Um Framework para o Desenvolvimento de Aplicações de Realidade Virtual Baseadas em Componentes Gráficos

Thiago de A. Bastos, Alberto B. Raposo*, Marcelo Gattass*

[1]Tecgraf - Grupo de Tecnologia em Computação Gráfica / Dep. de Informática
PUC-Rio - Pontifícia Universidade Católica do Rio de Janeiro, Brasil

{tbastos, abraposo, mgattass}@tecgraf.puc-rio.br

Abstract. *Virtual Reality (VR) applications differ from other applications for the usage of special devices and the concern with the user's sensation of immersion. The use of frameworks is important to such applications due to the high complexity and costs involved in their development. This paper presents a graphical component-based VR framework designed to be simple, flexible, and to allow the creation of easy-to-operate applications.*

Resumo. *As aplicações de Realidade Virtual (RV) se diferenciam de outras aplicações pelo uso de dispositivos especiais e pela característica imersiva do usuário no sistema. O uso de frameworks é importante para essas aplicações devido à alta complexidade e o custo de desenvolvimento envolvidos. Este artigo apresenta um framework de RV baseado em componentes gráficos, projetado para ser simples, flexível, e criar aplicações fáceis de serem operadas.*

1. Introdução

Realidade virtual é uma técnica avançada de interface, onde o usuário pode navegar e interagir em um ambiente sintético tridimensional gerado por computador, estando completa ou parcialmente imerso pela sensação gerada por canais multi-sensoriais, sendo o principal a visão [Burdea and Coiffet 1994]. Um dos benefícios dos ambientes de RV é a capacidade de prover perspectivas vantajosas impossíveis de se obter no mundo real, como a navegação por dentro do corpo humano, a análise de simulações físicas e a revisão de grandes projetos de engenharia [Corseuil et al. 2004]. A interação com o ambiente virtual é feita utilizando dispositivos próprios para RV, como capacetes de visualização, luvas, mouses tridimensionais, sensores de posicionamento, entre outros [Silva 2004].

Os sistemas de RV são compostos por dispositivos de entrada (rastreadores de posição, reconhecimento de voz, joysticks, etc) e saída (visual, auditiva e táctil) capazes de prover ao usuário imersão, interação e envolvimento [Morie 1994]. A idéia de imersão está ligada com o sentimento de se estar dentro do ambiente. A interação diz respeito à capacidade do computador detectar as entradas do usuário e modificar instantaneamente o mundo virtual e as ações sobre ele. O envolvimento, por sua vez, está ligado com o grau de engajamento de uma pessoa com determinada atividade, podendo ser passivo, como assistir televisão, ou ativo, ao participar de um jogo ou simulação interativa.

Além do *hardware*, os sistemas de RV precisam de um *software* para capturar a entrada dos dispositivos, simular o ambiente virtual e gerar as saídas multi-sensoriais. As

*Orientadores.

aplicações de RV são complexas por lidarem com dispositivos e técnicas complexas de interação e apresentação. Freqüentemente é necessário que elas sejam multi-plataforma e funcionem com qualquer combinação de dispositivos – por exemplo, para permitir a colaboração entre instituições com diferentes sistemas de RV.

Os *frameworks* de RV permitem que o usuário se concentre no desenvolvimento da aplicação, não sendo necessário se preocupar com a gerência do sistema de RV. Eles podem oferecer: *i)* abstração de dispositivos; *ii)* abstração de sistema de projeção; *iii)* grafos de cena especializados; *iv)* heurísticas de interação com o ambiente virtual; *v)* suporte a sistemas distribuídos; e *vi)* renderização distribuída, dentre outros recursos. De acordo com as soluções oferecidas, um *framework* apresentará vantagens para tipos particulares de aplicações. A independência de dispositivos (itens *i* e *ii*) é geralmente o recurso mais valioso para uma aplicação de RV. Porém, ambientes de RV muito complexos podem exigir que a aplicação seja distribuída, aumentando a importância dos itens *v* e *vi*.

Simplicidade e flexibilidade são qualidades primordiais em um *framework* de RV. Se for demasiado complexo, o *framework* pode comprometer a qualidade da aplicação ou tornar-se inapropriado para projetos mais simples. Os *frameworks* de RV também devem ser flexíveis o suficiente para se adaptarem aos requisitos voláteis inerentes aos projetos de pesquisa, impondo o mínimo possível de restrições e permitindo boa extensibilidade. Os principais *frameworks* de RV são desenvolvidos por laboratórios com orçamentos de dezenas de milhões de dólares, tendo como plataforma alvo os supercomputadores *Onyx*, da *Silicon Graphics*. Vários deles não funcionam na plataforma Windows/PC, e poucos se preocupam com a usabilidade da aplicação – normalmente operada por especialistas.

Este trabalho apresenta um novo modelo de *framework* de RV: mais simples e flexível, e que gera aplicações mais fáceis de operar. O modelo proposto se adequa bem aos sistemas de RV de baixo custo, baseados em PCs. O trabalho está organizado como se segue: a seção 2. discute os principais trabalhos relacionados. A seção 3. apresenta o *framework* proposto, fornecendo uma visão geral de sua arquitetura e detalhando os seus principais conceitos. A seção 4. mostra como novos dispositivos de RV podem ser integrados ao *framework*, e a seção 5. expõe o processo de construção de uma aplicação de RV utilizando o *framework*. Por fim, a seção 6. adiciona as considerações finais.

2. Trabalhos Relacionados

Existem diversos *frameworks* para o desenvolvimento de aplicações de RV, cada qual oferecendo vantagens para tipos particulares de aplicações. O instituto IMK/Fraunhofer criou o Avango [Tramberend 1999], um *framework* para o desenvolvimento de ambientes virtuais distribuídos com foco em sistemas de RV *high-end*. O Avango funciona como um grafo de cena distribuído de forma semi-transparente, e provê formas básicas de interação com o mundo virtual (clicar, mover objetos, etc). Ele depende do grafo de cena Performer [Rohlf and Helman 1994], e é relativamente limitado quanto às aplicações que podem ser criadas. O instituto *Virginia Tech* criou o Diverse [Kelso et al. 2002], um *framework* que também oferece suporte a sistemas de RV distribuídos: não tão simples, porém mais flexível que o Avango, com melhor suporte a dispositivos de RV, e menos dependente do Performer. Ambos os *frameworks* estão limitados aos sistemas IRIX e Linux.

O centro de RV da *Iowa State University* criou o VR Juggler [Bierbaum 2000] para ser uma plataforma virtual de desenvolvimento de aplicações de RV. Ele abstrai a

plataforma, os dispositivos e o sistema de projeção em uso, permitindo que a aplicação se adapte a qualquer sistema de RV. Arquivos XML descrevem como os dispositivos de entrada devem ser usados e como as saídas devem ser geradas. O *framework* é versátil e tem boa portabilidade, funcionando bem em PCs com Windows. As versões mais novas podem funcionar com qualquer grafo de cena baseado no OpenGL, e aceitam *plugins* para suporte a novos dispositivos de entrada. Contudo, o VR Juggler é grande, complexo e depende de muitas bibliotecas externas. Alguns subsistemas misturam C++, CORBA e Java desnecessariamente, e as aplicações finais não são fáceis de instalar ou operar.

Dentre os três *frameworks* citados, o modelo do VR Juggler é o mais escalável e flexível. O suporte à distribuição presente nos dois primeiros *frameworks* pode ser interessante para alguns projetos, mas certamente não satisfaz a todos. Tratar formas de interação com o ambiente virtual, como faz o Avango, envolve restrições sérias, como padronizar a representação do mundo virtual, e não faz sentido para todas as aplicações. O VR Juggler foca nos pontos essenciais em RV: interface com dispositivos e independência de sistema de RV, sem restringir a aplicação, que pode facilmente utilizar um segundo *framework* para tratar os aspectos de distribuição e interação com o mundo virtual.

O *framework* aqui apresentado segue a mesma filosofia do VR Juggler. Porém, diferente dos demais projetos — que são *frameworks white-box* convencionais — nossa idéia foi seguir um modelo *grey-box* baseado em componentes de software com interfaces gráficas. Esta idéia propiciou vantagens diretas e indiretas sobre os outros *frameworks*, permitiu implantar simplificações importantes e introduziu outros conceitos inovadores.

3. ViRAL, um Framework Baseado em Componentes Gráficos

O ViRAL (*Virtual Reality Abstraction Layer*) é um *framework* para o desenvolvimento de aplicações de RV baseadas em componentes gráficos [Bastos et al. 2004]. Ele facilita a criação de aplicações extensíveis totalmente operadas por interfaces WIMP (*Windows, Icons, Menus and Pointers*). Cada aplicação permite simular múltiplos ambientes virtuais, visitados por múltiplos usuários nos mais diversos sistemas de RV. As aplicações são operadas instanciando, configurando e conectando componentes via interfaces gráficas.

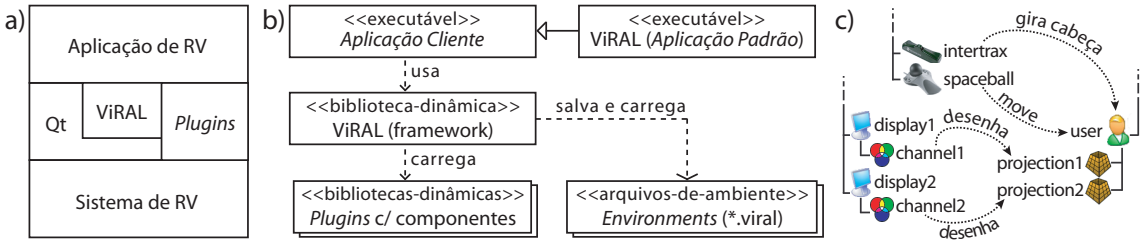

Figura 1. a) Acoplamento entre camadas; b) Diagrama de instalação; c) Alguns componentes e seus relacionamentos.

O *framework* é independente de plataforma e foi desenvolvido em C++. Além do Windows, que é sua plataforma principal, já foi utilizado em Linux e IRIX. Diferente do VR Juggler, que depende de muitas bibliotecas e tem sua própria camada da abstração de plataforma, o ViRAL utiliza uma única biblioteca multi-plataforma para todas as necessidades do projeto, inclusive interfaces gráficas: o Qt [Trolltech 2005a] (Figura 1a).

Devido à sua arquitetura, o ViRAL pode ser utilizado como uma biblioteca para desenvolver novas aplicações, ou como uma aplicação autônoma extensível (Figura 1b). As principais entidades presentes nos ambientes de RV (usuários, projeções, dispositivos, etc) são representadas no *framework* por *componentes*, dispostos em duas estruturas: uma árvore e um grafo (Figura 1c). A árvore representa a hierarquia lógica dos componentes, enquanto o grafo representa outros relacionamentos. Estas estruturas permitem visitar os componentes de forma ordenada, e determinar cadeias de dependência, respectivamente.

A interface gráfica de uma aplicação ViRAL é dinâmica, *construída em tempo de execução*, e reflete os componentes e *plugins* ativos. Nesse processo se utiliza a árvore de componentes, cuja raiz é o componente especial "ViRAL", responsável por operações de âmbito global como *iniciar* ou *pausar* as simulações de RV. Os componentes diretamente abaixo da raiz são os *subsistemas*, que representam os principais módulos da aplicação. O ViRAL possui sete subsistemas padrões, e outros podem ser introduzidos por *plugins*. Os demais componentes na árvore precisam ser filhos de um subsistema. A árvore pode se estender indefinidamente, de acordo com a variedade de componentes e sistemas, e a interface das aplicações deve se adaptar automaticamente. No caso da *aplicação padrão* do ViRAL (Figura 2a), a interface construída é uma janela com múltiplas abas, cada uma contendo a tela de um subsistema. Para facilitar a criação de interfaces, o ViRAL fornece seus próprios *widgets*: para editar vetores e quatérnios, ver a árvore de componentes, etc.

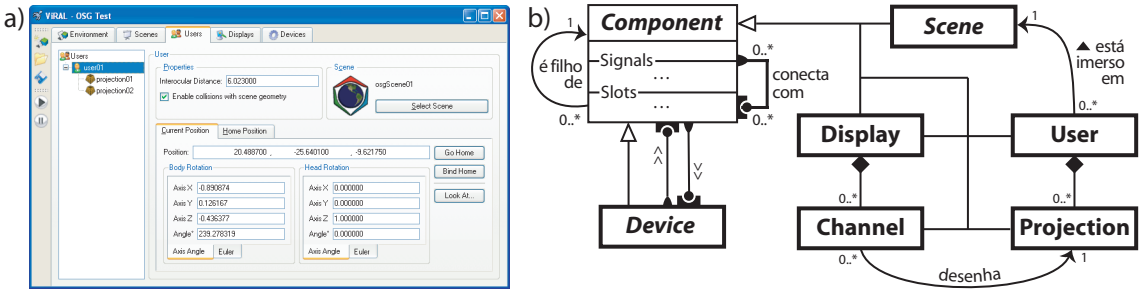

Figura 2. a) Interface da aplicação padrão; b) Componentes centrais do ViRAL.

3.1. Ambientes de Realidade Virtual

No modelo do ViRAL um ambiente de RV é descrito por um conjunto de componentes interconectados. Há componentes representando usuários, dispositivos de RV, projeções de vídeo e até mesmo ambientes virtuais. Os componentes são gerenciados via interfaces gráficas por um operador, e podem ser instanciados de forma a simular múltiplos mundos virtuais concorrentemente. Este modelo difere essencialmente do VR Juggler, onde as aplicações são desenvolvidas para simular um ambiente virtual específico. A seguir é feito um resumo dos principais componentes e como eles se relacionam (Figura 2b):

- ***Scene* (cena):** tipo de componente responsável pela simulação de mundos virtuais. Sua interface define *template methods* que são usados para desenhar o mundo. A implementação do componente pode ser feita usando qualquer grafo de cena ou técnica de visualização, contanto que desenhe no *color buffer* do OpenGL. Além da visão, é possível gerar outras saídas sensoriais: sonora, através de um módulo auxiliar; e táctil (ou qualquer outra), através do sistema de dispositivos do ViRAL.

- ***User* (usuário):** representa um usuário imerso em um ambiente virtual. Conhece a posição e a orientação do usuário no mundo virtual – e possivelmente também no mundo real –, e sua distância interocular, usada nos cálculos de estereoscopia.
- ***Projection* (projeção):** representa uma visão do mundo virtual de um usuário. Em sistemas de multi-projeção como a CAVE [Cruz-Neira et al. 1993] há uma instância deste componente para cada parede de projeção. As projeções definem um *frustum de visão* e uma *transformação de corpo rígido* aplicada à cabeça do usuário. Em sistemas de RV avançados as projeções são auto-calculadas em tempo real, dada a posição da tela e do usuário no mundo real [Pape et al. 1999].
- ***Display* (visor):** representa um *canvas* do OpenGL associado a uma saída de vídeo (ou tela de projeção). Em ambientes *desktop* é visto como uma janela.
- ***Channel* (canal):** componente responsável pela estratégia de desenho de uma *projeção* em um *visor* utilizando um método de estereoscopia escolhido.
- ***Device* (dispositivo):** componente de uso geral, usualmente proativo ou reativo, que gera e trata eventos. Seu papel principal é representar dispositivos reais, como rastreadores, mouses 3D e luvas. Também é comum haver dispositivos auxiliares para gerar, transformar, adaptar ou filtrar os eventos de outros dispositivos.

3.2. Componentes

O modelo de componentes do ViRAL dá suporte a comunicação, persistência e integração com interfaces gráficas. Os componentes podem trocar mensagens utilizando o paradigma de *signals* e *slots* [Trolltech 2005b], com conexões feitas de forma segura, em tempo de execução, graças a um mini-sistema de introspecção. Para que possam ser integrados de forma semi-automática nas aplicações, os componentes podem ter interface gráfica, menu de contexto e ícones associados. Esses elementos de interação gráfica são normalmente fornecidos pela mesma biblioteca do componente, mas de maneira desacoplada. Novos tipos de componentes podem ser carregados automaticamente de bibliotecas dinâmicas.

3.3. Persistência e Reconfiguração em Tempo de Execução

A estrutura de componentes do ViRAL descreve integralmente um ambiente de RV. Para que o operador não precise recriá-la toda vez que a aplicação é aberta, o *framework* provê persistência automática da estrutura em arquivos. Além de restaurar os componentes e suas configurações, o sistema de persistência permite salvar e resumir uma simulação de RV exatamente do ponto em que ela foi interrompida – como fazem os *save games* nos jogos de computador. Como os componentes são configurados estritamente via interfaces gráficas, eles podem ser re-configurados sempre que necessário, em tempo de execução, permitindo que os operadores façam testes e ajustes rápidos. Estas facilidades não são simples de se obter com o VR Juggler, onde as configurações são lidas de arquivos XML.

3.4. Dispositivos de RV como Componentes

Nos *frameworks* de RV convencionais, a abstração de dispositivos é implementada usando *interfaces padrões* para os principais tipos de dados que os dispositivos de RV podem gerar. No VR Juggler, por exemplo, os dispositivos provêem basicamente três tipos de dados: *posicional*, que é uma matriz de transformação; *analógico*, que é um número real; e *digital*, que é um valor booleano. Um dispositivo como um mouse 3D, por exemplo, gera um valor posicional e vários digitais (um para cada botão). Dessa forma as aplicações

de RV são programadas para receber apenas os tipos de entradas padronizadas, e arquivos de configuração são usados para definir de onde (quais dispositivos) cada entrada é obtida.

No ViRAL os dispositivos são implementados como componentes: geram entrada por meio de *signals*, e recebem dados por *slots*. Um mouse 3D pode ter *signals* para eventos de translação, rotação, e clique de botões; uma câmera pode emitir um *signal* com uma imagem sempre que uma foto é capturada; e um dispositivo com *force-feedback* pode ter *slots* para receber sinais de força. Este modelo é muito escalável e flexível por basear-se na infra-estrutura de comunicação inter-componentes do ViRAL. Os dispositivos podem ter *signals* e *slots* arbitrários: basta haver um outro componente com *signal* ou *slot* compatível para que ambos os componentes possam se comunicar.

Toda a comunicação com dispositivos é gerenciada pelo operador da aplicação via interface gráfica. Ele pode ver os *signals* e *slots* dos componentes disponíveis e fazer conexões entre eles, definindo o roteamento de eventos para a simulação de RV. Por exemplo, para que um usuário possa navegar pelo seu mundo virtual usando um mouse 3D, o operador pode conectar os *signals* de translação e rotação do mouse 3D com os *slots* de translação e rotação do componente *User* que representa o usuário (Figura 3).

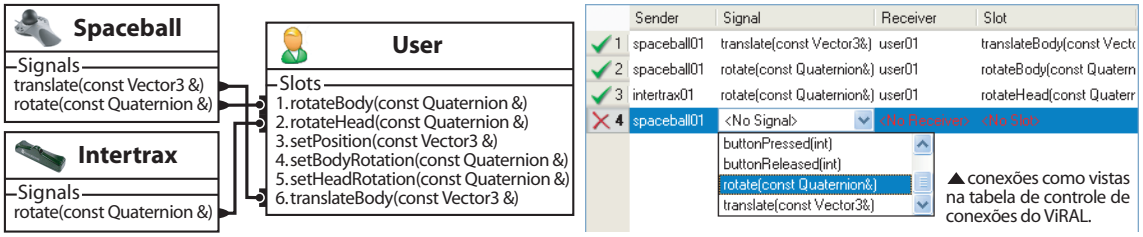

Figura 3. Exemplos de conexões entre um *User* e dois dispositivos.

3.5. Rastreamento e Movimento

Dispositivos de RV podem gerar *posições absolutas* ou *movimento relativo* como entrada para as aplicações. Rastreadores, por exemplo, geram amostras de posição e orientação absolutas em um sistema de coordenadas próprio. Já joysticks e mouses 3D geram sinais analógicos que descrevem rotações e translações em um eixo de coordenadas local. No VR Juggler essas diferenças são ignoradas e todos os dispositivos são tratados como sendo *absolutos*. Um mouse 3D, por exemplo, retorna para o sistema apenas uma matriz com a sua *posição final*, calculada como um acúmulo das transformações feitas com o mouse. O modelo do ViRAL, por outro lado, é bem mais flexível: um dispositivo pode ter *signals* com valores absolutos, relativos, ou ambos, e o operador pode escolher quais usar.

Um *User* pode ser guiado pelo seu mundo virtual através de entradas que recebe por seus *slots*. Há *slots* que atribuem posições absolutas, e outros que aplicam rotações e translações ao eixo local do *User*. No mundo virtual, a posição de um *User* é dada por três parâmetros: um vetor de posição $\mathbf{p}$, um quatérnio de orientação do corpo $\hat{\mathbf{q}}_{body}$ e outro de rotação da cabeça $\hat{\mathbf{q}}_{head}$. Uma matriz de visão centrada no usuário é calculada como:

$$\mathbf{M}_{userview} = translate(-\mathbf{p}) * rotate(\hat{\mathbf{q}}_{body}\hat{\mathbf{q}}_{head})^T$$

Na Figura 3, os *slots* 3, 4 e 5 do componente *User* atribuem novos valores às variáveis $\mathbf{p}$, $\hat{\mathbf{q}}_{body}$ e $\hat{\mathbf{q}}_{head}$, respectivamente, e devem ser usados com dispositivos que emitem posições

e orientações absolutas, como os rastreadores. Já os *slots* 1 e 2 recebem quatérnios de rotação que são acumulados às variáveis $\hat{\mathbf{q}}_{body}$ e $\hat{\mathbf{q}}_{head}$, respectivamente; o slot 6 translada o usuário em seu espaço local, multiplicando o vetor de translação pelo quatérnio $\hat{\mathbf{q}}_{body}$.

O ViRAL padroniza a representação de posições e translações com *vetores*, e orientações e rotações com *quatérnios*. Os dispositivos sempre emitem *signals* de posição e translação separados dos de orientação e rotação. Um componente pode usar apenas os eventos de rotação de um dispositivo, e obter eventos de translação de outro dispositivo. Múltiplos dispositivos podem ser conectados ao mesmo *slot* e usados simultaneamente.

3.6. Visualização

No ViRAL, uma simulação de RV pode ser iniciada ou pausada com o clique de um botão. Uma aplicação pode estar *parada* (como na Figura 2a) ou *simulando* (com o sistema de RV funcional). Internamente existem os estados intermediários *iniciando* e *parando*. Na fase de *iniciação* o sistema de RV é preparado para a simulação: *canvas* do OpenGL são criados, conexões com os dispositivos são estabelecidas, etc. Na fase de *parada* ocorre o processo inverso: a simulação é congelada e o sistema de RV é desativado. Tais transições são coordenadas pelo *framework* estritamente através de mensagens para os componentes.

O principal trabalho realizado pelo *framework* é o de renderização e simulação concorrente de ambientes virtuais (Figura 4a). Cada *Display* renderiza os seus quadros em um *thread* próprio, e as visões de um mesmo *User* são sincronizadas para que reflitam o mesmo estado e para que os visores sejam atualizados simultaneamente. Este sincronismo é essencial em sistemas de multi-projeção estereoscópica, pois evita a incoerência de quadros (que causa perda de imersão) e os desconfortos causados por *fantasmas* (quando um olho consegue ver imagens que só deveriam ser vistas pelo outro olho).

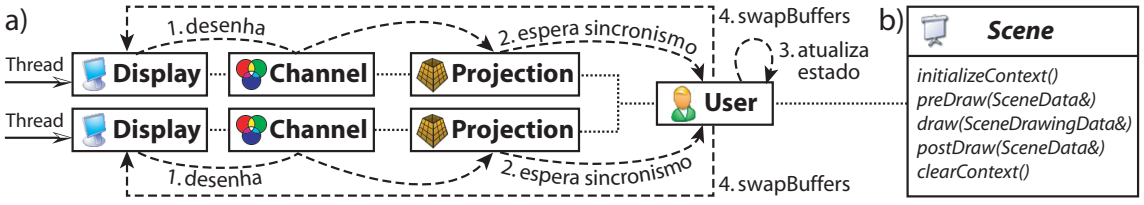

Figura 4. Processo de renderização concorrente (*multi-threaded*).

O ViRAL participa de parte do *pipeline* de renderização fazendo a abstração do sistema de projeção. Ele desenha os ambientes virtuais usando os métodos implementados pelos componentes *Scene* (Figura 4b). Os métodos recebem do *framework* a configuração exata que deve ser usada na renderização, incluindo *viewport*, máscara de cor e matrizes de *modelview* e *projection*. Um *Scene* não precisa conhecer o sistema de projeção ou o método de estereoscopia (*stereo*) em uso: basta usar as configurações fornecidas e ele já estará em conformidade com o sistema de RV. O ViRAL calcula as matrizes de *modelview* e *projection* para cada quadro renderizado, baseado na posição do *User* e as configurações do sistema de projeção. Para renderizar em *stereo*, os métodos de desenho são chamados duas vezes, com as matrizes calculadas para cada olho. O *framework* é capaz de gerar os principais tipos de *stereo*: por *quad-buffer*, anaglifos, dupla saída, lado-a-lado, etc.

As *Projections* do ViRAL podem funcionar em modo *manual* ou *rastreado*. O modo *manual* permite que o operador defina todos os aspectos da projeção diretamente,

o que é ideal em sistemas de RV simples (*e.g.* baseados em monitor ou capacetes de RV) onde o *frustum* de visão não se deforma de acordo com o movimento do usuário. Já o modo *rastreado* calcula automaticamente a projeção ao longo da simulação de RV, usando um sistema de rastreamento para descobrir, no mundo real, a posição dos olhos do usuário e as arestas da tela de projeção. Este último modo é complexo e requer dispositivos de rastreamento presos à cabeça do usuário (e possivelmente às telas, se elas forem móveis), porém ele é essencial em sistemas de RV imersivos onde o usuário e as telas podem se mover independentemente, fazendo com que a projeção precise ser recalculada *online*.

Quando em modo *manual*, uma *Projection* pode ser configurada com um *frustum* de visão, uma transformação $\mathbf{M}_{offset}$ aplicada à câmera, e para projetar em *stereo*: uma distância de paralaxe zero dpz. Com esses parâmetros, mais a distância interocular di e a matriz de visão $\mathbf{M}_{userview}$ do usuário, o ViRAL calcula as matrizes *modelview* e *projection* para cada olho. As *modelviews* são calculadas levando em conta a distância interocular:

$$\mathbf{M}_{modelview} = \mathbf{M}_{userview} * translate(\pm\frac{di}{2}, 0, 0) * \mathbf{M}^{-1}_{offset}$$

As *projections* são matrizes de perspectiva criadas de *frustums* assimétricos derivados do *frustum* especificado pelo operador. Os *frustums* derivados têm os planos esquerdo e direito deslocados para evitar que a visão *stereo* tenha paralaxe vertical (Figura 5a).

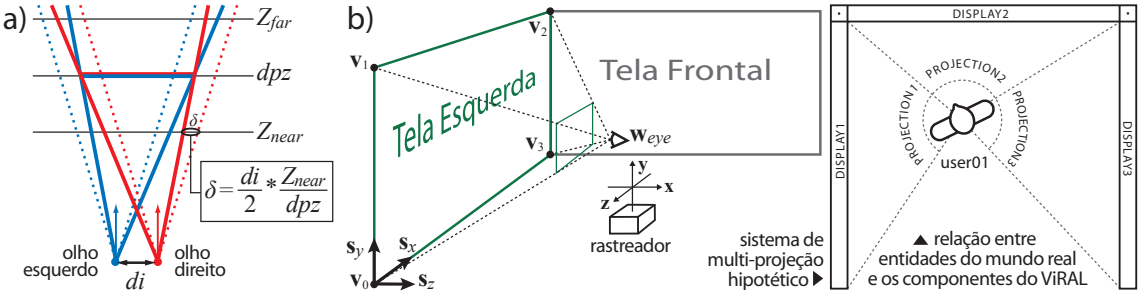

Figura 5. a) *Frustums* de visão *stereo*; b) Sistemas com projeções rastreadas.

Uma *Projection* em modo *rastreado* só precisa conhecer os vértices de sua tela de projeção no mundo real (Figura 5b), e todas as suas propriedades são auto-calculadas. Neste modo, é necessário que haja um rastreador preso à cabeça do usuário, fornecendo ao componente *User* a sua posição no mundo real. A *Projection* pode ser configurada com os quatro vértices da tela no mundo real (no caso da tela ser imóvel), ou apenas com a largura/altura da tela – e receber a posição e a orientação do vértice inferior-esquerdo através de *slots* (para telas móveis, rastreadas). Essas informações devem ser fornecidas no mesmo espaço de coordenadas do rastreador usado com o *User*. A partir delas, o ViRAL calcula os eixos do sistema de coordenadas da tela ($\mathbf{s}_x$, $\mathbf{s}_y$, $\mathbf{s}_z$). Sabendo a posição $\mathbf{w}_{eye}$ de um dos olhos do usuário, um *frustum* de visão é calculado fazendo projeções do olho para a tela (Figura 5b), e uma matriz $\mathbf{M}_{offset}$ para deslocar a câmera é calculada como: $translate(\mathbf{w}_{eye}) * rotate(\mathbf{s}_x, \mathbf{s}_y, \mathbf{s}_z)$. Fazendo estes cálculos uma vez para cada olho, obtem-se uma projeção *stereo* com o paralaxe zero exatamente na tela de projeção.

4. Integrando Dispositivos ao Framework

O ViRAL provê suporte a novos dispositivos através de *plugins*: um componente do tipo *Device* pode ser desenvolvido, distribuído como uma biblioteca, e posto no diretório de

plugins da aplicação para que possa ser usado. Os *Devices* que representam dispositivos reais têm a função de transformar em *signals* as entradas geradas pelos equipamentos; e quando apropriado, transmitir para eles as saídas que são recebidas via *slots*. A interface com os dispositivos é geralmente feita usando *kits* de desenvolvimento (SDKs) fornecidos pelos fabricantes. É comum que os *Devices* tenham um *thread* só para amostrar os dados dos dispositivos e convertê-los para uma representação compatível com o ViRAL, antes de emiti-los como *signals*. Por serem componentes, os *Devices* podem ter telas de interface que permitam a sua configuração ou até a visualização do estado do equipamento.

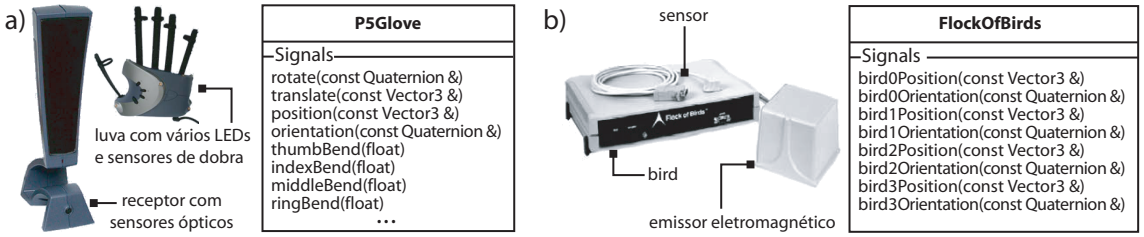

Figura 6. Dispositivos de entrada e seus *signals*: a) Luva P5; b) *Flock of Birds*.

A luva P5 foi um dos primeiros dispositivos integrados ao ViRAL. Ela fornece sensores de dobra para os dedos e rastreamento da mão dentro de uma pequena área em frente a um receptor (Figura 6a). O *Device* processa os dados da luva com *filtros* (*e.g.* filtro de Kalman) para amenizar erros de rastreamento, e emite *signals* quando há novas amostras. As dobras dos dedos são representadas por `floats` num intervalo de zero (dedo totalmente dobrado) a um (dedo esticado). Baseado em regras definidas pelo operador, o componente é capaz de interpretar os movimentos absolutos da mão e gerar também *signals* de rotação e translação; por exemplo, capturando os movimentos relativos feitos de mão fechada. Outro dispositivo que possui *plugin* para o ViRAL é o *Flock of Birds*, um sistema modular de rastreamento eletromagnético (Figura 6b). Ele pode rastrear até quatro sensores num raio de três metros, obtendo amostras a 144Hz com precisão média de $2mm$ para posições e 0.5^o para orientações. Seu *Device* possui uma tela de interface que permite a configuração de opções como a porta (RS-232) e o *baud rate* usados para a comunicação, e provê diagnósticos básicos sobre o estado do equipamento. Além dos dispositivos citados, o ViRAL tem *plugins* para joysticks, Spaceballs, rastreadores da InterSense, 5DT Data Gloves, e alguns dispositivos ópticos desenvolvidos no Tecgraf por alunos de pós-graduação — um deles, apresentado em [Silva 2004].

5. Construindo Ambientes de RV com o ViRAL

Uma aplicação desenvolvida com o ViRAL é capaz de simular uma variedade ilimitada de ambientes de RV, proporcional à diversidade de componentes disponíveis. A construção de um ambiente de RV é feita em três passos: primeiro, se necessário, são desenvolvidos novos componentes (*e.g.* um *Scene* para simular o mundo virtual desejado); em seguida, os componentes que formam o ambiente proposto são instanciados e configurados em uma aplicação ViRAL; finalmente, o estado da aplicação é salvo para um arquivo (Figura 1b) de onde o ambiente de RV pode ser restaurado quando desejado. Enquanto no VR Juggler as aplicações são desenvolvidas para simular um único ambiente de RV, no ViRAL cada *arquivo de ambiente* pode encapsular um ambiente de RV diferente, e uma única aplicação pode simular todos esses ambientes, graças à sua arquitetura baseada em componentes.

Esta seção mostra a construção de um ambiente de RV com o ViRAL. O objetivo é simular um ambiente submarino multi-usuário onde é possível operar um braço robótico para manipular peças que estão no fundo do mar (cada usuário opera um braço distinto). Neste exemplo, o sistema de RV seria composto por: dois usuários, dois monitores *stereo*, dois *flock-of-birds* e duas luvas com sensores de dobra. Cada usuário controlaria o seu braço robótico de forma intuitiva, usando um de seus braços: o *flock-of-birds* rastrearia a mão do usuário no mundo real e controlaria os movimentos do braço virtual; a luva controlaria a abertura da garra virtual, baseado na dobra dos dedos do usuário. Assumindo que já existem *Devices* para a luva e o *flock-of-birds*, seria necessário apenas desenvolver uma *Scene* para simular o ambiente virtual submarino com suporte a braços robóticos.

De fato, tal *Scene* já foi desenvolvida (Figura 7b): ela desenha o ambiente virtual usando o grafo de cena de código aberto *OpenSceneGraph*, e utiliza simulação física para controlar o braço robótico e tratar sua interação com as peças que estão no fundo do mar. Para prover suporte a múltiplos usuários (ou múltiplos braços robóticos), a *Scene* explora a arquitetura baseada em componentes do ViRAL e representa os braços robóticos como sendo seus *subcomponentes* (Figura 7a). Um componente de braço robótico (*Arm*) pode ser instanciado pelo menu do componente *ArmScene*, e posicionado no ambiente virtual através de sua própria tela de interface. Cada *Arm* possui *slots* para receber diretamente a entrada dos dispositivos que irão controlá-lo. Há um *slot* para receber a posição em que a mão do braço robótico deve estar, e outro para receber o grau de abertura da garra (um `float` no intervalo $[0, 1]$ – a mesma convenção usada nas luvas).

Dispondo de todos os componentes necessários para criar o ambiente de RV, o passo seguinte seria usar uma aplicação ViRAL para criar e configurar: uma cena do tipo *ArmScene* com dois subcomponentes *Arm*, dois *Displays*, dois *Users*, dois *flock-of-birds* e duas luvas (Figura 7a). Cada *User* seria pré-posicionado próximo ao seu braço robótico no ambiente virtual, e cada par de dispositivos de entrada seria conectado a um *Arm*. Por último, cada *Display* seria configurado para desenhar (em *stereo*) a *Projection* de um *User*. O ambiente de RV, então, estaria pronto para começar a ser simulado.

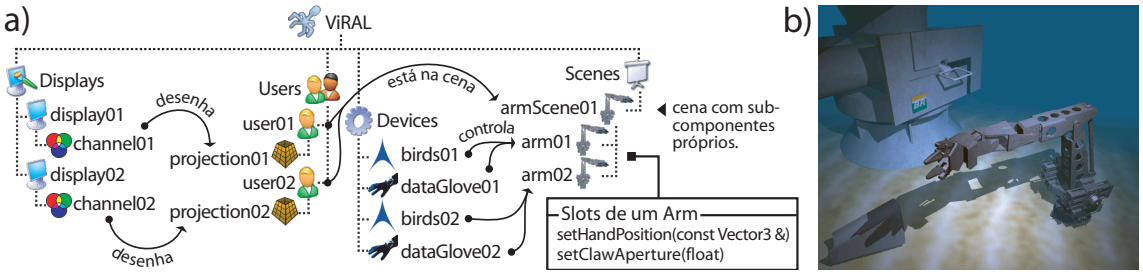

Figura 7. Exemplo: a) árvore de componentes; b) visão da cena.

6. Conclusão

Este artigo apresentou o ViRAL, um *framework* de RV baseado em componentes gráficos. Ele está sendo usado para criar aplicações independentes de sistema de RV e operadas por interfaces gráficas. Sua arquitetura baseada em componentes facilita o desenvolvimento e eleva o reuso de artefatos entre projetos. As aplicações são construídas instanciando e configurando componentes, podem simular vários mundos virtuais concorrentemente, e são capazes de salvar e restaurar os ambientes de RV usando arquivos. O uso do modelo

de comunicação por *signals* e *slots* mostrou-se muito adequado para fazer a integração entre componentes nos ambientes de RV. O ViRAL possui atualmente *plugins* para vários dispositivos; já foi testado com capacetes, projetores *stereo* ativos e passivos e sistemas de multi-projeção. Os mundos virtuais desenvolvidos vão desde jogos até visualizadores de modelos CAD. Vídeos do ViRAL em ação estão disponíveis em [Bastos 2005].

Agradecimentos A pesquisa em Realidade Virtual do Tecgraf/PUC-Rio é apoiada primordialmente pelo CENPES/PETROBRAS, pelo FINEP e a RNP (projeto GIGA).

Referências

Bastos, T. A. (2005). ViRAL Videos. *http://www.tecgraf.puc-rio.br/~tbastos/viral/videos/*.

Bastos, T. A., Silva, R. J. M., Raposo, A. B., and Gattass, M. (2004). ViRAL: Um Framework para o Desenvolvimento de Aplicações de Realidade Virtual. In *VII Symposium on Virtual Reality*, pages 51–62, São Paulo, SP, Brasil.

Bierbaum, A. D. (2000). VR Juggler: A Virtual Platform for Virtual Reality Application Development. Master's thesis, Iowa State University.

Burdea, G. and Coiffet, P. (1994). *Virtual Reality Technology*. Wiley-Interscience.

Corseuil, E., Raposo, A., Silva, R., Pinto, M., Wagner, G., and Gattass, M. (2004). ENVIRON - Visualization of CAD Models in a Virtual Reality Environment. In *Eurographics Symposium on Virtual Environments*, pages 79–82, Grenoble, França.

Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. (1993). Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. *ACM Computer Graphics*, 27(2):135–142.

Kelso, J., Arsenault, L., Satterfield, S., and Kriz, R. (2002). DIVERSE: A Framework for Building Extensible and Reconfigurable Device Independent Virtual Environments. In *Proceedings of IEEE Virtual Reality*, pages 183–192, Orlando, FL, USA.

Morie, J. F. (1994). Inspiring the Future: Merging Mass Communication, Art, Entertainment and Virtual Environments. *SIGGRAPH Computer Graphics*, 28(2):135–138.

Pape, D., Sandin, D. J., and DeFanti, T. A. (1999). Transparently Supporting a Wide Range of VR and Stereoscopic Display Devices. In *Proceedings of SPIE, Stereoscopic Displays and Virtual Reality Systems VI*, pages 346–353, San Jose, CA, USA.

Rohlf, J. and Helman, J. (1994). IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. In *Proceedings of SIGGRAPH*, pages 381–395.

Silva, R. J. M. (2004). Integração de um Dispositivo Óptico de Rastreamento a uma Ferramenta de Realidade Virtual. *Dissertação de Mestrado, DI/PUC-Rio*.

Tramberend, H. (1999). AVANGO: A Distributed Virtual Reality Framework. In *Proceedings of IEEE Virtual Reality*, pages 14–22, Houston, TX, USA.

Trolltech (2005a). Qt Product Overview. *http://www.trolltech.com/products/qt/*.

Trolltech (2005b). Signals and Slots. *http://doc.trolltech.com/3.3/signalsandslots.html*.