Camera Kombat - Interação Livre para Jogos

Luis Roberto Pereira de Paula Renato Bonini Neto Fábio R. de Miranda*

Centro Universitário Senac, Bacharelado em Ciência da Computação, São Paulo – SP, Brasil

Figura 1: Projeto Camera Kombat. (a) *Screenshot* do jogo. (b) Visualização em modo *debug*.

Resumo

Este trabalho mostra detalhes da criação do Camera Kombat - um jogo multijogador que permite, através do emprego de recursos de visão computacional, uma forma de interação livre entre o jogador e o sistema, que dispensa a utilização de dispositivos convencionais, como *joysticks*, mouse, teclado, dentre outros. Para atingir tal fim, gestos realizados pelo jogador diante de uma *webcam* são mapeados em ações no jogo.

Palavras-chave: Interação humano-computador, visão computacional, jogos eletrônicos, computação gráfica.

Contatos:
luisrpp@gmail.com
rbonininetto@gmail.com
* fabio.rmiranda@sp.senac.br (orientador)

Vídeo da aplicação:
http://tinyurl.com/nxyjx

Abstract

This work presents details of the creation of Camera Kombat, a two-player game that allows users to participate in an unemcumbered way, without the mediation of traditional input devices, such as joysticks, mouses and keyboards. In order to enable this level of interaction, computer vision techniques are used to recognize gestures in images of the user captured by a webcam.

Keywords: Human–computer interaction, computer vision, electronic games and computer graphics.

1. Introdução

Há muitas oportunidades para criação de novos jogos, especialmente os com propostas mais inovadoras em termos de roteiro, temáticas e também interfaces. Estas propostas inovadoras são as que apresentam maior potencial para atrair novos consumidores ao mercado de games.

Uma característica que ainda se encontra numa etapa inicial para a maioria dos jogos atuais é a utilização de tipos de interação mais diversificados entre o jogador e o jogo. Atualmente são mais comuns jogos que exploram apenas o modelo convencional, em que a interação é feita por um *joystick* ou teclado e mouse.

Esta modalidade tradicional de interação favorece uma postura sedentária prolongada, em que pouca atividade física é realizada durante o jogo, que pode durar muitas horas. Também são um problema característico deste tipo de interação com teclado e mouse as lesões por esforços repetitivos, que podem acometer jogadores de vídeo games.

Neste trabalho é proposta a exploração de um tipo diferente de interação, que faça com que não apenas um avatar se movimente, mas também o jogador, com seus braços, pernas, ou mesmo com todo seu corpo.

O caminho explorado neste trabalho para permitir este tipo de interação é a utilização de uma *webcam* para a captura de imagens do jogador, um posterior processamento por meio de visão computacional e a composição de tais imagens em tempo real com imagens do cenário do jogo.

2. Objetivos

O objetivo deste projeto é a construção de um jogo de combate virtual entre duas pessoas que apresente uma forma de interação que incentive maior atividade física por parte de seus participantes.

Figura 2: Exemplo do funcionamento da aplicação

Neste jogo de combate, os golpes são definidos como seqüências de movimentos realizados pelo jogador através de seu próprio corpo, utilizando os braços e pernas. Quando uma destas seqüências for identificada como uma das pré-definidas no jogo, o jogador que a realizou dispara um objeto virtual, como um raio de luz ou uma bola contra seu adversário.

Na Figura 2 pode-se observar o exemplo de golpes lançados pelos jogadores, um contra o outro. Estes golpes são representados pela bola de fogo, ao centro da figura.

3. Trabalhos Relacionados

Um trabalho pioneiro sobre interação em tempo real com imagens de câmeras é o Artificial Reality de Myron W. Krueger, de 1969. Neste trabalho (KRUEGER, 1991) são descritos e apresentados estudos e desenvolvimentos sobre meios inovadores para interação com computadores através do uso dos próprios movimentos humanos, sem a necessidade de qualquer hardware acoplado à pessoa. Tais movimentos eram capturados por uma câmera de vídeo e utilizados como entrada para o sistema.

Em Videoplace (KRUEGER, 2006), Krueger demonstrou um sistema para permitir que uma pessoa interagisse com objetos virtuais apresentados através de gráficos gerados por um computador.

A estrutura deste sistema era formada por uma câmera usada para captura da imagem, um computador para o processamento, um projetor responsável pela exibição e uma tela de plástico translúcida para ajudar o computador a distinguir a silhueta de uma pessoa do fundo da imagem.

Krueger conseguiu demonstrar o potencial de mais de 50 modos diferentes de interação através de seus testes, por meio da construção de protótipos em que a pessoa interagia com objetos virtuais de alguma forma, por exemplo desenhar imagens com a ponta dos dedos, brincar em um ambiente virtual com uma bolha, tentar dançar seguindo movimentos mostrados em um monitor ou interagir com balões virtuais que flutuam livremente.

Outro trabalho recente com uma abordagem experimental para o uso de câmeras como dispositivo de entrada é WARREN, 2003, em que foram desenvolvidos cinco jogos em que o jogador pode interagir de diversas maneiras com blocos virtuais. Todos os jogos desenvolvidos nesse trabalho têm como base de sua construção a detecção de colisão entre a silhueta da pessoa com os objetos virtuais.

No célebre Kids Room (BOBICK, 2000), realizado no MIT, foi desenvolvida uma sala com o intuito de atrair a atenção de crianças e fazê-las interagir em um ambiente fechado. Trata-se de um quarto que conta histórias infantis por meio de imagens projetadas nas paredes, sons e luzes ambientes para aumentar a imersão das crianças na história. Neste trabalho se utilizava visão computacional para promover a interação com objetos virtuais e para controlar a intensidade necessária de esforço físico exercido pelas crianças que estão no quarto, dependendo de quão ativas elas estejam.

Comercialmente já foram realizados alguns investimentos em jogos baseados em visão computacional. Para promover o lançamento de uma *webcam* própria, a Apple Computers Inc. lançou uma coleção de games intitulada TOYSIGHT (2006) que utilizam a captura de imagens dos jogadores como meio de interação. Também destaca-se o EYE TOY (2005), lançado pela Sony para seu console PlayStation 2. Trata-se de uma câmera, que também funciona como *webcam*, que captura a imagem do jogador e traz para a tela seus movimentos. Esta é uma tentativa de introduzir ao mercado de videogames jogos baseados em visão computacional. Entretanto, ainda há poucos títulos que utilizam este dispositivo.

4. Projeto

Este projeto, intitulado Câmera Kombat (C.K.), propõe uma forma de interação, em jogos eletrônicos, livre de dispositivos de entrada convencionais. Essa interação se dá pela captura de gestos executados por um jogador frente a uma *webcam*. Estes gestos são identificados pelo sistema, gerando ações no game.

Para jogar, os jogadores devem ficar afastados a uma distância que evite o risco de contato físico não-intencional e acidentes entre os mesmos. À medida que os gestos são identificados (como um soco, por

exemplo), é lançado (a partir do jogador que realizou o gesto) um objeto virtual sobre o adversário. O adversário, para se defender, deve desviar destes objetos.

Embora em Camera Kombat os jogadores precisem ficar afastados uns dos outros, ambos precisam ser enquadrados por uma mesma *webcam.* Porém, o sistema pode ser adaptado facilmente para capturar imagens de duas *webcams,* uma para cada jogador. Isto resolveria o problema de afastar os jogadores e permitiria golpes mais enérgicos como socos e chutes sem que seja necessário ter cuidado para evitar o contato físico.

Este trabalho explora o uso de técnicas de visão computacional aplicadas sobre uma seqüência de vídeo capturada em tempo real para criar um ambiente de realidade aumentada, em que os elementos virtuais são adicionados a uma representação aumentada do mundo real.

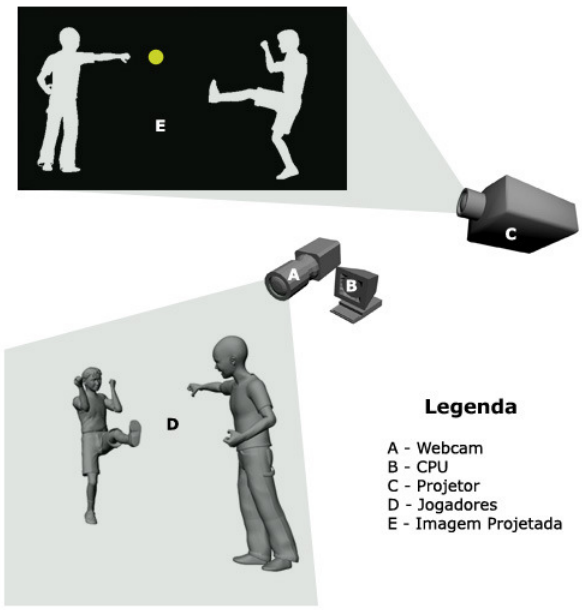

Figura 3: Visão esquemática do Sistema

4.1 Ferramentas Utilizadas

Para acesso a implementações de qualidade de técnicas e métodos de visão computacional, foi utilizada neste projeto a biblioteca OpenCV, da Intel. Esta biblioteca possui uma série de algoritmos preparados para trabalhar em aplicações que demandam processamento de imagens e técnicas de visão computacional em tempo real. O uso da OpenCV viabilizou a realização deste projeto, uma vez que não foi preciso gastar esforços na implementação de técnicas avançadas de visão computacional.

Além da OpenCV, foi utilizada a biblioteca OpenGL e sua biblioteca GLUT (Graphics Library Utility Toolkit) (KILGARD, 2006). Estas bibliotecas foram utilizadas para a criação de janelas, carregamento de texturas, criação e visualização de objetos virtuais que compõem o jogo.

4.2 Arquitetura

O sistema necessita de uma *webcam* para capturar os movimentos do jogador. Estas imagens são analisadas por um computador, processadas e exibidas num monitor. Existe a possibilidade de projetar a imagem gerada em uma parede, facilitando assim para os jogadores, a visualização dos elementos com os quais pode-se interagir. Na Figura 3 é apresentado um modelo da configuração do sistema, em que na parte inferior estão representados os jogadores no mundo real e, na parte superior, uma tela que contém a silhueta dos jogadores e um objeto virtual.

4.2.1 Módulos da Aplicação

Câmera Kombat é constituído basicamente por três módulos, que são: Processamento de Imagem (Visão), Processamento Gráfico e Controles.

No módulo de processamento de imagens (visão) são realizadas a captura da imagem a partir de uma *webcam*, a segmentação por objetos de interesse (jogadores) e o reconhecimento de gestos.

Em processamento gráfico são realizadas a criação dos objetos gráficos e suas animações.

No módulo de controle são definidas as regras do jogo e é estabelecido um canal de comunicações entre os módulos de processamento de imagem e gráfico.

4.2.2 Portabilidade

Todo o desenvolvimento foi realizado na plataforma Linux, mas não há nenhuma dependência exclusiva em relação a este sistema operacional. Basta recompilá-lo para que seja executado no sistema operacional Windows ou em Mac OS X, uma vez que seus pré-requisitos, especialmente OpenCV (OPENCV, 2005) e OpenGL (OPENGL REFERENCE MANUAL, 2005), são multi-plataforma.

4.3 Implementação

Nesta sessão, serão apresentados os detalhes de implementação deste projeto, bem como a técnica de subtração de fundo adotada, reconhecimento de gestos, detecção de colisões e outras peculiaridades do jogo.

4.3.1 Linguagem de Programação e Paradigma

Câmera Kombat foi desenvolvido em linguagem C, sob o paradigma de programação estruturada. Isto deu-

se principalmente pelo fato das bibliotecas utilizadas (OpenCV e OpenGL) permitirem um acesso mais simples e direto a partir desta linguagem, por serem desenvolvidas também em linguagem C.

Embora o código esteja sob forma estruturada, as funções foram bem modularizadas, de maneira que a conversão para o paradigma orientado à objetos não seria muito custoso.

4.3.2. Metodologia

Este projeto tomou inspiração em metodologias ágeis de desenvolvimento de software e seguiu-se a estratégia da elaboração de pequenos protótipos para a resolução de problemas independentes, como a inicialização, segmentação de imagens, criação e animação de objetos gráficos, detecção de colisões, dentre outros. A partir destes protótipos, foram desenvolvidas as técnicas utilizadas na aplicação, que serão expostas a seguir.

4.3.3 Fase de Inicialização

Ao iniciar a aplicação, antes de serem efetuadas as etapas de subtração de fundo, reconhecimento de gestos, regras do jogo, e outras posteriores, é preciso que duas condições sejam respeitadas. A primeira refere-se às *webcams*, e a segunda, aos elementos presentes na imagem capturada durante a inicialização do sistema.

Muitas das *webcams* atuais possuem o recurso de ajuste automático de brilho e contraste da imagem face às variações na iluminação do ambiente. Este é um recurso muito interessante, pois evita que correções na imagem precisem ser realizadas via software, porém, esse tipo de ajuste requer um tempo de adequação às condições do ambiente. Por isso, é preciso esperar que o ajuste automático se estabilize para que não prejudique as etapas posteriores no processamento, como a segmentação da imagem.

A segunda condição que precisa ser satisfeita refere-se aos elementos presentes na imagem capturada durante a inicialização do sistema. Neste momento é importante que haja no ambiente somente elementos estáticos e que os usuários estejam fora da visão da câmera. Todos esses elementos estáticos da imagem serão classificados como fundo durante a subtração. Caso haja elementos não estáticos, estes poderão ser classificados erroneamente como um objeto de interesse.

Figura 4: Imagem com luminosidade estabilizada e elementos estáticos

Na Figura 4 é apresentada uma imagem capturada que atende as duas condições. Neste projeto, esta etapa é realizada manualmente pelo usuário, que espera a estabilização da imagem e em seguida pressiona uma tecla no momento mais adequado. A partir deste evento, a subtração de fundo passa a ser realizada.

4.3.4 Subtração de Fundo

Em Câmera Kombat, a subtração de fundo foi implementada utilizando uma técnica de subtração simples. As etapas da subtração de fundo estão implementadas na função *doBackgroundSubtraction*, que pode ser vista na Figura 6.

Nesta implementação, a imagem utilizada como fundo é determinada pelo usuário através de um evento teclado. Este evento pode ser invocado tanto na fase de inicialização do sistema, como em qualquer outro momento que se queira atualizar o fundo da imagem.

Determinado o fundo, é efetuada uma subtração absoluta, pixel a pixel, do frame atual com a imagem de fundo.

Após a realização da subtração de fundo, é aplicada uma operação de limiarização, em que foi adotado como limiar o valor 40, ou seja, todos os pixels com intensidade inferior a este valor serão convertidos para preto (de modo a caracterizar o fundo da imagem) e os com intensidade superior ao limiar, para branco (elementos segmentados).

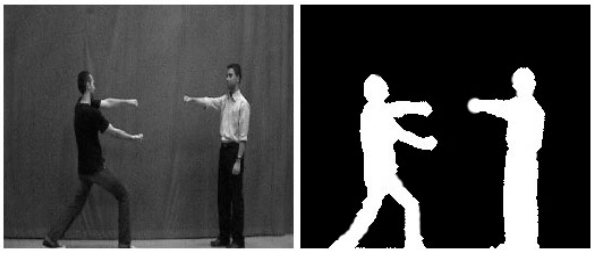

Figura 5: Aplicação da técnica de subtração simples com limiarização

Para melhorar os resultados da subtração de fundo, foi aplicado sobre a imagem binarizada a operação morfológica de abertura (erosão seguida de dilatação), com um elemento estruturante quadrado de dimensões 3 x 3 pixels. A aplicação da operação de abertura elimina pequenas ilhas na imagem e remove, então, alguns ruídos indesejáveis. O resultado dessas operações pode ser observado na Figura 5.

```
IplImage* doBackgroundSubtraction(IplImage* image){
   /* Alocando espaço para a imagem subtraída */
   IplImage*            subtraction            =
cvCreateImage(cvGetSize(image), 8, 1);

   /* Efetua a subtração absoluta pixel a pixel do
background com o frame atual */
   cvAbsDiff(image, background, subtraction);

   /* Aplicando o threshold na imagem subtraida */
   cvThreshold(subtraction,  subtraction,  40,  255,
CV_THRESH_BINARY);

   /* Definição do elemento estruturante 3x3 pixels
*/
   IplConvKernel*             matrix             =
cvCreateStructuringElementEx(3,     3,     0,     0,
CV_SHAPE_RECT, NULL);

   /* Erosão aplicada na imagem subtraída */
   cvErode(subtraction, subtraction, matrix, 1);

   /* Dilatação */
   cvDilate(subtraction, subtraction, matrix, 2);

   return subtraction;
}
```

Figura 6: Função responsável pela subtração de fundo

Embora tenha sido utilizada a técnica de subtração simples de fundo neste projeto, a mesma foi desenvolvida de forma que possibilite que outros métodos possam ser utilizados sem afetar o funcionamento do sistema. Para isso, é preciso somente reescrever a função *doBackgroundSubtraction*.

4.3.5 Reconhecimento de Gestos

Tratando-se de um jogo de luta, é importante que determinados gestos sejam reconhecidos, como socos, por exemplo. Para a identificação destes gestos, foram elaborados para o Câmera Kombat métodos baseados em formas geométricas presentes na silhueta dos jogadores para extração de tal informação.

Como primeiro passo, é preciso localizar algumas regiões do jogador importantes para determinar a ocorrência dos gestos de interesse. Essas regiões são: cabeça, posição das mãos, regiões próximas aos pés e ao centro de massa.

Para a determinação destas regiões, foi utilizada a técnica do fecho convexo (descobrir o menor polígono convexo que envolve todos os pontos da silhueta da pessoa), tendo como conjunto de pontos o contorno da silhueta do jogador. Ambas as técnicas são providas pela biblioteca OpenCV e foram utilizadas neste projeto, sendo que a função responsável pelo contorno recebe como parâmetro uma imagem que representa o resultado da subtração de fundo e a função do fecho convexo recebe o conjunto de pontos que representam o contorno e retorna os vértices do polígono.

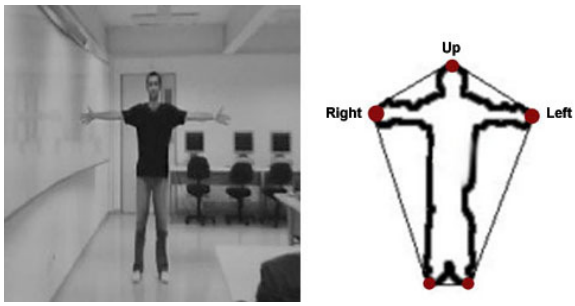

Figura 7: Pontos detectados a partir do fecho convexo

Na Figura 7, são evidenciados os vértices encontrados a partir do fecho convexo. Encontrados estes vértices, as regiões de interesse são definidas como:

- Cabeça: vértice posicionado mais acima, representado por *Up*, na Figura 7.
- Mão Esquerda: vértice posicionado mais à esquerda, representado por *Left*, na Figura 7.
- Mão Direita: vértice posicionado mais à direita, representado por *Right*, na Figura 7.
- Região Próxima aos Pés: é composto pelo valor x do vértice *Up* e pelo valor y do vértice posicionado mais abaixo na figura.
- Região Próxima ao Centro de Massa: é composto pelo valor x do vértice *Up* e o valor y é determinado pela média aritmética entre o valor y do vértice *Up* com o y do vértice da região mais próxima dos pés.

Na Figura 8 é apresentada a estrutura que representa os pontos das regiões de interesse. Durante o jogo, cada jogador possui um elemento do tipo DetectedPoints, que é atualizado a cada frame.

```
typedef struct _DetectedPoints{
          CvPoint right;
          CvPoint left;
          CvPoint center;
          CvPoint up;
          CvPoint down;
} DetectedPoints;
```

Figura 8: Estrutura que contém os pontos das regiões de interesse

Detectar os vértices das regiões de interesse é uma etapa importante, mas insuficiente para a verificação da ocorrência de um gesto, como um soco. Para determinar se um soco ocorreu ou não, é analisado o ângulo ɵ formado pela abertura dos braços em relação à cabeça e à região próxima ao centro de massa. O ângulo ɵ pode ser observado na Figura 9.

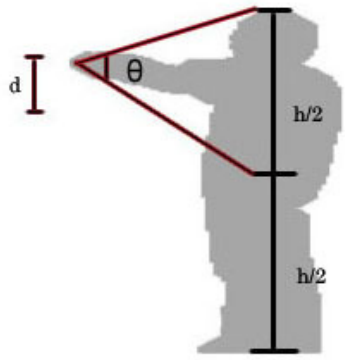

Figura 9: Condições para detecção de socos

Foi observado durante experimentos que quando o ângulo ɵ assumia valores menores que 95º, pequenos movimentos dos braços eram detectados. Porém, estes movimentos ainda não caracterizam um soco. Para determinar a ocorrência de um soco, foi adicionada uma nova condição, que é o estabelecimento de uma região em y onde é mais provável que este gesto aconteça. Esta região onde é mais provável que um soco ocorra é representada na Figura 9 por d.

Seja *detPoints* um elemento do tipo *DetectedPoints*, os limites de d , em y, são determinados por *d_min* e *d_max*, apresentados na Figura 10.

```
d_min = (detPoints.center.y + detPoints.up.y)
/ 2) * 0.9);

d_max = (detPoints.center.y + detPoints.up.y)
/ 2) * 1.1);
```

Figura 10: Intervalo em y com maior probabilidade de ocorrência de socos

Portanto, para a detecção de socos, foram estabelecidas duas condições que devem ocorrer ao mesmo tempo:

- ɵ < 95º;
- *detPoints.right.y* > *d_min* e *detPoints.right*.y < *d_max*, para o braço direito do jogador, ou *detPoints.left.y* > *d_min* e *detPoints.left.y* < *d_max* para o braço esquerdo do jogador.

Na Figura 11, é mostrada em (a) e (b) uma tentativa para a detecção de gestos de soco levando em consideração somente o ângulo ɵ. Como observado, o gesto é identificado para qualquer pequeno afastamento das mãos (ou braços) em relação ao corpo. Em (c) e (d), foi adicionada a regra de detectar gesto como soco apenas quando ocorre na região em que se considera mais provável em que um soco ocorra. Pode-se observar que os resultados obtidos foram mais precisos. O braço direito do jogador, que está levemente afastado do corpo não é identificado, enquanto o braço esquerdo que se encontra numa posição mais elevada, é reconhecido como um soco.

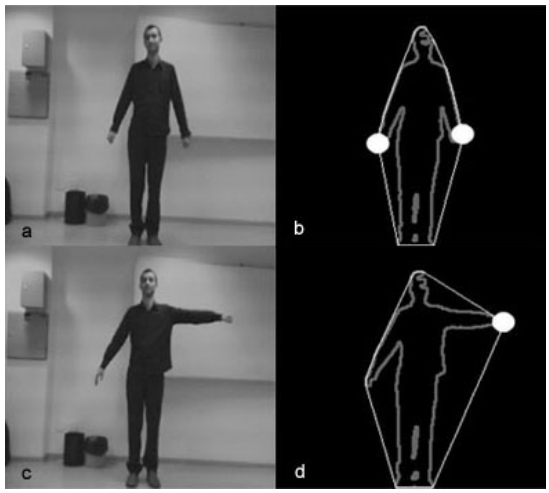

Figura 11: Reconhecimento de gestos de soco sem restrições em relação à região de ocorrência (superior) e com restrições (inferior)

4.3.5.1 Identificando Dois Jogadores

Até o momento foi mostrado como identificar um jogador em uma cena e reconhecer socos realizados pelo mesmo. Porém, em Câmera Kombat, por tratar-se de um jogo de luta (confronto entre dois participantes), é necessário que o sistema saiba identificar e reconhecer os gestos de cada um dos jogadores de forma independente.

Para identificar cada jogador, foram utilizados os contornos das silhuetas dos mesmos, obtidos após a fase de subtração de fundo. Ao obter todos os pontos do contorno, são identificados o ponto mais à esquerda e o mais à direita na tela. Destes, é definida uma região central, que é a distância média entre os dois pontos (Figura 12). Assim, todos os pontos da região à esquerda pertencem ao jogador 1, e os da direita, ao jogador 2. Definido o conjunto de pontos da silhueta de cada jogador, estes são tratados individualmente pelo sistema, podendo a partir deste ponto aplicar o fecho convexo e o reconhecimento de gestos (socos).

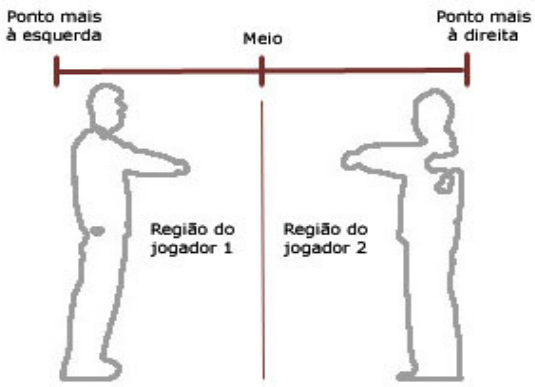

Figura 12: Divisão de áreas dos jogadores

4.3.5.2 Ação após Detecção de Gesto

Ao ser detectado pela aplicação um gesto de soco, um objeto (bola) é lançado a partir da mão do jogador em direção a seu adversário.

Para isto, a cada gesto detectado, um objeto é criado na posição correspondente a mão do jogador, representado na estrutura *DetectedPoints* por *right* ou *left*, dependendo da posição do jogador. Nos quadros seguintes, ocorre uma movimentação deste objeto, na horizontal, em direção a seu adversário, podendo seguir até sumir da tela, ou colidir em algum objeto, ou até mesmo no adversário.

A técnica utilizada para a detecção de colisões será apresentada a seguir.

4.3.5.3 Detecção de colisões

Na realização da detecção de colisões entre objetos virtuais e silhuetas de jogadores é necessário conhecer a localização de todos os objetos que se pode interagir em uma cena. Para isto, foram definidas algumas estruturas de dados com a finalidade de descrever para a aplicação como são estes objetos e em que posições se encontram.

Em Câmera Kombat, optou-se por desenvolver estruturas de dados que unissem todas as informações necessárias ao processamento do jogo com as características dos objetos.

Para descrever uma bola, objeto que representa os golpes efetuados pelos jogadores no jogo, foi utilizada a estrutura *Ball*, descrita na Figura 13. Nela estão presentes campos para a velocidade de deslocamento (*vel*), a posição que o centro da bola ocupa no ambiente virtual (*pos*), a direção do movimento (*dir*), a sua cor (*color*) e o tamanho de seu raio (*raio*).

```
typedef struct _Ball{
    GLfloat vel[3];
    GLfloat pos[3];
    GLfloat dir[3];
    GLfloat color[3];
    GLfloat raio;
} Ball;
```

Figura 13: Estrutura utilizada para descrever uma bola

Os campos *vel*, *dir* e *pos* estão descritos na forma de vetores de três dimensões. Esta representação foi escolhida para facilitar o agrupamento de informações geométricas e facilitar os cálculos a serem realizados.

Além de estruturas criadas para descrever os objetos virtuais, foi preciso também criar uma estrutura para representar o jogador dentro do ambiente virtual do jogo. Neste trabalho, a informação de maior relevância para a descrição de um jogador são os pontos da borda de sua silhueta, extraídos como descrito em 4.3.4, sendo estes armazenados em vetores (arrays da linguagem C), em que cada posição representa um ponto x, y da silhueta.

O vetor de dados que identifica a localização dos jogadores pode ser usado juntamente com a estrutura que define objetos gráficos, para que se faça a detecção de colisões pela análise das posições de cada objeto (bolas e jogadores).

Para esta análise é efetuado o cálculo da distância euclidiana entre dois pontos. Sua formula é definida por:

$$d(P1P2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

onde P1 e P2 são pontos formados pelos vetores (x1, y1, z1) e (x2, y2, z2).

Na Figura 14, é exemplificada a detecção de colisão entre um jogador e uma esfera virtual. São passados para a função os pontos referentes ao jogador e à esfera. Caso a distância retornada tenha o mesmo tamanho do raio da esfera, ocorreu uma colisão.

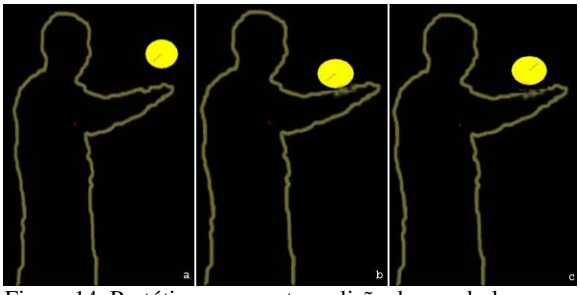

Figura 14: Protótipo que mostra colisão de uma bola com o braço de uma pessoa (silhueta) e subseqüente mudança de direção.

Em Câmera Kombat, sempre que ocorrer a colisão de um objeto com um jogador, este é penalizado. A cada penalização, uma barra de energia localizada na parte superior da tela é diminuída. Ao se esgotar esta energia, são definidos o vencedor e o perdedor da partida.

4.3.6 Custo Computacional

A taxa de captura da aplicação limita-se a 30 fps (quadros por segundo), pois está diretamente relacionada com a taxa de captura da *webcam*.

Em resolução 640 x 480 pixels, o processamento é de aproximadamente 30% de uso da CPU.

Os testes foram efetuados em uma máquina com processador Pentium Celeron 2.4 GHz com 512 MB de RAM e placa de vídeo NVidia GeForce 5200.

5. Resultados

Há um vídeo em que é possível observar o estado atual do projeto em http://tinyurl.com/nxyjx. O jogo é operacional e pode ser jogado por dois participantes.

Ao detectar socos realizados por um jogador, é disparado um golpe (bola) em direção a seu oponente. O oponente deve se desviar destes golpes para que não seja atingido, e conseqüentemente, sofrer punições. Cada punição causa uma diminuição na barra de energia (localizadas na parte superior da tela) do jogador que a sofreu. Ao ser esgotada esta barra de energia é definido o vencedor e o perdedor da disputa. Na Figura 15, é apresentada uma cena típica do estado atual do projeto, em que há uma disputa entre dois jogadores, sendo lançados golpes um contra o outro.

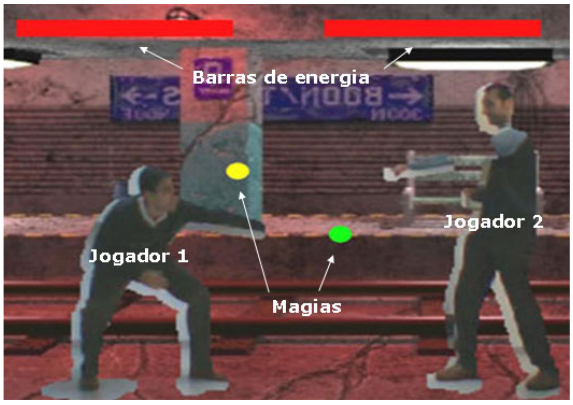

Figura 15: Câmera Kombat: disputa entre dois jogadores

6. Conclusões

Em relação ao projeto, pode-se concluir que os objetivos iniciais foram atingidos. Câmera Kombat é um jogo de luta, que utiliza técnicas da visão computacional para identificar o movimento dos jogadores, promovendo uma interação livre de dispositivos que precisam ser carregados por seus participantes. Os jogadores podem interagir entre si, lançando magias um contra o outro. A atividade física propiciada pelo jogo também é elevada, pois os jogadores devem pular ou agachar para desviar das magias que são lançados contra eles.

Durante a fase de desenvolvimento houve dificuldades para se encontrar material mais aplicado referente à biblioteca OpenCV. As principais fontes de informação são uma documentação que faz parte da distribuição da biblioteca, apenas com explicações muito sucintas das funções, e a lista de discussão oficial.

Este trabalho é mais um exemplo que é viável, em termos de disponibilidade de bibliotecas de programação adequadas e também de poder computacional facilmente acessível, construir jogos eletrônicos baseados em visão computacional. Em particular, o barateamento e a grande popularidade das *webcams*, decorrência do aumento do uso de softwares de comunicação instantânea como o Microsoft MSN Messenger, torna o cenário muito favorável ao desenvolvimento de jogos com características similares ao Camera Kombat.

6.1 Trabalhos Futuros

A seguir serão listadas algumas sugestões a serem realizadas em trabalhos futuros, que podem ser desenvolvidos a partir do código disponível em http://sourceforge.net/projects/camera-kombat/, são:

- Implementar técnicas baseadas em histograma para a determinação automática de um limiar adequado, que será usado para segmentar o fundo.
- Desenvolver novas técnicas de subtração de fundo, de modo que as torne mais adaptativas. Como sugestão, pode-se utilizar técnicas baseadas em entropia, ou através de GPUs, passando para a placa de vídeo todo o processamento.
- Tentar mesclar funções de processamento da OpenCV com as da OpenVidia para aliviar um pouco o trabalho da CPU através da transferências de algumas operações para a GPU.
- Estruturar o trabalho na forma de biblioteca, para que seja simples criar outros jogos.
- Utilização de duas *webcams*, uma para cada jogador.
- Criar uma versão em rede, do trabalho, em que duas pessoas possam jogar à distância.

Referências

BOBICK, AARON F. INTILLE, STEPHEN S. DAVIS, JAMES W. BAIRD, FREEDOM. PINHANEZ, CLAUDIO S. CAMPBELL, LEE W. IVANOV, YURI A. SCHÜTTE, ARJAN AND WILSON ANDREW, 2000. THE KIDSROOM. COMMUNICATIONS OF THE ACM, 43(3), 60-61.

EYE TOY [online]. DISPONÍVEL EM: HTTP://WWW.EYETOY.COM. [ACESSO EM: 20 NOV. 2005].

KRUEGER, MYRON W, 1991. *ARTIFICIAL REALITY II*. ADDISON-WESLEY.

KRUEGER, MYRON W. *VIDEOPLACE* [online]. DISPONÍVEL EM: HTTP://WWW.JTNIMOY.NET/ITP/NEWMEDIAHISTORY/VIDEOPLACE/. [ACESSO EM: 10 DEZ. 2006].

LANDRÉ, JÉRÔME [online]. *PROGRAMMING WITH INTEL IPP AND INTEL OPENCV UNDER GNU LINUX - A BEGINNER'S TUTORIAL*. DISPONÍVEL EM: HTTP://JLANDRE.IFRANCE.COM/RECHERCHE.HTML [ACESSO EM: 25 AGO. 2005].

MARK, KILGARD. [online]. *GLUT API*. DISPONÍVEL EM: HTTP://WWW.OPENGL.ORG/RESOURCES/LIBRARIES/GLUT/SPEC3/SPEC3.HTML. [ACESSO EM: 10 JAN. 2006].

OPENCV: OPEN SOURCE COMPUTER VISION LIBRARY. [online] DISPONÍVEL EM: HTTP://WWW.INTEL.COM/TECHNOLOGY/COMPUTING/OPENCV/INDEX.HTM. [ACESSO EM: 05 SET. 2005].

OPENGL REFERENCE MANUAL [online]. DISPONÍVEL EM HTTP://WWW.RUSH3D.COM/REFERENCE/OPENGL-BLUEBOOK-1.0/>. [ACESSO EM: OUT. 2005].

SHREINER, DAVE. NEIDER, JACKIE. WOO, MASON. DAVIS, TOM, 2006. *OPENGL PROGRAMMING GUIDE VERSION 1.4*. 4º EDIÇÃO PEARSON EDUCATION, INC. 2004.

TOYSIGHT [online]. DISPONÍVEL EM HTTP://WWW.APPLE.COM/GAMES/ARTICLES/2003/12/TOYSIGHT/. [ACESSO EM 07 MAI. 2006].

WARREN, JONAH, 2006 [online]. *UNENCUMBERED FULL BODY INTERACTION IN VIDEO GAMES*. DISPONÍVEL EM: HTTP://A.PARSONS.EDU/~JONAH/FULL_BODY/. [ACESSADO EM: 16 MAR. 2006].