A VR Framework for Desktop Applications

Lucas Teixeira, Daniel Trindade, Manuel Loaiza, Felipe G. de Carvalho, Alberto Raposo
Tecgraf, Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro,Brazil
{*lucas, danielrt, manuel,kamel, abraposo*}*@tecgraf.puc-rio.br*

Ismael Santos
CENPES
Petrobras
Rio de Janeiro,Brazil
ismaelh@petrobras.com.br

***Abstract*—The emergence of cheaper technologies for immersive environments has increased considerably the interest on applications of Virtual Reality (VR). However, currently available VR frameworks force user applications to be developed specifically for them. This increases the cost of converting an existing graphical application to virtual reality environments. This paper proposes a new framework, the LVRL (*Lightweight Virtual Reality Libraries*), which allows both creation and conversion of existing applications to VR without changing the structure of the application. The LVRL main objective is to provide a minimalist programming interface and non intrusive allowing the development of VR applications by non VR developers. This article describes the architecture of LVRL, its features, usage and the benefits obtained by the applications that use it.**

***Resumo*—Com o barateamento dos recursos de hardware para ambientes imersivos, o interesse por aplicações de Realidade Virtual (RV) vem aumentando consideravelmente. Entretanto os frameworks atualmente disponíveis exigem que as aplicações do usuário sejam desenvolvidas especificamente para eles. Isso faz com que o custo de se portar uma aplicação legada para ambientes RV seja alto, na maioria das vezes até proibitivo. Este artigo propõe um novo framework, o LVRL (*Lightweight Virtual Reality Libraries*), que permite a criação ou conversão de aplicações existentes para RV sem alteração na estrutura da aplicação. O principal objetivo do LVRL é fornecer uma interface de programação minimalista e não intrusiva permitindo o desenvolvimento de aplicações RV por desenvolvedores sem conhecimento específico em RV. Este artigo descreve a arquitetura do LVRL, suas funcionalidades, forma de usar e os benefícios obtidos pelas aplicações que o utilizam.**

***Keywords*-Realidade Virtual; Framework; interação 3D**

I. Introdução

Nos últimos anos a RV vem se popularizando devido a redução dos custos de hardware e consequentemente dos sistemas imersivos. No passado tais ambientes eram extremamente caros, chegando a custar alguns milhões de dólares no caso de CAVEs [1]. Com o advento das televisões 3D de alta definição a preços acessíveis, ambientes imersivos como a NexCAVE [2] estão ao alcance da maioria das empresas de engenharia, design, jogos e outras que trabalham com 3D no dia a dia.

Apesar da redução dos custos do hardware a produção de software para ambientes imersivos ainda é uma atividade custosa. Um aspecto que dificulta a criação de novas aplicações é a complexidade para transformar uma aplicação desktop funcional em uma aplicação RV. Além das peculiaridades desse tipo de ambiente, pesa ainda o fato que praticamente todos os SDKs para RV modernos estarem dependentes de algum tipo de renderizador, como é o caso do VrJuggler [3], BlenderCave [4], Eon Studio [5], Avango NG [6], 3DVIA Virtools [7] e INVRS [8]. Por consequência as aplicações imersivas precisam ser inteiramente refeitas para se adequar a uma dessas plataformas.

Hoje em dia já existem muitas aplicações 3D baseadas em renderizadores específicos como os visualizadores de modelos massivos [9], de modelos de elementos finitos [10], de nuvens de pontos [11], de dados sísmicos [12] e de dados visuais resultantes de simuladores científicos em geral. O custo de reescrevê-los dentro de plataformas específicas seria, em muitos casos, demasiadamente alto, dado o nível de especificidade do renderizador ou a plataforma ser fechada como no caso do EON Studio e do Virtools.

No entanto, todo esse trabalho não é de fato necessário pois os componentes necessários para a criação de uma aplicação RV podem ser construídos de forma a serem utilizados em aplicações já existentes. Uma aplicação imersiva se diferencia de uma aplicação desktop pelo fato de ter de suportar múltiplas saídas de vídeo de pontos de vista diferentes. Ainda, o usuário não pode usar dispositivos de interação convencionais, como mouse e teclado, por estar em pé em frente ao ambiente imersivo (Figura 1).

Nesse trabalho propomos um framework não intrusivo que possibilita a conversão de aplicações desktop em imersivas por um desenvolver não especializado em RV. O framework permite que o desenvolvimento da aplicação seja feito exclusivamente em desktop e garante que ao mudar para o modo RV os dados serão visualizados considerando o formato do ambiente imersivo, assim como a navegação terá as a mesmas funcionalidades adaptadas para dispositivos RV.

Na seção II apresentamos as ferramentas existentes hoje e as razões que dificultam a conversão dos softwares 3D desktop para ambientes imersivos. Em seguida, na seção III explicamos cada uma das bibliotecas que compõem nosso framework e mostramos como essas tornam transparente para o usuário o fato da aplicação estar sendo executada em

ambientes desktop ou imersivos. Na seção IV discutimos as características e contribuições que o nosso framework propõe. Por fim, algumas conclusões e trabalhos futuros são apresentados na seção V.

II. Trabalhos Relacionados

Os frameworks para RV existentes tentam assumir algumas funções. São elas: renderização de múltiplos pontos de vista, captura de eventos de dispositivos de interação 3D e distribuição da renderização em diferentes maquinas da rede. Como já citado, ao mesmo tempo em que tentam resolver esses problemas, também requerem, em contrapartida, que o desenvolvimento seja orientado para suas plataformas específicas.

BlenderCave [4], Virtools [7] e EON Studio [5] são ferramentas de autoria para desenvolvimento de aplicações 3D interativas. Elas tem um sistema de renderização interno o qual o desenvolvedor da aplicação não tem acesso. Os dados são modelados em ferramentas de terceiros como o 3DStudio Max e importados para dentro da ferramenta. Nesse tipo de software não é possível fazer uma renderização específica, como por exemplo técnicas de renderização baseadas em pontos. Quanto à interação, esses frameworks são capazes de ler os dispositivos de RV, mas o desenvolvedor precisa desenvolver o suporte e o mecanismo de interação necessário para o dispositivo que deseja suportar, a partir dos dados obtidos diretamente desse último.

Frameworks programáticos de mais baixo nível, como o VrJuggler [3], Avango NG [6] e INVRS [8] em geral são baseadas em grafos de cena desenvolvidos por terceiros para fazer a renderização. Grafos de cena bastante comuns são o OpenSceneGraph [13], OpenSG [14] e SGI OpenGL Performer [15]. Caso a aplicação a ser convertida já use um desses grafos de cena, provavelmente a parte do render será mais facilmente convertida para ambientes imersivos usando esses frameworks, uma vez que eles já resolvem, ou podem ser facilmente adaptados para resolver, questões como a renderização de múltiplos pontos de vista e a distribuição do rendering por diferentes maquinas da rede. No entanto, em geral, é necessário se manter duas versões do software, uma desktop e outra para ambientes imersivos. Isso acontece porque esses frameworks precisam controlar o laço principal de controle da aplicação para ser possível fazer o desenho sincronizado em todas as telas. Já com relação à interação 3D elas têm o mesmo problema das ferramentas de autoria, isto é, elas passam o dado cru dos dispositivos e o desenvolvedor é o responsável por tratar cada tipo de dispositivo que queira usar na aplicação.

A Cavelib [16] foi a primeira biblioteca para ambientes imersivos e foi criada junto com a primeira CAVE. Ela até hoje é mantida e vendida. A sua única diferença para os frameworks citadas acima é que ela não é dependente de um grafo de cena. Sua estratégia é assumir que toda a informação dinâmica de desenho precisa estar numa área

Figura 1. LVRL sendo usado em uma CAVE com suporte a Flystick.

de memória protegida. Essa memória é compartilhada e pode ser distribuída entre os computadores. No entanto, mesmo não dependente de um grafo de cena específico, ela ainda exige que o desenvolvimento seja orientado a ela. Adicionalmente, ela também precisa que os manipuladores de câmera sejam feitos da mesma forma que as outras bibliotecas, só que usando uma camada de entrada de dispositivos que tem uma interface parecida com o VRPN [17]. Essa dependência de grafos de cena ou estratégias de uso de memória em todas as soluções citadas está intimamente ligada com a distribuição da renderização por várias máquinas. O gestor dessa distribuição precisa saber o que distribuir e assim, poder sincronizar o desenho nas diferentes máquinas. No entanto, assim como fornece uma importante funcionalidade, ele adiciona uma restrição muito forte ao framework. Ao mesmo tempo, existem outras soluções de renderização distribuída genéricas [18], [19], que não têm o cálculo dos múltiplos pontos de vista nativo, mas fornecem a mesma funcionalidade se a aplicação for capaz de calcular internamente os múltiplos pontos de vista.

O LVRL fornece toda a parte de manipulação de câmera e os cálculos dos múltiplos pontos de vista. Acreditamos que os métodos de interação de câmera devem convergir a formatos já sedimentados e testados. Embutindo-se tais métodos no framework, pode-se reduzir o esforço dos desenvolvedores na criação de aplicações que são ao mesmo tempo desktop e imersivas. Sendo assim, ao invés de eventos de dispositivos de entrada, o framework já fornece uma posição e orientação 3D.

III. Framework

O framework provê ferramentas para o programador leigo em algoritmos de RV converter um programa desktop em um programa com interface 3D multimodal que funcione em ambientes desktop e imersivos apenas trocando a configuração do ambiente em tempo de execução. Para isso foram criados módulos que realizam as seguintes tarefas: auxiliar o sistema de renderização já existente a criar as

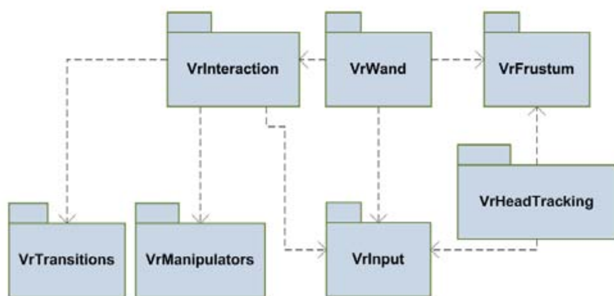

Figura 2. Diagrama com as dependências entre as bibliotecas

câmeras virtuais para os múltiplos pontos de vista; fornecer manipuladores de câmera já projetados para funcionar nos dois tipos de ambiente de forma idêntica apenas mudando o tipo de dispositivo utilizado; fornecer uma interface de *Wand*, como uma generalização do mouse para interação 3D. A seguir segue a descrição da arquitetura geral do framework e cada uma das bibliotecas que o integram.

A. *Arquitetura*

O framework é formado por 7 componentes. O diagrama de dependência pode ser visto na figura 2. Três dessas, são bibliotecas matemáticas sem dependência nenhuma, que inclusive podem ser usadas em outros frameworks de RV. São eles o *VrFrustum*, *VrManipulators* e *VrTransitions*. As outras quatro são de mais alto nível e fazem todo o tratamento de eventos e interação para câmera. Elas também podem ser usadas com outros frameworks, mas é necessário usar o componente de leitura de dispositivos, *VrInput* descrito a seguir, ao invés da fornecida pelos outros frameworks.

O *VrFrustum* é responsável por calcular os parâmetros das câmeras dos diferentes pontos de vistas correspondentes a cada tela do sistema imersivo dada a configuração física das telas e a posição da cabeça do usuário. No modo desktop ela gera uma única câmera correspondente ao monitor.

O *VrManipulator* contem um conjunto de manipuladores de câmera que foram projetados para funcionar com dados de entrada de mouse e teclado da mesma forma que com dados de entrada de dispositivos de RV. Esses manipuladores também são projetados para funcionar tanto em ambientes imersivos quanto em desktop.

Os outros quatro componentes são ligados a captura e interpretação dos dispositivos de entrada: *VrInput*, *VrInteraction*, *VrHeadTracking* e *VrWand*.

O *VrInput* é responsável pelo gerenciamento dos dispositivos de entrada. O *VrHeadTracking* monitora o sensor que lê a posição da cabeça do usuário e alimenta o *VrFrunstum* que precisa conhecer essa posição para calcular os parâmetros das múltiplas câmeras. O *VrWand* informa a direção para onde o usuário está apontando o dispositivo de entrada. No modo RV os dispositivos geralmente fornecem a posição absoluta dentro do ambiente imersivo, assim como a orientação do dispositivo. Usando o *VrFrunstum* para transformar a posição do dispositivo de entrada de coordenadas do ambiente imersivo para coordenadas do mundo, a biblioteca é capaz de calcular um vetor direção e uma posição para o raio. Já no modo desktop o processo é semelhante. No entanto o mouse só fornece uma posição 2D do cursor na tela. Para transformar isso num raio calculamos o vetor que sai da posição da câmera no mundo e passa por aquela posição no plano de projeção da câmera (o plano near).

Por fim o *VrInteraction* é responsável por interpretar os eventos de entrada e transformá-los em operações de manipulação da câmera. Essa biblioteca fornece uma matriz de view que posiciona e orienta a câmera no mundo. Quando o *Vrinput* gera um evento de um dos dispositivos de entrada suportados pelo *VrInteraction*, esse interpreta o evento segundo um mapeamento, previamente especificado entre o dispositivo e o manipulador, e chama o *VrManipulator* para efetuar os cálculos matemáticos que atualizam a matriz de view corrente. Outra responsabilidade do *VrInteraction* é o gerenciamento das transições. Quando o usuário muda de manipulador ativo o *VrInteraction* verifica se existe algum pulo da última pose do manipulador que está sendo desativado para a posição inicial do manipulador que está sendo ativado e efetua as transições necessárias entre as duas poses. A seguir são apresentados maiores detalhes de cada um desses componentes.

B. *VrInput*

O *VrInput* é o módulo responsável pelo acesso de dispositivos de entrada como mouse, teclado, joysticks e rastreadores. Atualmente, existe suporte para os seguintes dispositivos:

- Teclado e mouse;
- Controle wii;
- Tablet Ipad;
- Celulares e tablets Android;
- Controle Flystick2;
- Rastreador BraTrack;
- SpaceBall;
- Joysticks (somente Windows);
- Kinect (somente Windows);

Além dos dispositivos listados também há suporte a leitura de dispositivos presentes na biblioteca VRPN [17].

O funcionamento do *VrInput* é baseado em eventos, criados quando um novo input é gerado pelo dispositivo. Através de um gerenciador , o *VrInput* envia esses eventos para a aplicação, que decide então quais ações devem ser executadas.

O acesso ao *VrInput* é feito através de uma única interface, responsável por criar o acesso aos dispositivos, além de permitir ativá-los ou desativá-los. Para garantir a estabilidade e o controle eficiente do ciclo de vida dos diversos dispositivos

todo o gerenciamento fica a cargo do *VrInput*, sendo portanto vedada à aplicação o acesso direto aos dispositivos.

Só é permitida uma instância do *VrInput* por aplicação. Isso impede que haja acesso concorrente aos dispositivos entre diferentes instâncias que poderiam ter sido criadas pela aplicação. No momento em que a instância do *VrInput* é criada, é disparado um processo secundário responsável por se comunicar diretamente com os dispositivos. A comunicação entre o *VrInput* e esse processo se dá por meio de chamadas entre processos com uso de memória compartilhada. Essa arquitetura assegura que em caso de erros causados pelo dispositivo a aplicação continuará executando, uma vez somente o processo secundário é afetado.

Uma vez definidos os dispositivos a serem usados, assim como suas configurações, qualquer operação só é realizada pelo *VrInput* quando a aplicação chama a função *ProcessEvents*. Nesse momento, o *VrInput* se comunica com o processo secundário e verifica se existem eventos pendentes a serem enviados para a aplicação. Também nesse momento verifica-se se o processo secundário ainda está em execução. Caso contrário, isso pode significar que algum erro aconteceu. Nesse caso, o *VrInput* tenta recuperar o estado anterior do processo secundário, afim de garantir que a aplicação continue executando. Somente se essa recuperação não é possível a aplicação é informada do erro.

C. VrFrustum

O *VrFrustum* é responsável por configurar as múltiplas câmeras necessárias para o ambiente imersivo a partir da posição e orientação de uma câmera principal. Essa funcionalidade é comum nos frameworks de RV, no entanto o principal diferencial do *LVRL* é que o *VrFrustum* não precisa ter acesso à abstração de câmera da aplicação. Para a realização dos cálculos, basta apenas conhecer a geometria do sistema de projeção e a pose da cabeça do usuário em cada quadro.

No caso de um ambiente desktop com um monitor o cálculo fica restrito a uma única tela. Porém, no caso de uma CAVE ou NexCAVE, comumente composta por mais de 3 telas, terão que ser calculados n pontos de vista correspondente às n telas do sistema. No caso de um sistema com suporte a estereoscopia serão 2 x n.

A configuração do sistema de projeção é feita através de um arquivo escrito em XML e agrega os parâmetros que são inerentes a um determinado ambiente imersivo. Esse arquivo é comum a todas aplicações que queiram funcionar nesse ambiente. Como informação principal, esse arquivo contém a posição 3D dos 4 cantos das telas. Uma tela pode ser um monitor, um projetor ou dois projetores (no caso de projeção com estéreo passivo). Essas informações, juntamente com a posição da cabeça do usuario fornecida pelo sistema de tracker, são suficientes para o cálculo dos *frustums* das diversas câmeras.

O arquivo de configuração também contém informações sobre o mapeamento entre as diferentes saídas de vídeo dos projetores e as respectivas regiões (ou telas) do sistema de projeção (mapeamento de viewports). A viewport do olho direito é opcional, dado que se o sistema de projeção usar estéreo ativo ou não suportar estereoscopia só é necessário uma viewport. No caso do sistema imersivo funcionar distribuído, além da viewport, é necessário um identificador que diz qual máquina é responsável por ela.

A última propriedade que uma tela pode ter é se ela é a *referência* (no caso do nosso exemplo é a primeira tela). Essa propriedade existe porque os planos de clipping *near* e *far* só podem ser definidos em relação a uma tela. As outras telas precisam ser calculadas em função da tela de referência para que os planos de clipping se encaixem sem descontinuidades.

Outras informações importantes para a criação de uma cena 3D em ambientes imersivos são o *Pivot* e o *OffsetTracking*. O *Pivot* é uma posição 3D na cena e tem duas funções. A primeira é ser usada como a posição padrão da cabeça do usuário no caso em que o sistema não tenha suporte a headtracking. A segunda é ser a origem do sistema de coordenadas da matriz de view fornecida pela biblioteca para o posicionamento das câmeras no ambientes imersivo.

O *OffsetTracking* é uma posição 3D que representa a origem do sistema de coordenadas do equipamento de rastreamento. Consideramos que todos sistemas de coordenadas têm a mesma orientação e que a descrição dos cantos da CAVE estão na mesma unidade fornecida pelo equipamento de rastreamento, em geral metros.

Para a aplicação, o resultado do *VrFrustum* consiste apenas nas matrizes de projeção e de view referentes aos diferentes pontos de vista definidos no arquivo de configuração. Cabe à aplicação aplicar corretamente essas matrizes em sua abstração de câmera. Dessa forma, o *VrFrustum* não é dependente de nenhum tipo de biblioteca de visualização. Isso facilita sua utilização em ambientes legados, onde toda o mecanismo de visualização já está definido. Na figura 3 é mostrada como seria a disposição final dos frustums calculados pelo *VrFrustum* para um arquivo de configuração de um sistema de projeção no formato L.

D. VrHeadTracking

O *VrHeadTracking* é o módulo mais simples do *LVRL*. Sua única função é repassar os dados provenientes do sensor responsável por rastrear a cabeça do usuário para o *Vrfrustum*. Esse então recalcula os frustums de acordo com a nova posição da cabeça, criando assim a sensação requerida pelo efeito de head tracking.

Para usar o *VrHeadTracking*, a aplicação precisa apenas informar qual o dispositivo de rastreamento deve ser usado.

E. VrManipulators

O *VrManipulators* contém as implementações dos manipuladores disponíveis. Um manipulador cria e modifica

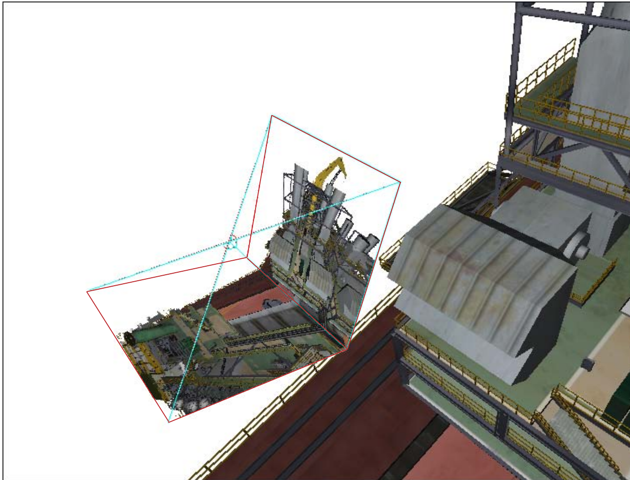

Figura 3. Visualização do sistema imersivo do arquivo de exemplo.

uma matriz de view que, se aplicada à câmera da aplicação, produz um tipo específico de comportamento. Atualmente existem quatro tipos de manipuladores:

- *Fly*: produz na câmera o comportamento de um voo livre. Dentre os manipuladores atuais é o menos restritivo. É possível apontar a câmera em todas as direções. A partir dessa direção de referência, movimenta-se a câmera para frente, para trás, para os lados, para cima e para baixo.
- *Examine*: a câmera realiza movimentos de rotação em torno de um ponto, o *pivot*. O posicionamento do *pivot* em um ponto específico de um objeto do cenário virtual, por exemplo no centro desse último, permite simular a inspeção em torno do objeto escolhido para *pivot*.
- *Walk*: tem um comportamento similar ao do manipulador *fly*, mas com a restrição de que a câmera se movimenta sempre no nível do plano do chão, mesmo quando a câmera é apontada para cima. A referência de onde é o chão em um dado momento deve ser fornecida pela aplicação. Como resultado, tem-se a sensação de se estar caminhando pela cena.
- *Rail*: nesse manipulador o movimento de translação da câmera somente pode ser feito em cima de um caminho específico atribuído previamente pela aplicação, o *trilho*. É possível movimentar a câmera nos dois sentidos do *trilho*, ao mesmo tempo em que se pode apontar a câmera em todas as direções.

Esses manipuladores possuem parâmetros de configuração, que devem ser ajustados pela aplicação. Atualmente estão disponíveis os seguintes parâmetros:

- *Velocidade de navegação*: Usado no cálculo do movimento de translação da câmera.
- *Velocidade da rotação*: Usado no cálculo de uma rotação realizada na câmera.
- *Vetor Up do mundo*: Estabelece qual o vetor UP do mundo. Em nossa implementação todos os manipuladores podem realizar rotações apenas em torno de dois eixos: o eixo UP do mundo e o eixo RIGHT da câmera. Com essa restrição, garante-se que a câmera sempre fique alinha com o UP do mundo, o que se revelou ser um requisito importante em ambientes imersivos.
- *Ponto de pivot*: É o ponto em torno do qual a câmera fará rotações. Atualmente é usado apenas pelo *examine*.
- *Ponto no chão*: Em conjunto com o vetor UP do mundo, esse ponto estabelece qual o plano do chão usado pelo manipulador *andar*.
- *Caminho do trilho*: Estabelece qual o caminho que o manipulador *trilho* deve seguir.
- *Permitir rotação em torno do vetor RIGHT*: Em testes realizados, percebeu-se que em alguns momentos é conveniente limitar a liberdade de rotação da câmera a fim de facilitar a conclusão de algumas tarefas de interação. Além disso, em alguns tipos de ambientes de visualização, algumas rotações podem provocar desorientação ao usuário. Esse é caso por exemplo quando se permite em uma CAVE rotações em torno do vetor RIGHT. Isso tem o efeito de fazer que o usuário tenha a impressão de que a cena está ”torta“ para algumas das telas. Pensando nisso, esse parâmetro permite travar a rotação em torno do eixo RIGHT.
- *Permitir rotação em torno do vetor UP*: Tem o mesmo efeito do parâmetro descrito no tópico anterior, só que atua nas rotações em torno do vetor UP do mundo.

Os manipuladores descritos acima fornecem o suporte básico para tarefas de navegação e inspeção em aplicações 3D e foram desenvolvidas para funcionar em ambientes que vão desde o desktop padrão até ambientes imersivos como CAVEs. Há planos para a criação de novos manipuladores, alguns desses sendo apenas extensões dos já existentes.

F. *VrInteraction*

O *VrInteraction* é o gerenciador da interação. Ele controla a forma como os dispositivos de entrada fazem uso dos manipuladores, assim como o processo de troca entre eles. Para isso ele faz uso dos módulos *VrInput*, *VrManipulators*, e de conjuntos de *mapeamentos* associados aos manipuladores.

Um *mapeamento* define a relação entre um dispositivo de entrada e um manipulador. Por exemplo, um mapeamento pode especificar que o movimento de arraste do mouse causa uma rotação no manipulador *examine*. Um mapeamento é projetado para escutar eventos de um dispositivo e mapeá-los em ações em um manipulador específico.

Atualmente existem mapeamentos pré-definidos para teclado e mouse, e os controles wii e flystick (figura 1). Tais mapeamentos permitem realizar as tarefas básicas de navegação e inspeção em aplicações 3D. Especificamente, os dispositivos teclado e mouse são usados em conjunto, formando os mapeamentos usados no desktop padrão, onde

é garantido que eles estarão presentes. Esses mapeamentos foram feitos para os 4 manipuladores descritos na seção III-E. Dessa forma, hoje há um total de 12 mapeamentos no *VrInteraction*: 4 para teclado e mouse, 4 para o wii e 4 para o flystick. Novos mapeamentos envolvendo dispositivos como kinect, celulares android e ipads estão sendo desenvolvidos.

Os recursos usados por esses mapeamentos não devem ser acessados diretamente pela aplicação, pois isso pode gerar uma situação de conflito em que o recurso seria usado para duas tarefas distintas simultaneamente. Por esse motivo, cada mapeamento foi desenvolvido para aproveitar as funcionalidades do dispositivo da melhor forma possível usando o mínimo de recursos do mesmo. Por exemplo, o flystick possui 6 botões e no máximo 2 são usados no *VrInteraction*. Os botões restantes são deixados para que o desenvolvedor possa, através do uso direto do *VrInput*, usá-los para realizar tarefas especificas da aplicação. Por exemplo, um dos botões restantes pode ser usado para iniciar a visualização de uma simulação.

A presença de mapeamentos pré-definidos tem como vantagem minimizar o trabalho do desenvolvedor que deseja criar uma aplicação RV. Não é preciso se preocupar com a criação de código relacionado a interação pois ele já está embutido no *VrInteraction*. Também não é preciso pensar em um tratamento específico para diferentes ambientes RV uma vez que os manipuladores e mapeamentos foram criados para funcionar tanto em desktop quanto em ambientes imersivos. Isso permite que o *VrInteraction* tenha uma interface mínima, onde a aplicação precisa apenas informar que dispositivo e manipulador deve ser usado em um dado momento. Isso contribui para minimizar o custo de criação de novas aplicações 3D, ou a transformação de aplicações legadas em aplicações imersivas. Por último, os mapeamentos pré-definidos padronizam a interação nas aplicações que o utilizam. Do ponto de vista do usuário, isso simplifica o aprendizado tornando-o mais rápido e eficaz uma vez que ele não terá que aprender um novo sistema de interação ao usar uma aplicação diferente. O uso desses mapeamentos, entretanto, também é flexível: caso o desenvolvedor não queira usar um desses mapeamentos pré-definidos, é possível desativá-lo e registrar um próprio.

Quando a aplicação solicita a troca do manipulador corrente por outro, o *VrInteraction* se certifica que essa troca não produzirá uma descontinuidade na interação. Isso é feito através do uso de *transições*. Uma *transição* é uma interpolação entre duas poses diferentes de câmera que tem como objetivo garantir a fluidez da interação. Por exemplo, se o manipulador *rail* for escolhido, o *VrInteraction* criará uma transição que levará a câmera da posição atual até a posição inicial do trilho. A presença das transições é importante no sentido de impedir situação que possam causar desorientação no usuário.

Apesar de depender do módulo *VrInput*, o *VrInteraction* pode ter várias instâncias em uma mesma aplicação. Cada instância do *VrInteraction* fornece uma matriz de view, resultante da aplicação dos inputs dos dispositivos nos manipuladores. Com isso é possível controlar mais de uma câmera. Isso é uma característica que pode ser usada em aplicações RV colaborativas, onde cada usuário teria controle de uma câmera diferente.

G. VrWand

Ações como seleção de objetos ou opções em menus são realizadas comumente pelo mouse quando em ambiente desktop. Ambientes imersivos, entretanto, não permitem o uso do mouse e, dessa forma, dispositivos diferentes devem ser usados. Tais dispositivos, chamados de *wands*, são geralmente controles com sensores adaptados cuja posição e orientação podem ser determinadas por um sistema de rastreamento.

Através da orientação e posição de um desses dispositivos, aplicações podem obter um raio que pode ser usado como um apontador virtual 3D. O *VrWand* tem como função calcular a posição e orientação desse raio no espaço do mundo virtual definido pela aplicação. Através do *VrInput*, o *VrWand* obtém a posição e orientação da wand em coordenadas do sistema de tracker. Com a matriz de view fornecida pelo *VrInteraction* esses dados são transformados para coordenadas do mundo e repassados para a aplicação. Essa é responsável por realizar as ações seleção, assim como a renderização do raio.

IV. DISCUSSÃO

Nesta seção discutimos as principais características e contribuições do framework *LVRL*.

A. Interface de programação minimalista e transparente

Utilizando o framework por completo, a aplicação só tem contato com os componentes *VrInteraction*, *VrWand*, *VrInput* e *VrFrustum*. Na seção anterior foi demonstrado que todos elas têm os mesmos resultados estando em modo desktop e RV. Dessa forma, o desenvolvedor pode programar sua aplicação sem se preocupar com detalhes inerentes ao tipo de ambiente de visualização.

B. Mudança de desktop para imersivo em tempo de execução

Com o design do *VrInteraction* e do *VrFrustum* é possivel mudar de desktop para formato imersivo e vice-versa em tempo de execução apenas fazendo chamadas de métodos. Para reconfigurar as telas, basta carregar um novo arquivo de configuração no *VrFrustum*. Para usar um dispositivo de RV ao invés dos tradicionais mouse e teclado, é preciso somente dizer ao *VrInteraction* qual dispositivo usar.

C. Não intrusivo

Todas as bibliotecas descritas não tomam conta da aplicação. Elas podem ser consultadas sempre que a aplicação principal precisar. O desenvolvedor pode usar a engine gráfica que quiser e não é exigido que sua aplicação seja reescrita em função do LVRL. O controle de execução continua pertencendo à aplicação e não ao LVRL.

D. Multiplataforma

Todas as bibliotecas não têm nenhuma dependência do sistema e foram desenvolvidas com o objetivo de rodar em vários sistemas operacionais. Atualmente há suporte para Windows e Linux. A exceção reside apenas no *VrInput*, que depende dos drives do dispositivo. Dessa forma, alguns dispositivos não são suportados em todas as plataformas. Apesar dessa dependência, o design baseado em um gerenciador central e vários leitores cujo acoplamento são eventos baseados apenas em texto, permite que o *VrInput* seja usado em várias plataformas, mesmo quando um determinado dispositivo não possui driver para a plataforma.

E. Independência do Hardware

Com o uso do *VrInput* a presença ou não de um driver de dispositivos não influência no uso da biblioteca. Apenas o leitor daquele dispositivo não estará disponível.

F. Portável

A implementação do framework usa apenas tipos nativos do C++. Dessa forma, conseguimos portá-lo facilmente para C e Unity3D, mostrando assim que a arquitetura do framework não impõe nenhuma restrição a portabilidade entre as linguagens.

G. Compatível com renderização distribuída

Apesar da biblioteca não ter internamente primitivas de renderização distribuída, a configuração do VrFrustum fornece as ferramentas necessárias para se atingir esse objetivo.

V. Conclusões e trabalhos futuros

O LVRL, com sua abordagem minimalista e não intrusiva, demonstrou ser um framework com as características necessárias para ser usado na maioria das aplicações legadas de visualização científica do grupo. Por esse motivo, acreditamos que isso possa ser estendido a várias outras aplicações existentes. A arquitetura não intrusiva e a interface de programação transparente são as principais características que permitem que programadores fora do universo de Realidade Virtual convertam ou desenvolvam novas aplicações para ambientes imersivos.

O principal trabalho futuro a ser feito é um estudo de ergonomia e de usabilidade para alcançar a melhor forma de interação para os mapeamentos presentes no *VrInteraction*. Além disso, há estudos em andamento sobre novas formas de interação envolvendo dispositivos como kinect, smartphones e pads.

Agradecimentos

O Tecgraf/PUC-Rio é um grupo financiado principalmente pela Petrobrás.

Referências

[1] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, "The cave: audio visual experience automatic virtual environment," *Commun. ACM*, vol. 35, no. 6, pp. 64–72, Jun. 1992.

[2] T. DeFanti, D. Acevedo, R. Ainsworth, M. Brown, S. Cutchin, G. Dawe, K.-U. Doerr, A. Johnson, C. Knox, R. Kooima, F. Kuester, J. Leigh, L. Long, P. Otto, V. Petrovic, K. Ponto, A. Prudhomme, R. Rao, L. Renambot, D. Sandin, J. Schulze, L. Smarr, M. Srinivasan, P. Weber, and G. Wickham, "The future of the cave," *Central European Journal of Engineering*, vol. 1, pp. 16–37, 2011.

[3] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira, "Vr juggler: A virtual platform for virtual reality application development," in *Proceedings of the Virtual Reality 2001 Conference (VR'01)*. Washington, DC, USA: IEEE Computer Society, 2001.

[4] J. Gascón, J. M. Bayona, J. M. Espadero, and M. A. Otaduy, "Blendercave: Easy vr authoring for multi-screen displays," *SIACG 2011: V IBERO-AMERICAN SYMPOSIUM IN COMPUTER GRAPHICS*, 2011.

[5] F. Wang, "Research on virtual reality based on eon studio," in *Proceedings of the 2010 Fourth International Conference on Genetic and Evolutionary Computing*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 558–561.

[6] R. Kuck, J. Wind, K. Riege, and M. Bogen, "Improving the avango vr/ar framework: Lessons learned," 2008.

[7] Dassault Systemes, "3DVIA Virtools," http://www.virtools.com, April 2012.

[8] C. Anthes, M. Satomi, A. Wilhelm, C. Sommerer, and J. Volkert, "Space trash : An interactive networked virtual reality installation." in *11th Virtual Reality International Conference (VRIC 09)*, Laval, France, April 2009, pp. 107–118.

[9] A. Raposo, I. H. F. dos Santos, L. P. Soares, G. N. Wagner, E. T. L. Corseuil, and M. Gattass, "Environ: Integrating vr and cad in engineering projects," *IEEE Computer Graphics and Applications*, vol. 29, no. 6, 2009.

[10] Y. Zhou, M. Garland, and R. Haber, "Pixel-exact rendering of spacetime finite element solutions," in *Proceedings of the conference on Visualization '04*, ser. VIS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 425–432.

[11] S. Rusinkiewicz and M. Levoy, "Qsplat: a multiresolution point rendering system for large meshes," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 343–352.

[12] P. Silva, M. Machado, and M. Gattass, "3d seismic volume rendering," in *Eighth International Congress of The Brazilian Geophysical Society*, 2003.

[13] "Openscenegraph," http://www.openscenegraph.org.

[14] "Opensg," http://www.opensg.org/.

[15] "Opengl performer," http://oss.sgi.com/projects/performer/.

[16] C. Cruz-Neira, "Virtual reality based on multiple projection screens: The cave and its applications to computational science and engineering," Chicago, Illinois, USA, 1995.

[17] R. M. Taylor, II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, "Vrpn: a device-independent, network-transparent vr peripheral system," in *Proceedings of the ACM symposium on Virtual reality software and technology*, ser. VRST '01. New York, NY, USA: ACM, 2001, pp. 55–61.

[18] G. Humphreys and P. Hanrahan, "A distributed graphics system for large tiled displays," in *Proceedings of the conference on Visualization '99: celebrating ten years*, ser. VIS '99. Los Alamitos, CA, USA: IEEE Computer Society Press, 1999, pp. 215–223.

[19] S. Eilemann, M. Makhinya, and R. Pajarola, "Equalizer: A scalable parallel rendering framework," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 3, pp. 436–452, May 2009.