *pixel* (restritos aos tipos definidos em *FormatType*)
chCaptureZ()	double	z_coordinate[3][4]	Matriz de transformação com coordenadas do marcador detectado
chGetThresh()	int	buf	*Pixel* a ser analisado

5.2.2 **Definição do algoritmo**

A definição do algoritmo implementado nas funções *chSwapBackgroundBlue()* e *chSwapBackgroundGreen()* foi baseada na análise de protótipos desenvolvidos para executar diversas soluções para o *chromakey*. Os detalhes desta análise são descritos nesta Subseção.

Para identificação da cor do plano de fundo e extração do elemento de interesse, implementaram-se três algoritmos: o algoritmo da predominância da cor, o da faixa de cor, e o de van den Bergh e Lalioti (1999). Todos os protótipos foram definidos para substituir o plano de fundo por uma constante de cor, isolando-se o elemento de primeiro plano.

Como foi descrito na Subseção 3.2.1, as tonalidades de azul não são encontrados na pele humana. Por esse motivo, essa cor é normalmente utilizada em planos de fundo, quando se realiza o *chromakey*. No entanto, também se observou que as câmeras digitais modernas conseguem identificar mais facilmente os canais de verde.

Objetivando verificar possíveis diferenças na qualidade da composição produzida, foram implementados protótipos para isolamento das cores azul e verde, a partir dos três algoritmos citados.

Nesta Seção, em todos os protótipos desenvolvidos, o plano de fundo foi substituído pelo amarelo, para que impurezas sejam facilmente visualizadas. Os pixels analisados estão no formato BGRA, que é o padrão de captura do ARToolkit na plataforma Win32, e são apontados por *dataPtr* , ponteiro do tipo *ARUint8*, já citado anteriormente.

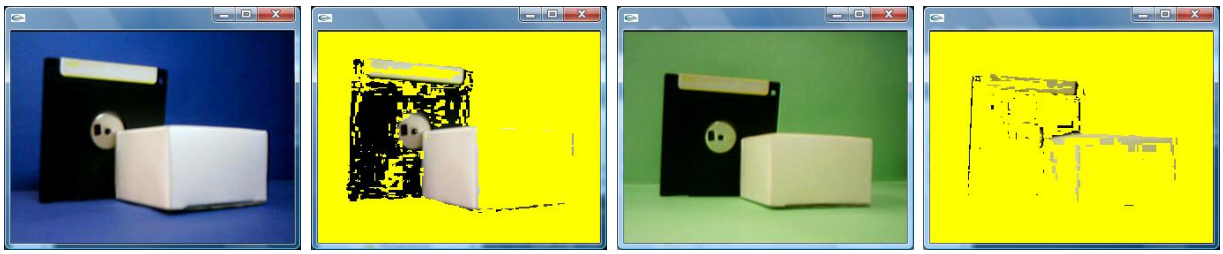
5.2.2.1 Algoritmo da predominância da cor

O algoritmo da predominância da cor é simples e intuitivo. Consiste basicamente em analisar cada *pixel* e substituir os *pixels* onde o canal da cor possui valor maior que os demais.

O trecho do algoritmo que, no exemplo, procura eliminar o azul, comparando esse canal aos canais de verde e vermelho, é exibido na Figura 5.9. Caso o canal azul predomine, valores são atribuídos para tornar o *pixel* amarelo. Os resultados são mostrados na Figura 5.10.

```
ARUint8         *dataPtr; // BGRA
 for(i=0;i<imagesize;i+=4){
  if (dataPtr[i] > dataPtr[i+1] && dataPtr[i] > dataPtr[i+2] ){
    dataPtr[i]=0;   // Canal Azul
    dataPtr[i+1]=255;   // Canal Verde
    dataPtr[i+2]=255;   // Canal Vermelho
  }
} /* dataPtr [i+3] = Canal Alfa */
```

Figura 5.9: Implementação do algoritmo da predominância da cor



(a) **(b)** **(c)** **(d)**

Figura 5.10: Protótipos do algoritmo da predominância da cor em execução

Na imagem resultante da execução do protótipo do algoritmo da predominância da cor, em um plano de fundo azul, pode-se observar que, embora o fundo tenha sido substituído por completo, partes dos elementos de primeiro também foram substituídas (Figura 5.10b). Isso ocorre devido a esta

abordagem provocar também a eliminação de alguns tons de cinza, como mencionado na Subseção 3.4.3.

O algoritmo mostrou-se ainda mais ineficiente em relação à qualidade da imagem quando o verde é a cor a ser substituída (Figura 5.10d). Constatou-se que o tom de verde utilizado era mais reflexivo que o azul e, por esse motivo, seriam necessários ajustes no *threshold* do *chromakey*.

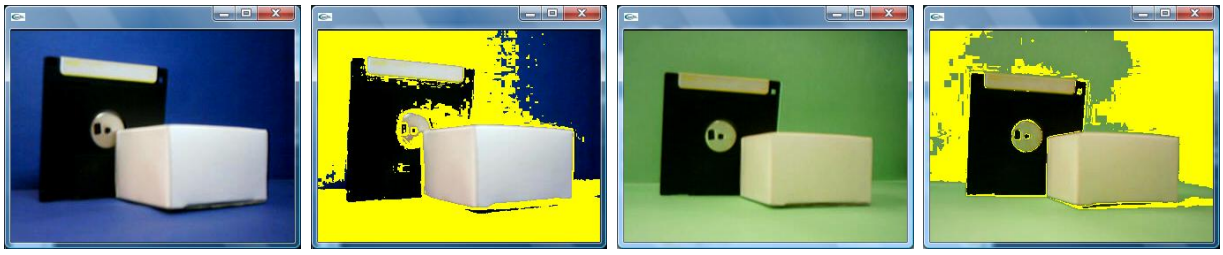
5.2.2.2 Algoritmo da faixa de cor (Pirollo, 2006)

O segundo algoritmo implementado foi o algoritmo da faixa de cor. Nesse algoritmo, também bastante intuitivo, a idéia é isolar toda tonalidade que se deseja eliminar. Limites máximos e mínimos são estabelecidos para cada canal: azul, verde, vermelho e alfa, buscando-se o isolamento do azul.

O trecho de código mostrado na Figura 5.11, é a implementação do algoritmo da faixa de cor aplicado às tonalidades de azul.

```
ARUint8 *dataPtr; // BGRA
/* Limites do Chromakey */
int blue_min  = 0; int blue_max  = 255;
int green_min = 0; int green_max = 100;
int red_min   = 0; int red_max   = 100;
int alpha_min = -1;int alpha_max = 256;
for(i=0;i<imagesize;i+=4){
  if (dataPtr[i] > blue_min && dataPtr[i] < blue_max &&
      dataPtr[i+1]> green_min && dataPtr[i+1] < green_max &&
      dataPtr[i+2] > red_min && dataPtr[i+2] < red_max &&
      dataPtr[i+3] > alpha_min && dataPtr[i+3] < alpha_max){
    dataPtr[i]=0;       // Canal Azul
    dataPtr[i+1]=255;   // Canal Verde
    dataPtr[i+2]=255;   // Canal Vermelho
  }
} /* dataPtr [i+3] = Canal Alfa */
```

Figura 5.11: Implementação do algoritmo da faixa de cor



(a) (b) (c) (d)

Figura 5.12: Protótipos do algoritmo da faixa de cor em execução

O algoritmo da faixa de cor, como mostrado na Figura 5.12, também não apresentou resultados satisfatórios em relação à qualidade da composição exibida. Parte do plano de fundo permanece visível, e ruídos podem ser visualizados nos elementos de interesse (Figura 5.12b).

Estas imperfeições podem ser minimizadas alterando-se os valores limites das faixas de azul. No entanto, esse ajuste é trabalhoso e, na maioria das vezes, insuficiente. As sombras dos elementos reais sobre o fundo e mudanças na iluminação, mesmo que mínimas, provocam ruídos na imagem resultante.

O algoritmo apresentou os mesmos problemas verificados no protótipo que isola o fundo azul em relação ao verde. A delimitação da faixa de verde é igualmente trabalhosa, e a composição não apresenta resultados satisfatórios (Figura 5.12d).

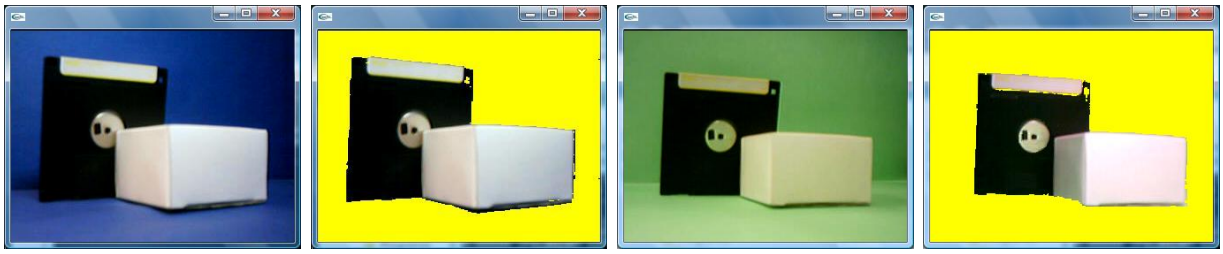
5.2.2.3 Algoritmo de van den Bergh e Lalioti (1999)

O terceiro algoritmo implementado para o módulo "Chromakey" foi o de van den Bergh e Lalioti (1999). Como descrito detalhadamente na Seção 3.4.3, esse algoritmo consiste em calcular a distância do canto azul do cubo

RGB em relação a um plano que o corta. Isso é feito multiplicando-se o valor do canal de azul por dois, e subtraindo-se os canais de verde e vermelho, como mostrado na Figura 5.13.

```
ARUint8         *dataPtr; // BGRA
for(i=0;i<imagesize;i+=4){
  k = (dataPtr[i]<<1) - dataPtr[i+2] - dataPtr[i+1];
    if (k > lim){
      dataPtr[i]=0;       // Canal Azul
      dataPtr[i+1]=255;   // Canal Verde
      dataPtr[i+2]=255;   // Canal Vermelho
    }
}
/* dataPtr [i+3] = Canal Alfa */
```

Figura 5.13: Implementação do algoritmo de van den Bergh e Lalioti (1999)



(a) **(b)** **(c)** **(d)**

Figura 5.14: Protótipos do algoritmo de van den Bergh e Lalioti (1999) em execução

O algoritmo de van den Bergh e Lalioti (1999), como se pode observar na Figura 5.14, mostrou-se eficiente em relação à substituição do fundo azul. Os elementos de primeiro plano foram totalmente preservados, e a interferência das sombras dos elementos reais sobre o plano de fundo não é percebida (Figura 5.14b). O protótipo mostrou-se também menos sensível a mudanças de iluminação que os analisados anteriormente.

Aplicado sobre fundo verde, o algoritmo apresentou um comportamento semelhante ao verificado no fundo azul, exceto pelo fato de exigir ajuste no *threshold*, caso a mesma iluminação seja utilizada (Figura 5.14d).

Provavelmente, isso ocorre devido à característica mais reflexiva do tom de verde utilizado como plano de fundo, o que já foi relatado neste estudo.

5.2.3 Novos planos de fundo

Além da eliminação do fundo original, é necessário inserir outra imagem como novo plano de fundo da cena. A captura da imagem ou filme utilizado na substituição pode ser feita por meio de um aplicativo (PIROLLO, 2006), que armazena a entrada de vídeo.

Pretende-se estender a funcionalidade da função *chLoadFile()*, responsável por esta tarefa, para que suporte formatos de imagem como BMP e JPEG, e de vídeo, como AVI ou MPEG, possibilitando que esses arquivos possam ser carregados como novo plano de fundo. Parte do código fonte do aplicativo citado é mostrada na Figura 5.15.

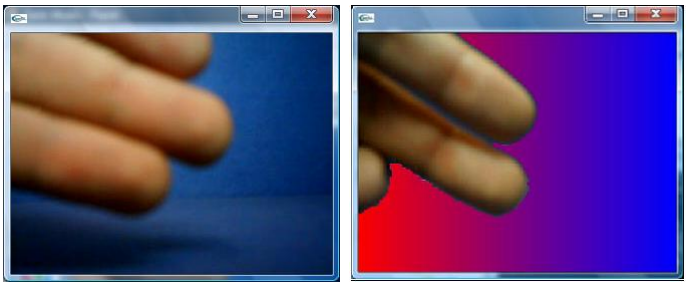
```
if((flag==1) && (frame < frame_number)){
i=0; frame++;
  for(j=0;j<image_size;j++{
    r.buffer[j].pixel[0] = dataPtr[i]; r.buffer[j].pixel[1] = dataPtr[i+1];
    r.buffer[j].pixel[2] = dataPtr[i+2]; r.buffer[j].pixel[3] = dataPtr[i+3];
    i+=4;} fwrite(&r,sizeof(rec),1,fptr);
}
```

Figura 5.15: Trecho do código fonte do aplicativo para captura do plano de fundo

Utilizando-se o algoritmo de van den Bergh e Lalioti (1999), foram implementados três protótipos, onde o fundo azul é substituído por uma imagem estática, por uma seqüência de *frames*, ou é preenchido por um objeto virtual. Obtiveram-se a imagem estática e a seqüência de *frames* por meio do aplicativo citado.

No primeiro protótipo implementado, o azul foi retirado, e o fundo

da cena foi preenchido por um elemento virtual; no caso, um retângulo do tamanho da janela. Nesse protótipo, as funções relacionadas a abertura e leitura de arquivos não são utilizadas. O resultado pode ser visualizado na Figura 5.16.



(a) Ambiente real **(b)** Objeto virtual como plano de fundo

Figura 5.16: Substituição do plano de fundo por um objeto virtual

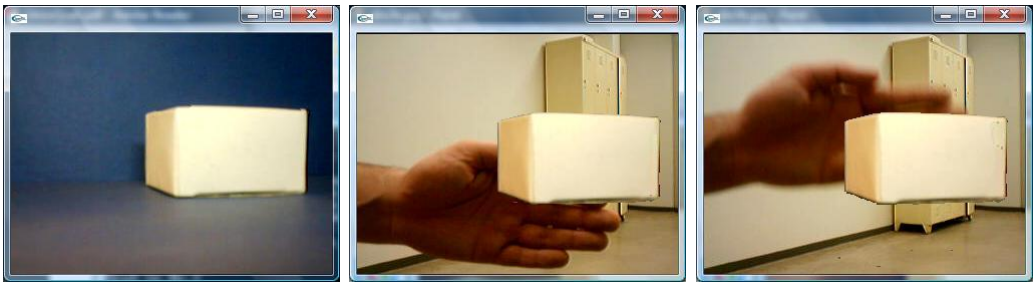
Em um segundo protótipo, realizou-se a mudança do fundo azul por uma imagem pré-capturada por meio do aplicativo mencionado. A imagem da câmera, formada por um único *frame*, é analisada, e os *pixels* azuis são substituídos pela imagem armazenada. O resultado obtido pode ser visualizado na Figura 5.17.

(a) Ambiente real **(b)** Imagem estática como plano de fundo

Figura 5.17: Substituição do plano de fundo por uma imagem estática

No terceiro protótipo, exibe-se a substituição do fundo azul por uma

seqüência de *frames*, também capturados pelo aplicativo citado. O arquivo que contém o novo plano de fundo corresponde a trinta *frames* capturados e está no formato binário. Na Figura 5.18 mostra-se o resultado da execução do protótipo.



(a) Ambiente real **(b)** Imagem dinâmica como plano de fundo (*frame* 1) **(c)** Imagem dinâmica como plano de fundo (*frame* 2)

Figura 5.18: Substituição do plano de fundo por uma seqüência de *frames*

5.2.4 Marcadores invisíveis

Comum a sistemas de RM baseados em rastreamento de marcadores, uma limitação encontrada é a permanência destas marcas no ambiente misturado. Muitas vezes, o elemento virtual gerado não obstrui por completo o marcador real a que esteja vinculado. Isso ocorre em aplicações baseadas no ARToolkit e outras bibliotecas semelhantes.

O fato de a abordagem proposta não utilizar o plano de fundo original para compor a cena resultante pode ser considerado uma alternativa para contornar esse problema.

Para que não seja visualizado no ambiente misturado, o marcador real deve ser retirado, juntamente com o plano de fundo, no momento da aplicação do *chromakey*. Para isso, esses marcadores devem ser construídos

em tons de cores próximos à cor do plano de fundo a ser extraído.

A identificação dos marcadores em cena, por meio do ARToolkit, é feita a partir de uma imagem binária, como descrito na Seção 2.5.5.1. Por esse motivo, os marcadores "invisíveis" devem ser elaborados em tons de azul ou verde, diferentes o suficiente (ou distantes na escala de cores), para que possam, a partir do *threshold* do ARToolkit, serem identificados como preto e branco.

Ao mesmo tempo, estas tonalidades devem ser parecidas (ou próximas na escala de cores) o suficiente, para que sejam reconhecidas como azul (ou verde) pelo *chromakey*, fazendo com que os marcadores sejam extraídos no processo. Naturalmente, a qualidade do dispositivo de captura da imagem e a iluminação do ambiente são importantes na utilização desse tipo de recurso.

Na Figura 5.19(a), mostrou-se um dos marcadores, elaborado em preto e branco, e utilizado pelo ARToolkit. Obviamente, devido ao contraste, estas são as cores ideais a serem utilizadas em marcadores; no entanto, as marcas mostradas nas Figuras 5.19(b) e 5.19(c) foram testadas e apresentaram resultados satisfatórios nos processos de extração do *chromakey* e de detecção do ARToolkit, como será mostrado no protótipo da Subseção 5.2.5.1.

(a) **(b)** **(c)**

Figura 5.19: Marcadores elaborados em tons semelhantes ao plano de fundo

Nesse tipo de abordagem, um detalhe importante a ser observado diz respeito ao o processo de detecção do marcador que deve ser feito na imagem original. Isso significa que a função *arDetectMarker()* do ARToolkit deve ser

chamada antes da troca do plano de fundo, devido ao marcador não estar mais visível na nova imagem.

5.2.5 Composições com profundidade utilizando *chromakey*

Verificada a viabilidade da utilização do *chromakey* para a extração dos elementos de primeiro plano, definidos os algoritmos que compõem as funções do módulo "Chromakey", os tipos de planos de fundo que podem ser carregados e os marcadores que podem ser utilizados, é possível desenvolver os protótipos do módulo "Aplicações", que implementa a estrutura descrita na Subseção 5.1.3.

As demais funções do ARToolkit, além das responsáveis pela captura de vídeo, devem ser utilizadas. Por conseqüência, importa importa o entendimento do funcionamento básico das aplicações baseadas na biblioteca.

Resumidamente, as tarefas realizadas por uma aplicação ARToolkit podem ser organizadas da seguinte forma (KATO *et al.*, 1999):

1. Inicialização do vídeo e leitura dos arquivos de parâmetros, dos marcadores e da câmera.
2. Captura do quadro de vídeo.
3. Detecção dos marcadores e padrões conhecidos no *frame* capturado.
4. Cálculo da transformação da câmera em relação ao padrão detectado.
5. Geração do objeto virtual sobre o padrão detectado.
6. Encerramento da captura do vídeo.

As tarefas 1 e 6 são executadas, respectivamente, apenas na inicialização e no término da aplicação, ao passo que as tarefas de 2 a 5 são executadas repetidamente até o encerramento da aplicação (KATO *et al.*, 1999).

Baseado na seqüência descrita, o módulo implementado pode ser utilizado como um componente da biblioteca para o desenvolvimento dos protótipos que serão mostrados nas Subseções 5.2.5.1 e 5.2.5.2.

Primeiramente, detalham-se os passos para sobreposição dos objetos virtuais por elementos reais. Em seguida, implementa-se o protótipo que adiciona a consideração de profundidade entre os elementos da cena, reais e virtuais.

5.2.5.1 **Sobreposição do objeto virtual**

As funções da biblioteca ARToolkit, como foi mencionado, não possibilitam o desenvolvimento de aplicações em que os elementos reais sejam exibidos à frente de objetos virtuais. Utilizando-se o módulo desenvolvido neste trabalho, torna-se viável implementar esse tipo de recurso.

Descrevendo-se passo a passo este processo:

Inicialmente, o arquivo contendo o novo plano de fundo deve ser carregado, utilizando-se a função *chLoadFile()*, ainda em fase de inicialização.

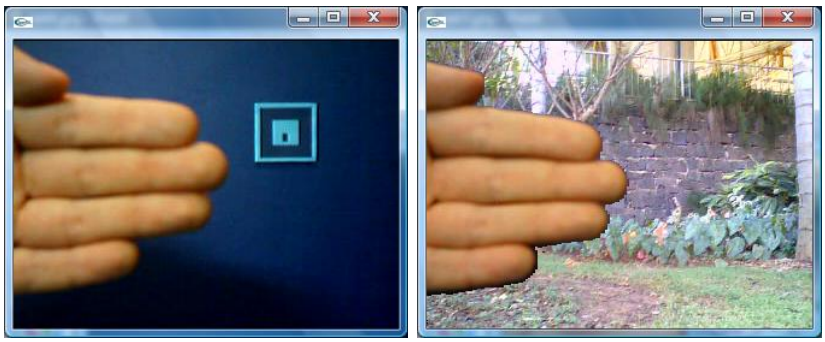
Por meio da função *arVideoGetImage()* do ARToolkit, um *frame* da imagem capturada pela câmera é armazenado em um *buffer* (*Buffer#1*), cujo formato é mostrado na Figura 5.5. Os marcadores são detectados pela função *arDetectMarker()*, e o arquivo contendo o novo plano de fundo pode ser lido por meio da função *chReadFile()*.

Utilizando-se o algoritmo de *chromakey*, implementado na função *chSwapBackgroundBlue()*, a imagem é analisada, *pixel* a *pixel*, substituindo-

se o fundo azul pela imagem contida no arquivo, e em seguida, transferida ao *framebuffer*.

Nesse mesmo processo, o *alphamap* é gerado no *Buffer#2* atribuindo-se o valor zero (0) aos *pixels* pertencentes ao plano de fundo, e o valor duzentos e cinqüenta e cinco (255) aos *pixels* pertencentes ao elemento de primeiro plano.

O resultado desse processo é mostrado na Figura 5.20. Na Figura 5.20(a) tem-se a cena capturada apenas com um marcador, fixa no fundo azul. Na Figura 5.20(b), exibe-se a imagem visualizada após a chamada da função *chSwapBackgroundBlue()*.



(a) Marcador fixado no fundo **(b)** Aplicação do *chromakey*

Figura 5.20: Substituição do plano de fundo

Como pode ser observado, o marcador foi elaborado em tons de azul, como mostrado na Subseção 5.2.4, para que este não seja visualizado na imagem resultante. Em seguida, utilizando-se as coordenadas obtidas pela função *arGetTransMat()*, os objetos virtuais podem ser gerados e transferidos ao *framebuffer*.

É importante notar que, nesse ponto, a imagem apresenta um erro de registro, pelo fato de o objeto virtual ter sido transferido diretamente ao *framebuffer*, sobre a imagem capturada pela câmera. Como se pode observar na Figura 5.20(a), o marcador está fixo ao fundo. Portanto, o elemento real

(mão) deveria sobrepor o objeto virtual renderizado (chaleira).

Para facilitar a observação do erro e evitar o problema da oclusão, o objeto virtual foi transladado. Assim, como mostrado na Figura 5.21, tal objeto é exibido em primeiro plano, obstruindo o elemento real e caracterizando o erro de registro mencionado.

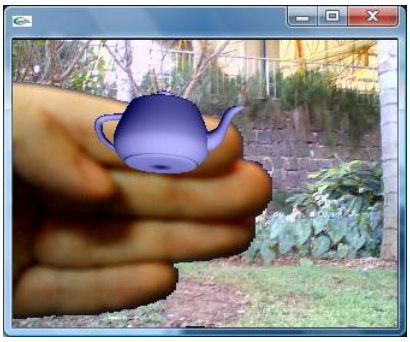


Figura 5.21: Erro de registro observado

Para correção do erro exemplificado, a imagem mostrada na Figura 5.21 é recuperada do *framebuffer* e trazida ao *Buffer#1*. Nesta imagem, o objeto virtual, gerado a partir de coordenadas tridimensionais, já é parte do plano de fundo.

O *alphamap*, armazenado no *Buffer#2* é aplicado sobre a imagem, que, em seguida, é escrita novamente no *framebuffer*. Todo esse processo, que foi descrito de forma simplificada, é realizado pela função *chDrawAlphamap()*. Na Figura 5.22 apresenta-se o resultado obtido.

5.2.5.2 **Consideração da profundidade**

Como foi demonstrado, as funções do módulo "Chromakey" permitem realizar a sobreposição do objeto virtual por algo real na cena. No entanto, no

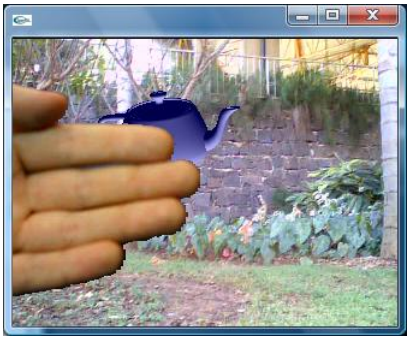


Figura 5.22: Correção do problema de registro

decorrer da execução, este elemento real também poderá estar posicionado mais distante ao observador que algum marcador vinculado a um objeto virtual e, portanto, deve ser obstruído por esse objeto.

Para que se obtenha coerência na composição da cena, torna-se, então, necessário um método que permita a identificação automática da profundidade dos elementos (reais e virtuais). Para isso, deve-se obter a localização do elemento real.

A solução utilizada, como mencionado na Subseção 5.1.3, foi a fixação de um marcador no elemento real (elemento de interesse), como mostrado na Figura 5.4. Desse modo, sua profundidade pode ser obtida pela função *chCaptureZ()*, definida dentro da função *chVisibilityAndDepth()*.

Neste protótipo, buscando maior facilidade na implementação, o elemento real foi associado ao primeiro marcador da lista de marcadores da cena (*obj_id=0*). Foram utilizados três marcadores, definidos em um arquivo de configuração do usuário , que foram mantidos na cena para visualização de suas posições.

Como no protótipo anterior, a função *chLoadFile()* é chamada durante a inicialização. A imagem da câmera, capturada por meio da função *arVideoGetImage()*, é armazenada no *Buffer#1*. Os marcadores são detectados (*arDetectMarker()*) e o arquivo contendo o novo plano de fundo é

lido (*chReadFile()*). A função *chSwapBackgroundBlue()* substitui o plano de fundo e armazena o *alphamap* da imagem no *Buffer#2*.

Após o teste de visibilidade, feito por meio da função *chVisibilityAndDepth()*, para composição do ambiente baseado na posição dos marcadores, as coordenadas "z" obtidas são comparadas à coordenada "z" do marcador vinculado ao elemento real, como mostrado no trecho do código fonte da Figura 5.23.

```
chVisibilityAndDepth(objectnum,marker_num,marker_info,object);

for (i=1;i<objectnum;i++){
  if(object[0].z_coord<object[i].z_coord){draw(object, objectnum,1,i);extraction=1;}
}
if (extraction==1){chDrawAlphamap(comprimento, altura, dataPtr, &foreground,_BGRA);}
for (i=1;i<objectnum;i++){
  if(object[0].z_coord>=object[i].z_coord){draw(object, objectnum,1,i);
  }
}
```

Figura 5.23: Trecho do código fonte que compara as coordenadas "z" dos marcadores

Se existirem objetos virtuais cuja coordenada "z" possua valor maior (mais distante) que a coordenada "z" do marcador real, esses objetos são gerados sobre a imagem do *Buffer#1* (elemento real em frente ao novo plano de fundo), e escritos no *framebuffer*.

Se for encontrado, pelo menos, um objeto virtual, a imagem contida no *Buffer#1* deve ser recuperada e o *alphamap*, armazenado no *Buffer#2*, deve ser inserido sobre a imagem. Caso existam vários objetos virtuais, faz-se o controle de profundidade — nesta etapa realizado somente entre esses objetos — pela utilização do *Depth Buffer* (*buffer* de profundidade do *framebuffer* OpenGL) que é habilitado pela chamada da função e do respectivo parâmetro *glEnable(GL_DEPTH_TEST)*.

Com a aplicação do *alphamap*, o elemento real será exibido à frente dos objetos virtuais já renderizados. As tarefas de recuperação da imagem

do *framebuffer* e da inserção do *alphamap* sobre esta imagem, como já foi mencionado, são realizadas pela função *chDrawAlphamap()*.

É importante observar que, caso não existam objetos virtuais mais distantes do observador que o elemento real, não há necessidade de utilização do *alphamap*.

O passo seguinte é a renderização dos objetos virtuais posicionados mais próximos do observador do que o elemento real. Novamente, nenhum controle adicional em relação à profundidade desses objetos virtuais é necessário além da habilitação do *Depth Buffer*.

Para a visualização dos resultados, na Figura 5.24, têm-se o ambiente real e três marcadores em cena. A localização dos objetos virtuais renderizados é obtida pela detecção dos marcadores laterais, e o posicionamento do ator real, pela detecção do marcador ao centro.

Figura 5.24: Geração dos objetos virtuais pelo ARToolkit

Na Figura 5.25, em que se aplica o *chromakey*, o marcador localizado ao centro (vinculado ao ator real) está posicionado mais próximo do observador que os demais. Portanto, o elemento real é gerado em primeiro plano.

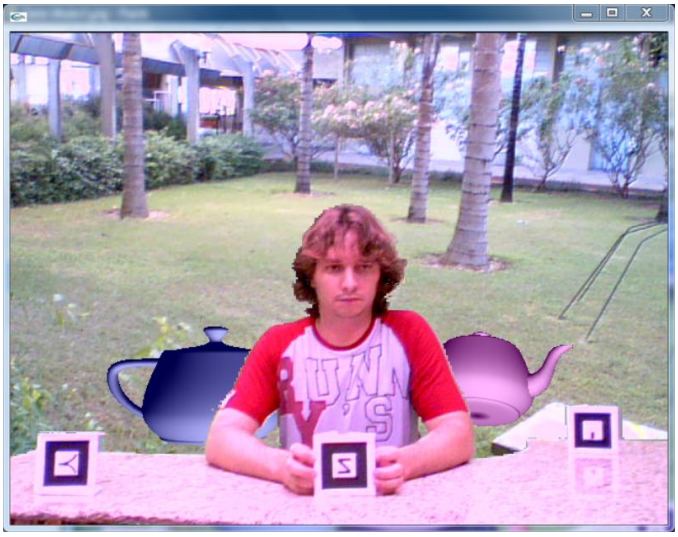


Figura 5.25: Elemento real visualizado em primeiro plano

Na Figura 5.26, o marcador vinculado ao ator está posicionado mais distante do observador que o marcador à esquerda e mais próximo que o marcador à direita. Portanto, o objeto virtual à esquerda é gerado em primeiro plano e o elemento real obstrui apenas o objeto virtual à direita (marcador posicionado mais ao fundo).

Figura 5.26: Elemento real visualizado entre os objetos virtuais

O posicionamento do marcador vinculado ao elemento real mais ao fundo que os demais marcadores faz com que ambos os objetos virtuais sejam visualizados em primeiro plano, como mostrado na Figura 4.26. Em todas as situações descritas, exceto na Figura 4.23, a imagem final é exibida coerentemente em relação à profundidade.

Figura 5.27: Objetos virtuais visualizados em primeiro plano

5.3 Testes e análise de desempenho

Nesta Seção, detalham-se os testes e análises de desempenho realizado nos protótipos desenvolvidos. Na Subseção 5.3.1, descreve-se o ambiente experimental utilizado, e, na Subseção 5.3.2, exibem-se os resultados obtidos da execução dos protótipos e sua respectiva análise.

5.3.1 Ambiente experimental

Para a realização dos experimentos, montou-se um ambiente composto por uma estrutura em formato de caixa, com dimensões de 500 x 300 x 300 mm, com três lados adjacentes, mostrado na Figura 5.28.

Figura 5.28: Ambiente experimental 1

O interior da estrutura foi revestido com papel cartão na cor azul e, para a iluminação, utilizaram-se lâmpadas fluorescentes de 9 *watts* e 160 *lumens*.

Os testes finais, como o protótipo da Subseção 5.2.5.2, foram realizados em um estúdio para *chromakey* com fundo na cor verde, mostrado na Figura 5.29. A câmera foi posicionada a uma distância de 1,5 m do plano de fundo.

Considerando que o projeto do ARToolkit se preocupa em manter a biblioteca compatível com várias plataformas, e, buscando-se a manutenção desta característica, as funções do módulo "Chromakey" foram implementadas para utilização nos sistemas operacionais Windows e Linux, e testadas

com a versão 2.72.1 da biblioteca.

Figura 5.29: Ambiente experimental 2

O sistema operacional Windows Vista Ultimate 32 bits e Windows XP Professional SP2 foram utilizados para os testes, dado que estes sejam amplamente utilizados por usuários e desenvolvedores.

Para o desenvolvimento, utilizou-se a ferramenta Microsoft Visual Studio 2005 devido à facilidade de acesso a documentação e ao projeto do ARToolkit ser baseado nesta IDE (*Integrated Development Environment*).

Entre as distribuições Linux, o Ubuntu 6.06 LTS 32 bits (Ubuntu, 2006) foi escolhido para realização dos testes, levando-se em consideração fatores como: simplicidade de instalação do sistema, disponibilidade de *driver* para a câmera de vídeo e facilidade de instalação de bibliotecas dependentes para o funcionamento do ARToolkit, feito por meio do APT (*Advanced Packaging Tool*), presente na distribuição.

No Linux, algumas IDEs para o compilador C foram testadas, e o Anjuta versão 1.2.4a (Naba Kumar, 2007) mostrou-se mais adequado, devido à facilidade de instalação (feita por meio do APT), à documentação de auxílio ao desenvolvedor disponível e ao recurso do assistente de importação

de projeto. Esse último, possibilita a importação do projeto ARToolkit para o ambiente Anjuta.

Os modelos de câmeras utilizados no desenvolvimento dos protótipos, mostrados na Figura 5.30, foram a Webcam Microsoft LifeCam VX-3000 (Figura 5.30a), a PC-Cam Avox/Vivicam 3780 (Figura 5.30b), e a Webcam Clone 11125 (Figura 5.30c). Somente a última possui driver para funcionamento em sistemas Linux. Os demais componentes de *hardware* utilizados durante o desenvolvimento do projeto serão descritos na Subseção 5.3.2.6.

(a) Microsoft LifeCam VX-3000 **(b)** PC-Cam Avox/Vivicam 3780 **(c)** Webcam Clone 11125

Figura 5.30: Câmeras utilizadas no desenvolvimento no projeto

5.3.2 **Análise de desempenho**

Considerados de tempo real, os sistemas de RM têm restrições rígidas quanto ao tempo de resposta. Por esse motivo, direcionaram-se os testes realizados considerando alguns critérios de desempenho relevantes, a saber: a taxa de *frames* por segundo resultante da execução do protótipo e de acordo com o tipo de plano de fundo; o sistema operacional; a quantidade de polígonos que formam o objeto virtual renderizado; a quantidade de marcadores

em cena; o posicionamento dos marcadores e os componentes de *hardware* utilizados.

Em todos os testes, foram utilizadas as funções do ARToolkit versão 2.72.1 e o Ambiente Experimental 1, descrito na Subseção 5.3.1, com a câmera posicionada a 450 mm do plano de fundo. Em todos os casos, utilizou-se o mesmo arquivo de imagem em o fundo azul foi substituído por uma imagem estática. O mesmo procedimento foi adotado em relação ao plano de fundo dinâmico.

A iluminação do ambiente também não foi alterada no decorrer dos testes, exceção feita aos testes de análise de desempenho em diferentes componentes de *hardware*. Nesse caso, uma pequena variação pode ter ocorrido devido às máquinas utilizadas estarem em diferentes locais.

Os valores mostrados nos gráficos das Subseções seguintes dizem respeito à média da taxa de *frames* por segundo, obtida das execuções do protótipo em questão. Cada protótipo foi executado sessenta vezes — trinta analisando imagens com resolução 320x240 *pixels* e trinta analisando imagens com resolução 640x480 *pixels* —, permanecendo em execução durante aproximadamente 20 segundos.

A cada dez execuções de um mesmo protótipo, a máquina era reiniciada; em seguida, executava-se um novo protótipo até que as dez execuções de cada protótipo fossem realizadas.

Ao final, repetiam-se os passos citados, até que cada protótipo fosse executado sessenta vezes, somando um total de seis baterias de testes. Procurou-se, com esses procedimentos, evitar que instabilidades provocadas por outros *softwares* influenciassem os resultados da avaliação.

5.3.2.1 Diferentes Planos de Fundo

Primeiramente, foram realizados testes para avaliar o desempenho do protótipo, quando o fundo azul é substituído pelos diferentes planos de fundo, detalhados na Subseção 5.2.3. Para tanto, ao aplicar-se o *chromakey*, o plano de fundo foi substituído por um elemento virtual, uma cor constante, uma imagem estática e uma seqüência de *frames*.

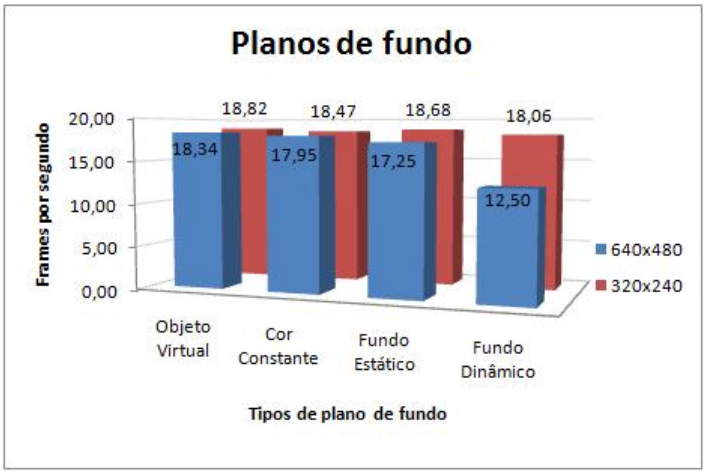


Figura 5.31: Substituição do fundo azul por diferentes tipos de imagens

Como se pode observar, nos gráficos mostrados na Figura 5.31, à medida que aumentava a complexidade do plano de fundo inserido, ocorria uma queda no desempenho do protótipo. Essa queda foi mais acentuada no protótipo utilizando imagem com resolução 640x480, em que um arquivo de filme foi inserido como novo plano de fundo.

Em relação à mudança de resolução de 320x240 para 640x480, a queda de desempenho não pode ser considerada significativa, pois a quantidade de *pixels* analisados foi quadruplicada (de 76800 para 307200). Novamente, a maior variação ocorreu no protótipo em que o plano de fundo é substituído por uma seqüência de *frames*.

5.3.2.2 Diferentes Sistemas Operacionais

Para a avaliação das funções do módulo "Chromakey" em diferentes sistemas operacionais, valeu-se dos sistemas Linux Ubuntu 6.06 LTS, Windows Vista Ultimate, Windows XP Professional SP2. Para os testes realizados, entre outros elementos que consomem tempo de processamento, aplicativos como antivírus, *firewalls* foram desativados, e apenas os processos essenciais ao funcionamento do computador permaneceram em execução.

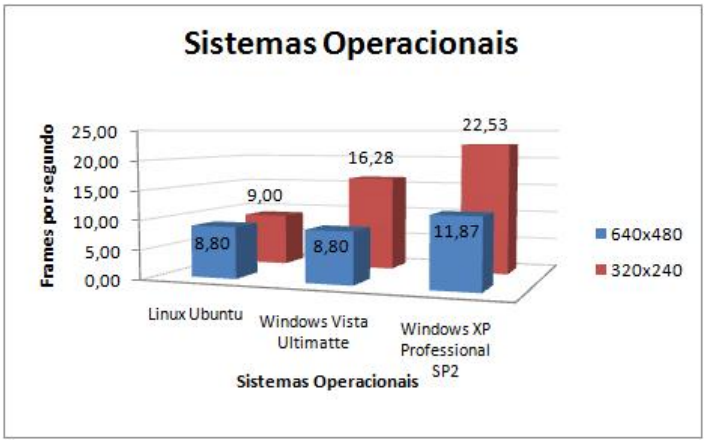


Figura 5.32: Diferentes sistemas operacionais

Nos gráficos mostrados na Figura 5.32, pode-se observar um melhor desempenho na execução do protótipo sobre o Windows XP, em relação a outros dois sistemas analisados. Isso se deve, provavelmente, ao fato de o Windows Vista exigir maior quantidade de memória para seu funcionamento.

Em relação ao sistema Linux Ubuntu, o desempenho do protótipo é, com efeito, prejudicado pela baixa taxa de *frames* capturados pela câmera. A taxa de captura abaixo do esperado pode ser resultado da pouca eficiência do *driver* genérico para sistemas Linux, da *webcam* utilizada no teste.

Isso pode ser constatado comparando-se os valores referentes às imagens de diferentes resoluções. As quedas de desempenho dos sistemas Windows XP e Windows Vista Ultimate, quando modificada a resolução, foram proporcionais, ao passo que, no Linux Ubuntu, praticamente não houve alteração na taxa de *frames* por segundo obtida.

5.3.2.3 Quantidade de polígonos renderizados na formação do objeto virtual

Outro fator analisado trata da complexidade do objeto virtual. Objetos mais complexos, obviamente, são modelados a partir de grandes quantidades de polígonos, que devem ser renderizados em tempo real.

Para avaliar o impacto no desempenho proporcionado pela quantidade de polígonos do objeto gerado sobre o marcador, construíram-se, por meio do OpenGL, conjuntos de triângulos que representam o objeto virtual.

Avaliaram-se objetos formados por dez, cem, mil, cinco mil e dez mil polígonos, como mostrado no gráfico da Figura 5.33.

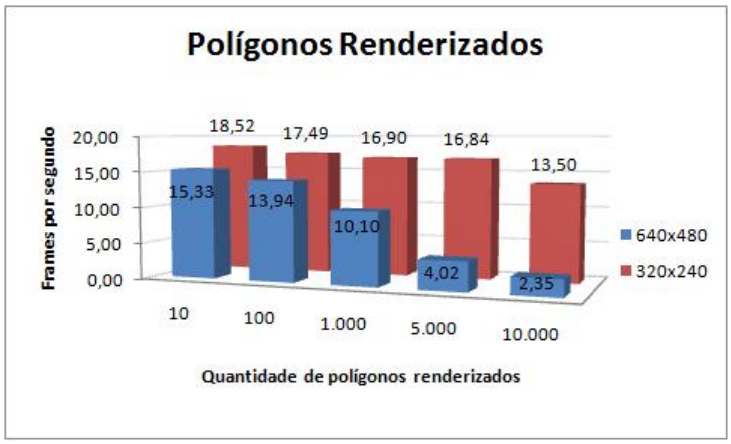


Figura 5.33: Quantidade de polígonos renderizados

À medida que a complexidade do objeto aumenta, nota-se uma queda no desempenho do protótipo. Em relação à mudança de resolução de 320x240 para 640x480, essa queda foi proporcional, considerando-se o aumento da quantidade de *pixels* analisados.

5.3.2.4 Quantidade de marcadores na cena

Em aplicações baseadas no ARToolkit, um dado importante a se considerar relaciona-se à quantidade de marcadores visíveis na cena. Embora regiões quadradas, semelhantes a marcadores, estejam visíveis à câmera, o ARToolkit dispara as rotinas de comparação de padrões, ocasionando um atraso no processamento.

Considerando-se que o algoritmo de *chromakey* analisa a imagem *pixel* a *pixel*, o que aumenta ainda mais o custo computacional, realizaram-se testes para verificar a influência do aumento da quantidade de marcadores, visíveis à câmera, no desempenho do protótipo.

Nesses testes, os marcadores foram posicionados para que, durante todo tempo de execução do protótipo, permanecessem visíveis à câmera. Para certificar-se desta visibilidade, objetos virtuais foram renderizados sobre essas marcas.

No gráfico da Figura 5.34, pode-se observar que a presença de marcadores na cena provoca uma queda acentuada no desempenho do protótipo — o que é verificado comparando-se os valores obtidos das execuções dos protótipos em que nenhum marcador é utilizado, aos valores obtidos das execuções dos protótipos em que apenas um marcador é reconhecido pelo ARToolkit.

Em relação à presença de dois ou três marcadores, não houve alter-

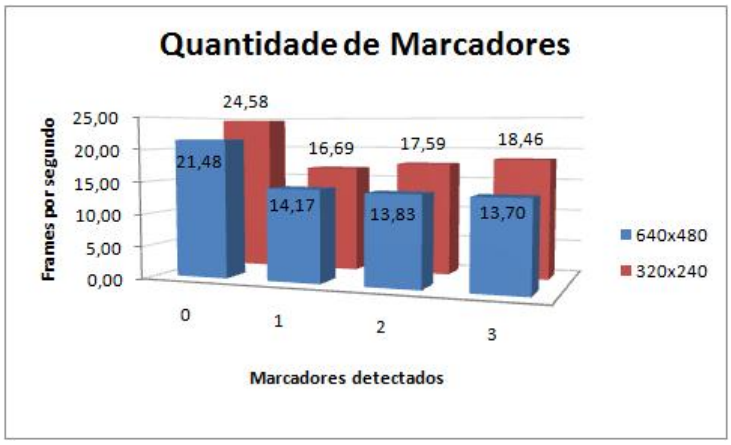


Figura 5.34: Quantidade de marcadores visíveis

ação significativa no desempenho, exceto quando a resolução foi modificada de 320x240 *pixels* para 640x480 *pixels*. Nesse caso, a queda foi proporcional em todos os protótipos, provavelmente por estar mais relacionada ao aumento da quantidade de *pixels* analisados do que propriamente ao número de marcadores reconhecidos.

5.3.2.5 Posicionamento dos marcadores na cena

Como descrito na Subseção 5.2.5.2, o protótipo que constrói a cena misturada, considerando a profundidade dos objetos, verifica o valor das coordenadas "z" dos marcadores, e decide pela utilização ou não do *alphamap*. Por esse motivo, o posicionamento dos marcadores em cena pode influenciar o desempenho do protótipo.

Os testes realizados consideraram três possíveis situações de ocorrência quando três marcadores estão visíveis na cena. Definindo-se a localização do marcador vinculado ao elemento real como referência para os testes, é possível identificar três situações: "o marcador está posicionado à frente dos

demais", "o marcador está posicionado entre os dois outros marcadores" e "o marcador está posicionado mais ao fundo (mais distante) que os outros marcadores". Os resultados dos testes são mostrados no gráfico da Figura 5.35.

Figura 5.35: Posicionamento dos marcadores

Como era esperado, o desempenho é afetado quando o elemento real está posicionado à frente de objetos virtuais. Nesse caso, a função *chDrawAlphamap()* é chamada para se aplique o *alphamap*, afetando o desempenho do protótipo. Obviamente, esse impacto é mais acentuado em imagens maiores (640x480 *pixels*), pois é maior a quantidade de *pixels* redesenhados.

5.3.2.6 Diferentes componentes de *hardware*

O *hardware* utilizado na execução dos protótipos tem influência direta no seu desempenho. Obviamente, quanto melhor a qualidade dos componentes de *hardware*, maiores serão as taxas de frames obtidas. No entanto, fatores específicos como a placa gráfica, a quantidade de memória RAM e a velocidade do processador principal devem ser considerados.

Diante da impossibilidade de uma avaliação mais completa de cada um dos componentes citados, procurou-se identificar variações nas taxas *frames* obtidas em duas configurações de *hardwares*, descritas a seguir.

Hardware 1: *Notebook* HP Pavilion, Processador AMD Turion 64 ML-32 1,8 GHz, Placa Gráfica ATI Mobility Radeon Xpress 200, 1,00 GHz de memória RAM e HD Toshiba ATA 80 GB.

Hardware 2: *Desktop*, Processador Intel Pentium 4 CPU 3,06 GHz, 1,00 GHz de memória RAM e HD Philips ATA 120 GB.

Neste teste, o fator a ser destacado é a presença da placa gráfica no *Hardware 1*. Por outro lado, o *Hardware 2* possui um processador que atinge velocidades mais altas. Os resultados obtidos são mostrados no gráfico da Figura 5.36.

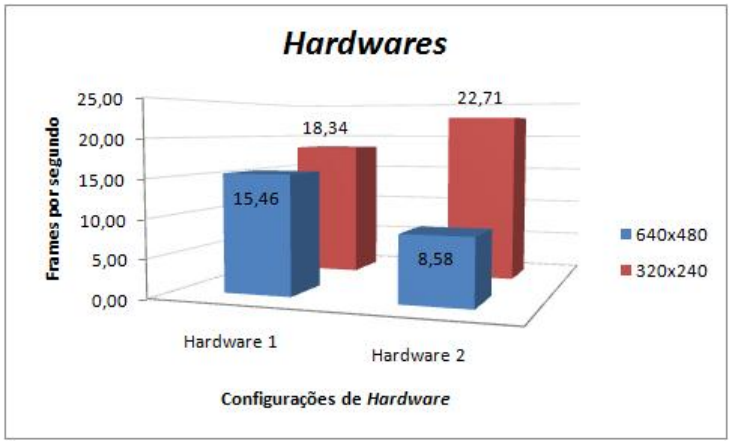


Figura 5.36: Diferentes componentes de *hardware*

Como se pode verificar, a velocidade do processador principal foi o fator preponderante no desempenho do protótipo que utilizava imagens com resolução de 320x240 *pixels*. No entanto, nas imagens com resolução 640x480, o protótipo executado no *Hardware 1* mostrou-se mais eficiente em relação à taxa de *frames* obtida.

O motivo dos melhores resultados do *Hardware 1* em relação ao *Hard-*

ware 2, na execução dos protótipos que utilizam imagens com resolução de 640x480 *pixels*, provavelmente esteja relacionado à presença da gráfica.

Os componentes utilizados na realização de cada teste são descritos na Tabela 5.2.

Tabela 5.2: Componentes utilizados nos testes

		Diferentes Planos de Fundo	Diferentes Sistemas Operacionais	Quantidade de Polígonos	Quantidade de Marcadores	Posição dos Marcadores	Diferentes *Hardwares*
Processador	AMD Turion 64 ML-32 1,8 GHz	X	X	X	X	X	X
	Intel Pentium 4 CPU 3,06 GHz						X
Memória RAM	1,00 GHz	X	X	X	X	X	X
Placa Gráfica	ATI Radeon Xpress 200	X	X	X	X	X	X
Webcam	Webcam Clone 11125		X				
	Microsoft LifeCam VX-3000	X		X	X	X	X
Sistema Operacional	Windows Vista Ultimate 32 bits						X
	Windows XP Professional SP2	X	X	X	X	X	X
	Ubuntu 6.06 LTS 32 bits		X				
Ferramentas de programação	Microsoft Visual Studio 2005	X	X	X	X	X	X
	Anjuta versão 1.2.4a		X				
Outros detalhes	Número de marcadores reconhecidos	3	3	3	0,1,2 e 3	3	3
	Número de execuções do Protótipo	60	60	60	60	60	60
	Tipo do plano de fundo utilizado	Todos	Vídeo	Vídeo	Vídeo	Vídeo	Vídeo
	Tempo de cada execução do protótipo	20 seg.	20 seg.	20 seg.	20 seg.	20 seg.	20 seg.

5.4 Considerações Finais

No presente Capítulo, apresentou-se o processo de desenvolvimento do módulo "Chromakey" e dos protótipos utilizados para os testes finais.

Além de abordar a motivação para o projeto, aqui se discutiram sua estrutura, seus objetivos, suas limitações e a metodologia aplicada no seu desenvolvimento. Em relação à implementação, apresentaram-se as principais estruturas de dados, constantes e funções definidas no módulo, utilizado como um componente para a biblioteca ARToolkit.

Alguns algoritmos de *chromakey* foram implementados, assim como os vários tipos de planos de fundo que se podem utilizar em aplicações. A elaboração de marcadores invisíveis no ambiente misturado também foi apresentada neste Capítulo.

Demonstrou-se, de forma detalhada, a composição de cenas com profundidade utilizando o componente desenvolvido, seguida dos testes e análise de desempenho dos protótipos implementados.

Capítulo 6

CONCLUSÕES E TRABALHOS FUTUROS

Neste estudo experimental, apresentou-se um componente para a biblioteca ARToolkit que possibilita a extração de elementos da cena real, com a finalidade de realizar composições coerentes, em relação à profundidade de objetos reais e virtuais em ambientes de RM. As funções do módulo desenvolvido possibilitam a implementação da consideração da profundidade, o que pode evitar problemas de registro nas aplicações.

Buscando o isolamento do elemento real, propôs-se a técnica do *chromakey*, utilizada há vários anos na indústria do cinema e da televisão. Normalmente, a técnica é aplicada por meio de *hardware* caro, especializado e protegido por patente. No trabalho apresentado, o *chromakey* é implementado inteiramente em *software*, utilizando-se os serviços de captura de vídeo da biblioteca ARToolkit.

Apesar de concentrados no desempenho, os testes realizados indicaram a viabilidade do processo. Muitos aspectos mostraram-se determinantes no que se refere à utilização das funções do módulo, tais como: a iluminação, a qualidade da câmera, da placa gráfica e do processador principal.

Pretende-se, como trabalhos futuros, o estudo das formas de minimização do efeito de "serrilhamento" na silhueta dos elementos de primeiro plano, bem como a adição de sombras, de métodos para identificação de colisão, de múltiplas câmeras e redundância de marcadores para minimizar problemas de oclusão. Além disso, pretende-se estender as funções do módulo para torná-lo compatível com diversos formatos de imagem e vídeo como plano de fundo.

Outra proposta de trabalhos futuros relaciona-se à utilização de modelos virtuais mais complexos. Nesse caso, a biblioteca OSGART (Loosea *et al.*, 2007), que é baseada no ARToolkit poderá vir a ser utilizada. Técnicas para extração de elementos de primeiro plano em imagens arbitrárias, como a implementada na biblioteca OpenCV (Bernardes *et al.*, 2007), também podem ser avaliadas.

Um aspecto a ser observado diz respeito ao fato de a utilização do componente estar limitada a aplicações que aceitem planos de fundo de cor única. Por outro lado, tal restrição possibilita a não visualização do marcador no ambiente. Para tanto, as marcas devem ser elaboradas da mesma cor do plano de fundo para que sejam extraídas pelo *chromakey*.

REFERÊNCIAS

ADOBE SYSTEMS Macromedia director. Disponível em : <http://www.adobe.com/products/director/> Acesso em: 27 jun. 2007.

ARTOOLKIT Artoolkit versão 2.72. Disponível em : <http://sourceforge.net/projects/artoolkit> Acesso em: 19 out. 2006.

AZEVEDO, E.; CONCI, A. *Computação gráfica - teoria e prática.* Rio de Janeiro, RJ, Brasil: Editora Campus, 2003.

AZUMA, R. A survey of augmented reality. *Teleoperators and Virtual Environments*, v. 6, n. 4, p. 355–385, 1997.

AZUMA, R. Overview of augmented reality. In: *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, New York, NY, USA: ACM Press, p. 26, 2004.

AZUMA, R.; BAILLOT, Y.; BEHRINGER, R.; FEINER, S.; JULIER, S.; MACINTVRE, B. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, v. 21, n. 6, p. 34–47, 2001.

BBC RESEARCH Projects: Flash keying. Disponível em : <http://www.bbc.co.uk/rd/projects/virtual/flash-keying> Acesso em: 07 jan. 2007.

VAN DEN BERGH, F.; LALIOTI, V. Software chroma keying in an immersive virtual environment. *South African Computer Journal*, , n. 24, p. 155–162, 1999.

Bernardes, J.; Miranda, F.; Calife, D.; Trias, L.; Tori, R. *Opencv.* Recife, PE, Brasil: In: Cardoso, A. (edit); Kirner, C. (edit); Lamounier, E. (edit); Kelner, J. (edit). Tecnologias para o desenvolvimento de sistemas de Realidade Virtual e Aumentada. p. 111-135, 2007.

Bimber, O.; Raskar, R. Modern approaches to augmented reality. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, New York, NY, USA: ACM Press, p. 1, 2005.

Bowskill, J.; Downie, J. Extending the capabilities of the human visual system: an introduction to enhanced reality. *SIGGRAPH Comput. Graph.*, v. 29, n. 2, p. 61–65, 1995.

Brain, M. Howstuffworks - como funcionam os cromaquis. Disponível em : <http://lazer.hsw.com.br/cenarios-virtuais.htm> Acesso em: 20 jun. 2007.

Braz, J.; Pereira, J. Tarcast: Uma taxonomia para sistemas de realidade aumentada. In: *Actas do 13º Encontro Português de Computação Gráfica*, Vila Real, Portugal, 2005.

Buxton, B.; Fitzmaurice, G. W. Hmds, caves & chameleon: a human-centric analysis of interaction in virtual space. *SIGGRAPH Comput. Graph.*, v. 32, n. 4, p. 69–74, 1998.

Chuang, Y. Y. *New models and methods for matting and compositing.* 2004. 129 f. Tese (Grau em Ciência da Computação & Engenharia) - University of Washington, 2004.

Chuang, Y. Y.; Curless, B.; Salesin, D. H.; Szeliski, R. A bayesian approach to digital matting. In: *CVPR '01: Proceedings of IEEE Computer Vision and Pattern Recognition*, Los Alamitos, CA, USA: IEEE Computer Society, p. 264–271, 2001.

Dias Junior, J. B. *Suporte para captura de movimentos baseado em marcadores passivos utilizando múltiplas câmeras.* 2005. 113 f. Dissertação de Mestrado (Grau em Ciência da Computação) - Centro

Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, 2005.

Douglas, S. A.; Kirkpatrick, A. E. Model and representation: the effect of visual feedback on human performance in a color picker interface. *ACM Trans. Graph.*, v. 18, n. 2, p. 96–127, 1999.

Geiger, C.; Reimann, C.; Sticklein, J.; Paelke, V. Jartoolkit - a java binding for artoolkit. In: *Augmented Reality Toolkit, The First IEEE International Workshop*, p. 5, 2002.

Geogia Tech Dart :the designer's augmented reality toolkit. Disponível em : <http://www.cc.gatech.edu/dart> Acesso em: 02 dez. 2006.

Gibbs, S.; Arapis, C.; Breiteneder, C.; Lalioti, V.; Mostafawy, S.; Speier, J. Virtual studios: An overview. *IEEE MultiMedia*, v. 05, n. 1, p. 18–35, 1998.

Gvili, R.; Kaplan, A.; Ofek, E.; Yahav, G. Depth keying. *Stereoscopic Displays and Virtual Reality Systems X*, v. 5006, n. 1, p. 564–574, 2003.

Haller, M.; Hartmann, W.; Luckeneder, T.; Zauner, J. Combining artoolkit with scene graph libraries. In: *Augmented Reality Toolkit, The First IEEE International Workshop*, p. 2, 2002.

HIT Lab Artoolkit. Disponível em : <http://www.hitl.washington.edu/artoolkit> Acesso em: 08 set. 2006.

Johnson, G. S.; Lee, J.; Burns, C. A.; Mark, W. R. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph.*, v. 24, n. 4, p. 1462–1482, 2005.

Joshi, N.; Matusik, W.; Avidan, S. Natural video matting using camera arrays. In: *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, New York, NY, USA: ACM Press, p. 779–786, 2006.

Kato, H. Inside artoolkit. Disponível em : <www.hitl.washington.edu/artoolkit/Papers/ART02-Tutorial.pdf> Acesso em: 24 jan. 2007.

KATO, H.; BILLINGHURST, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In: *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, Washington, DC, USA: IEEE Computer Society, p. 85, 1999.

KATO, H.; BILLINGHURST, M.; BLANDING, R.; MAY, R. *Artoolkit pc version 2.11.* 1999.

KIYOKAWA, K.; KURATA, Y.; OHNO, H. An optical see-through display for mutual occlusion of real and virtual environments. *isar*, v. 0, p. 60, 2000.

KIYOKAWA, K.; KURATA, Y.; OHNO, H. An optical see-through display for mutual occlusion with a real-time stereovision system. *Computers & Graphics*, v. 25, n. 5, p. 765–779, 2001.

KONTTINEN, J.; PATTANAIK, S.; HUGHES, C. E. Real-time illumination and shadowing by virtual lights in a mixed reality setting. Disponível em : <http://graphics.cs.ucf.edu/MAR-Sumant/1568936265.pdf> Acesso em: 28 jan. 2007.

LOOSEA, J.; GRASSET, R.; SEICHTER, H.; LAMB, P. Osgart artoolkit for openscenegraph. Disponível em : <http://www.artoolworks.com/community/osgart> Acesso em: 09 fev. 2007.

MACINTYRE, B.; GANDY, M.; DOW, S.; BOLTER, J. D. Dart: a toolkit for rapid design exploration of augmented reality experiences. In: *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, New York, NY, USA: ACM Press, p. 197–206, 2004.

MCGUIRE, M.; MATUSIK, W.; PFISTER, H.; HUGHES, J. F.; DURAND, F. Defocus video matting. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, New York, NY, USA: ACM Press, p. 567–576, 2005.

MILGRAM, P.; TAKEMURA, H.; UTSUMI, A.; KISHIRO, F. A class of displays on the reality-virtuality continuum. In: *Telemanipulator and Telepresence Technologies*, Boston, MA, USA: SPIE, p. 282–292, 1994.

MITSUNAGA, T.; YOKOYAMA, T.; TOTSUKA, T. Autokey: human assisted key extraction. In: *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM Press, p. 265–272, 1995.

NABA KUMAR Anjuta devstudio. Disponível em : <http://anjuta.sourceforge.net> Acesso em: 30 ago. 2007.

OPENCV Open computer vision library. Disponível em : <http://sourceforge.net/projects/opencvlibrary> Acesso em: 05 mar. 2007.

PAULA, L. R. P.; BONINI NETO, R.; MIRANDA, F. R. Camera kombat - interação livre para jogos. Disponível em : <http://camerakombat.googlepages.com> Acesso em: 05 mar. 2007.

PIEKARSKI, W.; AVERY, B.; THOMAS, B. H.; MALBEZIN, P. Integrated head and hand tracking for indoor and outdoor augmented reality. In: *Virtual Reality, Proceedings. IEEE*, p. 11–276, 2004.

PIEKARSKI, W.; THOMAS, B. H. The tinmith system: demonstrating new techniques for mobile augmented reality modelling. In: *AUIC '02: Proceedings of the Third Australasian conference on User interfaces*, Darlinghurst, Australia: Australian Computer Society, p. 61–70, 2002.

PINTARIC, T. An adaptive thresholding algorithm for the augmented reality toolkit. In: *Augmented Reality Toolkit Workshop,IEEE International*, p. 71, 2003.

PIROLLO, H. S. *O estudo e a implementação da técnica de chroma key para utilização com a biblioteca artoolkit.* 2006. 54 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, 2006.

PORTER, T.; DUFF, T. Compositing digital images. In: *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM Press, p. 253–259, 1984.

ROMÃO, T. Realidade aumentada móvel. Disponível em : <http://www.iilab.com/fcsh> Acesso em: 24 jul. 2007.

SCHLOERB, D. W. A quantitative measure of telepresence. *Presence : Teleoperators and Virtual Environments*, v. 4, n. 1, p. 64–80, 1995.

SCHMALSTIEG, D.; FUHRMANN, A.; HESINA, G.; SZALAVARI, Z.; ENCARNACAO, L. M.; GERVAUTZ, M.; PURGATHOFER, W. The studierstube augmented reality project. *Presence: Teleoperators & Virtual Environments*, v. 11, n. 1, p. 33–54, 2002.

SEANET CORPORATION The blue screen page. Disponível em : <http://www.seanet.com/ bradford/bluscrn.html> Acesso em: 05 jan. 2007.

SEGAL, M.; AKELEY, K. The opengl graphics system: A specification version 2.1. Disponível em : <http://www.opengl.org/> Acesso em: 07 fev. 2007.

SEMENTILLE, A. C.; LOURENÇO, L. E.; BREGA, J. R. F.; RODELLO, I. A motion capture system using passive markers. In: *VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, New York, NY, USA: ACM Press, p. 440–447, 2004.

SGI Open inventor. Disponível em : <http://oss.sgi.com/projects/inventor/> Acesso em: 17 jun. 2007.

SIMSARIAN, K.; AKESSON, K.-P. Windows on the world: An example of augmented virtuality. In: *Interfaces 97: Man-Machine Interaction*, p. 68–71, 1997.

SMITH, A. R.; BLINN, J. F. Blue screen matting. In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM Press, p. 259–268, 1996.

STUDIERSTUBE Augmented reality for collaborative and ubiquitous computing. Disponível em : <http://studierstube.icg.tu-graz.ac.at> Acesso em: 02 jan. 2007.

SUN, J.; JIA, J.; TANG, C.-K.; SHUM, H.-Y. Poisson matting. In: *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, New York, NY, USA: ACM Press, p. 315–321, 2004.

THINKQUEST Compositing. Disponível em : <http://library.thinkquest.org/10015/data/info/reference/techniques/compositing.html> Acesso em: 01 jun. 2007.

THOMAS, G. Mixed reality techniques for tv and their application for on-set and pre-visualization in film production. Disponível em : <www.rm.is.ritsumei.ac.jp/MR-PreVizProject/pdf/thomas.pdf> Acesso em: 12 jan. 2006.

THOMAS, G. A.; RUSSEL, R. T. Video processing. Europe Patent EP1499117A1 19 jan. 2005.

UBUNTU . Disponível em : <http://www.ubuntu.com> Acesso em: 01 mai. 2006.

ULTIMATTE CORPORATION Ultimatte feature comparison chart. Disponível em : <http://www.ultimatte.com> Acesso em: 04 jan. 2007.

UNIVERSIDAD POLITECNICA DE VALENCIA Openscenegraph. Disponível em : <http://www.openscenegraph.org/projects/osg> Acesso em: 15 ago. 2007.

VALLINO, J. Introduction to augmented reality. Disponível em : <http://www.se.rit.edu/ jrv/research/ar/introduction.html> Acesso em: 09 dez. 2005.

VLAHOS, P. Composite photography utilizing sodium vapor illumination. United States Patent n. 3,095,304 25 jun. 1963.

VLAHOS, P. Eletronic composite photography. United States Patent n. 3,595,987 27 jul. 1971.

WAGNER, D.; SCHMALSTIEG, D. First steps towards handheld augmented reality. In: *ISWC '03: Proceedings of the 7th IEEE International Symposium on Wearable Computers*, Washington, DC, USA: IEEE Computer Society, p. 127, 2003.

Woo, M.; Neider, J.; Davis, T.; Shreiner, D. *Opengl programming guide: The official guide to learning openg.* 3ª ed. Massashusetts, EUA: Addison-Wesley, 1997.

Yamashita, A.; Kaneko, T.; Matsushita, S.; Miura, K. T. Region extraction with chromakey using striped backgrounds. In: *Proceedings of IAPR Workshop on Machine Vision Applications (MVA2002)*, p. 455–458, 2002.

Zongker, D. E.; Werner, D. M.; Curless, B.; Salesin, D. H. Environment matting and compositing. In: *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., p. 205–214, 1999.