AVRLib - An Object Oriented Augmented Reality Library

Douglas Coelho Braga de Oliveira, Felipe Andrade Caetano, Rodrigo Luis de Souza da Silva
Departamento de Ciência da Computação
Universidade Federal de Juiz de Fora
Juiz de Fora, Brasil
dcoelhobo@gmail.com, felipecaetano@ice.ufjf.br, rodrigoluis@ice.ufjf.br

***Resumo*—This paper presents an object oriented library named *AVRLib* (*Augmented and Virtual Reality Library*). This library is based on the computer vision algorithms presented in the ARToolkit Library with a new object oriented programming interface. The AVRLib architecture and built-in examples creates an easy interface to make either simple Augmented Reality enviroments or even new features, based on the object oriented paradigm.**

***Keywords*-Augmented Reality Library; Object-oriented programming.**

***Resumo*—Este artigo apresenta uma biblioteca orientada a objetos chamada *AVRLib* (*Augmented and Virtual Reality Library*). Esta biblioteca é baseada nos algoritmos de visão computacional presentes na biblioteca ARToolKit com uma nova interface de programação orientada a objetos. A arquitetura da AVRLib e os exemplos embutidos criam uma interface fácil para fazer qualquer aplicativo de Realidade Aumentada simples ou mesmo incluir novas características, baseadas no paradigma da orientação a objetos.**

***Keywords*-Biblioteca de Realidade Aumentada; Programação orientada a objetos.**

I. Introdução

A construção de bibliotecas de Realidade Aumentada (RA) de fácil utilização é uma necessidade para ampliar o número de aplicações deste segmento. Uma das principais bibliotecas de RA utilizadas atualmente foi apresentada na conferência SIGGRAPH em 1999. Neste evento, os pesquisadores Hirokazu Kato e Mark Billinghurst apresentaram a biblioteca ARToollkit [1] como parte do projeto "*Shared Space*". A partir daí, várias aplicações e modificações desta biblioteca foram desenvolvidas e disponibilizadas para utilização pública. Versões comerciais desta biblioteca também foram desenvolvidas pela empresa *ARToolworks*[1].

Este artigo apresenta uma biblioteca orientada a objetos (OO) baseada nas rotinas de rastreamento disponíveis no ARToolkit. Esta biblioteca tem por objetivo facilitar não só a criação de aplicações em Realidade Aumentada mas oferecer também uma interface mais intuitiva de desenvolvimento das próprias rotinas presentes na biblioteca original. Um dos problemas mais comuns encontrados nas versões públicas do ARToolkit é sua organização interna. Desta forma, a arquitetura da presente biblioteca foi reformulada para facilitar a inclusão e modificação de características específicas de seu funcionamento. Outro objetivo que deve ser destacado é o de estender as funcionalidades já existentes de modo a criar novas perspectivas no desenvolvimento de aplicações desta área.

Neste artigo os trabalhos relacionados estão sumarizados na seção II. A descrição e representação gráfica da arquitetura interna da biblioteca estão expostas na seção III. Na seção IV duas aplicações de Realidade Aumentada semelhantes são avaliadas, uma desenvolvida com o ARToolKit e a outra desenvolvida utilizando a AVRLib. A seção V apresenta alguns exemplos construídos com esta biblioteca. Por fim, a seção VI traz propostas de trabalhos futuros e a conclusão deste trabalho.

II. Trabalhos Relacionados

Inúmeras bibliotecas e *frameworks* foram propostas nos últimos anos para o desenvolvimento de aplicações em Realidade Aumentada tanto para *desktops* quanto para dispositivos móveis.

Em [2] uma arquitetura demonstrando o uso do ARToolkit com API gráficas de alto nível baseadas em grafos de cena foi proposta. Nesta arquitetura foi criada uma camada de abstração orientada a objetos para integrar o rastreamento de padrões e a sobreposição das imagens de vídeo em tempo real ao *framework*, minimizando assim a integração de diferentes tecnologias de rastreamento com a implementação de subclasses específicas. Neste trabalho as informações de profundidades dos objetos na cena foram alteradas, permitindo que objetos reais sobrepusessem objetos virtuais.

A biblioteca ARToolkit foi novamente utilizada como base na criação de uma proposta para utilização da linguagem Java na criação de aplicações em RA por Geiger et al em [3]. Neste trabalho as rotinas presentes no ARToolkit foram encapsuladas em duas classes, uma contendo as rotinas de rastreamento e a outra com as rotinas responsáveis pelo acesso às entradas de vídeo da câmera. O acesso à essas rotinas, todas escritas em C, foi possível devido à *Java Native Interface (JNI)*. Foram testados duas API's gráficas semelhantes ao OpenGL, mas específicas para a linguagem Java (GL4Java e Java3D). Segundo o autor, os resultados se apresentaram estáveis e extensões para o *framework* já haviam sido iniciadas.

[1] ARToolworks - http://www.artoolworks.com/

O SudaRA - Suporte ao Desenvolvimento de Aplicações em Realidade Aumentada - é um *framework open source* desenvolvido em C++ e baseado no ARToolKit para o desenvolvimento de aplicações de RA [4]. Este *framework* fornece suporte a utilização de modelos 3D, som, rede, entre outros, sendo o ARToolkit apenas o módulo de visão computacional do *framework*.

O trabalho encontrado que mais se aproxima do proposto neste artigo foi desenvolvido por Reimann e Paelke em [5]. Neste trabalho foi apresentada uma interface orientada a objetos construída sobre o ARToolKit. O objetivo principal deste trabalho foi criar uma camada de software orientada a objetos para facilitar o uso do ARToolkit, mas o artigo não deixa claro de qual forma esta biblioteca poderia ser utilizada por desenvolvedores que pretendem estender as funcionalidades já presentes no ARToolkit.

O principal objetivo da biblioteca proposta no presente trabalho é não só prover uma interface fácil e intuitiva de programação de aplicações de Realidade Aumentada usando o paradigma de orientação a objetos, mas também reorganizar as rotinas de visão computacional presentes no ARToolkit, tornando mais fácil o processo de estudo, expansão e aprimoramento de suas funcionalidades.

III. Arquitetura da biblioteca

A arquitetura da biblioteca proposta neste trabalho é composta por oito classes (Figura 1). A classe de controle *avrApplication*, é a principal classe da biblioteca. Grande parte dos códigos presentes nos exemplos do ARToolKit estão encapsulados na mesma. Entre estes códigos estão incluídos os de inicialização das configurações iniciais (inicialização da câmera, arquivo que define o marcador, valor de *threshold* e as *callbacks* de desenho e eventos) e o código do *loop* principal, que é executado automaticamente pela biblioteca. A classe *avrApplication* é agregada de sistemas de marcadores (herdeiras da classe abstrata *avrSystemMarker*). São estes sistemas que permitem a automação do loop principal pela biblioteca, através do uso de *polimorfismo*.

Atualmente três sistemas de marcadores compõem a biblioteca. Além dos dois sistemas já existentes no ARToolKit, *single marker* (*avrSystemSingle*) e *multi marker* (*avrSystemMulti*), existe também um sistema representado pela classe *avrSystemAutoMulti*, que estende a funcionalidade de múltiplos marcadores do Artoolkit, possibilitando que vários marcadores sejam combinados em uma única projeção, de forma redundante e sem a necessidade de que as posições dos marcadores sejam conhecidas. Estes sistemas gerenciam os marcadores adicionados à aplicação permitindo a verificação da visibilidade dos mesmos, realizam a transformação da câmera no espaço do marcador, chamam a *callback* de renderização, entre outras operações específicas de cada sistema. Estas classes possuem uma lista de objetos da classe *avrPattern* (marcadores que agregam o sistema) e um objeto da classe *avrMatrix3x4* (matriz de transformação).

A classe *avrPattern* armazena todas as informações de um determinado marcador da aplicação, como o identificador, nome, coordenadas do centro, tamanho, matriz de transformação, acurácia e visibilidade.

A classe *avrMatrix3x4* é uma especificação da classe *avrMatrix* que são matrizes de dimensão 3 x 4 e n x m, respectivamente. O objetivo de tais classes é facilitar as operações entre matrizes, muito exigidas em aplicações RA, a partir de uma interface OO intuitiva. A classe *avrMatrix3x4*, em específico, representa as matrizes de transformação de cada marcador e de cada sistema de marcadores. A referida classe possui não só operações diretas entre matrizes como também métodos auxiliares, providos do *ARToolKit*, que são frequentemente utilizados na renderização dos objetos virtuais na cena real. Um exemplo é o método *getMatrixGLFormat()* que retorna a matriz de transformação no formato usado pelo OpenGL.

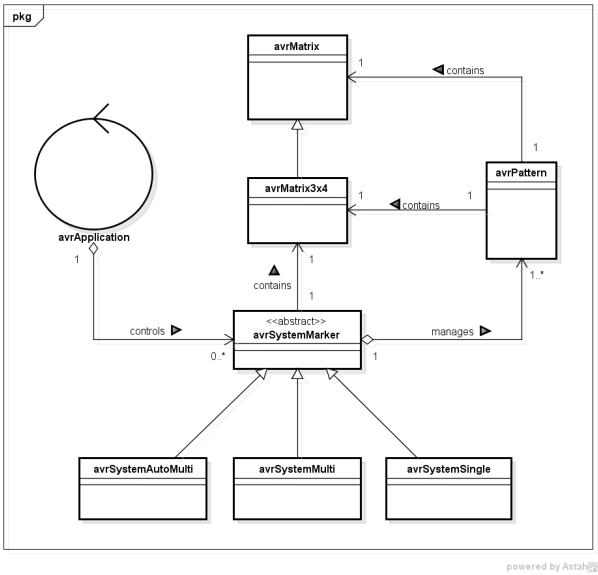


Figura 1. Diagrama de classes da AVRLib

IV. Utilização da biblioteca

Nesta seção serão apresentados dois exemplos de aplicações RA semelhantes, uma utilizando a biblioteca ARToolKit e outra utilizando a biblioteca AVRLib. Primeiro será mostrada a aplicação *SimpleTest*, um dos exemplos presentes no ARToolKit. Em seguida será apresentada a aplicação *SingleObject* que faz parte dos exemplos da biblioteca proposta neste trabalho. Ambas aplicações renderizam um cubo virtual sobre um marcador fiducial previamente conhecido.

A. Simple Test

Neste exemplo existem as rotinas *init*, *mainLoop*, *keyEvent*, *cleanup*, *draw* e a função *main*. Será apresentado aqui apenas os códigos destas duas últimas, além das variáveis

globais necessárias e uma breve descrição das duas primeiras em forma de comentários.

- *static void init()*: Responsável por configurar e inicializar a câmera além de carregar o padrão do marcador usado na aplicação;
- *static void mainLoop()*: *Loop* principal da aplicação;
- *static void keyEvent(unsigned char key, int x, int y)*: *Callback* de eventos de teclado;
- *static void cleanup()*: Responsável por parar a captura de vídeo e limpar as renderizações.

```
#include <...>

ARParam  cparam;
char     *vconf
= "../media/WDM_camera_AVRLib.xml";
char     *cparam_name
= "../media/camera_para.dat";
char     *patt_name      = "../media/patt.hiro";
int      xsize, ysize;
int      thresh = 100;
int      count = 0;

int      patt_id;
double   patt_width     = 80.0;
double   patt_center[2] = {0.0, 0.0};
double   patt_trans[3][4];

/* main loop */
static void mainLoop(void)
{
   // Esta função possui cerca de 30 linhas
   // e dentre as inúmeras operações que
   // realiza, destaca-se o tratamento do
   // video, busca por marcadores na cena
   // transformações entre os espaços da
   // câmera e do marcador e manipulação de
   // buffers
}

static void init( void )
{
   // Esta função possui cerca de 20 linhas
   // Basicamente a função abre o contexto
   // de video, inicializa os parâmetros de
   // câmera e abre a janela de renderização
}

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

   arVideoCapStart();

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

void draw()
{
   double    gl_para[16];

   argDrawMode3D();
   argDraw3dCamera( 0, 0 );

   glClearDepth( 1.0 );
   glClear(GL_DEPTH_BUFFER_BIT);
   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LEQUAL);

   argConvGlpara(patt_trans, gl_para);
   glMatrixMode(GL_MODELVIEW);
   glLoadMatrixd( gl_para );

   glTranslatef( 0.0, 0.0, 25.0 );
   glutSolidCube(50.0);

   glDisable( GL_DEPTH_TEST );
}
```

B. Single Object

Com este exemplo será possível perceber como o desenvolvimento de aplicações RA com a AVRLib é simplificado. Apenas três das seis rotinas presentes no *SimpleTest* são necessárias neste exemplo. A *keyEvent*, que possui o mesmo papel da anterior, é uma delas. Portanto será apresentado aqui apenas a função *main* e a *callback* de renderização *draw*.

```
int main(int argc, char **argv)
{
   singleObjApp = new avrApplication();

   char* config = "Data/WDM_camera_AVRLib.xml";
   char* param = "Data/camera_para.dat";
   singleObjApp->setCameraFiles(config, param);

   char* pattName = "Data/avr.patt";
   singleObjApp->addPattern(pattName, 50.0,
                            NULL, draw);

   singleObjApp->setKeyCallback(keyEvent);
   singleObjApp->setThreshold(100);

   singleObjApp->start();
   return 0;
}

static void draw()
{
   glTranslatef( 0.0, 0.0, 25.0 );
   glutSolidCube(50.0);
}
```

Estes códigos demonstram a simplicidade de desenvolvimento com a AVRLib. Esta simplicidade vem do fato da biblioteca realizar grande parte dos códigos automaticamente, o que diminui não só o número de rotinas a serem escritas como também o número de linhas de código em cada função desenvolvida. A *callback draw* é uma demonstração disto. Toda parte de transformação de matrizes, troca entre os contextos de renderização, troca e limpeza de buffers, é automática.

Além disto, os exemplos do ARToolKit apresentam várias variáveis globais. Neste exemplo, entretanto, a única variável global é a instância da classe *avrApplication* para uso na keyEvent.

Por fim, a inclusão de bibliotecas é outro ponto simplificado. Enquanto que aplicações do ARToolKit exigem a

inclusão de muitas bibliotecas, aplicações da AVRLib impõem apenas que o cabeçalho *avrApplication.h* seja incluso, uma vez que todas as dependências já são resolvidas dentro da própria AVRLib.

V. Exemplos disponíveis na biblioteca

Muitas aplicações de RA simples estão contidas na biblioteca AVRLib na forma de exemplos. Boa parte destas aplicações foram importadas da biblioteca ARToolKit e foram, então, reestruturadas e remodeladas de forma a deixar os exemplos mais intuitivos do que os originais. Nesta seção, todos estes exemplos serão descritos e ilustrados.

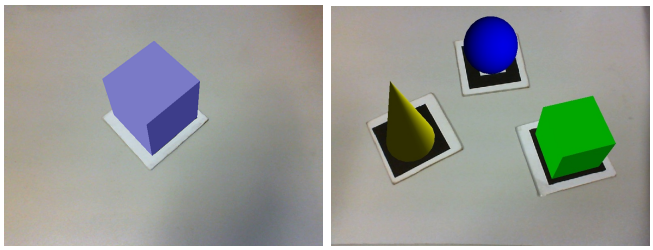
Todos os exemplos têm como entrada os arquivos de configuração e parâmetros da câmera, os arquivos que definem os marcadores utilizados, o tamanho (em milímetros) e as coordenadas do centro de cada marcador, as *callbacks* de renderização e eventos de teclado e o valor de *threshold* da aplicação.

Considere nestes exemplos Mr como sendo a matriz de transformação que se busca, Mp a matriz de transformação do sistema *avrSystemMulti* e Mq a matriz de transformação do sistema *avrSystemSingle*.

A. SingleObject

Este exemplo foi apresentado na seção IV. Seu objetivo é apresentar os passos iniciais de desenvolvimento com a AVRLib. Sua renderização é de um simples cubo virtual (Figura 2.a).

- *Marcador utilizado*: avr



(a) (b)

Figura 2. Aplicação SingleObject em (a) e Aplicação MultiObject em (b)

B. MultiObject

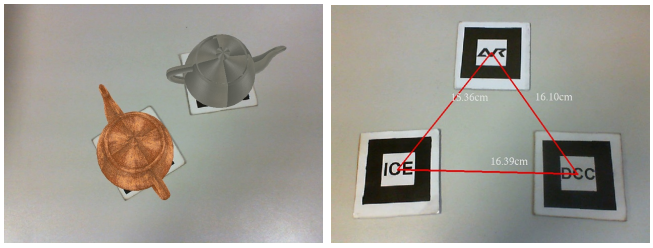
A aplicação *MultiObject* é similar à anterior, porém múltiplos objetos são renderizados em vez de apenas um (Figura 2.b). Seu objetivo é mostrar como renderizar variados objetos sobre diferentes marcadores sem relações entre si.

- *Marcadores utilizados*: avr, dcc e ice

C. TextureObject

Neste exemplo os objetos renderizados apresentam texturas (Figura 3.a).

- *Entrada específica*: Imagens para a textura
- *Marcadores utilizados*: avr e dcc



(a) (b)

Figura 3. Aplicação TextureObject em (a) e Aplicação MarkersDistance em (b).

D. MarkersDistance

Esta aplicação renderiza a distância entre os marcadores da cena (Figura 3.b). Sejam Ma e Mb marcadores visíveis na cena, a distância entre eles é calculada pelo módulo do vetor $\vec{MaMb}$. Eventos de teclado controlam o brilho das arestas e do texto.

- *Marcadores utilizados*: avr, dcc e ice

E. MarkerCameraDistance

Semelhante à aplicação anterior, mas neste caso a distância calculada e renderizada é em relação à câmera real (Figura 4). A distância entre o marcador da cena e a câmera real é obtida através da equação 1. As coordenadas do marcador também são renderizadas no vídeo.

- *Marcadores utilizados*: avr

$$dist = \sqrt{\sum_{n=1}^{3} (Mq_{n,4})^2} \tag{1}$$

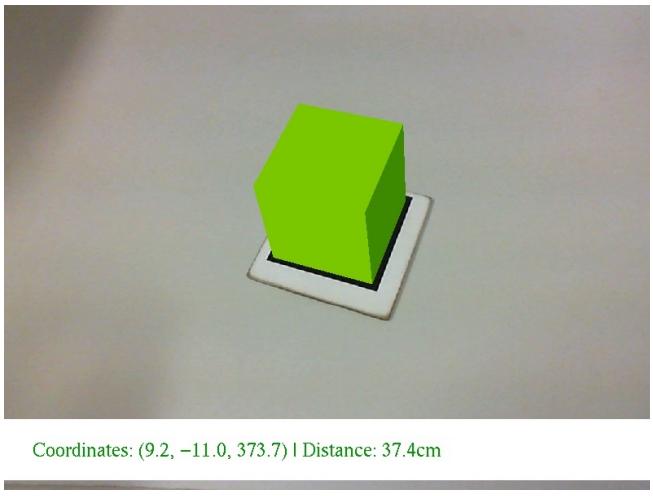
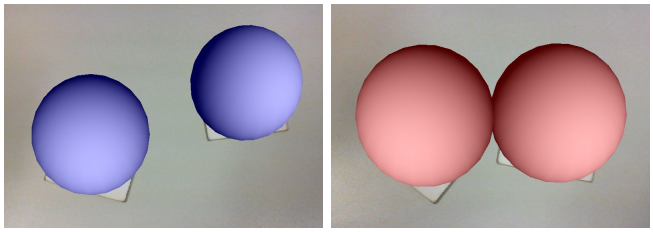


Figura 4. Aplicação MarkerCameraDistance

F. Collision

Simula uma colisão entre duas esferas virtuais. No estado de colisão as esferas mudam de cor (Figura 5). O teste de colisão é feito verificando se a distância entre os marcadores da cena é menor ou igual a soma do raio da esfera A com o raio da esfera B.

- *Marcadores utilizados*: avr e dcc



(a) (b)

Figura 5. Aplicação Collision. Em (a) Não há colisão e em (b) as esferas colidem.

G. CameraAxisView

Esta aplicação é um exemplo um pouco mais complexo em relação aos anteriores. O objetivo deste exemplo é simular a posição e pose da câmera real em relação ao marcador da cena. Para isto é renderizado um ambiente virtual em uma saída de vídeo secundária. Neste ambiente tem-se um objeto para representação da câmera real e um cubo, que representa o marcador. Ao movimentar a câmera real, o objeto que a representa virtualmente se move na mesma direção e sentido. A posição e pose da câmera virtual é estimada pela inversa da matriz de transformação do marcador.

Eventos de *mouse* e teclado controlam o ambiente virtual como o ângulo de visão e *zoom*. Ainda é mostrada a imagem limiarizada numa terceira saída de vídeo. No vídeo principal, o marcador projeta os eixos coordenados no espaço 3D. Informações presentes na matriz de transformação, tais como as rotações, também são apresentadas no vídeo. A Figura 6 mostra um quadro do vídeo desta aplicação. O código deste exemplo, em relação ao exemplo análogo do ARToolkit, foi bastante simplificado.

- *Entradas específicas*: *callbacks* de eventos de *mouse* e movimento
- *Marcadores utilizados*: avr e dcc

H. MultiMarker

A partir desta aplicação começam os exemplos utilizando o sistema de múltiplos marcadores com relação entre si previamente calculadas (*avrSystemMulti*). Neste exemplo, um cubo virtual é projetado sobre cada marcador que agrega o sistema. Quando um determinado marcador não está visível na cena, o cubo sobre ele continua sendo renderizado desde que ao menos um marcador do sistema esteja visível. Para

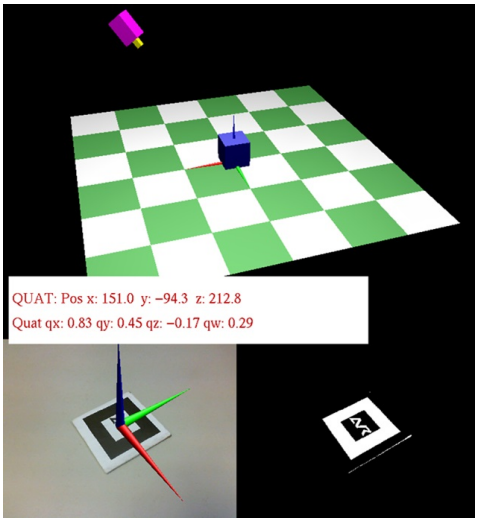


Figura 6. Aplicação CameraAxisView

representar os marcadores não-visíveis, os cubos projetados sobre eles mudam de cor (Figura 7).

- *Marcadores utilizados*: marcadores numéricos de 1 à 9

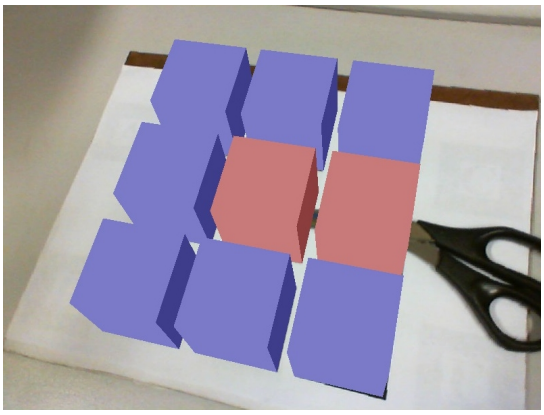


Figura 7. Aplicação MultiMarker

I. PaddleInteraction

Neste exemplo há uma integração entre dois sistemas, *avrSystemSingle* e *avrSystemMulti*. Cubos, em posições aleatórias, são projetados pelos marcadores que agregam o sistema *multi*. Sobre o marcador do sistema *single* é projetada uma espécie de "raquete". Quando o centro desta raquete intercepta uma faixa central de um dos cubos, este cubo interceptado muda sua forma (Figura 8).

Para verificar se a raquete está interceptando algum cubo, é calculada uma matriz de transformação que leve o espaço do sistema *multi* ao referencial do marcador *single*. A equação 2 descreve este cálculo.

$$Mr = Mp^{-1} \times Mq \quad (2)$$

Tendo a matriz Mr sido calculada, basta calcular a distância entre esta matriz e o centro dos cubos ou, dentro do contexto, calcular a distância da raquete em relação aos cubos. Se esta distância for menor ou igual a um valor pré-definido o cudo é dado como interceptado.

- *Marcadores utilizados*: avr e marcadores numéricos de 1 à 9

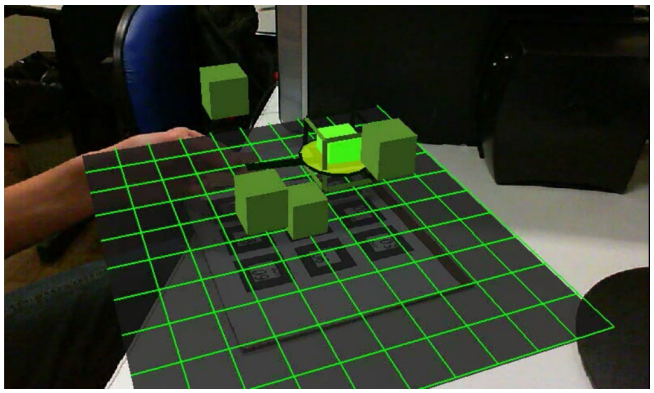


Figura 8. Aplicação PaddleInteraction

J. PaddleDemo

Esta aplicação também integra os sistemas *avrSystemSingle* e *avrSystemMulti*. Nesta, porém, são realizados mais eventos entre os sistemas. Além de checar a interseção entre os objetos projetados pelo sistema *multi* e o marcador *single*, é realizado também um teste de inclinação.

A renderização simula uma mesa de *ping-pong*, as "bolinhas" e a raquete (Figura 9). Quando uma bolinha está sobre a raquete, se esta última se inclina em relação ao plano da mesa, a bolinha rola sobre a raquete até cair sobre a mesa. É possível pegar novamente a bolinha que está sobre a mesa realizando um teste semelhante ao de interseção feito no exemplo anterior.

Para verificar a inclinação da raquete em relação ao plano da mesa é calculada uma matriz de transformação que leva o espaço do marcador *single* ao referencial do sistema *multi*. A equação 2 vista anteriormente descreve este cálculo.

Calculada Mr, o ângulo de inclinação é calculado. Se este ângulo for superior a um valor pré-definido é dado como marcador inclinado e portanto a bolinha se move proporcionalmente ao valor do ângulo e na direção da inclinação.

- *Marcadores utilizados*: avr e marcadores numéricos de 1 à 9

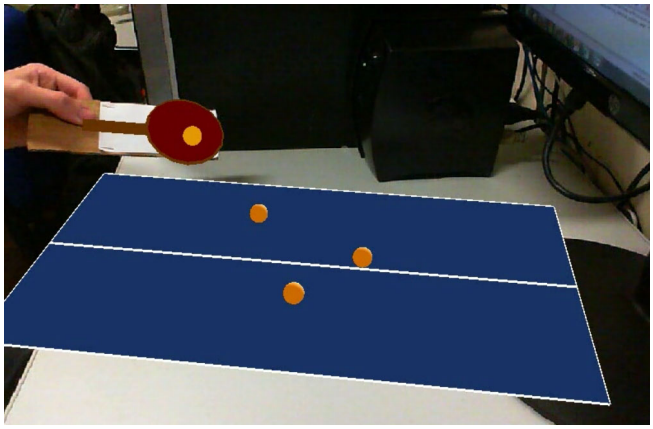


Figura 9. Aplicação PaddleDemo

VI. Conclusão e Trabalhos futuros

Este artigo apresentou uma biblioteca orientada a objetos (OO) baseada nas rotinas de rastreamento disponíveis no ARToolkit com o objetivo de facilitar a criação de aplicações de Realidade Aumentada bem como o desenvolvimento e aprimoramento das rotinas presentes na biblioteca original. A arquitetura da presente biblioteca foi projetada para facilitar a inclusão e modificação de características específicas de seu funcionamento pois os módulos da biblioteca foram agrupadas por funcionalidade.

A biblioteca foi inicialmente projetada para ser compilada na plataforma *Windows*, contudo, uma versão para sistemas baseados em *Linux* está em desenvolvimento, como parte do objetivo de portabilidade.

Uma proposta de trabalho futuro é adicionar à biblioteca a possibilidade de criar novos sistemas de múltiplos marcadores sem a necessidade de uma etapa de pré-processamento dos mesmos.

Referências

[1] H. Kato and M. Billinghurst, "Artoolkit documentation," 2002. http://www.hitl.washington.edu/artoolkit/documentation. Acessado em 06-Setembro-2013.

[2] M. Haller, W. Hartmann, T. Luckeneder, and J. Zauner, "Combining artoolkit with scene graph libraries," in *Augmented Reality Toolkit, The First IEEE International Workshop*, pp. 2 pp.–, 2002.

[3] C. Geiger, C. Reimann, J. Sticklein, and V. Paelke, "Jartoolkit - a java binding for artoolkit," in *Augmented Reality Toolkit, The First IEEE International Workshop*, pp. 5 pp.–, 2002.

[4] C. H. Cunha and S. M. Fernandes, "Prototipação de ambientes físicos com realidade aumentada," *Symposium on Virtual and Augmented Reality*, p. 4, 2010.

[5] C. Reimann, S. Engel, and V. Paelke, "Object-oriented toolkit for augmented reality," in *Augmented Reality Toolkit Workshop, 2003. IEEE International*, pp. 32–34, 2003.