UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE UM AMBIENTE VIRTUAL TRIDIMENSIONAL PARA UTILIZAÇÃO NO CÁLCULO DE IMPEDIMENTO DE JOGADORES DE FUTEBOL

DIOGO CRISTOFOLINI

BLUMENAU
2004

2004/2-12

DIOGO CRISTOFOLINI

PROTÓTIPO DE UM AMBIENTE VIRTUAL TRIDIMENSIONAL PARA UTILIZAÇÃO NO CÁLCULO DE IMPEDIMENTO DE JOGADORES DE FUTEBOL

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Paulo César Rodacki Gomes - Orientador

BLUMENAU
2004

2004/2-12

PROTÓTIPO DE UM AMBIENTE VIRTUAL TRIDIMENSIONAL PARA UTILIZAÇÃO NO CÁLCULO DE IMPEDIMENTO DE JOGADORES DE FUTEBOL

Por

DIOGO CRISTOFOLINI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo César Rodacki Gomes, Dr. – Orientador, FURB
Membro:	Prof. Jomi Fred Hubner, Dr. FURB
Membro:	Prof. Mauro Mattos, Dr. FURB

Blumenau, 03 de novembro de 2004

Dedico este trabalho aos meus pais e meus familiares por todo o amor, incentivo e apoio que me foi oferecido, ao meu orientador que acreditou na realização deste, e aos meus inseparáveis amigos(as).

AGRADECIMENTOS

À minha família, que sempre me apoiou em todos meus momentos, principalmente nos momentos difíceis.

Ao meu orientador, por acreditar e colaborar para a realização deste trabalho.

Aos meus amigos que compreenderam minha ausência em algumas festas e nas memoráveis partidas de futebol de terça feira à noite, decorrente da realização deste.

RESUMO

Este trabalho apresenta um método para calibração de câmeras a partir de pontos próprios, para a utilização no cálculo de impedimento em filmes ou imagens de partidas de futebol. A partir do momento em que a câmera está calibrada, pode-se realizar medições de distância entre dois pontos quaisquer do campo de futebol, bem como verificar a posição de impedimento de algum jogador. Com a câmera calibrada, calcula-se a posição dos objetos da cena no mundo real, possibilitando assim reconstruir a cena em um ambiente virtual. Esse ambiente virtual tridimensional oferece os recursos para visualizar a cena de qualquer posição e ângulo, movimentar-se pelo campo, medir distâncias e calcular o impedimento.

Palavras chaves: Visão computacional; Computação gráfica; Calibração câmeras.

ABSTRACT

This work presents a camera calibration method obtained from particular points, to be used in pictures soccer games off-side calculation. When the camera is calibrated, distances measurements can be realized between two distinct points and the player's off-side position can be verified. Using the calibrated camera, the objects position of real world scene is calculated and as a result a three-dimensional virtual environment is rebuilt. This virtual environment provides resources for visualizing scenes without regards with angle and position, in addition to move in the field game, to measure distances and finally to calculate the off-side position.

Key-Words: Computer vision; computer graphics; camera calibration.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CBF – Confederação Brasileira de Futebol

3D – Três dimensões

2D – Duas dimensões

GPL – General Public License

UML – Unified Modeling Language

SUMÁRIO

1 INTRODUÇÃO

É de conhecimento geral que todo ser humano está sujeito a cometer erros, sejam eles causados pelas mais diversas causas, como falta de atenção, por exemplo. Estes erros causam transtornos nas mais variadas áreas de aplicação, e no meio esportivo não é diferente.

Em todas as modalidades esportivas, uma das mais populares, e que gera o maior número de polêmicas ocasionadas por erros humanos, é o futebol. Neste esporte, a figura dos árbitros auxiliares (populares “bandeirinhas”) é muito delicada, pois em uma fração de segundos eles podem decidir contra ou a favor de determinada equipe, assinalando (ou não) um impedimento, ou uma jogada irregular, e nem sempre a marcação está correta.

Utilizando as imagens geradas pela televisão dos lances de uma partida de futebol, pode-se através de um processo matemático denominado calibração de câmeras, encontrar os parâmetros que definem uma câmera virtual equivalente à câmera que gerou a imagem, e assim reconstruir a imagem visualizando-a de um outro ponto que ofereça um melhor entendimento da cena em questão. Desta forma pode-se oferecer melhores condições para a determinação ou não da posição de impedimento.

Os objetos do mundo real são tridimensionais. Quando uma cena é capturada por uma câmera de vídeo, a câmera realiza uma transformação e uma projeção em um plano, resultando em uma imagem em 2D. O processo de calibração de câmeras consiste em encontrar um modelo matemático que possibilite reverter este processo, ou seja, a partir de uma imagem em 2 dimensões pode-se obter as coordenadas de pontos da imagem em um espaço de 3 dimensões equivalente ao original da cena real.

Após o processo de calibração da câmera, pode-se reconstruir a cena em um ambiente virtual tridimensional. A cena então pode ser analisada de vários pontos de vista, oferecendo assim uma maior interatividade, entendimento e análise da cena. Conforme Foley (1996, p. 4), “A computação gráfica interativa permite extensivamente uma interação usuário-computador em larga escala. Isso acentua significantemente a habilidade de compreender os dados, de perceber as tendências, e de visualizar objetos reais ou imaginários – de criar “mundos virtuais” que podem ser explorados de diferentes pontos de visão.”

Este trabalho é uma extensão do trabalho desenvolvido por Starosky (2003), no qual a calibração de câmeras utiliza somente 4 pontos de referência definidos na imagem, tornando

assim o resultado pouco preciso. No presente trabalho, são utilizados “n” pontos de referência na calibração da câmera, oferecendo assim maior precisão na definição dos parâmetros de câmera. Após calibrar a câmera, o usuário pode interagir indicando a posição dos jogadores e da bola na imagem, para a montagem da cena virtual tridimensional, a qual oferece os recursos de medição de distâncias, movimentação em primeira pessoa, visualização da cena de qualquer ângulo e posição e cálculo do impedimento.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal do presente trabalho é aperfeiçoar o processo de calibração de câmeras implementado por Starosky (2003) para utilizar “n” pontos de referência do campo, oferecendo assim um resultado mais preciso no cálculo do impedimento, bem como criar um ambiente virtual tridimensional para reconstrução da cena, no qual o usuário possa interagir visualizando-a de qualquer ângulo.

Os objetivos específicos deste trabalho são:

a) possibilitar ao usuário escolher vários pontos de referência do campo de futebol para aperfeiçoar o processo de calibração de câmera;
b) utilizar uma biblioteca livre para desenvolvimento de jogos para modelar a cena virtual em um ambiente tridimensional;
c) oferecer possibilidade ao usuário de caminhar pelo campo, visualizando a cena virtual de qualquer ângulo, inclusive do ponto de vista dos próprios jogadores;
d) possibilitar ao usuário tirar medidas diversas na cena virtual, como por exemplo, a distância entre a bola e o gol.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 apresenta a fundamentação teórica na qual este trabalho é embasado. Inicialmente são apresentados os trabalhos correlatos, seguindo de assuntos relativos a câmeras de vídeo, computação 3D e *game engines*.

No capítulo 3 são abordados os processos envolvidos no desenvolvimento do protótipo, tais como a definição dos principais requisitos do sistema, especificação apresentando os diagramas de casos de uso e diagrama de atividades, técnicas e ferramentas utilizadas, a operacionalidade do protótipo e uma explanação sobre os resultados obtidos.

O capítulo 4 apresenta as conclusões sobre a aplicação das técnicas propostas, bem como uma discussão a respeito de propostas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais assuntos inerentes a este trabalho, tais como: câmeras de vídeo, computação 3D e *game-engines*.

2.1 COMPUTAÇÃO 3D

Esta seção apresenta conceitos, técnicas e características que se aplicam na construção de uma cena virtual. Serão abordados os assuntos: sistema de coordenadas, projeção e objetos 3D.

2.1.1 SISTEMA DE COORDENADAS CARTESIANO 3D

Para determinar a posição de um ponto no espaço $\Re^3$, é utilizado um sistema de coordenadas baseado em três dimensões que correspondem às projeções do ponto sobre três eixos mutuamente ortogonais (ANGEL, 2003). Os eixos são chamados de X, Y, Z.

Existem duas formas de orientar este tipo de sistema de coordenadas, através da regra da “mão direita” ou através da regra da “mão esquerda”, como demonstra a Figura 1.

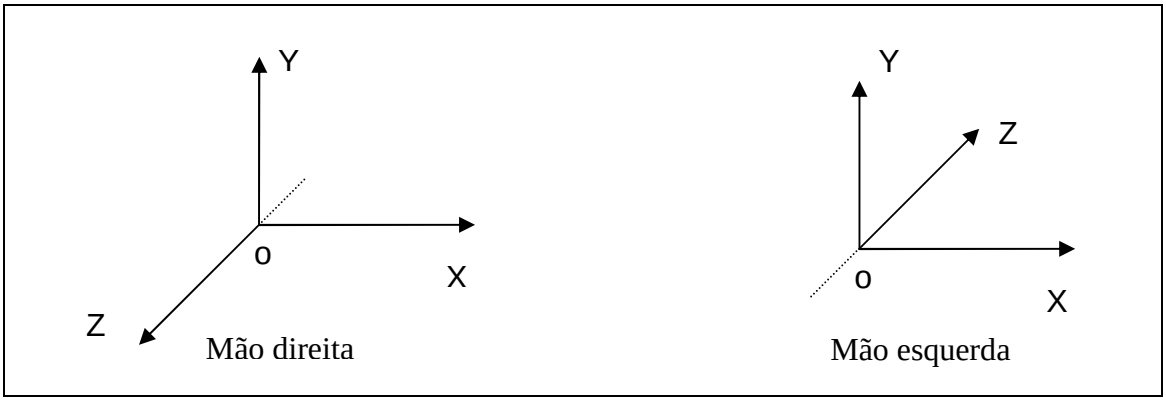


Figura 1 – Sistemas de coordenadas 3D

Neste trabalho é utilizado o sistema de coordenadas de mão esquerda.

2.1.2 PROJEÇÃO

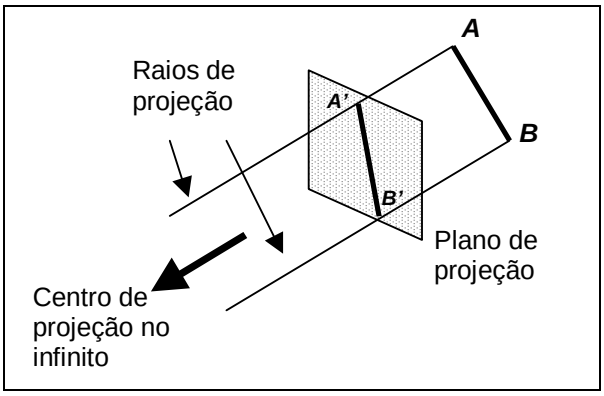
Segundo Foley (1996, p. 229) o processo de visualização em 3D é mais complexo do que o processo de visualização em 2D. Em 2D, o processo consiste simplesmente em projetar a cena em 2D na superfície do plano de projeção (tela de televisão, monitor de computador)

que também é definido em 2D. Já em 3D existe a profundidade que representa uma dimensão geométrica a mais na cena real em relação ao plano de projeção.

O problema de visualização de uma cena 3D em um dispositivo 2D, como os monitores de computador, é solucionado através de uma projeção da cena em 3D para o plano de projeção em 2D. Portanto, projeção é a técnica para transformar e exibir uma cena definida em um mundo tridimensional em uma superfície bidimensional (HILL, 2001).

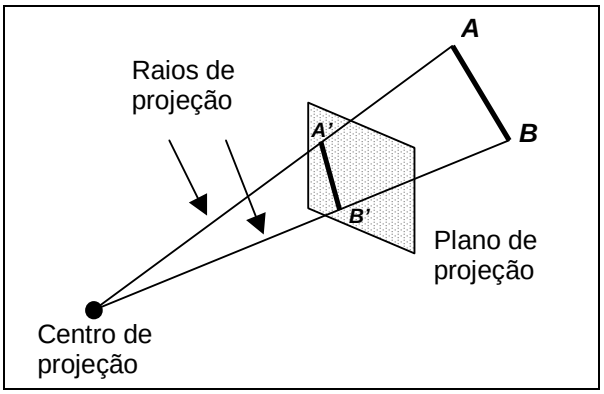
Segundo Foley (1996, p. 230), “a projeção de objetos 3D é definida por raios retos de projeção, chamados projetores”. Estes projetores são emanados do centro de projeção, e que, passando através de cada ponto do objeto 3D, intercepta o plano de projeção 2D, formando nele a imagem projetada.

Os tipos de projeção são divididos em dois grandes grupos: projeção em perspectiva e projeção paralela (ou ortogonal). A diferença entre os dois grupos é representada pela relação entre o centro de projeção e o plano de projeção. Na projeção em perspectiva essa distância é finita, e os projetores formam ângulos oblíquos entre si. Na projeção paralela, por outro lado, essa distância é infinita, assim, os projetores estão alinhados paralelamente uns com os outros. As formas de projeção em perspectiva e paralela podem ser observadas respectivamente nas Figuras 2 e 3.



Fonte: Wangenheim (2004)

Figura 2 – Projeção em perspectiva



Fonte: Wangenheim (2004)

Figura 3 – Projeção paralela

O tipo de projeção utilizado neste trabalho é a projeção em perspectiva, que é detalhada na seção seguinte.

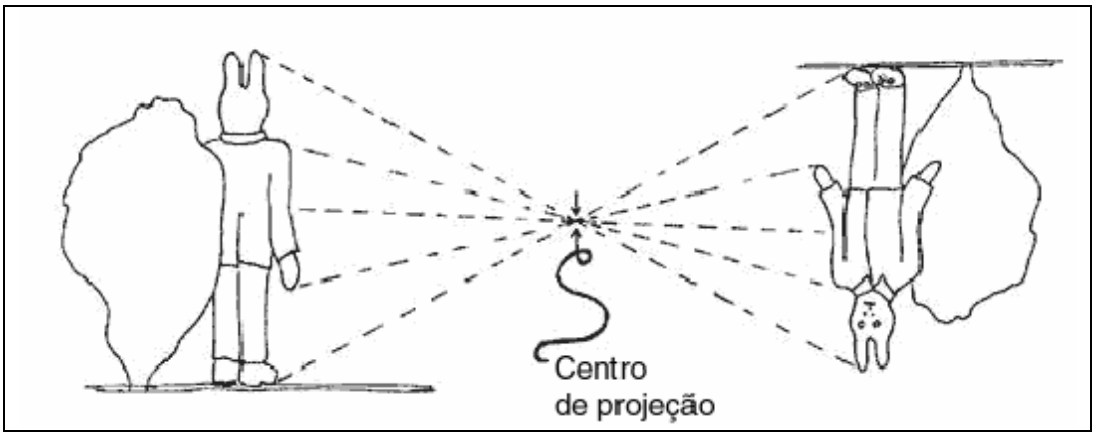
2.1.2.1 PROJEÇÃO EM PERSPECTIVA

Esta é a forma de visualização semelhante à visão humana. O volume, ou área visível, de uma cena pode ser representado por uma pirâmide sem o teto (denominada *frustum*) na horizontal, de forma que o ponto superior da pirâmide represente o centro de projeção.

Conforme Watt (1989, p. 14) a principal característica da projeção em perspectiva é a seguinte: quanto mais longe um objeto está do centro de projeção, menor ele será projetado no plano de projeção. Esta característica oferece a noção de profundidade da cena projetada em um plano de projeção 2D.

2.2 CÂMERAS DE VÍDEO

Este trabalho baseia-se no modelo de câmera de vídeo *pinhole,* que é apresentada em Szenberg (2001). O princípio de funcionamento deste modelo de câmera consiste em captar as imagens do mundo real que passam por um orifício, projetando-as invertidas em um plano, conforme demonstra a Figura 4.

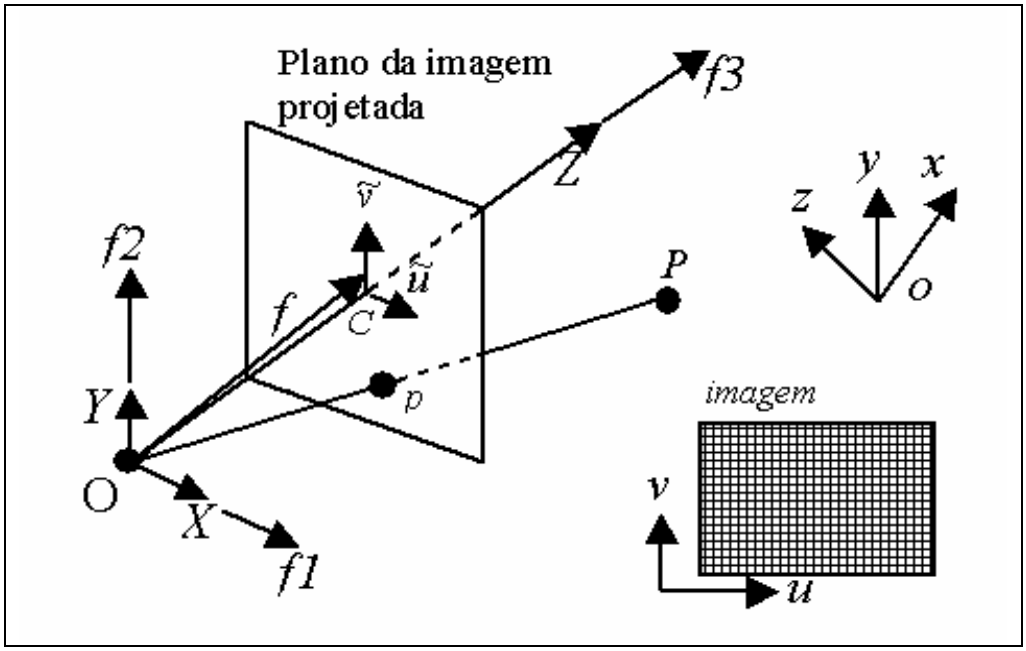


Fonte: Szenberg (2001, p. 98)

Figura 4 - Projeção através de um ponto

Definições e explicações detalhadas sobre as propriedades da câmera, centro óptico, tipos de projeção e posicionamento relativo à cena podem ser encontrados em Starosky (2003).

Para aplicações do tipo “tira-teima” é de suma importância o entendimento do modelo matemático da câmera de vídeo (modelo *pinhole* no caso), o qual pode ser visto na Figura 5.



Fonte: Szenberg (2001, p. 98)

Figura 5 - Modelo de câmera *Pinhole*

O ponto $C(\tilde{u},\tilde{v})$ da Figura 5 representa o centro de projeção da Figura 4. O sistema de coordenadas $O(X,Y,Z)$ é o sistema de coordenadas do olho humano, que está olhando para o

plano da imagem projetada (que pode ser o monitor de uma televisão ou de um computador). O sistema de coordenadas $o(x, y, z)$ é o sistema de coordenadas do mundo real, onde a cena capturada realmente ocorre. A imagem da cena desejada é apresentada em duas dimensões no plano da imagem, que fica perpendicular ao eixo óptico definido pelo centro da lente alinhado com o eixo z. Esse é o modelo matemático da câmera de vídeo utilizado neste trabalho.

A partir deste modelo, pode-se chegar a uma equação para mapear um ponto definido no mundo tridimensional para um ponto que será projetado no plano da imagem, em duas dimensões. Este assunto é tratado na próxima seção.

2.2.1 MAPEAMENTO DE UM PONTO 3D PARA UM PONTO 2D

Tendo como base o modelo de câmera definido na Figura 5, pode-se mapear um ponto $P(X, Y, Z)$ do sistema de coordenadas *OXYZ*, para o ponto $p = \tilde{u}\tilde{v}$ através da equação definida na Figura 6.

$$(\tilde{u}, \tilde{v}) = \left(f\left(\frac{X}{Z}\right), f\left(\frac{Y}{Z}\right)\right)$$

Fonte: Szenberg (2001, p. 99)

Figura 6 - Mapeamento de pontos em 3D para pontos em 2D

Com o desenvolvimento da equação descrita na Figura 6 obtém-se a equação apresentada na Figura 7, que representa a transformação projetiva planar (homografia) para mapear pontos coplanares do mundo 3D para 2D. Este desenvolvimento está descrito em Starosky (2003).

$$\begin{bmatrix} us \\ vs \\ s \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Fonte: Szenberg (2001, p.101)

Figura 7 – Transformação projetiva planar (homografia)

2.2.2 HOMOGRAFIA

Existem casos onde os pontos do mundo tridimensional que serão utilizados para calibração da câmera são todos coplanares, isto é, estão no mesmo plano. Este tipo de calibração de câmeras é chamado de Homografia.

Em seu trabalho de conclusão de curso, Starosky (2003) define conceitos e o processo básico para encontrar a homografia, utilizando 4 pares de pontos de referência, construindo um sistema linear com 9 equações e 9 incógnitas como demonstra a Figura 8, o qual é resolvido diretamente pelo método de Gauss-Jordan.

$$\begin{bmatrix} x_1 & y_1 & w_1 & 0 & 0 & 0 & 0 & 0 & 0 & -u_1 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & w_1 & 0 & 0 & 0 & -v_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_1 & y_1 & w_1 & -t_1 & 0 & 0 \\ x_2 & y_2 & w_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -u_2 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & w_2 & 0 & 0 & 0 & 0 & -v_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_2 & y_2 & w_2 & 0 & -t_2 & 0 \\ x_3 & y_3 & w_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -u_3 \\ 0 & 0 & 0 & x_3 & y_3 & w_3 & 0 & 0 & 0 & 0 & 0 & -v_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_3 & y_3 & w_3 & 0 & 0 & -t_3 \\ x_4 & y_4 & w_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_4 & y_4 & w_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_4 & y_4 & w_4 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ u_4 \\ v_4 \\ 1 \end{bmatrix}$$

Fonte: Szenberg (2001, p. 102)

Figura 8 - Sistema de equações para determinar os parâmetros da câmera com 4 pontos

A matriz descrita na Figura 8 é formada pelos pares de pontos determinados tanto no sistema de coordenadas do mundo real $o(x, y, z)$ que representam as variáveis x, y, w quanto no sistema do plano da imagem $C(\tilde{u}, \tilde{v})$ que representam as variáveis u, v, e pelo vetor h que representa os parâmetros da câmera que serão encontrados através do cálculo da homografia.

Quando o sistema possuir mais do 4 pontos de referência, existirão mais equações do que incógnitas. Conforme Szenberg (2001), isto permite que se use o método dos mínimos quadrados, aplicando equações normais ou decomposição em valores singulares, com a finalidade de minimizar o erro encontrando-se uma solução mais precisa para a homografia.

Existe outra maneira de formular o problema, baseado apenas em pontos próprios, que são pontos conhecidos do campo de futebol, como por exemplo a intersecção das linhas da que formam a grande área, ou a marca do pênalti. As coordenadas destes pontos próprios são fixas e são definidas pelas regras do jogo de futebol, que pode ser encontrada em CBF (2000). Esta maneira consiste em determinar o vetor H que satisfazem as equações apresentadas na Figura 9 e na Figura 10.

$$u_k = \frac{h_{11}x_k + h_{12}y_k + h_{13}w_k}{h_{31}x_k + h_{32}y_k + h_{33}w_k}$$

Fonte: Szenberg (2001 p. 103)
Figura 9 - Equação geral da coordenada “u” no cálculo da homografia

$$v_k = \frac{h_{21}x_k + h_{22}y_k + h_{23}w_k}{h_{31}x_k + h_{32}y_k + h_{33}w_k}$$

Fonte: Szenberg (2001 p. 103)
Figura 10 – Equação geral da coordenada “v” no cálculo da homografia

Com a aplicação das equações apresentadas pelas Figuras 9 e 10, obtemos as coordenadas p=(u,v) através da relação dos parâmetros da câmera (vetor h) e o ponto selecionado no sistema de coordenadas do mundo real (variáveis x, y,w).

Existe um problema relacionado às equações das Figuras 9 e 10 que é a não linearidade. Para resolver esta questão, pode-se expressar as mesmas equações conforme demonstra a Figura 11, onde o vetor h representa os parâmetros da câmera calculados na homografia, e as variáveis x, y, z representam o ponto selecionado no sistema de coordenadas do mundo real.

$$\begin{aligned} h_{11}x_k + h_{12}y_k + h_{13}w_k - u_k(h_{31}x_k + h_{32}y_k + h_{33}w_k) = 0 \\ h_{21}x_k + h_{22}y_k + h_{23}w_k - v_k(h_{31}x_k + h_{32}y_k + h_{33}w_k) = 0 \end{aligned}$$

Fonte: Szenberg (2001 p. 103)
Figura 11 – Equações lineares baseadas em pontos próprios

Decompondo as equações definidas na Figura 11, pode-se representá-las na forma matricial, como demonstra a Figura 12.

$$M = \begin{bmatrix} x_1 & y_1 & w_1 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1 \\ x_2 & y_2 & w_2 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 & -u_2 \\ \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} \\ x_n & y_n & w_n & 0 & 0 & 0 & -u_nx_n & -u_ny_n & -u_n \\ 0 & 0 & 0 & x_1 & y_1 & w_1 & -v_1x_1 & -v_1y_1 & -v_1 \\ 0 & 0 & 0 & x_2 & y_2 & w_2 & -v_2x_2 & -v_2y_2 & -v_2 \\ \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} \\ 0 & 0 & 0 & x_n & y_n & w_n & -v_nx_n & -v_ny_n & -v_n \end{bmatrix}$$

Fonte: Szenberg (2001 p. 103)

Figura 12 – Sistema de equações para determinar a homografia utilizando “n” pontos

Finalizando o cálculo da homografia, deve-se resolver o sistema linear multiplicando a matriz M descrita na Figura 12 pelo vetor t descrito na Figura 13, onde o vetor t representa as incógnitas do sistema.

$$t^T = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Fonte: Szenberg (2001, p. 103)

Figura 13 – Vetor das incógnitas do sistema

2.3 OBJETOS 3D

Existem várias formas de se modelar um objeto tridimensional. A forma mais utilizada é a especificação da geometria do objeto, seguido pelo mapeamento das faces e finalmente a aplicação da textura.

Para a especificação geométrica do objeto tridimensional, Watt (1989, p. 31) define que o método mais popular é o método do modelo de arame (*wireframe*). Conforme este método, um objeto é definido como sendo um conjunto de linhas que se encontram em vértices em comum.

As ligações entre as linhas do modelo de arame formam polígonos, portanto a primeira etapa da criação de objetos 3D pode ser considerada como uma modelagem geométrica, onde são aplicadas várias primitivas da geometria 3D. Um exemplo da modelagem geométrica de um objeto 3D pode ser visto na Figura 14.

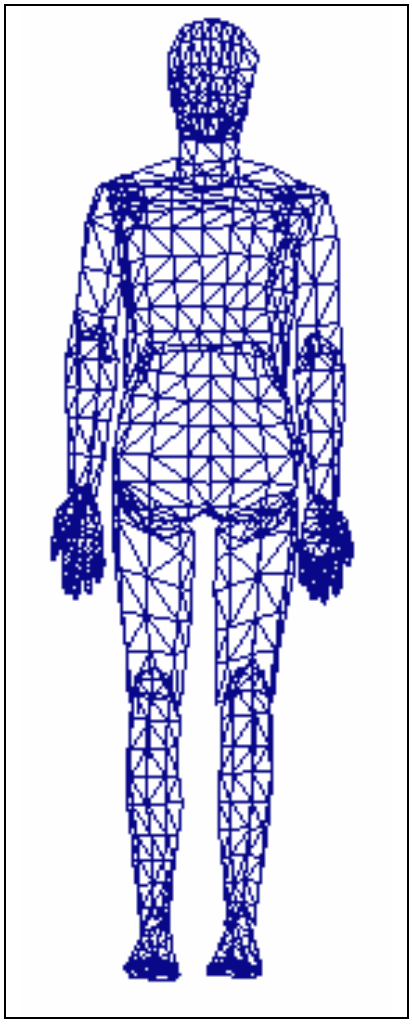


Figura 14 – Modelo de arame de um humanóide

Henkels (1997) exibe detalhadamente os conceitos aplicados sobre as primitivas que formam a base da geometria 3D utilizada para a especificação do modelo de arame, dentre elas: vértices, linhas, faces.

Depois de definido o modelo geométrico do objeto, suas faces devem ser mapeadas para a aplicação da textura. Este processo de mapeamento consiste em mapear as faces do objeto com pontos da textura, para que a textura possa ser aplicada e redesenhada independentemente do ponto de visualização.

2.4 GAME ENGINES

Com a popularização dos jogos em primeira pessoa, como *Quake* (Id Software, 1991), o mecanismo chamado de *game engine* ou motor de jogos foi ganhando destaque, e as empresas que desenvolvem os maiores jogos do mundo começaram a separar a parte de *engine* de outros componentes que são somente do jogo.

O termo *game engine* representa toda a tecnologia do núcleo de um jogo, exceto os componentes do jogo propriamente dito, tais como regras de funcionamento, módulo de inteligência artificial, detecção de colisão e movimentação de câmeras. Isso possibilita que outros desenvolvedores utilizem este núcleo para a criação de novos jogos, ou de aplicações que utilizem as características do *engine*.

Simpson (2002) define em seu artigo que os *game engines* devem ser modulares, extensivos e que permitam que programadores o utilizem para criar novos jogos com novos modelos, cenários e sons, ou que adaptem os materiais já existentes a fim de criar novos materiais.

Pode-se fazer uma analogia de *game engines* como sendo o motor de um automóvel, ou seja, o motor pode ser retirado de um automóvel e pode-se criar um novo automóvel com este motor. O mesmo conceito é aplicado aos *game engines:* como visto anteriormente, programadores podem utilizar um *engine* existente para criar novos jogos.

Um *game engine* deve oferecer uma série de funcionalidades, dentre as quais pode-se destacar: movimentação de câmeras e ângulo de visão, iluminação do ambiente, detecção de colisão e renderização da cena em tempo real, adicionar modelos na cena.

Existem vários sistemas disponíveis no mercado, a maioria é comercial, porém existem comunidades de software livre que desenvolvem *engines* sobre licenças de software livre, assim como a GPL. Para o desenvolvimento deste trabalho foi escolhida a biblioteca Irrlitch Engine, que é apresentada na próxima seção.

2.4.1 IRRLITCH ENGINE

A Irrlitch é uma biblioteca para desenvolvimento de jogos 3D, totalmente orientada a objeto, de código aberto e multiplataforma. Ela oferece todas as funcionalidades de um *game*

engine comercial, e possui uma boa performance de renderização das cenas 3D (IRRLITCH, 2004).

Todos os objetos que compõem a cena virtual são tratados como *nodes* pela Irrlitch, portanto para inserir um novo objeto, deve-se criar um novo *node* para representá-lo, definir sua posição no sistema de coordenadas, definir outras possíveis propriedades e inserir o *node* na cena virtual.

Para o processo de renderização da cena, a biblioteca oferece três modos distintos: OpenGL, Direct3D e um algoritmo próprio para renderização. O processo de renderização de uma cena consiste em efetuar a leitura e o processamento dos modelos geométricos que compõem uma cena em 3D, aplicando as texturas nos objetos e projetando-os em uma superfície em 2D, como o monitor do computador (WATT 1989, p. 97). Maiores informações sobre a Irrlitch engine podem ser encontradas em Irrlitch (2004).

2.5 TRABALHOS CORRELATOS

Este trabalho é uma extensão do trabalho iniciado por Starosky (2003), cujo objetivo era implementar um protótipo de um sistema do tipo “tira-teima” para ser aplicado no cálculo de impedimento de jogadores de futebol.

No protótipo definido por Starosky (2003), para o cálculo da homografia foram utilizados apenas 4 pares de pontos de referência, e o sistema linear resultante foi resolvido diretamente utilizando o método de Gauss-Jordan, pois ele possui 9 equações e 9 incógnitas. Com a escolha de mais pontos de referência, obtém-se um resultado mais preciso no cálculo da homografia, e conseqüentemente as medições entre dois pontos do campo de futebol também apresentarão maior precisão.

Um recurso importante não contemplado no trabalho definido por Starosky (2003) é o de remontar a cena trabalhada em um ambiente virtual. Com a montagem de um ambiente virtual, a cena pode ser visualizada de qualquer ângulo e posição, e pode-se mover a câmera de dentro da cena. Isso facilita a compreensão de alguns lances, como por exemplo no caso em que a câmera real estava mal posicionada no momento de captura da cena no jogo de futebol.

3 DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo tem por objetivo apresentar as etapas envolvidas no desenvolvimento do protótipo proposto por este trabalho. Serão abordadas as seguintes etapas: requisitos do protótipo, especificação, implementação e finalmente serão apresentados os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O sistema deve permitir a entrada de uma imagem de uma partida de futebol obtida a partir da transmissão do jogo por alguma emissora de televisão. A partir desta imagem, o usuário deve ser capaz de indicar os pontos de referência para efetuar a calibração da câmera, possibilitando assim efetuar as demais operações.

Depois de realizado o processo de calibração da câmera, o protótipo deve retornar a distância entre dois pontos quaisquer no campo de futebol, calculada no sistema de coordenadas real, inferido a partir do processo de calibração da câmera. Deve também indicar se determinado atacante estava em posição de impedimento ou não, em relação à defesa adversária, na cena representada pela imagem.

Ainda com a câmera calibrada, o sistema deve gerar uma cena virtual tridimensional baseada na cena da imagem utilizada como entrada para o processo de calibração.

O protótipo deve utilizar uma biblioteca livre para o desenvolvimento de jogos para geração e apresentação da cena virtual tridimensional.

3.2 ESPECIFICAÇÃO

Para fazer a especificação do protótipo foi utilizada a linguagem UML, descrita em Quatrani (2001). Como diagramas de especificação são apresentados os diagramas de caso de uso e o diagrama de atividades. Foi utilizada a ferramenta Rational Rose (Quatrani, 2001) para gerar a especificação do protótipo.

3.2.1 DIAGRAMA DE CASOS DE USO

A Figura 15 apresenta o diagrama de casos de uso, gerado na especificação do protótipo.

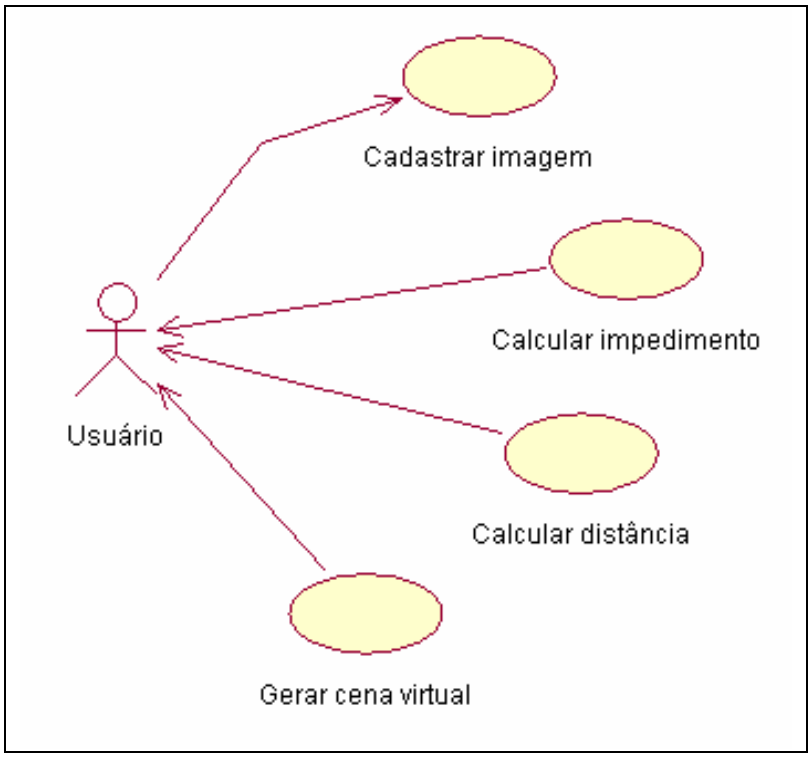


Figura 15 – Diagrama de casos de uso

Como entrada, o usuário deve cadastrar uma imagem de uma partida de futebol no sistema, calibrando assim a câmera. Após a câmera calibrada, o sistema deve gerar como retorno para o usuário a condição de impedimento de um atacante, ou a distância entre dois pontos quaisquer no campo e também gerar uma cena virtual tridimensional representando a cena da imagem cadastrada inicialmente no sistema.

3.2.2 DIAGRAMA DE ATIVIDADES

Inicialmente, o usuário deve abrir um arquivo de imagem de uma partida de futebol. A partir da carga da imagem, é necessário indicar os pontos de referência para a calibração da câmera. Para isso, deve-se clicar no botão “Iniciar” e selecionar o ponto de referência primeiramente na imagem de vídeo, e depois da imagem do campo pré-definida no protótipo. Quando todos os pontos desejáveis forem indicados, deve-se clicar no botão “Encerrar”, executando assim o processo do cálculo da homografia para encontrar a câmera que gerou esta imagem de vídeo.

A Figura 16 apresenta o diagrama de atividades, representando as atividades do protótipo, com seus respectivos fluxos de operação.

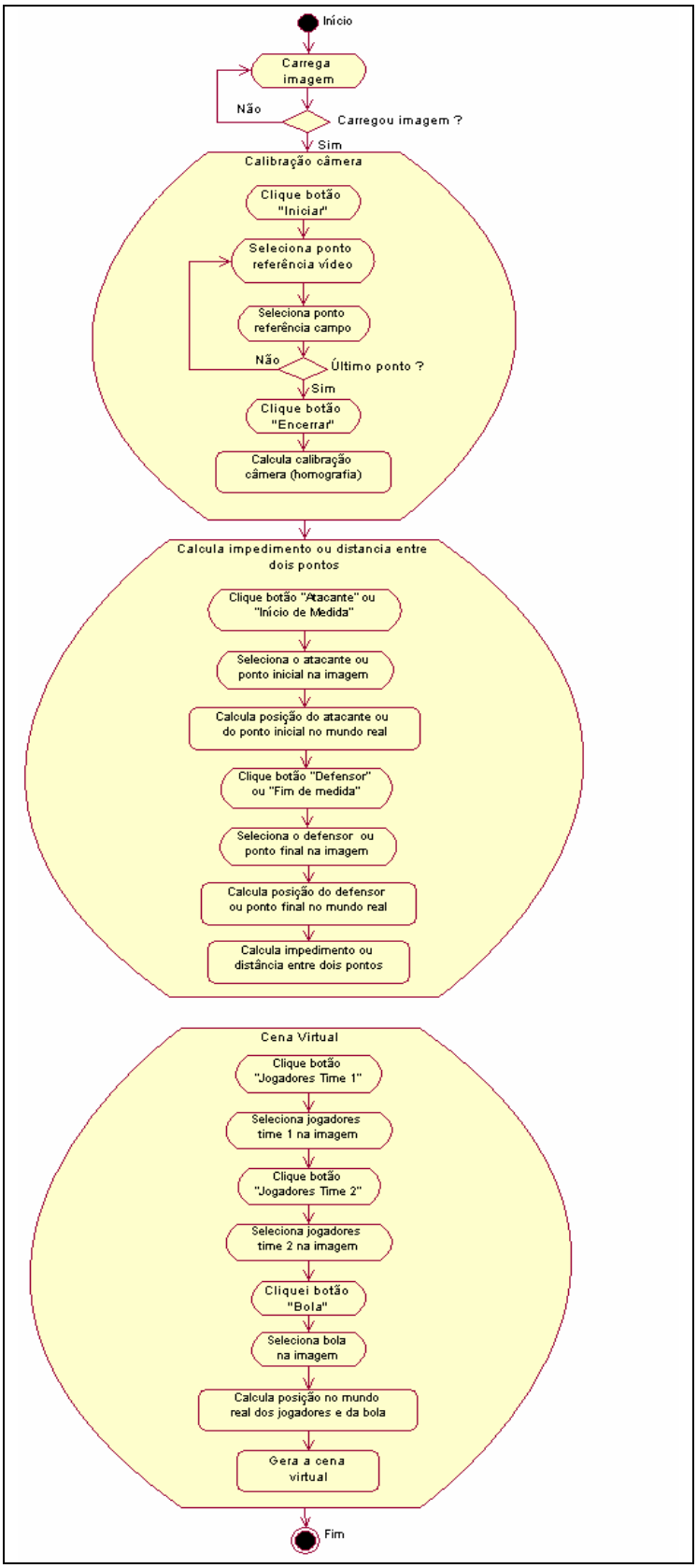


Figura 16 – Diagrama de atividades

Conforme ilustrado na Figura 16, depois de calibrada a câmera o usuário pode escolher entre três atividades: calcular o impedimento, medir a distância entre dois pontos quaisquer no campo ou gerar a cena virtual. As funções de cálculo de impedimento e medição de distâncias também estão disponíveis dentro do ambiente virtual tridimensional.

Para calcular o impedimento, deve-se clicar no botão “Atacante” e selecionar a posição do atacante na imagem do vídeo. Após selecionar o atacante, deve-se clicar no botão “Defensor” e selecionar o defensor na imagem de vídeo. O sistema apresentará como resultado uma mensagem se o atacante estava impedido ou não, bem como a distância entre os dois jogadores.

Outra atividade disponível é a medição de distâncias no campo. Para isso, deve-se clicar no botão “Início medida”, selecionar o ponto inicial na imagem de vídeo. Deve-se repetir o processo para o ponto final da medida, clicando antes no botão “Final Medida”. Como resultado, o sistema apresentará a distância entre os dois pontos selecionados.

Para gerar a cena virtual, deve-se clicar no botão “Jogadores time 1” e selecionar todos os jogadores do time cujo campo de defesa está do lado esquerdo ou abaixo na imagem. Para selecionar os jogadores do time mais a direita ou acima na imagem, deve-se clicar no botão “Jogadores time 2” e selecionar todos os jogadores que irão participar da cena virtual na imagem de vídeo. Para incluir a bola na cena virtual deve-se clicar no botão “Bola” e indicar a posição da bola na imagem de vídeo. Após selecionados os jogadores e a bola, clicando no botão “Gerar cena” o sistema abrirá uma nova janela com a cena virtual tridimensional.

3.3 IMPLEMENTAÇÃO

Esta seção apresenta considerações sobre a implementação do protótipo, como técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para o desenvolvimento do protótipo foi utilizada a linguagem C++, com o ambiente de desenvolvimento Microsoft Visual C++ 6.0 em conjunto com as bibliotecas de geração de interface IUP (TECGRAF, 2003), de manipulação e apresentação gráfica IM (TECGRAF, 2003) e CD (TECGRAF, 2003), e de *game engine* Irrlitch (IRRLITCH, 2004). Para a modelagem dos objetos da cena virtual foi utilizado o software 3D Studio MAX (DISCREET,

2004). A linguagem C++ foi escolhida por este trabalho ser uma continuação do trabalho realizado por Starosky (2003), cujo trabalho foi desenvolvido em C++ com a utilização das bibliotecas citadas para facilitar o tratamento e manipulação das imagens.

Informações sobre as bibliotecas IUP, IM e CD podem ser encontradas em TecGraf (2003), bem como informações complementares sobre a biblioteca de *game engine* utilizada podem ser encontradas em Irrlitch (2004). Informações sobre o software 3D Studio Max podem ser encontradas em Discreet (2004).

Tendo em vista que o presente trabalho é uma extensão do trabalho definido por Starosky (2003), nesta seção será dada mais ênfase para as alterações e melhorias do protótipo, principalmente as novas funcionalidades. Portanto, os processos que não sofreram alterações não serão descritos, informações sobre eles podem ser encontradas em Starosky (2003).

Toda a estrutura de dados utilizada no protótipo baseia-se em seleção de pontos. Para armazenar os dados foi criada uma estrutura composta por X e Y, tanto para seleção de pontos do plano da imagem de vídeo (números inteiros), quando para a posição dos objetos no mundo real (números com ponto flutuante). O quadro 1 apresenta as estruturas de dados utilizadas no protótipo.

```
// Estrutura para armazenar pontos com coordenadas inteiras
struct TIntPoint2D {
      int x;
      int y;
};

// Estrutura para armazenas pontos com coordenadas
// com ponto flutuante
struct TFloatPoint2D {
      float x;
      float y;
};
```

Quadro 1 – Estrutura de dados para representar os pontos do protótipo

Durante a utilização do protótipo, o usuário deve selecionar pontos na imagem de vídeo e na figura que representa o campo de futebol. Internamente, o protótipo deve identificar qual é a atividade em que o usuário está selecionando os pontos, por exemplo, ele pode estar selecionando o ponto para identificar o atacante para calcular o impedimento, ou

pode estar selecionando a bola para montar a cena virtual. Para isso foram criados contextos diferentes, que são representados por uma enumeração que é apresentada no quadro 2.

```
enum TContextoSelecao {csNone, csCampo, csAtacante, csDefensor,
csInicioMedida, csFimMedida, csJogadoresTime1, csJogadoresTime2, csBola};
```

Quadro 2 – Contextos de seleção de pontos

Os contextos são selecionados através dos eventos dos botões do protótipo. Quando o usuário clica em algum botão, o contexto atual é alimentado com o contexto que representa a atividade do botão. O quadro 3 apresenta a seleção de contexto para selecionar os jogadores do time 1 e do time 2 que irão participar da cena virtual.

```
int Sel_Campo_Time1_cb (Ihandle *self) {
      contextoSelecao = csJogadoresTime1;
      return IUP_DEFAULT;
}

int Sel_Campo_Time2_cb (Ihandle *self) {
      contextoSelecao = csJogadoresTime2;
      return IUP_DEFAULT;
}
```

Quadro 3 – Seleção de contextos

Para que a calibração da câmera possa ser calculada, deve-se informar os pares de pontos de referência, primeiramente na imagem de vídeo e depois na imagem pré-definida do campo de futebol. Como já exposto, estes pontos são armazenados nas estruturas apresentadas no quadro 1, e as estruturas são armazenadas em um vetor específico para cada contexto. A função que trata o clique no vídeo verifica qual o contexto atual, e armazena o ponto selecionado no respectivo vetor. O quadro 4 apresenta um trecho do código para a seleção de pontos de referência no vídeo

```
/* Rotina que recebe os cliques do usuário na área de vídeo/imagem */
int click_video_cb(Ihandle* iup_Canvas_Video, int button, int estado, int
u,int v) {
// .............................................................
      if (contextoSelecao == csCampo) {
            Status_Video = 1;
            /* Guarda a posição do canvas para redesenhar as referências */
            Medidas_Canvas_Video[Qde_Pontos_Selecionados].x = u;
            Medidas_Canvas_Video[Qde_Pontos_Selecionados].y = v;

            /* Apresenta as referências no vídeo */
            Repinta_Referencias_Video(0,u,v);
      }
      else if (contextoSelecao == csJogadoresTime1) {
            /* Guarda a posição do canvas para redesenhar as referências */
            Medidas_Canvas_Time1[Qde_Time1].x = u;
            Medidas_Canvas_Time1[Qde_Time1].y = v;
            Qde_Time1++;
            /* Apresenta as referências no vídeo */
            Repinta_Referencias_Video(7,u,v);
      }
      else if (contextoSelecao == csJogadoresTime2) {
            /* Guarda a posição do canvas para redesenhar as referências */
            Medidas_Canvas_Time2[Qde_Time2].x = u;
            Medidas_Canvas_Time2[Qde_Time2].y = v;
            Qde_Time2++;
            /* Apresenta as referências no vídeo */
            Repinta_Referencias_Video(8,u,v);
      }
      else if (contextoSelecao == csBola) {
            Medidas_Canvas_Bola.x = u;
            Medidas_Canvas_Bola.y = v;
            Repinta_Referencias_Video(9, u, v);
      }
  // .............................................................
}
```

Quadro 4 – Seleção de pontos de referência na imagem de vídeo

Após a seleção dos pontos de referência do campo, com um mínimo de 5 pontos é possível calcular a homografia, obtendo assim a câmera que gerou a imagem em questão. Para efetuar o cálculo da homografia, é construída a matriz apresentada na Figura 9, e é utilizado o método de Gauss para a resolução do sistema linear gerado pela multiplicação da matriz descrita na Figura 9 com o vetor de incógnitas do sistema descrito na Figura 10. O quadro 5 apresenta o código responsável por resolver o sistema linear, e o quadro 6 apresenta o código que retorna a transformação projetiva planar (homografia), ou seja, recebe os pares de pontos de referência, constrói as matrizes para formar o sistema linear, e retorna as raízes do sistema linear calculadas pela função descrita no quadro 5.

```
static int Gauss (int n,                    // [in] Numero de equacoes
                  float a[MAXDIM][MAXDIM], // [in] Matriz dos coeficientes
                  float *b)                 // [in] Vetor das constantes
                                            // [out] Vetor solucao
                                            // Valores de retorno
                                            // 0=> sem solução ou infinitas solucoes
                                            // 1=> solucao unica
{
 int   i, j, k;
 int   imax;             // Linha pivot
 float amax, rmax, temp;  // Auxiliares

 for (j=0; j<n-1; j++) {  // Loop nas colunas de a
  rmax = 0.0f;
  imax = j;
  for (i=j; i<n; i++) {   // Loop para determinar maior relacao a[i][j]/a[i][k]
   amax = 0.0f;
   for (k=j; k<n; k++)    // Loop para determinar maior elemento da linha i
   if (ABS(a[i][k]) > amax)
    amax = ABS(a[i][k]);
   if (amax < TOL)        // Verifica se os elementos da linha sao nulos
    return 0;             // Finaliza subrotina pois o sistema nao tem solucao
   else if ((ABS(a[i][j]) > rmax*amax) && (ABS(a[i][j]) >= TOL)) { // Testa relacao
da linha corrente
    rmax = ABS(a[i][j]) / amax;
    imax = i;             // Encontra linha de maior relacao - linha pivot
   }
  }
  if (ABS(a[imax][j])<TOL) {         // Verifica se o pivot e' nulo
   for (k=imax+1; k<n; k++)          // Procura linha com pivot nao nulo
    if (ABS(a[k][j]) < TOL)
     imax = k;                      // Troca linha j por linha k
   if (ABS(a[imax][j]) < TOL)
    return 0;                       // nao existe solucao unica para o sistema
  }
  if (imax != j) {                  // Troca linha j por linha imax
   for (k=j; k<n; k++)
    SWAP(a[imax][k], a[j][k]);
   SWAP(b[imax], b[j]);
  }
  for (i=j+1; i<n; i++) {           // Anula elementos abaixo da diagonal
   float aux = a[i][j] / a[j][j];
   for (k=j+1; k<n; k++)            // Transforma os demais elementos da linha
    a[i][k] -= aux * a[j][k];
   b[i] -= aux * b[j];
  }
 }
 if (ABS(a[n-1][n-1]) <= TOL)       // Verifica unicidade da solucao
  return 0;                         // Sistema sem solucao
 else {
  b[n-1] /= a[n-1][n-1];            // Calcula ultima coordenada
  for (i=n-2; i>=0; i--) {          // Inicia retro-substituicao
   for (j=i+1; j<n; j++)
    b[i] -= a[i][j] * b[j];
   b[i] /= a[i][i];
  }
 }
 return 1;                          // Finaliza a subrotina com solucao unica
}
```

Quadro 5 – Código para encontrar as raízes do sistema linear

```
int EncontraTransformacao(float *X, int n0, TFloatPoint2D *p1, TIntPoint2D
*p2)
{
 int n=n0-1;
 int i, j, k;
 float A[36][22];

 for (i=0; i<3*n+3; i++)
  for (j=0; j<9+n; j++)
   A[i][j] = 0.0f;

 for (i=0; i<n; i++) {
  A[i+2*n+2][6] = A[i+n+1][3] = A[i][0] = p1[i].x,
  A[i+2*n+2][7] = A[i+n+1][4] = A[i][1] = p1[i].y,
  A[i+2*n+2][8] = A[i+n+1][5] = A[i][2] = 1.0f,
  A[i][9+i] = -p2[i].x,
  A[i+n+1][9+i] = -p2[i].y,
  A[i+2*n+2][9+i] = -1.0f;
 }
 A[3*n+2][6] = A[2*n+1][3] = A[n][0] = p1[n].x,
 A[3*n+2][7] = A[2*n+1][4] = A[n][1] = p1[n].y,
 A[3*n+2][8] = A[2*n+1][5] = A[n][2] = 1.0f;

 for (i=0; i<9+n; i++)
  for (j=0; j<9+n; j++) {
   M[i][j] = A[0][j]*A[0][i];
   for (k=1; k<3*n+3; k++)
    M[i][j] += A[k][j]*A[k][i];
  }

 X[6] = p1[n].x,
 X[7] = p1[n].y,
 X[2] = p2[n].x,
 X[0] = X[6]*X[2],
 X[1] = X[7]*X[2],
 X[5] = p2[n].y,
 X[3] = X[6]*X[5],
 X[4] = X[7]*X[5],
 X[8] = 1.0f;
 for (i=9; i<9+n; X[i++] = 0);

 return Gauss(9+n, M, X);
}
```

Quadro 6 – Código para encontrar a homografia da câmera

Após a calibração da câmera, pode-se mapear qualquer posição da imagem de vídeo para uma posição do mundo real. O quadro 7 apresenta o código responsável por mapear um ponto da imagem de vídeo em 2D para um ponto do mundo real.

```
void Calcula_Posicao_Mundo_Real(int u, int v, TFloatPoint2D &Pos) {
      matrix15 Cam;
      double Ti[15];
      double Pw[15]; // Points of World
      // Alimenta a matriz que representa a câmera com os parâmetros da
      // câmera, encontrados na homografia
      Cam[1][1] = C[0];
      Cam[1][2] = C[1];
      Cam[1][3] = C[2];
      Cam[2][1] = C[3];
      Cam[2][2] = C[4];
      Cam[2][3] = C[5];
      Cam[3][1] = C[6];
      Cam[3][2] = C[7];
      Cam[3][3] = C[8];

      // Ajusta os pontos do canvas para pontos do vídeo.
      TIntPoint2D xPonto;
      TIntPoint2D xPontoCanvas;
      xPontoCanvas.x = u;
      xPontoCanvas.y = v;
      AjustaMedidasCanvas(xPontoCanvas, xPonto);

      Ti[1] = xPonto.x;
      Ti[2] = xPonto.y;
      Ti[3] = 1.0;

      for (i=0;i<15;i++)
            Pw[i] = 0.0;

      Gauss_Jordan(Cam,Ti,Pw,3);

      /* Calcula as posições no mundo real */

      Pos.x = Pw[1]/Pw[3];
      Pos.y = Pw[2]/Pw[3];
}
```

Quadro 7 – Função para mapear um ponto do vídeo para o mundo real

Para construir a cena virtual foi utilizada a biblioteca Irrlitch para o desenvolvimento de jogos, descrita na seção 2.4.1. Quando inicializada, a biblioteca cria uma nova janela, onde são apresentados os objetos e a cena virtual. Os objetos que participam da cena virtual podem ser jogadores de qualquer time e a bola. Não é necessário selecionar todos os jogadores que aparecem na imagem para a geração da cena virtual. Por exemplo, o usuário pode selecionar somente jogadores de um time. O quadro 8 apresenta o trecho do código responsável por inicializar a cena virtual.

```
int Gera_Cena_Virtual() {
	// inicializa a biblioteca de jogos
	TratadorEventos trataEventos;
	device = createDevice(EDT_DIRECTX8, dimension2d<s32>(800, 600), 16, false,
false, &trataEventos);

	// Seta o caption da janela
	device->setWindowCaption(L"Cena virtual gerado pelo TiraTeima");

	// Armazena ponteiros para as estruturas,
	// para não ter que ficar chamando os métodos para retornar as estruturas
	driver = device->getVideoDriver();
	smgr = device->getSceneManager();
	guienv = device->getGUIEnvironment();

	// seta a cor de fundo das mensagens para ficar mais forte
	IGUIEnvironment* env = device->getGUIEnvironment();
	SColor col = env->getSkin()->getColor(EGDC_WINDOW);
	col.setAlpha(150);
	env->getSkin()->setColor((EGUI_DEFAULT_COLOR)EGDC_3D_FACE, col);
	// fonte das mensagens
	IGUIFont* font = env->getFont("imagens/fonteMsg.bmp");
	if (font)
		env->getSkin()->setFont(font);

	// Gera a cena com a câmera em primeira pessoa
	cameraFPS = smgr->addCameraSceneNodeFPS(0, 50, 200);
	cameraFPS->setPosition(vector3df(55*10,30,37.5*10));
	 ....................................................................
```

Quadro 8 – Inicialização da cena virtual

A funcionalidade de medir a distância entre dois pontos quaisquer e calcular o impedimento dos jogadores também está disponível na cena virtual, utilizando teclas de atalho. A tabela 1 apresenta as teclas de atalho com suas respectivas funções.

Tabela 1 – Teclas de atalho para as funções disponíveis na cena virtual

Tecla de atalho	**Funcionalidade**
M	Habilita / Desabilita a mira.
S	Posiciona a câmera no ponto de vista do jogador selecionado / Volta a câmera para a visão em primeira pessoa.
I	Marca o ponto corrente para inicio de medida. O ponto corrente é onde a câmera está posicionada.
F	Marca o ponto de final de medida e apresenta uma mensagem com a distância calculada.
A	Marca o jogador selecionado como atacante para o cálculo do impedimento.
D	Marca o jogador selecionado como defensor para o cálculo do impedimento.
H	Exibe uma mensagem de ajuda apresentando as teclas de atalho disponíveis.

Para tratar os eventos de teclado e mouse, a Irrlitch disponibiliza uma função de *callback*, que é passada por parâmetro na inicialização da biblioteca. A classe responsável por

tratar os eventos deve derivar de uma classe base da Irrlitch, denominada IEventReceiver. O quadro 9 apresenta o código do tratador de eventos implementado no protótipo.

```
virtual bool OnEvent(SEvent event) {
      if (event.EventType == EET_GUI_EVENT)  {
            if ((event.GUIEvent.EventType == EGET_MESSAGEBOX_OK) ||
                (event.GUIEvent.EventType == EGET_MESSAGEBOX_CANCEL)) {
                  smgr->setActiveCamera(cameraFPS);
                  device->getCursorControl()->setVisible(false);
            }
      }
      else if (event.EventType==EET_KEY_INPUT_EVENT && event.KeyInput.PressedDown){
            switch(event.KeyInput.Key) {
                  case KEY_KEY_M: {  // controla mira
                        HabilitaDesabilitaMira();
                        return true;
                  }
                  case KEY_KEY_S: {
                        if (visaoJogador)
                              visaoJogador = false;
                        else if (nodeSelected && nodeSelected->getID() ==
                                    idJogadores) {
                              vector3df camPos = nodeSelected->getPosition();
                              camPos.Y = camPos.Y + 1;
                              cameraFPS->setPosition(vector3df((int) camPos.X,
                                                (int)camPos.Y, (int)camPos.Z));
                              // se a bola estiver na cena virtual, coloca a
                              // câmera olhando para ela
                              if (nodeBola) {
                                    camPos = nodeBola->getPosition();
                                    cameraFPS->setTarget(vector3df(
                                                (int)camPos.X, (int)camPos.Y,
                                                (int)camPos.Z));
                              }
                              visaoJogador = true;
                        }
                        return true;
                  }
                  case KEY_KEY_I: { // inicio de medida
                        posIni = cameraFPS->getPosition();
                        marcouPosIni = true;
                        return true;
                  }
                  case KEY_KEY_F: { // final de medida
                        if (marcouPosIni) {
                              marcouPosIni = false;
                              vector3df posFin = cameraFPS->getPosition();
                              // divide a distancia por 10 por causa da escala
                              MostraMensagemDistancia(sqrtl((pow(posFin.X-
                                                posIni.X,2) + pow(posFin.Z
                                                - posIni.Z,2))) / 10);
                  }
                   return true;
                  // ..........................................................
}
```

Quadro 9 – Tratador de eventos da cena virtual

Com a cena virtual já inicializada, resta somente inserir os objetos na cena. Os objetos da cena virtual são: o campo de futebol, as traves, os jogadores e a bola.

Para inserir um objeto, a Irrlitch suporta vários formatos de arquivos de modelos em 3D, incluindo modelos criados pelo software 3D Studio, o qual foi utilizado neste projeto. Inicialmente são inseridos o campo de futebol e as traves, seguido dos jogadores e a bola, conforme a seleção do usuário.

Os objetos da cena virtual possuem um controle de colisão, oferecido pela biblioteca Irrlitch. Este controle de colisão possibilita a câmera a não atravessar objetos pré-definidos. No protótipo, o campo de futebol e os jogadores que participam da cena virtual não podem ser atravessados, funciona do mesmo modo que o mundo real: se a câmera for ao encontro de algum jogador, ela sofrerá uma colisão e o jogador em questão não será atravessado.

Para inserir os jogadores e a bola, deve-se calcular a posição deles no mundo real, e depois colocar na escala para ficar proporcional dentro da cena 3D. O quadro 10 exibe o código responsável por inserir os jogadores na cena virtual.

```
void iAddJogadores(IAnimatedMesh *aMesh, ITexture* aTextura, TIntPoint2D
*aPosCanvas, int aQtd, int aIdTime, bool aRotaciona) {
      TFloatPoint2D PosMundoReal;
      IAnimatedMeshSceneNode* node;
      int rotacaoY = 0;
      if (aRotaciona)
            rotacaoY = 180;

      for (int i=0;i<aQtd;i++) {
            // Calcula o ponto no mundo real
            Calcula_Posicao_Mundo_Real(aPosCanvas[i].x, aPosCanvas[i].y,
                                       PosMundoReal);

            // Cria os jogadores no mundo virtual na posição calculada
            // A posição é multiplicada por 10 por causa da escala do campo
            node = smgr->addAnimatedMeshSceneNode(aMesh, 0, aIdTime,
                        vector3df(PosMundoReal.x*10,15,PosMundoReal.y*10),
                        vector3df(0,rotacaoY,0), vector3df(3.5f,3.5f,3.5f));
            node->setMaterialTexture(0, aTextura);

            // adiciona os jogadores no controle de colisão
            addNodeToCollisionAnimator(aMesh, node, vector3df(4,4,4));
      }
}

void AddJogadoresSelecionados() {
      // Cria um mesh para o modelo do jogador
      IAnimatedMesh* meshMan = smgr->getMesh("3d/Man.3ds");
      ITexture *texturaTime = driver->getTexture("imagens/time1.jpg");
      iAddJogadores(meshMan, texturaTime, Medidas_Canvas_Time1, Qde_Time1,
                    idJogadores, true);
      texturaTime = driver->getTexture("imagens/time2.jpg");
      iAddJogadores(meshMan, texturaTime, Medidas_Canvas_Time2, Qde_Time2,
                    idJogadores, false);
}
```

Quadro 10 – Função para inserir os jogadores na cena virtual

3.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para a validação deste trabalho, foi desenvolvido um protótipo utilizando os conceitos e técnicas apresentadas. A seguir é apresentado um estudo de caso, com a finalidade de apresentar as funcionalidades do protótipo.

A partir de uma imagem estática de uma partida de futebol, deseja-se calcular a homografia da câmera, possibilitando assim a realizar medições de distância entre dois pontos quaisquer, calcular se um jogador está em posição de impedimento ou não e reconstruir a cena em um modelo virtual tridimensional.

A Figura 17 apresenta a tela principal do protótipo.

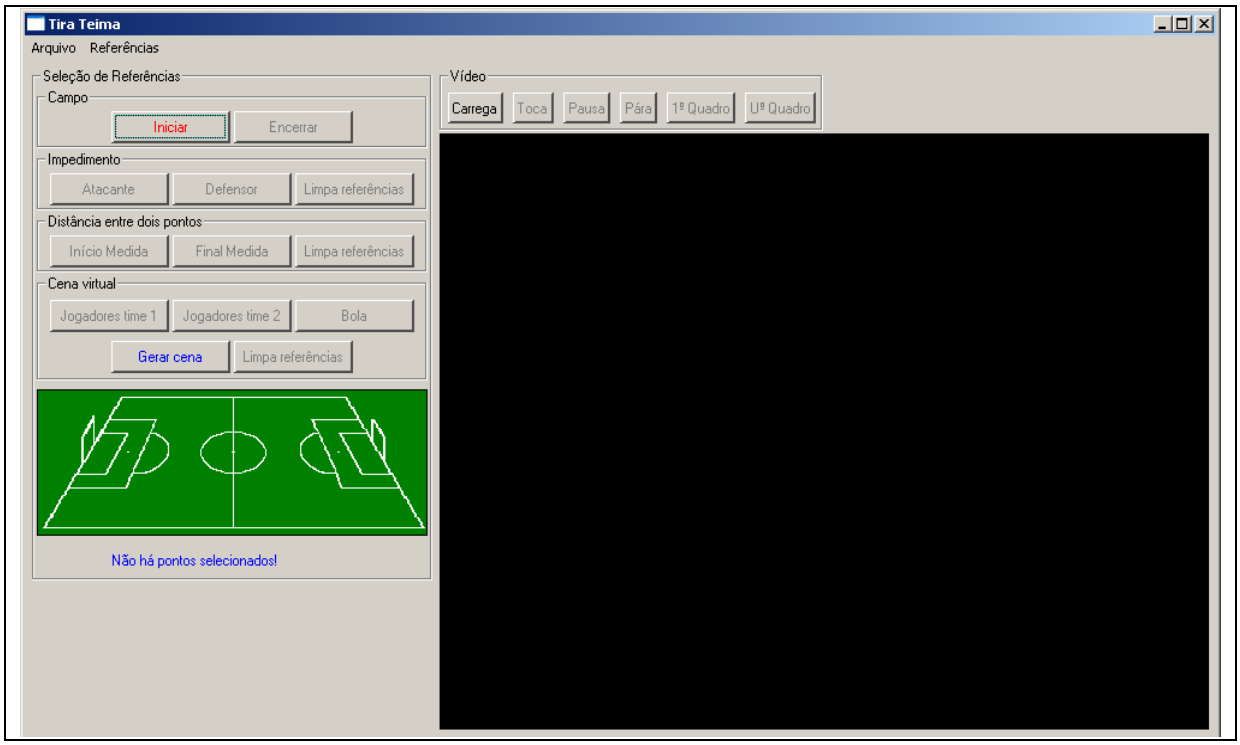


Figura 17 – Tela principal do protótipo

A seguir, o usuário deve clicar no botão “Carrega” para carregar a imagem estática de um lance de uma partida de futebol, conforme é apresentado na Figura 18.

Figura 18 – Tela principal do protótipo com a imagem carregada

Com a imagem carregada, a próxima atividade é a escolha dos pontos de referência para a calibração da câmera. A única alteração neste processo em relação ao trabalho definido por Starosky (2003) é em relação à quantidade de pontos indicados. Deve-se clicar no botão “Iniciar” para começar a marcar os pontos de referência. Quando todos os pontos desejáveis foram marcados, deve-se clicar no botão “Encerrar”. É neste momento que o protótipo vai realizar a calibração da câmera, e vai habilitar os demais botões com as outras funcionalidades do sistema. Para a seleção dos pontos, deve-se clicar nas intersecções das linhas do campo de futebol na imagem do vídeo, e após isso indicar o respectivo ponto na imagem pré-definida do campo de futebol.

A Figura 19 apresenta a tela do protótipo com os pontos de referência selecionados. Para este estudo de caso foram selecionados 6 pontos de referência para a calibração da câmera.

Figura 19 – Pontos de referência para a calibração da câmera

Após o processo de calibração da câmera, os recursos do protótipo estarão disponíveis para serem utilizados. Para calcular a posição de impedimento, deve-se clicar no botão atacante, selecionar o atacante na imagem de vídeo, depois clicar no botão defensor e selecioná-lo na imagem de vídeo. Será exibida uma mensagem confirmando ou não a posição

de impedimento, apresentando também a distância entre os dois jogadores. A Fig 20 apresenta o cálculo de impedimento da cena selecionada. Neste caso, o atacante estava em condição de impedimento, posicionado a 1,119 metros à frente do último defensor.

Figura 20 – Demonstração do cálculo do impedimento

Para realizar a medição entre dois pontos quaisquer do campo de futebol, inicialmente o usuário deve clicar no botão “Limpar referências” para limpar os pontos anteriores marcados para o atacante e defensor. Após limpar as referências, deve-se clicar no botão “Início Medida” e selecionar o ponto inicial na imagem de vídeo. Depois de marcar o ponto inicial, deve-se marcar o ponto final clicando-se no botão “Final Medida” e novamente selecionar o ponto na imagem de vídeo. A Figura 21 exibe a tela do protótipo com a distância entre dois pontos calculada, neste caso foi calculada a distância entre a marca do pênalti e a linha de fundo do campo de futebol.

Figura 21 – Demonstração do cálculo da distância entre dois pontos

Caso o usuário queira gerar a cena virtual tridimensional para representar esta imagem estática da partida de futebol, deve-se primeiramente selecionar os jogadores e a bola que irão participar da cena. Para selecionar os jogadores do time 1 (time que ataca para o lado direito do campo) deve-se clicar no botão “Jogadores time 1” e depois selecioná-los na imagem de vídeo. Deve-se repetir o processo para selecionar os jogadores do time 2, clicando no botão “Jogadores time 2” e selecionando-os na imagem de vídeo. Caso se deseje incluir a bola na cena virtual, deve-se clicar no botão “Bola” e selecionar a bola na imagem de vídeo. A Figura 22 apresenta a seleção dos jogadores e da bola para a montagem da cena virtual.

Figura 22 – Jogadores e bola selecionados para construir a cena virtual

Após selecionados os jogadores e a bola, deve-se clicar no botão “Gerar cena”. Uma nova janela se abrirá, com a cena virtual tridimensional. Utilizando as teclas direcionais do teclado o usuário pode caminhar pelo campo, e o mouse possibilita visualizar a cena de qualquer ângulo. A Figura 23 apresenta a cena virtual tridimensional gerada selecionando os jogadores e a bola de acordo com a Figura 22.

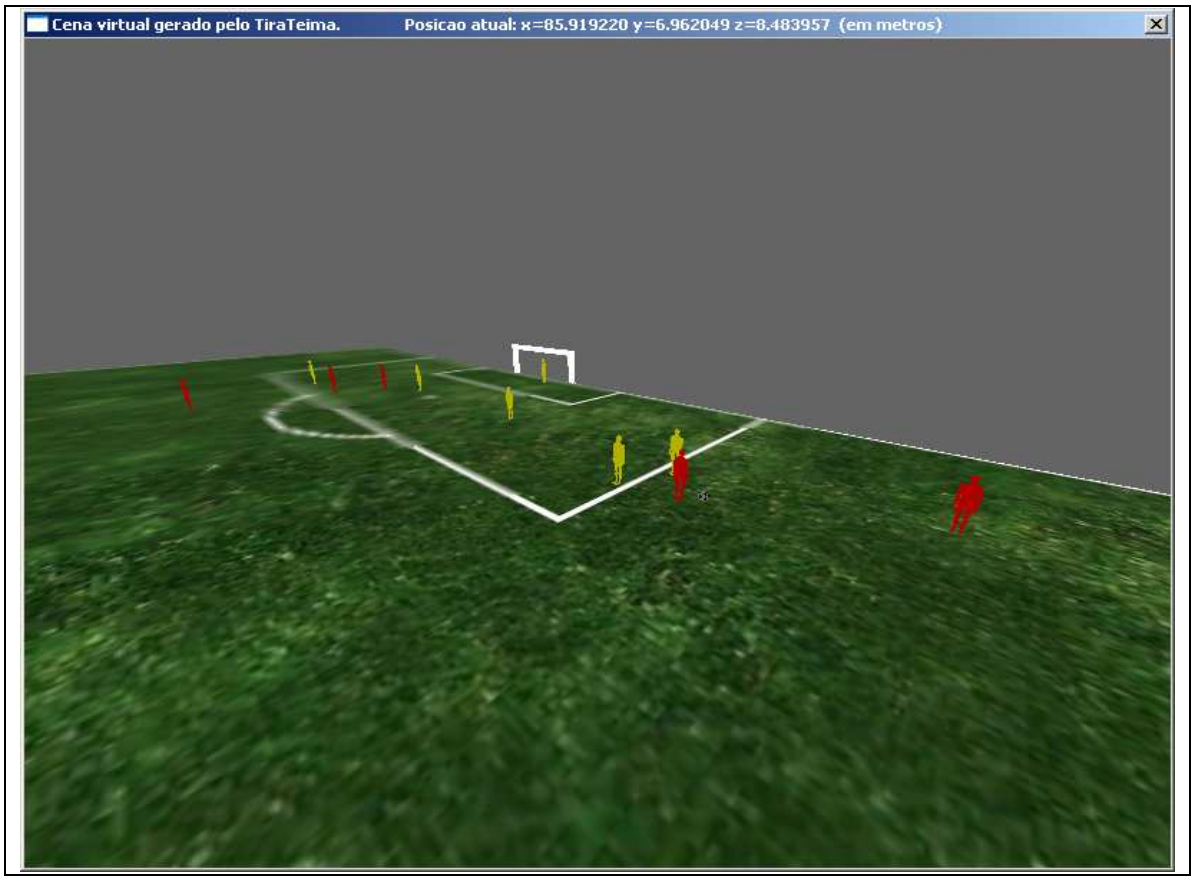


Figura 23 – Cena virtual tridimensional gerada pelo protótipo

Durante a execução da cena virtual tridimensional o protótipo oferece as funcionalidades já citadas de medição de distâncias, determinação da posição de impedimento ou não, e uma nova funcionalidade de visualizar a cena do ponto de vista de algum jogador.

Na cena virtual, as funções são disparadas por teclas de atalho. Pressionando a tecla “H” o protótipo exibe uma mensagem de ajuda informando todas as teclas de atalho e suas respectivas funções disponíveis. A Figura 24 apresenta a tela de ajuda disponível na cena virtual.

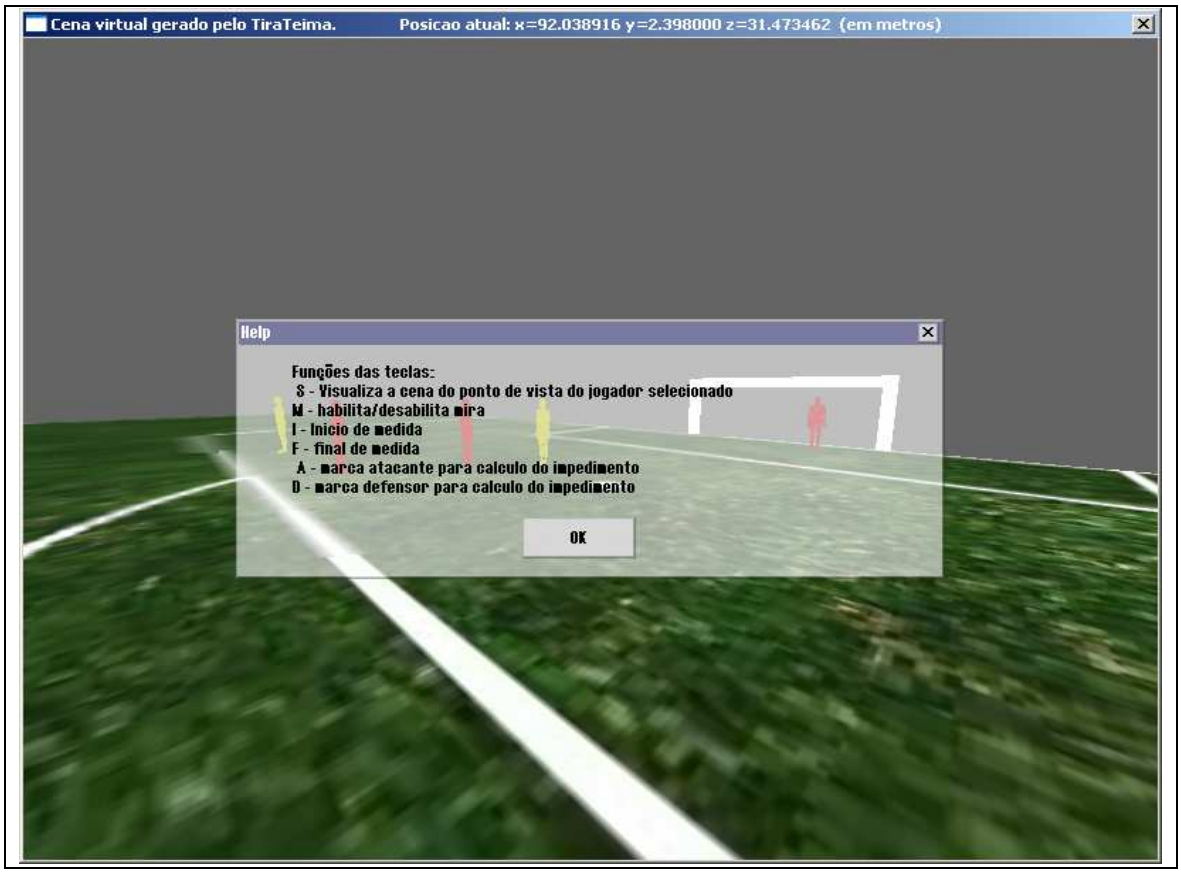


Figura 24 – Mensagem de ajuda disponível na cena virtual

Quando a função se refere a algum jogador, no caso a determinação ou não da condição de impedimento ou a visualização da cena do ponto de vista do jogador, ele deve estar selecionado para que a função seja disparada. Para selecionar um jogador, pode-se utilizar um recurso de mira disponível. A mira é acionada e escondida através da tecla “M”. Quando um jogador está selecionado, ele brilhará mais do que os outros. A Figura 25 apresenta uma imagem da cena virtual com um jogador selecionado.

Figura 25 – Jogador selecionado na cena virtual

Para visualizar a cena do ponto de vista de algum jogador, o jogador deve estar selecionado, ou seja, deve estar brilhando mais que os outros jogadores. Depois de selecionado o jogador, deve-se pressionar a tecla “S”. Automaticamente, o protótipo mudará o ponto de vista da cena, visualizando-a do ponto de vista do jogador selecionado. Para sair deste modo de visualização deve-se pressionar a tecla “S” novamente. A Figura 26 exibe a visão do goleiro sobre a cena utilizada neste estudo de caso.

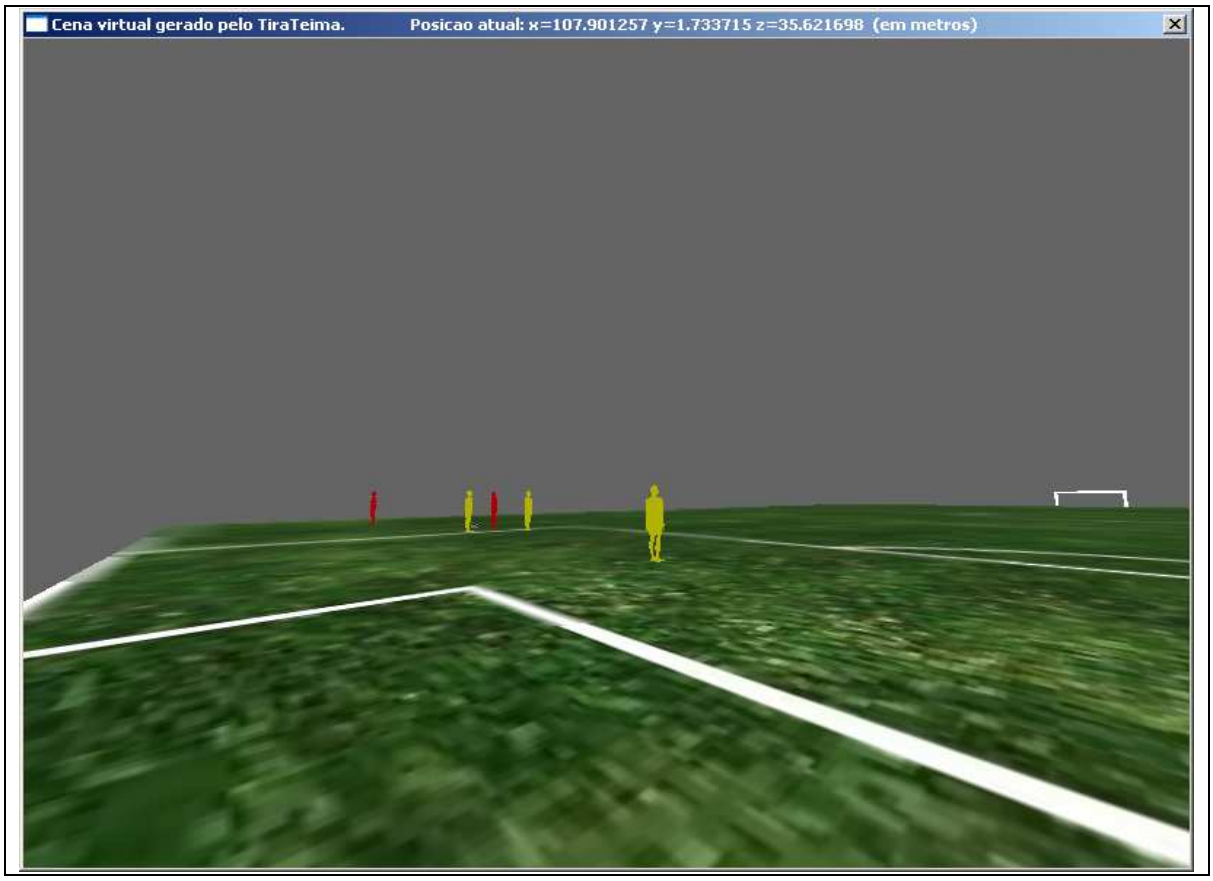


Figura 26 – Visualização da cena virtual do ponto de vista do goleiro

3.4 RESULTADOS E DISCUSSÃO

Um dos maiores problemas encontrados durante a realização deste trabalho, foi compreender e implementar os modelos matemáticos necessários para o processo de calibração de câmeras definidos na tese de Szenberg (2001), onde o processo de calibração de câmeras é dividido em duas etapas. No trabalho realizado por Starosky (2003) foi implementada a primeira parte da primeira etapa da calibração de câmeras. A continuação desta implementação foi um dos objetivos definidos para este trabalho, e como resultado toda a primeira etapa do processo de calibração de câmeras foi implementado.

Outra dificuldade encontrada foi o processo de modelagem 3D do jogador de futebol, para a montagem da cena virtual, devido à complexidade da modelagem do boneco completo com camiseta, shorts, detalhes da face dentre outros. Por essa razão, no presente trabalho foi utilizada somente a figura do sólido de um humanóide.

A bibliteca de desenvolvimento de jogos Irrlitch oferece uma boa documentação, e é de fácil utilização. É completamente orientada a objetos, com suas classes bem definidas e documentadas. Não foram encontradas maiores dificuldades quanto a utilização da biblioteca citada, e o resultado de sua utilização pôde ser considerado muito bom para o problema proposto.

A Tabela 2 apresenta um comparativo entre o protótipo original elaborado por Starosky (2003) e o tira teima aperfeiçoado, na medição de distâncias entre dois pontos. Para efeitos de comparação, foram realizadas medições cuja distância é definida pelas regras de futebol, como a distância entre a marca do pênalti e a linha de fundo, a distância da linha da grande área e a linha de fundo, e a distância da linha da pequena área com a linha de fundo.

Tabela 2 – Comparação de medição de distâncias entre dois pontos

Local comparado	**Tira teima original**	**Tira teima aperfeiçoado**	**Distância correta**	**% aperfeiçoado**
Marca pênalti	12,12 m	11,11 m	11 m	9,18
Linha grande área	17,18 m	16,17 m	16,50 m	4,12
Linha pequena área	6,6 m	5,54 m	5,50 m	19,28

Como o processo de determinação da condição de impedimento baseia-se também na distância entre dois pontos, no caso a distância entre o atacante e o defensor selecionado, seu princípio de funcionamento é o mesmo e também evoluiu consideravelmente.

4 CONCLUSÕES

O estudo realizado demonstrou a utilização de uma biblioteca de desenvolvimento de jogos para reconstruir em um ambiente virtual tridimensional um lance de uma partida de futebol a partir de uma imagem estática. Com a reconstrução da cena em um ambiente virtual, é oferecida a possibilidade do usuário caminhar dentro da cena, explorando-a de vários pontos de vista, o que proporciona um melhor entendimento do lance. Também pode-se realizar medições entre dois pontos quaisquer do campo, e avaliar a condição de impedimento de determinado atacante.

Todos os objetivos inicialmente pré-estabelecidos foram alcançados, destacando-se o aperfeiçoamento no processo de calibração de câmeras e a geração da cena virtual tridimensional representando a imagem estática do lance da partida de futebol.

Pode-se citar como potenciais problemas que se apresentam a frente à continuidade deste trabalho, a complexidade do modelo matemático da segunda etapa do processo de calibração de câmeras, que irá ajustar os parâmetros da câmera calculados na primeira etapa que foi completamente implementada.

4.1 EXTENSÕES

Como sugestões para futuros trabalhos, sugere-se que seja implementada a segunda etapa do processo de calibração de câmeras definida por Szenberg (2001), que consiste em ajustar os parâmetros encontrados na primeira etapa da homografia, com a finalidade de se obter uma solução mais exata.

Outra sugestão para trabalhos futuros seria aperfeiçoar o ambiente da cena virtual tridimensional, modelando completamente os jogadores e complementando a cena com os demais elementos de um estádio de futebol. A complementação da cena virtual com todos os seus detalhes proporcionará um ambiente mais completo e real da cena de origem, proporcionando uma melhor apresentação gráfica do sistema.

Sugere-se que em trabalhos futuros seja implementado um método de reconhecimento automático das linhas do campo de futebol e dos objetos de uma partida de futebol sobre vídeos de futebol, gerando uma calibração de câmera dinâmica, sem interação do usuário, obtendo o recurso do tira-teima em tempo real. O trabalho de Koser (2003) implementa a

primeira etapa deste processo, o reconhecimento automático das linhas do campo. A partir do levantamento automático destas linhas, pode-se calcular a operação de homografia nos moldes apresentados no presente trabalho.

Outra possível extensão do presente trabalho seria oferecer o recurso de configuração do tamanho do campo de futebol, pois sabe-se que as regras oficiais do futebol definem um tamanho mínimo e um tamanho máximo para o campo, e com isso as coordenadas dos pontos próprios utilizados no processo de calibração de câmeras sofrem alterações.

REFERÊNCIAS BIBLIOGRÁFICAS

ANGEL, Edward. **Interactive computer graphics:** a top-down approach using OpenGL. Boston: Addison Wesley, 2003.

CBF – CONFEDERAÇÃO BRASILEIRA DE FUTEBOL. **Regras oficiais de futebol.** Rio de Janeiro: Sprint, 2000.

DISCREET. **Discreet Company.** San Francisco, 2004. Disponível em: <http://www.discreet.com>. Acesso em: 05 nov. 2004.

FOLEY, James D. et al. **Computer graphics:** principles and practice. New York: Addison-Wesley, 1996.

HILL, Francis S. **Computers graphics using OpenGL.** Upper Saddle River, New Jersey: Prentice Hall, 2001.

ID SOFTWARE, **Id software.** Texas, 1991. Disponível em: <http://www.idsoftware.com>. Acesso em: 05 nov. 2004.

IRRLICHT. **Irrlicht,** Chicago, 2004. Disponível em: <http://irrlicht.sourceforge.net>. Acesso em: 05 nov. 2004.

HENKELS, Tarcísio. **Desenvolvimento de aplicações gráficas utilizando DirectDraw e Direct3D.** 1997. 79 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

KOSER, Everton Elvio. **Reconhecimento automático de linhas de campos de futebol em arquivos de vídeo para publicidade virtual.** 2003. 59 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau.

QUATRANI, Terry. **Modelagem visual com Rational Rose 2000 e UML**. Rio de Janeiro: Ciência Moderna, 2001. 206 p.

SIMPSON, Jake. **Game engine anotomy 101.** New York, 2002. Disponível em: <http://www.extremetech.com/article2/0,3973,594,00.asp>. Acesso em: 05 nov. 2004.

STAROSKY, Maiko. **Calibração de câmeras para utilização no cálculo de impedimentos de jogadores de futebol a partir de imagens.** 2003. 55 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SZENBERG, Flávio. **Acompanhamento de cenas com calibração automática de câmeras.** 2001. 170 f. Tese (Doutorado em Computação Gráfica) - Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro.

TECGraf. **Ferramentas de desenvolvimento de programas**, Rio de Janeiro, [2003?]. Disponível em: <http://www.tecgraf.puc-rio.br/f_prodp/ferr_c.htm>. Acesso em: 05 nov. 2004.

WANGENHEIM, Aldo von. **Visão computacional**, Florianópolis, 2004. Disponível em: <http://www.inf.ufsc.br/~visao>. Acesso em: 07 dez. 2004.

WATT, Alan. **Fundamentals o three-dimensional computer graphics**. New York: Addison-Wesley Publishing Company, 1989. 430 p.