JEFERSON JOSE DE MIRANDA

MOVE-IN: UMA API PARA INTERATIVIDADE COM WEBCAM

Joinville – SC

2008

UNIVERSIDADE DO ESTADO DE SANTA CATARINA

CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT

DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO

JEFERSON JOSE DE MIRANDA

MOVE-IN: UMA API PARA INTERATIVIDADE COM WEBCAM

Trabalho de Conclusão de Curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Marcelo da Silva Hounsell, PHD.

Joinville – SC

2008

JEFERSON JOSE DE MIRANDA

MOVE-IN: UMA API PARA INTERATIVIDADE COM WEBCAM

Este Trabalho de Conclusão de Curso (TCC-I) foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação Integral do CCT/UDESC.

Banca Examinadora

Orientador: _____

Marcelo da Silva Hounsell, PHD.

Co-orientador: _____

Alexandre Gonçalves Silva, M. Sc.

Membro : _____

Avanilde Kemczinski, Dra.

Membro : _____

Isabela Gasparini, M. Sc.

Joinville, ___ de _____ de 2008.

RESUMO

Este trabalho facilita o desenvolvimento de aplicações onde a comunicação com o *software* se dá com a movimentação do usuário capturada por uma *webcam*, sem o uso de marcadores. As imagens capturadas pela câmera são transformadas em entrada de dados que irão determinar a interação, em substituição aos dispositivos convencionais como *mouse* e teclado. O objetivo foi a criação de uma *Application Programming Interface* – API, isto é, um conjunto de funções ou métodos que venham a facilitar a implementação de aplicações que realizam este tipo de interação. Foram utilizados dispositivos (câmeras) de baixo custo, de forma a estender as possibilidades de uso das aplicações que se destinam ao usuário comum. Foi realizada uma pesquisa na literatura relacionada, tanto em trabalhos acadêmicos quanto em trabalhos comerciais, para levantar as funcionalidades necessárias para a biblioteca. A API foi intitulada Move-In, devido ao fato de que a interatividade proposta está baseada na movimentação do corpo do usuário. Esta API possibilita a identificação da presença e movimento de um usuário frente a uma *webcam*, a interação com objetos virtuais e a identificação de determinadas posturas das mãos ou do corpo humano. Exemplos de uso da API, recomendações de *design* para aplicações que derivam dela e a forma como a API Move-In deverá ser estendida para agregar novas funcionalidades também são detalhados no texto. Os exemplos apresentados demonstram que a API facilita e simplifica o desenvolvimento de aplicações de interação com o corpo humano capturado por uma *webcam*.

Palavras-chave: visão computacional, captura de vídeo, captura de movimentos.

ABSTRACT

This work facilitates the development of applications where the communication with the software is given by the movement of the user captured with a webcam, without the use of markers. The images captured by the camera are converted into input data that will determine the interaction, replacing the conventional devices such as mouse and keyboard. The objective was the creation of an Application Programming Interface – API, i.e., a set of functions or methods that facilitate the implementation of this kind of interaction. Low cost devices (cameras) were used in order to extend possibilities of use in applications that are intended for the common user. A literature review was carried out, both in academic work, as in commercial work, to raise the necessary features for the library. The API was entitled Move-In, due that fact that the interactivity proposed is based on user's body movement. This API makes possible the identification of the presence and movement of a user in front of the webcam, the interaction with virtual objects and the identification of certain postures of the human hand or body. Examples of use of the API, design recommendations to applications that are derived from it, and the way that the Move-In API can be extended to aggregate new functionalities are detailed in the text. The presented examples show that the API facilitates and simplifies the development of applications of interaction with human body captured by a webcam.

Keywords*: computer vision, video capture, motion capture.*

LISTA DE FIGURAS

LISTA DE QUADROS E TABELAS

LISTA DE ABREVIATURAS

2D	Bidimensional, ou contendo duas dimensões.
3D	Tridimensional, ou contendo três dimensões.
API	*Application Programming Interface.*
CCD	*Charge-Coupled Device.*
CM-PM	Com marcadores, presença e/ou movimento.
CM-PC	Com marcadores, partes do corpo.
CMOS	*Complementary Metal-Oxide-Semiconductor.*
CMY	*Cyan, Magenta, Yellow.*
FPS	*Frames Per Second.*
GHz	*Giga Hertz.*
GB	*Giga Byte.*
HSL	*Hue, Saturation, Lightness.*
HSI	*Hue, Saturation, Intensity.*
I420	Espaço de cor YUV, com *8 bits* para *Y*, *2 bits* para *U* e *2 bits* para *V*.
Mbps	*Mega bits per second.*
MB	*Mega Byte(s).*
MHz	*Mega Hertz.*
NTSC	*National Television System Committee.*
PAL	*Phase Alterning Line.*
PC	*Personal Computer.*
RAM	*Random Access Memory.*
SECAM	*Séquentiel Couleur à Mémoire.*
SM-PM	Sem marcadores, presença e/ou movimento.
SM-PC	Sem marcadores, partes do corpo.
RGB	*Red, Green, Blue.*
ROI	*Region of Interest.*
YCbCr	Espaço de cor utilizado em fotografias digitais.
YIQ	Espaço de cor utilizado na transmissão do sinal NTSC.
YUV	Espaço de cor codificado em termos de luminância e crominância.

SUMÁRIO

1. INTRODUÇÃO

O uso de interfaces guiadas por visão computacional está se tornando mais viável à medida que o custo das *webcams* diminui e o poder de processamento dos computadores aumenta (ZIVKOVIC 2004:1) (D'HOGE e GOLDSMITH 2004:1). Esse tipo de interface faz uso de, no mínimo, uma câmera de vídeo para fazer a captura de quadros do ambiente analisado. Esses quadros são então trabalhados para que se possa extrair deles informações relevantes à aplicação, como a localização do usuário e até a leitura de certas posturas ou gestos.

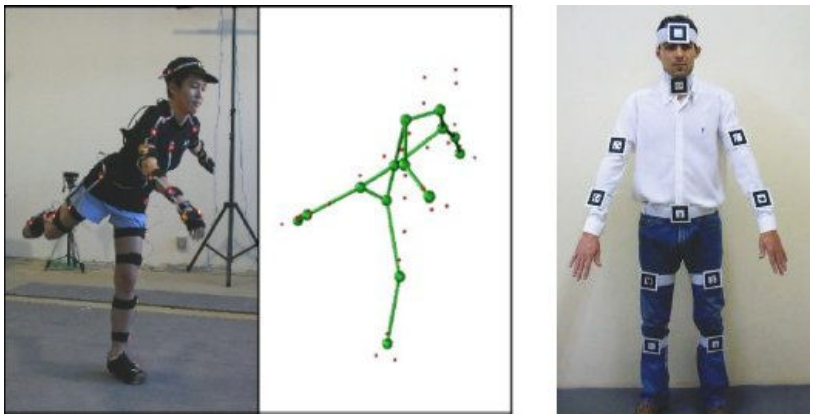
Durante a animação de personagens virtuais, especialmente na indústria do cinema e de jogos eletrônicos, é utilizada uma técnica conhecida popularmente como *motion capture.* Nessas aplicações comerciais, o sistema óptico, com marcadores passivos, é utilizado na maior parte das vezes, mas existem diversos outros tipos de marcadores (DA SILVA, 1997). Esta técnica de captura movimentos[1], rastreia os movimentos de um ator com base em marcadores localizados em pontos específicos do corpo (extremidades e regiões articuladas).

Os marcadores consistem de objetos previamente definidos pelo programador que deverão ser identificados pelos algoritmos de visão computacional, objetivando levantar dados de posição e orientação. A segmentação dos marcadores é feita a partir das características do objeto que os compõem, como cores, figuras desenhadas em objeto plano (HITLAB, 1999) ou formas geométricas, como quadrados, cubos ou esferas (STATE et al, 1996).

Algoritmos de visão computacional segmentam (isolam) os marcadores do resto da imagem capturada pela câmera e a imagem resultante serve de base para a determinação da posição do usuário. Para evitar o problema de oclusão, onde um ou mais marcadores assumem uma posição fora do campo de visão de uma câmera, diversas câmeras podem vir a ser utilizadas. O uso de mais de uma câmera também serve para recuperar a posição dos marcadores em três dimensões (KIRK et al, 2005:1).

[1] Neste trabalho, o termo "técnicas de captura de movimentos" se refere ao conjunto: algoritmos de processamento de imagens e *hardware* necessários para a execução da aplicação final. O hardware necessário inclui o *setup* do usuário e ambiente, como roupas especiais ou necessidade de fundo monocromático, por exemplo.

A biblioteca de código aberto, ARToolkit (HITLAB, 1999), que faz uso de marcadores fiduciais[1], não poderia deixar e ser mencionada, devido à grande quantidade de trabalhos acadêmicos já desenvolvidos. Esta biblioteca foi criada para facilitar o desenvolvimento de aplicações em Realidade Aumentada, mas também pode ser usada para estimar a posição de um usuário, tal como na técnica de *motion capture* com marcadores que é empregada comercialmente.



(A) **(B)**

Figura 1 – Marcadores de um sistema comercial e seu equivalente com a biblioteca ARToolkit.
(KIRK et al, 2005:1) (SEMENTILLE et al, 2004:2)

A Figura 1.A mostra os marcadores de *optical motion capture*[2], onde um usuário veste uma roupa especial com marcadores passivos reflexivos (KIRK et al, 2005:1). A Figura 1.B mostra o que seria equivalente a esta técnica utilizando os marcadores da biblioteca ARToolkit (SEMENTILLE et al, 2004:2). Pode-se observar a partir da Figura 1.B, que esta alternativa tem como uma de suas desvantagens, a fragilidade dos marcadores, que geralmente são impressos em papel. Há alternativas de baixo custo para esta técnica (STAPLES e DAVIS. 2006), porém, o tipo de marcador não é realmente o maior empecilho para larga utilização, mas sim o *setup* do conjunto de marcadores, que devem ser corretamente posicionados ao longo do corpo e o conseqüente possível desconforto sentido pelo usuário.

Existem situações onde o uso de marcadores artificiais, como os da Figura 1, não é uma opção viável, no sentido de que a necessidade de posicionar

[1] Neste caso, formas geométricas em preto e branco impressas em superfície rígida e quadricular.
[2] Existem diversos outros tipos, inclusive não envolvendo câmeras, mas que fogem do escopo deste trabalho.

corretamente um conjunto de marcadores ao longo do corpo pode vir a diminuir a aceitação e larga utilização do produto. Na área de jogos, onde o usuário da aplicação quer ter um momento de diversão, sem ter que estar preocupado com a localização e uso correto de objetos de captura, há soluções, de aplicação comercial, que são livres do uso de marcadores. Nesta área, pode-se citar a IntelPlay Me2Cam (D'HOGE e GOLDSMITH, 2001), Vivid Group GX System (VIVID GROUP, 1996) e Sony Eye Toy (SCEE, 2003). A Figura 2 ilustra aplicações desses sistemas.

(A) **(B)** **(C)**

Figura 2 – Exemplo de aplicações da IntelPlay Me2Cam, Vivid Group GX System e Sony Eye Toy.
(D'HOGE e GOLDSMITH, 2001:2), (KIZONY et al, 2002:3), (SCEE, 2003)

A identificação de presença ou de movimento[1] pode ser efetuada sem o uso de marcadores. Também é possível rastrear "marcadores naturais", como a face ou as mãos do usuário para identificar partes específicas do corpo, e é possível trabalhar com a silhueta do usuário extraída das imagens capturadas para a identificação de alguns gestos[2].

Ambas das abordagens, com e sem marcadores, trabalham com segmentação de imagens, que é a identificação das partes importantes, relevantes à aplicação, onde a primeira precisa localizar e segmentar marcadores e a segunda precisa efetuar a segmentação do usuário[3]. As aplicações sem uso de marcadores não requerem o uso de dispositivos especiais, mas podem requerer uma configuração de cenário especial (como fundo de cor única), ou, no mínimo, exigir

[1] Considerar "presença" quando o sistema percebe o usuário parado, e "movimento" quando o sistema percebe a movimentação efetuada e a localização desta.
[2] Considerar um gesto como sendo uma pose específica ou uma seqüência específica de movimentos.
[3] Neste caso, pode-se pensar no usuário como sendo um "marcador natural".

que o usuário use roupas com cores diferentes das cores do cenário de fundo e que o ambiente onde será efetuada a interação esteja bem iluminado.

A grande vantagem das técnicas que utilizam marcadores é a maior eficiência de processamento para a recuperação de posicionamento em três dimensões. Existem implementações (HORAIN e BOMB, 2002) que objetivam realizar esta ação, utilizando uma única câmera, sem marcadores. Porém, este tipo de trabalho exige uma complexidade muito alta de algoritmos de visão computacional para extração da posição 3D, necessitando para isso de *hardware* especial[1], e, ainda assim, não são robustas e possuem muitas das restrições das técnicas de recuperação de movimentos 2D ilustradas na Figura 2.

Já a grande desvantagem do uso de marcadores artificiais está na conseqüência do uso desses marcadores. É difícil evitar a oclusão utilizando uma única câmera, considerando aplicações que exigem movimentos constantes e variados, além do fato de que sua utilização leva ao uso de dispositivos fixados ao corpo, e, portanto, tende a ser incômoda para o usuário. Para cada um dos casos existem bibliotecas de código aberto que implementam as funcionalidades básicas. Como jogos e aplicações lúdicas estão incluídos entre os usos futuros do resultado deste trabalho, a segunda alternativa (sem marcadores) foi adotada.

Para tanto, foi feito um levantamento de aplicações relacionadas à captura de movimentos do corpo, com o intuito de identificar e até reproduzir algumas dessas técnicas. O objetivo está em identificar as vantagens e desvantagens de cada técnica, para poder criar exemplos práticos que poderão servir de base para aplicações posteriores.

O *hardware* utilizado como base de implementação foi uma *webcam* de resolução QVGA (*320x240*) com taxa de captura de quadros a *30* FPS e um computador *desktop 1.8* GHz, *512* Mb de memória RAM e 128 Mb para placa de vídeo. Maiores detalhes sobre o *hardware* e *software* utilizados são discutidos no capítulo 4.

[1] Câmeras e placas de vídeo de alto desempenho.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver uma biblioteca de funções que faça uso de técnicas de visão computacional, que venha a facilitar o desenvolvimento de interfaces que utilizam movimentos do usuário, capturados por uma *webcam*, como entrada de dados.

1.1.2 Objetivos Específicos

Os objetivos a serem alcançados no decorrer do trabalho são os seguintes:

- Levantar as técnicas que permitem a identificação eficiente (em tempo real) de movimentos do usuário;
- Implementar as técnicas de captura de movimento;
- Identificar as limitações das técnicas empregadas, estabelecer comparações entre as funcionalidades implementadas (condições de uso);
- Fornecer, ao desenvolvedor de interfaces ou ambientes virtuais, um conjunto de funções que venha a facilitar a tarefa de captura de movimentos.

1.2 METODOLOGIA

A seguinte metodologia foi adotada durante o desenvolvimento deste trabalho: primeiramente foi realizada uma pesquisa na literatura com o objetivo de levantar as funcionalidades que deveriam fazer parte da API. Essas funcionalidades foram baseadas nas soluções empregadas nos trabalhos da revisão literária. Esta revisão também levantou a fundamentação teória necessária para entendimento das características de *hardware* (*webcams*) e *software* (APIs auxiliares) utilizados.

A arquitetura da API Move-In (resultado deste trabalho) foi desenvolvida a partir deste levantamento. Para familiarizar o usuário da API com as técnicas de interação com webcam, foram criados exemplos de forma a expor as situações onde as principais funcionalidades devem ser utilizadas.

1.3 ESTRUTURA DO TRABALHO

Este trabalho está estruturado da seguinte forma: o primeiro capítulo faz a introdução e pontua os objetivos. O capítulo 2 aborda conceitos necessários para entendimento dos próximos capítulos e também apresenta a teoria de visão computacional relacionada à identificação dos objetos de interesse e os movimentos desses objetos.

O capítulo 3 expõe os trabalhos levantados que servem de base para o projeto do conjunto de funções de captura de movimentos. O capítulo 4 mostra detalhes do *hardware* e *software* utilizados na implementação dos resultados, e o capítulo 5 apresenta a arquitetura da API e as hipóteses levantadas a partir do estudo realizado na literatura relaciona, tais como a funcionalidade de identificação de partes específicas do corpo e das mãos, e formas de interação com objetos virtuais.

O capítulo 6 expõe conclusões extraídas a partir da implementação das propostas realizadas no capítulo 5 e também faz considerações sobre boas práticas de desenvolvimento com relação ao *design* das aplicações, além de indicar a forma como a API poderá ser expandida, caso novas funcionalidades venham a ser identificadas no futuro. O capítulo 7 apresenta as considerações finais e norteia futuras possíveis melhorias a serem desenvolvidas a partir da API Move-In.

2. VISÃO COMPUTACIONAL APLICADA À CAPTURA DE MOVIMENTOS

Este capítulo tem o objetivo de estabelecer a fundamentação teórica que serve de base para o entendimento do restante deste trabalaho. Para tanto, a seção 2.1 apresenta a classificação das técnicas de captura de movimentos. A seção 2.2 faz definições importantes sobre análise de imagens que são referenciadas do decorrer deste trabalho. A seção 2.3 apresenta a definição de visão computacional segundo diferentes visões. Por fim, as seções 2.4 e 2.5 iniciam a relacão de conceitos importantes que são necessários para o usuário da API, o desenvolvedor de aplicações, desenvolvida neste trabalho.

2.1 CLASSIFICAÇÃO DAS TÉCNICAS DE CAPTURA DE MOVIMENTOS

As interfaces guiadas por visão computacional, isto é, aplicações que utilizam câmeras para realizar a captura de movimentos, dentro do contexto deste trabalho, podem ser classificadas, conforme esquema da Tabela 1, quanto ao uso de marcadores – tipo de detecção – e quanto à funcionalidade da captura de movimentos – tipo de identificação.

Esta definição foi criada no presente trabalho, como o propósito de esclarecer o escopo de desenvolvimento da biblioteca de funções. Na Tabela 1, os termos SM e CM se referem, respectivamente, às técnicas “sem marcadores” e “com marcadores”. Os termos PM e PC estão relacionados ao propósito das técnicas, cujos significados são: identificação de “presença e/ou movimento” e de “partes do corpo”.

Considerando as aplicações abordadas por este trabalho, durante o processo de captura de movimentos, pode ser necessário identificar simplesmente a presença ou movimentação do usuário em uma determinada parte da área de focada pela câmera para disparar uma ação. Ou ainda, pode haver a necessidade de identificar partes específicas do corpo do usuário.

Por exemplo, em uma aplicação de “goleiro virtual” onde um usuário deve impedir que bolas de futebol passem pelo gol, basta calcular a colisão da bola virtual com o jogador (defesa) ou falta de colisão (gol), isto é, basta verificar se o usuário encontra-se na posição *(x, y)* durante o momento *t*. Considerando a classificação da

Tabela 1, esta situação poderia ser implementada com técnicas do tipo 1 (CM-PM) ou 3 (SM-PM), se for considerado que tal aplicação não necessita determinar qual parte do corpo do jogador defendeu a bola.

Tabela 1 – Classificação de Técnicas de Captura de Movimentos.

		Tipo de Identificação	
		Presença e/ou Movimento	Partes do Corpo
Tipo de Detecção	Com Marcadores	(1) CM-PM	(2) CM-PC
	Sem Marcadores	(3) SM-PM	(4) SM-PC

Já uma aplicação da área de fisioterapia, onde um membro específico do corpo humano precisa ser trabalhado, como os braços, por exemplo, pode ocorrer que seja necessário saber a posição específica de certas partes, como mãos, punhos, cotovelos e ombros. O rastreamento de partes específicas do corpo pode vir a possibilitar o diagnóstico do problema e o acompanhamento da evolução do tratamento.

Este tipo de rastreamento pode ser resolvido com marcadores do tipo 2 (CM-PC), já que pode se associar um marcador específico para cada parte do corpo. Pode haver ambigüidade, caso os marcadores não sejam distintos (tenham o mesmo formato). Isso requer algoritmos específicos para determinar a qual parte do corpo cada marcador está associado. Além disso, este tipo de configuração pode ser inconveniente para o usuário[1].

Também é possível rastrear “marcadores naturais”, como a face ou as mãos do usuário para identificar partes específicas do corpo, e é possível trabalhar com a silhueta do usuário extraída das imagens capturadas para identificar partes específicas do corpo. A API desenvolvida neste trabalho trata das técnicas dos tipos 3 e 4 da tabela 1. A utilização de marcadores não será abordada no desenvolvimento deste trabalho, devido ao problemas que foram citados no capítulo 1, como oclusão com uma única câmera e possível desconforto que será sentido por parte do usuário. Ainda assim, há a possibilidade de que a biblioteca de funções (API Move-In) seja extendida para este fim em trabalhos futuros.

[1] Configurações que envolvem uso de acessórios acoplados ao corpo ou até uma roupa especial.

2.2 ANÁLISE DE IMAGENS

A implementação de interfaces guiadas por visão computacional envolve a aquisição de imagens, a separação da área de interesse do plano de fundo e o processamento das imagens relativas à área de interesse. Na Figura 3 (UMBAUGH, 1997:38), a aquisição de imagens é descrita como imagem de entrada. O pré-processamento está relacionado à remoção de qualquer informação irrelevante para a extração de características, que é uma sub-etapa realizada durante a redução de dados. Esta etapa tem como como objetivo segmentar a área de interece, que contém características relevantes à aplicação. Essas características devem ser identificadas, também durante a redução de dados, para prover dados de entrada destinados à próxima etapa. Finalizando o processo de análise de imagens, análise de características determina como as características relevantes para a aplicação deverão ser interpretadas.

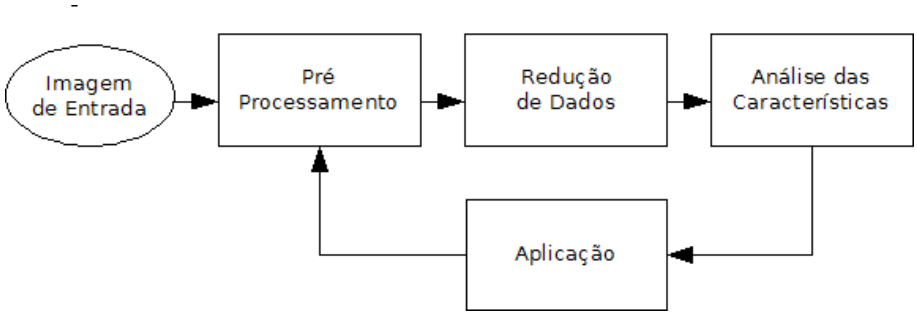


Figura 3 – Análise de Imagens.
(UMBAUGH, 1997:38)

Para exemplificar este processo dentro de uma aplicação de captura de movimentos, considera-se um botão virtual que deve ser acionado pelo toque do usuário. Neste caso, a imagem de entrada será um quadro de animação capturado a partir de uma *webcam*. Geralmente, imagens provenientes de uma webcam são capturadas no espaço de cor RGB (maiores explicações sobre espaço de cores se enconram na seção 2.4.3). Se for conveniente para a aplicação que esta imagem seja convertida para outro espaço de cor, este processo será a etapa de pré-processamento.

A seguir, a mesma imagem, inicialmente colorida, é convertida para duas cores: branco, representando a silhueta que corresponde ao usuário que está

interagindo com a aplicação e preto, representado o cenário de fundo e quaisquer objetos que não sejam o usuário (o objeto de interesse) em questão. Este processo, chamado de binarização de imagem, corresponde à etapa de redução de dados. Assim, a eliminação de dados irrelevantes (como as cores) acabou resultando na separação do usuário do cenário de fundo.

O botão virtual da aplicação deste exemplo deve ser “tocado” para ser acionado. O toque não é físico, mas é considerado quando o usuário, na imagem bidimensional, se encontra na mesma posição que o botão. Isto é, a característica a ser extraída é a posição do usuário em relação ao botão. Para analisar esta característica, ou seja, determinar se o toque ocorreu, verifica-se se a imagem resultante está branca (posição ocupada pelo usuário) dentro da área ocupada pelo botão.

Por fim, para evitar que o botão seja acionado acidentalmente, pode-se programar a aplicação para acioná-lo somente somente depois de capturar uma certa quantidade de quadros seguidos contendo a região ocupada pelo botão com parte da silhueta do usuário. Após o acionamento do botão, a aplicação pode então requisitar outra finalidade para o conjunto de etapas de pré-processamento, redução de dados e análise de características.

O processo de análise de imagens envolve uma série de algoritmos de visão computacional. A seção 2.3 apresenta a definição deste termo. A seção 2.4 introduz conceitos sobre as imagens de entrada que se fazem necessários para o desenvolvedor de aplicações que deseja trabalhar com a API implementada neste trabalho. Uma grande atenção é dada neste capítulo à imagem de entrada, pelo fato que desta questão não ser abordada com profuncidade nos trabalhos relacionados no capítulo 3. A seção 2.5 mostra algoritmos que correspondem às etapas de pré-processamento e redução de dados. A análise das características, etapa que depende do propósito de cada aplicação, deverá ser exemplificada nos trabalhos descritos no capítulo 3 e nos exemplos levantados no 5 e discutidos no capítulo 6.

2.3 VISÃO COMPUTACIONAL

De acordo com (UMBAUGH, 1997:3), o trabalho de interpretação das imagens em um sistema computacional pode ser classificado quanto ao fim a que

destina, isto é, quanto ao receptor da informação visual. Quando a informação processada tem como finalidade o consumo humano, a categoria a qual esta faz parte é a de processamento de imagens. Já quando se destina ao computador, a categoria é a de visão computacional.

Essas categorias não são totalmente distintas. A fronteira que as define não está bem definida, mas sua separação permite o estudo de suas diferenças e de como elas trabalham juntas. Por exemplo, o último autor citado afirma que, historicamente, o campo de processamento de imagens cresceu a partir da engenharia elétrica enquanto que a ciência da computação foi largamente responsável pelo desenvolvimento da visão computacional.

A visão computacional é a categoria onde a aplicação não necessita de uma pessoa no controle do *loop* no item “Aplicação” da Figura 3. As grandes questões dentro da visão computacional envolvem a análise de imagens para solucionar um problema de visão, substituindo o olho humano em processos demorados e repetitivos. Exemplos desse tipo de implementação incluem o controle de qualidade realizado de forma automática em sistemas de manufatura, o reconhecimento de digitais e a análise de ADN (Ácido Desoxirribonucléico).

Já na categoria de processamento de imagens, os maiores tópicos por ela compreendidos incluem a restauração de imagens, o melhoramento das imagens (aumento da qualidade visual) e a compressão de imagens. Nota-se que este campo requer o entendimento de como funciona o sistema de visão humano, já que o resultado se destina a um observador humano, ao contrário da visão computacional, onde os dados de saída se destinam à uma análise que será efetuada pelo computador.

Para (Gonzalez, 2000:1), o processamento digital de imagens surge a partir de duas grandes áreas de aplicação: (1) aprimoramento de informações na imagem para interpretação pelo olho humano e (2) processamento de imagens para armazenamento, transmissão e representação para percepção automática por máquinas autônomas. Isto é, existem diferenças na nomenclatura utilizada quando em comparação com (UMBAUGH, 1997), mas a idéia básica de imagem processada para fins de análise pelo olho humano (1) ou para uma máquina autônoma (2) permanece inalterada.

Jahne (2002:18) afirma que a diferença entre a visão humana e a visão computacional está no fato de que a visão humana é extremamente poderosa no reconhecimento de objetos, mas não não é tão precisa quanto a visão computacional na mensuração de distâncias e áreas. Este autor ainda afirma que importantes avanços na visão computacional foram realizados graças ao entendimento do sistema visual humano como, por exemplo, determinação de movimento por técnicas de filtragem.

Sebe et al (2005), considerando o termo visão computacional voltado ao aprendizado de máquinas, afirmam que o objetivo da visão computacional é prover aos computadores as capacidades de percepção humanas para que estes possam sentir o ambiente, compreender os dados sentidos, tomar uma ação apropriada e até aprender com as experiências para melhorar o desempenho de suas tarefas futuras. Esta versão do termo visão computacional, que dá aos computadores um conjunto de características humanas, possibilitanto o aprendizado, está fora do escopo deste trabalho. A API almejada por este trabalho objetiva a criação de aplicações de visão computacional no sentido de que estas serão destinas à interpretação automática de movimentos e posições de um usuário, a partir de uma câmera.

Até este ponto, este capítulo introduziu o processo de análise de imagens e estabeleceu a área de aplicação deste trabalho. A seção a seguir está relacionada a detalhes de aquisição da entrada de dados. Serão abordadas as principais característica relacionadas ao *hardware* responsável por fornecer as imagens de entrada e alguns conceitos de representação dessas imagens.

2.4 IMAGEM DE ENTRADA

As aplicações guiadas por visão computacional discutidas neste trabalho recebem imagens como entrada de dados. Essas imagens são obtidas a partir de câmeras digitais. A especificação técnica da câmera tem grande influência no que se pode esperar no resultado final da aplicação. As subseções a seguir têm o objetivo de apresentar as principais características que irão limitar, ou facilitar, a implementação das aplicações.

2.4.1 Imagem Digital

As imagens digitais são constituídas de unidades de cor chamadas de *pixels*, nomenclatura derivada de *picture element*. Uma imagem digital possui uma resolução, que também pode ser chamada de discretização espacial. A resolução de uma imagem é expressada por:

resolução = quantidade de linhas x *quantidade de colunas* (eq. 1)

Atualmente, as resoluções mais comuns de computadores *desktop* de tela no formato *4x3* estão entre *1024x768* e *1280x1024*. Os televisores convencionais (de baixa definição) possuem uma resolução máxima que varia de *640x480* a *1024x768*.

Alguns formatos de resolução são referenciados por siglas, principalmente em manuais de usuário de *webcams* ou dispositivos móveis acoplados de câmeras (como celulares ou câmeras digitais). A Tabela 2 (ICAMERAPHONES, 2005) mostra algumas das principais resoluções encontradas em dispositivos de baixo custo. Uma tabela com maiores resoluções podem ser encontra em (WOOTTON, 2005:116).

Tabela 2 – Resoluções comuns em dispositivos móveis e *webcams*.

(ICAMERAPHONES, 2005)

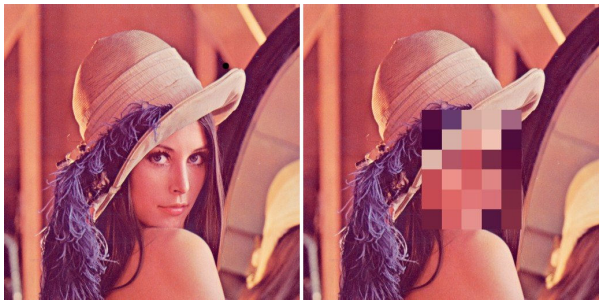
Sigla	Nome	Resolução	Megapixels
SXGA	Super Extended Graphics Array	1280x1024	1.3
SVGA	Super VGA.	800x600	0.5
VGA	Video Graphics Array	640x480	0.3
QVGA	Quarter VGA	320x240	0.07
CIF	Common Intermediate Format	352x288	0.1
16CIF	16 vezes CIF	1408x1152	1.6
4CIF	4 vezes CIF	704x576	0.4
QCIF	Quarter CIF	176x144	0.02
Sub-QCIF	Sub Quarter CIF	128x96	0.01

O termo *Megapixels* se refere à quantidade de linhas multiplicada pela quantidade de colunas em uma imagem, isto é, é a quantidade aproximada de

pixels, em milhões de unidades. A resolução utilizada no desenvolvimento deste trabalho está compreendida nesta tabela.

Quanto maior a resolução, mais nítida será a imagem. O efeito causado por imagens de baixa resolução, onde se percebe o tamanho e o formado quadriculado do *pixel* na imagem é conhecido como serrilhamento (*aliasing*) ou pixelização (*pixelization*). Uma resolução muito baixa pode tornar a imagem irreconhecível. Não raramente, a pixelização é utilizada para proporcionar o anonimato de uma pessoa em um vídeo (Figura 4.B).

Na Figura 4[1] se vê, respectivamente, uma imagem (original, para fins de comparação) seguida da mesma imagem com a região da face pixelizada. Não é difícil concluir que a diminuição de escala de resolução causa perda de informação. Por exemplo, na região da face da Figura 4.B, a imagem foi diminuída para uma resolução bastante baixa (5x6) e depois aumentada para o tamanho original.



(A) **(B)**

Figura 4 – Pixelização ou Serrilhamento.

Os *pixels* de uma imagem digital são acessados através de uma função *f(x, y)*, considerando *x* como sendo uma das duas dimensões, largura ou altura, e *y* como sendo a outra dimensão. Assim como o mapeamento das coordenadas *x* e *y* nas dimensões citadas, também a origem do plano cartesiano, *f(0, 0)*, depende da aplicação, podendo ser um dos quatro cantos da imagem. A origem do sistema de coordenadas na API desenvolvida neste trabalho está por padrão no canto superior

[1] A Figura 4.A é frequentemente utilizada em demonstrações de processamento de imagens, sem apresentar a fonte de referência. Ela pode ser encontrada em motores de busca como o *Google Images*.

esquerdo da imagem, tal como é feito na própria API OpenCV*,* biblioteca de funções em linguagem C na qual a implementação deste trabalho está baseada.

Uma imagem pode ser colorida, monocromática (tonalidades de cinza) ou binária. A imagem monocromática se refere à função bidimensional de intensidade da luz, onde o valor de *f(x, y)* é proporcional ao brilho (ou nível de cinza) da imagem naquele ponto (GONZALEZ e WOODS, 2000:4). Essas imagens necessitam de somente um canal de representação, isto é, uma única matriz de inteiros cujos valores representam a intensidade de iluminação do *pixel*, no caso das imagens em níveis de cinza, ou uma das cores, preto ou branco, nas imagens binárias.

Essa diferença entre a imagem binária e a imagem em tonalidades de cinza é chamada de profundidade de cor. As imagens binárias têm profundidade de um único bit, que assume o valor *0* para preto e *1* para branco. As imagens em tons de cinza geralmente apresentam uma profundidade de *8 bits*, com valores variando de *0* a *255*.

As imagens coloridas são representas por três canais, ou matrizes. Cada canal representa uma cor primária ou uma intensidade de iluminação. Esses valores são combinados para produzir a imagem colorida. Pode também ocorrer a existência de um quarto canal, chamado de canal *alpha*, que indica o nível transparência, variando a opacidade das cores na apresentação da imagem.

A intensidade de uma cor é definida por uma certa quantidade de *bits*, e pode variar de *0* a 2^n*-1*. Por exemplo, uma imagem de *32 bits* e quatro camadas contém quatro octetos (*4*8 bits*) que representam valores inteiros que variam de *0* a *255*.

Esta seção explicou como as imagens são discretizadas em termos espaciais, introduziu os conceitos de *pixel*, camadas e intensidades de cor. Para produzir um vídeo, deve-se também considerar a discretização ao longo do tempo. A discretização temporal será apresentada na seção a seguir.

2.4.2 Vídeo Digital

Vídeo digital é uma representação de uma cena natural discretizada em amostragens espaciais e temporais (RICHARDSON, 2003:9). A amostragem espacial é conhecida como quadro, ou *frame*. O processo de captura da cena natural é repetida várias vezes durante um segundo, em intervalos que podem ser

de *1/15*, *1/20*, *1/60*, *1/120* até intervalos menores ou iguais a *1/10.000* segundos (VICON APPLICATIONS, 2007). A Figura 5 (RICHARDSON, 2003:10) ilustra a relação entre amostragem espacial e temporal.

A quantidade de quadros capturados durante um segundo geralmente é referenciada como FPS (*frames per second*). A resolução e a quantidade de quadros de animação capturados pela câmera são dois dos principais fatores relacionados à sua qualidade. O preço varia bastante com pequenos incrementos desses dois fatores. Durante a elaboração deste trabalho, *webcams* de baixo custo, a uma resolução de QVGA (*320x240*) permitiam a captura de *15* a *30* quadros ou de *15* a *120* quadros, com a segunda custando de três a quatro vezes o preço da primeira.

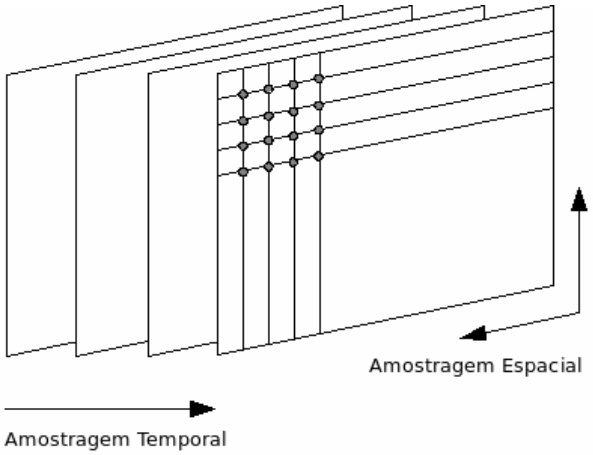


Figura 5 – Amostragem Temporal e Espacial.
(RICHARDSON, 2003:10)

Outra questão importante é a taxa de transferência (*bandwidth*). As *webcams* de baixo custo geralmente utilizam conectores USB 1.1. Algumas versões mais recentes utilizam conectores da versão USB 2.0 ou FireWire. A taxa de transferência para estes conectores são de *12* Mega *bits* por segundo (Mbps), *480* Mbps e *400* Mbps, respectivamente. Ou seja, a velocidade de transferência da versão USB 2.0 é *40* vezes superior a da versão USB 1.1.

A transferência de dados entre uma *webcam* e um computador pode se dar através da imagem crua[1] (*raw data*) ou da imagem comprimida. A compressão dos

[1] Imagem original, sem compressão.

dados pode se dar com perdas (*lossy*) ou sem perdas *(lossless).* As compressões com perdas são as mais eficientes. Nesse tipo de codificação, a imagem decodificada não corresponde a uma réplica exata da original (RICHARDSON, 2003:3). *Webcams* com baixa taxa de transferência (USB 1.1) permitem a transmissão da imagem crua, somente a resoluções realmente baixas. Por exemplo, em uma transmissão crua a Sub-QCIF, com três canais de *8 bits* operando a *30* FPS, têm-se:

*taxa de transferência = colunas * linhas * canais * profundidade_de_cor * FPS* (eq. 2)

*taxa de transferência = 128*96*3*8*30 = 8.847.360* (*≈9 Mbps*)

Isso é inferior aos *12 Mbps* permitidos. Mas, se a resolução for aumentada para QVGA, então se multiplica:

*taxa de transferência = 320*240*3*8*30 = 55.296.000* (*≈55 Mbps)*

Esse resultado supera a taxa de transferência suportada, o que exige que a imagem seja comprimida para poder ser transferida.

Também há diferença na tecnologia do sensor que faz a captura das imagens. O sensor pode ser CCD (*charge coupled device*) ou CMOS (*complementary metal oxide semiconductor*). Existem vantagens e desvantagens para cada um. O sensor CCD tende a ser mais sensível à iluminação, podendo operar inclusive em ambientes com pouca luz. Este sensor também tende a apresentar uma melhor qualidade de imagem. Porém, essas não são regras que valem para todas as câmeras. O nível de qualidade não depende apenas do tipo sensor, e um sensor CMOS de alto desempenho pode superar um sensor CCD de baixo custo.

O Quadro 3 resume o conjunto das características que devem ser consideradas na aquisição de câmeras digitais. Uma qualidade baixa das duas primeiras (resolução e FPS) limita consideravelmente a implementação de interfaces naturais. Como exemplo, pode-se citar a qualidade da apresentação, que ficará prejudica por baixa resolução em grandes monitores, caso o usuário necessite ver sua imagem espelhada. Uma baixa amostragem temporal (inferior a *120* FPS) não

permitirá a leitura precisa de movimentos rápidos (como simulações de atividades esportivas).

Quadro 3 – Resumo da principais características de câmeras digitais.

Tecnologia	Explicação
Resolução	A amostragem espacial determina a qualidade de nitidez da imagem. Grandes resoluções exigem altas taxas de transferência, e muitas vezes compressão das imagens.
FPS	É a amostragem temporal. Aplicações que trabalham com a leitura de movimentos rápidos exigem um FPS maior ou igual a *120*.
Taxa de Transferência	A tecnologia de transmissão de dados pode vir a exigir imagens, mesmo em baixas resoluções (inferiores à resolução de televisores convencionais, como QVGA – *320x240*), sejam transmitidas compactadas.
Compressão	A compressão de dados com perdas, utilizada nas câmeras digitais, devido à eficiência de compressão, produz ruídos que não apareceriam na imagem original.
Sensor	Nas *webcams* de baixo custo, a maior qualidade de imagem e tolerância à baixa iluminação tende a ser atribuída a câmeras com sensores CCD.

Além dos fatores apresentados, também há de se considerar, na câmera digital, a opção de transferência de imagens codificadas (espaço de cor) além do padrão RGB convencional. É claro que espaço de cor pode ser convertido, mas a conversão envolve custo computacional[1]. Maiores explicações sobre espaços de cor se encontram na subseção a seguir.

2.4.3 Modelos de Representação de Cor

Como já foi introduzido anteriormente, imagens coloridas podem ser modeladas em três canais, onde cada canal corresponde a uma cor. Um canal é uma matriz de tamanho correspondente à resolução da imagem cujos índices correspondem à intensidade de cor naquele ponto.

Um modelo típico de representação imagens coloridas, utilizado em monitores CRT e similares (VEZHNEVETS et al 2003:1), é o RGB (*red, green, blue*), que possui o vermelho, o verde e o azul como cores primárias. Durante a exibição das imagens, essas cores são combinadas para produzir o resultado final. RGB é um

[1] Apesar de envolver custo computacional, a conversão entre os espaços de cor é menor quando comparado aos processos de redução e de análise, como filtragem de ruídos e segmentação da região de interesse.

modelo de cor aditivo, isto é, a ausência de valores (zero para as três camadas) representa a cor preta. O equivalente subtrativo deste modelo é o CMY (*cyan, magenta, yellow*), onde a ausência de valores representa a cor branca. Essa representação é conveniente para uso em impressoras, que considera que esta cor não precisa ser impressa no papel, que já é branco por natureza.

Em muitas aplicações, a informação RGB é transformada de forma a separar as informações referentes à intensidade de iluminação (luminância ou *luminance*) e à quantidade de cor (crominância ou *chrominance*). Isso permite o armazenamento em diferentes níveis de amostragem espacial.

O objetivo de separar os dados em unidades de cor e de iluminação, do ponto de vista de armazenamento de dados, é priorizar a informação deste último, utilizando uma quantidade maior de *bits* para armazenamento quando comparada à informação do primeiro. Isso porque o sistema de visão humano é menos sensível à cor do que à iluminação (RICHARDSON, 2003:15).

Um dos padrões de codificação que separa os dados de iluminação dos dados de cor é o YUV. Ele é usado pelos padrões de composição de vídeo colorido *PAL* (*Phase Alternation Line*), *NTSC* (*National Television System Committee*) e *SECAM* (*Sequentiel Couleur Avec Mémoire*). O obsoleto sistema preto-e-branco (níveis de cinza, na verdade) utilizava somente a iluminação (Y) para compor os dados de cor. Os componentes de cor U e V foram adicionados para que a mesma informação pudesse ser transmitida para diferentes decodificadores. Equações matemáticas foram elaboradas para resgatar os valores RBG a partir dos componentes YUV. Variações deste modelo incluem os modelos YIQ e YCbCr (JACK, 1995:15).

Esses formatos permitem o armazenamento em de luminância em crominância em diferentes amostragens (ou resoluções, isto é, a quantidade de *bits*). A representação pode ser da forma *4:4:4*, *4:2:2* ou *4:2:0*. A primeira indica que não há diferença nas quantidades de armazenamento. A segunda indica *2* ocorrências de U e V para cada 4 ocorrências de Y, e a terceira indica apenas *1* ocorrência dos dados de cor (U e V) para cada dado de iluminação (Y).

Para exemplificar de forma prática, considera-se o exemplo apresentado na subseção anterior, da quantidade de *bits* para resolução QVGA. Nesse caso, se a

transmissão de dados for realizada em YUV *4:2:0*, tem-se como resultado, considerando a equação 2:

*taxa de transferência = 320*240*1*8*30 + 160*120*2*8*30 = 27.648.000* (*≈28 Mbps)*

Esta equação considera que os dados de cor possuem a metade da resolução dos dados de iluminação. É o equivalente dizer que cada pixel será representado por *12 bits* (*8+2+2*) ao invés de *24 bits* (*3*8*):

*taxa de transferência = 320*240*12*30 = 27.648.000* (*≈28 Mbps)*

O resultado, ainda com a diminuição dos dados de crominância, é quase duas vezes a taxa de transferência máxima do padrão USB 1.1.

Também há a representação no modelo HSL (*hue* ou tonalidade, *saturation* ou saturação, *lightness* ou brilho), onde se tem a tonalidade como sendo o valor de cor, o brilho como a intensidade de luz e a saturação como a quantidade de branco que a cor possui. HSL também pode ser referenciado por HSI, com o significado de I igual a *intensity*. Por exemplo, rosa é menos saturado que vermelho, por possuir uma maior quantidade de branco.

Uma variação do modelo HSL é o modelo HSV (*hue*, *saturation*, *value*). A diferença está no cálculo do componente de iluminação. Esses modelos foram criados para providenciar uma manipulação mais intuitiva da definição manual de cores em *softwares* editores de imagens. Tais modelos não são utilizados para transmissão de imagens.

Em resumo, os modelos de cores podem ser apresentados de três formas: os modelos de composicação de cores primárias (RGB e CMY), os que separam os dados de cor dos dados de iluminação (YUV, YIQ e YCbCr), e os que foram criados pensando na semelhança de representação com o modelo de percepção humana (HSL ou HSI e HSV). *Webcams* transmitem dados de imagem em um dos dois primeiros formatos, já que durante a comunicação periférico-computador não há necessidade da intuitividade do padrão HSV / HSI na respresentação das cores.

A API em desenvolvimento neste trabalho adotará o RGB como principal espaço de cor utilizado, mas nada impede que em determinadas situações o modelo

RGB seja convertido para outros modelos. Equações de conversão de RGB para HSV e YCbCr podem ser encontradas em (Vezhnevets et al, 2003).

Isto encerra a fundamentação sobre as imagens de entrada. A seção a seguir tem o objetivo de criar uma base teórica sobre os algoritmos de visão computacional e de mostrar o funcionamento de algumas técnicas que serão citadas no capítulo 3.

2.5 ALGORITMOS DE VISÃO COMPUTACIONAL

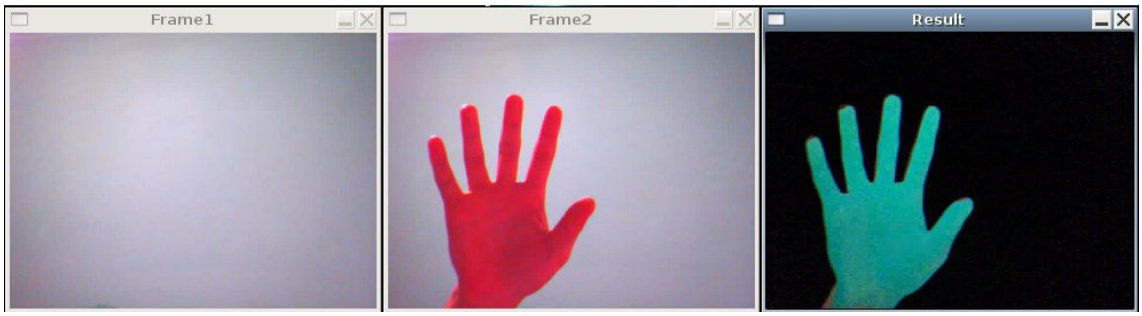
Esta seção fornecerá a base de conhecimento necessária para entendimento das propostas de solução apresentadas nos próximos capítulos. As subseções a seguir foram escolhidas e estão organizadas conforme a relevância para o que se pretende que seja realizado na implementação. As figuras não referenciadas são de autoria deste trabalho.

2.5.1 Diferença entre Quadros de Animação

A diferença entre quadros de animação é uma técnica de identificação de presença e de movimento. Para efetuá-la, dados dois quadros de animação, deve-se percorrer as imagens, *pixel* a *pixel*, realizando a diferença absoluta entre o quadro atual e o quadro de referência, que é efetuada da seguinte forma:

$$imagem_{resultante}(x, y) = | quadro_{referência}(x, y) - quadro_{atual}(x, y) | \qquad \text{(eq. 3)}$$

O quadro atual se refere ao último quadro capturado, que deve ser processado na transformação em entrada de dados. Pode-se extrair dados de presença ou de movimento, dependendo do que for considerado como quadro de referência.



(A) **(B)** **(C)**

Figura 6 – Presença com diferença absoluta.

Para identificação de presença, o quadro de referência deverá ser o primeiro quadro capturado, com a cena estática, sem objetos de interação, ainda que também estáticos. Nesse caso, a diferença absoluta efetuada nas três camadas de cor resultará na Figura 6.C. A Figura 6.A mostra o quadro de referência, seguido pelo quadro atual (Figura 6.B). Nessa demonstração se percebe que os objetos que forem inseridos na cena ficarão evidentes no resultado da diferença.

(A) **(B)** **(C)**

Figura 7 – Movimento com diferença absoluta.

Por outro lado, se o quadro de referência for o quadro anterior ao último capturado, o resultado será o da Figura 7. Observa-se agora que o quadro resultante (Figura 7.C) mostrará a movimentação que foi efetuada na cena. Nesse caso, o objeto de inteesse (neste caso a mão) não efetua entrada de dados quando parado.

Para que a diferença entre quadros seja efetuada de forma correta, em ambas as abordagens, percebeu-se que as seguintes condições devem ser respeitadas:

1. As dimensões das imagens devem ser as mesmas;
2. Não pode haver grandes variações de iluminação entre os quadros;

3. A câmera não pode mudar de posição;
4. Não pode haver movimentação de objetos que não tenham objetivo de interagir com a aplicação dentro do campo de visão da câmera;
5. O primeiro quadro capturado deverá ser um cenário estático (sem objetos em movimento).

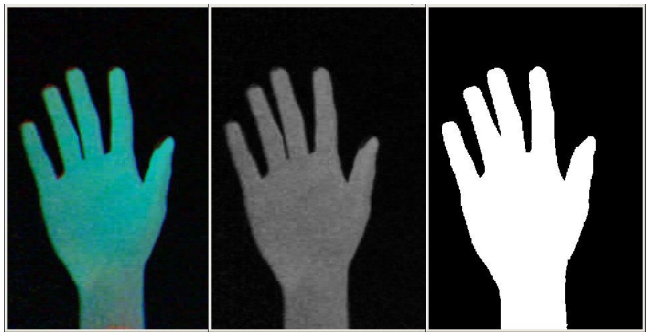
A quinta restrição somente é válida na diferenciação com o objetivo de identificar presença. Esta abordagem (presença) requer inicialização, com captura de tela do cenário em momento apropriado, quando não houver elementos de interação no campo de visão da câmera. Para esta técnica, os itens *2* e *3* são mais críticos, e podem requerer a reinicialização, com nova captura do quadro de referência caso sejam violados (apagar uma lâmpada ou mover a *webcam* de lugar, por exemplo). A funcionalidade de ganho automático de níveis de branco das câmeras digitais inviabiliza o uso desta técnica, já que isso provocará consideráveis mudanças de iluminação. Uma mudança na posição da câmera resulta que todos os novos quadros serão diferentes do quadro inicial, mesmo se todos os objetos continuarem estáticos.

Já a diferença entre os dois últimos quadros, apresentada na Figura 7, pode se adaptar aos itens críticos da identificação de presença, além de não exigir etapa de inicialização. Tentativas de se identificar presença, geralmente referenciada como a segmentação do primeiro plano a partir do plano de fundo (*foreground-background segmentation*) sem uma etapa de inicialização serão descritas na seção 2.3.3.

2.5.2 Redução a Níveis de Cinza e Binarização de Imagens

O resultado final de uma etapa de segmentação geralmente envolve uma imagem binária. Os motivos são o fato de que este tipo de imagem é menor para armazenar e processar, já que contém apenas um canal de profundidade de *1 bit*, e também pelo fato de que o resultado deixa claro quais partes integram a região de interesse (*foreground*) e quais partes fazem parte do cenário de fundo (*background*).

Ambas as figuras 6.C e 7.C estão usando três canais para representar o resultado da subtração. É possível reduzir a quantidade de dados transformando a para níveis de cinza (*1* canal de *8 bits* de profundidade) ou efetuando a binarização.



(A) **(B)** **(C)**

Figura 8 – Imagem resultante, níveis de cinza e binarizada.

A Figura 8.A mostra uma imagem pseudo-colorida resultante da diferença absoluta equivalente à subtração efetuada na Figura 6.C. Na Figura 8.B, tem-se a imagem em níveis de cinza e a Figura 8.C mostra o resultado da binarização. Ambos os resultados são adquiridos a partir da Figura 8.A.

Para efetuar a transformação a níveis de cinza, basta somar os valores dos três canais RGB em cada *pixel* da imagem em pseudo-cor e dividir o resultado por três, conforme:

$$imagem_{cinza}(x, y) = [\, R(x, y) + G(x, y) + B(x, y) \,] / 3 \qquad \text{(eq. 4)}$$

Esta é uma forma intuitiva (média aritmética) de calcular os valores para uma imagem em níveis de cinza. Outra abordagem seria calcular o nível de intensidade de iluminação (componente Y) do padrão YCbCr, conforme (BOURKE, 2000):

$$imagem_{cinza}(x, y) = 0.2989\, R(x, y) + 0.5866\, G(x, y) + 0.1145\, B(x, y) \qquad \text{(eq. 5)}$$

A binarização exige uma comparação da soma dos canais com um determinado limiar (*threshold*). Isso pode ser efetuado da seguinte fora:

$$imagem_{binária}(x, y) = \begin{cases} 1, \text{ se } \left[imagem_{cinza}\ (x_i, y_i) \right] \geq limiar \\ 0, \text{ caso contrário} \end{cases} \qquad \text{(eq. 6)}$$

Uma imagem binária armazena uma quantidade de dados oito vezes menor quando comparada a uma imagem em níveis de cinza e vinte e quatro vezes menor

quando comparada a uma imagem RGB de *24 bits*. Porém, uma imagem binarizada não possui informações sobre a intensidade de brilho. Portanto, a escolha do nível de redução dos dados vai depender de como as características da imagem simplificada precisam ser analisadas considerando o propósito da aplicação.

2.5.3 Segmentação de Cor da Pele

"Segmentação pode ser entendida como um processo que particiona a imagem em regiões" (GONZALEZ e WOODS, 2000:331). Existem diferenças na segmentação de acordo com as propriedades dos vários formatos de representação de cor. De acordo com os últimos autores citados, o espaço de cor RGB é o que produz os melhores resultados de segmentação da pele.

Considerando que segmentação de cores deve ser implementada de forma simples, sem alto custo computacional, para atender ao requisito de identificação em tempo real, este trabalho somente irá considerar formas paramétricas de segmentação da cor da pele. Isso porque formas não paramétricas geralmente estimam a região segmentada a partir de um histograma resultante de treinamento a partir de um modelo de tonalidade de pele específico (GASPARINI e SCHETTINI, 2006 *apud* JONES et al, 1999).

Dentro deste pré requisito, pode-se citar o seguinte conjunto de equações, desenvolvidas de forma empírica, extraídas de (PEER et al, 2003):

$$
\begin{aligned}
&pixel(R, G, B) = cor\_de\_pele\ se: \\
&(R > 95)\ e\ (G > 40)\ e\ (B > 20)\ e \\
&((\max\{R, G, B\} - \min\{R, G, B\}) > 15)\ e \\
&(|R - G| > 15)\ e\ (R > G)\ e\ (R > B)
\end{aligned}
\qquad \text{(eq. 7)}
$$

Um trabalho mais completo dentro do mesmo princípio ilustrado na equação 7 foi relatado em (GASPARINI e SCHETTINI, 2006). A Figura 9 apresenta resultados de segmentação considerando diferentes conjuntos de equações aplicados a vários espaços de cores. Respectivamente, tem-se a imagem original, sequida da segmentação em YCbCr, RGB, dois conjuntos de equações para o modelo HSV (HSV1 e HSV2), uma para o modelo HSI, e a última para o modelo RGB

normalizado (rgb). Neste último modelo, cada componente de cor (*red*, *green* e *blue*) é dividido pela soma dos três componentes para resultar na forma normalizada.

(A) Original

(B) YCbCr (C) RGB (D) HSV1 (E) HSV2 (F) HSI (G) rgb

Figura 9 – Segmentação da cor da pele em diferentes espaços de cor.
(adaptada de GASPARINI e SCHETTINI, 2006)

A Figura 9 mostra que nenhum dos resultados é perfeito, mas todos são satisfatórios no sentido que segmentam boa parte da região que corresponde à pele, levando em conta a simplicidade de implementação. Considerando os resultados apresentados, e o fato de que não são necessárias conversões para trabalhar com o modelo RGB, as equações referentes a este modelo (equação 7) serão utilizadas na API deste trabalho.

2.5.4 Fluxo Óptico

Fluxo óptico é um processo mais complexo em termos de cálculo matemático que a diferença entre quadros de animação ou segmentação de cor da pele por equações paramétricas, e compreende a etapa de redução de dados como um todo, desde a segmentação do objeto de interesse até a extração das características. Esta técnica não só é capaz de identificar movimento, com também estimar a direção e a magnitude.

Na literatura, esta técnica foi utilizada para resolver uma quantidade considerável de problemas. Stavens (2007) afirma que esta técnica pode ser utilizada para rastear um objeto em uma sequência de imagens, para determinar

como um objeto, ou a própria câmera, se moveu ou até para calcular profundidade com uma única câmera.

Existem várias abordagens para a formulação matemática do fluxo óptico (BAKER e MATTHEWS, 2004). Este trabalho usará a implementação de Lucas e Kanade (1981) implementada pela biblioteca OpenCV (OPENCV, 2007). O motivo está no fato de que esta implementação foi demonstrada por outros autores (GALVIN et al, 1998) como sendo a mais eficiente e robusta. Em resumo, o resultado desta técnica pode ser descrito conforme as equações 8 e 9 (VASSALO et al, 2005).

$$u = (x(t + \Delta t) - x(t)) / \Delta t \quad \text{(eq. 8)}$$

$$v = (y(t + \Delta t) - y(t)) / \Delta t \quad \text{(eq. 9)}$$

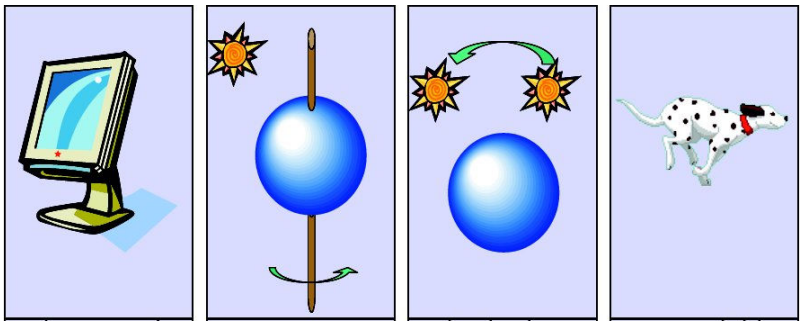
Onde, considerando duas imagens adquiridas nos instantes t e Δt, para cada ponto de contraste nessas imagens, com coordenadas (x, y), o fluxo óptico é vetor com os componentes u e v. A formulação matemática detalhada de como encontrar a coordenada $(x(t + \Delta t), y(t + \Delta t))$ está descrita em (LUCAS e KANADE, 1981).

A Figura 10 (ZELNIK, 2005) expõe as situações que causam resultados equivocados no cálculo do fluxo óptico. A Figura 10.A mostra a tela de um monitor. Este objeto pode estar totalmente estático e passar a impressão de movimento se houver mudança constante do que apresentado na tela, como um filme, por exemplo. Na Figura 10.B, tem-se uma esfera girando, sob a luz de fonte estática. A esfera se encontra girando no mesmo lugar, mas a mudança de posição dos pontos da textura que a envolvem pode levar ao resultado de movimento na direção lateral.

A Figura 10.C mostra a esfera parada, com fonte de iluminação mudando de posição. Nesse caso, mudanças no grau de iluminação também podem causar a falsa constatação de movimento. Por fim, na Figura 10.D, tem-se um exemplo de textura não rígida, que acaba por tornar o rastreamento por fluxo óptico complicado e bastante suscetível à falhas.

Devido à necessidade do cálculo do vetor que estima a direção e magnitude de movimentação dos *pixels*, esta técnica acaba sendo mais custosa do que as que já foram apresentadas. Na API implementada neste trabalho, o desenvolvedor pode determinar o cálculo do fluxo óptico para qualquer região dentro dos limites da

imagem de entrada. Apesar disso, seu uso é recomendado apenas em pequenas regiões, em situações onde o uso da diferença simples entre quadros de animação não resolve o problema.



(A) **(B)** **(C)** **(D)**

Figura 10 – Situações onde o fluxo óptico pode falhar.
(adaptado de ZELNIK, 2005)

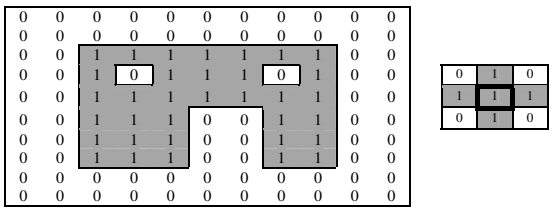
Uma dessas situações é a interação através de simulação de física com objetos virtuais. Por exemplo, em uma aplicação onde o usário deve rebater uma bola virtual, o cálculo do fluxo óptico pode ser efetuado somente para o interior do círculo que representa a bola. Neste caso, o vetor resultante dará a direção e velocidade com a qual a bola deve ser rebatida. Outras aplicações deste tipo estão descritas na seção 3.1.6.

2.5.5 Filtros Morfológicos

Morfologia matemática se refere à estrutura ou forma dos objetos. Filtros morfológicos são responsáveis por facilitar a busca de objetos de interesse. Neste trabalho, os filtros morfológicos são utilizados para tratamento de imagens binárias, especialmente na eliminação de componentes indesejados, resultantes de falhas do processo de binarização.

Os filtros morfológicos serão explicados a partir da Figura 11. A Figura 11.A mostra uma imagem binária de resolução *11x10*. A Figura 11.B é o elemento estruturante que será utilizado nas operações morfológicas. Nas operações de

dilatação e erosão, o elemento estruturante pode ser representado por uma matriz de *pixels* que será aplicada à imagem de entrada.



(A) **(B)**

Figura 11 – Imagem Original e Elemento Estruturante.

(adaptado de UMBAUGH, 1999:93)

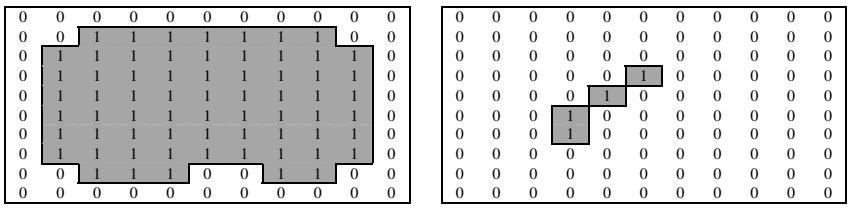
Conforme se pode observar na figura Figura 12.A, a dilatação irá expandir as bordas da imagem a partir da forma definida pelo elemento estruturante. A erosão tem efeito oposto, e irá diminuir a imagem ao "recortar" as regiões que não podem compreender espacialmente os *pixels* ocupados pelo elemento estruturante.

As seguintes operações são realizadas, no caso da dilatação:

1. Se a origem do elemento estruturante coincidir com o valor *0* na imagem original: mover para o próximo *pixel*;
2. Se a origem do elemento estruturante coincidir com o valor *1* na imagem original: realizar a operação morfológica *OR* para todos os pixels do elemento dentro da imagem original.

Como resultado da dilatação, na Figura 12.A, nota-se que todos os pixels de valor *1* são preservados, os limites serão expandidos e pequenos buracos serão fechados. No caso da erosão, as operações realizadas são as seguintes:

1. Se a origem do elemento estruturante coincidir com o valor *0* na imagem original: mover para o próximo *pixel*;
2. Se a origem do elemento estruturante coincidir com o valor *1* na imagem original: mudar para *0* na imagem original o *pixel* correspondente à posição da origem do elemento estruturante, caso algum *pixel* na imagem original tenha valor *0* onde um pixel do elemento estruturante tenha valor *1*.

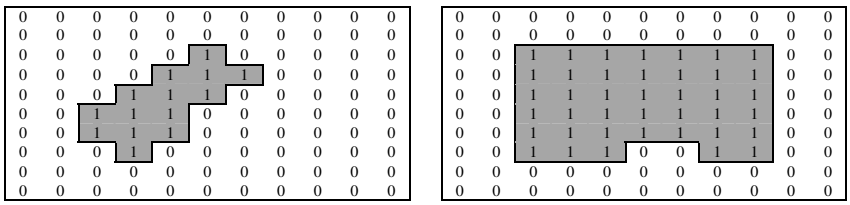


(A) **(B)**

Figura 12 – Dilatação e Erosão.

(adaptado de UMBAUGH, 1999:94)

O resultado da erosão (Figura 12.B) são os fragmentos da imagem original onde o elemento estruturante “encaixa” perfeitamente, sem que nenhum *pixel* de valor *1* sobre do lado de fora da imagem original. As operações de dilatação e erosão podem ser combinadas para resultar nos filtros de abertura e fechamento.



(A) **(B)**

Figura 13 – Abertura e Fechamento.

(adaptado de UMBAUGH, 1999:95)

As operações abertura e fechamento estão ilustrados na Figura 13. Na operação de abertura é realizada erosão seguida de uma dilatação. No contexto deste trabalho, a abertura (Figura 13.A) é responsável por eliminar da imagem binária todas as regiões que sejam muito pequenas para conter o elemento estruturante. Já a operação de fechamento consiste de uma dilatação seguida de uma erosão e tem a propriedade de preencher lacunas interiores à imagem de entrada (Figura 13.B).

O número de operações efetuadas por cada um dos filtros morfológicos, erosão e dilatação, é igual à quantidade de *pixels* da imagem vezes o total de *pixels* do elemento estruturante. Na abertura ou fechamento, a quantidade de operações é o dobro. Sendo assim, a utilização desses filtros não pode ser efetuada a todo momento, sob pena de limitar a velocidade de uma aplicação em tempo real. A

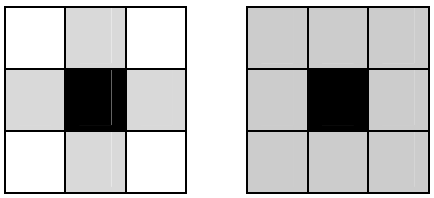
maior utilidade desses operadores para a API desenvolvida neste trabalho é, por exemplo, a filtragem para correção de falhas na imagem binária resultante da identificação de cor da pele, exemplificada na Figura 9.

2.5.6 Contornos

Conectividade entre elementos em uma imagem digital é um conceito utilizado para estabelecer limites entre elementos que compõem uma imagem. Para determinar se dois *pixels* vizinhos são conexos, é necessário verificar se eles são vizinhos segundo o tipo de vizinhança adotado e se esses elementos satisfazem determinados critérios de similaridade, tais como intensidade de cinza, cor ou textura (PEDRINI e SCHWARTZ, 2008).

No contexto deste trabalho, a extração de contornos é realizada a partir de imagens binárias ou em níveis de cinza. Para o entendimento desta subção, valores iguais a *0* (zero) devem ser considerados como *pixels* de intensidade *0* e o valores iguais a *1* devem ser considerados como qualquer valor de intensidade acima de *0*. Dessa forma, uma imagem em níveis de cinza equivale a uma imagem binária.

O cálculo dos contornos na API OpenCV considera dois possívies tipos de conectiviadade (Figura 14). Na Figura 14.A, a conectividade 4 considera os *pixels* horizontal e verticalmente adjacentes ao *pixel* central. Na Figura 14.B, de conectividade 8, os vizinhos diagonais também são considerados.



(A) **(B)**

Figura 14 - Conectividade 4 e conectividade 8.
(adaptado de INTEL, 2001)

A partir desta relação, uma imagem pode ser quebrada em vários componentes não sobrepostos de vizinhança 4 ou vizinhança. Cada componente constitui um conjunto de pixels de mesmo valor (*0* ou *1*). Na API OpenCV, os componentes de valor *0* são de conectividade 4 e os de valor *1* são de conectividade

8 (INTEL, 2001:3-2). Tanto na API OpenCV quanto na API Move-In (resultado deste trabalho), apenas a topologia dos componentes de valor *1* é considerada, e os componentes de valor *0* são considerados como sendo plano de fundo. Esses componentes de valor *0* ou *1* na imagem binária podem também ser informalmente chamados de *blobs* (*binary large objetcs*).

Cada componente de valor *1* possui um única borda externa (*outer border*) que o separa do componente de valor *0* que o envolve, e zero ou mais bordas internas (*hole borders*) que o separam dos componentes de valor que são envolvidos por ele. Na Figura 15, essas bordas estão em destaque para o maior *blob* de valor *1*.

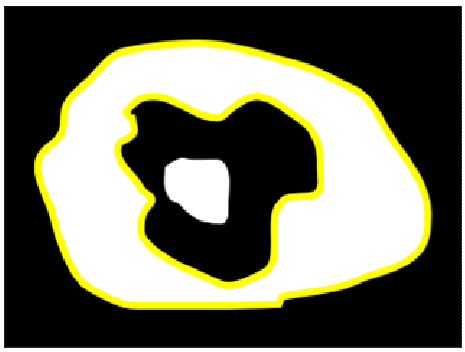


Figura 15 – Borda externa e borda interna.

A borda externa e as possíveis bordas internas podem descrever um *blob* de forma simplificada. São essas bordas que são chamadas de contornos. É a partir dos contornos que são calculados o fecho convexo e os defeitos de convexidade. Esses conceitos são abordados nas seções 2.5.7 e 2.5.8.

2.5.7 Fecho Convexo

O fecho convexo de um conjunto de pontos, ilustrado na Figura 16, é o menor subconjunto convexo que contém todos os pontos do conjunto (PREPARATA e SHAMOS, 1998 *apud* FISHER, 2004). É um conceito fundamental de geometria convencional, e existem muitas soluções propostas por diferentes autores (FISHER, 2004: 2).

A solução adotada pela biblioteca OpenCV (OpenCV, 2006), e consequentemente, a solução adotata na API desenvolvida neste trabalho utiliza

uma versão do algoritmo *quicksort* (*quickhull*) para determinar o conjunto de pontos extremos que compõem o fecho convexo. A formulação matemática desse algoritmo pode ser conferida no trabalho de Barber et al (1996).

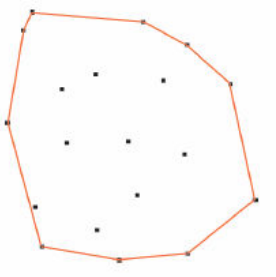


Figura 16 – Fecho convexo de um conjunto de pontos.
(FISHER, 2004)

A partir da definição de fecho convexo, pode-se concluir que, em uma imagem binarizada, este recurso pode dar início a identificação dos pontos extremos da silhueta de um objeto de interesse em uma imagem binarizada. Apenas esta característica não resolve o problema de identificar partes do corpo, mas pode ser usada como ponto de partida na identificação de extremidades ou articulações durante a identificação de poses das mãos ou do corpo humano.

2.5.8 Defeitos de Convexidade

Considerando um conjunto de pontos $P(p_1, p_2, ..., p_n)$ que representa o contorno e um conjunto de pontos $H(h_1, h_2, ..., h_n)$ que representa o fecho convexo desse contorno. É possível que exista um subconjunto do contorno *P* entre dois vértices do fecho convexo *H*. O defeito de concavidade representa a profundidade deste subconjunto do contorno em relação aos vértices do fecho convexo.

A Figura 17 mostra um exemplo dos defeitos de concavidade que podem ser extraídos a partir de uma imagem binária. A estrutura que compõe um defeito de concavidade consiste de um ponto inicial *s*, um ponto final *e* (vértices do fecho convexo) e um ponto de profundidade *d* (proveniente do contorno) e um valor que caracteriza esta profundidade.

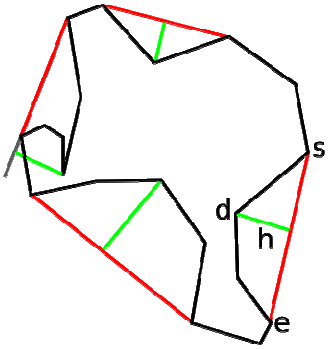


Figura 17 – Defeitos de Convexidade
(adaptado INTEL, 2001)

Na API Move-In, a estrutura de dados “pontos extremos” (*ExtremePoints*) armazena uma lista de pontos que são classificados como convexos ou côncavos. Portanto, esta estrutura pode ser utilizada para armazenar o fecho convexo em conjunto com os defeitos de convexidade. Além de armazenar os pontos, esta estrutura serve para agrupar pontos que se encontram muito próximos uns dos outros. A distância que determina se os pontos estão próximos pode ser definida pelo programador e o agrupamento ocorre através da substituição desses pontos pela posição média entre eles.

2.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a definição de análise de imagens, mostrando as etapas do processo, desde a aquisição de imagens até a análise das características. Logo após isso, foi definido o conceito de visão computacional, sob o ponto de vista de vários autores, indicando as principais diferenças desta quando comparada ao processamento de imagens (seção 2.1). A partir daí, importantes características, que devem ser consideradas durante o processo de aquisição de imagens foram relatadas na seção 2.2.

A seção 2.3 apresentou alguns algoritmos da etapa de redução de dados. Mais precisamente, foram mostradas a diferença de quadros, a identificação de cor da pele como etapas de segmentação, os filtros morfológicos que podem ser usados para melhorar o resultado da segmentação e o fluxo óptico e fecho convexo como etapa de extração (ou identificação) de características.

A etapa de pré-processamento não foi detalhada por se tratar mais de redução de ruídos, utilizada com mais frequência em processamento de imagens. Ainda assim, esta etapa será citada em alguns pontos do capítulo 5, durante a apresentação da proposta de solução. A etapa de análise de características não foi discutida em detalhes pelo fato de ser totalmente inerente aos propósitos da aplicação. Isto é, é nesta etapa que as características extraídas são utilizadas para, por exemplo, acionar um botão virtual, disparar um projétil ou substituir cenário de fundo por cenário virtual. Alguns processos desse tipo estão exemplificados no próximo capítulo, que tem por objetivo fazer a revisão da literatura que contribui com trabalhos relacionados aos propósitos da API Move-In.

3. TRABALHOS RELACIONADOS

Este capítulo apresenta trabalhos que foram desenvolvidos para fins acadêmicos ou comerciais, e que têm relevância para a implementação da biblioteca. A seção 3.1 detalha o propósito e a solução abordada por cada trabalho, muitas vezes fazendo referência às etapas de processamento de imagens discutidas no capítulo 2. Um comparativo final entre as técnicas levantadas é realizado na seção 3.2.

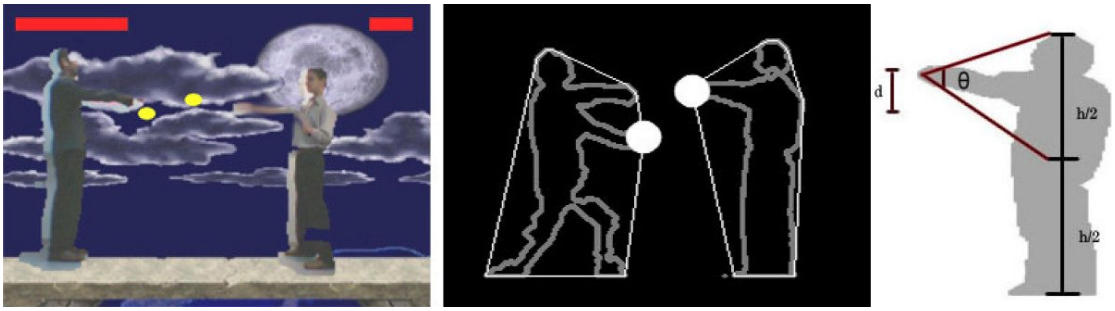
Os trabalhos aqui ilustrados estarão dispostos da seguinte forma: em cada subseção, logo após o título do trabalho, serão feitos comentários sobre o propósito da aplicação. A proposta de solução será apresentada, tentado estabelecer uma relação com a Figura 3, que ilustra as etapas de análise de imagens, sempre que possível. Os itens que serão explicados serão: pré-processamento, redução de dados (que consiste principalmente das sub-etapas de segmentação da área de interesse e extração de características) e análise de características. A imagem de entrada provém de câmeras ligadas ao computador, para todas as aplicações. Ainda, em cada subseção, serão descritos alguns dos fatores positivos e negativos relacionados à solução proposta pelos autores.

3.1 CAMERA KOMBAT

O trabalho realizado por De Paula et al (2006) consiste de um jogo que utiliza movimetos do corpo ao invés de teclado ou *joystick*, fazendo referência ao popular *videogame* de luta *Mortal Kombat* (MIDWAY, 1992). A interação de combate nesse jogo se dá através de lançamento de projéteis. Esses projéteis são disparados uma vez que pelo menos um dos braços seja erguido em direção ao adversário. O projétil diminui a barra de energia do adversário ao atingí-lo. A barra de energia se localiza na parte superior da tela (Figura 18.A).

Esta implementação utiliza a diferença entre dois quadros para realizar a segmentação dos personagens. A versão utilizada foi a que detecta presença, nesse caso, capturando através de evento do teclado o cenário de fundo que servirá como quadro de referência. Isso caracteriza a etapa de redução de dados, onde as partes da imagem que não interferem na interação, isto é, o cenário de fundo, é eliminado.

Este processo acaba também por realizar a segmentação da área de interesse, que são as silhuetas que representam os jogadores.



(A) **(B)** **(C)**

Figura 18 – Camera kombat.

(De Paula et al, 2006:1,6)

A análise de características, que determina o propósito do processo de análise de imagens, neste jogo em específico, tem duas tarefas a realizar: (1) determinar o momento em que um projétil deve ser disparado e (2) determinar se há colisão do projétil disparado pelo adversário com a silhueta que representa o jogador. As características extraídas da região segmentada (a silhueta que representa o usuário, Figura 18.C) são: cabeça, posição das mãos, posição dos pés e centro de massa. Essas características, a menos da última, são extraídas a partir dos pontos extremos do feixo convexo calculdado sobre o contorno que representa a região da imagem ocupada pelo jogador (Figura 18.B). A posição *(x,y)* aproximada do centro de massa é a coordenada *x* do ponto que representa a cabeça e a média aritmética entre a coodenada *y* que presenta a cabeça e os pés.

Para efetuar a tarefa (1), um cálculo é realizado para determinar o ângulo *Θ* formado entre a abertura dos braços e e a região próxima ao centro de massa (Figura 18.C). A tarefa (2), de colisão entre projétil e jogador é calculada com base da posição da esfera e contorno da silhueta. Se a esfera ocupar a mesma posição que a região do contorno, ocorre colisão e o jogador é penalizado com a dimunuição da barra de energia.

A vantagem desta aplicação está no uso de hardware comum, com processador Pentium Celeron de *2.4* GHz, *512* MB de RAM e placa de vídeo Nvidia Geforce 5200. O tipo de webcam não foi especificado, mas foi citada uma taxa de

transferência igual a *30* FPS. Quanto as desvantagens, um olhar cuidadoso direcionado à Figura 18 notará os defeitos do processo de segmentação primeiro plano e de plano de fundo, onde se pode perceber várias falhas ao longo das silhuetas segmentadas dos usuários. É comum que isso ocorra para a técnica utilizada quando não há muito contraste entre as cores do cenário de fundo e o objeto que deve ser segmentado. Esses defeitos também podem ser percebidos em detalhes no contorno da silhueta na Figura 18.B.

Quanto a forma de interação, pode-se afirmar que os jogadores são forçados a virar a cabeça em direção ao monitor para receber *feedback* visual, e devem permancer próximos um do outro para não fugir do espaço capturado pela câmera. *Camera Kombat* não é uma aplicação muito diversificada do ponto de vista de interação, já que esta é baseada em um única postura. Porém, os sistemas comerciais não costumam revelar como suas funcionalidades foram implementadas. Sendo assim, o trabalho apresentado nesta subseção pode ser utilizado como prova de conceito da utilização de algumas técnicas e de *hardware* de baixo custo, além de favorecer elementos de comparação com trabalhos comerciais.

3.2 DOG FIGHT: CONTROLE DE UM AVIÃO VIRTUAL ATRAVÉS DE POSTURAS

O trabalho entitulado *Dog Fight* (CLINE e LARKINS, 2004) segue a mesma linha do trabalho citado anteriormente, utilizando a mesma técnica para eliminação do plano de fundo, e realizando o reconhecimento de um conjunto de posturas. A lista das posturas reconhecidas é a da Figura 19. Cada figura corresponde a um gesto de ação no avião que é controlado pelo jogador: cima (Figura 19.A), baixo (Figura 19.B), direita (Figura 19.C), esquerda (Figura 19.D), neutro (Figura 19.E) e atirar (Figura 19.F).

Os autores de *Dog Fight* descreveram a seguinte estratégia de implementação:

1. Aquisição de dados: é o processo de captura de quadros de animação a partir da *webcam*;
2. Segmentação do plano de fundo: determinação da região contendo o jogador que controla o avião;

3. Detecção do torso / segmentação de região: particiona a região do jogador em componentes distintos, para facilitar o reconhecimento das posturas executadas pelos braços;
4. Reconhecimento das posturas: analisa os segmentos da etapa anterior para determinar a posição de cada braço e, consequentemente, a condição de orientação do avião ou disparo de fogo (postura “atirar”);
5. Display de simulação: projeção dos valores computados em ambiente tridimensional.

(A) **(B)** **(C)**

(D) **(E)** **(F)**

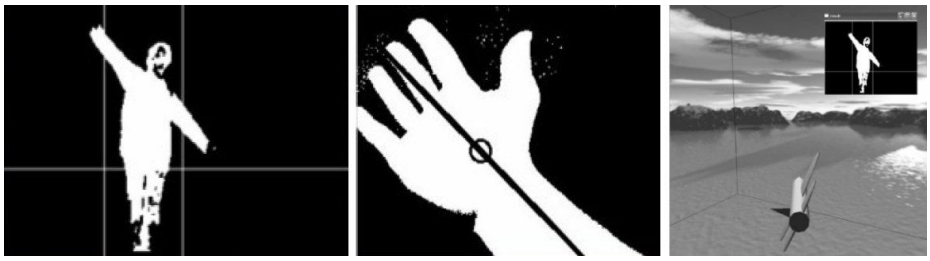
Figura 19 – Gestos de Interação *Dog Fight*.

(CLINE e LARKINS, 2004:5,6)

Nota-se que a estratégia de implementação se assemelha muito às etapas de análise de imagens descritas no capítulo 2, Figura 3. O primeiro passo é a aquisição das imagens. Os passos 2 e 3 são, respectivamente, a redução de dados com segmentação da área de interesse e a identificação de características que serão utilizadas durante o cálculo do comportamento do avião, que consiste das ações de manobrar e atirar. O reconhecimento das posturas é a análise de características.

Os autores ainda descrevem um passo extra, de saída de dados. Uma filtragem é citada como necessária para eliminar ruídos na imagem binarizada, principalmente para tornar mais robusta a classificação da postura “atirar”. Para

resolver este problema, os autores aplicam um filtro de mediana *3x3* para suavizar a imagem. Como esse processo foi realizado após a binarização, esta pode ser considerada como uma etapa de pós-processamento (etapa esta que não aparece na Figura 3).



(A) **(B)** **(C)**

Figura 20 – Segmentação de regiões de interesse, centróide e orientação do braço, saída do sistema.

(CLINE e LARKINS, 2004:4,7)

A Figura 20.A e a Figura 20.B ilustram, de forma mais detalhada, o processo de identificação dos gestos, da segmentação da área de interesse à extração e análise de características. Primeiramente, a imagem binarizada é dividida em três regiões: braço esquerdo, tronco e braço direito. As linhas verticais (Figura 20.A) são traçadas de fora para dentro, até passarem por uma determinada quantidade (*threshold*) de *pixels* brancos. A linha horizontal serve apenas de guia. Para cada braço, uma elipse é sobreposta de forma a envolver o membro. São calculadas a orientação da elipse e o centróide do braço. As orientações das elipses de cada braço indicam a orientação que o avião deve assumir (Figura 20.C). O cálculo do centróide serve para a ação de atirar, onde a ação de dobrar o braço acaba por trazer o centróide para fora da silhueta do braço.

Cline e Larkins (2004) relataram terem ficado satisfeitos com o desempenho da aplicação, a não ser pela forma não robusta da ação de atirar, quando o algoritmo acabava por orientar o avião de maneira estranha. Também de acordo com os autores, esse problema poderia ser parcialmente resolvido ao tornar as ações de orientar e atirar mutuamente exclusivas. Outras formas de realizar esta ação foram sugeridas: segurar com a mão e apontar um CD para a câmera, identificando uma elipse para efetuar a ação; balançar o braço para detectar uma

determinada quantidade de movimento; ou segurar um objeto grande que aumentaria de forma considerável as dimensões da silhueta. Ao contrário da movimentação para fins de orientação, nenhuma das alternativas da ação de atirar parece ser suficientemente intuitiva e robusta.

3.3 INTEL PLAY ME2CAM

A aplicação comercial da Intel foi colocada a venda em outubro de 1999 (D'HOGE e GOLDSMITH, 2001). Esta é uma implementação de interfaces naturais com visão computacional aplicada à jogos, tendo crianças de quatro a oito anos de idade como público alvo. O pacote comercial consistia em cinco jogos para computador e uma *webcam* provida de *drivers* para o sistema operacional *Windows*. A referência indicada foca sua argumentação nos princípios de *design* dessa tecnologia.

Como se trata de uma aplicação comercial, não foram divulgados detalhes de implementação. No entanto, considerar as questões relacionadas ao *design* das aplicações pode servir de guia para a implementação proposta por este trabalho.

Quando *software* é iniciado, uma rua virtual (Figura 21.A) é mostrada, com o usuário vendo sua imagem espelhada. O deslocamento pela rua ocorre ao inclinar a cabeça ou ao movimentar os braços para o lado. Ficar parado durante um determinado tempo em frente à uma das construções dá início a uma atividade.

Como já foi dito, são ao todo cinco jogos:

1. *Bubble Mania* (Figura 21.B): o jogador deve desviar de certos tipos de bolha e estourar outras;
2. S*now Surfin'* (Figura 21.C): permite que o jogador desça uma montanha, desviando de obstáculos que diminuem sua velocidade;
3. *Club Tune*: proporciona uma pista de dança, um teclado virtual e vários estilos musicais. O jogador deve se mover (dançar) para estimular uma banda a continuar tocando e também pode interagir de forma musical através da interação com o teclado virtual;
4. *Fun Zone*: não proporciona um objetivo específico. Nesta aplicação, o jogador pode passar por uma série de espelhos especiais que deformam e adicionam efeitos visuais variados à sua imagem projetada;

5. *Pinball*: é a implementação onde o mundo virtual é uma máquina de *pinball*, da qual o jogador faz parte, substituindo uma série de palhetas posicionadas ao longo da mesa. A movimentação dos braços interage com a bola.

(A) **(B)** **(C)**

Figura 21 – Aplicações da Intel Play Me2Cam.
(D'HOGE e GOLDSMITH, 2001:2)

No jogo *Bubble Mania*, as bolhas se comportam de forma diferente, caso sejam tocadas pelos braços ou pela cabeça. A aplicação, de acordo com os autores, atinge este fim através do uso de uma heurística simples (que não é especificada). É provável que o recurso de segmentação de cor da pele (redução de dados com segmentação da área de interesse) seja utilizado em conjunto com alguma outra técnica, como identificação da posição dos membros segmentados.

Em *Club Tone*, a quantidade de movimento determina o comportamento da banda virtual. O problema de quantificar a movimentação pode ser resolvido com a diferença entre os dois últimos quadros, conforme foi exemplificado na Figura 7, do capítulo 2. Este é um processo de extração de características, onde a característica a ser analisada é a quantidade de movimentação. Isso pode ser calculado a partir a imagem binarizada da diferença entre os quadros, dividindo-se a quantidade de *pixels* brancos pela quantidade total de *pixels* na imagem. O resultado será a porcentagem de movimentação.

De acordo com os autores das aplicações da IntelPlay, todas as aplicações desse pacote são baseadas nas técnicas de segmentação de plano de fundo, detecção de mãos e braços e detecção de movimentos. Isso caracteriza a etapa de redução de dados com segmentação da área de interesse. A segmentação do plano

de fundo é realizada da mesma forma que nas aplicações citadas anteriormente. O quadro de referência é capturado no início da aplicação, quando o jogador é convidado a se retirar do campo de visão da câmera durante alguns segundos. Na referência, é alertado que cortinas balançando ou animais de estimação passando no campo de visão da câmera podem influenciar na interação com as aplicações.

Para tentar ser um pouco mais robusto e tolerante a este tipo de falha, o *software* captura um novo plano de fundo ao detectar ausência de movimentação. O usuário deve estar ciente disso, quando ao perceber dificuldades de interação, este deve sair da frente da câmera por um instante para que os algoritmos possam capturar o novo quadro de referência. Não é especificado como a aplicação faz para não capturar erroneamente um quadro de referência do qual o jogador faz parte, caso ele fique parado durante alguns instantes a frente da tela. É possível que isso seja resolvido (capturar um novo quadro de referência) somente quando nenhuma tonalidade de cor de pele seja detectada.

Além disso, existem outras situações que são difíceis de tratar. Por exemplo, se o jogador estiver posicionado próximo a uma parede, dependendo do nível de iluminação, sombras serão projetadas. As sombras também constituem diferença em relação ao quadro de referência, e poderão ser tratadas como uma presença que não existe. Roupas nas mesmas cores do cenário de fundo também prejudicam o processo de segmentação do plano de fundo.

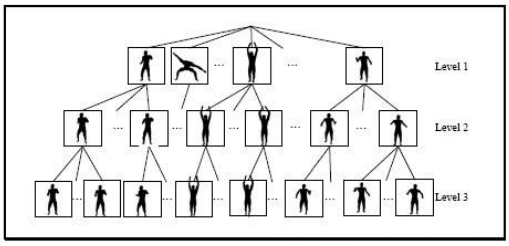
D'Hoge e Goldsmith (2001) ainda afirmam que estudos realizados comprovam iluminação turva ao redor de computadores pessoais, e que quando está muito escuro, fica difícil detectar detalhes da face do usuário. Isso indica a provável identificação da face, além da segmentação da pele, para localização da cabeça. A falta de iluminação é detectada automaticamente. Nesse caso, o ajuste manual da iluminação é sugerida pelo *software*.

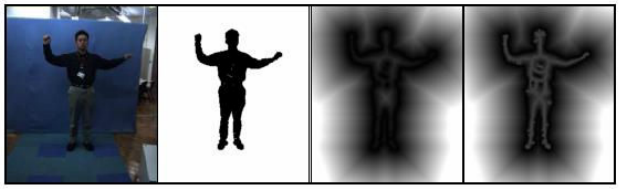
Por último, há mais um detalhe importante a ser citado quanto à transmissão de dados. É afirmado que a compressão e descompressão de imagens utilizadas para altas resoluções introduz uma latência indesejada no sistema (D'HOGE e GOLDSMITH, 2001:5). Para minimizar o nível de latência, foi utilizada uma resolução de vídeo de *180x120*, sem compressão.

3.4 IDENTIFICAÇÃO DE POSES EM TRÊS DIMENSÕES

Considerando o não uso de marcadores, existem também implementações que objetivam a estimação de posicionamento do corpo humano em três dimensões, utilizando apenas uma câmera, tal como a proposta definida por (OKADA et al, 2007). Para tanto, os autores desta aplicação iniciaram o processo com a construção de uma árvore hierárquica de posições (Figura 22.A) a partir de um modelo 3D projetado sobre um plano 2D. Cada nó da árvore contém uma silhueta e um vetor de pose a ela associado. A construção desses nós foi realizada com o auxílio de um sistema que utiliza marcadores. Porém, esses marcadores somente estão presentes na etapa inicial, de construção da árvore hierárquica de posições, e não são utilizados durante a execução da aplicação.

(A)





(B) **(C)**

Figura 22 – Rastreamento 3D sem uso de marcadores.

(OKADA et al, 2007:2,3,4)

Durante o processo de captura com marcadores, apenas eram armazenados os dados de posição que variavam em mais de 5 graus em algum ângulo quando comparado à outras poses. Os três níveis da árvore contêm imagens em diferentes resoluções, *80x60* no primeiro nível, *160x120* no nível 2 e *320x240* no nível 3. Isso possibilita um afinamento progressivo da precisão na procura da solução.

A etapa de construção da árvore não está relacionada ao processo de análise de imagens apresentado na Figura 3. Na implementação de teste, que rodava em tempo real, foi utilizada uma câmera colorida com resolução de *640x480*. Esta resolução era então diminuída para *320x240*, inicando assim, a etapa de pré-processamento. A transição do pré-processamento para a segmentação da área de interesse ocorre com a cor azul sendo eliminada da figura. Esta cor corresponde ao plano de fundo (Figura 22.B e Figura 22.C). Esta é uma técnica especial de segmentação através de cores chamada de *chroma key*.

A Figura 22.B mostra a imagem original, a silhueta estraída a partir da técnica de *chroma key*, e duas formas esqueletizadas. A técnica de *chroma key* realiza a segmentação do plano de frente através da remoção dos *pixels* de valor igual a um dos componentes RGB (geralmente são utilizadas telas azuis ou verdes). É a partir das formas esqueletizadas que são estraídas as características que devem ser analizadas. Isto é, as áreas mais claras dentro da silhueta correspondem maiores pesos para a função que irá comparar a silhueta presente nos nós da árvore hierárquica de posições. A determinação da pose final 3D (análise de características) ocorre quando a silhueta analisada possui alta probabilidade de corresponder ao nó folha da árvore a que está sendo comparada.

A grande vantagem desta aplicação está na recuperação estimada da posição em três dimensões, que trata inclusive de oclusões, efetuando o rastreando quanto a parte do corpo ocludida reapacere na cena. As desvantagens estão, principalmente, (1) na construção da estrutura de dados (árvore hierárquica de posições) necessária para a determinação da posição 3D, (2) no fato de que a posição ou pose a ser reconhecida deve obrigatoriamente estar presente na árvore e (3) no computador pessoal de alto desempenho que deve ser usado para realizar a computação descrita em tempo real.

Para demonstração da técnica apresentada, a árvore construída continha *7.828*, *26.805* e *44.108* nós para cada nível, respectivamente, da raiz às folhas. Além disso, para aumentar a robustez do rastreamento, é necessário que o modelo 3D corresponda em semelhança ao usuário da aplicação. Baseado em padrões da indústria japonesa, foram construídos *10* modelos de corpo humano para homens, *14* modelos para mulheres e *6* modelos para crianças. Todos esses dados, nós e modelos, eram carregados na memória RAM, ocupando cerca de *300* MB. O tempo

médio de processamento foi de *127* milisegundos para um PC com processador Two Opteron *280* Dual Core, *2.8* GHz, com *4* GB de RAM.

3.5 QUADRO NEGRO VIRTUAL

O trabalho proposto por (WU et al, 2000) consiste na habilidade de poder desenhar no espaço 3D utilizando a ponta do dedo indicador. Isso é possível através da utilização da combinação de duas técnicas de redução de dados e segmentação da área de interesse, mais precisamente, segmentação da cor da pele e diferença entre os dois últimos quadros de animação, acrescidas de um algoritmo que irá localizar a ponta do dedo. Este algoritmmo, juntamente com a computação dos valores de posição da cabeça, dos ombros e dos cotovelos, constitui a extração de características, isto é, a localização das partes do corpo. Por fim, a análise das caraterísticas objetiva usar as coordenadas de posição para determinar o caminho percorrido pelo dedo indicador no espaço tridimensional ao longo do tempo de interação com a aplicação.

Na etapa de redução de dados e segmentação da área de interesse, o conjunto de algoritmos inicia com a segmentação da cor da pele. Isso identifica a posição da cabeça e dos braços. Através detecção de movimento, com diferença entre os dois últimos quadros, é realizada a identificação do braço que está desenhanho. Isso impõe a restrição de que o outro braço não pode ter mudanças bruscas de movimento, para não confundir o algoritmo.

Durante a etapa de extração das características, o contorno é extraído da silhueta binarizada do braço gesticulante. A característica a ser identificada é a ponta do dedo. Isso é efetivado ao se percorrer o contorno com dois vetores, em busca do produto escalar de maior valor (Figura 23). Na Figura 23 se vê o resultado do produto escalar aplicado sobre segmentos do contorno.

A cabeça é identificada como sendo a região branca na imagem binarizada de maior centróide (mais alto valor no eixo Y do plano cartesiano), desde que esta região não seja o braço que está se movendo. Quando há oclusão da cabeça pelo braço que está desenhando, a posição desta não é calculada, por não exercer grande influência no cálculo de posição 3D.

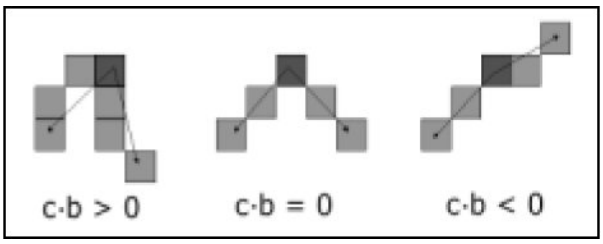


Figura 23 – Procura pelo produto escalar de maior valor.

(WU et al, 2000:3)

Para calcular a posição dos ombros, é assumido que, se o usuário estiver voltado para frente, o ombro mantém uma distância fixa do centro da cabeça, cuja posição pode ser calculada através de uma simples equação. O cotovelo é encontrado ao se erodir diversas vezes a silhueta binarizada (resultante da segmentação da pele) do braço que se encontra em movimento. Então, no resultado da erosão, é procurado o ponto que maximiza a distância euclidiana da ponta do dedo identificada anteriormente.

Considerando que os ossos mantêm um tamanho fixo, a distância entre os pontos calculados pode ser usada para estimar a posição da ponta do dedo no espaço tridimensional. Esse cálculo é efetuado através de cinemática esférica. Considerando que a distância entre o ombro e o cotovelo é fixa, o ombro pode se mover apenas através da tangente de uma certa esfera. O centro desta esfera é o cotovelo e seu raio é igual ao comprimento do antebraço. Partindo dessas considerações, as coordenadas 3D podem ser resgatadas a partir das coordenadas 2D, através de equações que descrevem os pontos extremos de uma esfera.

(A) **(B)** **(C)** **(D)**

Figura 24 – Passos da recuperação da posição 3D.

(WU et al, 2000:3,4,6)

A Figura 24 mostra alguns passos importantes da recuperação da posição tridimensional do dedo. Da esquerda para a direita, tem-se: a imagem de entrada (Figura 24.A), a segmentação das cores da pele (Figura 24.B), as características: posição estimada da cabeça, ombro direito, cotovelo direito e dedo indicador direito (Figura 24.C) e a uma sequência de pontos rastreada em 2D e traduzida para na terceira dimensão (Figura 24.D).

Outra funcionalidade desta implementação é a suavização da linha que descreve a trajetória do dedo ao longo da captura de movimentos. Este trabalho é um exemplo de como um conjunto de algoritmos com baixo grau de complexidade podem ser usados na recuperação de posicionamento de partes do corpo em situações específicas.

Quanto a robustez das técnicas empregadas, muitas informações importantes não foram divulgadas no trabalho de Wu et al (2000). Não foram citados detalhes de como é realizada a segmentação da cor da pele, e não foi comentado se o algoritmo seria suficientemente robusto para segmentar somente a pele no caso de quando o usuário estiver usando uma camisa de cor semelhante à pele. No caso apresentado na Figura 24, a cor da camisa tinha bastante contraste com a pele do usuário. Não foi comentado quando o algoritmo inicia o “desenho” no espaço 3D, isto é, se o software somente identificar a ponta do dedo ou pode se confundir com outra parte do corpo. Também não foi especificado o grau de correspondência com o desenho virtual e o gesto realizado pelo usuário.

3.6 OBJETOS VIRTUAIS GUIADOS POR FLUXO ÓPTICO

De acordo com (ZIVKOVIK, 2004), o fluxo óptico é uma extensão natural da técnica de diferença entre os dois últimos quadros de animação. Devido à sua funcionalidade de estimar a direção e o movimento dos pixels ao longo de uma sequência de vídeo, o fluxo óptico pode ser usado em aplicações muito mais ricas de interação. Para comprovar esta premissa, o autor faz uso de comparações entre essas duas técnicas através de simples aplicações.

Como o objetivo desta técnica é estimar valores de direção e velocidade, que são características a serem exploradas pela aplicação, pode-se afirmar que esta técnica compreende os passos de redução de dados até a extração das

características. Restará para o desenvolvedor da aplicação dar sentido à velocidade e à direção calculadas.

Zickovik (2004) entitula as técnicas de interação naturais com visão computacional de *videoplace*. Em uma aplicação típica de *videoplace*, algumas entidades computacionais são geradas para proporcionar interação com o usuário. Essas entidades, ou objetos que reagem às interações são chamados de "*vison-driven gadgets*". Existe uma variedade grande desses objetos, mas os mais comuns são:

- Objeto estático: são objetos virtuais que não reagem à interações. Entre eles estão as "barras de energia" do jogador e indicadores de pontuação;
- Botão: é o componente básico de toda aplicação do tipo *videoplace* que permite a interação exclusivamente através da movimentação natural. A implementação mais comum é o botão que se move, onde a interação (geralmente o simples toque) é usada para disparar uma mudança no seu comportamento. As bolhas do jogo *Bubble Mania*, apresentado na subseção 3.1.3 são um exemplo desse tipo de botão;
- Objeto movível: é outro integrante comum da interface com o usuário. São objetos que podem ser deslocados pela tela através de movimentos com o corpo. De acordo com o autor, jogos que utilizam somente a diferença entre quadros de animação apresentam este tipo de interação de forma bastante precária. Com a utilização de fluxo óptico, é possível mover esse tipo de objeto de maneira muito mais eficiente.

A Figura 25 exemplifica de forma didática o resultado de saída do fluxo óptico. Da esquerda para a direita, tem-se os dois últimos quadros capturados, seguidos da forma visual de movimentação de alguns *pixels*. As linhas representam movimento. O comprimento das linhas dá a magnitude. Cada linha na verdade é um vetor, que indica para onde o *pixel* está se movendo.

O autor citado ainda ressalta as desvantagens das técnicas apresentadas anteriormente em relação ao fluxo óptico. É argumentado que as implementações que utilizam a diferenciação de quadros com captura de cenário de fundo, *chroma key* e a segmentação das tonalidades da pele não se adaptam com robustez às aplicações do "mundo real". Isso porque, a primeira técnica citada requer inicialização, e possivelmente até a reinicialização do sistema no caso de falhas,

além de possuir uma série de restrições de uso, conforme apresentado na seção 2.3.1 do capítulo 2.

(A) **(B)** **(C)**

Figura 25 – Ilustração de fluxo óptico.

(ZIVKOVIK, 2004:5)

A tećnica de *chroma key* ainda tem a desvantagem de requerer *setup* especial. Quanto a segmentação de cor da pele, é dito que esta técnica depende muito das condições de iluminação, onde uma calibração é necessária para que esta técnica funcione corretamente.

Zivkovik (2004) ainda apresenta um quadro comparativo, contendo uma análise qualitativa ente a diferenciação entre os dois últimos quadros e o fluxo óptico. Segundo este autor, estas são as técnicas que melhor se adaptam ao "mundo real". Esta comparação, focada na funcionalidade e na facilidade de uso, aparece no quadro 4.

Para que o resultado do Quadro 4 fique claro para o leitor, convém comentar a comparação item a item. A primeira característica, sem inicialização, indica que ambas as técnicas não requerem um evento específico para iniciar a captura de movimento, tal como é o caso da diferença do tipo "presença" (vide seção 2.3.1, Figura 6).

A robustez quanto à iluminação significa que ambas as técnicas se adpatam rapidamente a variações desta condição no ambiente, visto que o cálculo é efetuado sempre entre os últimos quadros capturados. Isto é, se a mudança de iluminação ocorreu no quadro (ou *frame*) *F*, o resultado será prejudicado somente nos resultados que utilizam este quadro, afetando a resposta da avaliação entre *(F – 1)* e *F*, e entre *F* e *(F + 1)*.

Quanto a simplicidade de computação, vale mencionar que a diferença entre quadros é consideravelmente mais simples que fluxo óptico, visto que este último, de acordo com o próprio Zivkovic (2004) pode ser considerado como uma extensão do primeiro. O autor deste trabalho de conclusão de curso considera a simplicidade do fluxo óptico média, quando comparada à diferença entre quadros.

Quadro 4 – Comparação entre as técnicas de diferença entre quadros de animação e fluxo óptico.

(adaptado de ZIVKOVIC, 2004:8)

Característica	Diferença entre Quadros	Fluxo Óptico
Sem inicialização	(+)	(+)
Robustez com Relação à Iluminação	(+)	(+)
Simples de computar	(+)	(+)
Detecção de movimento	(+)	(+)
Magnitude do movimento	(+ | -)	(+)
Direção do movimento	(-)	(+)

A detecção de movimentos é robusta em ambas as técnicas, já que diferença entre as imagens sempre revelará as partes diferentes, ou que se encontram em movimento. A magnitude do movimento provém com clareza a partir dos vetores resultantes do fluxo óptico e pode ser estimada com a diferenta entre quadros através da divisão do número de *pixels* em movimento pelo total de *pixels* contidos na região avaliada.

Quanto à direção do movimento, não há técnica conhecida para levantar esta característica partindo simplesmente da diferença entre quadros. Portanto, é esta a característica que define o fluxo óptico como superior à diferença entre quadros quando há a necessidade de saber a direção do movimento em uma sequência de imagens.

A Figura 26 mostra as aplicações teste utilizadas pelo autor para comparar as técnicas do Quadro 4. A primeira (Figura 26.A) utilizou botões estáticos, que eram ativados quando tocados. A segunda, mostrava um menu complexo, com botões distribuídos por toda tela (Figura 26.B). A diferença de usabilidade entre as técnicas, estava na ativação do botão, onde a implementação com fluxo óptico somente aceitava ativação ao efetuar leitura de movimento em uma determinada direção.

(A) **(B)** **(C)**

Figura 26 – Aplicações de prova de conceito do fluxo óptico.
(ZIVKOVIK, 2004:9,10)

A tarefa era ativar o botão verde, que alternava de posição aleatoriamente. O teste reportou maior eficiência do usuário durante a utilização da implementação com fluxo óptico, em ambas as situações. No caso do menu complexo, a diferença entre quadros se mostrou inviável, já que a ativação de um botão na parte superior quase sempre resultará na ativação acidental de algum botão na parte inferior. Isso porque deslocar o braço até a parte superior pode causar leitura de movimento em grande parte da tela.

Outra aplicação mostra uma das principais motivações de uso de fluxo óptico nesse tipo de interface. Na Figura 26.C, crianças têm a tarefa de organizar os chamados *vision-driven gadgets*, de acordo com a forma e a cor. Foi reportado que o uso desse tipo de interface foi realizado de maneira natural e intuitiva. A grande desvantagem do fluxo óptico em relação à diferença entre os dois últimos quadros está no cálculo dos valores de direção e velocidade, que agrega um custo computacional a aplicação.

3.7 SONY EYE TOY

A implementação comercial mais rica em quantidade de aplicações, considerando os trabalhos apresentados neste capítulo, é a Sony Eye Toy (SCEE, 2003), que consiste de uma câmera digital, muito semelhante a uma *webcam* convencional, e que foi desenvolvida para uso no *videogame* Playstation 2. A esmagadora maioria das aplicações desse periférico, utilizam objetos virtuais do tipo botão, conforme descrito na subseção anterior. Entre essas aplicações, pode-se

citar o Eye Toy Play (Figura 27.A e Figura 27.B), que consiste em uma coletânea de mini-jogos baseados na movimentação aplicada em objetos do tipo mencionado.

Na Figura 27.A, o jogador deve se movimentar rapidamente para espantar um enxame de abelhas. A Figura 27.B mostra um jogador lutando contra ninjas virtuais (o jogador deve acertar os ninjas conforme vão aparecendo na tela). Essa ações podem ser realizadas com a redução de dados que calcula a diferença entre os dois últimos quadros de animação. As características extraídas são a posição da área correspondente ao movimento, no caso da luta contra os ninjas, e também da quantificação deste movimento, no caso da tarefa de espantar as abelhas.

Os botões virtuais dos menus dos jogos desenvolvidos para a Eye Toy são ativados através da leitura de movimentação na área que compreende esse botão, durante um determinado período de tempo. Isto é, ao invés de ativar um botão virtual com o pressionar de um botão real no *joystick* do *videogame*, é dada ao jogador a opção de ativá-lo com movimentos da mão, geralmente de forma parecida com o gesto de acenar para dizer adeus. A leitura de movimentos durante um determinado período de tempo é uma forma de contornar a possível ativação acidental provocada na experiência com botões complexos apresentada na subseção anterior.

(A) **(B)** **(C)** **(D)**

Figura 27 – Aplicações do periférico Eye Toy.

(SCEE-Play, 2003) (SCEE-Antigrav, 2003)

A implementação da Eye Toy que mais inovou na interação com o ambiente virtual foi a Eye Toy Antigrav. Neste jogo, o jogador controla um avatar, ao invés de ver sua imagem projetada na tela (Figura 27.C), de forma parecida com a implementação acadêmica Dog Fight. O avatar é provido de uma prancha voadora, semelhante a uma prancha de *snowboarding*, e deve atravessar um pista

ambientada em cidades com *design* futurístico, realizando manobras e coletando itens de aumento de velocidade.

Para controlar o avatar, o sistema rastreia a posição das mãos e da cabeça. Isto é, a característica analisada é a posição desses membros. E para poder realizar o rastreamento da cabeça, é necessária uma etapa de inicialização (Figura 27.D) antes de começar cada partida. Mas esta é realizada de forma bastante rápida, e consiste em apenas três passos onde o jogador deve posicionar o rosto na direção de um objeto virtual que indica quanto as características que descrevem o rosto (provavelmente[1] a presença de olhos e boca) são rastreadas com sucesso.

O problema é que durante a partida, se o jogador variar muito a ângulo de posição da cabeça, em relação ao que foi calibrado, o sistema se perde na localização desta. Não foi divulgado como o sistema realiza tal rastreamento. Apesar de não ser totalmente robusto, o último jogo citado prova que é possível realizar o rastreamento de partes do corpo em ambiente não controlado.

3.8 APLICAÇÕES EM NAVEGADORES DE INTERNET

Algumas aplicações relacionadas aos trabalhos citados também estão disponíveis via *Internet*. Uma coletânea de implementações pode ser testada em (EXTENDED REALITY, 2007). Essas implementações são utilizadas através de navegador, e não necessitam de instalação de *software* adicional, além do *plugin* com a tecnologia Flash 8. Esta tecnologia contém uma API de processamento de imagens (ADOBE, 2005) que pode ser utilizada para criar aplicações semelhantes ao trabalhos que foram citados.

Esta subseção não comentará sobre as etapas de análise de imagens apresentadas na Figura 3. O motivo é o fato que as aplicações aqui descritas não apresentam grandes novidades em relação ao que já foi apresentado nos demais trabalhos.

[1] Diz-se provavelmente, pois esta é uma aplicação comercial e não foi especificado o método de rastreamento.

(A) **(B)** **(C)**

Figura 28 – Aplicações de interfaces naturais para navegadores.
(EXTENDED REALITY, 2007)

A vantagem está na utilização via *Internet*, onde o usuário nada precisa instalar, além do *software* que habilita sua câmera no Sistema Operacional. Uma das desvantagens está no tamanho limitado da tela (entre *QVGA* e *CIF*) utilizado nas aplicações da Figura 28, que acaba por prejudicar a sensação de imersão. Com relação às técnicas utilizadas nas aplicações, é de se supor que utilizem:

1. A segmentação de cor da pele, já que algumas implementações parecem identificar presença sem a necessidade de inicialização;
2. A diferença entre os dois últimos quadros de animação, já que esta é uma técnica simples e bem adaptável a aplicações do “mundo real”;
3. O fluxo óptico, ou alguma alternativa equivalente, visto que algumas implementações proporcionam interação física (simulação de reação física, ao colidir) com objetos virtuais, onde a velocidade do movimento exerce influência na reação do objeto virtual.

A Figura 28 ilustra alguns exemplos que comprovam estas afirmações. A Figura 28.A ilustra uma implementação de Arkanoid, onde o usuário controla as barras que rebatem a bola com os braços. As barras podem ser manuseadas mesmo sem movimentação, o que indica a segmentação da cor da pele, ou técnica de identificação de presença equivalente. O jogo da Figura 28.C mostra uma implementação de limpeza de tela, que é facilmente resolvida com a diferença entre os dois últimos quadros. Na Figura 28.B, o jogador interage com uma bola de basquete, tentando direcioná-la para a cesta. A velocidade e a direção com que o jogador “toca” na bola influenciam na sua trajetória, sugerindo a utilização de fluxo óptico, ou técnica equivalente.

3.9 COMPARATIVO ENTRE AS TÉCNICAS APRESENTADAS

O Quadro 5 apresenta uma contribuição à abordagem qualitativa de (ZIVKOVIC, 2004), ao estender o número de técnicas abordadas com outras levantadas durante a pesquisa literária. Todas as técnicas desta tabela são referentes à etapa de redução de dados, fazendo parte da segmentação dos dados para então possibiltar a extração de suas características, ao não ser pelo fluxo óptico. Esta técnica de processamento de imagens, por ter como resultado um vetor que indica direção e velocidade, é considerada como sendo etapa de extração de características de imagens.

Quadro 5 – Comparação qualitativa entre as técnicas apresentadas.

Característica	**(1) Segmentação com Fundo Arbitrário**	**(2) Segmentação por Cor da Pele**	**(3) Diferença entre Quadros**	**(4) Fluxo Óptico**
Sem Inicialização	(-)	(+)	(+)	(+)
Robustez com Relação à Iluminação	(-)	(+ | -)	(+)	(+)
Simples de Computar	(+)	(+ | -)	(+)	(+ | -)
Detecção de Presença (usuário parado)	(+)	(+ | -)	(-)	(-)
Detecção de Movimento	(-)	(-)	(+)	(+)
Magnitude de Movimento	(-)	(-)	(+ | -)	(+)
Direção de Movimento	(-)	(-)	(-)	(+)
Possibilita a Substituição do Fundo por Cenário Virtual	(+ | -)	(-)	(-)	(-)

Assim como foi feito para o Quadro 4, o Quadro 5 será comentado item a item: primeiramente, a característica de não requerer inicialização apenas é inviabilizada na segmentação com fundo arbitário, já que a captura do primeiro quadro é o primeiro passo necesário a possibilitar a posterior separação do plano de

frente, que nada mais é do que tudo que está diferente quando comparado ao primerio quadro capturado (quadro de referência).

A robustez com relação à mudanças de iluminação é totalmente comprometida na segmentação com fundo arbitrário ao considerar que grandes mudanças nos valores de intensidade dos pixels os classificará como sendo diferentes do quadro de referência. A segmentação da pele utilizada na API deste trabalho já considera possívies variações de iluminação, mas é prejudicada em ambientes muito escuros. As técnicas de movimentação (diferença entre quadros e fluxo óptico) se adaptam à variações de iluminação ao considerar que estas sempre realizam seus cálculos com base nos últimos quadros capturados.

As técnicas de diferenciação entre quadros, (1) e (3), são simples de computar, ao passo que necessitam apenas da realização de uma subtração *pixel* a *pixel*. As demais técnicas são mais complexas, já que se baseiam na computação de um conjunto de equações. A segmentação com fundo arbitrário é mais adequada para identiciar a presença do usuário, já que irá segmentar pele, roupa e demais objetos diferentes do quadro de referência. A segmentação da cor da pele apenas identifica pele e outros ruídos (cores semelhantes à pele pertencentes a outros objetos) e as técnicas de indentificação de movimento não são capazes de perceber quando o usuário está parado.

O contrário é verdadeiro na identificação do movimento. Nesse caso, somente as técnicas (3) e (4) são capazes de realizá-lo. Como foi explicado na seção 3.1.6, é possível quantificar o movimento com a diferença entre quadros, mas o ideal é extrair este valor do vetor que é resultado do fluxo óptico. A direção do movimento apenas é possível (através das operações que foram apresentadas) com este último. Um cenário virtual deve ocupar todo o espaço do plano de fundo. A segementação com fundo arbitrário gera ruídos que não são incovenientes para este fim, a segmentação da pele frequentemente não irá classificar a roupa do usuário como plano de frente e as demais técnicas não são adequadas para este fim.

Não há como eleger qual das técnicas é a melhor, considerando que a algumas das implementações apresentadas criaram formas de interação a partir de um conjunto dessas técnicas. Uma API que tem o objetivo de facilitar implementações como as que foram apresentadas neste capítulo deve considerar todas as propriedades do Quadro 5 como um conjunto mínimo de funcionalidades.

3.10 CONSIDERAÇÕES FINAIS

Este capítulo comprova que ambas as pesquisas, em implementações acadêmicas e implementações comerciais são importantes quando o objetivo é o levantamento de funcionalidades que precisam ser implementadas em uma biblioteca. Após estudar algoritmos de visão computacional e analisar a implementação dos trabalhos acadêmicos, não fica difícil fazer uma engenharia reversa, a nível de funcionalidade, nas implementações comerciais. Isto é, pode-se afirmar que os esses trabalhos se utilizam de técnicas muito semelhantes as que foram adotadas como solução nos artigos acadêmicos.

A partir do estudo das implementações que foram lançadas como produto no mercado, pode-se identificar boas práticas de implementação que vão além do conjunto de algoritmos utilizados. Como essas soluções têm as vendas como objetivo final, suas implementações estão focadas, além da estética do produto, na facilidade de uso do sistema. A partir daí se pode extrair as situações e tipos de implementação que são ou não convenientes, do ponto de vista da usabilidade do sistema.

Por exemplo, uma questão a ser mencionada é o provável desconforto sentido pelo jogador durante o jogo *Camera kombat*, ao ter que virar a cabeça em direção ao monitor para receber *feedback* visual. Nesse tipo de interface natural, as implementações devem ser idealizadas com o jogador voltado para a tela, tanto em interações que projetam a imagem espelhada do usuário quando as que utilizam avatares. Isso auxiliará na garantia de *feedback* visual instantâneo, sem causar desconforto.

É claro que existem outras questões, como não forçar a permanência em uma posição desconfortável durante um longo período de tempo (como o *Arkanoid* apresentado na subseção 3.1.8). Isto é, uma análise completa de usabilidade é tarefa complexa, e foge do escopo deste trabalho.

As implementações que objetivam rastreamento 3D, sem uso de marcadores, geralmente requerem condições que dificultam o uso do sistema por usuários comuns. Essas condições podem incluir ambiente especialmente controlado, uso de *chroma key* (exige uma tela de fundo azul, verde ou vermelha) ou requisitos mínimos

de *hardware* muito elevados[1]. Além disso, o conjunto de algoritmos envolvido no processo da correta extração da posição em três dimensões dos membros do corpo humano é altamente complexo, muitas vezes incluindo abordagens que não se utilizam apenas de equações, mas sim de métodos de aprendizado de máquina (*machine learning*), e/ou a utilização de um banco de dados criado em trabalhos previamente desenvolvidos. O tempo de estudo e implementação desse tipo de funcionalidade excederia o escopo de implementação de um trabalho de conclusão de curso, caso esta funcionalidade também viesse a fazer parte da API Move-In.

Porém, existem soluções simples (não envolvem apredizado de máquina e um banco de dados previamente criado) que não são robustas para uma grande quantidade de casos, como em ambientes não controlados, mas acabam sendo razoáveis em casos específicos. Por exemplo, o caso do desenho virtual 3D, apresentado na subseção 3.1.5, parece resolver o problema que propõe de forma suficientemente robusta. É esse tipo de solução (simples, para atender casos específicos) que deverá ser buscada no cálculo da posição de partes do corpo proposta como parte dos requisitos da biblioteca de funções que resultará deste trabalho.

[1] Um exemplo foi citado ao final da seção 3.1.4.

4. HARDWARE E SOFTWARE UTILIZADOS

As seções deste capítulo têm o objetivo de detalhar características do *hardware* (*webcams*) e *software* utilizados neste trabalho. Para tanto, as características básicas das câmeras ilustradas no Quadro 3 serão expostas, conforme informação obtida a partir do manual de instruções e da experiência de utilização deste tipo de *hardware* durante o desenvolvimento deste trabalho. Também serão enumeradas as vantagens sobre a decisão de escolha da biblioteca de processamento de imagens e visão computacional (*OpenCV*) que serve de base para a implementação das técnicas apresentadas.

4.1 HARDWARE UTILIZADO

A *webcam* utilizada para fins de teste das funcionalidades desenvolvidas é uma *LG LIC-200*. As especificações técnicas desta câmera estão descritas na Tabela 6 e foram extraídas do manual de instruções que a acompanha. O único parâmetro que não foi discutido no capítulo 2, é o formato de cor I420. Este código é o espaço de cor YUV na composição de resolução *4:2:0*[1] (FOURCC, 2007).

Esta é uma *webcam* de baixo custo. Isso significa que as funcionalidades da biblioteca que forem desenvolvidas visando a utilização deste dispositivo poderão ser usadas para criar aplicações destinadas ao usuário geral.

Como foi explicado na subseção 2.4.2, a limitação da taxa de transferência de dados provida pelo padrão USB 1.1 não permite a transmissão de imagens não compactadas, a não ser em resoluções inferiores a QVGA (*320x240*). O problema mais crítico em relação a esta câmera é o fato de que ela não permite que o ganho automático dos níveis de branco seja desabilitado. Isto é, a implementação da diferença de quadros com incialização do quadro de referência é inviabilizada para algumas situações. Diz-se “para algumas situações” porque o comportamento do ganho automático de adaptação de iluminação é difícil de prever, já que este varia

[1] Conforme explicado na subseção 2.2.1, o modelo de representação *4:2:0* contém *N bits* para o valor de luminância e *N/4 bits* para os dois valores de crominância. No caso do *I420*: *Y = 8bits*, *U = 2 bits* e *V = 2 bits*.

entre as câmeras, varia para a quantidade de objetos em movimento dentro da área de foco e também para a distância desses objetos em relação à câmera.

Tabela 6 – Características da *Webcam LG LIC-200*.

WEBCAM LG LIC - 220			
Resolução	VGA(640x480)	CIF(352x288)	SIF(320x240)
	QCIF(176x144)	QSIF(160x120)	
FPS	15 FPS@VGA, 30 FPS@CIF		
Transferência	USB 1.1		
Compressão	JPEG(ISO/IEC 10918-1)		
Sensor	CMOS		
Formatos de Cor	RGB 24 bits, I420		

O que acontece é que o quadro inicial é capturado sem nenhum objeto de interação no campo de visão da câmera. Então, ao iniciar a interação, quando o usuário passa a estar à frente da câmera, esta adapta seus níveis de branco, causando a quase totalidade do quadro atual (usuário e plano de fundo) a ser diferente do quadro de referência. Isso resulta em uma segmentação do objeto de interesse com ruídos, que resultam em falsos positivos. Apesar disso, a mudança de ilimunição não é brusca caso o usuário se encontre a dois ou três metros longe da câmera.

(A) **(B)**

Figura 29 – *Hardware* utilizado.

No entanto, a impossibilidade de desabilitar a função que torna a técnica de presença com diferença entre quadros inviável não é uma característica presente em todas as *webcams*. Quando a utilização da câmera ilustrada na Figura 29.A não for possível, devido ao ajuste automático dos níveis de branco para situações onde o objeto de interesse se encontra próximo à câmera (distância inferior à 2 metros),

outra *webcam* foi utilizada. Este *hardware* (Figura 29.B) é o mesmo utilizado na implementação comercial da Sony (SCEE, 2003), descrita na subseção 3.1.7.

As especificações técnicas deste *hardware* são equivalentes às da câmera da LG (Figura 29.A). Isso foi constatado através da experiência de utilização desta *webcam*. Também há uma compensação automática dos níveis de branco para esta câmera, mas isso ocorre somente quando esta é inicializada.

4.2 SOFTWARE UTILIZADO

O *software* utilizado para desenvolvimento da biblioteca de interação natural é o Intel Open Computer Vision Library – OpenCV (OPENCV, 2006). Para outras possíveis alternativas, favor consultar (DE SOUZA, 2005). Uma alternativa para uso em navegadores é a API de processamento de imagens para Flash 8 (ABOBE, 2005). Porém, esta API apresenta um conjunto de funcionalidades bem menor quando comparada às da OpenCV, além de fazer parte uma aplicação comercial. Dentre os motivos para escolha da biblioteca OpenCV estão:

- A eficiência do código, escrito em linguagem C;
- É uma biblioteca de código aberto e de licença que permite utilização em projetos acadêmicos e em projetos comerciais;
- É de simples instalação nos sistemas operacionais *Windows XP* e *Linux* (ambientes de teste de implementação deste trabalho);
- Pode ser integrado com *gcc* e *g++* (compiladores de código aberto), possibilitando um ambiente de produção totalmente *open-source*;
- Possui documentação oficial uma quantidade razoável de exemplos disponíveis na *Internet* que podem servir de base de aprendizado. Nesse caso, a maior fonte de pesquisa recomentada é o grupo de discussão da biblioteca, que agrega mais de trinta mil membros (OPENCV YAHOO GROUPS, 2000);
- É utilizada por empresas como IBM, Microsoft, Sony, Siemens e Google.

Esta biblioteca (OpenCV) foi especialmente desenvolvida para tratar de processos em tempo real utilizando visão computacional. Possíveis aplicações dessa biblioteca incluem, de acordo com a página oficial (OPENCV OVERVIEW, 2007), *softwares* na área de Interface Humano Computador, vigilância,

entretenimento e robótica. Alguns exemplos de aplicações são: identificação de objetos, reconhecimento de faces e de gestos, rastreamento de movimento e visão artificial de robôs. Maiores detalhes sobre a arquitetura desta biblioteca serão descritos na seção que segue.

4.2.1 Arquitetura de Software da Biblioteca Opencv

A Figura 30 apresenta a arquitetura básica da biblioteca OpenCV (PISAREVSKY, 2007). Ao instalar a biblioteca, o usuário pode começar a se familiarizar com exemplos que mostram funcionalidades dos três principais componentes, CV e CVAUX, MML e HighGui.

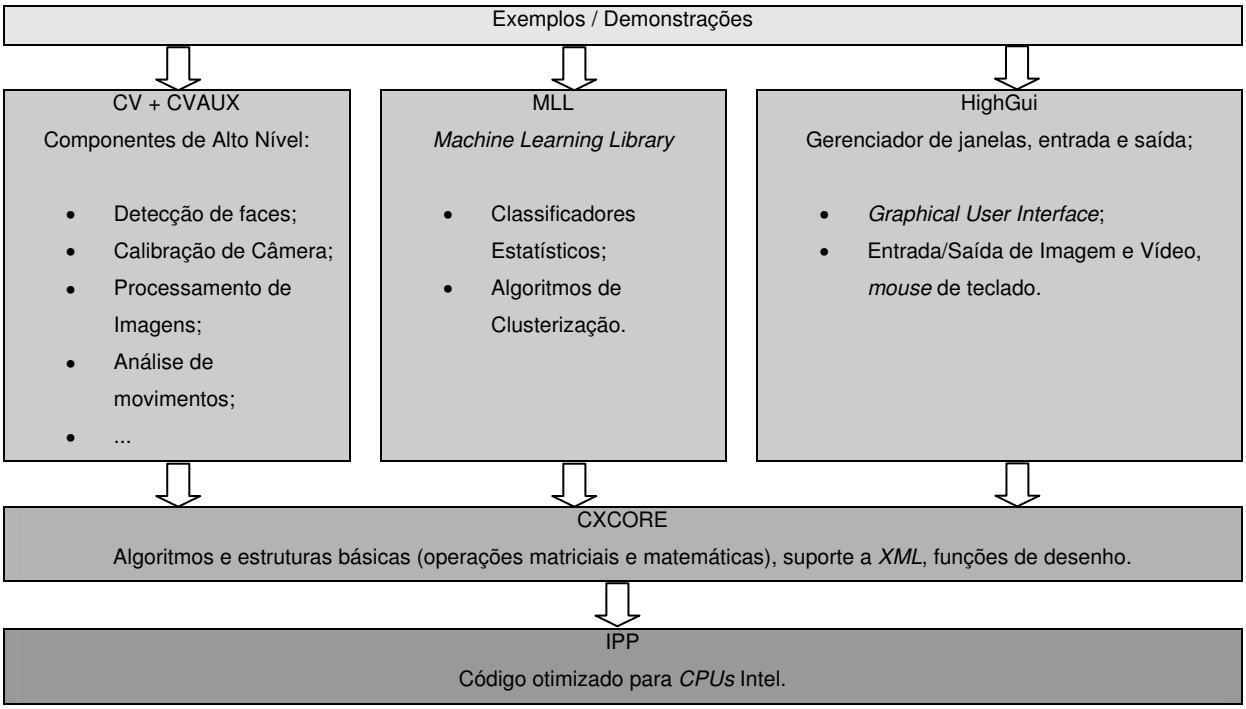


Figura 30 – Arquitetura da Biblioteca OpenCV.
(adaptado de PISAREVSKY, 2007)

CV e CVAUX agrupam as funcionalidades utilizadas na API desenvolvida neste trabalho. Machine Learning Library (MLL) não será utilizada por tratar de aprendizado de máquina, funcionalidade esta que não é requerida pela versão inicial da API. HighGui é um gerenciador de janelas e de entrada e saída básico. É

utilizada nos exemplos de demonstração da API deste trabalho, mas não é recomendada, devido ao pequeno conjunto de funcionalidades que apresenta, para uso em aplicações destinadas ao usuário final. Para tanto, recomenda-se o uso de bibliotecas mais completas e eficientes para este fim, tais como SDL (Simple Directmedia Layer) ou Qt (Cross-Platform Rich Client Development Framework).

Todo esse conjunto de funcionalidades faz uso de estruturas de dados e funções matemáticas básicas contidas em CXCORE. Essas funções foram elaboradas de forma a ficarem com seu desempenho otimizado em CPUs Intel, onde a real vantagem no desempenho somente é sentida quando o compilador da Intel é utilizado para gerar os executáveis. Neste caso, o desempenho pode aumentar de 1.5X até 8.0X, dependendo do conjunto de funções utilizadas (PISAREVSKY, 2007).

4.3 API CVBLOSLIB

A API *Computer Vision Blobs Library*, ou cvBlobsLib, é uma biblioteca de classes destinada à classificação de componentes conexos em uma imagem[1]. Ela também provê funções para manipular, filtrar e extrair resultados a partir dos *blobs* identificados (CVBLOSLIB, 2008). De acordo com autores API cvBlobsLib, o algoritmo de classificação de componentes conexos utilizado é baseado no código de Dave Grossman[2].

A extração de características de imagens binarizadas na API desenvolvida neste trabalho frequentemente requer um único *blob* para efetuar corretamente processos a partir de uma imagem binarizada, como a extração de contornos, do fecho convexo ou dos defeitos de convexidade. Nos exemplos apresentados neste trabalho, a API cvBlobsLib é utilizada para este fim.

A cvBlobsLib, tal como a Move-In, faz uso das funções estruturadas do OpenCV para criar classes em C++ que facilitam o uso de um conjunto de funcionalidades. No caso da cvBlobsLib, essas funcionalidades são a identificação e classificação e separação de *blobs* em imagens coloridas e em níveis de cinza. Nos exemplos de utilização da API Move-In, a API cvBlobsLib é utilizada para identificar

[1] A definição de componente conexo se encontra na subseção 2.5.6.
[2] Não foi encontrada referência formal sobre tal algoritmo, além do comentário de Grossman na página da API.

o maior blob em uma imagem binária resultante do processo de segmentação do objeto de interesse (exemplo na seção 6.1).

4.4 CONSIDERAÇÕES FINAIS

A justificativa de utilização da biblioteca OpenCV está no fato de que esta é a biblioteca mais completa e eficiente disponível para uso em meio acadêmico (e também comercial, já que sua licença é do tipo BSD[1]). Ela foi projetada para atender propósitos de solução em processamentos de imagens e visão computacional de uma forma geral, através de componentes que visam à reutilização de código. A primeira versão da API de captura de movimentos não utilizará as funções do módulo MML, de aprendizado de máquina e HighGUI, para exibição dos resultados em janelas (Figura 30). É possível que as funcionalidades do módulo MML sejam utilizadas em próximas versões da API. Com relação ao módulo HighGui, ele é utilizado na exibição dos resultados apresentados no capítulo 6, mas não é diretamente usado no código das classes que compõem da API Move-In.

No entanto, pela extensão do conjunto de funções e pelo nível de abstração (linguagem estruturada) se percebe que o tempo de aprendizado para a utilização da biblioteca, e identificação das funcionalidades que podem ser utilizadas para captura de movimentos é maior quando comparada à utilização de uma API orientada a objetos e especialmente voltada para esta finalidade. Sendo assim, a API desenvolvida neste trabalho visa minimizar esse tempo através do emprego da orientação a objetos aplicada sobre um conjunto de funções da biblioteca OpenCV. O objetivo é agrupar e organizar componentes da biblioteca para atender ao conjunto de funcionalidades que foi levantado no capítulo anterior. A forma como isso foi realizado está descrita no capítulo que segue.

[1] A licença BSD (*Berkeley Software Distribution*) revisada pode ser considerada de domínio público, e permite o uso e/modificação de seu *software* sem restrições. No caso da API OpenCV, a única restrição é que o a aplicação que a utilizar deverá levar a licença em conjunto com os demais arquivos da distribuição do *software*.

5. PROPOSTA DE IMPLEMENTAÇÃO DA API MOVE-IN

O capítulo 3 apresentou aplicações de variados autores, tanto no meio acadêmico quanto no meio comercial. As aplicações do meio acadêmico tendem a abordar um problema específico. Aplicações comerciais tendem a ser mais completas, mas a natureza da licença (código fechado) e alto custo de aquisição do *hardware* e *software* necessários restringe seu uso em larga escala (desenvolvedores), e restringe também o acesso à tecnologia (usuários).

A API desenvolvida neste trabalho será referenciada pelo nome Move-In. A pronúncia desta expressão é a mesma de "*moving*", referente à movimento. Outra interpretação seria "mova-se no interior do espaço focado pela câmera", ou, simplesmente "*move-in*". Ou ainda, pode-se considerar este nome como referente à movimento ou "*move*" como entrada de dados, "*input*", ou simplesmente "*in*".

O nome Move-In foi elaborado desta forma para referenciar a API através de um nome simples e fácil de lembrar, sem estar diretamente relacionado com questões sobre processamento de imagens ou visão computacional. Isso porque a API está disponível para *donwload* na Internet (Move-In API, 2008), e a maioria dos usuários desta – desenvolvedores de *software* – provavelmente estará mais interessado nas funcionalidades da API, e não na teoria que a sustenta.

Quanto a justificativa da escolha das funcionalidades, pode-se afirmar que, para fazer parte desta versão da API, uma funcionalidade deve ser simples de implementar[1] e deve minimizar o custo computacional necessário. A primeria restrição atende a possibilidade implementação durante o tempo de execução deste trabalho de conclusão de curso e a segunda está relacionada a execução em tempo real necessária nas aplicações a que a API se destina.

5.1 FUNCIONALIDADES BÁSICAS

O sistema de captura de movimentos desenvolvido neste trabalho consiste em uma biblioteca de funções que pode ser usada em diversos tipos de aplicações. Esta biblioteca possibilita a reprodução de muitas das soluções apresentadas no

[1] Procurou-se evitar estruturas de dados complexas, e/ou que envolvem aprendizado, tal como a árvore hierárquica de posições descrita na subseção 3.1.4.

capítulo 3. Exemplos para cada uma das principais funcionalidades são utilizados para fins de validação da biblioteca.

O conjunto de funcionalidades de propósito geral desenvolvidos para a API Move-In pode ser resumido no esquema da Figura 31. A tela de interação consiste na imagem de entrada, ou quadro capturado com a *webcam*. Uma região de interesse (*Region of Interest* - ROI) é uma área retangular de largura e altura arbitrárias, e que pode estar localizada em qualquer parte da tela de interação. A tela de interação pode conter *N* regiões de interesse. Regiões de interesse podem ser posicionadas em qualquer parte da tela, podem ser transladadas, podem mudar de tamanho, podem ser criadas ou destruídas a cada quadro de animação.

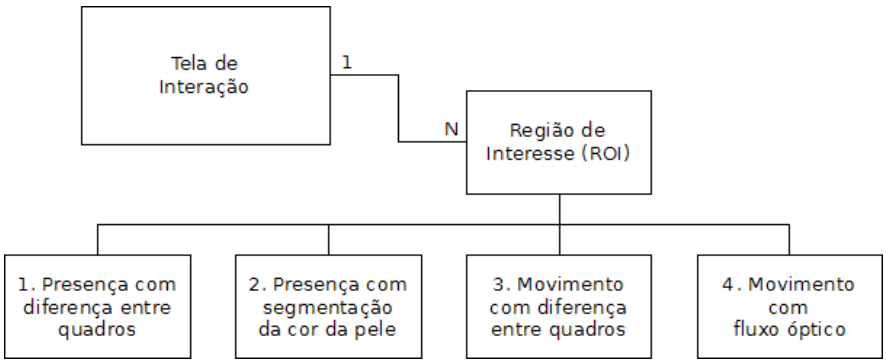


Figura 31 – Funcionalidades básicas da API Move-In.

Essas regiões de interesse são como ouvintes (*listeners*) que varrem a área compreendida pela ROI nos quadros de entrada, à procura de sinais de interação. As interações identificadas deverão ser associadas ao disparo de alguma ação. A ação disparada irá depender exclusivamente do propósito das implementações que fizerem uso da biblioteca. As ROIs podem assumir uma de quatro funcionalidades:

1. Presença com diferença entre quadros: é a subtração de quadros que requer inicialização. Através dessa funcionalidade é possível segmentar a silhueta completa usuário. Com esta técnica, é possível identificar a presença do usuário, ainda que este permaneça totalmente parado;
2. Presença com segmentação da cor da pele: segmenta a região exposta do corpo do usuário. Não requer inicialização. Identifica presença e não movimento, mas tende a ignorar as partes do corpo do usuário cobertas por

roupas. Produz falsos positivos quando há elementos do cenário de cor semelhante à cor da pele;

3. Movimento com diferença entre quadros: é a diferença entre os dois últimos quadros de animação. Apenas é possível identificar o usuário quando este está em movimento. É possível quantificar o movimento, mas esta técnica não extrái dados de direção;
4. Movimento com fluxo óptico: é uma extensão da técnica anterior, que permite estimar tanto a direção quanto a magnitude do movimento.

A necessidade da criação dessas ROIs foram identificadas a partir do estudo dos trabalhos apresentados no capítulo 3. A arquitetura de organização das funcionalidades básicas está descrita com mais detalhes nas seções a seguir.

5.2 ARQUITETURA DA API

O diagrama de classes da API Move-In pode ser conferida na Figura 32. Os métodos e atributos foram omitidos por questões de espaço e para facilitar o entendimento das relações entre as classes. Também estão omitidos a maior parte dos relacionamentos entre as classes do núcleo do sistema e classes auxiliares e de gerenciamento de ROIs. Uma discussão mais detalhada sobre esses agrupamentos de classes – módulos da API – segue com as próximas subseções.

Na Figura 32, pode-se observar que todas as funcionalidades estão centralizadas na classe *Screen*, que trata da implementação da tela de interação da Figura 31. O gerenciamento desses requisitos através de uma única classe deverá facilitar bastante o processo de implementação, já que o usuário poderá dedicar sua atenção ao funcionamento dos métodos que correspondem às classes núcleo do sistema.

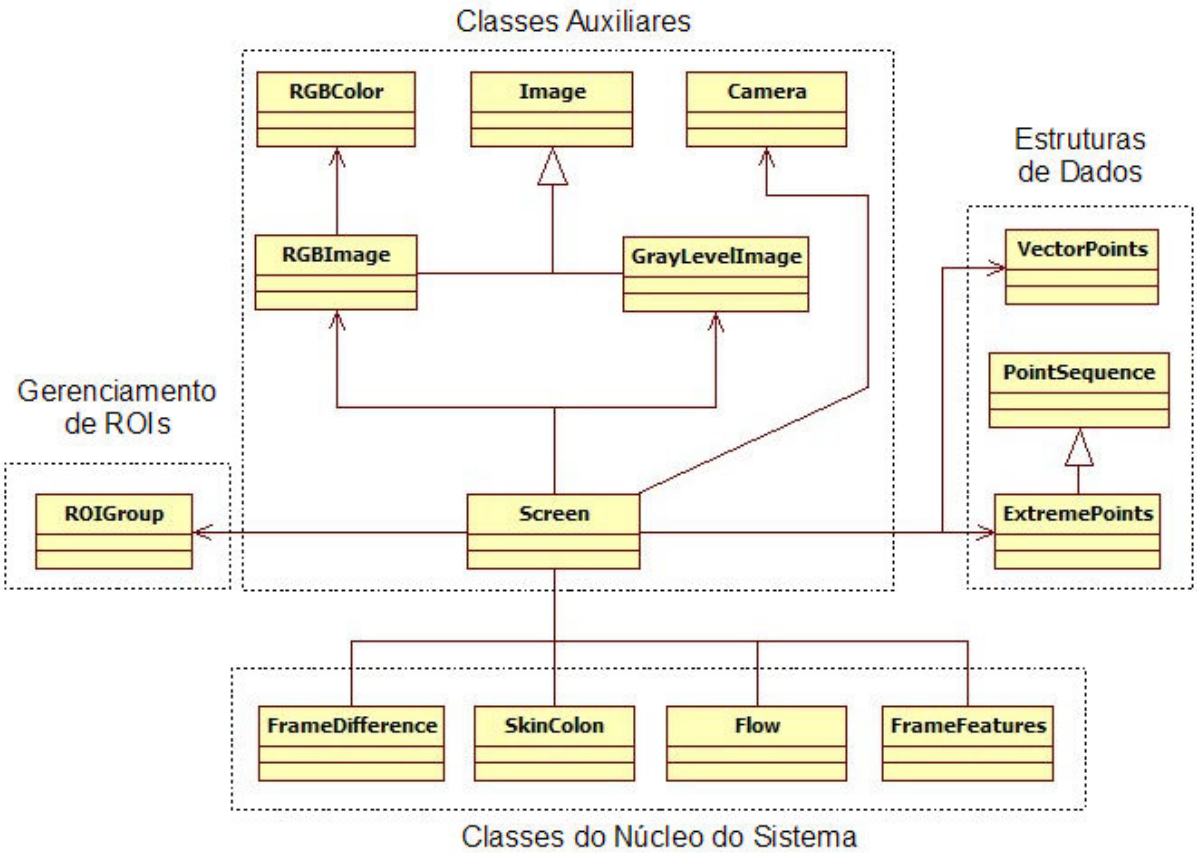


Figura 32 – Diagrama das principais classes da API Move-In.

Este estilo de programação atende ao *design pattern* conhecido como *Facade* (GAMMA et al, 1995). Sua estrutura básica está ilustrada na figura 30. O propósito do padrão Facade é prover uma interface unificada para um subsistema de forma a definir uma interface de alto nível, aumentando assim a facilidade de uso.

De acordo com Gamma et al (1995), este padrão é adequado quando (1) há necessidade de implementação de uma interface simples para um sistema complexo, (2) quando há muitas dependências entre os clientes e as classes do sistema e (3) quando se pretende adicionar uma camada de abstração superior ao sistema alvo.

Sendo assim, este padrão de *design* é perfeitamente adequado para a API Move-In ao se considerar que, para trabalhar com as funcionalidades básicas apresentadas na Figura 31, é necessário gerenciar a entrada de imagens provenientes da *webcam*, imagens coloridas e binárias e um conjunto de sub-regiões de interesse dinâmico, isto é, que pode criar, modificar ou excluir ROIs em tempo de execução.

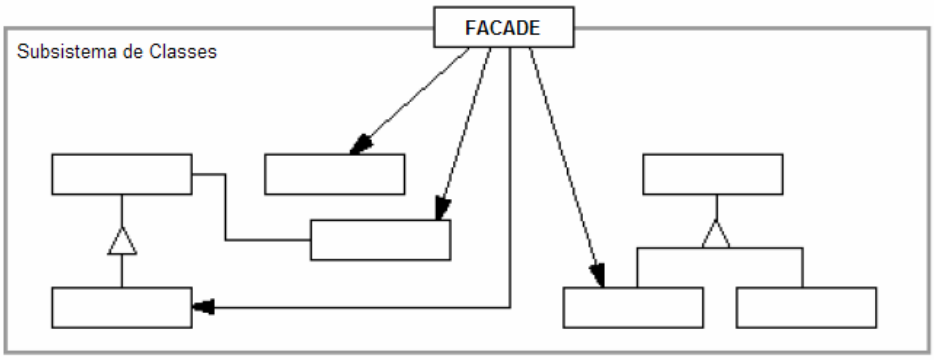


Figura 33 – Estrutura do padrão *Facade*.
(adaptado de GAMMA et al, 1995)

Uma alternativa à utilização do padrão *Facade* é utilizar as classes do núcleo da API Move-In em separado, sem a necessidade da utilização da classe *Screen* como ponto de gerenciamento. Isto é, o núcleo do sistema pode ser usado com exclusividade no caso do usuário necessitar utilizar a API em conjunto com uma aplicação pré-existente da biblioteca OpenCV. Isso ocorre porque as classes do núcleo do sistema são métodos estáticos que aceitam imagens de entrada tanto do tipo empregado pelo OpenCV (*IplImage*) quando do tipo criado para uso com a API Move-In (*Image*, *RGBImage* e *GreyLevelImage*, que são abstrações implementadas sobre a estrutura *IplImage*).

Toda a API foi projeta sob o conceito de orientação a objetos. As classes foram construídas sobre o paradigma estruturado de funções da versão *1.0* do OpenCV. A orientação à objetos, no caso da API Move-In, tem como objetivo facilitar o entendimento da dependência entre os componentes da biblioteca de interação com *webcams* e minimizar a quantidade de chamadas à métodos (ou funções, no caso do conceito de linguagem estruturada) para fazer uso das classes do núcleo do sistema.

As seções a seguir irão detalhar a utilidade dos quatro principais módulos do sistema. Em resumo, as classes auxiliares são abstrações em orientação a objetos das funções estruturadas da biblioteca OpenCV. Há métodos dessas classes que agrupam um conjunto de funções para facilitar o acesso às funcionalidades de manipulação de imagens e câmeras providas pela OpenCV, outros são adaptações da linguagem estruturada para a orientação a objetos.

A classe de gerenciamento de ROIs manipula conjuntos de regiões de interesse. Essas regiões são sub-janelas dentro da tela principal, e que estão associadas a uma das classes do núcleo do sistema. Essas últimas são responsáveis por prover o resultado de muitas das técnicas apresentadas ao longo deste trabalho. Por fim, muitos desses resultados serão armazenados nas estruturas de dados especialmente criadas para guardar e trabalhar com as características extraídas das imagens de entrada.

5.2.1 Classes Auxiliares

A principal característica envolvendo as classes auxiliares está no fato que elas são abstrações de orientação a objetos aplicadas sobre as funções disponibilizadas pela biblioteca OpenCV. A funcionalidade de cada uma dessas classes consiste em:

- *RGBColor*: armazena os componentes do padrão de cor RGB;
- *Image*: classe que facilita o acesso a componentes da estrutura *IplImage*, padão de imagem do OpenCV;
- *RGBImage*: imagem colorida, no esquema de cor RGB, derivada a partir de *Image*;
- *GreyLevelImage*: imagens de um único canal (*single channel*) devem ser declaradas com esta classe. Compreende tanto imagens binárias quanto imagens em níveis de cinza;
- *Camera*: conecta-se à *webcam* e obtém imagens a partir dela;
- *Screen*: classe que centraliza as operações de toda a API Move-In;

GreyLevelImage abrange tanto imagens em níveis de cinza quanto imagens binárias pelo fato de que a biblioteca OpenCV não permite a criação de imagens com apenas *1 bit* de profundidade. A classe *Screen* aborda todos os atributos e métodos necessários para lidar com os quatro tipos possíveis de conjuntos de ROIs definidas pela classes do núcleo do sistema e, portanto, pode ser considerada como a implementação do esquema proposto pela Figura 31, que mostra as funcionalidades básicas do sistema relacionadas à tela de interação.

5.2.2 Classe de Gerenciamento de Regiões de Interesse

O gerenciamento das sub-regiões de interesse é administrado pela classe Screen através da classe ROIGroup. Esta classe deve ser instanciada para que as sub-regiões de interesse possam ser adicionadas à instância da classe e, a partir daí, esta instância pode ser utilizada junto com chamadas aos métodos das classes do núcleo do sistema. Maiores detalhes, com exemplo de implementação, podem ser conferidos na seção 6.2.

5.2.3 Classes de Estruturas de Dados

Foram criadas duas estruturas de dados auxiliares para faciliar o acesso, organização e modificação de algumas das características que podem ser extraídas das imagens de entrada. Na versão da API concluída durante a finalização deste trabalho, eram três as estruturas de dados auxiliares:

- *PointSequence*: consiste em uma sequência de pontos que é utilizada para armazenar contornos, fecho convexo ou qualquer outra sequência de pontos;
- *ExtremePoints*: extende a classe anterior para conter pontos que podem ser classicados como côncavos ou convexos. Esta classificação é extremamente importante para a identificação de certas poses ou posturas das mãos e do corpo humano.
- *VectorPoints*: guarda a sequência de pontos iniciais e finais que correspondem aos vetores identificados pela técnica de movimento através de fluxo óptico.

As classes apresentadas nesta subsessão são traduções para orientação à objetos de formas de armazenamento proporcionadas pela API OpenCV. Mais precisamente, são listas de pontos (*CvPoints*), que são armazenados em uma sequência (*CvSeq*) e que ocupam uma determinada quantidade de memória (*CvMemStorage*). Em acréscimo a isso, o conjunto de pontos agrupados pode ser simplicado, para trabalhar com uma quantidade menor de dados, ou modificado para melhorar o resultado visual da imagem resultante.

Deve-se observar que estas classes estão relacionadas a características das imagens. Já as classes auxiliares, salvo a classe *Screen*, estão relacionadas à aquisição e armazenamento dessas imagens. É por esse motivo que essas classes foram consideradas como um módulo separado ao das classes auxiliares.

5.2.4 Classes do Núcleo do Sistema

As classes que foram apresentadas até agora servem para facilitar o desenvolvimento de aplicações para interatividade com *webcams* através da biblioteca OpenCV. A real funcionalidade da API desenvolvida neste trabalho está nas classes que compõem o núcleo do sistema. Essas classes são compostas por métodos estáticos. Esses métodos recebem referências para imagens previamente alocadas e realizam operações a partir dessas referências.

Além da escolha pelo conjunto de métodos estáticos, existe também o fato de que as classes do núcleo do sistema não possuem atributos e que os métodos dessas classes somente retornam estruturas de dados conhecidas pela biblioteca OpenCV. Isto se deve à facilidade de uso tanto com a o paradigma orientado a objetos da API Move-In tanto com o paradigma estruturado da biblioteca OpenCV.

Estas afirmações apenas não são válidas para a classe *Flow*, já que esta exige inicialização no caso de rastreamento de pontos específicos. Apesar disso, a ação de instanciar a classe *Flow* também está centralizada nos métodos da classe *Screen* e acaba por ser transparente ao programador.

No caso do uso completo das classes da API Move-In, as funções são gerenciadas pela classe *Screen*. Mas também há a opção de utilizar somente as classes do núcleo do sistema em conjunto com outro projeto desenvolvido em linguagem C a partir da biblioteca OpenCV. Tal projeto deverá ter a IplImage como estrutura de dados de armazenamento de imagens, e não será necessária uma conversão para uma classe da API Move-In que a estrutura IplImage seja utilizada como imagem de entrada.

Para que esta flexibilidade de utilização seja alcançada, todos os métodos das classes que compõem o núcleo do sistema têm suas assinaturas duplicadas para aceitar tanto a estrutura *IplImage*, quanto as classes *RGBImage* e

GreyLevelImage. Por exemplo, o método *skinColorRGB* da classe *SkinColor* possui duas assinaturas, a saber:

- *static void skinColor(RGBImage *source, GreyLeveImage *result)* e;
- *static void skinColor(IplImage *source, IplImage *result)*;

As funcionalidades de cada uma das classes do núcleo do sistema são:

- *FrameDifference*: calcula a diferença entre dois quadros de animação. Deve ser utilizada tanto para a técnica de presença quanto para a técnica de movimento, pois ambas as técnicas são efetuadas através da subtração entre duas imagens. Apesar do sentido ambíguo, existem métodos na classe *Screen* que gerenciam as imagens e métodos necessários para a correta utilização;
- *SkinColor*: identifica a cor da pele a partir de uma imagem de entrada. Armazena o resultado em uma imagem binária;
- *Flow*: A partir de duas imagens de entrada, calcula informações de movimento. Armazena os resultados em um vetor que corresponde ao movimento em uma ROI;
- *FrameFeatures*: classe responsável pela extração de características em imagens binárias provenientes do resultado de processamento dentro de *FrameDifference* e *SkinColor*, classes essas que são responsáveis pela segmentação do objeto de interesse.

As duas primeiras classes podem ser consideradas como responsáveis pela redução de dados (Figura 3, seção 2.2). A última classe, *FrameFeatures*, possui o papel de aquisição das características das imagens resultantes do processamento realizado em *FrameDifference* e *SkinColor*. A classe *Flow*, por conter métodos de cálculo de fluxo óptico, já é responsável tanto pela redução de dados quanto pela extração de características.

A classe *FrameFeatures* terá o seguinte conjunto de funcionalidades, cujos atributos descritos a seguir são retornos dos métodos estáticos desta classe:

- *activeArea* (proporção da área afetada): é quantidade de *pixels* brancos[1] dividido pelo total do número de *pixels* da sub-região de interesse. Isso dá a

[1] *Pixels* cujos valores na imagem binária são iguais a 255 (vide subseção 2.5.2). Representam o objeto de interesse na imagem segmentada.

quantificação (ou porcentagem) de presença, movimento ou cor da pele na sub-região;

- *centroid* (centróide): a localização média aproximada da região afetada;
- *contours* (contornos): identifica os contornos de uma imagem binária;
- *convexHull* (fecho convexo): o conjunto de vértices que compõem o fecho convexo;
- *extremePoints* (pontos extremos): ao passo que o fecho convexo identifica pontos extremos de convexidade em um conjunto de pontos, esta característica também identifica os pontos extremos de concavidade em um conjunto de pontos que corresponde aos contornos da imagem.

Isso conclui a argumentação sobre as funcionalidades da API para redução de dados e extração de características. A próxima seção objetiva mostrar como as estruturas apresentadas devem ser trabalhadas na organização dó código fonte.

A análise das características, que é dependente do propósito da aplicação final, está apresentada como um conjunto de propostas ou hipóteses na seção 5.4, onde os resultados da implementação são mostrados no capítulo 6. Dessa forma, a seção 5.4 pode ser entendida como proposta de validação das funcionalidades da API.

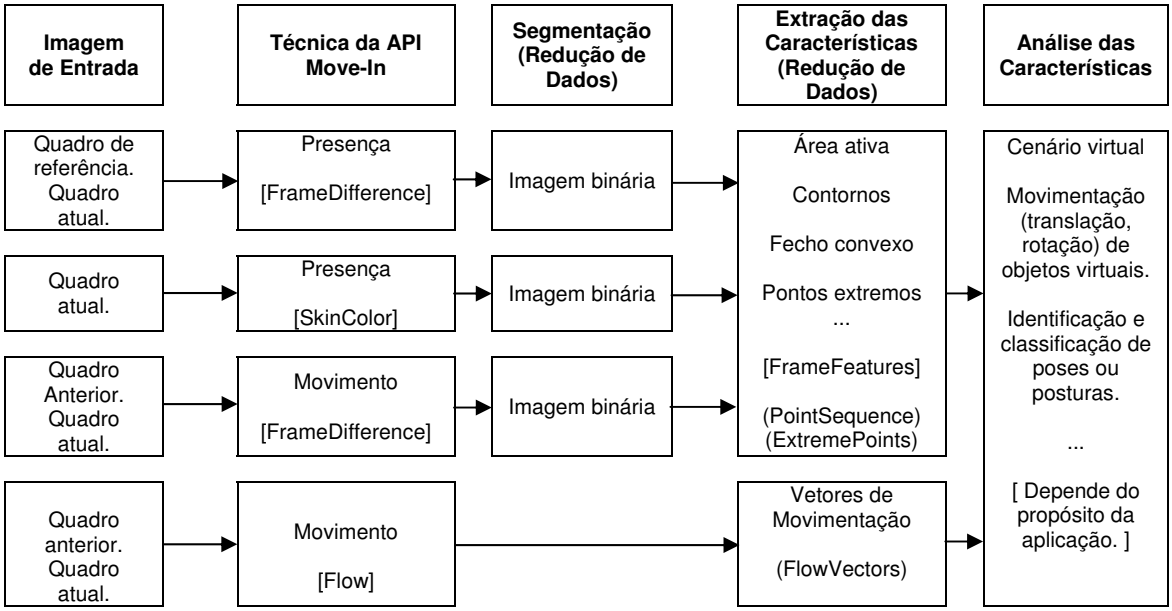
5.3 ORGANIZAÇÃO E USO DA API MOVE-IN

O Quadro 7 mostra a sequência de ações ou passos necessários para a utilização da API Move-In durante a interação com uma sequência de imagens. Através deste quadro se pode perceber as principais semelhanças e diferenças entre as técnicas utilizadas. As semelhanças estão nas técnicas que utilizam as classes *FrameDifference* e *SkinColor*. Essas classes produzem uma imagem binária com o objeto de interesse segmentado. Esta imagem serve de entrada para a classe *FrameFeatures*, que tem a função de extrair características destinadas ao propósito da aplicação.

A diferença está na técnica de movimento através da classe *Flow* que é responsável por extrair características relacionadas à imagem de entrada. No Quadro 7 os nomes das classes entre colchetes representam uma das classes do

núcleo da Move-In, enquanto que os nomes entre parênteses representam as estruturas de dados criadas para armazenar os pontos da imagem de entrada que correspondem às características.

Quadro 7 – Organização e uso da API Move-In.



A função da API Move-In é a extração das características da imagem. O que será realizado a partir da análise desses características depende da aplicação final. Os exemplos que aparecem nas seções do capítulo 6 são funcionais. Isto é, para construir uma aplicação completa, com gráficos 2D ou 3D avançados e sistemas de física e colisão, a Move-In deverá ser utilizada em conjunto com outras bibliotecas.

A Figura 34 mostra como estão organizados os arquivos fonte (os arquivos que contêm a função *main*) que constituem o conjunto de exemplos de utilização da API Move-In. Esta organização pode ser tomada como base para criação de novos projetos. Primeiramente, durante a etapa de inicialização, a classe *Screen* deve ser instanciada. Nesta etapa (1) é feita a conexão com a câmera e as imagens de entrada são criadas. Erros durante este processo, como câmera não encontrada ou memória insuficiente são retornados em forma de excessões.

Nas etapas 2 e 3, as estruturas de dados descritas na subseção 5.2.3 e o conjunto de regiões de interesse devem ser instanciadas. As estruturas de dados se fazem necessárias ao trabalhar com características referentes aos contornos e

vetores de movimentação. Referências dessas estruturas são passadas com chamadas aos métodos das classes *FrameFeatures* ou *Flow*. As ROIs delimitam o escopo de trabalho das técnicas que aparecem no Quadro 7.

A etapa 4 inicia o *loop* principal. Deve-se atualizar a classe *Screen* com o método *Screen::update*, para capturar um novo quadro a partir da câmera e atualizar o quadro anterior com a cópia do quadro atual.

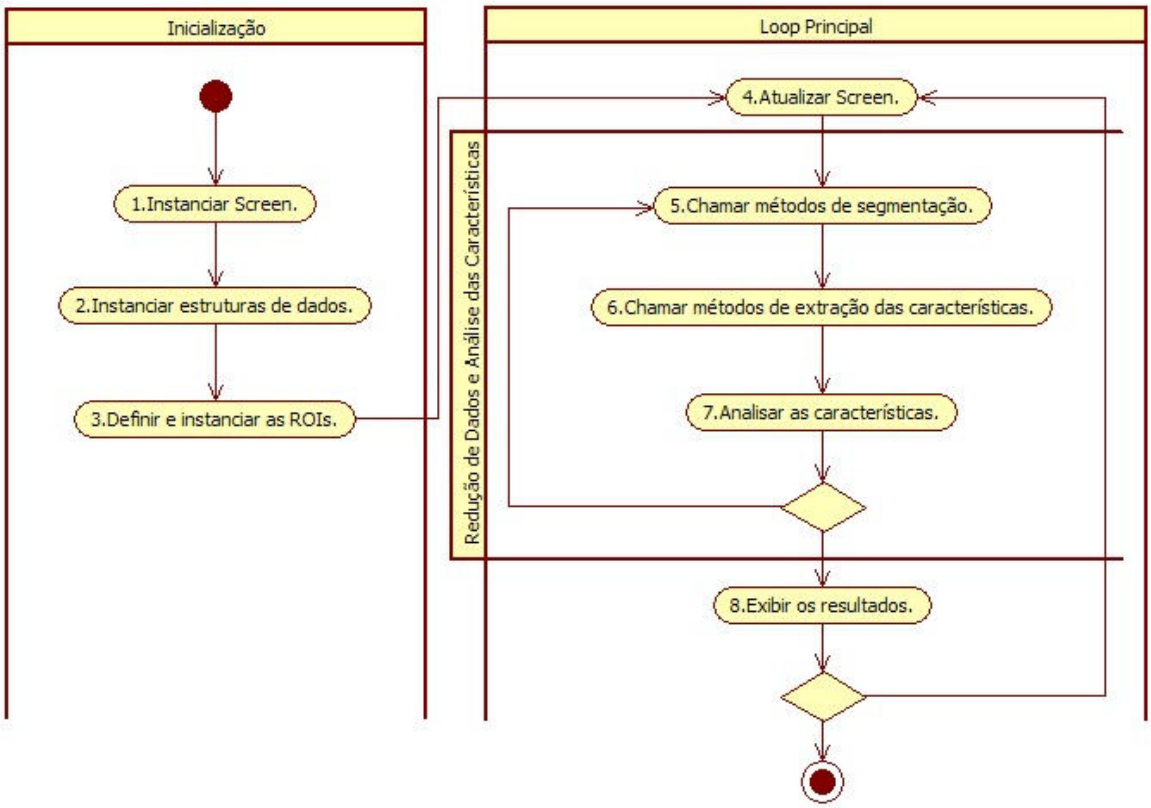


Figura 34 – Organização de um arquivo fonte.

As etapas 4, 5 e 6 constituem a utilização completa de uma das etapas ilustradas no Quadro 7. A etapa 5 atualiza a imagem binária. A partir dessa atualização é feita a extração das características (etapa 6) e, consequentemente, a análise dessas características (etapa 7). Esta sequência foi colocada em *loop* no sentido de que sua ordem deve ser respeitada. Isso porque durante a etapa 5, a imagem binária, que serve de entrada para a próxima etapa, é atualizada de acordo com as ROIs definidas na etapa 3.

Sendo assim, é incorreto chamar todos os métodos de segmentação, para só então chamar os métodos de extração das características. Quanto a análise dessas

características (etapa 7), vale citar que a atualização do estados das ROIs definidas na etapa 3, incluindo movimentação ao longo da tela de interação, também fazem parte da análise das características (exemplos na subseção 5.4.1).

Isso encerra a documentação sobre a arquitetura e o funcionamento da API. A próxima seção ilustra exemplos com propósitos definidos, com o objetivo de apresentar ao leitor algumas aplicações que podem surgir a partir do uso da API.

5.4 EXEMPLOS DE APLICAÇÃO DAS FUNCIONALIDADES DA API MOVE-IN

Esta seção mostra exemplos de aplicação para cada uma das funcionalidades básicas do sistema, ou, mais precisamente, possíveis implementações a partir do uso das classes que compõem o núcleo da API Move-In. O objetivo das subseções a seguir é familiarizar o leitor com os propósitos de cada uma dessas classes. O conjunto de possíveis aplicações vai muito além do que será ilustrado neste trabalho e depende somente da criatividade do desenvolvedor ou *designer* desse tipo de interface.

5.4.1 Movimento e Ativação de Botões, Deslocamento de Objetos Virtuais

Em aplicações que utilizam movimentos do corpo como entrada de dados, botões virtuais podem ser utilizados para disparar uma ação. Esses botões que normalmente seriam ativados com cliques do *mouse*, agora podem ser acionados com movimentos das mãos. A Figura 35 ilustra o diagrama de atividades que descreve esta ação.

A primeira ação é verificar movimento dentro da região de interesse. Se o movimento for detectado, o algoritmo incrementa uma variável (*activity* na Figura 35) que ativa o botão se atingir um determinado limiar. Isto é não se deve ativar o botão na primeira constatação de movimento para evitar que este seja acionado por acidente. Em caso de movimento não detectado, a variável *activity* deve ser decrementada, caso seu valor seja superior à zero. O botão será ativado depois de sucessivos quadros de animação onde for detectado movimento dentro da região de interesse.

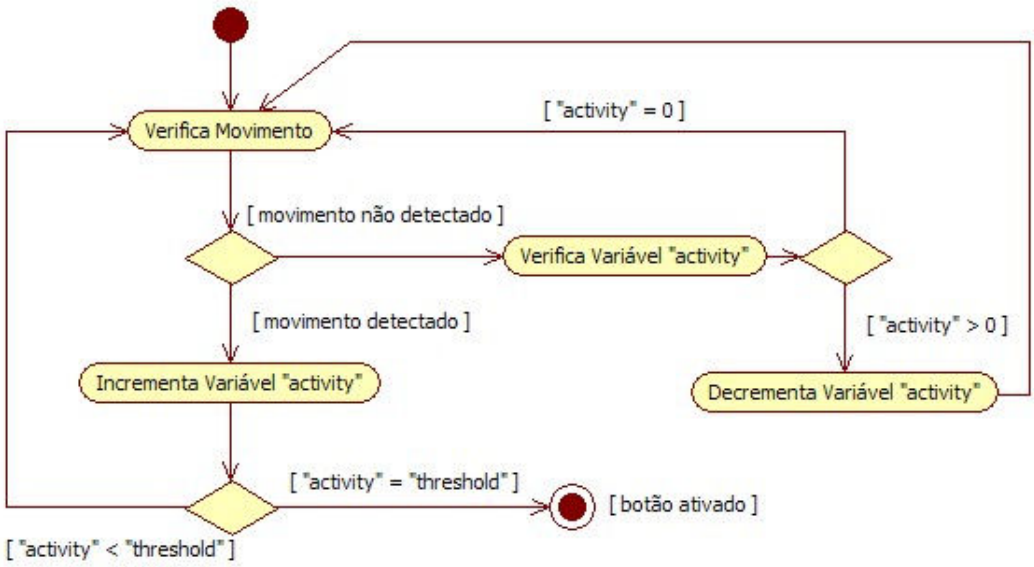


Figura 35 – Diagrama de atividade para botão virtual.

Para movimentar objetos virtuais, pode-se posicionar regiões de interesse do tipo movimento ao redor do objeto alvo. Na Figura 36, um *minigame* do tipo "labirinto" ilustra, a título de exemplo funcional, um círculo que deve ser movimentado ao longo de um trajeto, sem tocar nas paredes deste trajeto.

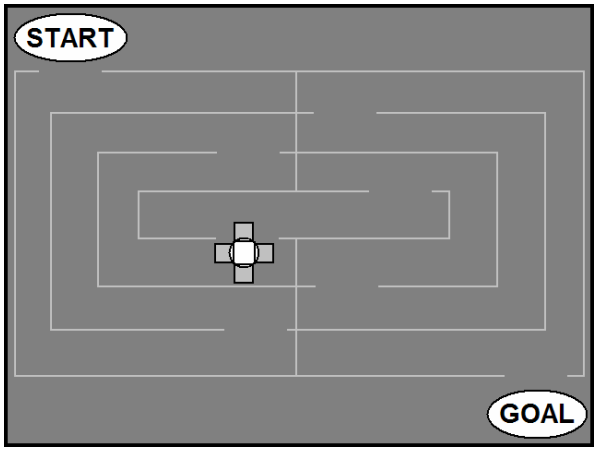


Figura 36 – Minigame "Labirinto"

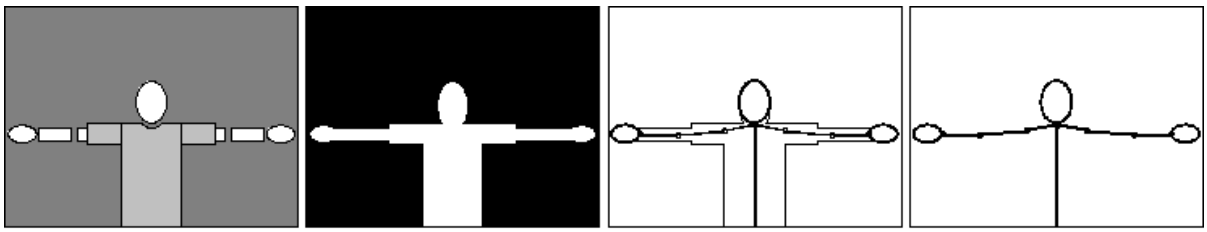
As regiões de interesse que envolvem o círculo determinam os dois graus de liberdade em que este pode se movimentar. Por exemplo, o usuário pode empurrá-lo

para baixo ao fazer movimento na parte superior do círculo, pode empurrá-lo para a direita ao movimentar sua mão na região da extremidade esquerda deste.

5.4.2 Presença e Identificação de Partes do Corpo

A correta identificação de partes específicas do corpo em ambientes não controlados (quaisquer combinações de cenário e iluminação) é um processo extremamente complexo (como foi apresentado no capítulo 3). Esta subseção mostra como a técnica de presença pode ser usada para identificar certas partes do corpo, desde que algumas restrições sejam respeitadas.

Nesta subseção podem ser conferidas as partes do corpo que este trabalho pretende identificar, o conjunto de algoritmos responsável pela identificação, as situações mais propensas ao sucesso de identificação e as situações onde o conjunto de algoritmos tende a falhar. Primeiramente, a Figura 37 exemplifica, em resumo, o processo de identificação.



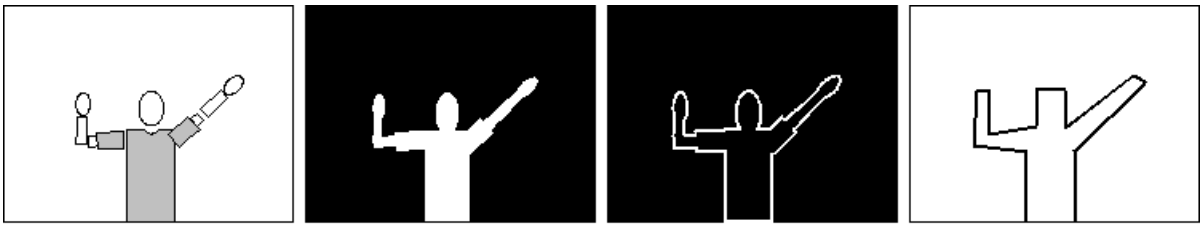
(A) **(B)** **(C)** **(D)**

Figura 37 – Processo de identificação das partes do corpo.

A Figura 37.A mostra a representação da imagem de entrada, que consiste de um usuário devidamente posicionado para que fique com a parte superior do corpo do raio de visualização da câmera. O segundo quadro (Figura 37.B) mostra a imagem binarizada extraída com a técnica do tipo presença com diferença entre quadros. Na Figura 37.C aparecem as partes do corpo que deverão ser identificadas a partir das características extraídas da imagem binarizada, e o resultado final (Figura 37.D). Isto é, as partes que devem ser identificadas são: cabeça, mãos, cotovelos, ombros e o pescoço. Este último é considerado como sendo o ponto médio entre os ombros.

A lista de algoritmos que deverá ser aplicada na identificação das partes do corpo aparece a seguir. Este processo pode ser chamado de identificação de pontos extremos e envolve, basicamente, a técnica de identificação de presença e a análise de contornos da silhueta do usuário, onde serão extraídos o fecho convexo e os defeitos de convexidade (vise subseções 2.5.7 e 2.5.8).

1. Obter a imagem de entrada (Figura 38.A);
2. Extrair a silhueta binarizada do usuário através da técnica de presença (Figura 38.B);
3. Extrair os contornos da imagem binarizada (Figura 38.C);
4. Simplificar o resultado do contorno para obter os pontos extremos (regiões de concavidade e convexidade - Figura 38.D).



(A) **(B)** **(C)** **(D)**

Figura 38 – Identificação de pontos extremos.

Pode-se explicar a sequência de passos do algoritmo de identificação dos pontos extremos a partir dos conceitos apresentados no capítulo 2. Neste caso, o processo inicia com a captura das imagens de entrada. Caso a resolução desta imagem seja superior a *320x240*, pode-se pensar em reduzí-la para este tamanho (pré-processamento). Após isso, a imagem é binarizada (redução de dados) e é realizada uma subtração entre imagens binárias. A partir da imagem binarizada, o contorno da silhueta é identificado (extração de características).

Os pontos que deverão identificar as partes do corpo aparecem em esboço na na Figura 39. Os pontos de concavidade são aqueles cujas setas apontam para dentro da silhueta e os pontos de convexidade são aqueles cujas setas apontam para fora da silhueta. A proximidade entre esses pontos deve indicar que correspondem à mesma região que deve ser identificada.

Por exemplo, pode-se oberservar a partir da Figura 39 que as mãos possuem dois pontos de convexidade próximos, o cotovelo direito possui um ponto de concavidade e um de convexidade. A ordem de disposição desses pontos ao longo da silhueta, e também a distância entre esses pontos, podem servir para indicar a qual região do corpo eles estão relacionados.

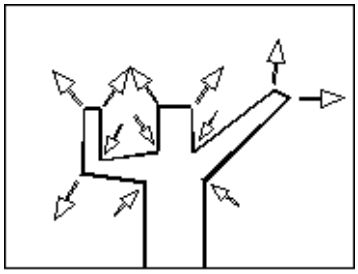


Figura 39 – Pontos de concavidade de convexidade.

Observa-se que para que este processo funcione de forma correta, as mãos devem estar afastadas umas das outras e da cabeça. Também é necessário que a cabeça esteja sempre próxima a uma região (aproximadamente o centro) da tela. A Figura 40 mostra o que seria a situação ideal para aplicação do algoritmo de identificação de pontos extremos. A região 1 deve ser ocupada pela cabeça, a região 2 deve ser ocupada pela mão direita e a região 3 pela mão esquerda.

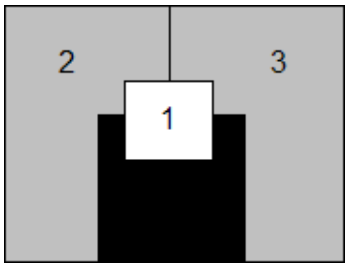
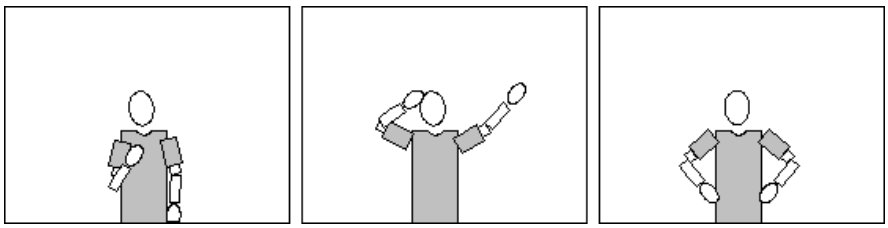


Figura 40 – Áreas de posicionamento que facilitam a identificação.

A Figura 41 exemplifica situações de oclusão, onde partes do corpo que deveriam ser identificadas estão sobrepostas a outras partes, e que, portanto, são difíceis de tratar. Da esquerda para a direita se tem: a mão em oclusão ou muito próxima do corpo, uma das mão em oclusão com a cabeça e uma pose complicada de tratar através da simples extração de contornos.



(A) **(B)** **(C)**

Figura 41 – Situações que devem ser evitadas.

Uma questão a ser considerada é que, apesar do fato de que estas posições não são possíveis de serem identificadas com o algoritmo proprosto, o contorno simplificado da silhueta constitui um número pequeno de pontos (coordenadas *x* e *y*), que podem ser armazenados para análise posterior. A princípio, por classificação através de observação humana, e, possivelmente em futuros trabalhos, através da utilização de técnicas que fazem uso de aprendizado de máquina.

5.4.3 Cor da Pele e Desenho com a Ponta do Dedo

A grande incoveniência da técnica de presença é captura do quadro inicial e a grande sensibilidade a variações de iluminação. A técnica de segmentação através da cor da pele é outra forma de identificação de presença do usuário (capacidade de perceber o usuário mesmo quando este está parado), e deve ser utilizada em aplicações que visam interação com as mãos, braços e cabeça, isto é, quaisquer partes do corpo onde é alta a probalididade dos membros alvo não estarem cobertos por roupas.

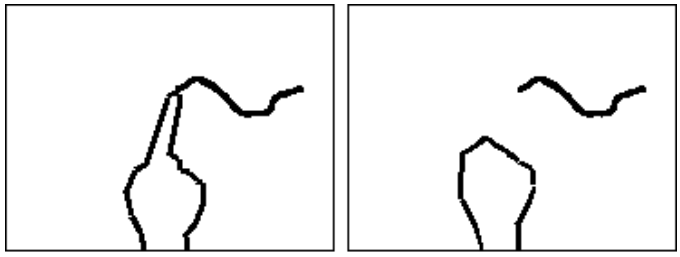


Figura 42 – Desenho com a ponta do dedo.

A Figura 42 mostra um exemplo de aplicação que pode ser realizada com técnicas de presença. De fato, para este exemplo, deve-se considerar o mesmo algoritmo de identificação de pontos extremos proposto na seção anterior. Neste caso, aplicação irá plotar círculos na tela sempre que identificar a ponta do dedo e deverar encerrar o desenho sempre que perder a identificação do dedo. Este é um um dos casos onde a técnica de identificação de cor da pele se aplica melhor, pelo fato desta ser superior à técnica de presença mostrada na seção anterior, em termos de iluminação e sensibilidade à variações de iluminação (vide Quadro 5).

5.4.4 Fluxo Óptico e Interação Física com Objetos Virtuais

A técnica de fluxo óptico foi utilizada em (ZIVKOVIC, 2004) para movimentar objetos virtuais, tal como exemplicado na subseção 5.3.1. Além disso, esta técnica também pode ser empregada na interação física com objetos virtuais, tal como em jogo de vôlei virtual, como ilustrado na Figura 43.

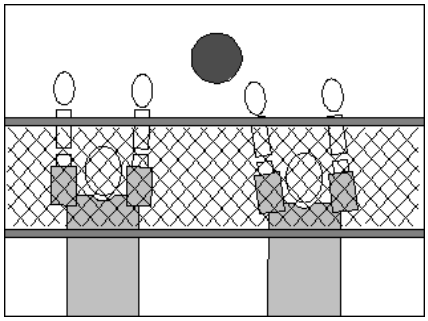


Figura 43 – Vôlei virtual.

Esta técnica é capaz de determinar, por exemplo, para onde e com qual velocidade os objetos devem ser rebatidos, a partir do cálculo efetuado interior a uma região de interesse. Na Figura 43 a idéia é movimentar uma bola virtual, compreendida pela região de interesse que calcula o fluxo óptico. O jogador deve rebater a bola com o braço no momento certo ou, com velocidade suficientemente alta para “furar o bloqueio” e marcar ponto.

Esta seção apresentou possíveis utilidades para as classes do núcleo da API Move-In, com especial atenção para a identificação de partes do corpo, que é uma

tarefa mais complicada do que somente determinar quando há presença ou quando há movimento. As hipóteses que foram aqui levantadas são provadas, ou refutadas com as devidas justificativas, no capítulo 6.

5.5 CONSIDERAÇÕES FINAIS

Na seção 5.2 foi comentado sobre as vantagens do padrão *Facade* e sobre como a API Move-In está baseada neste padrão. Em seguida, na seção 5.3, foi apresentado como a API em questão está organizada e as etapas que devem compor o arquivo fonte. Nesta última se percebe que as estruturas de dados e as regiões de interesse são de fato gerenciadas pelo programador e não pela classe *Screen*. O que a classe *Screen* gerencia de forma transparente é a conexão e comunicação com a câmera, bem como a atualização da imagens de entrada e a passagem delas como parâmetro para os métodos que correspondem a uma das quatro funcionalidades principais da API.

A implementação final obteve este formato por motivo de que seria possivelmente mais complicado e menos eficiente, do ponto de vista do autor deste trabalho, garantir que a classe *Screen* tivesse controle sobre todas as estruturas. Considerando o exemplo da classe *ROIGroup*, uma instância dessa classe deve ser criada para cada objeto virtual a qual ela representa.

Gerenciar as ROIs pela classe *Screen* implicaria no fato de que tal classe deveria conter um gerenciador de *ROIGroup*, pois não há como garantir quantos objetos virtuais estarão sendo utilizados pelo programador ou *designer* da aplicação. Esta estrutura seria uma lista da classe *ROIGroup*, ou uma multilista de *ROIs*. Esta estrutura necessitaria de ponteiros que poderiam até mesmo jamais ser instanciados em uma aplicação onde uso de *ROIs* não é necessário.

Os mesmos argumentos são válidos para as estruturas de dados que guardam as características das imagens. É por esse motivo que se pode concluir que a classe Screen compreende o padrão *facade* no que diz respeito à câmera e às imagens de entrada. As ROIs e as estruturas de dados *FlowVectors*, *PointSequence* e *ExtremePoints* devem ser criadas e gerenciadas pelo programador.

6. RESULTADOS

O capítulo 5 levantou hipóteses sobre os possíveis tipos de aplicações que podem ser desenvolvidas com a API Move-In. Este capítulo mostra o resultado da implementação para a maior parte das soluções propostas. A leitura cuidadosa deste capítulo é extremamente importante para os interessados na utilização da API Move-In, já que este contém exemplos das principais funcionalidades desenvolvidas.

Além disso, as próximas seções são utilizadas para detalhar as formas de segmentação do objeto de interesse, como as características da imagem podem ser trabalhadas para prover interação com o ambiente virtual, e também quanto a utilização e gerenciamento das ROIs (regiões de interesse) para interação com objetos virtuais.

6.1 FORMAS DE SEGMENTAÇÃO

Esta subseção exemplifica em detalhes as possíveis formas de segmentação do objeto de interesse. Basicamente, existem duas formas, que são a diferença entre quadros de animação e a identificação de cor da pele, respectivamente obtidas através das classes *FrameDifference* e *SkinColor*. A primeira delas pode utilizar a diferença entre os dois últimos quadros para caracterizar movimento, ou a diferença entre o último quadro capturado e um quadro de referência – cenário de fundo – para indicar presença, conforme Figura 44.

A Figura 44 mostra quadros capturados com a câmera ilustrada na Figura 29.B. Nela aparecem o quadro de referência (Figura 44.A), o quadro *N-1* (Figura 44.B), o quadro *N* (Figura 44.C), a segmentação do tipo presença para o espaço de cor RGB (Figura 44.D), o mesmo tipo de segmentação para o espaço de cor RGB normalizado (Figura 44.E), e a segmentação do tipo movimento (Figura 44.F). Para essas formas de segmentação, o parâmetro *threshold*, ou limiar de segmentação, pode ser crucial para a correta separação do objeto de interesse do cenário de fundo.

A diferença entre quadros é efetuada sobre a intensidade dos *pixels*, onde o limiar, que pode variar de *0* a *255*, determina que os *pixels* brancos na imagem binarizada resultante serão aqueles cuja diferença de intensidade está acima do

valor do limiar. Um limiar alto tende a produzir falsos negativos, conforme Figura 44.D, onde parte da roupa do usuário não foi identificada. Já um limiar baixo irá produzir falsos positvos. Isso aparece na Figura 44.D, onde parte da parede foi identificada como objeto de interesse.

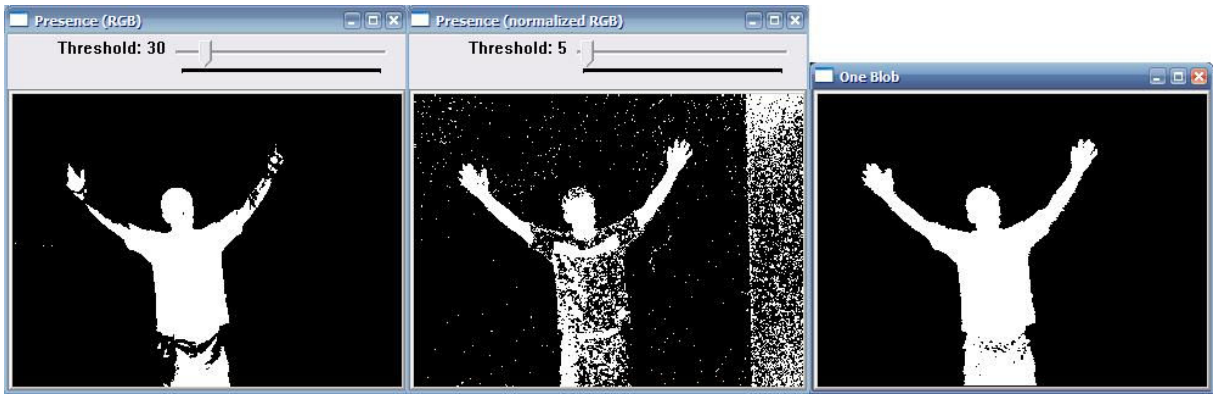
(A) **(B)** **(C)**

(C) **(D)** **(E)**

Figura 44 – Segmentação com presença e movimento.

É possível observar a partir da Figura 44 que a identificação do objeto de interesse no espaço de cor RGB normalizado requer limiares mais baixos, com valores que variam entre *5* e *20*, e para este mesmo espaço de cores não normalizado, os valores ficam entre *20* e *50*. Isso ocorre porque a normalização das cores diminui as variações bruscas de intensidade entre os *pixels*. Outro fato que não está ilustrado na Figura 44 é o de que um limiar baixo para o espaço de cor RGB tende a ser extremamente sensível a mudanças de iluminação, resultando em uma quantidade de falsos positivos que pode chegar a mais de *90%* do cenário de fundo.

Na Figura 44 também se nota que a segmentação no espaço de cor RGB normalizado (Figura 44.C) e não normalizado (Figura 44.D) produz diferentes resultados, não sendo raras as situações onde uma melhor segmentação é obtida a partir da combinação desses dois tipos. Também é possível eliminar falsos positivos ao se considerar somente o objeto de interesse com maior número de pixels brancos (maior *blob – binary large object*). A Figura 45.C mostra o resutado dessa combinação, já com a separação do maior *blob*. Esta funcionalidade, muito importante para muitos métodos das classes da API Move-In, é efetuada com métodos da API cvBlobsLib (CVBLOBSLIB, 2007)[1].



(A) **(B)** **(C)**

Figura 45 – Combinação de segmentações e extração do maior *blob*.

A segmentação do tipo presença pode ser utilizada para extração dos contornos da silhueta ou substituição do cenário de fundo real por cenário de fundo virtual. É por esse motivo que bons resultados de segmentação são exigidos e é necessário pensar em todo um processo para chegar a este fim, conforme ilustrado na Figura 45. Ao contrário, a diferença entre quadros para extrair movimento, que geralmente é utilizada para interação com objetos virtuais por não ser capaz de identificar a presença de um objeto de interesse estático, não requer a mesma quantidade de processamento. Para segmentação deste tipo, bons resultados (imagens binárias que indicam corretamente o movimento dentro da região de interesse) são obtidos para imagens no espaço de cor RGB não normalizado, com limiares variando entre *40* e *60*.

[1] Vide seção 4.3.

Por último, há ainda a segmentação através de cor da pele. Este tipo de segmentação na API Move-In é realizado através de equações paramétricas desenvolvidas de forma empírica (subseção 2.5.3), e que classificam os *pixels* da imagem como fazendo parte do conjunto de cores cuja tonalidade constitiu pele humana ou como não fazendo parte deste conjunto. A Figura 46.B e a Figura 46.C mostram o resultado das equações para os espaços de cor RGB e RGB normalizado. Essas figuras provam que este tipo de segmentação não é adequada para ambiente não controlado. Neste caso, nem mesmo a combinação dos resultados e a separação dos maiores *blobs* resolveria o problema de segmentar os braços e a cabeça do usuário.

(A) **(B)** **(C)**

Figura 46 – Segmentação da cor pele e falsos positivos.

No entanto, em situações mais específicas como, por exemplo, apontar a câmera para uma parte lisa da parede e trabalhar com a identificação de uma das mãos, é mais apropriada que a diferença entre quadros com inicialização. As vantagens estão na independência de um limiar e de uma etapa de inicialização para capturar o quadro de referência.

6.2 CONTORNOS E PONTOS EXTREMOS

A extração de contornos é um recurso muito importante para interação com o tipo de ambiente virtual descrito neste trabalho. É a partir dos contornos que se pode realçar a imagem do objeto de interesse, e também extrair outras características como pontos côncavos e convexos (neste trabalho referenciados como pontos

extremos) de uma silhueta. Nesta seção está exemplificado como os contornos podem ser usados em conjunto com cenário virtual, e como os pontos extremos podem ser utilizados na identificação de certas poses. Estas características são obtidas através imagens binarizadas que são passadas à chamadas da classe *FrameFeatures*.

A Figura 47 parte do resultado da imagem apresentada na Figura 45. A partir deste resultado, o contorno do objeto de interesse (usuário) é identificado (Figura 47.A), para que então possa ser suavizado (Figura 47.B) e utilizado no realce da imagem do objeto de interesse frente ao cenário virtual (Figura 47.C). A suavização do contorno não é um fator tecnicamente importante neste caso, mas pode proporcionar um resultado visual mais agradável.

(A) **(B)** **(C)**

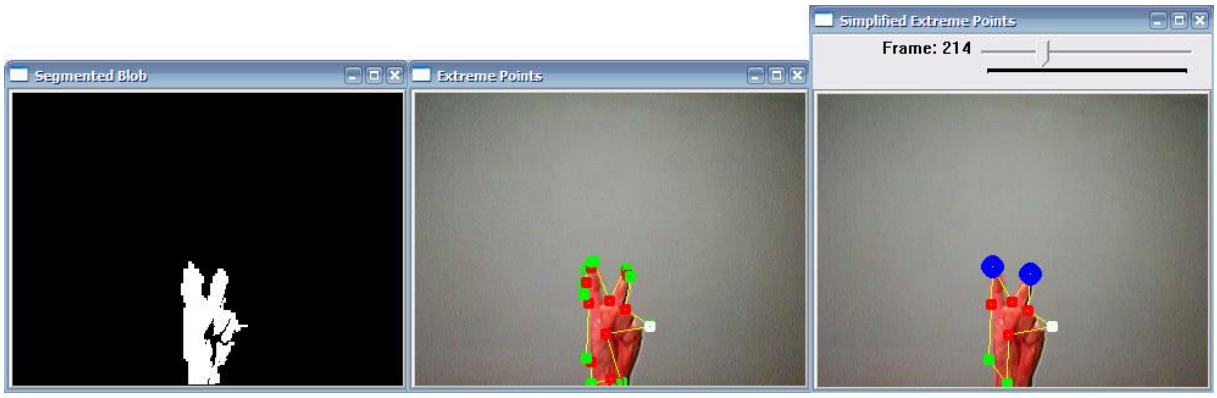
Figura 47 – Contorno, contorno suavizado e cenário virtual.

A identificação de poses ou posturas pode ser feita através do estabelecimento de heurísticas sobre as características extraídas da imagem. Isso constitiu a etapa de análise de características e com este exemplo deve ficar claro porque esta etapa depende dos propósitos da aplicação. Ou seja, a análise das características é derivada a partir do objetivo da aplicação. Por exemplo, é objetivo da aplicação identificar quantos dedos de sua mão o usuário está erguendo, considerando a resposta do algortimo será um número entre *0* e *5*, e que a entrada de dados constitui uma das mãos, fechada ou com um ou mais dedos erguidos.

Considerando a característica de pontos extremos, que apresenta pontos de convexidade e concavidade identificados através do contorno, pode-se pensar nas formas em que estes pontos estarão dispostos para compor uma determinada pose,

e então criar heurísticas a partir destas formas para então classificar esses pontos. A Figura 48.C desenha um círculo maior nos pontos extremos que atendem os seguintes requisitos:

1. O ponto *point(i)*, sendo *i* sua posição na lista de pontos, deve ser convexo;
2. Os pontos *point(i – 1)* e *point(i + 1)* devem ser côncavos;
3. O ponto *point(i)* deve se encontar em uma posição (altura) superior à dos pontos *point(i – 1)* e *point(i + 1)*;



(A) **(B)** **(C)**

Figura 48 – Identificação da quantidade de dedos erguidos.

Esses requisitos, ou heurísticas, são capazes de determinar com sucesso quantos dedos estão erguidos para a situação proposta. Na Figura 48.A se observa que isto é verdadeiro mesmo para ocasiões onde a silhueta da imagem binarizada, obtida através de indentificação da cor da pele, apresenta uma qualidade inferior à da apresentada na Figura 47. A Figura 48.B mostra o resultado da extração dos pontos extremos, de concavidade e de convexidade, e a Figura 48.C mostra a sequência simplificada desse conjunto de pontos, onde aqueles que estão muito próximos foram eliminados. Também aparece na Figura 48.C a identificação da ponta dos dois dedos erguidos, através da plotagem de um círculo maior ao redor da extremidade desses dedos.

A partir daí se pode implementar sem grandes dificuldades a aplicação que foi proposta na subseção 5.4.3. A Figura 49 mostra uma versão um pouco mais elaborada que a proposta, no sentido de que há um objetivo de onde se deve ou não desenhar. Neste exemplo há uma face na tela que deve ser colorida com a ponta do dedo. Nesta aplicação, o círculo maior indica sempre o ponto mais alto do maior *blob*

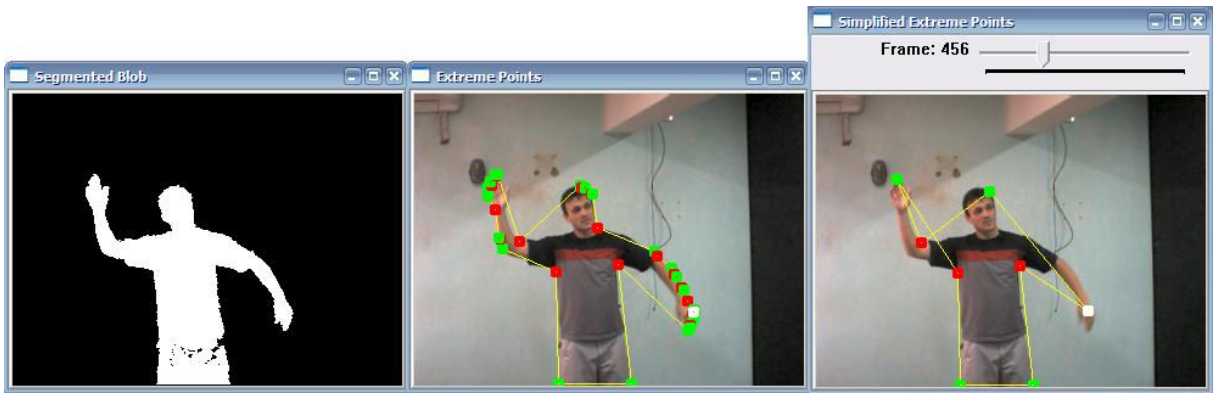
identificado como cor de pele, para que o usuário possa se localizar mesmo quando seu dedo não está pintando.

Figura 49 – Pintura com o dedo.

A pintura com o dedo será ignorada na parte inferior, onde as instruções são mostradas ao usuário. Dentro da face a pintura é em tonalidade de verde, para indicar acerto, e fora da área de desenho a pintura é feita em vermelho, para indicar erro. Pode-se pensar em transfomar esta demonstração em um *minigame*, onde haveria um tempo para completar a terafa e um *feefback* com a relação em acerto e erro.

O algoritmo de identificação de pontos extremos apresentado na Figura 48 é baseado nas funções providas pela API OpenCV (fecho convexo e defeitos de convexidade). Como foi demonstrado, tal algoritmo funciona muito bem para a simplificação dos contornos da mão humana, mas o mesmo não ocorre na simplificação da silhueta do corpo humano. Conforme demonstrado pela Figura 50, os pontos de concavidade (citados pelo OpenCV como *convexity defects*) aparecem apenas uma vez entre os pontos de convexidade. Isto é, sempre existirá no máximo um ponto côncavo entre dois pontos convexos.

Na Figura 50 também se pode observar que o algoritmo de simplicação de pontos extremos produz resultados indesejados para contornos muito complexos. Este algoritmo percorre a lista de pontos eliminando aqueles que são do mesmo tipo e cuja distância entre eles é menor do que um valor definido pelo programador. Isso acabou por eliminar pontos convexos que indicariam a posição aproximada dos cotovelos.



(A) **(B)** **(C)**

Figura 50 – Identificação dos pontos extremos para a parte superior do corpo.

A identificação de uma quantidade grande de possíveis posturas, considerando a dinâmica de movimento do corpo humano, mesmo com as restrições da Figura 40, é mais complexa de tratar quando comparada à possíveis poses da mão identificadas na seção 6.3. Isso ocorre pela maior quantidade de graus de liberdade que o corpo humano pussui quando comparado à mão com a palma voltada para frente da câmera.

Até o momento não foi identificado um método que validasse a proposta apresentada na subseção 5.3.2. Esta proposta ainda parece razoável e não foi descartada, mas ainda há de se criar um algoritmo que simplifique de forma correta a silhueta do corpo humano, para só então poder classificar os pontos, através de heurísticas que deverão estar baseadas, possivelmente, na localização dos pontos, na sequência de pontos de concavidade e de convexidade e na distância entre esses pontos.

A proposta de Wu et al (2000) que percorre o contorno com dois vetores realizando a leitura do produto escalar se mostrou inviável para esta aplicação. Na identificação do ponto de maior produto escalar esta alternativa é mais adequada por tratar da localização de um único ponto. Porém, durante a identificação dos pontos extremos apresentados na Figura 39, idependentemente do intervalo especificado para valores do produto escalar que classificam um ponto do contorno como ponto extremo, o resultado acaba produzindo muitos falsos positivos e/ou falsos negativos entre a transição dos quadros de animação.

6.3 USO E GERENCIAMENTO DE REGIÕES DE INTERESSE

Tendo apresentado as formas de segmentação disponíveis na API Move-In e a utilidade da identificação de características da imagem, esta seção apresenta como as ROIs podem ser utilizadas para interagir com objetos virtuais. Os resultados de segmentação para as figuras 41 à 47 foram processados sem determinação de uma região de interesse específica. Quando assim é feito, tando para OpenCV quanto para a Move-In, a região a ser processada consiste da totalidade dos *pixels* nas imagens de entrada.

Porém, aplicações mais completas, que vão além dos exemplos funcionais, exigem que a imagem de entrada seja particionada em vários sub-conjuntos de *pixels*. Isso é válido para o uso de botões virtuais, para proporcionar ao usuário um *menu* que também pode ser controlado com movimentos do corpo e para movimentar objetos virtuais. O botão virtual proposto na subseção 5.4.1 pode ser implementado sem dificuldades com o uso de ROIs na API Move-In.

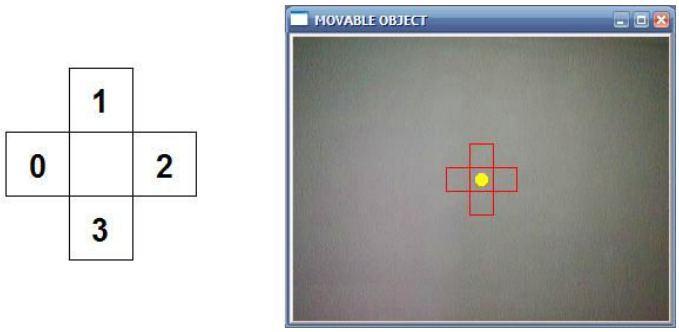
A dinâmica proposta na subseção 5.4.1 fica melhor demonstrada através de vídeo. A Figura 51 apresenta uma versão equivalente que ilustra o mesmo conceito. Na Figura 51.A pode se notar a imagem binária obtida através de segmentação do tipo movimento. Nota-se que a área que contém *pixels* brancos nesta imagem corresponde apenas à região de interesse indicada pelo quadrado na região superior direita da Figura 51.B. Esta figura também indica a porcentagem *pixels* ativos (brancos) dentro desta ROI. Para finalizar o botão virtual bastaria, por exemplo, disparar uma ação após uma sequência de quadros de animação com atividade superior a *25%* dentro da ROI que corresponde ao botão.

De forma semelhante, o mesmo conceito pode ser aplicado para mover objetos virtuais, ao inserir ROIs ao redor do objeto de interesse. Para tanto, o programador deve inicialmente declarar uma instância da classe *ROIGroup* e inserir nesta lista as regiões de interesse necessárias. Testes funcionais indicaram que *4* ROIs são suficientes para mover objetos ao longo da tela. Após isso, o programador deve definir qual ação será disparada quando houver atividade nas subregiões de interesse.

(A) **(B)**

Figura 51 – Simulação de botão virtual.

Por exemplo, na Figura 52.A, as subregiões estão numeradas conforme foram inseridas na *ROIGroup*. Deve-se determinar que atividades na subregião *0* moverão o objeto virtual para a direita, a subregião *1* moverá o objeto para baixo, e assim por diante. Como a leitura de toda a *ROIGroup* (segmentação) é efetuada antes do movimento (análise das características), o movimento resultante é realizado conforme a leitura de atividade em todas as subregiões. Por exemplo, atividades lidas nas subregiões *0* e *3* moverão o objeto na diagonal superior direita. Na Figura 52.B, o ponto localizado no centro da *ROIGroup* indica o ponto central do conjunto de subregiões a ela pertencentes. Na apliação final, a figura que representa o objeto virtual deverá ter seu centro associado e deslocado conforme o centro da *ROIGroup*. Ao se transladar uma *ROIGroup*, todas as subregiões nela inseridas irão acompanhar o deslocamento.



(A) **(B)**

Figura 52 – ROIGroup para movimentação de objeto virtual.

Em resumo, para trabalhar com regiões de interesse, o programador deve instanciar um objeto da classe *ROIGroup*, deve definir quantas subregiões precisa e qual o tamanho dessas regiões, e deve programar uma função que será chamada quando houver atividade dentro da subregião. A atualização de uma *ROIGroup* é feita de forma transparente junto com os métodos de segmentação (de presença ou movimento) da classe *Screen*, e a atividade em cada região, por se tratar de uma característica da imagem, deve ser lida com chamadas a métodos da classe *FrameFeatures*.

6.4 APLICAÇÕES DE FLUXO ÓPTICO

Na API Move-In, o fluxo óptico pode ser utilizado para interação com objetos virtuais, e também para rastreamento de pontos. Apesar disso, sua utilização é mais recomendável para o primeiro caso, onde a probabilidade de sucesso é maior, tal como será exemplificado ao longo desta seção. Em *webcams* comuns, onde a quantidade de quadros de animação varia entre *15* e *30* FPS, movimentos ou ações de alta velocidade não são capturados pela técnica em questão.

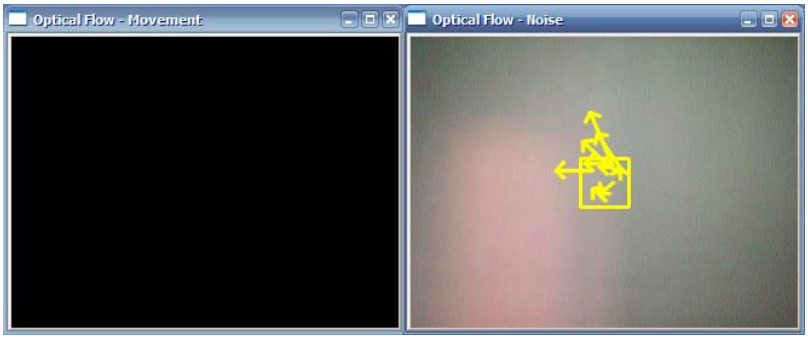
A exemplo disso, pode-se citar a aplicação de vôlei virtual sugerida na seção 5.3.4. A idéia geral, de “cortar” a bola com movimento do braço, seria perfeitamente válida se não fosse a limitação de FPS provida pelas *webcams* utilizadas. Isso significa que as aplicações que pretendem utilizar esta técnica devem ser idealizadas de forma a limitar a velocidade de movimento do usuário.

Não é objetivo deste trabalho entrar em detalhes sobre dinâmicas de jogos, mas exemplos seriam uma bomba de ar, cuja haste deve ser empurrada “vagarozamente”, por causa da pressão dentro da bomba, ou uma manivela, que pode ser “pesada” para girar. Ambos os exemplos são passíveis de implementação, mesmo para *webcams* comuns. No primeiro, haveria uma ROI fixa sobre a haste da bomba de ar, para ler movimentos na vertical. O mesmo é válido para o caso da manivela, que teria uma ROI para a classe *Flow*, girando conforme leitura do movimento aplicado sobre a ROI.

Durante a interação com objetos virtuais, é mais comum que a leitura de movimento seja feita dentro de uma ROI (Figura 52). Então, da mesma forma que na diferença entre quadros que lê movimento, o programador deverá primeiramente

especificar uma ROIGroup para que a leitura do fluxo óptico seja efetuada dentro de suas sub-regiões. Feito isso, o número máximo de pontos a serem rastreados deve ser determinado para que seja passado como parâmetro do método *Screen::Flow*. O resultado da chamada deste médodo será uma estrutura de dados com os valores de posição dos pontos rastreados entre as últimas duas imagens. O que determina os pontos a serem rastreados é o algoritmo de (SHI e TOMASI, 1994), implementado na biblioteca OpenCV através da função *cvGoodFeaturesToTrack*.

Uma detalhe observado durante os testes com fluxo óptico e a *webcam* da Figura 29.B, foi o fato de que esta técnica é extremamente sensível a ruídos e a consequente produção de falsos positivos em situações onde não há movimento na tela de interação. Na figura Figura 53 aparecem, respectivamente, a imagem binária resultado da segmentação do tipo movimento demonstrada na seção 6.4, e a imagem resultante da execução da leitura dos vetores de movimento interiores à ROI. Ós módulos dos vetores foram aumentados em um fator de *10X* para facilitar visualização.



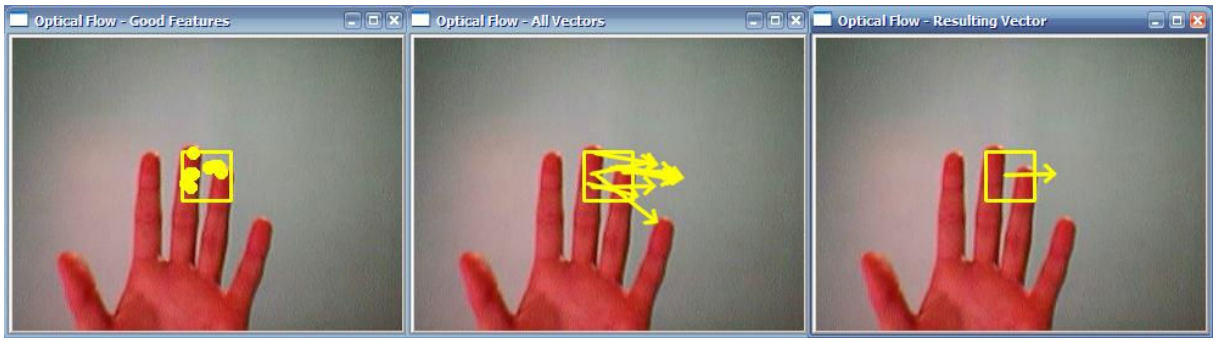
(A) **(B)**

Figura 53 – Ruído produzido pelo fluxo óptico.

Na Figura 53.A, a imagem completamente nula representa a ausência de movimentação durante a transição entre quadros[1]. Ainda assim, os vetores (*10* ao todo) da Figura 53.B que representam o fluxo óptico estão indicando leitura de movimentação entre os *pixels*. Isto é, há uma contradição entre as técnicas de leitura de movimento. Isto ocorre porque a última técnica citada é extremamente sensível à quaisquer variações de iluminação, enquanto que na primeira, esta sensibilidade pode ser controlada pelo parâmetro *threshold*.

[1] Ao contrário da Figura 53.A, na Figura 51.A se pode observar *pixels* brancos dentro da ROI, representando movimento.

Este problema pode ser resolvido através da combinação das duas técnicas, onde o fluxo óptico somente deverá ser calculado quando a atividade de movimento com a diferença entre quadros for superior a um deterninado valor[1]. Este valor irá depender do tamanho da ROI e do propósito da aplicação. Quanto menor o tamanho, menor deverá ser a taxa de atividade necessária para ativação. Na Figura 54, foi utilizado um valor de ativação igual a *2,5*% (dois e meio por cento). Com relação aos propósitos da aplicação, o programador deverá definir se quer uma ROI mais sensível, onde qualquer toque na região a ativa, ou menos sensível, quando é necessário que a maior parte da ROI esteja ativa para então calcular o fluxo óptico.



(A) **(B)** **(C)**

Figura 54 – Pontos rastreados, vetores de movimentação e vetor resultante.

A Figura 54 mostra a região de interesse com os pontos de rastreamento identificados (Figura 54.A), os vetores de movimento correspondentes a esses pontos (Figura 54.B) e o vetor resultante (Figura 54.C). Este último tem sua direção calculada pela soma dos vetores, e seu módulo calculado pela média aritimética entre as magnitudes do conjunto de vetores. Este método de calcular o vetor resultante foi assim definido para que o resultado fosse independente da quantidade de pontos rastreados. Isto é, com a realização da soma de vetores, quanto maior é a quantidade, maior será o módulo. É a partir da análise do vetor resultante que o programador pode determinar a movimentação e a velocidade de deslocamento dentro da ROI.

O fluxo óptico também foi testado para rastreamento de pontos. Neste caso, recomenda-se que o programador defina de forma explícita quais os pontos deverão

[1] Este valor é obtido através do método *FrameFeatures::activaveArea()*.

ser rastreados. Ou seja, a segmentação do objeto de interesse para o fluxo óptico pode ser entendida como a definição dos pontos a serem rastreados, que nos exemplos anteriores, foi feita de forma automática. Outra opção seria definir o conjunto de pontos inicial de forma explícita e não automática, com eventos de *mouse* para adicionar pontos à lista, ou criar um método customizado de "boas características para rastrear" – *cvGoodFeaturesToTrack* – para então passá-los como parâmetro ao método *Flow* da classe *Screen*.

Quanto a esta alternativa, percebeu-se que o correto rastreamento dos pontos especificados somente ocorrerá de forma satisfatória para ambientes controlados, onde o cenário de fundo apresenta pouca variação de textura e relevo, tal como a parede lisa que aparece na Figura 54. Também não é possível rastrear pontos do objeto de interesse que se encontram em rotação durante a mudança entre os quadros, se a rotação resultar na oclusão desses pontos.

6.5 COMPARATIVO ENTRE TÉCNICAS APRESENTADAS

Com o intuito de enfatizar o que foi demonstrado neste capítulo, esta seção apresenta dois quadros comparativos entre as técnicas que foram explicadas.

Quadro 8 – Comparativo entre as técnicas de segmentação

Técnica	**Necessita de:**	**Inviável quando:**	**Recomendado para:**
Presença com diferença entre quadros.	Inicialização com captura do quadro de referência.	Há constante mudança de iluminação no cenário.	Substituição de cenário real por cenário virtual, identificação de poses e posturas.
Presença com cor da pele.	Na identificação de braços e/ou cabeças apenas, a camiseta do usuário não pode ser de cor semelhante à da pele.	Há muitos objetos no cenário que são erroneamente identificados como pele humana.	Quando não é necessário ter qualidade na imagem binarizada que indica presença.
Movimento com diferença entre quadros.	Objeto de interesse em movimento.	Há outros objetos se movendo no cenário que não sejam o próprio objeto de interesse.	Interação com objetos virtuais. Principalmente movimentação de objetos e ativação de botões virtuais.
Movimento com fluxo óptico.	Idem à anterior.	Idem à anterior. Também é limitada pelo *FPS* da câmera. Não é possível trabalhar com "movientos rápidos" com *webcams* comuns.	Quantificação de direção e velocidade de movimento, rastreamento de pontos definidos pelo programador (somente em ambiente controlado).

O primeiro deles (Quadro 8), compara as possíveis formas de segmentação do objeto de interesse, e o segundo (Quadro 9) comenta sobre as características que podem ser extraídas da imagem. O objetivo desta seção é enfatizar que não há uma melhor solução para tratar da interação com *webcams*, mas que há situações ou eventos onde certas técnicas são mais adequadas que outras.

Por exemplo, é mais adequado que se use uma técnica de segmentação do tipo presença para realizar a substituição de cenário real por cenário virtual. As segmentações do tipo presença também são mais indicadas quando é necessário trabalhar com identificação de configurações espefíficas de posição, tal como é o caso de desenhar com o dedo (seção 6.2), ou disparar um projétil ao erguer o braço (subseção 3.1.1). Já as segmentações do tipo movimento são mais adequadas para interagir com objetos, ao passo que são mais tolerantes à mudanças de iluminação.

Quadro 9 – Comparativo entre as técnicas de extração de características.

Característica	**Necessita de:**	**Inviável quando:**	**Recomendado para:**
Contornos.	Uma técnica de segmentação do tipo presença, quando o contorno de todo o objeto de interesse for necessário.	Para segementação do tipo movimento, os contornos serão correspondentes ao movimento (diferença entre os últimos quadros).	Enfatizar o limite entre ousuário e o cenário virtual. Indicar a localização do movimento efetuado, sem alterar consideravelmente a imagem de saída.
Fecho convexo.	Segmentação do tipo presença.	Idem à anterior.	Identificação de formas, poses ou posturas.
Pontos Extremos.	Idem à anterior.	Idem à anterior.	Idem à anterior.
Percentual de atividade.	Qualquer técnica de segmentação.	Movimentos extremamente rápidos, como o gesto de rebater uma bola de tênnis de mesa com uma “cortada”, não são detectados com *webcams* comuns.	Quantificar presença e/ou movimento dentro de uma ROI, com o objetivo de disparar ações ao atingir um determinado limiar.
Vetores de movimentação.	Fluxo óptico.	Esta técnica é inviável para movimentos rápidos, e também para utilização de forma independente[1], já que tende a produzir muitos ruídos.	Definir a direção e a velocidade de movimentação de objetos virtuais, roptacionar objeto virtual, rastrear pontos específicos.

[1] O fluxo óptico foi utilizado em conjunto com a diferença entre quadros na demonstração de movimento de objeto virtual da seção 6.5.

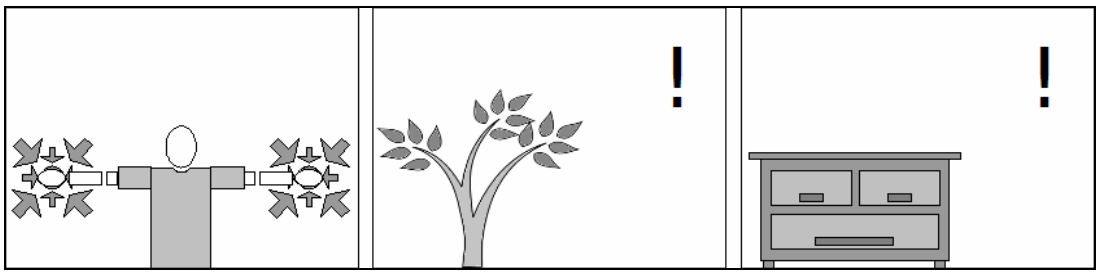
Com relação às características demonstradas neste capítulo, a identificação de contornos foi exemplificada apenas para técnicas do tipo presença. Portanto, convém mencionar ao menos uma situação onde os contornos devem ser utilizados em conjunto com segmentação do tipo movimento. Considerando um cenário virtual que ocupa toda a tela e onde não há um total *feedback* visual do corpo do usuário, de forma semelhante à da demonstração de pintura com o dedo da Figura 49, os contornos podem ser utilizados para indicar a posição do jogador quando este se movimenta.

6.6 RECOMENDAÇÕES DE DESIGN

As recomendações feitas nesta seção estão baseadas em observações de boas práticas estabelecidas por *software* comercial (SCEE, 2003) ou são sugestões do autor deste trabalho que podem vir a melhorar o desempenho técnico ou funcional de uma aplicação da API Move-In. Entenda-se como desempenho técnico uma aplicação com menor custo computacional, e depempenho funcional como uma aplicação com maiores chances de realizar com sucesso as técnicas de interação.

Primeiramente, pode-se pensar em acrescentar etapas de inicialização no início de cada aplicação de interação com *webcam*. Por exemplo, na Figura 55.A, botões virtuais do tipo “Movimento” (exemplo da Figura 35) devem ser ativados com movimentos de ambas as mãos do usuário para posicioná-la de forma correta para interagir na aplição que objetiva ter a parte superior do corpo no campo de visão da câmera.

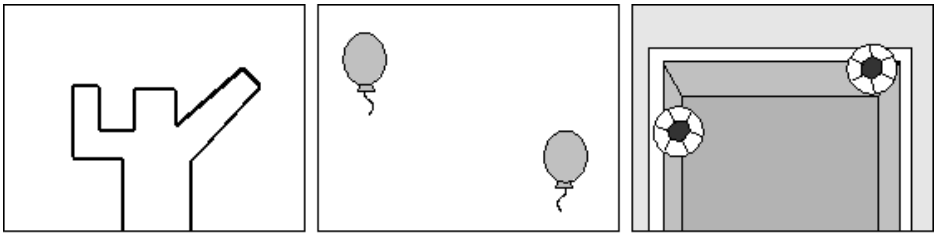
A Figura 55.B exemplifica objetos em movimento, como plantas balançando ao vento, que fazem parte do cenário e que devem ser eliminados da cena antes de iniciar uma aplicação baseada nas técnicas de movimento e fluxo óptico. De forma semelhante a Figura 55.C, estão identificados objetos que serão classificados como pele durante o processo de interação. Nesse caso, um móvel em um tom de madeira semelhante a cor da pele humana será erroneamente classificado como pele pelos algoritmos da classe *SkinColor*.



(A) **(B)** **(C)**

Figura 55 - Sugestões de inicialização para as aplicações da API Move-In.

A Figura 56 apresenta sugestões de dinâmica de jogo que visam estimular posições corporais mais simples, de forma a facilitar algoritmos de identificação de poses ou gestos. Na Figura 56.A, a aplicação sugere a posição que o usuário deve efetuar, para ganhar ou para não perder pontos. A Figura 56.B apresenta balões, que aparecem aos pares e devem ser estourados pelas mãos do usuário. De forma semelhante, a Figura 56.C mostra uma trave e bolas de futebol que devem ser defendidas.



(A) **(B)** **(C)**

Figura 56 – Sugestões de dinâmicas de jogo.

Se a questão de identificação de partes do corpo não for importante, as aplicações da Figura 56.B e da Figura 56.C podem ser implementadas apenas utilizando a técnica de movimento. Apenas a aplicação da Figura 56.A requer uso da técnica de presença para calcular a porcentagem do corpo do usuário que ocupa a posição indicada e que não ocupa espaços fora desta posição.

A técnica de identificação de cor de pele pode ser usada na aplicação de futebol da Figura 56.C para forçar o usuário a defender apenas com as mãos, braços ou cabeça. Neste caso, esta técnica seria combinada com a identificação de

movimento ao verificar, dentro da região ocupada pelo objeto virtual bola, a região com movimento que também contém cor de pele.

Ainda, se as sugestões de design mostradas na Figura 56 forem adotadas pelo programador, o processo de visão computacional responsável por identificar as partes do corpo pode ser aplicado somente nos momentos onde houver objetos de interação na tela (exemplos na Figura 56). Isso deverá reduzir consideravelmente o custo computacional da aplicação, já que o processamento de imagens será realizado em momentos específicos.

6.7 EXTENSÕES DA API MOVE-IN

Este trabalho identificou quatro funcionalidades principais para compor as classes do núcleo do sistema. Com o passar do tempo, novas funcionalidades ou necessidades podem vir a ser identificadas. Para acrescentar uma nova funcionalidade dentro do padrão que foi mostrado no diagrama da Figura 32, três questões devem ser analisadas:

1. A segmentação do objeto de interesse;
2. A extração das características;
3. O armazenamento dos dados correspondentes às características.

Os itens 1 e 2 dizem respeito à configuração com relação às classes núcleo do sistema. Se o tipo de segmentação resultar em uma imagem binária, tal como as classes *FrameDifference* e *SkinColor*, a nova classe funcionará da mesma forma com a das citadas, onde a extração das características é de responsabilidade da classe *FrameFeatures*. É possível que novos métodos de identificação das características necessitem ser adicionados a esta última. Caso seja difícil, ou menos conveniente separar os itens 1 e 2, a nova classe pode funcionar da mesma forma que a da classe *Flow*, que tem por função realizar tanto a segmentação do objeto de interesse quanto a extração das características.

Idependente disso, há de se pensar como as características deverão ser armazenadas. Se o resultado dos métodos de extração das características forem valores, como um percentual ou uma estrutura simples como um ponto, não há razão para a criação de novas estruturas. Caso contrário, se houver necessidade de armazenar um conjunto de dados que futuramente deverá ser trabalhado, é mais

conveniente que seja criada uma classe com a estrutura de dados especialmente criada para este fim.

Por exemplo, é de responsabilidade da classe *VectorPoints* armazenar o conjunto de pontos que corresponde aos vetores identificados pela técnica de fluxo óptico. Esta classe também possui métodos de soma de vetores e do cálculo do vetor resultante. Em resumo, novas funcionalidades implicarão em expansões do módulo “classes do núcleo do sistema” e, possivelmente do módulo “estruturas de dados”.

6.8 CONSIDERAÇÕES FINAIS

Este capítulo apresentou as observações extraídas a partir da implementação das hipóteses que foram propostas na seção 5.4. A falta de sucesso com relação à prova dessas hipóteses não invalida a teoria proposta.

Isto é, quanto à identificação de pontos do corpo humano, o leitor deve considerar que ainda não se chegou à implementação da simplificação de contornos proposta pela Figura 38.D, mas a comprovação desta hipótese pode ser possível através de uma abordagem diferente, ainda a ser desenvolvida, já que está baseada em um lógica semelhante a que foi aplicada na identificação da quantidade de dedos nas mãos. Esta lógica consiste no fato de é possível identificar certas poses ou posturas da dinâmica de movimento do corpo humano, caso restrições de movimento sejam impostas e respeitadas.

Já a questão do vôlei virtual está limitada pelo desempenho técnico do *hardware*, e não do conjunto de algoritmos. Também foi mostrado, através dos exemplos funcionais e das tabelas comparativas, que cada técnica atende melhor a um conjunto de tarefas, e que uma aplicação destinada ao usuário final tende à utilização de um conjunto dessas técnicas.

Além das demonstrações que foram apresentas como resultado, este capítulo fez sugestões que devem ser adotadas para facilitar o uso das aplicações que utilizem a API Move-In. Também foi indicado como se deve proceder para acionar novas funcionalidades. O capítulo 7 faz as considerações finais e sugere futuros trabalhos.

7. CONCLUSÃO

Este capítulo tem o objetivo de esclarecer como os objetivos propostos no capítulo 1 foram alcançados e discutir o que foi aprendido com a pesquisa nos trabalhos relacionados e com o desenvolvimento da API (seção 7.1). Além disso, detalhes de possíveis futuras implementações sobre a API Move-In que não foram discutidas anteriormente são discutidos na seção 7.2.

7.1 CONSIDERAÇÕES FINAIS

Os objetivos que foram propostos na seção 1.1 estão relacionados ao desenvolvimento de uma API para captura de movimentos / interação com *webcams*. Foi proposto que a API resultante seria desenvolvida através do levantamento e implementação de técnicas de visão computacional. Esses objetivos foram alcançados, tal como foi mostrado nos capítulos 3 (trabalhos relacionados), capítulo 5 (arquitetura e funcionamento da API Move-In) e capítulo 6 (resultados finais).

As técnicas relacionadas às classes núcleo da API Move-In foram levantadas a partir dos trabalhos apresentados no capítulo 3 e comparadas de forma qualitativa na seção 3.2. Nas seções 6.1 a 6.4 foram feitas demonstrações dessas técnicas através de exemplos e foram expostas suas limitações, condições de uso (seção 6.5) e sugestões de *design* para aumentar a probalibilidade de sucesso (seção 6.6) com relação à usabilidade e ao correto funcionamento.

As demonstrações implementadas nas seções 6.1 a 6.4 foram idealizadas nas principais características que o programador deve conhecer ao iniciar seus projetos com a API Move-In. Tais demonstrações também estão baseadas nas hipóteses de possíveis aplicações destinadas ao usuário final levantadas na seção 5.4. Essas hipóteses compreendiam todas as etapas de processamento de imagens relacionadas à implementação de aplicações para interação com *webcams*, inclusive a análise das características que, no contexto desta versão da API, são de responsabilidade do *designer* da aplicação.

Demonstrações completas, que abordam as etapas de aquisição das imagens de entrada, redução de dados e análise de características, são necessárias para:

1. Guiar o usuário da API quanto às possibilidades de implementação;
2. Estabeler quais técnicas são mais apropriadas para quais situações;
3. Quantificar a dificuldade de implementação de determinadas abordagens.

Por exemplo, com relação ao item 2, pode-se concluir a partir das demonstrações que as técnicas de presença carecem de ambiente controlado e que as técnicas de movimento são as mais adequadas para uso em ambiente não controlado. Isto é, as técnicas de presença exigem a correta segmentação do objeto de interesse. Para tanto, tal como foi demonstrado na seção 6.1, há de se trabalhar os limiares de segmentação (*threshold*), e possivelmente com a combinação do resultado de diferentes técnicas.

A tarefa de definir as melhores condições de segmentação é difícil de automatizar, e não se pode confiar que o usuário a realizará de forma correta. Já as técnicas de movimento, que são isentas de inicialização e menos sensíveis à variações de iluminação são mais adequadas para utilização em residências, onde não há controle do ambiente e do comportamento do usuário.

Com relação a este último fator, pode-se concluir[1] que a tarefa que deve ser realizada e suas condições de uso devem ser explicitadas antes da apresentação de tal aplicação. Por exemplo, no desenho com o dedo (Figura 49) outras partes do corpo do usuário que não sejam a mão não devem aparecer na imagem de entrada e o dedo que desenha deve estar apontando para cima para facilitar a identificação.

7.2 TRABALHOS FUTUROS

As subseções que seguem indicam possíveis trabalhos futuros a serem desenvolvidos com relação à API.

7.2.1 Arquitetura Multiprocessada

A taxa de quadros de animação por segundo nos exemplos implementados no capítulo 6 está primariamente limitada pela velocidade de atualização das imagens que a *webcam* captura. Isto é, idependentemente do modelo de

[1] Esta conclusão foi constatada a partir de observação da utilização da aplicação com usuários, realizada de maneira informal.

processador ou placa de vídeo, o FPS máximo com as câmeras utilizadas estará entre *20* e *30* quadros por segundo. Além disso, esta taxa cai com a chamada aos métodos de processamento de imagens e, dependendo do conjunto de técnicas aplicadas, o FPS pode ficar entre *6.4* e *18* quadros por segundo. É por isso que uma implementação que separa a atualização do processamento dos quadros (Figura 57) se faz necessária.

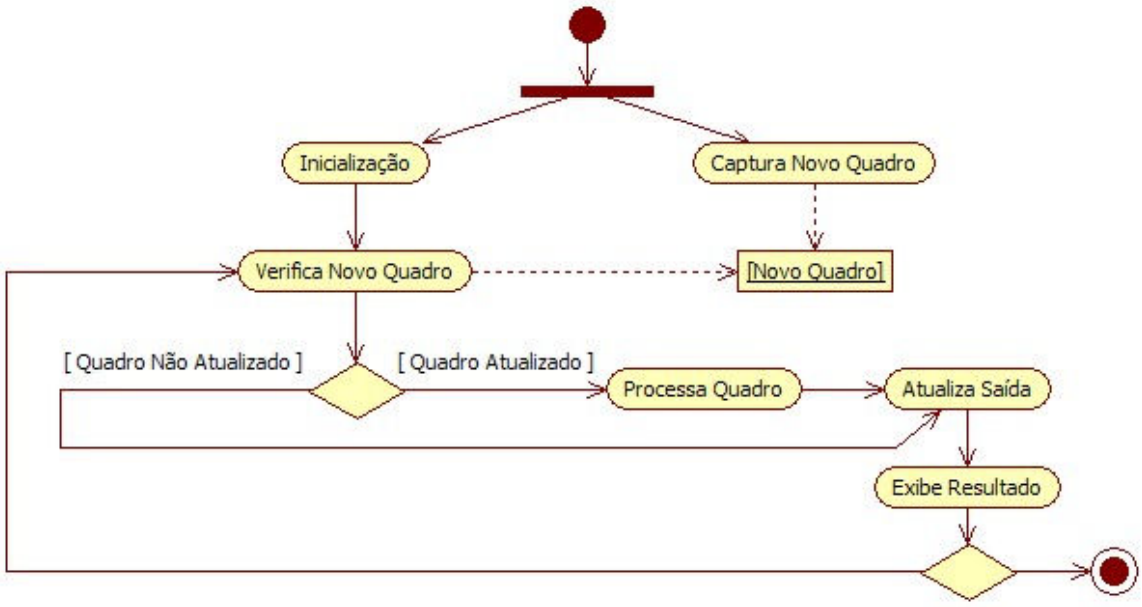


Figura 57 – Captura de quadros em processo separado.

Ainda, há de se verificar a viabilidade de criar um processo separado para cada vez que uma das técnicas é utilizada. Na versão atual da classe *Screen* a imagem binária resultante é uma só, para evitar desperdício de memória (vide seção 5.3). Para uma abordagem multiprocessada, cada processo teria que ter sua própria imagem com o resultado da segmentação e suas próprias estruturas de dados para armazenamento das características extraídas da imagem. Na versão atual, a reutilização das estruturas de dados apresentadas na subseção 5.2.3 fica a cargo do programador.

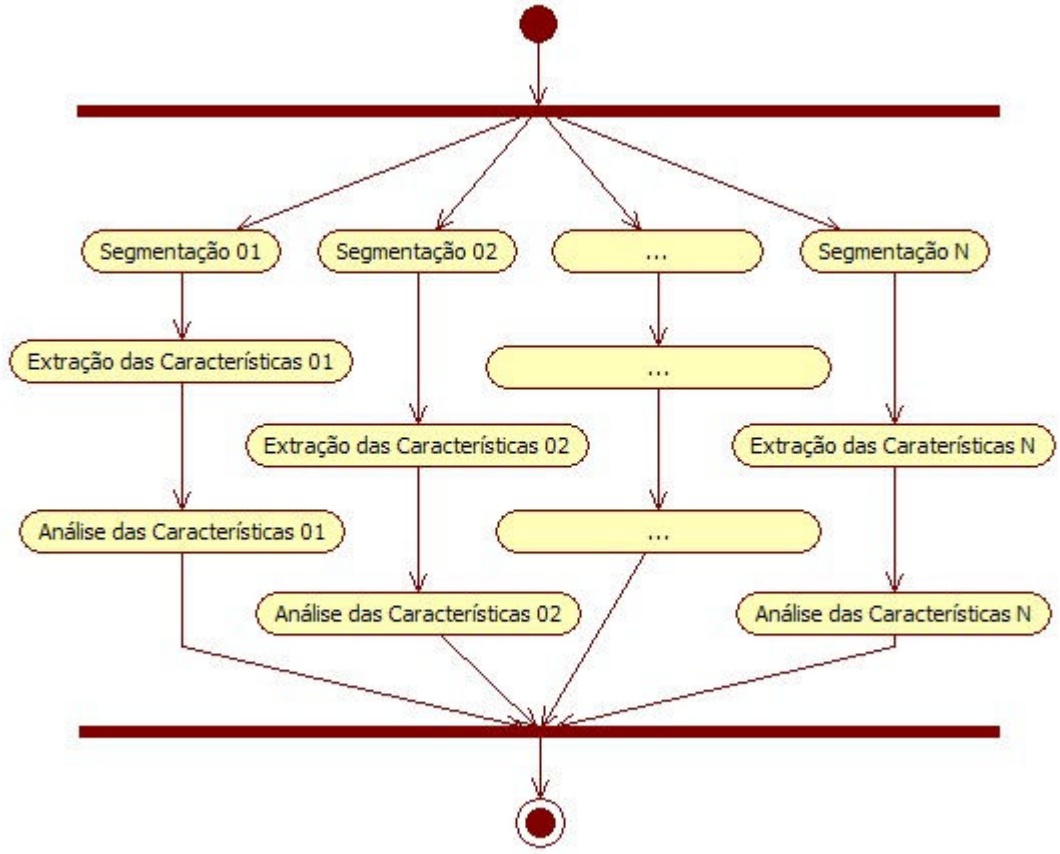


Figura 58 – Redução de dados e análise de características multiprocessados.

A implementação dessas abordagens pode ocorrer sem modificações no código atual. Como a chamada dos métodos que correspondem às técnicas de visão computacional estão centralizadas na classe *Screen*, pode-se decidir mantê-la como está, e acrescentar uma nova classe *ThreadedScreen* ou simplesmente *TScreen*. Esta classe poderia ser utilizada opcionalmente, de acordo com a necessidade do programador / *designer* da aplicação.

7.2.2 Versão para *Web*

A escolha da API OpenCV se deu pelo fato de que o processamento de imagens envolvido é computacionalmente caro (vide queda de FPS citada na subseção 7.2.1). Sendo assim, a utilização de uma API desenvolvida em C se mostrou como a melhor opção.

Para divulgar uma API pode-se criar uma página na Internet com código para os exemplos funcionais e vídeos *online* com a demonstração dos resultados. No entanto, além do código e vídeo, uma forma ainda mais rica de divulgação seria permitir ao usuário testar a aplicação através do próprio *browser*, tal como foi exemplificado na subseção 3.1.8.

Para isso será necessário estudar qual a melhor alternativa em termos de API e máquina virtual. Em princípio, as opções de linguagem de programação disponíveis seriam Java ou Actionscript. O autor deste trabalho acredita que o maior potencial está nesta última, devido à quantidade de *websites* especializados em jogos *online* com a tecnologia.

Com relação ao desempenho, da mesma forma que jogos convencionais, as versões para *web* são mais simples em termos de gráficos e jogabilidade que as versões *standalone*, e nem por isso deixam de ser interessantes. Portanto, uma alternativa para *web* seria um passo importante para aumentar o número de usuários da API. Neste caso, as aplicações online poderiam fazer uso da técnica de movimento com diferença entre quadros, por serem mais adaptáveis e robustas para ambientes não controlados.

7.2.3 Facilidade de Uso

No capítulo 2 deste trabalho foi apresentada uma figura (Figura 3) que descreve uma sequência de etapas de processamento de imagens. Esta figura foi utilizada como base teórica para no levantamento de aplicações que produzem interação a partir de imagens capturadas em vídeo ao vivo (capítulo 3) e também para a construção da arquitetura da API (capítulo 5).

Na proposta apresentada no capítulo 5, a API Move-In se destina a facilitar as estapas de redução de dados e extração das características da imagem, deixando a análise dessas características, frequentemente também referenciada como sendo “de propósito da aplicação”, a cargo do programador ou *designer*.

Um trabalho com foco nesta questão pode revelar componentes que são comuns a um grande número de aplicações, como por exemplo, os botões virtuais, um conjunto de ROIs destinada à movimentação de objetos virtuais e a substituição de cenário de fundo por cenário virtual. A partir desta idéia, pode-se considerar

como trabalho futuro a construção de classes que utilizam componentes pré-construídos para imediata utilização.

Esses componentes podem ser acrescentados em um novo módulo adicional (vide Figura 32), chamado de componentes pré-construídos (*pre-build components*). A opção pelo uso desses componentes deverá faciliar ainda mais a construção de aplicações para interação com *webcam*.

REFERÊNCIAS

ADOBE. **Introducing the Image API in Flash 8**. 2005. Disponível em: http://www.adobe.com/devnet/flash/articles/image_api.html. Acessado em novembro de 2007.

BAKER, Simon. MATTHEWS, I. Lucas-Kanade 20 Years On: A Unifying Framework. In: **International Journal of Computer Vision**, Vol. 56, No. 3, March, 2004, pp. 221 – 255. Também disponível em: http://www.ri.cmu.edu/projects/project_515.html. Acesso em dezembro de 2007.

BARBER, C. B. DOBKIN, D. P. HUHDANPAA, H. **The Quickhull Algorithm for Convex Hulls**. In: ACM Transactions on Mathematical Software, Vol. 22, No. 4, pp 469-483, 1996.

BOURKE, Paul. **YCC Colour Space and Image Compression**. Abril de 2000. Disponível em: http://ozviz.wasp.uwa.edu.au/~pbourke/texture_colour/ycc/. Acessado em março de 2008.

CLINE, Bryan. LARKINS, D. Brian. **Dogfight: A Computer Vision Controlled Flight Simulator**. Final Project for a Computer Vision Course. Ohio State University, 2004. Também disponível em: http://www.cse.ohio-state.edu/~clineb/cse694i/paper.pdf. Acessado em março de 2008.

CVBLOBSLIB. **Blob Extraction Library**. Versão 6 publicada em março de 2007. Disponível em: http://opencvlibrary.sourceforge.net/cvBlobsLib. Acessado em maio de 2008.

D'HOGE, Herman. GOLDSMITH, Michael. Game Design Principles for the IntelPlay Me2Cam Virtual Game System. **Intel Technology Journal**. 2001.

DA SILVA, Fernando Wagner. **Motion Capture - Introdução à Tecnologia**. Internal report (preprint), LCG - COPPE/SISTEMAS, UFRJ, abril de 1997.

DE PAULA, Luis Roberto Pereira. BONINI NETO, Renato. DE MIRANDA, Fábio R. **Camera Kombat - Interação Livre para Jogos**. SBGames 2006.

DE SOUZA, Eder Jonck. **Uma Alternativa de Interface Homem-Máquina Utilizando uma Câmera de Baixo Custo**. Trabalho de Conclusão de Curso. Universidade do Estado de Santa Catarina – UDESC, 2005.

EXTENDED REALITY. Disponível em: http://www.extendedreality.com/webcam_games_info.html. Acesso em novembro de 2007.

FISHER, John. **Visualizing the Connection Among Convex Hull, Voronoi Diagram and Delaunay Triangulation**. In: Proceedings of Midwest Instruction and Computing Symposium, University of Minnesota, Morris, April, 2004.

FONSECA, Pedro. NESVADBA, Jan. **Face Detection in the Compressed Domain**. In: International Conference on Image Processing. Volume 3, Issue, pp: 2015 – 2018, October, 2004.

FOURCC. **YUV Formats**. Disponível em: http://www.fourcc.org/yuv.php. Acesso em dezembro de 2007.

GALVIN, B. MCCANE, B. NOVINS, K. MASON, D. MILLS, S. **Recovering Motion Fields: An Evaluation of Eight Optical Flow Algorithms**. In: Proceedings of the British Machine Vision Conference 1998, BMVC 1998, Southampton, UK, 1998.

GAMMA E. HELM R. JOHNSON R. VLISSIDES J. **Design Patterns: Elements of Reusable Software**. Addison-Wesley, 1995.

GASPARINI, Francesca. SCHETTINI, Raimondo. **Skin Segmentation Using Multiple Thresholding**. Publicação online: SPIE - The International Society for Optical Engineering. Publicado em 16 Janeiro de 2006.

GONZALEZ, Rafael C. WOODS, Richard E. **Digital Image Processing**, Addison Wesley, 2000.

HITLAB. **ARToolkit**, 1999. Disponível em: http://www.hitl.washington.edu/artoolkit/. Acessado em julho de 2007.

HORAIN, P. BOMB, M. **3D Model Based Gesture Acquisition Using a Single Camera**. IEEE Workshop on Applications of Computer Vision (WACV 2002), Orlando, Florida, December, 2002.

ICAMERAPHONES. **Camera Mobile Resolution**, 2005. Disponível em: http://www.icameraphones.com/buying-guide/camera-resolution.htm. Acessado em outubro de 2007.

INTEL. **Open Source Computer Vision Library - Reference Manual**. USA, 2001.

JACK, Keith. **Video Demystified: A Handbook for Digital Engineer**. LLH Technology Publishing, 3rd Edition, 1995.

JAHNE, Bernd. **Digital Image Processing**. Springer, 5th Edition. Berlin, 2002.

JONES, M. REHG, J. **Statistical Color Models with Application to Skin Detection**. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR '99, pp. 274-280, 1999.

KIZONY, R. KATZ, N. WEINGARDEN, H. WEISS, P.L. **Immersion without encumbrance: adapting a virtual reality system for the rehabilitation of individuals with stroke and spinal cord injury**. In: Proceedings of the 4th International Conference on Disability, Virtual Reality and Associated Technologies, Vezprem, Hungary, September, 2002.

KIRK, Adam G. O'BRIEN, James F. FORSYTH, David A. **Skeletal Parameter Estimation from Optical Motion Capture Data**. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2005.

LUCAS, B. KANADE, T. **An Iterative Image Registration Technique with an Application to Stereo Vision**. In: Proceedings of Imaging Understanding Workshop, pp. 121-130, 1981.

MIDWAY. **Mortal Kombat 2**, 1992. Análise disponível em: http://www.gamespot.com/arcade/action/mortalkombat/. Acessado em novembro de 2007.

MOVE-IN API. ***Move-In is an API for development of applications that use body movements as input data***. Disponível em: http://code.google.com/p/move-in/. Acessado em março de 2008.

OKADA, Ryuzo. KONDOH, Nobuhiro. STENGER, Bjorn. **A Video Motion Capture System for Interactive Games**. Proc. Machine Vision Applications (MVA), pages 186-189, Tokyo, Japan, May 2007.

OPENCV. **Open Source Computer Vision – OpenCV**. Versão 1.0. Novembro de 2006. Disponível em: http://sourceforge.net/projects/opencvlibrary/. Acesso em julho de 2007.

OPENCV OVERVIEW. **Visão Geral das funcionalidades da biblioteca OpenCV**. Disponível em: http://www.intel.com/technology/computing/opencv/overview.htm. Acessado em dezembro de 2007.

OPENCV YAHOO GROUPS. **Grupo de discussão da biblioteca OpenCV**. Junho de 2000. Disponível em: http://tech.groups.yahoo.com/group/OpenCV/. Acessado em março de 2008.

PEDRINI, Hélio. SCHWARTZ, William Robson. **Análise de Imagens Digitais – Princípios, Algoritmos e Aplicações**. Thomson Learning, São Paulo, 2008.

PEER, P. KOVAC, J. SOLINA, F. **Human Skin Colour Clustering For Face Detection**. In: EUROCON 2003 – International Conference on Computer as a Tool, 2003.

PISAREVSKY, Vadim. **Introdution to OpenCV**. Junho de 2007. Disponível em: http://fsa.ia.ac.cn/files/OpenCV_China_2007June9.pdf. Acessado em março de 2008.

PREPARATA, F. P. SHAMOS, M. I. **Computational Geometry: An Introduction**. 2nd Edition, Springer-Verlag, 1988.

RICHARDSON. E. G. Iain. **H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia**. John Wiley & Sons ltd., England. 2203.

SEBE, Nicu. COHEN, Ira. GARG, Ashutosh. HUANG, Thomas S. **Machine Learning in Computer Vision**. Springer. New York, 2005.

SCEE, Sony Computer Entertainment Europe. **Eye Toy**. 2003. Disponível em: http://www.us.playstation.com/PS2/Games/EyeToy_Play/OGS/. Acessado em julho de 2007.

SCEE-Antigrav, Sony Computer Entertainment Europe. **Eye Toy Antigrav**. 2003. Avaliação disponível em: http://www.gamespot.com/ps2/action/eyetoy/. Acessado em novembro de 2007.

SCEE-Play, Sony Computer Entertainment Europe. **Eye Toy Play**. 2003. Disponível em: http://www.gamespot.com/ps2/sports/eyetoyantigrav/. Acessado em novembro de 2007.

SEMENTILLE, Antonio C. LOURENÇO, Luís E. BREGA, José R. F. RODELLO, Ildeberto. **A Motion Capture System Using Passive Markers**. In: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry, SESSION: 8-1 Human-computer interaction & tracking, 2004.

SHI, Jianbo. TOMASI, Carlo. **Good Features to Track**. IEEE Conference on Computer Vision an Pattern Recognition, pages 593–600. Seatle, Junho de 1994.

STAPLES, Joshua A. DAVIS, Timothy A. Motion Capture for the Masses. **INFOCOMP Journal of Computer Science**, July, 2006.

STATE, Andrei. HIROTA, Gentaro. CHEN, David T. GARRETT, Willian F. LIVINGSTON, Mark A. **Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking.** In: Proc. Siggraph 96, ACM Press, New York, 1996.

STAVENS, David. **Introduction to OpenCV**. Standford Artificial Intelligence Lab, 2007. Disponível em: http://ai.stanford.edu/~dstavens/cs223b/. Acessado em novembro de 2007.

UMBAUGH, Scott E. **Computer Vision and Image Processing: A Practical Approach Using Cviptools with Cdrom**, Prentice Hall PTR, Upper Saddle River, NJ, 1997.

VASSALO, R. F. FRANCA, A. S. SCHNEEBELI, H. A. **Detecção de Obstáculos através de um Fluxo Óptico Padrão Obtido a Partir de Imagens Omnidirecionais**. Anais do VII Simpósio Brasileiro de Automação Inteligente - VII SBAI - II IEEE LARS, 7, 2005.

VEZHNEVETS V. SAZONOV V. ANDREEVA A. **A Survey on Pixel-Based Skin Color Detection Techniques**. In: Proc. Graphicon-2003, pp. 85-92, Moscow, Russia, September 2003.

VICON APPLICATIONS. **Sports Performance**. Disponível em: http://www.vicon.com/applications/sports.html. Acesso em setembro de 2007.

WOOTTON, Cliff. **A Practical Guide to Video and Audio Compression**. Focal Press, USA, 2005.

WU, Andrew. SHAH, Mubarak. LOBO, Vitoria. **A Virtual 3D Blackboard: 3D Finger Tracker Using a Single Camera**. In: Fourth IEEE International Conference on Automatic Face and Gesture Recognition March 26-30. Grenoble, France, 2000.

ZELNIK, Lihi. **Optical Flow Lesson**. 2005. Disponível em: http://www.vision.caltech.edu/lihi/Presentations/. Acessado em outubro de 2007.

ZIVKOVIC, Zoran. **Optical-flow-driven Gadgets for Gaming User Interface**. In: Proceedings of the 3rd International Conference on Entertainment Computing, 2004.