

ARTOOLKIT – ASPECTOS TÉCNICOS E APLICAÇÕES EDUCACIONAIS

VERSÃO ELETRÔNICA DO CAPÍTULO 6 DO LIVRO:

Cardoso, A.; Lamounier Jr, E. editores - **Realidade Virtual: Uma Abordagem Prática**. Livro dos Minicursos do SVR2004, SBC, São Paulo, 2004.

REFERÊNCIA DA VERSÃO IMPRESSA DO CAPÍTULO 6:

Consularo, L.A.; Calonego Jr, N.; Dainese, C.A.; Garbin, T. R.; Kirner, C.; Trindade, J.; Fiolhais, C. - **ARToolKit: Aspectos Técnicos e Aplicações Educacionais**. In: Cardoso, A.; Lamounier Jr, E. editores. **Realidade Virtual: Uma Abordagem Prática**. Livro dos Minicursos do SVR2004, SBC, São Paulo, 2004, p. 141-183.

SUMÁRIO DO CAPÍTULO 6

6	ARTOOLKIT – ASPECTOS TÉCNICOS E APLICAÇÕES EDUCACIONAIS.....	3
6.1	Introdução	3
6.2	Versões e Instalação.....	5
6.2.1	Instalação no PC-Windows	6
6.2.2	Programação de Aplicações	8
6.2.3	Hardware necessário	9
6.2.4	A saída do simple.exe	9
6.2.5	A saída do simpleVRML.exe	11
6.2.6	O funcionamento do ARToolKit.....	12
6.2.7	Escrevendo a aplicação	13
6.2.7.1	init().....	15
6.2.7.2	mainLoop	18
6.2.7.3	cleanup	20

6.2.8	Reconhecendo outros padrões.....	20
6.2.9	Outros programas de exemplo	23
6.2.9.1	ExView	23
6.2.9.2	Simple	24
6.3	Calibração de Câmeras com o ARToolKit.....	24
6.3.1	Executando o <code>calib_dist</code>	25
6.3.2	Executando o <code>calib_cparam</code>	30
6.4	Bibliotecas e Funções do ARToolKit.....	34
6.4.1	Estruturas de Dados Básicas	36
6.4.2	Funções de Realidade Aumentada	36
6.4.2.1	<code>arInitCparam</code>	36
6.4.2.2	<code>arLoadPatt</code>	37
6.4.2.3	<code>arDetectMarker</code> e <code>arDetectMarkerLite</code>	37
6.4.2.4	<code>arGetTransMat</code>	38
6.4.2.5	<code>arSavePatt</code>	38
6.4.3	Funções de Aquisição de Vídeo.....	39
6.4.3.1	<code>arVideoOpen</code>	39
6.4.3.2	<code>arVideoClose</code>	39
6.4.3.3	<code>arVideoInqSize</code>	39
6.4.3.4	<code>arVideoGetImage</code>	40
6.5	Limitações da RA baseada em Visão Computacional	40
6.6	Aplicações Educacionais em Ambientes de Realidade Aumentada com ARToolKit .41	
6.6.1	Introdução	41
6.6.2	Intervenções Educacionais com Realidade Aumentada	42
6.6.3	Sistema de Realidade Aumentada.....	43
6.6.4	Quebra-cabeças	45
6.6.5	Livro Interativo com Realidade Aumentada	45
6.6.6	Pá de transporte de objetos virtuais.....	46
6.6.7	Aprendizagem de mecânica quântica com realidade aumentada: O <i>Orbitário</i>	46
6.6.7.1	Dificuldades de aprendizagem da Mecânica Quântica.....	47
6.6.7.2	O computador e a aprendizagem da Mecânica Quântica	48
6.6.8	Conclusões	51
6.7	REFERENCIAS BIBLIOGRÁFICAS.....	51

1 ARToolKit – Aspectos Técnicos e Aplicações Educacionais

¹Luís Augusto Consularo, ¹Nivaldi Calonego Júnior Carlos A. Dainese¹, Tania R. Garbin¹, Claudio Kirner¹, Jorge Trindade², Carlos Fiolhais²

¹Programa de Pós Graduação em Ciência da Computação
Faculdade de Ciências Matemáticas da Natureza e Tecnologia da Informação Universidade
Metodista de Piracicaba (UNIMEP)
Rodovia do Açúcar, Km 156 – 13.400-911 - Piracicaba – SP

²Centro de Física Computacional - CFC, Universidade de Coimbra - UC, Portugal

{cdainese, ckirner, trgarbin}@unimep.br

{alberto, tcarlos}@teor.fis.uc.pt

1.1 Introdução

Este documento apresenta o pacote ARToolKit, enfatizando a instalação e o uso. O ARToolKit é uma biblioteca em linguagem C que permite aos programadores desenvolver aplicações de Realidade Aumentada. Realidade Aumentada (RA) é a sobreposição de imagens virtuais de computação gráfica sobre cenas do mundo real. Esse modelo de interface tem mostrado potencial para muitas aplicações em pesquisa industrial e acadêmica.

Nos tutoriais do ARToolKit [KATO 2002], os autores encorajam os leitores a enviar comentários e questões sobre o ARToolKit e relatos de erros e falhas para Hirokazu Kato (kato@sys.im.hiroshima-cu.ac.jp) ou para Mark Billinghurst (grof@hitl.washington.edu). Há também uma lista de discussões que é usada para difundir notícias sobre novas versões do ARToolKit, correções e aplicações que a comunidade que trabalha com ARToolKit está desenvolvendo. Para assinar esta lista, envie uma mensagem para majordomo@hitl.washington.edu com as palavras "*subscribe artoolkit*" no corpo da mensagem. Depois disso, qualquer mensagem que você enviar para artoolkit@hitl.washington.edu será enviada a todos os membros desta lista de discussões. As listas são muito úteis para encontrar dicas sobre o uso das funções, sobre equipamentos que estão sendo utilizados pelos desenvolvedores^{1,2} e ainda sobre novas versões e revisões do pacote.

¹ <http://onyx.sys.im.hiroshima-cu.ac.jp/people/kato/>

² <http://www.hitl.washington.edu/people/grof/>

Atualmente, o ARToolKit executa nas plataformas SGI Irix³, PC Linux⁴, PC Windows 95/98/NT/2000/XP⁵ e Mac OS X⁶. Há versões separadas para cada uma destas plataformas. A funcionalidade de cada versão do kit é a mesma, mas o desempenho pode variar conforme as diferentes configurações de hardware. Na plataforma SGI, por exemplo, o ARToolKit foi testado apenas para computadores SGI O2, contudo deve também executar nos computadores SGI atuais que contém com entradas de vídeo.

A versão atual do ARToolKit oferece suporte para realidade aumentada com visão direta por vídeo ou visão direta óptica. A RA por visão direta por vídeo é aquela cujas imagens virtuais são sobrepostas às imagens de vídeo ao vivo adquiridas do mundo real. A outra opção é a RA por visão direta óptica, na qual modelos de computação gráfica (os objetos virtuais) são sobrepostos diretamente às imagens do mundo real percebidas pelo sujeito. A RA por visão direta requer um dispositivo HMD (*Head Mounted Display*) e exige também um procedimento de calibração da câmera que adquire imagens do mundo real [KATO 2002; KATO & BILLINGHURST 1999]. Estes procedimentos podem variar entre algo mais simples (que será mostrado na seção 1.3), quando a visão direta é por vídeo, ou mais complexo quando por visão direta óptica.

Uma das partes mais trabalhosas no desenvolvimento de uma aplicação em RA é calcular precisamente o ponto de vista do usuário em tempo-real para que imagens virtuais sejam alinhadas com precisão às imagens dos objetos do mundo real [HARTLEY & ZISSERMAN 2003]. O ARToolKit usa técnicas de visão computacional para calcular a posição no espaço real da câmera e sua orientação em relação aos cartões marcadores, permitindo ao programador sobrepor objetos virtuais aos cartões. O pacote inclui bibliotecas de rastreamento e disponibiliza o código fonte completo, tornando possível o transporte do código para diversas plataformas ou adaptá-los para resolver as especificidades de suas aplicações. Várias aplicações simples são fornecidas com o ARToolKit para que programadores comecem rapidamente a desenvolver suas aplicações. Além disso, o ARToolKit é livre para uso em aplicações não-comerciais e é distribuído com código aberto sob licença GPL. Os interessados no uso do ARToolKit para o desenvolvimento de aplicações comerciais devem entrar em contato com Hirokazu Kato¹ ou com Mark Billinghurst².

O ARToolKit é uma novidade em termos de modelo e programação de interfaces, em que alguns tutoriais apresentados pelos autores do ARToolKit e outros interessados [BERRE et al. 2002; KATO, BILLINGHURST & POUPYREV 2000; KE 2000] são as principais fontes de informação. Assim, a elaboração de um texto que trata de mostrar os elementos necessários ao desenvolvimento desse tipo de aplicação exige uma organização de texto que é o resultado da compilação desses tutoriais e de experimentações dos próprios autores deste texto, e está distribuído em cinco sessões.

A primeira apresenta a instalação do ARToolKit. A segunda trata da programação de aplicações com o ARToolKit, explorando os exemplos mais simples de execução, que são distribuídos com o Kit, e detalhando os passos do desenvolvimento de uma aplicação típica. A terceira sessão apresenta o procedimento necessário à calibração de câmeras, usando os

³ <http://www.sgi.com/products/software/irix/>

⁴ <http://www.linux.org/>

⁵ <http://www.microsoft.com/windows>

⁶ <http://www.apple.com/macosx/>

utilitários disponíveis no ARToolKit. Os utilitários podem ser usados isoladamente, pois seus resultados são armazenados em arquivos que podem ser lidos por outras aplicações. A quarta sessão descreve as bibliotecas, enfocando as funções mais importantes utilizadas para implementar aplicações com o ARToolKit em computadores PC-compatíveis com Windows. A quinta e última sessão discute algumas limitações deste tipo de implementação de RA e mostra também algumas perspectivas para esta biblioteca.

1.2 Versões e Instalação

Há diversas versões do ARToolKit disponíveis para desenvolvimento. De uma certa forma, estas versões acompanharam a evolução do tratamento que os sistemas operacionais davam aos dispositivos de vídeo. Atualmente, há versões para o Windows, Linux, SGI e até para o MacOS X. A versão 1.0 disponibilizou uma versão para o Windows e outra para SGI. Para arquiteturas de 16 bits do Windows, o *driver* de vídeo era o VFW (*Video for Windows*) [NOMAD ELECTRONICS.COM 2004]. A versão para SGI foi a original do desenvolvimento, principalmente pelo desempenho oferecido.

A versão 2.11 foi a primeira a incluir uma distribuição para Linux. Já as revisões 2.33 e 2.40 aperfeiçoaram as versões Linux e SGI, mas não incluíram estes aperfeiçoamentos para Windows.

A revisão 2.43 foi a primeira a se aproveitar da nova arquitetura de *drivers* do Windows, a WDM (*Windows Driver Model*) [MICROSOFT 2002; NOMAD ELECTRONICS.COM 2004], que utiliza efetivamente a arquitetura 32 bits dos sistemas baseados no Windows NT, tal como a versão XP do Windows. Nesta revisão, este suporte WDM era oferecido por um SDK (*Software Development Kit*) da Microsoft que disponibilizava funções de visão computacional, o *Vision SDK* [MICROSOFT 2000]. Esta distribuição inovou ainda por adicionar um suporte à adição de modelos 3D por meio de um grafo de cena VRML [CAREY & BELL 1997]. Embora bastante limitado, o grafo de cena viabilizou a implementação de aplicações simples de Realidade Aumentada.

A versão 2.52 é exclusiva para o Windows. Inclui suporte ao VRML com melhorias na iluminação dos objetos do mundo virtual. Há também o suporte para o *Vision SDK* e ao *DirectShow* [MICROSOFT 2004], que é a inovação do software para uso de uma biblioteca gráfica que usa mais efetivamente os recursos gráficos no Windows.

A versão 2.60 é um aperfeiçoamento das versões para Linux que incluem o suporte às interfaces de câmeras Firewire⁷ (IEEE 1394) [APPLE 2004]. Este tipo de interface oferece largura de banda muito maior (próximo a 500 Mbps) que câmeras com E/S USB (entre 40 Mbps – 400 Mbps). Além disso, alguns modelos de câmeras *Firewire* tratam o sincronismo de aquisições de quadros de vídeo para aplicações com rastreamento.

A revisão 2.61 estende o ARToolKit ao Mac OS X. Esta revisão permite usar também a Firewire no próprio Mac, máquina a qual esta interface é nativa.

A revisão 2.65 é a que Thomas Pintaric⁸ denominou "não-oficial". É um desenvolvimento que está buscando oferecer o suporte VRML ao DirectShow, já que até agora este suporte só está disponível para OpenGL [SGI 2003], usando a biblioteca LibVRML97 [MORLEY 2000] (atualmente OpenVRML [OPENVRML.ORG 2003]). A iniciativa deve permitir

⁷ <http://www.ptgrey.com/>

⁸ <http://www.ims.tuwien.ac.at/~thomas/artoolkit.php>

ainda que se conecte câmeras Firewire no Windows para que a aquisição das imagens seja mais rápida, possibilitando o aumento da taxa de processamento dos quadros. O autor alega que o suporte ao DirectShow tem um desempenho em torno de 3 vezes superior ao desempenho das versões com OpenGL.

A versão 2.68 é a mais recente e tem versões disponíveis para Windows e para Linux/SGI/MacOSX. Ainda não oferece suporte ao VRML e está disponível em <http://www.eden.net.nz/phil/develop/artoolkit/>. Esta versão corrige alguns problemas da versão 2.65, que invertia as coordenadas de altura nas imagens adquiridas das câmeras. Ainda há implementações independentes do ARToolKit para Matlab (<http://mixedreality.nus.edu.sg/software.htm>), uma outra que conecta as bibliotecas do ARToolKit com Java, o JARToolKit⁹ [GEIGER et al. 2002], e também uma iniciativa de software para aplicações de RA para PDAs da linha iPaq¹⁰ [WAGNER & SCHMALSTIEG 2003].

A **Erro! A origem da referência não foi encontrada.** ilustra um diagrama resumindo as versões convencionais disponíveis do ARToolKit. As versões PC, SGI e Linux do ARToolKit estão disponíveis gratuitamente em http://www.hitl.washington.edu/research/shared_space/download/. Este tutorial descreve a instalação da versão 2.65 com VRML, que está disponível em <http://www.hitl.washington.edu/artoolkit/download.htm>.

1.2.1 Instalação no PC-Windows

O ARToolKit para o PC-Windows é distribuído em um único arquivo comprimido, no caso da versão 2.65 como ARToolKit2.65.zip ou como ARToolKit2.65VRML.zip. Uma vez que esta cópia de arquivo esteja no diretório no qual você queira que o ARToolkit fique, descomprima-o usando o utilitário de descompressão de sua preferência (WinZip, Rar, PKZip, etc..).

O utilitário de descompressão deve criar a estrutura de diretórios ilustrada na Figura 1-2. O diretório **bin** contém alguns programas que podem ser executados conforme descreve a próxima seção. Os códigos-fonte destes programas ficam no diretório **examples**. As bibliotecas do ARToolKit ficam no diretório **lib** e o código-fonte completo para as bibliotecas, no diretório **lib/SRC**.

Esta versão do ARToolKit foi compilada e testada em PCs executando Windows XP com instalação das câmeras QuickCam Express da Logitech¹¹ e Webcam 3 modelo da Creative Labs¹² conectadas a portas USB¹³ versão 1.1 (40 Mbps). O ARToolKit usa ou o Vision SDK (existe uma distribuição do ARToolKit específica para o Vision SDK) ou o DirectX 9¹⁴ da Microsoft (segundo a página do ARToolKit, da versão 8 em diante) para que não

⁹ <http://www.c-lab.de/jartoolkit>

¹⁰ http://studierstube.org/handheld_ar

¹¹

<http://www.logitech.com/index.cfm/support/products/details/US/EN,CRID=1795,CONTENTID=6041>

¹² <http://www.creative.com/products/product.asp?prodid=15>

¹³ <http://www.usb.org/home>

¹⁴ <http://www.microsoft.com/directx>

haja motivo de incompatibilidade entre câmeras e placas de aquisição de vídeo que sejam suportados pelo Vision SDK, Video for Windows ou pelo WDM (Windows Driver Model) da Microsoft. Os códigos-fonte foram compilados com o MS Visual C++ 6.0 e o MS Visual C++ 7.0 (ambos com instalação do Visual Studio¹⁵).

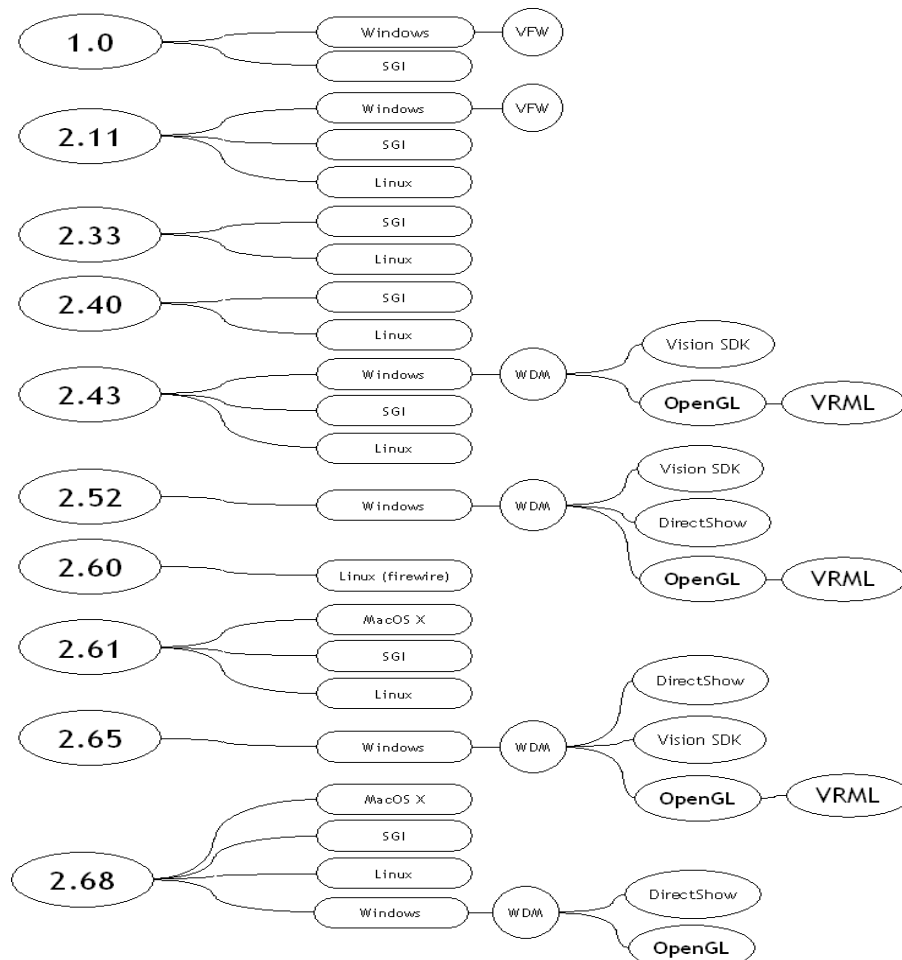


Figura 1-1 - Versões do ARToolKit.

Os programas de exemplo incluídos com o ARToolKit, todos, utilizam a biblioteca GLUT de interface com o OpenGL. Portanto, para compilar estes programas, deve estar instalada a versão 3.6 ou alguma versão mais atualizada. As bibliotecas GLUT estão disponíveis em <http://opengl.org/resources/libraries/glut.html>. Para economizar tempo, copie os diretórios do OpenGL nos diretórios do ARToolKit. O diretório **GL** com os protótipos (arquivos com extensão **.h**) do OpenGL no diretório **include** do ARToolKit, o arquivo de biblioteca **glut32.lib** no diretório **lib** e o **glut32.dll** no diretório **bin**. Se existe uma instalação das bibliotecas em outro diretório, com os caminhos todos definidos na configuração de projetos do Visual C, então isto não é necessário. Para definir estes caminhos do Visual C, é necessário abrir o arquivo do *workspace* de sua aplicação, selecionar no projeto de sua aplicação o item *Project* e, então, a opção *Settings*. Surgirá uma caixa de diálogo com várias configurações. Você precisará das opções *C/C++* e *Link*.

¹⁵ <http://msdn.microsoft.com/vstudio>

Na opção *C/C++* você poderá adicionar novos diretórios de protótipos selecionando a opção *Preprocessor* em *Category*. Na caixa de texto *Additional include directories*, inclua o diretório de sua preferência. Na opção *Link*, selecione *Input* em *Category*. Na caixa de texto *Additional library path*, inclua os diretórios de bibliotecas de sua preferência, isto é, aqueles nos quais você incluiu os arquivos com extensão **.lib**.

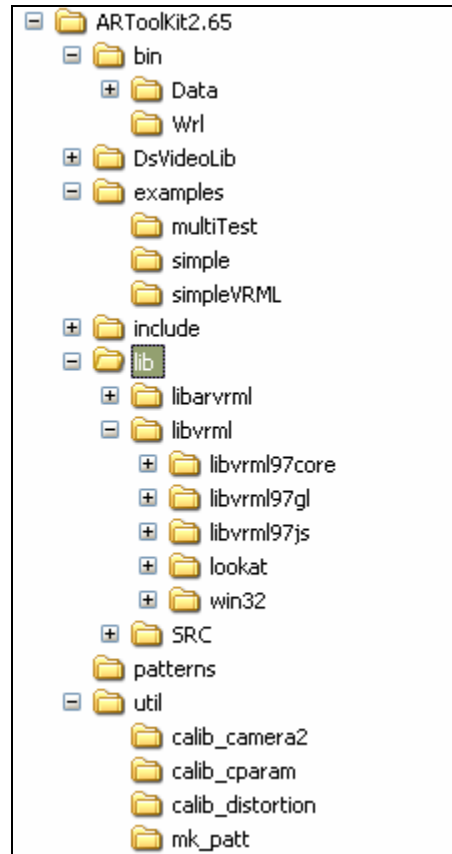


Figura 1-2 – Estrutura de diretórios depois de instalado o ARToolKit 2.65 com VRML

1.2.2 Programação de Aplicações

A programação de aplicações no ARToolKit, em sua distribuição original, não disponibiliza qualquer ferramenta de autoria. Portanto, é necessário conhecer aspectos de configuração do hardware no qual a aplicação executará, bem como o fluxo de controle de uma aplicação típica. O fluxo de controle disponibilizado pelo ARToolKit é bastante simples e inclui funções para leitura de eventos de teclado e de mouse, inicializações e finalizações de dispositivos de vídeo além, é claro, do próprio laço principal. Uma vez instalado corretamente, o ARToolKit inclui um programa-exemplo denominado **simple.exe**, que pode ser localizado no diretório **bin**, e executado para testar se o ARToolKit está funcionando. É necessário imprimir os marcadores de referência, que estão contidos no diretório **patterns**, para testar o programa. Estes marcadores estão nos arquivos **pattSample1.pdf**, **pattSample2.pdf**, **pattKanji.pdf** e **pattHiro.pdf**. O desempenho será melhor desempenho se os marcadores forem fixados (com cola, por exemplo) em uma superfície plana e rígida.

As próximas seções descrevem o hardware necessário para executar o ARToolKit e depois mostram como executar os exemplos do ARToolKit em um PC-Windows. Em qualquer uma das instalações das versões anteriores, a saída do programa **simple.exe** deve ter o mesmo comportamento.

1.2.3 Hardware necessário

Os requisitos básicos de hardware para desenvolver e executar aplicações do ARToolKit são: uma câmera de vídeo e uma interface ou um dispositivo de aquisição de vídeo com seus respectivos *drivers*. Em um PC-Windows (95/98/2000/XP) a captura de vídeo pode se dar por uma câmera USB, por um dispositivo de aquisição de vídeo ou por placas gráficas com alguma entrada de vídeo. O dispositivo escolhido exige que se instale *drivers* Vfw ou WDM fornecidos pelo próprio fabricante do dispositivo.

1.2.4 A saída do simple.exe

Depois de invocar o **simple.exe**, aparecerá na tela uma janela de vídeo. Quando apontar a câmera para cada um dos marcadores aparecerá uma cena. Por exemplo, quando a câmera captura a imagem do marcador “Hiro” (Figura 6a), é apresentada na tela do computador a imagem de um cubo azul sobre esse marcador, conforme ilustra a Figura 5. Conforme os marcadores reais se moverem, os objetos virtuais devem se mover e aparecer exatamente alinhados com os marcadores reais. As figuras a seguir mostram os passos da execução do programa, antes que possam ser vistas a imagem do vídeo e o objeto virtual sobre ela.

A primeira ação que pode ser requisitada ao usuário é uma caixa de diálogo (veja Figura 1-3) para configurar a câmera (se foi configurado para que isso aconteça).

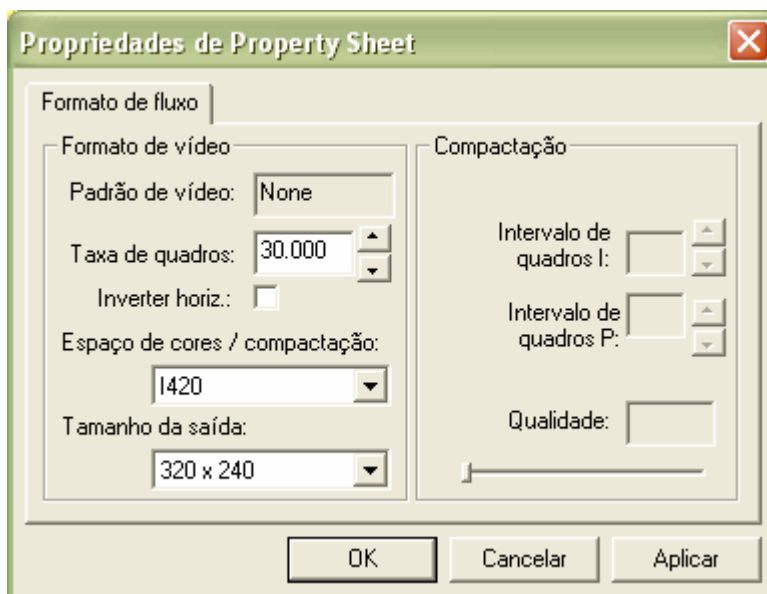


Figura 1-3 – Janela de propriedades da câmera. Os elementos configuráveis desta caixa de diálogo depende da câmera que foi instalada.

Clicando OK nesta caixa de diálogo, você verá uma janela para entrada de comandos mostrando as saídas textuais do programa (veja Figura 1-4).

```
C:\ VARToolkit\VARToolkit2.65vrm\bin\simple.exe
Image size (x,y) = (320,240)
*** Camera Parameter ***
-----
SIZE = 320, 240
Distotion factor = 215.000000 65.000000 -111.000000 0.955272
380.80979 0.000000 165.000000 0.000000
0.000000 422.44376 98.000000 0.000000
0.000000 0.000000 1.000000 0.000000
-----
```

Figura 1-4 – Saída em janela de texto do programa `simple.exe`, mostrando os parâmetros de exibição e de calibração da câmera.

Finalmente, aparecerá a janela de visualização da cena com um cubo sobre um marcador (veja a Figura 1-5).

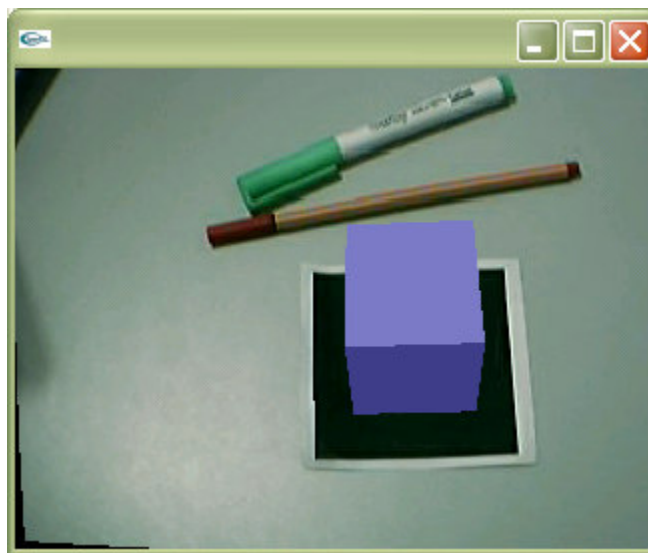


Figura 1-5 – Resultado da execução do arquivo `simple.exe`, mostrando um cubo virtual sobre o marcador real.

Para que o objeto virtual apareça, toda a borda preta do quadrado que envolve o desenho padrão deve estar enquadrada pela imagem da câmera, assim como o próprio desenho padrão. Se as imagens virtuais não aparecerem ou piscarem aparecendo e desaparecendo da imagem, é por causa das condições de iluminação do ambiente. Isto pode ser corrigido alterando o valor do limiar usado pelas rotinas de processamento de imagens. Se você pressionar a tecla `<t>` no teclado, será pedido a você que entre com o valor de um novo limiar. Este novo limiar pode estar entre 0 e 255; o normal é 100.

Pressionando a tecla `<esc>` sai do programa e mostra a informação da taxa de atualização em quadros por segundo (*frame rate*). Esta informação é útil par testar o desempenho de sua configuração. Uma taxa pequena (em torno de 10 quadros/segundo) pode ser inviável para sua aplicação. *OBS:* Se o programa sair anormalmente ou por alguma outra maneira

que não seja pressionando <esc>, recomenda-se que você reinicie seu computador antes de usar novamente alguma aplicação de vídeo.

O programa **simple** mostra como uma aplicação baseada no software ARToolKit é capaz de calcular o ponto de vista da câmera em tempo real e usar esta informação para sobrepor precisamente o objeto do mundo real com imagens de objetos virtuais.

1.2.5 A saída do simpleVRML.exe

Além do **simple.exe**, há outro programa para testar sua configuração, o **simpleVRML.exe**. Este programa permite testar se a biblioteca VRML está funcionando e também se os parâmetros de câmera, marcadores e objetos virtuais estão de acordo com suas necessidades. Este programa só está disponível nas distribuições que integram a biblioteca LibVRML97 ao ARToolKit.

O programa **simpleVRML** usa arquivos de configuração como o arquivo **camara_para.dat**, que contém os parâmetros da câmera; bem como o **vrml_data.dat** que associa os objetos virtuais com os padrões dos marcadores. Estes arquivos devem estar no diretório **bin/Data**. O arquivo **vrml_data.dat**, no localizado no diretório **bin/Wrl**, armazena referências para arquivos VRML e para os mapas de bits (*bitmaps*), que ficam no diretório **bin/Data**. Os conteúdos desses arquivos pode ser verificados abrindo-se o arquivo **vrml_data.dat** em um editor de texto qualquer. O arquivo que vem com a distribuição 2.65 do ARToolKit e que acompanha este tutorial tem o conteúdo mostrado no Código 1.

```
#the number of patterns to be recognized
2

#pattern 1
VRML Wrl/bud_B.dat
Data/patt.hiro
80.0
0.0 0.0

#pattern 2
VRML Wrl/snoman.dat
Data/patt.kanji
80.0
0.0 0.0
```

Código 1 – Código contido no arquivo vrml_data.dat, que descreve uma aplicação.

Neste caso, são dois padrões: um com a palavra “Hiro” (que são as iniciais do autor *Hirokazu*, que está no arquivo **Data/patt.hiro**) escrita no quadrado e outra com um ideograma *kanji* (**Data/patt.kanji**). O padrão “**patt.hiro**” é uma referência ao objeto VRML que está descrito no arquivo **Wrl/bud_B.dat**, que é uma animação de uma abelha sobrevoando uma flor. O padrão *kanji* referencia o arquivo **Wrl/snoman.dat**, que associa código VRML para o desenho de um boneco de neve. Portanto, os arquivos “.dat” associam as marcas aos arquivos de cena. As duas últimas linhas de cada padrão são parâmetros que especificam o tamanho com que o objeto deve aparecer na imagem e o local do centro deste objeto em relação ao centro da placa.

A execução deste programa, assim como no **simple.exe**, mostrará uma tela de configuração da câmera e depois uma tela de comandos. Na janela de visualização da cena poderão ser vistos os padrões (veja Figura 1-6a) e seus respectivos objetos (veja Figura 1-6b).

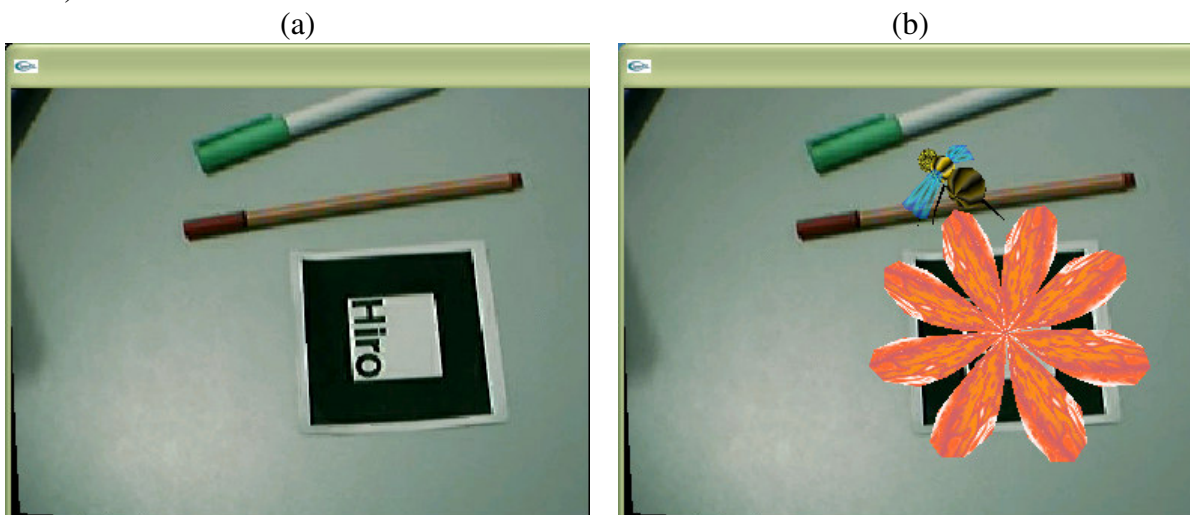


Figura 1-6 – Resultado da execução do programa simplevrml.exe: (a) mostrando apenas a aquisição de vídeo e (b) mostrando um objeto virtual e animado sobre o marcador.

1.2.6 O funcionamento do ARToolKit

O ARToolKit usa técnicas de visão computacional para calcular o ponto de vista real da câmera em relação a um marcador no mundo real. Há vários passos ilustrados nas figuras para mostrar o funcionamento do ARToolKit. O primeiro passo mostra que a imagem de vídeo capturada, veja Figura 1-7a, é transformada em uma imagem binária (em P&B) baseada no valor do limiar de intensidade, veja Figura 1-7b. Depois, o ARToolKit encontra todos os quadrados na imagem binária, muitos dos quais não correspondem a marcadores de referência. Para cada quadrado, o desenho padrão dentro dele é capturado e comparado com alguns gabaritos pré-treinados. Se houver alguma similaridade, então o ARToolKit considera que encontrou um dos marcadores de referência. O ARToolKit usa então o tamanho conhecido do quadrado e a orientação do padrão encontrado para calcular a posição real da câmera em relação a posição real do marcador. Uma matriz 3x4 conterá as coordenadas reais da câmera em relação ao marcador. Esta matriz é usada para calcular a posição das coordenadas da câmera virtual. Se as coordenadas virtuais e reais da câmera forem iguais, o modelo de computação gráfica pode ser desenhado precisamente sobre o marcador real (veja Figura 1-7c). A API OpenGL é usada para calcular as coordenadas virtuais da câmera e desenhar as imagens virtuais.

(a)

(b)

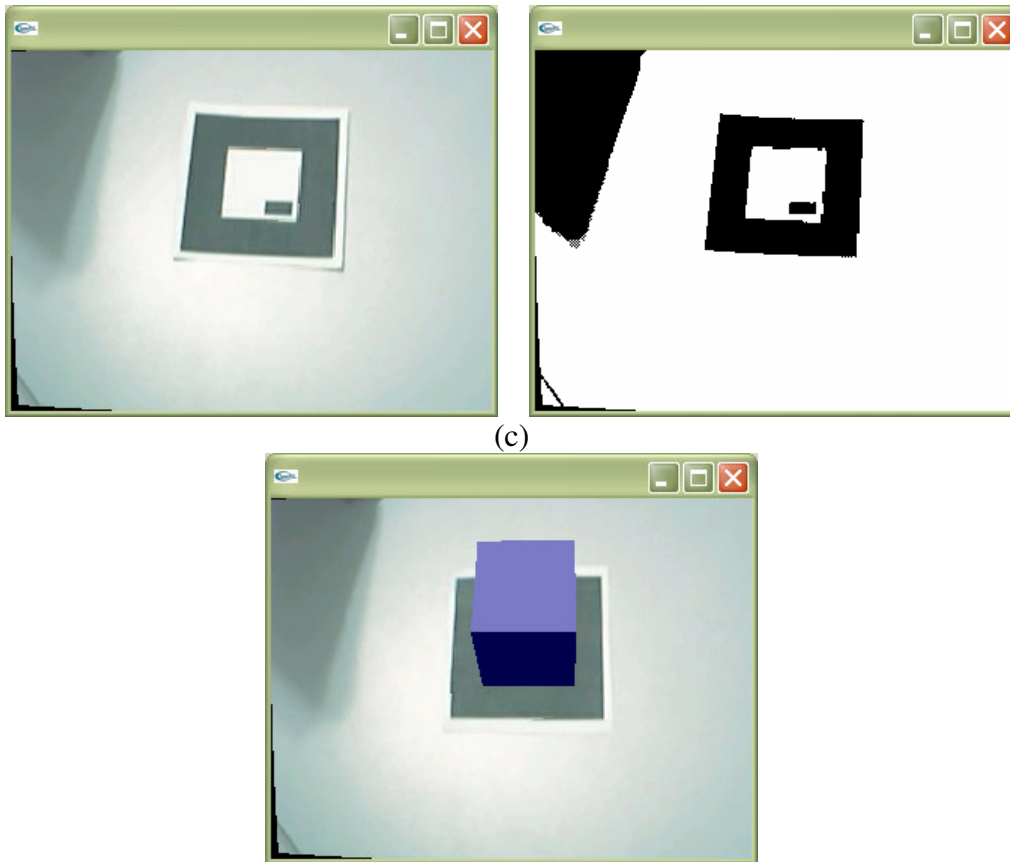


Figura 1-7 – Resultado da execução do programa `simple.exe` mostrando (a) a imagem da cena com um marcador; (b) a imagem limiarizada e (c) o cubo virtual sobreposto ao marcador.

O diagrama da Figura 1-8 ilustra os passos do processamento de imagens usado para detectar a geometria do marcador e depois o posicionamento de um objeto virtual sobre este marcador detectado.

1.2.7 Escrevendo a aplicação

O desenvolvimento de aplicações de RA com o ARToolkit requer duas etapas: escrever a aplicação e treinar as rotinas de processamento de imagens sobre os marcadores do mundo real que serão usadas na aplicação.

Para escrever aplicações com o ARToolkit, deve-se seguir os seguintes passos:

<i>Passo 1:</i>	<ul style="list-style-type: none"> • Inicializar o caminho dos parâmetros de vídeo; • Ler os arquivos de padrões de marcadores; • Ler os parâmetros de câmera;
<i>Passo 2:</i>	<ul style="list-style-type: none"> • Capturar uma quadro da entrada de vídeo;
<i>Passo 3:</i>	

<ul style="list-style-type: none"> • Detectar os marcadores e reconhecer os padrões no quadro capturado da entrada de vídeo;
<i>Passo 4:</i>
<ul style="list-style-type: none"> • Calcular a transformação da câmera em relação aos padrões detectados;
<i>Passo 5:</i>
<ul style="list-style-type: none"> • Desenhar os objetos virtuais nos padrões detectados;
<i>Passo 6:</i>
<ul style="list-style-type: none"> • Fechar a entrada de vídeo.

Os passos 2 até 5 são repetidos continuamente até que a aplicação termine, já os passos 1 e 6 são executados, respectivamente, apenas na inicialização e na finalização da aplicação. Além destes passos, a aplicação pode precisar responder ao mouse, ao teclado ou a outros eventos específicos da aplicação.

Para mostrar em detalhes como desenvolver uma aplicação, seguiremos cada passo no código fonte do programa **simpleVRML**. Este exemplo é encontrado no diretório **examples**.

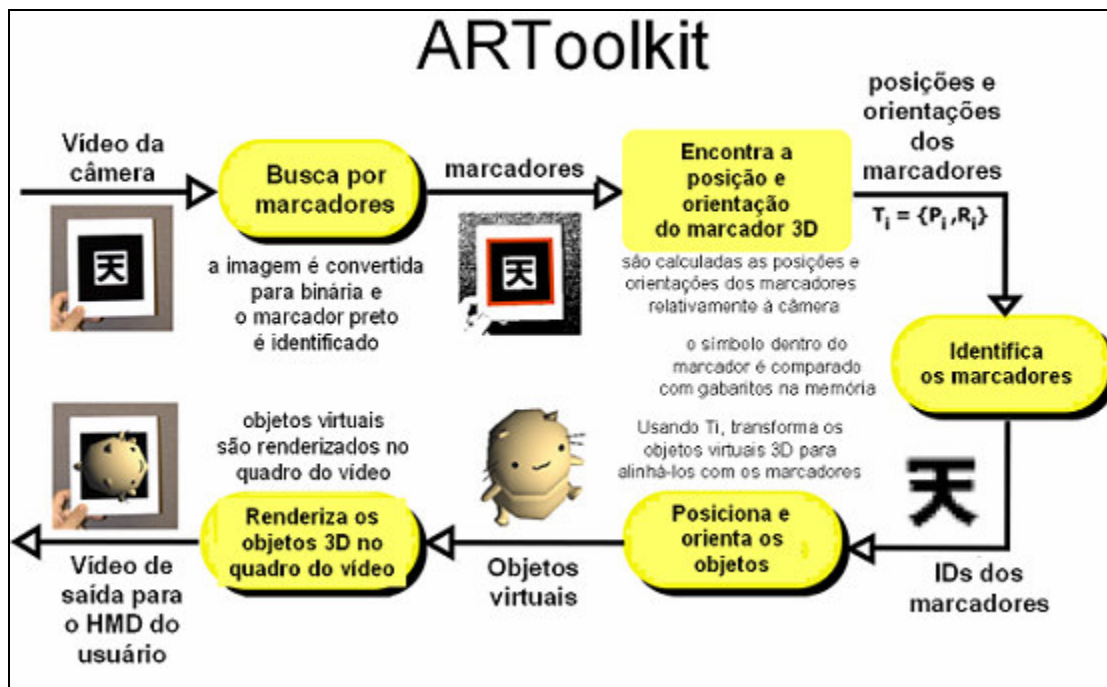


Figura 1-8 – Diagrama descrevendo os passos da detecção dos marcadores e o posicionamento de objetos virtuais sobre os marcadores detectados na cena. Traduzido do tutorial dos autores [KATO, BILLINGHURST & POUPYREV 2000].

O arquivo que estaremos visualizando é o **simpleVRML.c**. Este programa consiste de uma função **main** e várias funções gráficas. A função principal (**main**) é mostrada no Código 2.

```

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

```

Código 2 - Função principal de uma aplicação de RA no ARToolkit.

Esta função chama uma outra função de inicialização (**init()**) que contém o código para definição do caminho dos parâmetros do dispositivo de vídeo, da leitura dos parâmetros dos marcadores e da câmera e da janela gráfica. Isto corresponde ao Passo 1. Depois, a função **arVideoCapStart()** inicia a captura de imagens de vídeo. Finalmente, a função **argMainLoop** é chamada para iniciar o laço do programa principal, associar a função **keyEvent** aos eventos do teclado e a função **MainLoop** com o laço principal da renderização gráfica. A definição de **argMainLoop** está contida no arquivo **gsub.c**, que pode ser encontrado no diretório **lib/SRC/Gl/**.

O arquivo **simpleVRML.c** contém as funções que correspondem aos seis passos de aplicação acima são mostradas na tabela abaixo. As funções correspondentes aos passos 2, 3, 4 e 5 são chamadas dentro da função **mainLoop**. Estas chamadas de função serão explicadas com mais detalhes no restante da seção.

Passo do ARToolkit	Função
1.Inicializa a aplicação	init
2.Captura um quadro de vídeo	arVideoGetImage
3.Detecta os marcadores	arDetectMarker
4.Calcula a transformação da câmera	arGetTransMat
5.Desenha os objetos virtuais	draw
6.Fecha o dispositivo de vídeo	cleanup

1.2.7.1 init()

A função **init()** é chamada a partir da função principal e é usada para definir o caminho dos parâmetros para o dispositivo de vídeo e para ler os parâmetros iniciais da aplicação ARToolkit. Os parâmetros principais para uma aplicação ARToolkit são:

1. os padrões que serão usados para comparar os padrões e encontrar a correspondência entre o desenho no padrão e um objeto virtual.
2. as características da câmera de vídeo que está sendo usada.

Estes dois parâmetros principais podem ser lidos de arquivos de configuração, ou fornecidos na linha de comando da aplicação ou ainda definidos em constantes diretamente no código. O parâmetro *default* da rotina **init** para o nome do arquivo de parâmetros da câmera é **Data/camera_para.dat**, enquanto o nome *default* para o arquivo de objetos é **Data/vrml_data**. O arquivo que contém os nomes dos padrões e dos objetos virtuais é lido com a chamada de função do Código 3.

```

/* carrega os dados do objeto -

```

```
marcadores treinados e arquivos de mapas de bits */
if ((object = read_VRMLdata(modelname, &objectnum)) == NULL)
    exit(0);
```

Código 3 – Chamada da função que carrega os padrões a serem reconhecidos.

A função **read_VRMLdata** obtém todos os padrões treinados, que correspondem aos nomes dos padrões lidos na biblioteca **AR**. Antes destes nomes serem lidos, o dispositivo de vídeo é aberto e é encontrado o tamanho configurado da imagem de vídeo (veja o Código 4)

```
/* abre o dispositivo de vídeo */
if (arVideoOpen(vconf) < 0)
    exit(0);

/* encontra o tamanho da janela */
if (arVideoInqSize(&xsize, &ysize) < 0)
    exit(0);
printf("Image size (x, y) = (%d, %d)\n", xsize, ysize);
```

Código 4 – Abertura do dispositivo de vídeo e recuperação da imagem de vídeo.

A variável **vconf** contém a configuração inicial do vídeo e é definida no topo do **simpleVRML.c**. Esta variável é uma cadeia de caracteres e sua configuração deve seguir uma sintaxe de configuração de vídeo definida pelo **ARToolKit**. Há um comentário explicando esta sintaxe e mostrando alguns exemplos. Agora, os parâmetros da câmera são lidos, como mostra o Código 5.

```
/* configura os parâmetros iniciais da câmera */
if (arParamLoad(cparaname, 1, &wparam) < 0) {
    printf("Camera parameter load error!\n");
    exit(0);
}
```

Código 5 – Leitura dos parâmetros da câmera.

Depois, os parâmetros são transformados para o tamanho real da imagem, pois os parâmetros da câmera mudam de acordo com o tamanho da imagem, mesmo que seja utilizada a mesma câmera (veja o Código 6).

```
arParamChangeSize(&wparam, xsize, ysize, &cparam);
```

Código 6 – Configuração dos parâmetros da câmera.

Os parâmetros da câmera são configurados de acordo com a leitura anterior e então são mostrados na tela (veja o Código 7). Logo após a janela gráfica é aberta:

```
arInitCparam( &cparam );
printf("*** Camera Parameter ***\n");
arParamDisp( &cparam );
/* abre a janela grafica */
argInit( &cparam, 1.0, 0, 0, 0, 0 );
/* cparam, zoom, fullflag, xwin, ywin, hmd */
```

Código 7 – Inicialização dos parâmetros e sua exibição na tela.

Cabem alguns comentários relativos aos parâmetros de **argInit**. O primeiro parâmetro é na verdade a estrutura que contém os parâmetros da câmera. O segundo parâmetro é o *zoom*,

isto é, o quanto se quer aumentar a imagem de aquisição para ser exibida na tela. Lembre-se que o *zoom* aumenta o tamanho do *pixel*. O terceiro parâmetro é um sinal indicando se a janela gráfica deve cobrir toda a tela (1, neste caso) ou se deve abrir uma janela sobre a área de trabalho (zero, neste caso). Os outros dois parâmetros são os tamanhos de eventuais extensões da tela contendo a imagem da câmera e, finalmente, o último parâmetro indica se a inicialização deve considerar que o display é um HMD, isto é, um capacete.

A atribuição ao parâmetro **arImageProcMode = AR_IMAGE_PROC_IN_FULL** indica ao ARToolkit para processar as imagens com o tamanho completo e original adquirido da câmera (veja o Código 8). Isto pode tornar o processamento muito lento, dependendo do resultado desejado. Por isso, é possível reduzir a resolução da imagem a ser processada fazendo **arImageProcMode = AR_IMAGE_PROC_IN_HALF**. Esta configuração deixa o processamento mais rápido, porém menos preciso.

```
arImageProcMode = AR_IMAGE_PROC_IN_FULL;
// arImageProcMode = AR_IMAGE_PROC_IN_HALF;

argDrawMode = AR_DRAW_BY_TEXTURE_MAPPING;
// argDrawMode = AR_DRAW_BY_GL_DRAW_PIXELS;
```

Código 8 – Configurações dos parâmetros **arImageProcMode** e **argDrawMode**.

O parâmetro **argDrawMode = AR_DRAW_BY_TEXTURE_MAPPING** permite desenhar os *pixels* do vídeo como um mapeamento de texturas (veja o Código 8). Se a placa gráfica do computador tiver esta capacidade implementada, o processamento do vídeo ficará bem mais rápido. Com **argDrawMode = AR_DRAW_BY_GL_DRAW_PIXELS** os *pixels* da imagem são desenhados individualmente, e portanto, para equipamentos mais novos é menos recomendado.

Agora é o momento de carregar os dados VRML, isto é, os objetos 3D representados em arquivos cujo nome é passado em **model_name** (veja o Código 9). Nesta chamada de função retorna ainda uma estrutura descrevendo este objeto e o número de objetos que devem ser lidos.

```
/* carrega os dados dos objetos - marcadores treinados e mapas de
bits associados */
if ((object = read_VRMLdata(model_name, &objectnum)) == NULL )
    exit(0);
printf("Objectfile num = %d\n", objectnum);
```

Código 9 – Carregamento dos objetos virtuais.

É momento de verificar se o objeto pode ser renderizado pela biblioteca **LibVRML** ou não (veja o Código 10). Algum impedimento desta renderização em um dos objetos fará com que a aplicação não prossiga.

```
/* testa a renderização de todos os objetos VRML */
printf("about to render VRML objects \n");
glEnable(GL_TEXTURE_2D);
for (i = 0; i < objectnum; i++ ) {
    printf("rendering %d \n",i);
    arVrml97Draw( object[i].vrml_id );
}
glDisable(GL_TEXTURE_2D);
```

Código 10 – Renderização dos objetos virtuais carregados no Código 9.

Depois de renderizados os objetos, os parâmetros de iluminação dos modelos deverão ser inicializados (veja o Código 11). Segundo um dos próprios autores do ARToolkit, brevemente haverá uma versão com tratamento de sombreado em tempo real no ARToolkit.

```
/* inicializa a iluminação */
init_lights();
```

Código 11 – Inicialização da iluminação.

1.2.7.2 mainLoop

Esta é a rotina na qual a maior parte das chamadas de função do ARToolkit. Ela contém o código correspondente aos passos 2, 3, 4 e 5 da aplicação.

Primeiro o quadro do vídeo é capturado usando a função **arVideoGetImage** (veja o Código 12).

```
/* captura um quadro de vídeo */
if ((dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
    arUtilSleep(2);
    return;
}
```

Código 12 – Captura de um quadro de vídeo.

Algumas câmeras não possuem configuração de inversão vertical da imagem. Nas câmeras testadas para escrever este tutorial, as imagens apareciam invertidas. Portanto, foi incluído um código para inverter a imagem. Dependendo do processamento que estiver sendo feito, esta etapa pode deixar a aplicação mais lenta. Se esta inversão não for imprescindível, pode ser retirada.

Um passo importante em aplicações que incluem a biblioteca LibVRML no ARToolkit é a inicialização do temporizador do renderizador.

Depois disso, a imagem é mostrada e um outro quadro é capturado (veja o Código 13).

```
argDrawMode2D();
/* mostra a imagem na tela como um mapa de bits
argDispImage( dataPtr, 0, 0 );
/* captura o próximo quadro de vídeo */
arVideoCapNext();
```

Código 13 – Exibição da imagem e captura.

Então os quadrados que contenham os marcadores corretos na imagem capturada devem ser detectados. Para isso são passados os parâmetros: **dataPtr**, que é o ponteiro indicando o início da memória de vídeo; **thresh** que é o limiar para separar o que é fundo de um eventual objeto; **marker_info**, que é a estrutura que contém informações sobre o marcador e; **marker_num** que é um valor contendo o número de marcadores detectados. Um retorno negativo indica uma falha que deve ocasionar a saída da aplicação (veja o Código 14).

```
/* detecta os macadores no quadro de vídeo */
if (arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
    cleanup();
    exit(0);
}
```

```
}
```

Código 14 – Chamada da função de detecção de marcadores.

O número de marcadores encontrados está contido na variável **marker_num**. O parâmetro **marker_info** é um ponteiro para uma lista de estruturas de marcadores contendo a informação de coordenadas, valores de confiança do reconhecimento e os identificadores dos objetos correspondentes a cada um dos marcadores.

Todos os valores de confiança dos marcadores detectados são comparados e associados ao número identificador correto, isto é, com o maior valor de confiança (veja Código abaixo).

```
/* verifica se o objeto deve ou não ser visível */
for ( i = 0; i < objectnum; i++ ) {
    k = -1;
    for ( j = 0; j < marker_num; j++ ) {
        if (object[i].id == marker_info[j].id ) {
            if (k == -1 )
                k = j;
            else {
                if (marker_info[k].cf < marker_info[j].cf )
                    k = j;
            }
        }
    }
    if (k == -1 ) {
        object[i].visible = 0;
        continue;
    }
}
```

Código 15 – Verificação do valor de confiança para exibição do objeto.

A transformação entre os cartões marcadores e a câmera pode ser encontrada usando a função **arGetTransMat**. A posição real da câmera e sua orientação com relação ao objeto marcador **i** estão contidas na matriz 3x4 **object [i] .trans** (veja o Código 16).

```
if ( object[i].visible == 0 ) {
    arGetTransMat( &marker_info[k],
                  object[i].marker_center,
                  object[i].marker_width,
                  object[i].trans);
}
else {
    arGetTransMatCont(&marker_info[k], object[i].trans,
                     object[i].marker_center, object[i].marker_width,
                     object[i].trans);
}
object[i].visible = 1;
```

Código 16 – Chamadas das funções para obter o posicionamento e a pose da câmera.

Finalmente, os objetos virtuais podem ser desenhados na placa usando a função **draw** (veja o Código 17).

```
/* desenha os objetos virtuais associados aos padrões marcadores */
glClearDepth( 1.0 );
glClear(GL_DEPTH_BUFFER_BIT);
glDisable(GL_BLEND);
draw( object, objectnum );
```

Código 17 – Código para desenhar os objetos virtuais.

A função **draw** e as rotinas gráficas associadas ao OpenGL estão no arquivo `draw_object.c`. Na função **draw**, a matriz 3x4 contida em `object[k].trans` é convertida em um arranjo de 16 posições, `glpara`, usando a chamada de função **argConvGLpara**. O arranjo `glpara` é então passado para a função **draw_object**. Estes 16 valores são os valores da posição e orientação da câmera real, portanto usá-los para posicionar a câmera virtual faz com que os objetos gráficos desenhados pareçam estar exatamente alinhados com o marcador físico correspondente.

A posição da câmera virtual é configurada na função **draw_object** usando a função **glLoadMatrix(gl_para)** do OpenGL. Diferentes objetos gráficos são então desenhados de acordo com o marcador na placa, tal como um cubo para o padrão cujo nome é "cubo" e um cone para o padrão cujo nome é "cone". É neste momento que é chamado o renderizador VRML, na biblioteca LibVRML97, que vai associar o grafo de cena ao marcador detectado. O relacionamento entre os padrões e os objetos virtuais mostrados sobre os padrões é determinado no arquivo `vrml_data` no diretório `bin/Data`.

Ao final da execução da função **mainLoop**, o temporizador do renderizador VRML é atualizado para que eventuais e o *buffer* de atualização disponibilizado pela troca entre o que está sendo desenhado e o que está sendo mostrado (veja o Código 18).

```
/* atualiza a animação VRML */
arVrml97TimerUpdate();

/* disponibiliza um novo quadro de vídeo */
argSwapBuffers();
```

Código 18 – Atualizações do temporizador do renderizador e do buffer de vídeo.

Os passos mencionados acima executam a cada interação do laço de renderização. Enquanto o programa estiver executando, os eventos do mouse são tratados pela função **mouseEvent** e os eventos de teclado pela função **keyEvent**.

1.2.7.3 cleanup

A função **cleanup** é chamada para finalizar o processamento de vídeo e desconectar o dispositivo de vídeo, liberando-o para outras aplicações. Isto acontece quando se chama as rotinas **arVideoCapStop()**, **arVideoClose()** e **argCleanup()**.

1.2.8 Reconhecendo outros padrões

O programa **simpleVRML** faz uma comparação com padrões pré-definidos para reconhecer os diferentes padrões dentro dos quadrados marcadores. Os quadrados no fluxo de entrada de vídeo são comparados com os padrões pré-treinados. Estes padrões são carregados em tempo de execução e ficam no diretório `bin/Data`. Neste diretório, o arquivo texto `vrml_data` especifica quais objetos marcadores devem ser reconhecidos e os padrões são associados a cada objeto. O arquivo `vrml_data` começa com o número de objetos que serão especificados, seguido de uma estrutura de dados para cada objeto. Cada

um dos marcadores no arquivo **vrml_data** é especificado por uma estrutura com os elementos:

- Nome
- Nome do arquivo de padrões a ser reconhecido
- Largura do marcador a ser rastreado

Por exemplo, a estrutura correspondente ao marcador com o cubo virtual é mostrada no Código 19.

```
#padrao 1
cube
Data/patt.hiro
80.0
```

Código 19 – Exemplo de estrutura que define o padrão do marcador e seu objeto virtual.

Observe que as linhas que começam com um caracter **#** são linhas de comentário e são ignoradas pelo analisador do arquivo.

Para mudar os padrões que são reconhecidos, o nome de arquivo do padrão (no exemplo, `patt.hiro`) deve ser substituído pelo nome do arquivo com o padrão desejado. Estes arquivos de padrões pré-definidos são simplesmente um conjunto de exemplos de imagens do padrão desejado. O programa que cria estes arquivos de padrões pré-definidos é chamado **mk_patt** e fica no diretório **bin**. O código fonte do **mk_patt** é o arquivo **mk_patt.c**, que fica no diretório **util**.

Para criar um novo padrão pré-definido, primeiro edite em um editor de imagens o arquivo **blankPatt.gif**, localizado no diretório **patterns**. É apenas um quadrado preto com um quadrado branco no meio. Então crie uma imagem em preto e branco do padrão desejado que caiba no meio deste quadrado e imprima-o. Os melhores padrões são assimétricos e sem muitos detalhes. A Figura 1-9 mostra alguns padrões possíveis. Insira o novo padrão no centro do quadrado vazio.

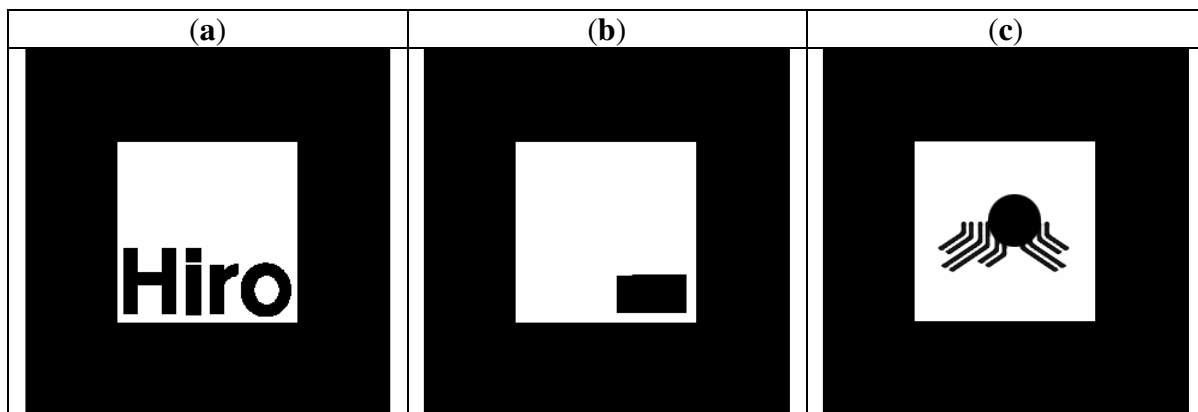
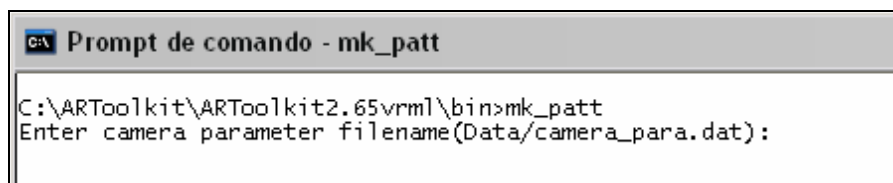


Figura 1-9 – Exemplos de padrões que podem ser usados como marcadores.

Uma vez confeccionado o novo padrão, vá para o diretório **bin** e **execute** o programa **mk_patt**. Será pedido para que você entre com um nome de arquivo de parâmetros de

câmera (veja Figura 1-10). Entre com o nome do arquivo **camera_para.dat**. Este é o nome *default* para o arquivo de parâmetros de câmera.



```
C:\ARToolkit\ARToolkit2.65vrm1\bin>mk_patt
Enter camera parameter filename(Data/camera_para.dat):
```

Figura 1-10 – A execução do utilitário `mk_patt`.

O programa abrirá então uma janela de vídeo conforme mostra a Figura 11. Coloque o padrão a ser treinado em uma superfície plana em condições de iluminação similares àquelas que existirão quando a aplicação de reconhecimento estará executando. Então, coloque a câmera de vídeo apontando diretamente para o padrão e vire-o até que um quadrado com dois lados vermelhos e dois lados verdes apareça em torno do padrão (veja a Figura 1-11). Isto indica que o software **mk_patt** encontrou o quadrado em torno do padrão de teste. A câmera deve ser rotacionada até que os lados vermelhos do quadrado estejam no topo e à esquerda do quadrado na imagem de vídeo, conforme mostra a Figura 11. Uma vez que o quadrado encontrado esteja orientado corretamente clique o botão esquerdo do mouse. Será então pedido um nome de arquivo para o padrão. Fornecido o nome do arquivo, é gerada uma imagem em mapa de bits do padrão que é criado e copiado para este arquivo. Este padrão é então usado na comparação de padrões do ARToolKit. Outros podem ser treinados simplesmente apontando a câmera para novos padrões e repetindo o processo, ou clicando o botão direito do mouse para sair da aplicação. É preciso copiar os novos arquivos de padrões no diretório **bin/Data** antes de usá-los. É preciso também que se edite o arquivo **vrm1_data** para associar os padrões com os objetos virtuais.

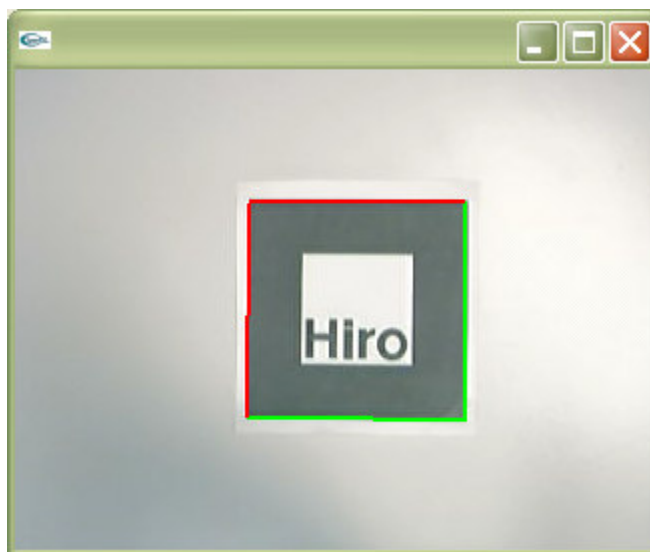


Figura 1-11 – Janela mostrando a cena visualizada no utilitário `mk_patt`.

1.2.9 Outros programas de exemplo

O programa **simpleVRML** mostra como o ARToolKit pode ser usado para desenvolver aplicações que sobreponham imagens virtuais com objetos virtuais. No diretório **bin** há outros programas de exemplos: o **exview** e o **simple**. Estes programas usam os mesmos padrões de marcações da aplicação **simple.exe**.

1.2.9.1 ExView

A aplicação **ExView** (veja a Figura 1-12) permite que você tenha uma visão externa da câmera, como se a câmera estivesse sendo rastreada. Isto é útil para verificar a acurácia do rastreamento. Esta aplicação permite também mostrar como as bibliotecas do ARToolKit podem ser usadas para encontrar uma representação *quaternion* da orientação do ponto de vista. No exemplo **simpleVRML**, o ponto de vista da câmera foi definido usando transformações matriciais. Isto é difícil de se fazer corretamente para aplicações 3D que usem representações por *quaternion*, que é o caso do VRML. O ARToolKit também suporta a saída de *quaternions*, permitindo que rotinas de rastreamento de outros pacotes de renderização sejam também utilizadas.

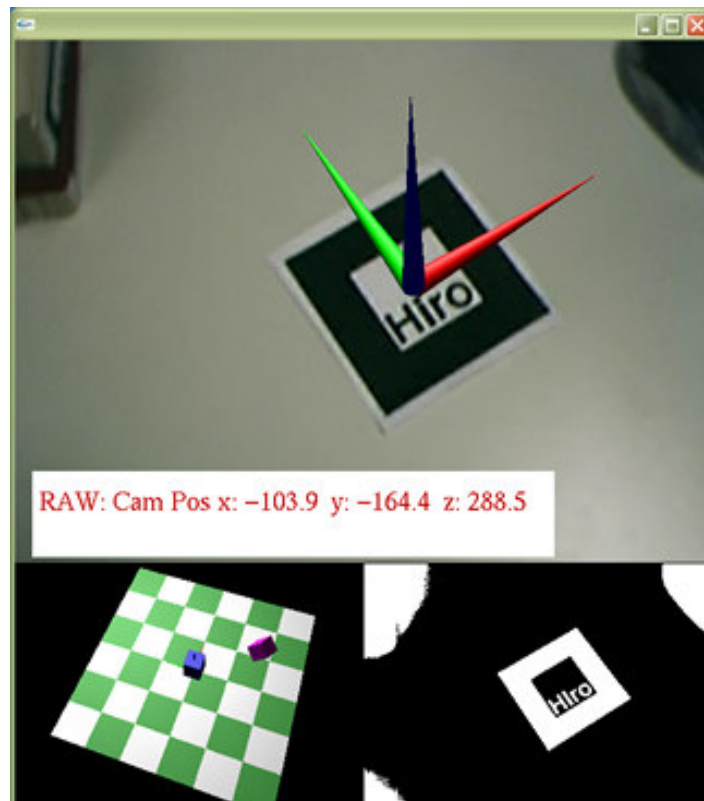


Figura 1-12 – Execução do utilitário ExView.

Esta aplicação mostra também um ponto forte do ARToolKit. Embora o **ExView** disponibilize um mapeamento monocular impreciso, muitas aplicações colaborativas são viáveis com um rastreamento que não exija muita precisão. Por exemplo, interfaces na qual

o usuário esteja imerso e interagindo com ícones exigem apenas que se aponte para os objetos e não uma manipulação direta.

1.2.9.2 Simple

O **simple.exe** é o programa mais simples do ARToolKit. Na verdade ele não interage com o VRML. Isto é útil quando há algum problema do qual se suspeita alguma relação com o grafo de cena VRML. Se o **simple** executar e o **simpleVRML** não, isto é uma indicação de que há algum problema na configuração dos objetos ou no próprio renderizador do grafo de cena VRML. A Figura 1-13 mostra um instantâneo do **simple**.

1.3 Calibração de Câmeras com o ARToolKit

As propriedades *default* da versão atual do ARToolKit estão contidas no arquivo de parâmetro da câmera, **camera_para.dat**, que é lido sempre que a aplicação é iniciada. Os parâmetros com os quais o ARToolKit vem configurado conseguem abranger um amplo conjunto de modelos e fabricantes de câmeras. Contudo, usando uma técnica relativamente simples de calibração de câmera, é possível gerar um outro arquivo de parâmetros para câmeras específicas. Em uma interface de Realidade Aumentada por visão direta (*see-through*), é desejável conhecer os parâmetros da câmera para remover distorções da câmera e posicionar com mais acurácia os objetos virtuais sobre a cena real.

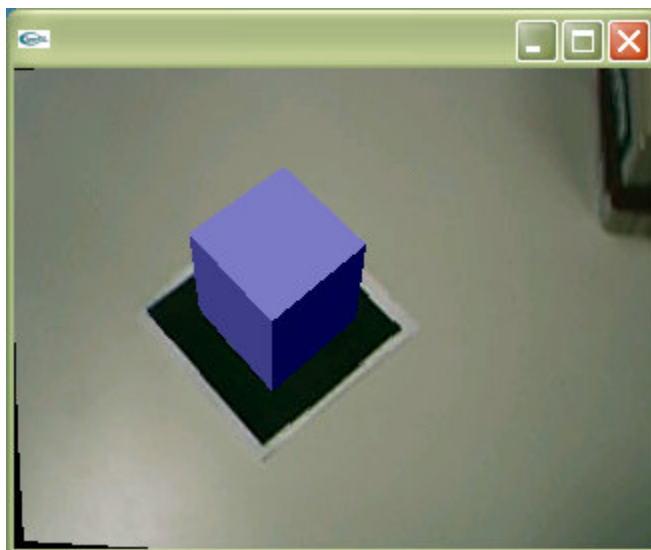


Figura 1-13 – Resultado da execução do exemplo **simple.exe**.

Nesta seção será descrito o uso das rotinas de calibração de câmera do ARToolKit. O início do processo de calibração é a impressão dos arquivos de padrões de calibração **calib_cpara.pdf** e **calib_dist.pdf**. Estes arquivos ficam no diretório **patterns**. O arquivo **calib_cpara.pdf** (veja Figura 1-14a) é uma grade de linhas e deverá ser impresso em escala para que as linhas fiquem separadas de exatamente 40 mm. O arquivo **calib_dist.pdf** (Figura 1-14b) contém um padrão de 6x4 pontos e deverão também ser impressos em escala para que os pontos fiquem separados de 40 mm. Uma vez

que estes arquivos estejam impressos, eles deverão ser colados separadamente em algum material plano e rígido, tais como dois pedaços de papelão. As Figura 14a e 14b mostram estes padrões como vistos pelas lentes das câmeras.

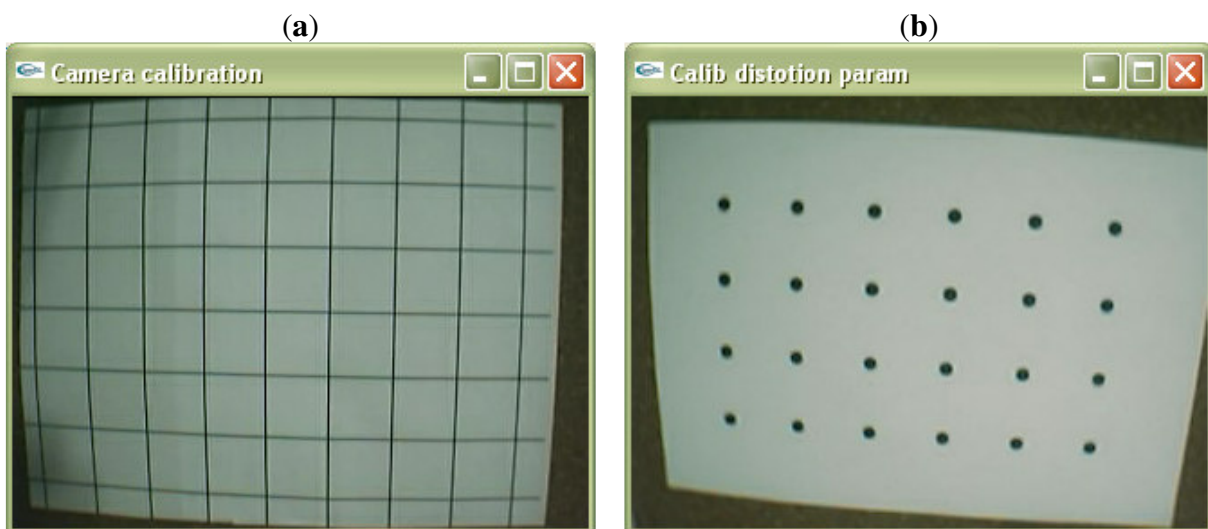


Figura 1-14 – Padrões de calibração (a) `calib_cpara.pdf` e (b) `calib_dist.pdf`.

As principais propriedades de câmera que devem ser extraídas deste processo são o ponto central da imagem da câmera, as distorções da lente e a distância focal da câmera. O programa `calib_dist` é usado para calcular o ponto central da imagem e as distorções das lentes. Já o programa `calib_param` calcula a distância focal da câmera. Os códigos executáveis de ambos os programas podem ser encontrados no diretório `bin` e seus códigos-fonte ficam respectivamente nos diretórios `util/calib_distortion` e `util/calib_cparam`.

O programa `calib_dist` deverá ser executado antes do `calib_cparam`, pois o `calib_cparam` usa o resultado do `calib_dist`. No restante desta seção explicaremos como e o que executa cada um destes programas.

1.3.1 Executando o `calib_dist`

O programa `calib_dist` usa a imagem `calib_dist.pdf` de uma grade de pontos 6x4 igualmente separados. Quando vistos pela lente da câmera, sua distorção causa um efeito chamado *pin-cushion* (do inglês "almofada de alfinetes") que produz um espaçamento irregular na distribuição dos pontos na imagem da câmera (por exemplo, as imagens da Figura 1-14). O programa `calib_dist` mede o espaçamento entre os pontos e usa isto para calcular a distorção das lentes.

Para executar o software, conecte a câmera que você quer calibrar na porta adequada (USB, por exemplo) e então digite `calib_dist` na linha de comando (veja a Figura 15).

```
C:\ Prompt de comando - calib_dist
C:\ARToolkit\ARToolkit2.65vrm1\bin>calib_dist
Image size (x,y) = (320,240)
-----
Mouse Button
Left   : Grab image.
Right  : Quit.
-----
```

Figura 1-15 – Execução do utilitário calib_dist.

Aparecerá uma janela mostrando o vídeo que está sendo adquirido. Aponte a câmera para o padrão de calibração de tal modo que todos os pontos estejam na imagem capturada e então clique com o botão esquerdo do mouse. Isto congelará a imagem de vídeo, como mostrado na imagem capturada da Figura 16. Agora, pressione o botão esquerdo do mouse sobre a imagem e desenhe (segurando o botão do mouse pressionado) um retângulo preto (que aparecerá quando você clicar) em volta de cada ponto da imagem. Comece com o ponto mais próximo ao canto superior esquerdo da imagem e prossiga até que todos os pontos tenham sido desenhados. Os pontos devem ser cobertos na seguinte ordem:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Depois de um retângulo ser desenhado, um software de processamento de imagens encontrará o ponto coberto pelo retângulo e colocará uma cruz vermelha em seu centro. Se não aparecer uma cruz vermelha, redesenhe o retângulo até que o ponto seja encontrado, isto é, até que apareça a cruz vermelha.

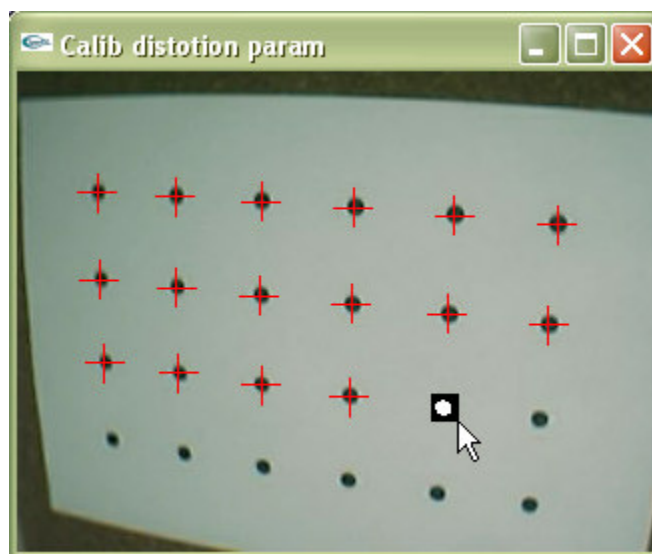


Figura 1-16 – Um retângulo sendo desenhado em torno de um ponto.

Uma vez capturada cada imagem, enquanto cada retângulo estiver sendo desenhado, aparecerá a seguinte saída (veja a Figura 17), mostrando cada ponto capturado.

```
-----  
Mouse Button  
Left   : Rubber-bounding of feature. (6 x 4)  
Right  : Cansel rubber-bounding & Retry grabbing.  
-----  
# 1/24  
# 2/24  
# 3/24  
# 4/24  
# 5/24  
# 6/24  
# 7/24  
# 8/24  
# 9/24  
# 10/24  
# 11/24  
# 12/24  
# 13/24  
# 14/24  
# 15/24  
# 16/24  
# 17/24  
# 18/24  
# 19/24  
# 20/24  
# 21/24  
# 22/24  
# 23/24  
# 24/24
```

Figura 1-17 – Saída de texto enquanto cada ponto estiver sendo desenhado.

Uma vez encontrados todos os 24 pontos da imagem, pressione botão esquerdo do mouse novamente. Isto armazenará a posição dos pontos e descongelará a imagem de vídeo. A saída ilustrada na Figura 1-18 mostra a saída da tela depois disso, isto é, a posição capturada de cada um dos pontos, bem como suas respectivas posições na grade.

```
C:\> Prompt de comando - calib_dist
-----
Mouse Button
Left   : Save feature position.
Right  : Discard & Retry grabbing.
-----
### No.1 ###
1, 1: 51.11, 59.80
2, 1: 91.79, 61.44
3, 1: 134.67, 63.67
4, 1: 179.00, 66.24
5, 1: 223.98, 69.59
6, 1: 268.07, 73.20
1, 2: 51.39, 101.34
2, 2: 91.13, 104.13
3, 2: 133.00, 107.00
4, 2: 176.59, 109.84
5, 2: 220.28, 112.90
6, 2: 263.50, 116.00
1, 3: 52.41, 141.32
2, 3: 91.21, 144.88
3, 3: 132.06, 148.06
4, 3: 174.19, 151.11
5, 3: 216.74, 154.35
6, 3: 258.91, 156.78
1, 4: 54.43, 178.83
2, 4: 92.04, 182.71
3, 4: 131.55, 186.42
4, 4: 172.31, 189.55
5, 4: 213.37, 192.63
6, 4: 253.71, 194.93
```

Figura 1-18 – Saída mostrando as posições capturadas de cada ponto.

Você deverá agora tomar outra imagem e repetir o processo para outras 5 a 10 imagens, em vários ângulos e posições diferentes. Quanto mais imagens tomadas, mais precisa será a calibração. As figuras 19a e 19b mostram exemplos de imagens típicas.

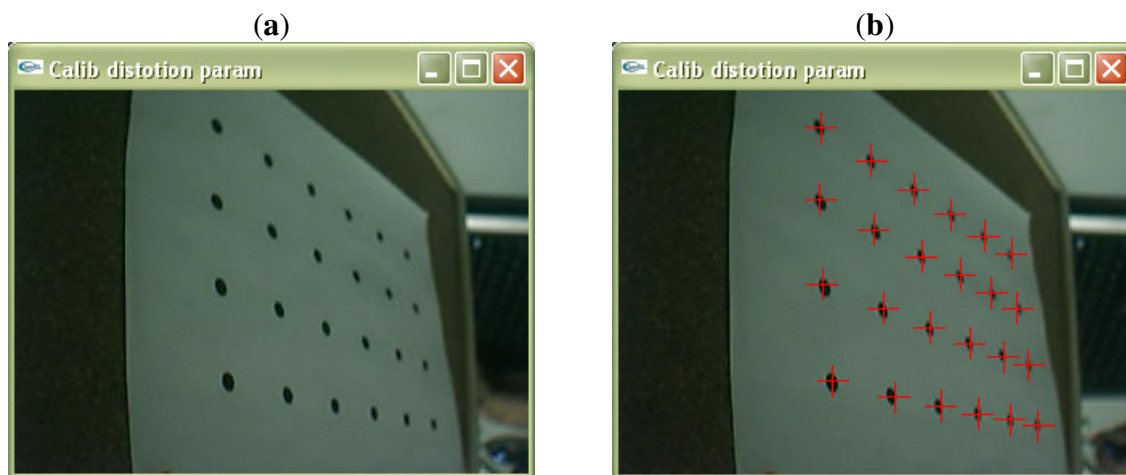


Figura 1-19 – (a) Pose e posição diferentes do padrão de calibração da distorção.
(b) A mesma imagem com os pontos marcados.

Uma vez que você tenha tomado de 5 a 10 imagens, pressione o botão direito do mouse para parar a captura de imagens e começar a calcular os valores de distorção da câmera.

O cálculo destes parâmetros pode demorar alguns minutos, seja paciente. Os valores centrais **x** e **y** e o fator de distorção são os valores finais principais gerados pelo programa **calib_dist**. Estes valores serão diferentes para cada câmera e deverão ser salvos para que se possa usá-los no programa **calib_cparam**. A Figura 1-20 mostra a saída ao término desta etapa de calibração.

```

C:\> Prompt de comando - calib_dist
Mouse Button
Left   : Grab next image.
Right  : Calc parameter.
-----
[160.0, 120.0, 180.8] 78.834831
[160.0, 120.0, 180.8] 78.834831
[210.0,  75.0, 125.8] 77.090859
[210.0,  80.0, 132.1] 74.180695
[210.0,  85.0, 138.5] 71.321889
[210.0,  90.0, 144.8] 68.586200
[210.0,  95.0, 150.9] 66.062559
[210.0, 100.0, 156.6] 63.856720
[210.0, 105.0, 161.7] 62.088244
[205.0, 110.0, 170.5] 60.811336
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.0, 115.0, 174.1] 60.242238
[205.5, 115.0, 173.6] 60.240986
[205.5, 115.5, 173.9] 60.228360
[205.5, 116.0, 174.1] 60.224113
[205.5, 116.0, 174.1] 60.224113
[205.5, 116.0, 174.1] 60.224113
[205.5, 116.0, 174.1] 60.224113
[205.5, 116.0, 174.1] 60.224113
[205.5, 116.0, 174.1] 60.224113
[205.5, 116.0, 174.1] 60.224113
[205.5, 116.0, 174.1] 60.224113
[205.5, 116.0, 174.1] 60.224113
Olen = 205.500000, Ilen = 225.450412
Olen = 114.500000, Ilen = 117.310680
Olen = 116.000000, Ilen = 118.928583
Olen = 124.000000, Ilen = 127.618601
-----
Center X: 205.500000
        Y: 116.000000
Dist Factor: 174.100000
Size Adjust: 1.024547
-----

```

Figura 1-20 – Saída do resultado final da calibração da distorção da câmera.

Para verificar se estes parâmetros estão corretos, pressione o botão esquerdo do mouse novamente. Isto mostrará aquela primeira imagem capturada com as linhas vermelhas desenhadas passando pelos pontos de calibração. Estas linhas deverão se cruzar no centro de cada um destes pontos (veja Figura 21). Cada vez que o botão esquerdo do mouse é pressionado, a próxima imagem capturada é mostrada.

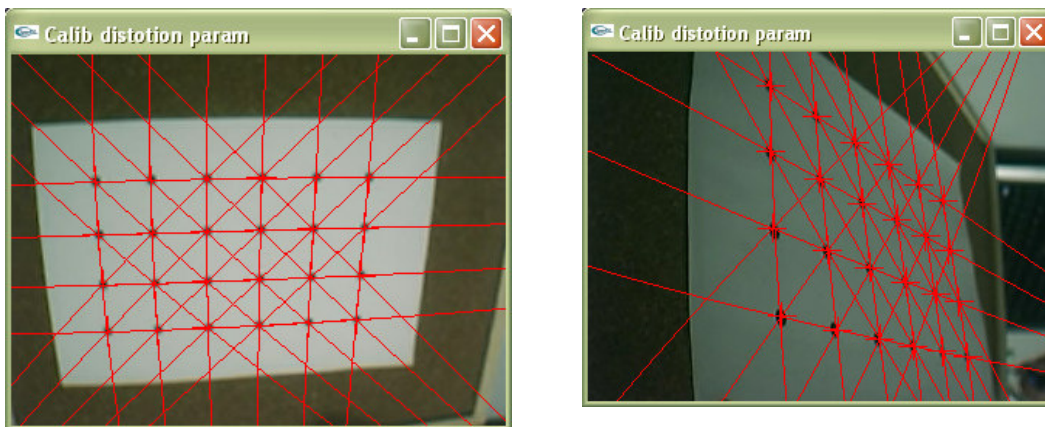


Figura 1-21 – Verificações da aplicação da calibração mostrando linhas passando pelos pontos do padrão de calibração.

Uma vez satisfeito com os resultados do `calib_dist`, clique no botão direito do mouse para sair e executar o código `calib_cparam`.

1.3.2 Executando o `calib_cparam`

O `calib_cparam` é usado para encontrar a distância focal da câmera, além de outros parâmetros. Ele usa o padrão contido no `calib_cparam.pdf`, um padrão de grade de 7 linhas horizontais e 9 linhas verticais. Este padrão deve ser impresso e colado a um pedaço de papelão ou alguma outra placa rígida.

O `calib_cparam` é executado como segue:

1) Digite `calib_cparam` na linha de comando (veja a Figura 22) e entre com as coordenadas do centro e com o fator de distorção encontrado pelo `calib_dist`.

```
C:\> Prompt de comando - calib_cparam
C:\ARToolkit\ARToolkit2.65vrm1\bin>calib_cparam
Input center coordinates: X = 205.5
                        : Y = 116.0
Input distortion ratio: F = 174.1
Input Size Adjustment factor: S = 1.024547
Number of horizontal lines (7):
Number of vertical lines (9):
Number of iteration (5):
Distance among lines (40.000000):
Distance to move (100.000000):
Image size (x,y) = (320,240)
```

Figura 1-22 – Entrada, pela linha de comando, dos parâmetros do utilitário `calib_cparam`.

Aparecerá uma janela de captura de vídeo.

- 2) Coloque o padrão de calibração em frente à câmera de tal modo que: (a) a placa fique perpendicular ao eixo óptico da câmera; (b) todas as linhas da grade sejam visualizadas; e (c) a grade esteja cobrindo o maior espaço possível na imagem.
- 3) Pressione o botão esquerdo do mouse para capturar a imagem. Aparecerá então uma linha branca horizontal na imagem (veja a Figura 23).

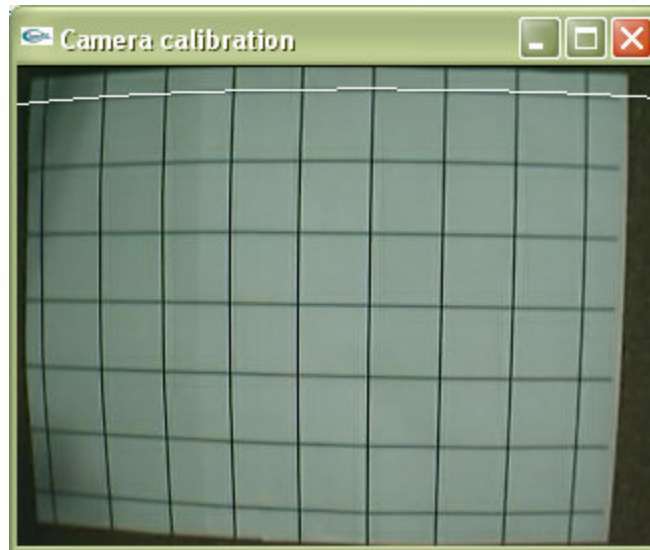


Figura 1-23 – A linha branca aparecendo sobre a imagem do padrão de calibração

- 4) Mova a linha branca de modo a cobrir o máximo possível a linha preta do topo da grade. A linha pode se movimentar para cima ou para baixo usando-se as teclas de seta para cima e para baixo. A linha pode ainda ser rotacionada no sentido horário e anti-horário usando as teclas de setas para a direita e para a esquerda. Uma vez que a linha branca esteja alinhada com a linha do topo da grade, pressione a tecla *enter*. Esta linha se tornará azul e então aparecerá outra linha branca (ver Figura 24). Este processo deverá ser repetido para todas as linhas horizontais.



Figura 1-24 – A linha se torna azul e surge uma outra linha branca.

Uma vez que a última linha horizontal tenha sido ajustada, uma linha branca vertical aparecerá e o processo deverá ser repetido para as linhas verticais. A primeira linha branca vertical deverá ser ajustada sobre a linha da grade mais à esquerda e as outras linhas se ajustarão da esquerda para a direita (ver Figura 1-25).

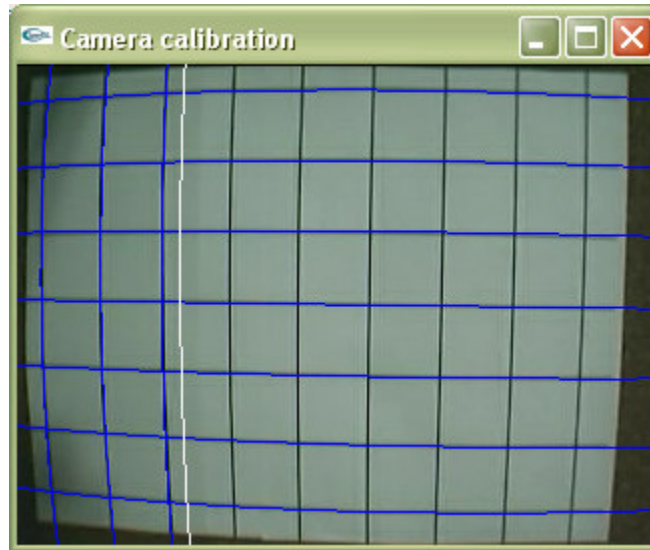


Figura 1-25 – Ajuste das linhas verticais.

A ordem das linhas é muito importante. Elas deverão ser ajustadas de cima para baixo e da esquerda para a direita até que todas as 16 linhas tenham sido desenhadas na tela.

5) Uma vez que o processo tenha terminado para uma imagem, o padrão de grade deverá ficar mais distante 100mm da câmera (mantendo a câmera perpendicular ao padrão) e o processo deve ser repetido novamente. A Figura 26 ilustra a última linha ajustada para uma das imagens capturadas.

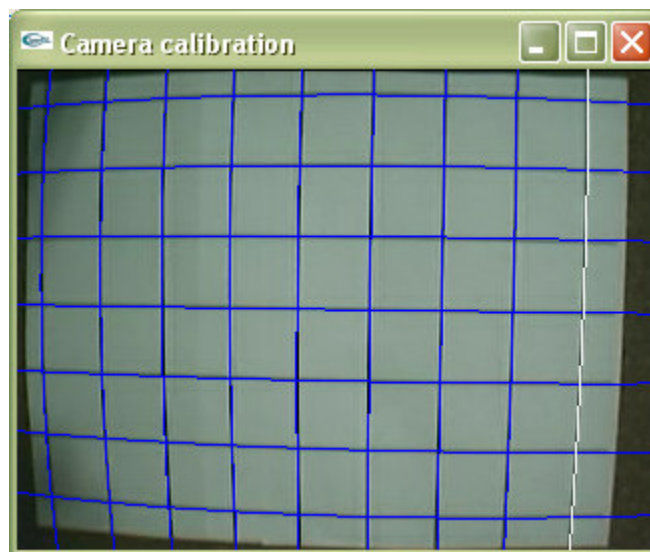


Figura 1-26 – O ajuste da última linha branca vertical.

6) Repita o processo cinco vezes, afastando o padrão de calibração por um total de 500mm da câmera. Depois da quinta etapa de calibração, o programa automaticamente calculará os parâmetros da câmera. Será pedido que você entre com um nome de arquivo para armazenar estes parâmetros nele (veja Figura 1-27).

Uma vez que os valores estejam armazenados em arquivo de dados da câmera, pressione o qualquer botão do mouse para sair.

```
C:\> Prompt de comando
C:\ARToolkit\ARToolkit2.65vrm1\bin>calib_cparam
Input center coordinates: X = 205.5
                        : Y = 116.0
Input distotion retio: F = 174.1
Input Size Adjustment factor: S = 1.024547
Number of horizontal lines (?):
Number of vertical lines (9):
Number of iteration (5):
Distance among lines (40.000000):
Distance to move (100.000000):
Image size (x,y) = (320,240)
point_num = 315
-----
SIZE = 320, 240
Distotion factor = 205.500000 116.000000 174.100000 1.024547
404.95948 -0.51559 110.38180 0.00000
0.00000 404.53953 69.29708 0.00000
0.00000 0.00000 1.00000 0.00000
-----
Input filename: webcam3_para.dat
C:\ARToolkit\ARToolkit2.65vrm1\bin>
```

Figura 1-27 – Resultado final do utilitário `calib_cparam`.

Mudando o nome deste arquivo de dados da câmera para `camera_para.dat` e colocando-o no diretório `bin/Data` ele já pode ser usado nos programas exemplos do ARToolKit. A calibração da sua câmera deverá produzir resultados de rastreamento bem melhores que o da configuração padrão de parâmetros.

A distância padrão entre as linhas da grade no arquivo `calib_cparam.pdf` é exatamente 40 mm, enquanto o padrão deve ser afastado da câmera em 100mm para cada mensuração, que deve ser repetida 5 vezes. Estes valores são todos determinados no código fonte `calib_cparam.c` no diretório `util/calib_cparam` (veja Código 20).

Se você quiser modificar a distância entre as linhas da grade, o trecho de código fonte abaixo deve ser modificado.

```
inter_coord[k][j][i+7][0] = 40.0*i;
inter_coord[k][j][i+7][1] = 40.0*j;
inter_coord[k][j][i+7][2] = 100.0*k;
```

Código 20 – Trecho do programa `calib_cparam.c` para alterar o distanciamento dos padrões de calibração.

Neste código, 40 é a distância entre as grades e entre os pontos, e 100.0 é a distância padrão em que devem ser afastados os padrões a partir da câmera, a cada vez. O número de medidas que devem ser tomadas pode ser alterado na variável `loop_num` (veja Código 21).

```
*loop_num = 5;
```

Código 21 – Variável que deve ser alterada para mudar o número de distâncias do padrão de calibração até a câmera.

1.4 Bibliotecas e Funções do ARToolKit

Esta seção fornece uma lista parcial das funções fornecidas pelo ARToolKit. Neste documento não são listadas as funções de calibração da câmera.

A biblioteca ARToolKit consiste de 3 pacotes encontrados no diretório **lib**:

- **libAR.lib**: biblioteca para rastreamento de marcadores, calibração e entrada de parâmetros. Contém as funções de suporte para a detecção dos marcadores.
- **libARvideo.lib**: biblioteca para capturar quadros da entrada de vídeo. Estas bibliotecas de vídeo variam a cada versão e, dependendo do suporte oferecido, disponibiliza bibliotecas para Windows WDM, Linux Video 4 Linux, para câmeras Firewire ou então para placas de aquisição. É a parte que mais têm se alterado no surgimento de novas versões.
- **libARgsub.lib**: contém rotinas gráficas, baseadas nas bibliotecas OpenGL e Glut, para mapeamento do vídeo como textura em um ambiente 3D, bem como para o posicionamento dos objetos virtuais sobre o marcador.

A figura 28 ilustra a estrutura hierárquica das bibliotecas. A exceção a esta estrutura é a função **argCalibHMD** no **libARgsub.lib** que usa tanto o **libAR.lib** quanto **libARvideo.lib**.

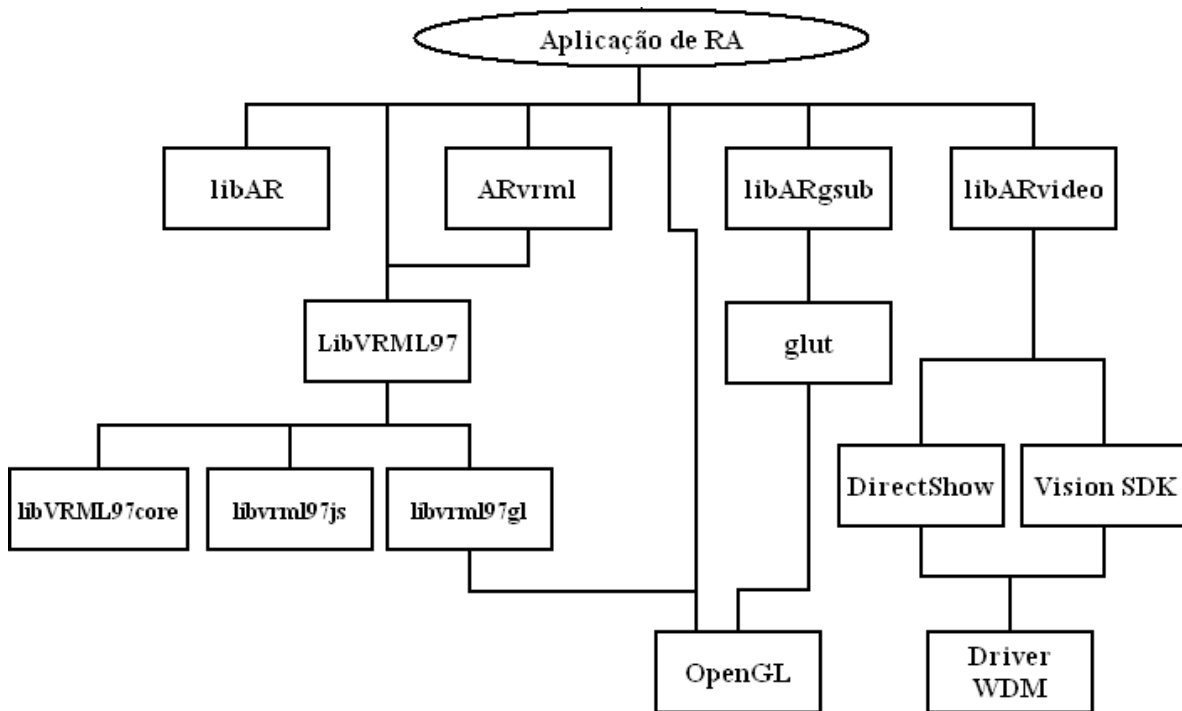


Figura 1-28 – Estrutura de bibliotecas do ARToolKit

A partir da versão 2.43 foram incluídas as bibliotecas de grafo de cena VRML, LibVRML97. Esta biblioteca de grafo de cena inclui outras 4 bibliotecas:

- **libARvrm1.lib**: esta biblioteca implementa um visualizador VRML sobre o OpenGL. Na verdade ela implementa um visualizador utilizando as funções da biblioteca **LibVRML97** que atuam diretamente sobre o OpenGL. Estende ao ARToolKit a classe **OpenGLViewer** fornecida pela biblioteca **libvrm197gl**.
- **libvrm197core.lib**: é a implementação das funções VRML97. Contém todos os comandos para gerar uma cena a partir de um código VRML e, claro, o seu analisador léxico e sintático (*parser*).
- **libvrm197js.lib**: é a implementação do *script* (**javascript**) versão 1.1 do *engine* Mozilla. Estes *scripts* permitem gerar animações 3D, porém nem todos os aspectos são considerados.
- **libvrm197gl.lib**: é uma implementação de uma classe **OpenGLViewer** que renderiza as cenas VRML sobre uma janela OpenGL.

Essas bibliotecas são, na verdade, implementações agrupadas pela LibVRML97. Uma discussão detalhada sobre suas funções não está no escopo deste tutorial.

1.4.1 Estruturas de Dados Básicas

As informações sobre a detecção de marcadores ficam contidas na estrutura **ARMarkerInfo** definida no arquivo de protótipos **ar.h** no diretório **include**. Na estrutura do Código 22, **id** é um número de identificação do marcador, **cf** é um valor de confiança de que o marcador foi identificado corretamente, **pos [2]** é o centro do marcador em um sistema de coordenadas de tela ideal, **line [4] [3]** é a equação das retas para os quatro lados dos marcadores em um sistema de coordenadas de tela ideal e **vertex [4] [2]** são as coordenadas dos vértices no sistema de coordenadas de tela. Para as equações das retas em **line [4] [3]** os valores **line [X] [0]**, **line [X] [1]**, e **line [X] [2]** são os valores **a**, **b**, **c** da equação da reta **ax+by+c=0**.

```
typedef struct {
    int area;
    int id;
    int dir;
    double cf;
    double pos[2];
    double line[4][3];
    double vertex[4][2];
} ARMarkerInfo;
```

Código 22 – Estrutura de descrição de marcadores.

1.4.2 Funções de Realidade Aumentada

As funções comumente usadas em aplicações de realidade aumentada estão armazenadas na biblioteca **libAR.lib**. Há funções para:

- carga inicial dos padrões e dos padrões treinados:

```
int arInitCparam( ARParam *param);
int arLoadPatt( char *filename );
```

- detectar marcadores e posições da câmera:

```
int arDetectMarker( ARUint8 *dataPtr, int thresh,
    ARMarkerInfo **marker_info,
    int *marker_num );
int arDetectMarkerLite( ARUint8 *dataPtr, int thresh,
    ARMarkerInfo **marker_info,
    int *marker_num);
int arGetTransMat( ARMarkerInfo *marker_info,
    double pos3d[4][2],
    double trans[3][4] );
int arSavePatt( ARUint8 *image,
    ARMarkerInfo *marker_info,
    char *filename);
```

A seguir estas funções são descritas com um pouco mais de detalhe.

1.4.2.1 arInitCparam

1.4.2.1.1 Função: Iniciar os parâmetros especificados na estrutura de parâmetros da câmera. O argumento **param* aponta para uma memória estática na

biblioteca AR. Estes parâmetros de câmera são normalmente lidos pelo programa de inicialização a partir de um arquivo de dados. Em aplicações de RA por visão direta, parâmetros default de câmera são suficientes, ou seja, não é preciso qualquer calibração de câmera.

1.4.2.1.2 Variável: **param* – a estrutura do parâmetro de câmera.

1.4.2.1.3 Retorno: *sempre zero.*

1.4.2.2 arLoadPatt

1.4.2.2.1 Função: *Carregar o mapa de bits do padrão especificado no arquivo em filename, no vetor de padrões para uso posterior pelas rotinas de detecção de marcadores.*

1.4.2.2.2 Variável: **filename* – nome do arquivo que contém o mapa de bits a ser carregado

1.4.2.2.3 Retorno: *o número do padrão carregado ou -1 em caso de falha.*

1.4.2.3 arDetectMarker e arDetectMarkerLite

1.4.2.3.1 Função: *detectar os marcadores quadrados no frame de entrada de vídeo.*

1.4.2.3.2 Variáveis: *ARUint8 *dataPtr* – um ponteiro para imagem colorida que é para ser pesquisada para marcadores quadrados. O formato do pixel é ABGR, mas as imagens são tratadas em níveis de cinza. Portanto, a ordem dos componentes BGR não importa. Entretanto, a ordem do componente alfa, A, é importante.

1.4.2.3.3 *int thresh* – especifica o valor de limiar (entre 0 e 255) a ser usado para converter a imagem de entrada em uma imagem binária.

ARMarkerInfo **marker_info – um ponteiro retornado para uma matriz de estruturas ARMarkerInfo, que contém todas as informações sobre os quadrados detectados na imagem.

`int *square_num` – número de marcadores detectados na imagem.

1.4.2.3.4 Retorno: *zero quando a função completa normalmente, -1 caso contrário.*

A função `arDetectMarkerLite(...)` é uma versão mais simples da `arDetectMarker`, mas que não chama as funções de correção de erro e, portanto, executa um pouco mais rápido, mas com mais erros de detecção.

1.4.2.4 `arGetTransMat`

1.4.2.4.1 Função: *calcular a transformação entre um marcador detectado e a câmera real, isto é, a posição e orientação da câmera relativas à marcação de rastreamento.*

1.4.2.4.2 Variáveis: `ARMarkerInfo *marker_info` – a estrutura que contém os parâmetros para o marcador para o qual a posição e orientação da câmera será relativa. Essa estrutura é encontrada usando `arDetectMarker`.

`double pos3d[4][2]` – posições físicas dos vértices do marcador.

`arGetTransMat` assume que o marcador está no plano **xy**, e que o eixo **z** está apontando para baixo do plano do marcador. Deste modo, as posições dos vértices podem ser representadas em coordenadas 2D, ignorando a informação do eixo **z**. A ordem das posições dos vértices são especificadas no sentido horário. Por exemplo, para um marcador com lado medindo 50 mm e com a origem das coordenadas do marcador no seu centro:

```
pos3d[4][2] = { { -25.0, -25.0 }, { 25.0, -25.0 },  
               { 25.0, 25.0 }, { -25.0, 25.0 } };
```

`pos3d[4][2]` é especificado no arquivo do parâmetro `vrml_data` e lido na inicialização do programa.

`double conv[3][4]` – é a matriz de transformação entre as coordenadas do marcador e as coordenadas do frame da câmera, que é a posição relativa da câmera real em relação ao marcador real.

1.4.2.4.3 Retorno: *Sempre zero.*

1.4.2.5 `arSavePatt`

1.4.2.5.1 Função: *usado no `mk_patt` para gravar um mapa de bits do padrão do marcador que está sendo detectado.*

1.4.2.5.2 Variáveis: `ARUint8 *image` – *ponteiro para a imagem contendo o padrão do marcador a ser treinado.*

`ARMarkerInfo *marker_info` – um ponteiro para a estrutura `ARMarkerInfo` do padrão a ser treinado.

`char *filename` – o nome do arquivo em que a imagem do mapa de bits será gravada.

1.4.2.5.3 Retorno: *zero – se a imagem foi salva com sucesso, -1 caso contrário*

1.4.3 Funções de Aquisição de Vídeo

As seguintes funções da biblioteca `libARvideo.lib` são mais comumente usadas:

```
int arVideoOpen( void );
int arVideoClose( void );

int arVideoInqSize( int *x, int *y);
unsigned char *arVideoGetImage( void );
```

As funções de vídeo são definidas abaixo com mais detalhe.

1.4.3.1 arVideoOpen

1.4.3.1.1 Função: *abre o driver de vídeo para tentar encontrar todos os dispositivos de vídeo registrados no Windows*

1.4.3.1.2 Variáveis: *nenhuma*

1.4.3.1.3 Retorno: *zero – sucesso, -1 se o driver de vídeo não puder ser aberto.*

1.4.3.2 arVideoClose

1.4.3.2.1 Função: *é uma função que precisa ser chamada para finalizar a captura de vídeo.*

1.4.3.2.2 Variáveis: *nenhuma*

1.4.3.2.3 Retorno: *zero – se finalizou com sucesso, -1 caso contrário.*

1.4.3.3 arVideoInqSize

1.4.3.3.1 Função: *retorna tamanho do quadro de vídeo capturado.*

1.4.3.3.2 Variáveis: `int *x, *y` – *ponteiros para largura (*x) e altura (*y) da imagem capturada.*

1.4.3.3 Retorno: zero – se as dimensões foram encontradas com sucesso, -1 caso contrário

1.4.3.4 arVideoGetImage

1.4.3.4.1 Função: capturar um único quadro de vídeo

1.4.3.4.2 Variáveis: nenhuma

1.4.3.4.3 Retorno: retorna um ponteiro para o quadro de vídeo capturado ou *NULL* se um quadro não puder ser capturado.

1.5 Limitações da RA baseada em Visão Computacional

Há algumas limitações de sistemas de Realidade Aumentada baseados apenas em visão computacional. Os objetos virtuais só podem ser exibidos quando as marcas estiverem sendo rastreadas. Esta dependência do rastreamento limita o tamanho dos objetos virtuais e seus movimentos. Portanto, se os usuários cobrirem parte do padrão com suas mãos ou outros objetos, o objeto virtual desaparecerá. Além disso, há limites na inclinação dos marcadores. Se o usuário inclinar muito o marcador em relação ao eixo óptico da câmera, o padrão inscrito no marcador pode se tornar irreconhecível pelo programa, impedindo-o de tomar a decisão do padrão que deve ser exibido.

Há também limitações de espaço. Quanto maior for o tamanho físico do padrão, maior será a distância da qual você poderá detectá-lo e, portanto, maior será o volume do espaço de interação no qual o usuário poderá ser rastreado. Contudo, aumentar o tamanho do padrão não resolve sempre, pois pode haver alguma dificuldade em seu enquadramento pela câmera se houver a necessidade de aproximação da câmera. A Tabela 1 mostra alguns limites máximos típicos para marcadores quadrados de diferentes tamanhos de um experimento relatado em [KATO, BILLINGHURST & POUPLYREV 2000]. Estes resultados foram obtidos a partir de padrões de marcação de diferentes tamanhos, colocando-os perpendicularmente à câmera e movendo a câmera para trás até que o objeto virtual desapareça.

Tabela 1 – Resultado de experimento comparando o tamanho dos padrões de marcação com o espaço útil de interação.

Tamanho do Padrão (em cm)	Espaço de Interação Útil (em cm)
6,985	40,64
8,89	63,5
10,795	86,36
18,72	127

Esta limitação de espaço também é afetada de alguma maneira por padrões cheios de detalhes. Quanto mais simples o padrão, melhor. Padrões com grandes regiões pretas e brancas (isto é, padrões de baixa frequência) são mais eficazes. Para se ter uma idéia, neste experimento, o padrão quadrado de 10,795 centímetros, usado no resultado da tabela acima,

foi substituído por um padrão do mesmo tamanho, mas com mais detalhes. Esta simples troca reduziu o espaço de rastreamento de 86,36 para 38,1 centímetros.

O rastreamento também é afetado pela orientação dos marcadores em relação à câmera. Conforme os marcadores se inclinam, menos os padrões do centro serão visíveis e o reconhecimento torna-se menos confiável. Padrões confeccionados sobre superfícies maleáveis (como o papel sulfite comum), que se curvam facilmente também não são adequados, tornando-se sujeitos a indecisões de reconhecimento.

Os resultados do rastreamento são afetados também por condições de iluminação. A saturação luminosa pode criar reflexões e regiões brilhantes nas marcas do papel tornando mais difícil a tarefa de encontrar as marcas quadradas. Para reduzir o brilho, os padrões podem ser confeccionados com material não reflexivo. Por exemplo, os marcadores podem ser confeccionados com papel camurça, encontrado em papelarias.

Pode-se perceber, a partir destas limitações, que o ambiente no qual a aplicação de RA ocorrerá assim como as interações entre os objetos, exigem que as aplicações sejam planejadas para que se possa determinar os tamanhos adequados dos marcadores, suas possíveis inclinações, ou ainda a iluminação do ambiente. Marcadores que se movimentam ao alcance das mãos dos usuários devem ter um tamanho menor que aqueles que são utilizados apenas para visualizados a uma certa distância, como por exemplo, uma decoração virtual em um apartamento real. É fácil notar que marcadores estacionários são menos suscetíveis a problemas de reconhecimento que aqueles que se movimentam muito. Uma taxa de quadros deve dar ao usuário a ilusão de uma interação real. Atualizações de quadro com menos que 20 fps podem comprometer a fidelidade da aplicação, principalmente se esta aplicação envolver a manipulação de objetos virtuais. Outros experimentos explicitando estas limitações podem ser encontrados em [DAVIS, CLARKSON & ROLLAND 2003; MALBEZIN, PIEKARSKI & THOMAS 2002; OWEN, XIAO & MIDDLELIN 2002].

1.6 Aplicações Educacionais em Ambientes de Realidade

Aumentada com ARToolkit

1.6.1 Introdução

Os sistemas que permitem maior interatividade entre o homem e a máquina, tornam-se cada vez mais utilizados em todos os setores, pois enriquece a relação homem-máquina e possibilita um controle maior do usuário frente às tarefas.

Nesse sentido, os sistemas virtuais possibilitam experiências com a sensação de presença, através da integração dinâmica de diferentes modalidades perceptivas, que envolvem imagens, sons, tato, etc. Assim, torna-se possível a capacidade de manipular, relacionada às reações sensorio-motora em tempo real [Lévy, 1999].

Por outro lado, os ambientes educativos devem oferecer condições favoráveis à criação, comportando-se como um espaço agradável e permitindo aplicações práticas e a relação do conhecimento com experiências, necessidades e realidade do aluno (usuário).

De maneira geral, a construção do conhecimento dá-se através da reflexão, da crítica, da identificação e da busca de resoluções dos problemas, propiciando situações que determinem o desafio - papel importante na formação de atitudes [Valente. 1991, 2001].

Os ambientes podem contribuir, estimulando a curiosidade e auxiliando no desenvolvimento da autonomia.

A aprendizagem ocorre, quando o indivíduo está engajado e utiliza de forma consciente estratégias de resolução de problemas para a construção significativa. Não se deve questionar o valor da instrução, mas é importante a descoberta de novos conhecimentos, através da relação do novo com a experiência anterior.

Assim, a possibilidade de interação entre objetos reais e virtuais, que ocorre através da Realidade Aumentada (RA), pode oferecer ao usuário maiores informações sensitivas, facilitando a associação e a reflexão sobre a situação. Os sistemas de Realidade Aumentada permitem que o usuário decida sobre os ambientes, compondo cenas com imagens de objetos tridimensionais geradas por computador misturadas com imagens reais, aumentando as informações do cenário e oferecendo condições para a imersão no ambiente criado. A principal característica destes ambientes é que as informações do mundo real são utilizadas para criar um cenário incrementado com elementos gerados por computador [Dainese, 2003].

1.6.2 Intervenções Educacionais com Realidade Aumentada

Três características são responsáveis por tornar as situações de intervenção educacionais interessantes: *curiosidade, fantasia e desafio*. Através dos ambientes de realidade aumentada, é possível proporcionar ao usuário (aluno) situações lúdicas, tornando as atividades mais motivadoras.

Deve-se destacar a importância das relações sociais para o aluno garantir seu envolvimento com situações novas, considerando aquelas vividas anteriormente. Assim, ele poderá construir o novo, através do fazer, motivado pelo envolvimento afetivo. O ambiente deve ser favorável ao interesse do usuário, além de ser um ambiente contextualizado e significativo. Os problemas emergem no ambiente e o usuário com autonomia deve decidir resolvê-lo. O professor (usuário) deve ter preparo para utilizar a tecnologia e aproveitar os recursos que as ferramentas podem oferecer, de forma a garantir flexibilidade intelectual, capacidade de criar, inovar e, principalmente, enfrentar o desconhecido para promover o desenvolvimento cognitivo.

Em ambientes de Realidade Aumentada, o mundo real é “aumentado” com informações que não estão presentes na cena capturada, e o usuário passa ser um elemento participativo no cenário em que imagens reais são misturadas com virtuais para criar uma percepção aumentada. [Azuma, 2001]

A interface deve ser entendida como um espaço de comunicação, um sistema semiótico, onde signos são usados para interação, possibilitando o acesso ao ambiente [Garbin, 2004]. Para garantir uma boa usabilidade, os fatores humanos devem ser respeitados. Isso remete à questão da diversidade dos usuários, suas características cognitivas, de personalidade, cultura, idade, comportamento e habilidades e necessidades especiais [Baranauskas, 2003]. Os estudos sobre a memória humana (principalmente a de curta duração), vêm oferecendo subsídios para soluções inteligentes sobre a interface, cuja idéia central é liberar o usuário da memorização de comandos para tornar o ambiente mais agradável e natural através de interfaces gráficas.

Um ambiente educativo deve ser atrativo e interessante, oferecendo, através de situações lúdicas e espontâneas, atividades que proporcionem o desenvolvimento cognitivo. A interface deve ser planejada para oferecer flexibilidade ao usuário e, para facilitar a

aprendizagem, o sistema não deve ser linear-fechado, onde apenas uma resposta é correta frente a um tipo de estímulo apresentado. A aceitação de uma interface depende da capacidade de sua linguagem de interação em comunicar suas funções com clareza. Assim, no desenvolvimento de interfaces de realidade aumentada como mediador pedagógico, a questão de qualidade deve ter como objetivo a intenção do usuário e, principalmente, a usabilidade, possibilitando criar o novo a partir das experiências vividas. Os requisitos de ambientes de realidade aumentada para satisfazer as necessidades educativas, enquanto mediador pedagógico para sistemas complexos, são:

- Oferecer flexibilidade, em função do ambiente;
- Exibir uma conduta adaptativa;
- Operar em tempo real;
- Oferecer a possibilidade de interação entre o real e virtual;
- Operar através de interação direta com linguagem natural;
- Oferecer um ambiente complexo, e aberto para:
 - Identificação de grande quantidade de dados;
 - Identificação de reações perceptuais e motoras com muitos graus de liberdade.Nos ambientes de realidade aumentada, espera-se que o usuário possa:

- Utilizar símbolos e abstrações;
- Utilizar linguagem natural;
- Realizar ações que compõem, alteram ou criam novas situações;
- Interagir com objetos virtuais em tempo real;
- Relacionar ou compor cenas e ambientes com a integração entre o real e o virtual;
- Realizar ações autônomas no ambiente conduzidas pelo desejo e imaginação.

1.6.3 Sistema de Realidade Aumentada

Um sistema de Realidade Aumentada típico é formado de uma ou mais câmeras, software para construção de objetos virtuais, sistema gráfico e dispositivo de interação para as tarefas de: a) captura da cena real, b) criação de objetos virtuais, c) sobreposição dos objetos reais e virtuais no mesmo cenário, d) rastreamento para posicionamento e orientação espacial do usuário e, e) interação em tempo real.

O processo de criar um ambiente de Realidade Aumentada consiste em obter imagens reais, via câmera, e misturá-las com objetos virtuais criados por computador dentro do mesmo ambiente. Uma tarefa importante é extrair informações para instruir o sistema gráfico no processo de formação de um ambiente, a partir do ponto de vista do usuário. Uma das formas para efetuar esta operação é utilizar marcadores que permitem, ao sistema gráfico, definir coordenadas espaciais e orientação dos objetos, a partir do ponto de vista do usuário, além de identificar alterações de posicionamento e interação do usuário com os objetos. Após alguns experimentos, desenvolveu-se aplicações como: quebra-cabeça; livro interativo, pá de transporte de objetos virtuais e aprendizagem de mecânica quântica com o Orbitário. Estas aplicações tiveram como base brinquedos e objetos reais, utilizados em

situações de ensino, com o objetivo de oferecer ao usuário a possibilidade de interagir e visualizar novas situações.

Nesse contexto, a Realidade Aumentada [Azuma, 1997] enriquece o ambiente real com objetos virtuais, com base em tecnologias que permitem misturar o cenário real com virtuais. Isto pode ser feito, através do uso de WebCam que permite a captura do cenário real e o rastreamento das mãos ou de algum dispositivo auxiliar (placa com marcador). Esse ambiente permite fornecer maior realismo às aplicações e o uso das mãos para manipulação dos objetos, minimizando os inconvenientes da Realidade Virtual.

Com esse fator motivacional, a Realidade Aumentada possibilita implementar ambientes que permitam a aplicação de estratégias diferenciadas para o desenvolvimento cognitivo. O trabalho consistiu de duas etapas: a) desenvolvimento tecnológico e b) aplicações educacionais.

Uma vez que o ARToolKit é um software aberto, o desenvolvimento tecnológico consistiu na alteração de seu código de programação para dar funcionalidades específicas não contempladas no software original, mas necessárias para o desenvolvimento de aplicações diferenciadas. Nesse sentido, foram introduzidos trechos de programas que permitiram a introdução de som (Santin, 2004), controle de posicionamento (Lahr, 2004), controle de sequência de objetos e transporte de objetos (Santin, 2004).

As aplicações baseadas em ARToolKit dependem de três elementos: o cenário real, objeto virtual e uma placa quadrada (marcador) com uma moldura e um identificar desenhado dentro dela. Ligando-se o sistema, a imagem capturada pela WebCam aparece na tela do monitor. Ao introduzir o marcador no campo de visão da câmera (usando as mãos), o software ARToolKit posiciona o objeto virtual sobre o marcador no cenário real, misturando as imagens. Ao movimentar-se o marcador, o objeto virtual acompanha este movimento, permitindo sua manipulação com as mãos.

Com o uso de som [Santin, 2004], associa-se, a cada objeto, um som correspondente (ruídos, locuções, etc) que é disparado quando a placa entra no campo de visão da WebCam, além do aparecimento do objeto virtual sobre a placa. Como exemplo, pode-se citar a visualização de um carro virtual sobre a placa acompanhada do som do motor.

Com o uso de uma placa adicional de controle [Santin, 2004], foi possível alterar o objeto virtual associado a uma placa, trocando suas características, ou mesmo, a troca do objeto. Como exemplo, pode-se citar a presença de um carro em uma placa, que muda de cor, ou de modelo, com cada introdução da placa de controle. O som associado também pode ser alterado por esta placa.

Através do posicionamento espacial, foi possível registrar a trajetória dos objetos, capturando-se cada posicionamento e desenhando-se o deslocamento da trajetória [Utiyama, 2004]. Como exemplo, a movimentação do carro no campo de visão da câmera, pode ser visualizada.

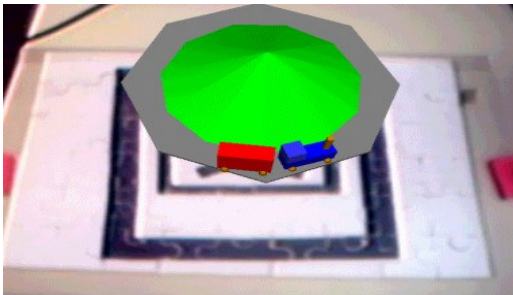
No transporte de objetos [Santin, 2004] [Lahr, 2004], uma placa de controle, ao aproximar-se da placa com objeto virtual, permitiu a movimentação do objeto de uma placa para outra, ou a cópia de objetos entre placas. Com isto, foi elaborada uma ferramenta de autoria [Galana, 2004], que propiciou a montagem de cenários de Realidade Aumentada com uso das mãos pelo usuário.

1.6.4 Quebra-cabeças

O quebra-cabeça, com Realidade Aumentada consistiu em conjunto de peças que formam uma imagem de um lado e no seu verso um marcador. Ao montar corretamente o quebra-cabeça, a pessoa visualiza a imagem final na parte da frente. Virando-o e expondo o marcador, montado no seu verso, à câmera, aparece o objeto virtual correspondente. Como exemplo, foi utilizado o quebra-cabeça de um trem (Figuras 1 e 2), que, na frente, mostra a figura de um trem e, sobre o marcador no verso, um trem virtual em movimento



Figura 1-29 – Parte superior do quebra-cabeças



1.6.5 Livro Interativo com Realidade Aumentada

Outra aplicação foi o desenvolvimento do Livro Interativo de Realidade Aumentada (LIRA) (Akagui, 2004) que consistiu na montagem de um livro explicativo sobre poliedros. Cada folha do livro apresentou um marcador de forma que, ao expô-lo no campo de visão da câmera, apareceu o poliedro virtual (Figuras 3 e 4) em 3D sobre o livro.

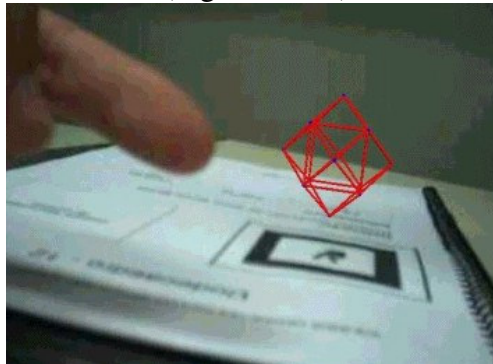


Figura 1-30 - Poliedro virtual

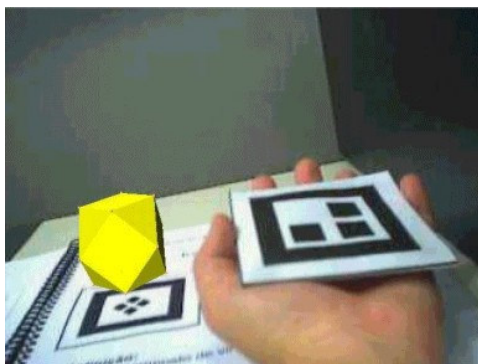
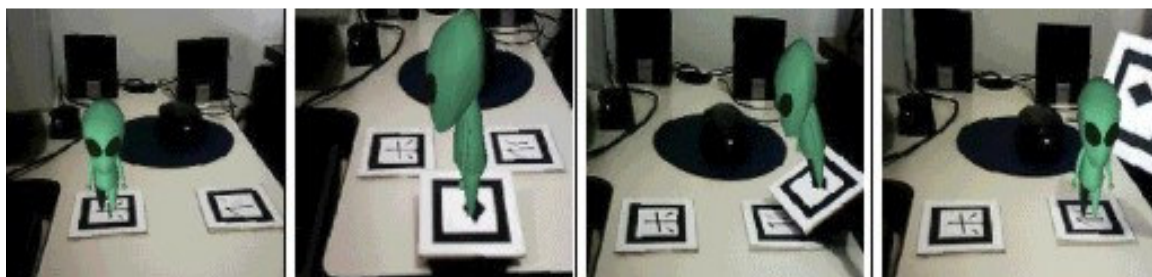


Figura 1-31 - Alteração do poliedro

O uso do marcador permitiu uma variação da visualização do poliedro, alterando suas cores, mostrando sua versão amarrada e colocando-o para girar de forma a permitir a visualização de vários pontos de vista.

1.6.6 Pá de transporte de objetos virtuais

A Figura 5 mostra uma sequência de transporte de um objeto de uma placa para outra, usando uma pá [Santin, 2004] [Lahr, 2004], implementada com o software ARToolKit, permitindo assim a obtenção de um mecanismo de autoria de ambientes virtuais sobre uma mesa ou algo equivalente.



1.6.7 Aprendizagem de mecânica quântica com realidade aumentada:

O Orbitário

A Mecânica Quântica, em conjunto com a teoria da Relatividade, é a pedra angular da Física do século XX e influenciou alguns dos mais recentes e estupendos avanços científicos que ocorreram em áreas como a Biologia, a Química e a Física. Sendo a base de sustentação da física das partículas elementares, nuclear, atômica, molecular e do estado sólido, os seus impactos foram consideráveis com desenvolvimentos tecnológicos como o microscópio eletrônico de varrimento por efeito túnel (STM), o relógio atômico do sistema de posicionamento global (GPS), a produção de imagens por ressonância magnética (NMR) e a tomografia por emissão de pósitrons (PET). Estes avanços foram possíveis graças ao uso de computadores, que permitiram novas formas de ver e compreender a estrutura atômica e molecular.

Contudo, a aprendizagem da Mecânica Quântica é normalmente considerada difícil pelos alunos, devido ao seu carácter axiomático. Quando os alunos estudam Mecânica Quântica, têm frequentemente de visualizar informação 3-D mostrada em gráficos 2-D. Uma dificuldade adicional dos alunos é a utilização de conceitos que são muito diferentes dos que são aprendidos na Mecânica Clássica (por exemplo, um elétron num dado estado pode ser detectado em diferentes pontos do espaço, com diferentes probabilidades dadas pelo módulo do quadrado da função de onda). Contudo, no ensino tradicional da Mecânica Quântica, supõe-se que o conceito de probabilidade é de fácil assimilação para os alunos. A noção de ocupação probabilística do espaço parece, porém, oferecer algumas dificuldades dos alunos.

Existe uma defasagem entre a inovação tecnológica e a sua aplicação no ensino. Contudo, desde os primórdios da utilização dos computadores no ensino, têm sido reconhecidas as potencialidades dos computadores como ferramenta de ensino e aprendizagem. Uma das mais promissoras tecnologias de apoio ao ensino e aprendizagem das ciências é a realidade virtual. Entre as fronteiras desta e da realidade existe o espectro da realidade aumentada na qual cenários do mundo real (por exemplo, os gráficos 2-D habituais dos manuais de Mecânica Quântica) podem ser combinados com modelos virtuais (por exemplo, os respectivos modelos 3-D correspondentes às representações 2-D). Uma das vantagens pedagógicas da realidade aumentada é a possibilidade de permitir a visualização simultânea de processos a diferentes escalas. Pode ainda juntar os gráficos 2-D dos textos de Mecânica Quântica com os respectivos modelos 3-D.

Motivados por estas idéias, está sendo desenvolvido, aplicado e avaliado um programa baseado em realidade aumentada para ajudar no estudo de conceitos de Mecânica Quântica para alunos dos anos terminais do ensino secundário e do primeiro ano da universidade. Os conceitos pertencem ao âmbito da Mecânica Quântica não relativista, incluindo a interpretação probabilística da função de onda, as propriedades das soluções da equação de Schrödinger, os princípios da incerteza e de exclusão de Pauli, etc. Também serão ilustradas algumas experiências importantes como a de difração de Rutherford, a de Thompson sobre a razão e/m para um elétron, etc.

1.6.7.1 Dificuldades de aprendizagem da Mecânica Quântica

Algumas dificuldades associadas à aprendizagem da Mecânica Quântica estão ligadas com a necessidade de ver o invisível. A utilização de modelos bidimensionais nos livros de texto e a manipulação de modelos matemáticos (em geral mal compreendidos) não facilitam a compreensão, sobretudo para alunos com menores aptidões espaciais [Allendoerfer, 1990]. Isto explica a afirmação comum entre estudantes de que a Mecânica Quântica é difícil, pois “é tudo matemática”. De acordo com Hurwitz *et al.* [Hurwitz, 1998], um aluno com maiores dificuldades de raciocínio espacial estará em desvantagem face a outros mais dotados desse ponto de vista.

Tem havido alguma movimentação neste domínio, com alguns autores a aperceberem-se do poder da imagem para ajudar à compreensão de certos conceitos. A título de exemplo, refere-se o livro de Brandt e Dahmen “The Picture Book of Quantum Mechanics” [Brandt, 2000], que revela a importância dada hoje à visualização no ensino da Física. Esta obra utiliza várias Figuras tridimensionais para representar, por exemplo, superfícies com densidade de probabilidade constante do átomo de hidrogênio. Contudo, a maior diversidade de aplicações é apoiada pelo uso de computadores, como será visto em seguida.

1.6.7.2 O computador e a aprendizagem da Mecânica Quântica

Vários autores [Redish, 2000] [Rebello, 2000] defendem o uso regular de ferramentas computacionais de simulação e visualização no ensino, com particular destaque para a utilização de *software* que permita a interatividade [Boyce, 1997]. Com efeito, a utilização de recursos computacionais no ensino introdutório da Mecânica Quântica é uma tendência que tem vindo a crescer rapidamente nos últimos anos. Existem já vários programas cujo objectivo é auxiliar os estudantes a visualizar alguns aspectos do mundo microscópico. *Atomic Orbitals CD*, por exemplo, é um *software* multimídia desenvolvido por Y. Wong que permite visualizar a três dimensões as formas das orbitais do hidrogênio e as densidades eletrônicas, entre outros aspectos. Rebello e Zollman [Rebello, 2000] também desenvolveram um projeto denominado *Visual Quantum Mechanics*, cujo objetivo é introduzir tópicos de Mecânica Quântica, com a ajuda de simulações, de atividades interativas e de laboratórios, recorrendo a um mínimo de ferramentas matemáticas. Alguns casos práticos começaram mesmo já a ser incorporados no contexto de sala de aula, no ensino secundário [Jones, 1999] e no universitário. Por exemplo, Shotsberger e Vetter [Shotsberger, 2001], da Universidade de North Carolina, Wilmington, nos EUA, referem a utilização frequente do *HyperChem* (*software* de modelação e visualização de estruturas moleculares) por alunos, em aulas de Química no ensino secundário, para criar, modificar e medir estruturas moleculares. Aqueles autores defendem que os alunos devem ter um papel activo na utilização das ferramentas computacionais.

Uma das tecnologias computacionais recentes é a realidade virtual. Apontada como um poderoso instrumento de ensino e treino, esta tecnologia disponibiliza uma vertente inovadora, quer através da interação com modelos tridimensionais quer através de experiências multisensoriais [Trindade, 1999]. O grande interesse por esta área permitiu a sua especialização em vários setores e a possibilidade de diferentes combinações. Uma dessas possibilidades é a realidade aumentada.

Considerada uma variante da realidade virtual, a realidade aumentada disponibiliza um tipo não convencional de interface, que permite misturar imagens de um ambiente real, obtidas por câmara de vídeo ou por outro processo, com objetos 3-D virtuais, enriquecendo a visão do usuário. No ambiente assim gerado, o utilizador tem a sensação de que os objetos reais e virtuais coexistem no mesmo espaço na medida em que os objetos virtuais são passíveis de visualização e de interação como se existissem no mundo real. Desta forma, a realidade aumentada enfatiza a visualização em conjunto com a interação, pois, com o auxílio de dispositivos de visualização mais ou menos imersivos, objetos virtuais podem ser sobrepostos ao ambiente real, de maneira altamente realista, incrementando a percepção do usuário [Trindade, 1999].

Vários trabalhos têm sido desenvolvidos com o intuito de avaliar as potencialidades da realidade virtual no ensino e na aprendizagem das ciências. Por exemplo, o Centro de Física Computacional da Universidade de Coimbra, em colaboração com o Exploratório Infante D. Henrique e o Departamento de Matemática da Universidade de Coimbra, desenvolveram um ambiente virtual sobre a estrutura da água denominado *Água Virtual*. Trata-se de um programa que aborda conceitos relacionados com fases de água, transições de fase, agregados moleculares, estruturas de gelo para além da visualização de elementos básicos de mecânica quântica como orbitais atômicas do átomo de hidrogênio e orbitais moleculares da água. O programa é voltado para alunos dos anos terminais do ensino

secundário e do primeiro ano do ensino superior e pode ser disponibilizado gratuitamente a pedido [Água Virtual, 2004].

O Orbitário

O Orbitário [Trindade, 2004] é destinado ao estudo de orbitais hidrogenóides, num nível introdutório, e encontra-se no início da sua avaliação. A escolha da Mecânica Quântica encontra a sua justificação nas razões anteriormente invocadas, nomeadamente a exploração de conceitos abstractos, que são ensinados nas aulas, mas para os quais não existem modelos de referência acessíveis (conceitos de orbital e de densidade eletrônica) [Barton, 1997] [Story, 1998]].

O programa foi desenvolvido com o *ARToolKit*, que é o software livre normalmente utilizado para desenvolver aplicações de realidade aumentada. Trata-se de uma ferramenta de código aberto com frequentes atualizações, encontrando-se acessível em vários sites [Kato, 2003].

Um procedimento fundamental no desenvolvimento da aplicação consiste na sobreposição dos modelos virtuais (as orbitais) com o ambiente real (por exemplo, o manual). Para tal, é preciso obter a posição e a orientação dos objetos da cena real e associá-los com os modelos previamente armazenados no computador. Isto é normalmente conseguido com o recurso a determinados padrões, isto é, placas com marcas fiduciais que contêm símbolos para se diferenciarem uma das outras, tornando-as singulares.

Usando como exemplo a associação do modelo da orbital 2p à respectiva marca fiducial (Figura 6), a imagem da placa fiducial, captada por uma *webcam*, é convertida para um formato binário. Uma vez reconhecido o padrão, é calculada a posição e orientação do marcador relativamente à *webcam*. O modelo da orbital associado ao padrão é identificado e é feito o *rendering* do modelo virtual no vídeo. O modelo da orbital (estático ou dinâmico) irá aparecer sobreposto ao respectivo padrão e a sua representação poderá ser feita de forma imersiva (num capacete de visualização ou óculos com ou sem estereoscopia) ou de forma não imersiva no monitor do computador (Figura 1-33). Ao manipular a placa (que pode estar incorporada num livro de texto), o objeto virtual realizará os mesmos movimentos da placa, como se a ela estivesse preso.

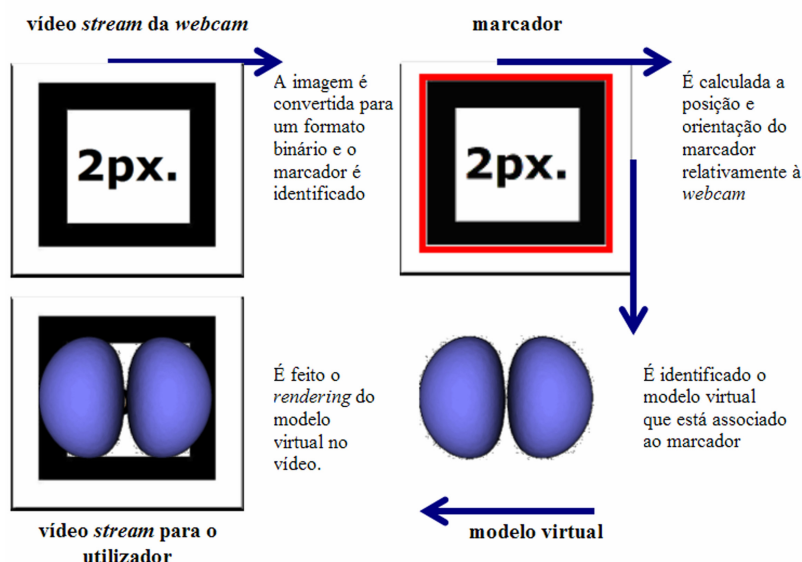


Figura 1-33 - Sobreposição do modelo virtual ao objecto real tomando como exemplo a orbital

$2p_x$.

A *webcam* capta o padrão pré-definido e converte-o para um formato binário para reconhecimento. O modelo da orbital associado ao padrão é identificado e é calculada a posição e a orientação do marcador relativamente à *webcam*. Finalmente efetua-se o *rendering* do modelo virtual no vídeo que aparecerá sobreposto ao respectivo marcador.

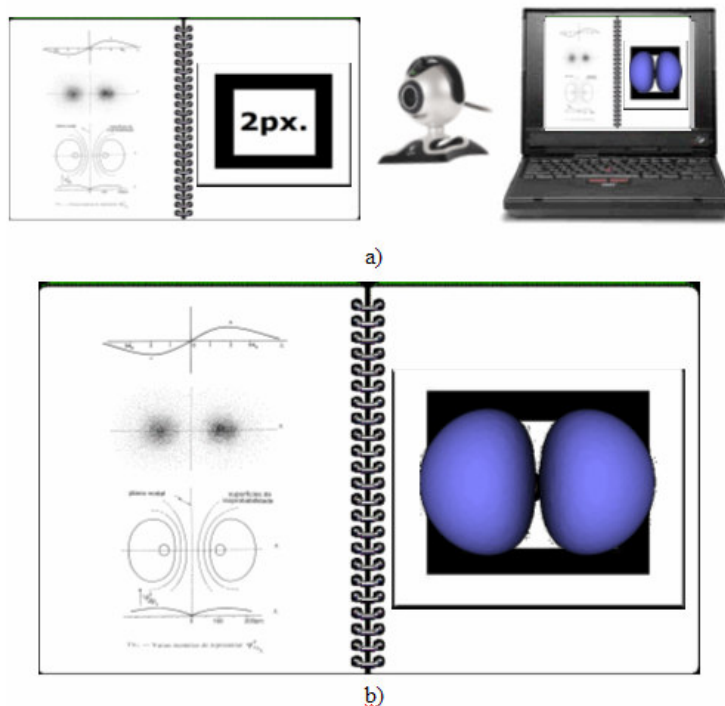


Figura 1-34 – modelo de orbital

Uma vez reconhecido o padrão, o modelo da orbital (estático ou dinâmico) irá aparecer sobreposto ao respectivo marcador e a sua representação poderá ser feita, por exemplo, de forma não imersiva no monitor do computador; b) Ao manipular a placa (que pode estar incorporada num manual), o objeto virtual realizará os mesmos movimentos do marcador, como se a ele estivesse preso.

Como o *ARToolKit* usa código aberto, foi possível desenvolver recursos adicionais no código referentes à interação da aplicação e à implementação de som.

No que toca à interação, é possível alterar o modelo virtual, a partir da introdução de uma placa adicional de controle. A alteração dinâmica do objeto virtual, a partir da introdução de uma placa adicional de controle, consiste na utilização de uma placa que possa interferir no objeto virtual de outra placa, de modo a trocá-lo ou alterá-lo, o que viabiliza a associação de mais objetos virtuais a uma só placa, de acordo com a necessidade da aplicação. Isto permite, por exemplo, que se possa visualizar a mesma orbital mas para diferentes orientações espaciais ou para diferentes distribuições de densidade eletrônica. Esta (simples) interação, que consiste apenas na troca de uma placa de controle, é acompanhada de som que explica as alterações visíveis no modelo, como por exemplo a variação da densidade eletrônica. Com a inserção do som, consegue-se com a modificação

do *ARToolKit* para que seja executado o arquivo de áudio com o surgimento do objeto virtual, quando se introduz a placa em frente à câmara. Assim, foi possível combinar num ambiente o melhor da exibição bidimensional (dos manuais) com a exibição computacional 3D, apelando-se a uma utilização conjunta do livro de texto e do computador, contrariamente ao que habitualmente acontece com a utilização do software.

1.6.8 Conclusões

Em atividades educativas com uso de recursos computacionais, é necessário o planejamento adequado do uso de tecnologias em software e hardware, que possibilitam explorar: a carga cognitiva do usuário, o senso de observação, a atenção, a exploração e a retenção de informações, utilizando elementos de percepção visual, tátil e auditiva. Para isso, é necessário desenvolver sistemas computacionais que apresentem: 1) facilidade na identificação dos componentes; 2) adaptabilidade ao nível do usuário; 3) adequação dos programas às necessidades curriculares; 4) existência de recursos motivacionais; 5) possibilidade constante de alterações do sistema (inclusão de novos elementos); 6) fornecimento de retorno; 7) integração do sistema com outros recursos; 8) capacidade de armazenamento de respostas; 9) registro de tempo de resposta e latência e 10) liberdade de manipulação e navegação - controle sobre os elementos.

Uma das vantagens do uso da Realidade Aumentada é a possibilidade de criar interfaces multisensorias, sem a necessidade de periféricos de alto custo, como óculos e capacete de visualização, luva, caverna (CAVE), utilizados em experimentos de realidade virtual. Com isto, é possível o desenvolvimento de ambientes relevantes e interessantes para o usuário, utilizando materiais acessíveis e disponíveis nos laboratórios de informática das escolas.

Considerando a importância do processo de construção do conhecimento para a formação do indivíduo, os ambientes de realidade aumentada podem contribuir de forma a oferecer condições de experimentar, criar, descobrir, sem instruções previamente elaboradas ou definidas pelo sistema, pois o ambiente funciona como mediador, auxiliando o educando na descoberta e construção do novo.

A construção do ambiente gráfico ficou muito mais simplificada e rápida, reduzindo-se exclusivamente ao que é essencial: os modelos que se pretendem explorar no ensino e aprendizagem.

A tecnologia envolvida desempenha um papel muito menos dominante e distrativa para o aluno. Tal como referido por Cornu [Cornu, 1995], “as novas tecnologias apenas estarão integradas no ensino quando elas não forem ferramentas suplementares mas sim quando elas se tornarem ‘naturais’ e ‘invisíveis’ como o telefone, o televisor e as calculadoras de bolso. As novas tecnologias só estarão realmente integradas na pedagogia quando, dessa união, surgirem novos métodos pedagógicos bem sucedidos”.

1.7 REFERENCIAS BIBLIOGRÁFICAS

Água Virtual (2004) <http://aguavirtual.mediaprinter.pt>

Akagui, D.; Kirner, C. (2004) “LIRA - "Livro Interativo com Realidade Aumentada", Proc. of VII Symposium on Virtual Reality, SP, outubro de 2004.

- Allendoerfer, R. (1990) "Teaching the shapes of the hydrogenlike and hybrid atomic orbitals". *Journal of Chemical Education*, 67 (1990) 37-40.
- Azuma, R. (1997) "A Survey of Augmented Reality", *Presence: Teleoperators and Virtual Environments*, vol. 6, 4, August 1997), p. 355-385.
- Azuma, R, et al. (2001) "Recent Advances in Augmented Reality". *IEEE Computer Graphics and Applications*, November/December 2001, vol. 21, p. 34-37.
- Baranauskas, M. C. C.; Rocha, H. V. (2003) "Design e Avaliação de Interfaces Humano-Computador". Campinas – SP: NIED/UNICAMP, 2003.
- Barton, G. (1997) "Quantum dynamics of simple systems". *Contemporary Physics*, 38 (1997) 429-430.
- Boyce, W. (1997) "Interactive Multimedia Modules in Mathematics, Engineering, and Science". *Computers In Physics*, 11 (1997) 151-157.
- Brandt, S., Dahmen, H. (2000) "The Picture Book of Quantum Mechanics". Springer, New York (2000).
- Cornu, B. (1995) "New technologies: integration into education". In D. Watson e D. Tinsley (Eds.), *Integrating Information Technology into Education*. Chapman & Hall, New York (1995).
- Dainese, C.A., Garbin, T.R. e Kirner, C. (2003) "Sistema de Realidade aumentada para o Desenvolvimento da Criança Surda", In: VI Symposium on Virtual Reality:, 2003. Ribeirão Preto - SP. SBC, 2003. p.273-281.
- Galana, S.C., Silva, R.R.P.C.L., Kirner, C. (2004) "Autoria Colaborativa de Mundos Virtuais Educacionais com Realidade Misturada" Anais do 1º Workshop de Realidade Aumentada, Piracicaba, SP, maio de 2004, p. 17-20.
- Garbin, T.R., Dainese, C.A., Kirner, C., Santos, A.M., Jesus, M.A. (2004) "Avaliação de Interface de um Sistema de Realidade Aumentada para a Criança Surda com base no Construcionismo" Anais do 1º Workshop de Realidade Aumentada, Piracicaba, SP, maio de 2004, p. 33-36.
- Hurwitz, C., Abegg, G., Garik, P., Nasr, R. (1998) "High school students' understanding of the quantum basis of chemistry". *J. of Research in Science Teaching*, 34(1998) 535-545.
- Jones, L. (1999) "Learning Chemistry through design and construction. *UniServe Science News*", 14 (1999) 3-7.
- Kato, H.; Billingham, M.; Poupyrev, I. (2003) "ARToolKit version 2.33 Manual", Nov.,2003. http://www.hitl.washington.edu/research/shared_space
- Lahr, P; Lourenço, P. C.; Dainese, C. (2004) "Desenvolvimento de uma ferramenta para Rastreamento em Sistemas de Realidade Aumentada". Anais do I WorkShop sobre Realidade Aumentada – WRA 2004, UNIMEP, Maio, 2004, 37-40.
- Lévy, P. (1999) "O que é virtual?", Tradução de Paulo Neves, São Paulo: Ed. 34, 1999.
- Rebello, N., Zollman, D. (2000) "Conceptual understanding of quantum mechanics after using hands on experiments and visualization instructional materials". In *Annual*

- Meeting National Association for Research in Science Teaching, Boston (<http://www.phys.ksu.edu/perg/papers/narst>, Novembro de 2000).
- Redish, E., Bao, L. (2000) "Student difficulties with energy in quantum mechanics". In AAPT Winter Meeting, Phoenix (<http://www.physics.umd.edu/perg/cpt.html>, consultado em Outubro de 2000).
- Shotsberger, P., Vetter, C. (2001) "Teaching and learning in the wireless classroom. Computer", 34 (2001) 110-111.
- Santin, R.; Kirner, C. (2004) "Desenvolvimento de Técnicas de Interação para Aplicações de Realidade Aumentada com o ARToolKit". Anais do I WorkShop sobre Realidade Aumentada – WRA 2004, UNIMEP, Maio, 2004, 13-16.
- Story, R. (1998) "Bridging a quantum-mechanical barrier". IEEE Transactions on Education, 41 (1998) 54-60.
- Trindade, J., Fiolhais, C., Gil, V. (1999) "Virtual Water, an application of virtual environments as an education tool for physics and chemistry". In G. Cumming, T. Okamoto, L. Gomez (Eds.), Proceedings of ICCE'99, 7th International Conference on Computers in Education, Chiba, Japão, (1999) 655-658.
- Trindade, J., Kirner, C., Fiolhais, C. (2004) "An Augmented Reality Application for Studying Atomic Orbitals: Orbitário" Proc. of IADIS International conference on Cognition and Exploratory Learning in Digital Age, Lisboa, Portugal, Dec., 2004.
- Utiyama, F., Kirner, C. (2004) "Rastreamento e Visualização de Trajetórias para Treinamento com Realidade Aumentada", Proc. of VII Symposium on Virtual Reality, SP, outubro de 2004.
- Valente, J.A. (2001) "Aprendendo para a vida: o uso da informática na educação especial". In Valente, J.A. Aprendendo para a vida: os computadores na sala de aula. São Paulo. Cortez Editora, p.29-42, 2001.
- APPLE, C. FireWire. (<http://developer.apple.com/firewire/>). 2004.
- BERRE, A., INGVALDSEN, K., SYVERINSEN, T. & SALOMONSEN, M. ARToolKit for Dummies, 2002. (<http://rasmus.uib.no/~st02204/HCI/>).
- CAREY, R. & BELL, G. *The Annotated VRML 2.0 Reference Manual*. Addison-Wesley, 504pp, 1997.
- DAVIS, L., CLARKSON, E. & ROLLAND, J.P. Predicting Accuracy in Pose Estimation for Marker-based Tracking. In: *Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR '03)*, Tokyo, Japan, p28-35, 2003.
- GEIGER, C., REIMANN, C., STICKLEIN, J. & PAELKE, V. JARToolKit - A Java binding for ARToolKit. In: *The First IEEE Augmented Reality Toolkit International Workshop*, p124, 2002.
- HARTLEY, R. & ZISSERMAN, A. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 655pp, 2003.

- KATO, H. *Inside ARToolKit*. 2002. (http://www.ikg.uni-hannover.de/lehre/katalog/Augmented_Reality/pdf_files/ART02-Tutorial.pdf)
- KATO, H. & BILLINGHURST, M. Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. In: *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, San Francisco, CA, USA, p85-94, 1999.
- KATO, H., BILLINGHURST, M. & POUPYREV, I. ARToolKit Version 2.33, 44pp, 2000. (<http://www.hitl.washington.edu/people/grof/SharedSpace/Download/ARToolKit2.33doc.pdf>).
- KE, X. Tutorial: Installation of the ARToolkits and the relevant hardware (Version 2.11), 2000. (<http://www.ece.nus.edu.sg/stfpage/eleadc/ARToolkit-tutorial-v2.pdf>).
- MALBEZIN, P., PIEKARSKI, W. & THOMAS, B.H. Measuring ARToolKit Accuracy in Long Distance Tracking Experiments. In: *Proceedings of the 1st Augmented Reality Toolkit Workshop*, Darmstadt, Germany, p124-125, 2002.
- MICROSOFT, CO. Introduction to DirectShow. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directshow/hlm/introductiontodirectshow.asp>. 2004.
- MICROSOFT, CO. Microsoft Vision SDK, Version 1.2. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvis/sdk/html/vis/sdk.asp>. 2000.
- MICROSOFT, CO. WDM: Introduction to Windows Driver Model. <http://www.microsoft.com/whdc/archive/wdm.msp>. 2002.
- MORLEY, C. LibVRML97 / Lookat. <http://www.vermontel.net/~cmorley/vrml.html> . 2000.
- NOMAD ELECTRONICS.COM. Capture using Video for Windows. [http://www.nomadelectronics.com/VidCap/Capture using VFW/captureVFW.htm](http://www.nomadelectronics.com/VidCap/Capture%20using%20VFW/captureVFW.htm). 2004.
- OPENVRML.ORG. OpenVRML. <http://www.openvrml.org/>. 2003.
- OWEN, C.B., XIAO, F. & MIDDLETON, P. What is the best fiducial? In: *Proceedings of The First IEEE Augmented Reality Toolkit International Workshop*, p124, 2002.
- SGI OpenGL. Ver. 1.4. Silicon Graphics Incorporated, 2003. (<http://www.opengl.org/>)
- WAGNER, D. & SCHMALSTIEG, D. ARToolKit on the PocketPC platform. In: *Proceedings of the 2nd Augmented Reality Toolkit Workshop*, p14-15, 2003.