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

HEITOR SPONTON PIROLLO

O ESTUDO E A IMPLEMENTAÇÃO DA TÉCNICA DE CHROMA KEY PARA UTILIZAÇÃO COM A BIBLIOTECA ARTOOLKIT

Marília
Dezembro / 2006

HEITOR SPONTON PIROLLO

O ESTUDO E A IMPLEMENTAÇÃO DA TÉCNICA DE CHROMA KEY PARA UTILIZAÇÃO COM A BIBLIOTECA ARTOOLKIT

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação, do Centro Universitário Eurípides de Marília, mantida pela Fundação de Ensino Eurípides Soares da Rocha, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. Antonio Carlos Sementille.

Marília
Dezembro / 2006

PIROLLO, Sponton Heitor. **O estudo e a implementação da técnica de Chroma Key para utilização com a biblioteca ARToolkit. 2006.** 53 f. **–** Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

RESUMO

Este trabalho apresenta a técnica de Chroma Key, implementa dois algoritmos e faz uma análise destas soluções comparando a eficiência de reconhecimento do fundo. Chroma Key é uma técnica de processamento de imagens cujo objetivo é eliminar o fundo de uma imagem para isolar os personagens ou objetos de interesse que posteriormente são combinados com uma outra imagem de fundo (*background*). Inicialmente, serão implementados protótipos que utilizam a biblioteca ARToolKit para a captura de imagens em tempo real e que aplicam a técnica de Chroma Key nas imagens capturadas, porém cada protótipo utiliza um *background* diferente.

Palavras – Chave: Chroma Key, ARToolKit

PIROLLO, Heitor. **O estudo e a implementação da técnica de Chroma Key para utilização com a biblioteca ARToolkit. 2006.** 53 f. **–** Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

ABSTRACT

This work presents the technique of Chroma Key, implements two algorithms and makes an analysis of these solutions comparing the efficiency of recognition of the background. Chroma Key is one technique of image processing whose objective is to eliminate the background of an image to isolate the personages or objects of interest that later is combined with one another image of background. Initially, archetypes will be implemented that use the ARToolKit library for the capture of images in real time and that they apply the technique of Chroma Key in the captured images, however each archetype will use one background different.

Keywords: Chroma Key, ARToolKit

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ARTOOLKIT - *Augmented Reality ToolKit*

AVI – *Audio Video Interleave*

BGRA – *Blue Green Red Alpha*

GLUT - *Graphics Library Utility*

MB - *MegaBytes*

OPENGL - *Open Graphics Library*

RGB - *Red Green Blue*

SDK – *Software Development Kit*

USB - *Universal Serial Bus*

WMV – *Windows Media Video*

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

O efeito ou técnica de Chroma Key é utilizado em vídeos em que se deseja substituir o *background* por algum outro vídeo ou foto.

Tanto os estúdios de televisão quanto os estúdios de cinema dispõe de recursos para utilização de Chroma Key, porém normalmente fazem uso de hardware especializado, caro e protegido por patentes.

Porém com o advento das câmeras digitais e dos microcomputadores é possível se implementar as técnicas de Chroma Key totalmente em software e a baixo custo.

É neste contexto que este trabalho se insere: o estudo das técnicas de Chroma Key e seus problemas, bem como a implementação das abordagens principais.

Dentre as dificuldades a serem analisadas na técnica do Chroma Key, destaca-se a dificuldade oferecida pelo sistema de cores RGB, pois este sistema de cores não é muito adequado ao sistema de percepção visual humano. Outro ponto importante é a suavização da borda dos objetos ou personagens principais da cena, de forma que se não perca a naturalidade da cena no momento da combinação. Deve-se também, preservar os efeitos "naturais" como sombras, transparências e brilho, para harmonizar a produção final. Por fim, a possibilidade de detectar diferentes cores de fundo (constante) de imagem. Isto tudo, sem prejudicar o desempenho da técnica, para que não prejudique o desenvolvimento da ferramenta que será desenvolvida.

Para a implementação utilizou-se a biblioteca ARToolKit, a qual facilita o desenvolvimento de aplicações em Realidade Aumentada e oferece funções para captura de imagens em tempo real.

1.1 - Motivação

A motivação principal para este trabalho vem das pesquisas recentes relacionadas ao assunto e o interesse na proposta de se desenvolver uma ferramenta em Realidade Aumentada

usando a técnica de Chroma Key para recortar um objeto do mundo virtual e projetar um pouco à frente do mundo dando uma noção de profundidade no mundo virtual.

1.2 - Objetivo

O projeto tem como objetivo implementar uma ferramenta usando a biblioteca ARToolKit que implementará a técnica de Chroma Key em imagens dinâmicas, ou seja, serão feitas trocas de *background* por imagens estáticas e vídeos.

1.3 – Organização do Trabalho

Este trabalho está dividido em 7 Capítulos. No Capítulo 2 são apresentados os conceitos básicos sobre ARToolKit e suas características. O Capítulo 3 apresenta como funciona a técnica de Chroma Key e mostra alguns exemplos. O Capítulo 4 mostra um manual de instalação da biblioteca ARToolKit e alguns testes feitos no início do trabalho. O Capítulo 5 apresenta o contexto, objetivos e estruturação do projeto. O Capítulo 6 apresenta testes e análise de resultados. O Capítulo 7 apresenta as conclusões e trabalhos futuros.

CAPÍTULO 2 – ARTOOLKIT

ARToolKit foi desenvolvido em 1999 quando Hirokazo Kato chegou no HITLab. A primeira demonstração esteve em SIGGRAPH 1999 para o projeto de espaço compartilhado de Realidade Aumentada. A Realidade Aumentada é um subconjunto da Realidade Virtual que trata da modificação do mundo virtual pela sobreposição de objetos virtuais. Para que as imagens do mundo real e virtual possam ser registradas é preciso que a posição e orientação da câmera seja rastreada constantemente. A maioria das aplicações de Realidade Aumentada utiliza técnicas de Visão Computacional para realizar o rastreamento.

O ARToolKit é uma biblioteca de programação para o desenvolvimento de aplicações de Realidade Aumentada. Esta biblioteca utiliza técnicas de visão computacional para calcular precisamente a posição e orientação da câmera relativa a um marcador em tempo-real. O programador pode então usar esta informação para desenhar objetos 3D exatamente alinhados com o objeto real. Desenvolvido pelo HIT Lab e distribuído em forma de código aberto (Open Source).

2.1 - Como funciona o ARToolKit

As aplicações de ARToolKit permitem que a imagem virtual seja vista sobreposta a imagem do mundo real. O segredo está nos quadrados pretos usados como marcadores. O ARToolKit trabalha da seguinte forma:

1 - A câmera captura o vídeo do mundo real e transmite-o ao computador.

2 - O software no computador procurará através de cada quadro de vídeo por todas as formas quadradas.

3 - Se um quadrado for encontrado, o software usa alguma matemática para calcular a posição da câmera relativa ao quadrado preto.

4 - A posição da câmera é conhecida uma vez que um modelo dos gráficos do computador está extraído dessa mesma posição.

5 - Este modelo é extraído no alto do vídeo do mundo real e assim que parece o espaço em branco dentro do marcador quadrado.

6 - A saída final é mostrada, assim quando o usuário olha através da exposição ele vê o objeto no mundo real.

Desta forma é possível resumir os passos de uma maneira mais simples. O programa busca o marcador até achar a posição e a orientação do marcador 3D. Após localizado, o programa identifica o marcador, procura o qual o objeto virtual que está relacionado com aquele tipo de marcador e renderiza o objeto virtual naquele frame do vídeo. A Figura 1 ilustra um pouco mais o funcionamento do ARToolKit.

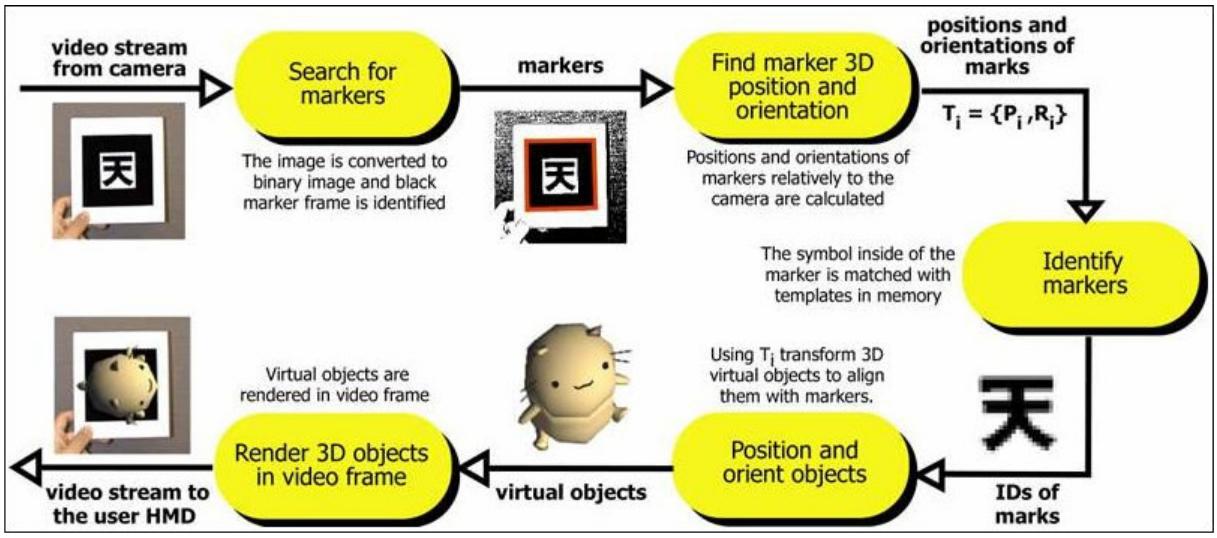

Figura 1 – Funcionamento do ARToolKit (WASHINGTON)

2.2 - Características do ARToolKit

- Uma estrutura simples para criar aplicações de Realidade Aumentada
- Uma biblioteca multiplataforma(Windows, Linux, Mac)
- Cobre os objetos 3D virtuais nos marcadores reais
- Suporta fontes múltiplas de entrada(USB, Firewire)
- Suporta formatos de cores diferentes (RGB, YUV)
- Uma biblioteca gráfica simples (baseada no GLUT)

- Render rápido baseado em OpenGL
- Opensource (código aberto) com licença para uso não comercial.

CAPÍTULO 3 – CHROMA KEY

3.1 - Definição

Chroma Key é uma técnica de processamento de imagens cujo objetivo é eliminar o fundo de uma imagem para isolar os personagens ou objetos de interesse que posteriormente são combinados com uma outra imagem de fundo. Como identificar o fundo da imagem a ser processada e "o quê" e "o quanto" eliminar deste fundo são as maiores dificuldades da técnica (ARAPIS; et all, 1998).

3.2 - Objetivo da técnica

A técnica basicamente tem por objetivo, a partir de uma imagem *foreground* (ou de primeiro plano), buscar os pontos (pixels) que são considerados como do fundo desta imagem - ou então, como não fazendo parte do personagem ou objeto de interesse - e, de certa forma, eliminá-los. O produto deste processo é posteriormente combinado com uma outra imagem de fundo (chamada de *background*), a fim de colocar o personagem como parte natural desta última. A Chroma Key é uma técnica muito utilizada em estúdios de produção de vídeo que, geralmente, utilizam equipamento apropriado para a realização deste processo. Estes equipamentos são usados para fazer a captação de sinais de câmeras (uma para *foreground* e outra para *background*) para posterior combinação em uma mesma cena.

3.3 - Exemplos

Um exemplo bem simples e comum é o das produções de vídeo em telenoticiários e apresentação de boletins meteorológicos. Nestes casos, os personagens são filmados sob um fundo constante (em geral azul, ou às vezes verde) – sendo que os mesmos não podem estar com alguma peça de roupa de algum tom muito próximo ao deste fundo. Posteriormente, ou durante a própria filmagem, é gerado um sinal de câmera que será o fundo da "tela" (por exemplo, um mapa meteorológico), então, os dois sinais (*foreground* e *background*) são combinados para que se produza a cena.

O processo de eliminação do *foreground* da imagem é ilustrado na Figura 2:

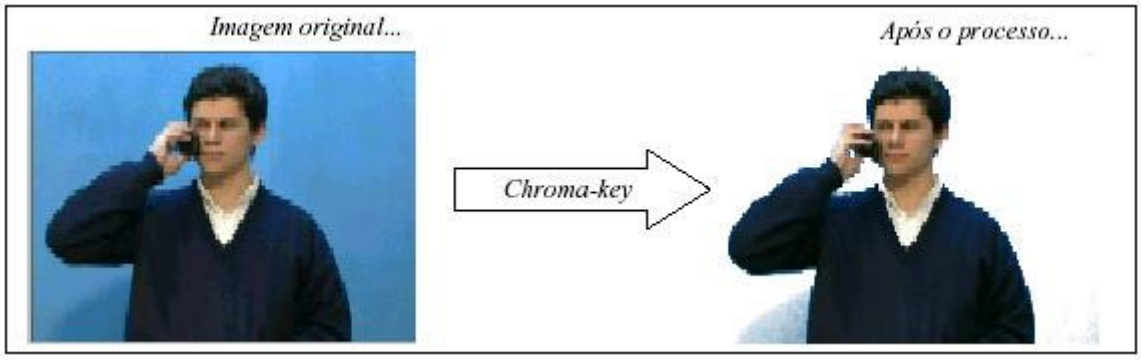

Figura 2 – Processo de eliminação do fundo (TONIETTO, 2000)

O primeiro problema é um tanto óbvio: o fundo deveria ser completamente de uma só cor, portanto de uma cor constante. Isto, porque senão, fica mais difícil distinguir o que é parte do fundo (portanto eliminado) e o que deve sofrer um tratamento (não totalmente eliminado). Um fato intrínseco ao processo de aquisição de imagens é que a cor de fundo nunca é constante na imagem gerada como se nota na Figura 2, devido às variações naturais de iluminação na cena no momento de captura da imagem.

O que se pode fazer é controlar melhor os efeitos produzidos na cena como: posicionamento da luz, reflexo e brilho dos objetos, de forma a diminuir estes problemas. Geralmente usa-se a cor azul como de fundo, num pano, fazendo-se um "fundo infinito" (o pano é colocado de forma que fique na parede e no chão, sendo que a ligação entre os dois, não obedece a um "canto", ou um ângulo de 90 graus, mais uma curva, dando a impressão de que o fundo tende ao infinito), ou então uma tela azul. Opcionalmente, tem-se usado o fundo verde, mas o azul é mais usado porque é dito como complementar para os tons de pele humana.

Outro fato importante é no momento de captação das imagens de foreground e a de *background.* Estas imagens devem ser obtidas observando-se um mesmo posicionamento das câmeras para captação dos sinais, pois qualquer desvio ou movimentação das mesmas, ocorre na perda de noção de espaço na cena. Um outro problema encontrado e de fácil observação, é

o fato de que, se eliminar somente uma cor pura, por exemplo, o azul, as demais cores muito próximas a ela não serão descartadas, porém, visualmente, estas seriam consideras como um azul, deixando a impressão de que não se eliminou o fundo totalmente. Portanto, a "cor-chave", deveria ser substituída por "região-chave num espaço de cores", pois assim, descarta-se completamente o fundo da imagem. Na Figura 3 o exemplo deste processo não bem sucedido:

Figura 3 – Processo de Chroma Key não bem sucedido (TONIETTO, 2000)

Além disso, um outro problema intimamente relacionado ao fato de escolher uma região de cores, é definir neste espaço, quais cores são elimináveis e o quanto "eliminável" é cada cor, e isto implica diretamente no detalhamento ao redor do personagem. Se uma determinada região de cores é considerada descartável completamente, nas partes da imagem onde aparece algum pixel daquela região, este será simplesmente desconsiderado e aparecerá somente o ponto equivalente da imagem de fundo. Sendo assim, as bordas ao redor do objeto ficariam muito rígidas, o que deixa a combinação muito "forçada", não havendo uma harmonização entre a imagem de primeiro plano e a de fundo.

Pode-se citar ainda, na eliminação de um ponto, é que se perderá efeito natural do objeto, tais como: sombra, transparência e movimentação, que não são totalmente da "cor-chave", porém devem também, sofrer um tratamento especial, para que façam parte da combinação final e mantendo também uma boa relação com a imagem de fundo. A Figura 4 demonstra o processo não consistindo uma suavização e nem a transparência da sombra entre a imagem do primeiro plano e a de fundo:

Figura 4 – Suavização e transparência do fundo (TONIETTO, 2000)

Como solução para estes problemas de eliminação, é feito um processo de suavização da transparência. A Figura 5 é o resultado final do processo de Chroma Key feita no software ChromaK-Tchê implementado por Leando Tonietto – Unisinos, Junho de 2000 (TONIETTO,2000).

Figura 5 – Resultado Final (TONIETTO, 2000)

3.4 - Algoritmos

Existem vários algoritmos que implementam a técnica de Chroma Key, a seguir temos a descrição e o funcionamento dos algoritmos estudados nesse trabalho.

3.4.1 – Algoritmo de faixa

Esse algoritmo consiste em verificar se o pixel capturado encontra-se dentro de uma faixa considerada da cor azul, ou seja, são definidos valores máximos e mínimos para cada fator do BGRA (Blue Green Red Alpha) e depois compara se o pixel capturado e verifica se o mesmo encontra-se nessa faixa de valores. A Figura 6 mostra um trecho de código implementado em Linguagem de programação C que mostra como funciona esse algoritmo.

```
int blue_min   = 100;      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;

if ( dataPtr[j] > blue_min && dataPtr[j] < blue_max &&
  dataPtr[j+1] > green_min && dataPtr[j+1] < green_max &&
  dataPtr[j+2] > red_min && dataPtr[j+2] < red_max &&
  dataPtr[j+3] > alpha_min && dataPtr[j+3] < alpha_max )
```

Figura 6 – Trecho de código do algoritmo de faixas

Esse trecho mostra a comparação feita para verificar se o pixel é ou não da cor azul. Obedecendo ao modo como o pixel está organizado, o dataPtr[j] é o valor para o fator B, dataPtr[j+1] é o valor para o fator G, dataPtr[j+2] é o valor para o fator R e dataPtr[j+3] é o valor para o fator A. Caso o pixel seja considerado da cor azul ele é alterado, caso contrário nada é modificado no pixel. A vantagem desse algoritmo é a sua rapidez em relação aos outros algoritmos, pois o custo de processamento é menor do que nos outros dois. A desvantagem é a grande dificuldade de definir uma faixa de azul que isole todos os azuis.

3.4.2 – Algoritmo de Distância Euclidiana

Esse algoritmo é baseado na distância euclidiana, onde estipula-se uma distância mínima que a cor de um pixel deverá estar em relação a posição da cor chave no espaço RGB (cubo RGB) e, no momento da varredura dos pixels da imagem, verifica-se se o pixel está dentro da região das cores eliminávéis (distância entre a cor do pixel e a cor chave é menor que a distância mínima, ou seja considerado como de fundo) ou caso contrário, não tem nenhuma transparência (distância maior ou igual que a distância mínima). Visualmente, a região demarcada por uma distância mínima (chamada de *dmin*) é uma esfera, sendo que o raio é o próprio *dmin*. Na Figura 7 está uma representação desta esfera dentro do no espaço RGB (representado pelo cubo RGB):

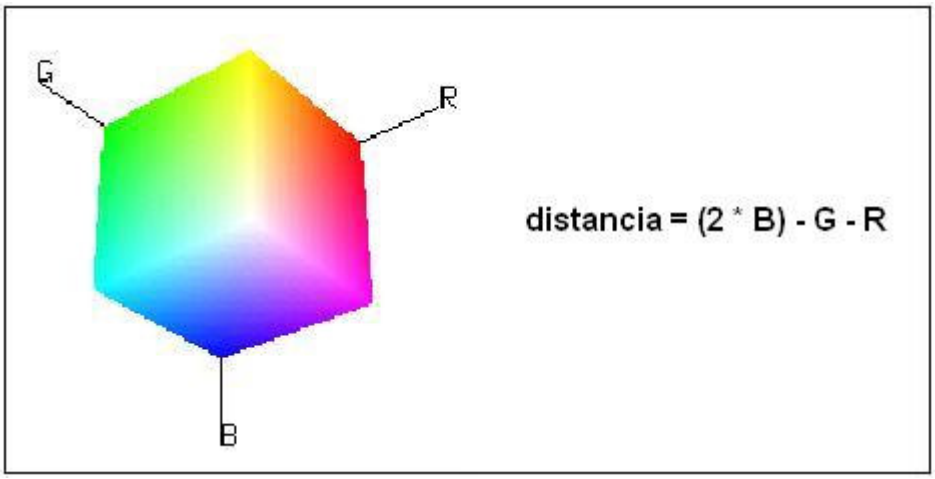

Figura 7 – Cubo RGB

A desvantagem desse algoritmo é o custo de processamento no cálculo da distância do pixel em relação ao canto do cubo. Porém, sua vantagem é a melhor eficiência no reconhecimento dos pixels considerados da cor azul.

3.4.3 – Algoritmo de Bergh e Lalioti

É necessário haver uma maneira confiável de se identificar e isolar o canto azul do cubo, que é, logicamente, onde estão as cores a serem eliminadas ou a sofrerem algum tipo de processamento. Segundo BERGH;LALIOTI, 1999 uma solução para isolamento de um dos cantos do cubo, no caso o azul: fazer com que um plano *S* corte a extremidade azul do mesmo, de forma que, algoritmicamente, as cores acima deste plano são consideradas como parte do fundo (azul) da imagem e o que está abaixo, logicamente, é considerado como um pixel do objeto ou personagem de interesse como é mostrado na Figura 8.

O plano S é paralelo a linha chamada de “linha dos cinzas”, que é a diagonal entre os cantos de cor preta e cor branca. Esta relação existe porque considera-se que todos os pontos deste plano possuem assim, uma distância igualmente azul, isto quer dizer que eles possuem uma distância aproximadamente igual ao cinza perpendicularmente correspondente, ou seja numa intensidade de cor igual. Conforme aparece neste artigo, no momento em que explicam o objetivo deste plano:

"... Isto implica que todos os pontos no polígono cinza-escuro são igualmente 'azul', ou em outras palavras, eles estão numa distância aproximadamente igual a partir de seus valores 'cinzas' com igual intensidade" (BERGH;LALIOTI, 1999).

A Figura 8 foi tirada deste artigo, e exemplifica o plano S:

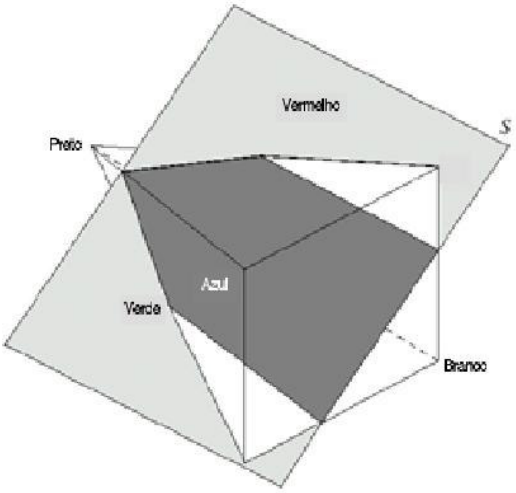

Figura 8 – Cubo RGB cortado pelo plano S para isolar o canto azul

CAPÍTULO 4 – INSTALAÇÃO DA BIBLIOTECA ARTOOLKIT

O primeiro passo é a instalação do Microsoft Directx SDK, que é exigido para o funcionamento do ARToolKit. Logo após é necessário instalar o Microsoft Visual C/C++ 6.0.

Após cumprir os primeiros passos, começa a instalação do ARToolKit. A instalação consiste nos seguintes passos:

1 - Descompactar o arquivo “ARToolKit.zip” no diretório “C:\ARToolKit”

2 - Descompactar o arquivo “DSVideoLib.zip” no diretório “C:\ARToolKit\DSVL”

3 - Copiar o arquivo DSVL.DLL e DSVLd.DLL de “C:\ARToolKit\DSVL\BIN”

para o diretório “C:\ARToolKit\BIN”

4 – Instalar as bibliotecas glut32.dll e glu32.dll no diretório “C:\WINDOWS\system32”

5 – Executar o arquivo ConfigureWin32.bat que está no diretório “C:\ARToolKit”

6 – Abrir o Visual C e construir o programa

No manual de instalação usado, eram somente esses passos. Seguindo corretamente todos os passos a biblioteca não funcionou, pois ainda faltavam vários ajustes nas configurações, dentre eles “setar” as variáveis de ambiente do Windows, adicionar as bibliotecas nos projetos do Visual C(Microsoft Visual C ~ Project ~ Settings ~ Link e incluir as bibliotecas glut32.lib glu32.lib opengl32.lib) e colocar os diretórios dos Includes do Microsoft Directx SDK e do ARToolKit nas opções do Visual C (Microsoft Visual C ~ Tools ~ Options ~ Directories e colocar o SDK em primeiro e o ARToolKit em segundo).

Seguindo os passos corretamente, a biblioteca ARToolKit está pronta para ser usada. O Simpletest.c (programa exemplo da biblioteca) identifica o marcador patthiro e desenha na tela um cubo azul alinhado com o marcador. O código do programa na Linguagem de Programação C encontra-se no Apêndice A.

Após a compilação do programa e a criação do arquivo executável, coloca-se o marcador na frente da webcam e começa os primeiros testes.

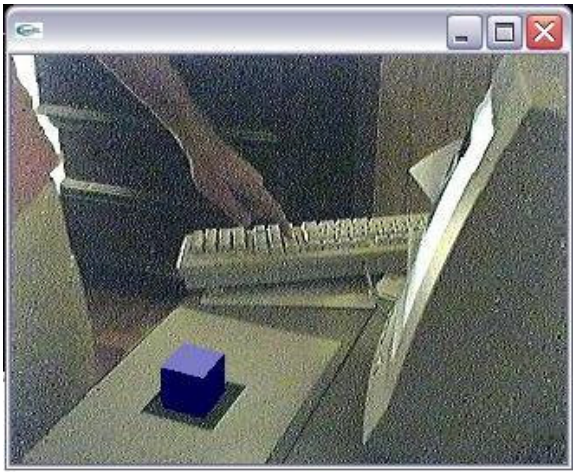

Figura 9 – Webcam posicionada acima do marcador

Na Figura 9, nota-se de cima o cubo azul projetado. Vale ressaltar que o marcador pode estar em qualquer posição, basta que a câmera identifique por completo.

Também deve-se dizer que essas imagens foram obtidas com uma Webcam de pouca qualidade. Na Figura 10 temos uma caneta “atravessando” o cubo azul.

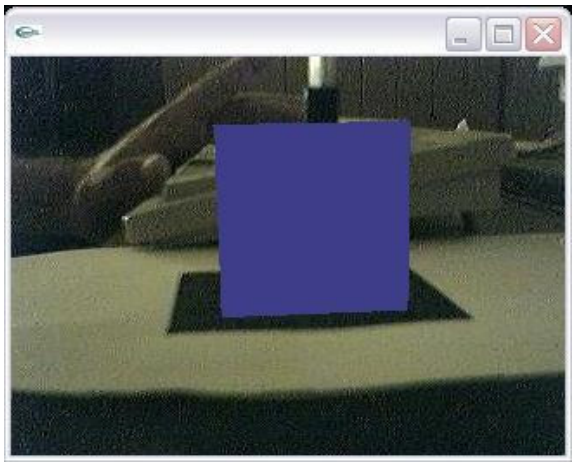

Figura 10 – Objeto real atravessando o objeto virtual

Na parte superior do cubo, dá para ver uma caneta “entrando” no cubo. A imagem não é prejudicada porque a caneta não intercepta nenhum ponto entre o marcador e a webcam.

Na Figura 10, vê se uma imagem lateral comprovando que o marcador pode estar em qualquer posição. Na Figura 11 veremos um teste de distância entre o marcador e a webcam.

Figura 11 – Distância máxima entre o marcador e a webcam

Na Figura 11, o marcador se encontra cerca de 120cm da webcam. Essa foi a distância máxima testada que obteve sucesso, já que a webcam não tem qualidade suficiente para capturar e identificar o marcador de uma distância maior. Futuramente, seria ncessário testar esse programa com uma câmera de maior qualidade, já que não se encontra uma distância máxima estabelecida.

CAPÍTULO 5 – O TRABALHO DESENVOLVIDO

5.1 – Contexto

Atualmente, os recursos computacionais são cada vez mais usados. Na medida em que a tecnologia cresce, os recursos tendem a crescer junto. A técnica do Chroma Key é muito importante e está presente no nosso cotidiano. Presente em Jornais da TV e em filmes, a técnica auxilia cada vez mais e facilita a filmagem de cenas mais complicadas, como por exemplo, um ator qualquer frente a frente com algum animal selvagem. Com o uso do Chroma Key se torna mais fácil, pois não há necessidade de colocar o ator frente a frente com o animal, já que o ator pode ser filmado atuando na cena com um fundo azul (foreground) e depois combinar a cena com a imagem do animal (background) e chegando ao resultado da cena sem correr riscos de pessoas frente aos animais.

Os softwares e os matériais sobre Chroma Key disponíveis, em sua grande maioria estão relacionados com imagens estáticas, ou seja, trabalham com imagens .jpeg ou .bmp, o que torna mais fácil a aplicação da técnica, pois a cor do fundo é linear, facilitando o reconhecimento dos pixels. Há uma escassez de materiais de Chroma Key com captura de imagens em tempo real. As imagens dinâmicas são capturadas por meio de filmadoras, o que torna mais difícil o reconhecimento dos pixels, pois a cor de cada pixel do fundo varia muito dentro de uma faixa de valores, deixando mais complicado o reconhecimento do fundo.

5.2 – Objetivos

- Estudo da técnica de Chroma Key
- Projeto e Implementação dos principais algoritmos
- Testes e análise de desempenho

5.3 – Estruturação do Projeto

O projeto está estruturado da seguinte forma:

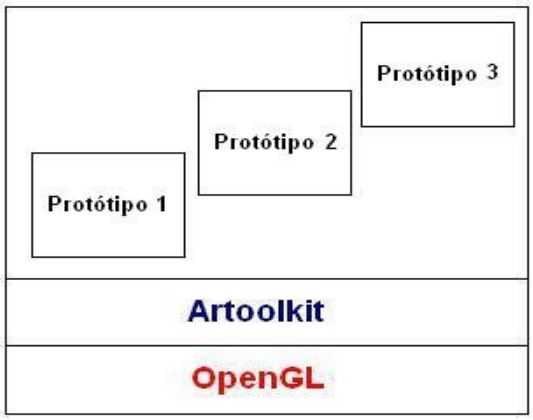

Figura 12 – Estruturação do projeto

Na camada mais baixa, encontra-se o OpenGL, que tem sua funções usadas pela camada superior. A camada ARToolKit utiliza funções OpenGL na implementação de suas bibliotecas. Na camada mais acima, encontram-se os protótipos desenvolvidos ao longo do trabalho. Ambos utilizam um padrão de captura de imagem oferecido pela biblioteca ARToolKit. Esse padrão é o BGRA (*Blue Green Red Alpha*), cada pixel capturado consiste nesse padrão. Os valores variam entre 0 e 255 para cada elemento.

O elemento B(*blue*) contém a quantidade da cor Azul presente no pixel, o G(*green*) contém a quantidade da cor verde presente, o R(*red*) contém a quantidade da cor vermelha presente no pixel. Esse padrão é semelhante ao RGB, porém tem um fator A(*alpha*) que significa o quanto um pixel é transparente. O valor 0 para o A significa que ele é visível, já o valor 255 significa que o pixel é totalmente transparente.

Outro ponto importante é a maneira como o ARToolKit trabalha. A Tabela 1 mostra o funcionamento passo a passo:

Tabela 1 – Passos do funcionamento

Passo do ARToolKit	Função
1. Inicializa a aplicação	init
2. Captura um quadro de vídeo	arVideoGetImage
3. Analisa o fundo	Analyzedeep
4. Mostra a imagem na tela	argDispImage(dataPtr, 0, 0);
5. Fecha o dispositivo de vídeo	cleanup

Basicamente, todos os protótipos utilizam as mesmas funções, mudando somente as estruturas de dados utilizadas e o algoritmo de substituição do fundo. Os passos 2, 3 e 4 são repetidos continuamente até que termine a aplicação.

Quando o programa é executado a função Main() é chamada. Esta função chama uma outra função de inicialização que contém o código para definição do caminho dos parâmetros do dispositivo de vídeo, da câmera e configura a janela gráfica. Depois, a função arVideoCapStart inicia a captura de imagens de vídeo. Finalmente, a função argMainLoop é chamada para iniciar o laço do programa principal, associar a função keyEvent aos eventos do teclado e a função MainLoop com o laço principal da renderização gráfica. Agora, analisaremos as funções de cada passo.

No 1º passo a função chamada é a Init. Ela é chamada da função principal e é usada para definir o caminho dos parâmetros para o dispositivo de vídeo e ler os parâmetros iniciais da aplicação ARToolKit. Os passos 2, 3 e 4 estão contidos na função MainLoop. Esta é a função na qual a maior parte das chamadas de função do ARToolKit é realizada. No passo número 2, a função arVideoGetImage é chamada, essa função captura um quadro de vídeo e o salva num buffer chamado dataPtr. No 3º passo, a função chamada é a *Analyzedeep*, ela analisa o fundo pixel a pixel varrendo o buffer, e troca os pixels considerados da cor azul pelo fundo adequado dependendo de cada protótipo.

O 4º passo consiste na chamada da função argDispImage, essa que por sua vez mostra na tela o frame armazenado no buffer. Vale ressaltar que a função do 3º passo, Analyzedeep, foi desenvolvida durante o trabalho e não faz parte da biblioteca ARToolKit. Outra função que devemos conhecer o seu funcionamento é a keyEvent, ela faz parte da biblioteca ARToolKit, porém para cada protótipo foi alterada conforme o necessário.

Na sua estrutura original, a função keyEvent é acionada quando a tecla ESC é pressionada. Essa função é responsável pelo passo de número 5, ou seja, quando é chamada

ela verifica se a tecla pressionada realmente é o ESC, e se assim for, chama a função cleanup. A função cleanup é chamada para finalizar o processamento de vídeo e desconectar o dispositivo de vídeo, liberando-o para outras aplicações. Isto acontece quando se chama as funções arVideoCapStop**,** arVideoClose e argCleanup.

5.4 - Descrição dos Protótipos

5.4.1 – Protótipo 1

Esse protótipo inicial é o mais simples. Ele foi implementado à partir do simpleTeste.c e consiste apenas em trocar a cor do fundo. O programa inicia com a resolução de 320 x 240, logo após a captura do quadro os pixels são analisados, os que se encaixam na faixa considerável de azul tem o seu fator BGRA alterado para outra cor, ou seja, suponha-se que um *pixel* tenha os valores 200, 50, 20, 0. Usando os algoritmos implementados esse pixel é considerado azul e troca-se o valor para 0, 255, 255, 0. Portanto, o fundo que era Azul agora passa a ser amarelo(Apêndice B).

5.4.2 – Protótipo 2

Esse protótipo utiliza uma estrutura de dados auxiliar que foi chamada de dataPtr2. O dataPtr2 é um *buffer* onde é armazenado um quadro capturado para servir de fundo. Inicia-se o programa, setando resolução de 320 x 240 pixels. Quando o programa estiver executando, posiciona-se a câmera para capturar a imagem de fundo e pressiona uma tecla qualquer chamando a função *KeyEvent*.

Nesse protótipo, dentro da *KeyEvent* foi adicionado um *else* onde o programa verifica se a tecla pressionada não for o ESC ele entra no *else* e captura a imagem que servirá de fundo usando a função *arVideoGetImage* e armazenando a imagem no buffer auxiliar chamado de dataPtr2. Após capturada a imagem de fundo, o programa captura um novo quadro, analisa os pixels que são considerados da cor azul e os troca pelo pixel da imagem que está armazenada no dataPtr2, lembrando que é trocado pelo pixel de mesma posição, ou seja, o programa troca

o fundo que anteriormente era azul por uma imagem capturada e escolhida pelo usuário no início do programa(Apêndice C).

5.4.3 – Protótipo 3

Esse protótipo tem por objetivo trocar o fundo por um vídeo. O vídeo que servirá como fundo para o programa é previamente capturado usando um programa de captura salvando-o em um arquivo binário. É necessário uma estrutura de dados para armazenar os frames lidos do arquivo e que posteriormente será usada para substituir o fundo. Na figura 13 segue o trecho de código que define a estrutura usada.

```
#DEFINE altura 240
#DEFINE comprimento 320
typedef struct {
        ARUint8 pixel[4];
}pixel;
typedef struct{
        pixel buffer[comprimento * altura];
} rec;
rec r;
```

Figura 13 – Trecho de código que define a estrutura de dados

Esse trecho de código nada mais é que um vetor de n posições (0...n-1) chamado de buffer, onde n é o número de pixels contidos no buffer, que contem em cada posição um vetor de 4 posições (0...3) chamado pixel. O tipo ARUint8 é definido pela biblioteca ARToolKit. O ARUint8 é um inteiro de 8 bits que armazena para cada fator do BGRA o valor entre 0 e 255. A Figura 14 mostra como está organizada essa estrutura de dados.

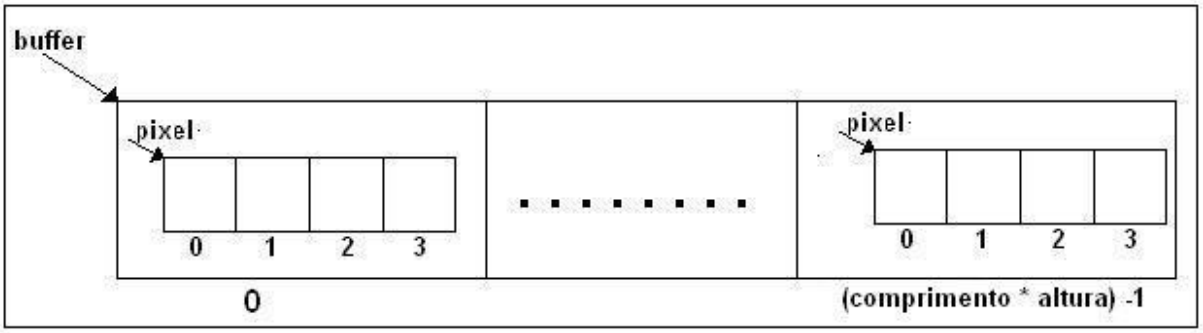

Figura 14 – Estrutura de dados

Com o arquivo binário contendo o vídeo que servirá de fundo, executa-se o protótipo. Inicia-se com a resolução 320 x 240. Com o programa rodando, pressiona-se qualquer tecla, quando a tecla é pressionada a função *KeyEvent* é chamada. A adaptação feita nesse protótipo consiste em verificar a tecla pressionada, se esta for diferente de ESC (finaliza a aplicação) o programa entra em um *else*, seta um Flag e abre o arquivo binário que contem o vídeo capturado anteriormente. O programa volta a sua função principal e tendo o flag igual a 1 (verdadeiro) o programa começa a ler os quadros do arquivo e trocar o fundo pelo vídeo que está salvo no arquivo(Apêndice D).

5.5 – Funcionamento dos protótipos

Basicamente, todos protótipos funcionam da mesma maneira. Na Figura 15 segue um fluxograma que mostra o funcionamento.

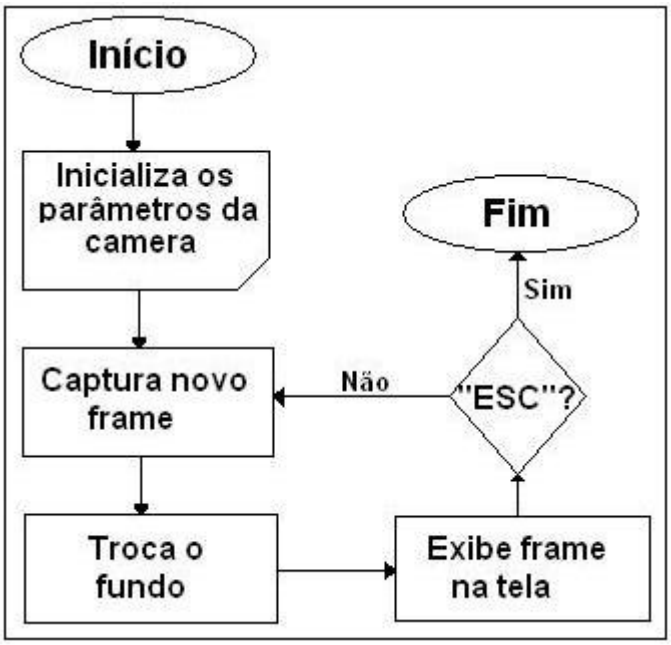

Figura 15 – Fluxograma de funcionamento dos protótipos

O fluxograma acima mostra como os protótipos se comportam. No primeiro passo os parâmetros da câmera são inicializados. Logo após, começa um "*loop*" onde o quadro é capturado, tratado e exibido na tela, ou seja, após a captura do quadro ele é tratado pixel a pixel para verificar se pertence ou não a área da cor azul no cubo RGB. Depois de feita a troca

dos pixels da cor azul, o frame é exibido na tela. A tecla ESC, quando pressionada, aciona a função keyEvent, esta que por sua vez verifica qual tecla foi pressionada, se for a tecla ESC a aplicação é finalizada.

CAPÍTULO 6 – TESTES E ANÁLISE DE RESULTADOS

6.1 – Ambiente experimental

6.1.1 – Hardware

- Processador Intel Pentium 4 2.4GHz
- 512 MB de memória RAM
- Placa de vídeo Geforce4 MX 440 with AGP8X
- Vimicro USB PC Câmera

6.1.2 – Software

- Windows XP Professional Versão 2002 (Service Pack 2)
- Microsoft Visual C++ 6.0
- ARToolKit 2.71.2

Figura 16 – ambiente experimental

6.2 – Testes

6.2.1 – Quadros por segundo

O número de quadros por segundo capturado é diferente a cada vez que o programa é executado. Com base em várias execuções foram tiradas médias. O protótipo 1 na resolução 320 por 240 foi o que obteve o melhor resultado. Foram capturados 12,5 quadros por segundo, já que é o mais simples e não tem um custo elevado de processamento. Já na

resolução 640 x 480, o protótipo 1 obteve um resultado de 11,5 quadros por segundo. Uma queda relativamente baixa se comparada aos outros protótipos.

O protótipo 2, por ter um buffer adicional, e na função de substituição do fundo receber um valor que está gravado na memória e não uma constante, a taxa de quadros por segundo foi de 9,8 na resolução 320 x 240 pixels. Na resolução 640 x 480 esse número baixou para 6,2 quadros por segundo.

O protótipo 3 busca o background salvo em um arquivo no disco. Por ter esse custo adicional de processamento, o número de quadros por segundo baixou em relação ao protótipo 2. O número capturado na resolução 320 x 240 foi de 7,4 frames por segundo. Usando a resolução 640 x 480 pixels, o protótipo 3 caiu 44% a taxa, obtendo 4,2 quadros por segundo.

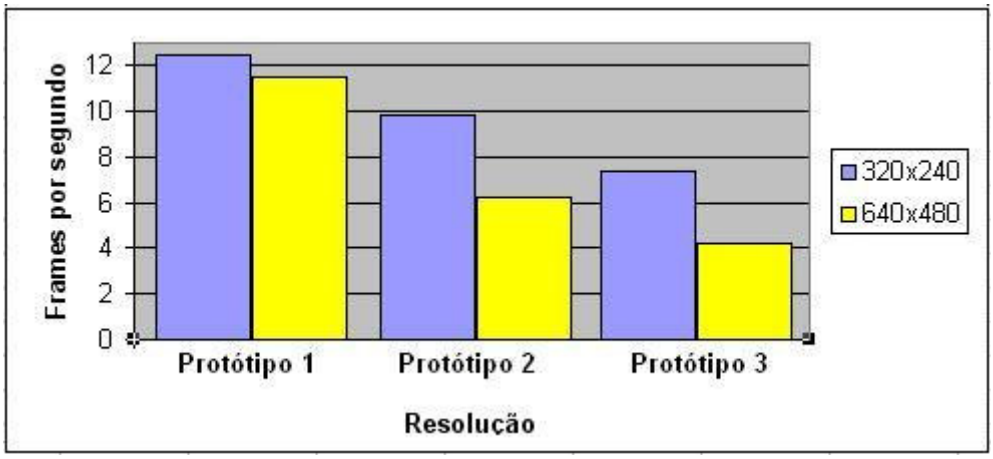

Figura 17 – Gráfico de comparação de quadros por segundo

6.2.2 – Eficiência do reconhecimento do fundo

Foram usados dois algoritmos para reconhecimento do fundo. No primeiro, foi testada uma faixa de valores máximos e mínimos para cada fator do pixel e o resultado obtido não foi satisfatório. Com o intuito de obter um melhor resultado, foi estudado e implementado outro algoritmo. A distância Euclidiana é baseada no Cubo RGB, é calculada a distância em que o pixel se encontra do canto do cubo. Esse algoritmo apresentou um resultado melhor em

relação ao primeiro algoritmo usado. As Figuras 19 e 20 demonstram o resultado obtido com cada um dos algoritmos utilizando o Protótipo 1.

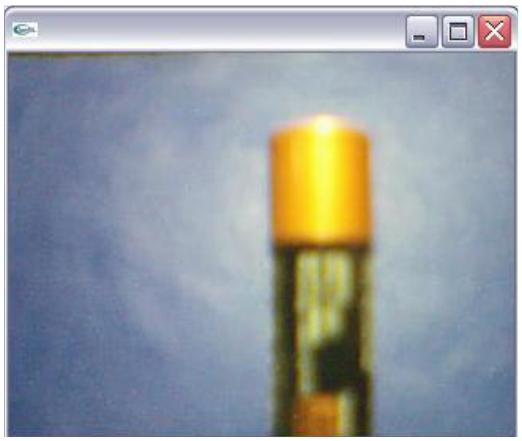

Figura 18 – Imagem Inicial Capturada

A Figura 18 mostra a imagem capturada, sem qualquer alteração. A Figura 19 mostra o resultado obtido pelo protótipo 1, usando o algoritmo de faixas.

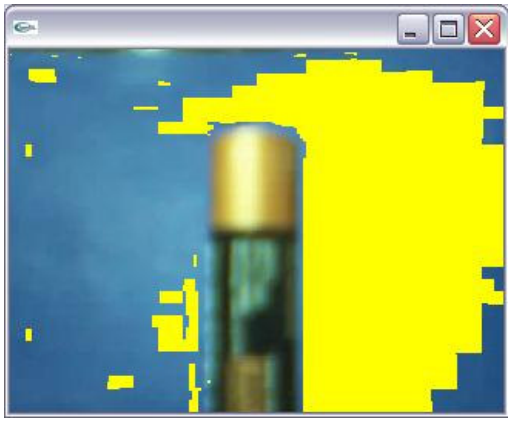

Figura 19 – Algoritmo de Faixas no protótipo 1

A Figura 19 exibe o resultado desse algoritmo isolando uma média de 49% dos pixels considerados da cor azul.

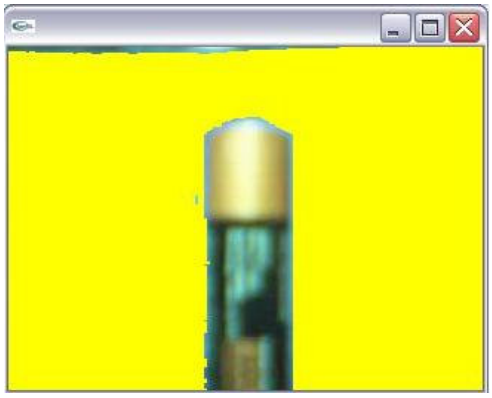

Figura 20 – Algoritmo de Distância Euclidiana no protótipo 1

No segundo algoritmo, o resultado apresentado foi satisfatório. Nota-se que em média 98% dos pixels foram considerados azuis e, conseqüentemente, trocados para amarelo.

Contudo, nota-se que no segundo algoritmo duplicou o número de pixels considerados da cor azul, demonstrando sua maior eficiência em relação ao primeiro algoritmo. Nas Figuras 21 e 22 temos os resultados apresentados pelos dois algoritmos utilizando o Protótipo 2.

Figura 21 – Algoritmo de Faixas no protótipo 2

Figura 22 – Algoritmo de Distância Euclidiana no protótipo 2

Os resultados apresentados pelo protótipo 2 foram semelhantes aos resultados do Protótipo 1, tanto utilizando o algoritmo de faixas quanto utilizando o algoritmo de Distância Euclidiana. Nas Figuras 23 e 24 veremos os resultados obtidos pelo protótipo 3.

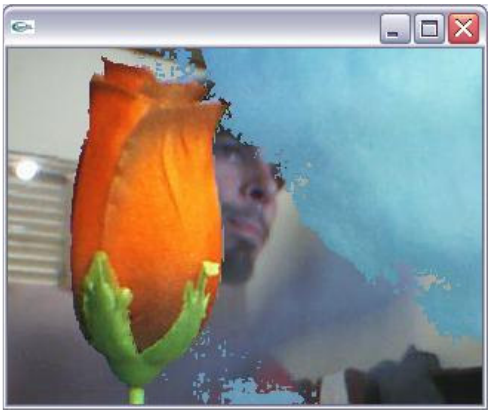

Figura 23 – Algoritmo de Faixas no protótipo 3

Na Figura 24 temos uma seqüência de figuras que representa o resultado obtido pelo protótipo 3, já que nesse protótipo o background é um vídeo.

Figura 24 – Algoritmo de Distância Euclidiana no protótipo 3

Em relação ao reconhecimento dos pixels, o protótipo 3 apresentou o mesmo resultado obtido pelos outros protótipos. Na Figura 25 temos um teste feito com a utilização de marcador.

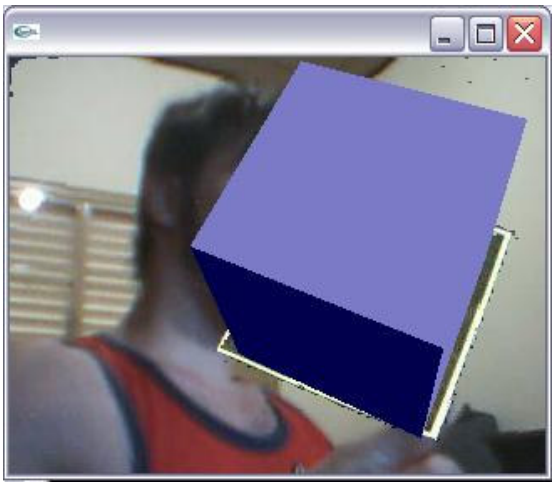

Figura 25– Distância Euclidiana utilizando marcador

CAPÍTULO 7 – CONCLUSÕES E TRABALHOS FUTUROS

A principal conclusão deste trabalho é que o algoritmo de Distância Euclidiana produziu resultados com qualidade superior ao algoritmo de faixas. A principal desvantagem, contudo, é o seu maior tempo de processamento devido ao cálculo da Distância de cada pixel.

Outro ponto importante são as condições de teste, já que a iluminação influencia demasiadamente nos valores de cada fator BGRA (*Blue Green Red Alpha*), ou seja, se a iluminação for pouca o pixel tende para o lado do preto e se a iluminação for muita o pixel tende mais para cor branca, modificando assim a verdadeira cor do pixel.

Conclui-se também, que os protótipos implementados em Linguagem de Programação C necessitam de aperfeiçoamento para reconhecer 100% dos pixels azuis.

Propõe-se em trabalhos futuros, criar novos protótipos que usem vídeos no formato .AVI ou .WMV como backgrounds, incluir novos algoritmos de análise de fundo bem como criar uma função que faça a suavização dos pixels do contorno do objeto e o tratamento da sombra do mesmo para que não haja um "serrilhado" no resultado.

Desta forma, os trabalhos futuros podem ser resumidos em:

- Estudar e implementar novos algoritmos: procurar algoritmos que obtenham resultado com qualidade igual ou superior aos algoritmos apresentados nesse trabalho. Importante enfatizar que o algoritmo busque sempre um melhor reconhecimento do fundo e também um melhor custo de processamento.

- Acrescentar um algoritmo de Anti-Aliasing: estudar e implementar um algoritmo que possibilite esta técnica, suavizando o contorno dos objetos adicionando pontos de cores intermediarias e evitando o aparecimento do "serrilhado" na volta do objeto, tornando o recorte perfeito.

- Estudar os formatos de vídeo .AVI e .WMV: possibilitar a criação de novos protótipos que utilizem como background um vídeo nos formatos indicados, permitindo que vídeos de alta qualidade sejam usados de fundo.

- Captura de fundo feita por uma segunda câmera (opcional): estudar a criação de um protótipo que utilize duas câmeras, ou seja, a primeira câmera captura a imagem foreground e a segunda câmera captura o background. O protótipo então implementa a técnica de Chroma Key combinando a imagem de câmeras diferentes.

- Integrar protótipos: seria interessante integrar os protótipos em uma única aplicação, ou seja, com a aplicação aberta, o usuário escolhe qual o protótipo a ser usado e qual o algoritmo desejado. Possibilitando também uma troca de algoritmos no decorrer da execução do programa.

REFERÊNCIAS

BERGH, F. van den; LALIOTI, V. Software Chroma Keying in na Immersive Virtual Environment. **South African Computer Journal**. Nº 24, nov. 1999, pág. 155-162.

ARAPIS, Costas; BREITENEDER, Christian; GIBBS, Simon; LALIOTI, Vali; OSTAFAWY, Sina; SPEIER, Josef. **Virtual Studios: Na Overview**. IEEE Multimedia. p. 18-36, January-March 1998.

TONIETTO, Leandro, Análise de algoritmos para Chroma Key, junho 2000, pág 09-42.

WASHINGTON, Disponível em www.hitl.washington.edu/artoolkit, acessado em 16/08/2006

APÊNDICE A – SIMPLETEST

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#ifndef __APPLE__
#include <GL/gl.h>
#include <GL/glut.h>
#else
#include <OpenGL/gl.h>
#include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>

#ifdef _WIN32
char                    *vconf = "Data\\WDM_camera_flipV.xml";
#else
char                    *vconf = "";
#endif

int             xsize, ysize;
int             thresh = 100;
int             count = 0;

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

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

static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);
static void   draw( void );

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

   arVideoCapStart();
```

```
    argMainLoop( NULL, keyEvent, mainLoop );
        return (0);
}

static void   keyEvent( unsigned char key, int x, int y)
{
    /* quit if the ESC key is pressed */
    if( key == 0x1b ) {
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
        cleanup();
        exit(0);
    }
}

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

    /* grab a vide frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
        arUtilSleep(2);
        return;
    }
    if( count == 0 ) arUtilTimerReset();
    count++;

    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );

    /* detect the markers in the video frame */
    if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
        cleanup();
        exit(0);
    }

    arVideoCapNext();

    /* check for object visibility */
    k = -1;
    for( j = 0; j < marker_num; j++ ) {
        if( patt_id == marker_info[j].id ) {
            if( k == -1 ) k = j;
            else if( marker_info[k].cf < marker_info[j].cf ) k = j;
        }
    }
    if( k == -1 ) {
```

```
        argSwapBuffers();
        return;
    }

    /* get the transformation between the marker and the real camera */
    arGetTransMat(&marker_info[k], patt_center, patt_width, patt_trans);

    draw();

    argSwapBuffers();
}

static void init( void )
{
    ARParam  wparam;

    /* open the video path */
    if( arVideoOpen( vconf ) < 0 ) exit(0);
    /* find the size of the window */
    if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
    printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

    /* set the initial camera parameters */
    if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
        printf("Camera parameter load error !!\n");
        exit(0);
    }
    arParamChangeSize( &wparam, xsize, ysize, &cparam );
    arInitCparam( &cparam );
    printf("*** Camera Parameter ***\n");
    arParamDisp( &cparam );

    if( (patt_id=arLoadPatt(patt_name)) < 0 ) {
        printf("pattern load error !!\n");
        exit(0);
    }
    /* open the graphics window */
    argInit( &cparam, 1.0, 0, 0, 0, 0 );
}

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

static void draw( void )
{
```

```
    double    gl_para[16];
    GLfloat   mat_ambient[]     = {0.0, 0.0, 1.0, 1.0};
    GLfloat   mat_flash[]       = {0.0, 0.0, 1.0, 1.0};
    GLfloat   mat_flash_shiny[] = {50.0};
    GLfloat   light_position[]  = {100.0,-200.0,200.0,0.0};
    GLfloat   ambi[]            = {0.1, 0.1, 0.1, 0.1};
    GLfloat   lightZeroColor[]  = {0.9, 0.9, 0.9, 0.1};

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

    /* load the camera transformation matrix */
    argConvGlpara(patt_trans, gl_para);
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd( gl_para );
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambi);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_flash);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_flash_shiny);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMatrixMode(GL_MODELVIEW);
    glTranslatef( 0.0, 0.0, 25.0 );
    glutSolidCube(50.0);
    glDisable( GL_LIGHTING );

    glDisable( GL_DEPTH_TEST );
}
```

APÊNDICE B – PROTÓTIPO 1

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#ifndef __APPLE__
#include <GL/gl.h>
#include <GL/glut.h>
#else
#include <OpenGL/gl.h>
#include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>

/* set up the video format globals */

#ifdef _WIN32
char                    *vconf = "Data\\WDM_camera_flipV.xml";
#else
char                    *vconf = "";
#endif

int             xsize, ysize;
int             thresh = 100;
int             count = 0;

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

static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);

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

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

static void   keyEvent( unsigned char key, int x, int y)
```

```
{
   /* quit if the ESC key is pressed */
   if( key == 0x1b ) {
      printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
      cleanup();
      exit(0);
   }
}

/* main loop */
static void mainLoop(void)
{
   ARUint8         *dataPtr;
   int             j, k;

   /* grab a vide frame */
   if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
      arUtilSleep(2);
      return;
   }
   if( count == 0 ) arUtilTimerReset();
   count++;

   argDrawMode2D();

         for(j=0;j<307200;j+=4)//307200 é o tamanho do buffer 320 * 240
                        {
                                 k = (2 * dataPtr[j]) - dataPtr[j+2] - dataPtr[j+1];
                                    if (k > 150)
                                    {
                                             dataPtr[j]=0;
                                             dataPtr[j+1]=255;
                                             dataPtr[j+2]=255;
                                    }
                        }

   argDispImage( dataPtr, 0,0 );

   arVideoCapNext();

   argSwapBuffers();
}

static void init( void )
{
   ARParam  wparam;

   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
```

```
    if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
    printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

    /* set the initial camera parameters */
    if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
        printf("Camera parameter load error !!\n");
        exit(0);
    }
    arParamChangeSize( &wparam, xsize, ysize, &cparam );
    arInitCparam( &cparam );
    printf("*** Camera Parameter ***\n");
    arParamDisp( &cparam );

     /* open the graphics window */
    argInit( &cparam, 1.0, 0, 0, 0, 0 );
}

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

APÊNDICE C – PROTÓTIPO 2

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#ifndef __APPLE__
#include <GL/gl.h>
#include <GL/glut.h>
#else
#include <OpenGL/gl.h>
#include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>

/* set up the video format globals */
#ifdef _WIN32
char                    *vconf = "Data\\WDM_camera_flipV.xml";
#else
char                    *vconf = "";
#endif

int             xsize, ysize;
int             thresh = 100;
int             count = 0;
ARUint8         *dataPtr, *dataPtr2;

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

static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);

int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        init();
   arVideoCapStart();
        argMainLoop( NULL, keyEvent, mainLoop );
        return (0);
}

static void   keyEvent( unsigned char key, int x, int y)
{
```

```
    /* quit if the ESC key is pressed */
    if( key == 0x1b ) {
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
      cleanup();
                system("pause");
      exit(0);
        }
        else
        {
                do{
                        dataPtr2 = (ARUint8 *)arVideoGetImage();
                }while(dataPtr2 == NULL);
        }
}

/* main loop */
static void mainLoop(void)
{
    int           j=0,k=0;

        /* grab a vide frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
                arUtilSleep(2);
      return;
    }

        if( count == 0 ) arUtilTimerReset();
    count++;

        argDrawMode2D();

        if( dataPtr2 != NULL){
                for(j=0;j<307200;j+=4)
                        {
                                k = (2 * dataPtr[j]) - dataPtr[j+2] - dataPtr[j+1];
                                if (k > 80)
                                {
                                        dataPtr[j]= dataPtr2[j];
                                        dataPtr[j+1]= dataPtr2[j+1];
                                        dataPtr[j+2]= dataPtr2[j+2];
                                        dataPtr[j+3]= dataPtr2[j+3];
                                }
                        }
        }

    argDispImage( dataPtr, 0,0 );

    arVideoCapNext();

    argSwapBuffers();
```

```
}

static void init( void )
{
   ARParam  wparam;

   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
   if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
   printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

   /* set the initial camera parameters */
   if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
      printf("Camera parameter load error !!\n");
      exit(0);
   }
   arParamChangeSize( &wparam, xsize, ysize, &cparam );
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );

   /* open the graphics window */
   argInit( &cparam, 1.0, 0, 0, 0, 0 );
}

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

APÊNDICE D – PROTÓTIPO 3

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#ifndef __APPLE__
#include <GL/gl.h>
#include <GL/glut.h>
#else
#include <OpenGL/gl.h>
#include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>

/* set up the video format globals */
#ifdef _WIN32
char                    *vconf = "Data\\WDM_camera_flipV.xml";
#else
char                    *vconf = "";
#endif

int             xsize, ysize;
int             thresh = 100;
int             count = 0,  flag=0;
ARUint8         *dataPtr;

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

typedef struct
{
        ARUint8 pixel[4];
}pixel;

typedef struct
{
        pixel buffer[76800]; // comprimento * altura 320 x 240
} rec;
rec r;
FILE *fptr;

static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);
```

```
static void   draw( void );

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

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

static void   keyEvent( unsigned char key, int x, int y)
{
   /* quit if the ESC key is pressed */
   if( key == 0x1b ) {
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
      cleanup();
                exit(0);
        }
        else
        {
                flag=1;
                fptr=fopen("video_fundo.dat","rb");
                if (!fptr)
                        exit(0);
        }
}

static void Analyzedeep()
{
        int i=0, j, k;
                        if(feof(fptr))
                        {
                                fseek(fptr,0,0);
                                fread(&r,sizeof(rec),1,fptr);
                        }
                        else
                        {
                                fread(&r,sizeof(rec),1,fptr);
                        }
                        for(j=0;j<76800;j++)
                        {
                                k = (2 * dataPtr[i]) - dataPtr[i+2] - dataPtr[i+1];
                                if (k>80)
                                {
                                        dataPtr[i]   = r.buffer[j].pixel[0];
                                        dataPtr[i+1] = r.buffer[j].pixel[1];
                                        dataPtr[i+2] = r.buffer[j].pixel[2];
                                        dataPtr[i+3] = r.buffer[j].pixel[3];
```

```
                                }
                        i+=4;
                }
}

/* main loop */
static void mainLoop(void)
{
   int           j=0,k=0;

        /* grab a vide frame */
   if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
                arUtilSleep(2);
      return;
   }

        if( count == 0 ) arUtilTimerReset();
   count++;

        argDrawMode2D();

        if(flag==1)
                Analyzedeep();

        argDispImage( dataPtr, 0,0 );

        arVideoCapNext();

   argSwapBuffers();
}

static void init( void )
{
   ARParam  wparam;

   /* open the video path */
   if( arVideoOpen( vconf ) < 0 ) exit(0);
   /* find the size of the window */
   if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
   printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

   /* set the initial camera parameters */
   if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
      printf("Camera parameter load error !!\n");
      exit(0);
   }
   arParamChangeSize( &wparam, xsize, ysize, &cparam );
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );
```

```
    /* open the graphics window */
    argInit( &cparam, 1.0, 0, 0, 0, 0 );
}

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