2
Realidade Virtual

Com aplicação em grande parte das áreas do conhecimento, entre elas a medicina, mecânica, treinamento militar, ergonomia, jogos e entretenimento, e com um grande investimento das indústrias na produção de software e hardware para dispositivos de entrada e saída, a realidade virtual vem experimentando um desenvolvimento acelerado nos últimos anos e indicando perspectivas bastante promissoras para os diversos segmentos vinculados com a área.

Um dos benefícios dos ambientes de RV é a capacidade de prover perspectivas vantajosas, impossíveis de serem obtidas no mundo real, como por exemplo, a navegação por dentro do corpo humano ou a análise de simulações físicas ou reações químicas em tempo real. 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.

As aplicações de RV se diferenciam das aplicações convencionais pelos tipos de dispositivos que utilizam e pela característica presencial ou imersiva do usuário no sistema. O usuário se sente imerso quando as sensações e eventos que acontecem no mundo virtual são sentidos por ele no mundo real. Atualmente se desenvolvem pesquisas buscando atuar sobre as percepções visuais, auditivas e tácteis.

Quanto à visão, o objetivo principal é fornecer ao usuário a sensação de profundidade, que não deve ser confundida com a noção de profundidade obtida pela distorção de perspectiva. A distorção de perspectiva permite distinguir objetos que estão próximos dos que estão mais distantes, enquanto a sensação de profundidade permite que o usuário seja imerso no ambiente virtual.

Para atingir esse objetivo, o sistema deve gerar, ao mesmo tempo, duas imagens diferentes, correspondendo às visões dos olhos esquerdo e direito. O cérebro humano processa a diferença entre essas duas imagens gerando uma representação precisa da posição e da forma tridimensional do objeto dentro da cena. Essa é a profundidade tridimensional que experimentamos no mundo real.

Esse tipo de geração de imagem é chamado de visão estereoscópica ou estereoscopia [23].

2.1. Sistemas de Realidade Virtual

Os sistemas de realidade virtual são compostos por dispositivos de entrada (rastreadores de posição, reconhecimento de voz, *joysticks* e mouses, por exemplo) e saída (visual, auditiva e táctil) capazes de prover ao usuário imersão, interação e envolvimento [18]. A idéia de imersão está ligada ao 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 com algum parceiro. A realidade virtual tem potencial para os dois tipos de envolvimento ao permitir a exploração de um ambiente virtual e ao propiciar a interação do usuário com um mundo virtual dinâmico.

Além do hardware, os sistemas de realidade virtual necessitam do software cuja composição é geralmente formada por três componentes: a componente de apresentação, de interação e de aquisição de dados.

A componente de apresentação é responsável pela representação, envolvendo aspectos de interface homem-computador e semiótica [23], carregamento do modelo e saída do sistema.

O componente de interação é responsável pela manipulação e navegação no ambiente. A dificuldade nesse caso é que, ao contrário dos ambientes convencionais, existem poucas metáforas de interação bem definidas para ambientes imersivos.

A aquisição dos dados é a componente mais complexa do sistema de RV. O sistema pode ser criado para funcionar especificamente com um determinado dispositivo de saída (um monitor, por exemplo) e um determinado dispositivo de entrada (um mouse 3D) ou ser escalável e funcionar com qualquer composição de hardware. Nesse caso, a componente de aquisição de dados deve ser capaz de suportar a vasta gama de dispositivos, inclusive os que ainda não foram criados.

Geralmente, os sistemas escaláveis utilizam bibliotecas ou *frameworks* que permitem que o desenvolvedor abstraia a aplicação dos dispositivos utilizados.

2.2. Dispositivos de Rastreamento

Uma das conseqüências do advento da realidade virtual foi a necessidade de se redefinir o paradigma de interface homem-computador. O sistema tradicional mouse-teclado-monitor foi substituído por dispositivos que permitiram maior imersão do usuário no ambiente virtual e o manuseio de todas as potencialidades dessa nova tecnologia.

O modo como os participantes interagem com o sistema de realidade virtual influencia enormemente suas experiências no ambiente virtual, facilitando seu uso, aumentado a sensação de imersão e ampliando a variedade de ações que se pode tomar dentro do ambiente virtual.

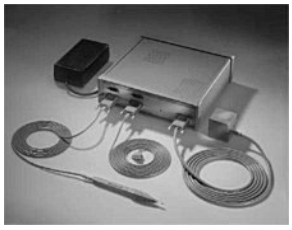
Um importante dispositivo de interação é o rastreador de posição que pode ser utilizado para acompanhar a posição do corpo e os movimentos do usuário, assim como a posição de outros objetos sendo por ele utilizados.

Existe uma variedade de dispositivos de rastreamento, cada um utilizando uma tecnologia diferente, entre eles, os eletromagnéticos, mecânicos, acústicos, inerciais e ópticos. Ao analisar as tecnologias utilizadas pelos rastreadores, três fatores devem ser levados em consideração: precisão e velocidade de resposta do sensor; interferência do meio; restrições (fios, conexões mecânicas, etc.):

- **Eletromagnéticos**
  - Princípio de funcionamento: os rastreadores eletromagnéticos utilizam campos magnéticos para medir posição e orientação. O sistema é composto por transmissor e receptor em forma de bobina. Um sensor unidimensional para estimar a posição no eixo Z, por exemplo, é composto por uma única bobina transmissora orientada na direção Z. Quando uma corrente é aplicada à bobina, um campo magnético é gerado. No receptor, o campo induz uma voltagem máxima proporcional à intensidade do campo magnético medido em uma bobina orientada na mesma direção do campo. A voltagem

induzida fornece a distância do transmissor ao receptor, assim como a diferença de alinhamento entre os eixos.

- Precisão/Velocidade: esses sistemas são bastante precisos, cerca de 1 a 2 mm para posição e 0,1º para orientação. A velocidade de captura de dados é de 100 a 200 medidas/segundo.
- Interferência do meio: a presença de metais e o próprio tubo de raios catódicos do monitor podem causar interferência eletromagnética.
- Restrições: pequeno espaço de utilização devido ao alcance do campo magnético gerado. O receptor deve estar cerca de 1-3 metros do transmissor, não havendo necessidade de linha de visada desobstruída.
- Exemplos: FasTrack da Polhemus (Figura 2i) e o Flock of Birds da Ascension (Figura 2ii).



(i)

(ii)

Figura 2: Dispositivos eletromagnéticos: (i) FasTrack e (ii) Flock of Birds.

➢ **Mecânicos**

- Princípio de funcionamento: os rastreadores mecânicos medem ângulos e distância entre juntas. Dada uma posição conhecida, todas as outras podem ser determinadas pela relação entre as juntas. Os rastreadores podem estar presos ao chão ou anexos ao corpo do usuário, usualmente na forma de um exoesqueleto. As rotações e as distâncias podem ser medidas por engrenagens, potenciômetros ou sensores de dobra (Figura 3).

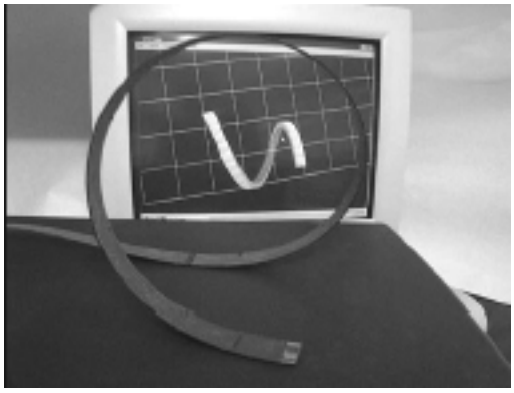
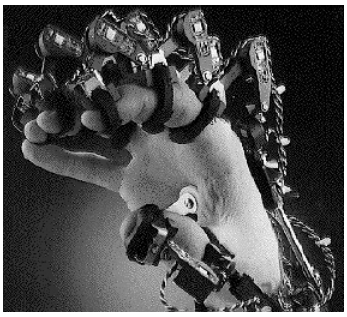


Figura 3: Sensor de dobra.

Uma vantagem dos sistemas mecânicos é a facilidade de adição da funcionalidade de *force feedback*. Esse tipo de dispositivo restringe o movimento do usuário aplicando uma força contrária ao seu movimento.

- Precisão/Velocidade: por serem mecânicos, possuem alta precisão (0,1º de rotação). A latência média é de 200 ms.
- Interferência do meio: não sofrem interferência do meio.
- Restrições: a própria arquitetura do rastreador pode restringir o movimento do usuário, caso o mesmo seja preso ao chão ou possua muitas juntas.
- Exemplos: o Phantom (Figura 4i) da Sensable Technologies e a mão mecânica da EXOS (Figura 4ii).

(i)



(ii)

Figura 4: Dispositivos mecânicos: (i) Phantom da Sensable e (ii) mão mecânica da EXOS.

➢ **Acústicos**

- Princípio de funcionamento: rastreadores acústicos utilizam, tipicamente, ondas sonoras ultra-sônicas para medir distância. Os

métodos mais usados são o cálculo do tempo de vôo e a coerência de fase. Em ambos, o objetivo é converter tempo em distância.

Um único par transmissor/receptor fornece a distância do objeto em relação a um ponto fixo. O resultado é uma esfera em cuja superfície o objeto está localizado. Como visto na Figura 5, a adição de um segundo receptor restringe a região a um círculo e um terceiro receptor restringe a dois pontos, sendo um deles geralmente descartado. Portanto, para estimar a posição são necessários um transmissor e três receptores ou um receptor e três transmissores. Para estimar posição e orientação, são necessários três transmissores e três receptores.

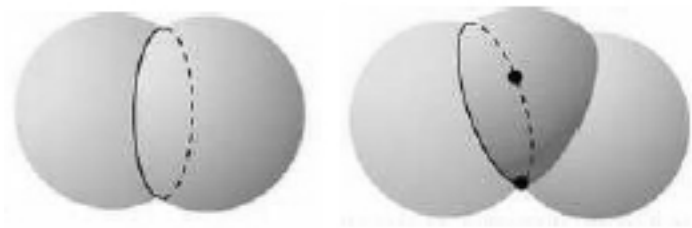


Figura 5: Interseção entre duas esferas (um círculo) e entre três (dois pontos).

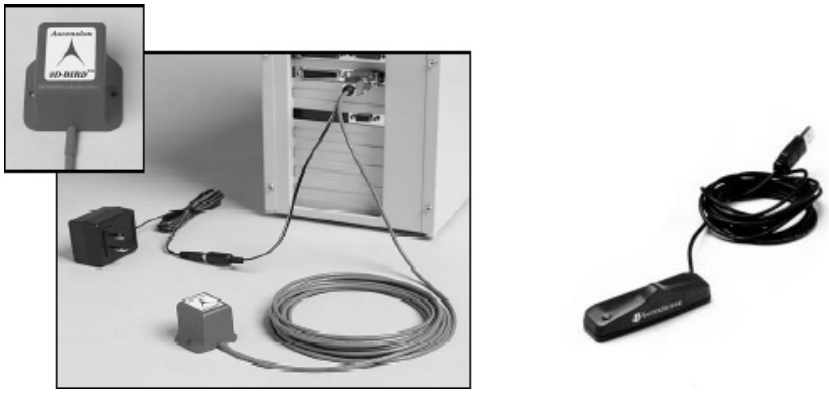
- Precisão/Velocidade: existe um atraso inerente a espera do sinal. Esse atraso é intensificado devido à baixa velocidade de propagação do som.
- Interferência do meio: as propriedades do som limitam esse método. O desempenho é degradado na presença de um ambiente ruidoso ou devido a geração de ecos. O som deve percorrer um caminho sem obstrução entre os alto-falantes e os microfones.
- Restrições: a configuração do sistema não é cara, pois o equipamento necessário é composto de microfones, alto-falantes e um computador.

  Devido às restrições de interferência, a distância média entre receptor e transmissor são alguns metros, contudo, sistemas mais precisos podem cobrir áreas de até 40x30m.
- Exemplos: Logitech Tracker (Figura 6i) com alcance médio de 15 metros e o FarReach da Infusion Systems (Figura 6ii) com alcance de 12 metros.

(i) (ii)

Figura 6: Dispositivos acústicos: (i) Logitech Tracker e (ii) FarReach.

- **Inerciais**
  - Princípio de funcionamento: utilizam magnetômetros passivos, acelerômetros e girômetros. Os magnetômetros passivos medem o campo magnético do ambiente (geralmente da Terra) e fornecem medidas angulares. Os girômetros fornecem medidas angulares mais precisas e os acelerômetros fornecem medidas lineares. Todos são baseados na segunda lei de movimento de Newton, sendo assim, o sistema deve integrar a leitura para obter a velocidade e a posição.
  - Precisão/Velocidade: Devido à etapa de integração, o erro obtido a cada passo tende a aumentar. A utilização de filtros de correção e outros sensores ajuda a diminuir esse erro.
  - Interferência do meio: Não existe interferência, pois o sistema é autocontido, não havendo necessidade de um ponto externo para obtenção de dados;
  - Restrições: Não existe limitação física para o espaço de trabalho, sendo o mesmo limitado somente pela conexão entre o dispositivo e o computador.
  - Exemplos: 3D-Bird da Ascension Technology (Figura 7i) e o Intertrax2 da InterSense (Figura 7ii).



(i) (ii)

Figura 7: Dispositivos inerciais: (i) 3D-Bird e (ii) Intertrax2.

- **Ópticos**
  - Princípio de funcionamento: baseado na análise da projeção bidimensional de uma imagem ou na determinação dos ângulos de feixes da varredura para calcular a posição e orientação de um dado objeto. Os sensores ópticos são geralmente câmeras (por exemplo, CCD), um detector 4Q ou um diodo de efeito lateral.

    Um CCD é um conjunto de detectores recebendo imagens no plano focal da câmera. Um detector 4Q é um componente plano capaz de gerar sinais especificando o centro do feixe de luz que incide em sua superfície. Um diodo de efeito lateral é um componente que gera um sinal proporcional à posição da luz chegando em um eixo.

    Quando o sensor utilizado é uma câmera, técnicas de visão computacional devem ser utilizadas para determinar a posição do objeto. Se somente uma câmera for utilizada, é possível determinar um segmento de reta que passa pelo objeto detectado e pelo centro de projeção da câmera. Usando mais de uma câmera, pode-se determinar a posição e orientação do objeto.
  - Precisão/Velocidade: a velocidade de captura depende muito do sensor empregado. Uma câmera padrão NTSC consegue capturar imagens a taxas de 30 quadros por segundo, limitando a amostragem, enquanto câmeras digitais podem capturar a taxas de 200 a 1000 quadros por segundo. A precisão dos dados depende das técnicas de visão computacional empregadas: calibração de câmera,

extração de informação da imagem e utilização de filtro para evitar tremidos.

- Interferência do meio: O laser e outros emissores podem refletir em objetos próximos atrapalhando a medição.
- Restrições: A câmera deve estar sempre enxergando o objeto sendo rastreado e o emissor de luz não pode estar obstruído. Uma solução com três ou quatro câmeras oferece redundância e permite que uma ou duas sejam bloqueadas antes do sistema deixar de funcionar.
- Exemplos: Existem muitos equipamentos ópticos utilizados em captura de movimento [25], porém poucos em realidade virtual. Alguns exemplos de dispositivos usados em RV são o DragonFly [24] (Figura 8), o LaserBird da Ascension Technology (Figura 9) e o produto final desta tese, descrito no capítulo 3.

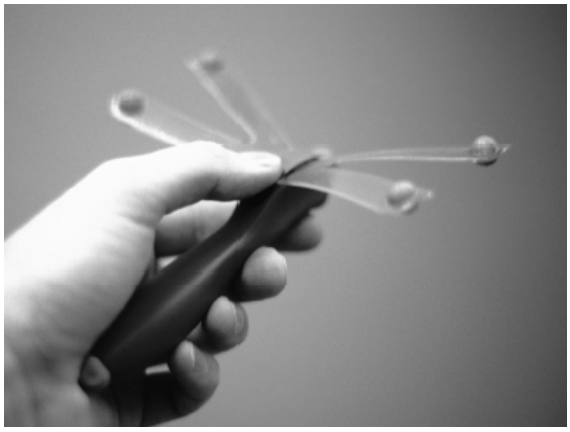


Figura 8: DragonFly - dispositivo óptico desenvolvido pelo Instituto Fraunhofer de Engenharia Industrial da Alemanha.

Figura 9: LaserBird da Ascension Technology: sensor e feixe de laser.

2.3. Ferramentas

Os sistemas de RV precisam de um software para apresentar a aplicação para o usuário final. Os softwares de RV são complexos, pois seus dispositivos são complexos. A fim de facilitar a programação dessas aplicações, foram criadas várias soluções com o intuito de prover um ambiente único de desenvolvimento.

Um ambiente de desenvolvimento permite que o usuário se concentre na programação da aplicação ao invés de se preocupar com a gerência do ambiente de RV. Ele define uma arquitetura que inclui componentes para gerência de dispositivos de entrada, apresentação das saídas visual, auditiva e outras, além de um mecanismo de configuração. Alguns ambientes podem ainda estender a arquitetura a fim de gerenciar recursos como alocação de memória, multi-tarefas e comunicação.

Várias ferramentas para desenvolvimento de aplicações em RV existem disponíveis comercialmente ou gratuitamente, entre elas, o Avango [27], Vess [8], Diverse [1], VRJuggler [5] e o ViRAL (*Virtual Reality Abstraction Layer*), sendo sendo as duas últimas analisadas neste trabalho.

2.3.1. VRJuggler

O VRJuggler é uma biblioteca de realidade virtual, com código aberto, multi-plataforma e independente de dispositivo. Ela fornece uma abstração dos dispositivos de entrada e saída.

O VRJuggler é composto de vários módulos, sendo *vrj*, *jccl*, *gadgeteer* e *vpr* os principais. O módulo *vrj* é responsável pela integração de todos os módulos. Ele fornece uma camada acima do sistema operacional, permitindo que uma aplicação desenvolvida com o VRJuggler possa executar em diferentes sistemas, não só operacionais, mas em diversos sistemas de RV, como em um monitor, em uma CAVE ou em um HMD.

O módulo *jccl* é um sistema de configuração baseado em XML. O usuário utiliza arquivos de configuração para definir quais os dispositivos de entrada serão utilizados e a disposição dos dispositivos de saída. Baseado nos arquivos de configuração, a biblioteca carrega os módulos de controle (*drivers*) adequados. Os

arquivos de configuração possuem um meta-arquivo para que o sistema de configuração identifique os parâmetros dos dispositivos sendo carregados.

Como o sistema está baseado em meta-arquivo, não é possível estender o tipo das propriedades dos dispositivos. Com isso, os únicos tipos que podem ser utilizados são os que o VRJuggler fornece, como inteiro, real, vetor, caracteres e alguns tipos abstratos de dados.

A biblioteca permite que novos dispositivos sejam criados pelo desenvolvedor, porém, ele não é capaz de incorporar esses novos dispositivos na interface de configuração, a não ser que o meta-arquivo seja modificado e a descrição do novo dispositivo inserida a ele.

O módulo *gadgeteer* é responsável pela gerência de dispositivos. Ele trata da configuração, controle, aquisição e representação do dado dos dispositivos de RV. Novos dispositivos são criados através da implementação de uma ou mais interfaces ilustradas na Figura 10.

A classe *vjPosition* representa um dispositivo que fornece como dado uma matriz de transformação. Um objeto *vjDigital* é um dispositivo que possui dados binários, do tipo 'verdadeiro' ou 'falso'. Um exemplo desses dispositivos são os botões do *mouse*. Um objeto *vjAnalog* é um dispositivo que possui informações reais contínuas.

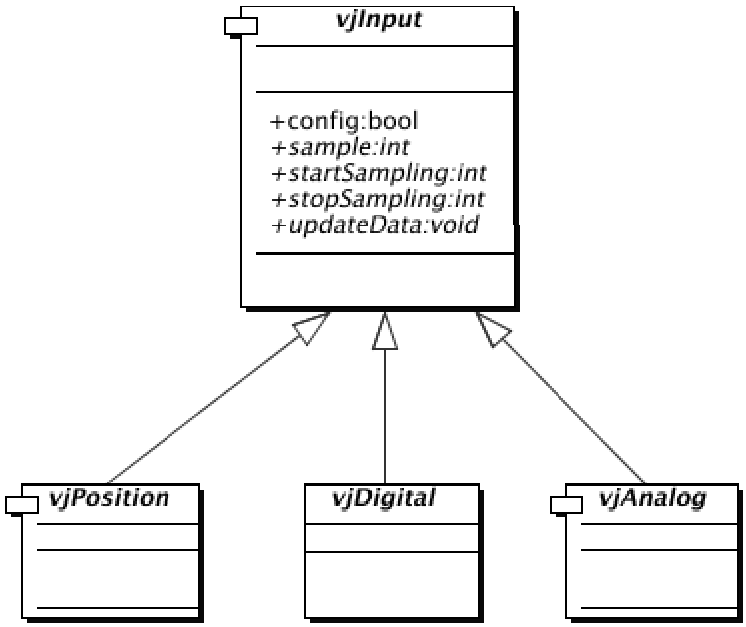


Figura 10: Classes de dispositivo do VRJuggler.

Um *joystick*, por exemplo, pode ser criado herdando sua interface das classes *vjPosition* e *vjDigital*, pois o mesmo é capaz de prover informações de posição e digitais, através de seus botões.

O método *config* é chamado durante a leitura dos arquivos de configuração. O método *sample* é onde as leituras dos dispositivos são feitas. A leitura é feita é uma *thread* separada e os dados são armazenados para uso futuro. Os métodos *startSampling* e *stopSampling* são responsáveis por iniciar e parar a execução dessa *thread*. O método *updateData* atualiza os dados do dispositivo, ou seja, recupera os dados que a *thread* estava armazenando e disponibiliza este dado para a aplicação. Esse método, portanto, deve ser *thread-safe*.

Finalmente, o módulo *vpr* é responsável por prover uma abstração independente de plataforma para *threads*, *sockets* (TCP/UDP) e comunicação serial.

Uma aplicação no VRJuggler é um objeto e sua classe deve derivar da interface de aplicação da biblioteca. A Figura 11 ilustra as principais interfaces de aplicação disponíveis nativamente no VRJuggler.

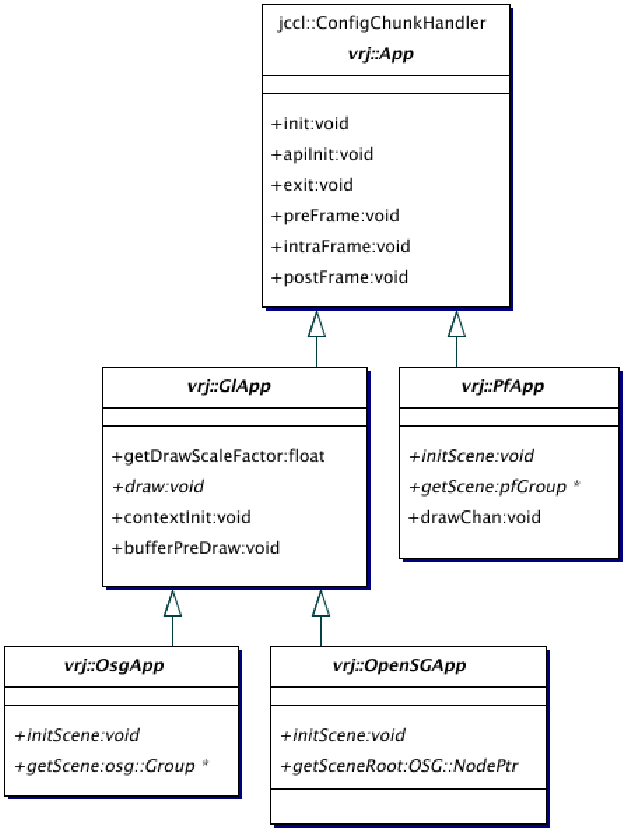


Figura 11: Hierarquia de classes de aplicação do VRJuggler.

A classe *vrj::App* é a interface que define os principais métodos que uma aplicação deve implementar. A classe *vrj::GlApp* é, também, uma interface para uma aplicação que irá utilizar OpenGL [19], e a classe *vrj::PfApp* é uma interface para uma aplicação que irá utilizar OpenGL Performer [20]. Derivadas de *vrj::GlApp* estão as aplicações que utilizam grafos de cena para renderização, como a *vrj::OsgApp* que utiliza o OpenSceneGraph [21].

Ao definir a interface da aplicação, o VRJuggler tem controle do laço principal aplicação. A função ***main*** de uma aplicação desenvolvida com o VRJuggler está descrita na Tabela 1.

```
int main( int argc, char *argv[] )
{
1 vrj::Kernel *kernel=vrj::Kernel::instance();  // pega o kernel
2  simpleApp *app = new SimpleApp(); // Cria o objeto da aplicação
3  kernel->loadConfigFile(argv);     // Configura o kernel
4  kernel->start();                  // Inicia a thread do kernel
5  kernel->setApplication( app );    // Passa a apl. para o kernel
   kernel->waitForKernelStop();    // Bloqueado até o kernel parar
   return 0;
}
```

Tabela 1: Função main de uma aplicação utilizando o VRJuggler.

Na linha 1 é retornado o objeto núcleo (*kernel*) do VRJuggler. Na linha 2, um objeto da aplicação (S*impleApp*) é instanciado. Na linha 3, os arquivos de configuração são passados para o objeto *kernel* para que ele se configure.

Na linha 4, o VRJuggler começa a ser executado. Nesse momento, o *kernel* cria uma nova *thread* de execução e começa o seu processamento interno.

Qualquer mudança de estado do *kernel* a partir de agora exige uma reconfiguração do VRJuggler, seja através da inclusão de novos arquivos de configuração ou da mudança da aplicação ativa. Isso permite que o VRJuggler seja reconfigurado em tempo de execução, porém, somente para adição de novos dispositivos. Os dispositivos que já estavam sendo utilizados não podem ser modificados.

A aplicação que será executada pelo *kernel* foi configurada na linha 5.

A Tabela 2 apresenta algumas vantagens e desvantagens no uso da biblioteca VRJuggler:

Vantagens	Desvantagens
• Código aberto; • Multi-plataforma, multi-dispositivos de RV; • Abstração dos dispositivos através de arquivos de configuração; • Desenvolvimento da aplicação é simples, bastando criar uma classe que derivará de alguma das interfaces ilustradas na Figura 11; • Compilação de uma nova aplicação é bastante rápida (tendo já os arquivos binários pré-compilados do núcleo do VRJuggler).	• O arquivo de configuração não é fácil de ser editado manualmente; • A interface de edição de arquivos de configuração, desenvolvida em Java, é complicada e não permite que novos dispositivos sejam configurados através dela; • A compilação dos módulos do VRJuggler é bastante complicada, exigindo que sejam sempre utilizados os arquivos binários pré-compilados; • O VRJuggler depende de muitas bibliotecas externas, tornando ainda mais difícil a sua compilação; • O usuário não possui controle do laço principal da aplicação; • Uma aplicação desenvolvida com o VRJuggler é completamente imersiva, ou seja, não é possível utilizar ao mesmo tempo o VRJuggler e um sistema de diálogos convencional (menus, listas, botões, etc.); • Os eventos gerados pelos dispositivos de entrada são pré-definidos e não podem ser estendidos; • Reconfiguração em tempo de execução é limitada.

Tabela 2: Vantagens e desvantagens do VRJuggler.

2.3.2.
ViRAL (*Virtual Reality Abstraction Layer*)

O ViRAL é uma ferramenta desenvolvida em C++ e Qt [28], utilizada para facilitar o desenvolvimento de aplicações de RV. As aplicações que utilizam o ViRAL não precisam saber, por exemplo, em quantas janelas, com quantos usuários ou com quais dispositivos ela irá executar, porque o ViRAL abstrai o contexto onde elas serão executadas.

O ViRAL pode ser usado de duas maneiras: como a aplicação principal ou embutido. Ao ser executado como a aplicação principal, ele carrega as suas janelas e menus e os diversos *plugins* criados pelos desenvolvedores. Um *plugin* é um arquivo utilizado para alterar, melhorar ou estender as operações de uma aplicação principal (no caso, o ViRAL).

A segunda maneira de se utilizar o ViRAL é embuti-lo em uma aplicação. Ele funcionará como escravo e a aplicação principal será o mestre. O ViRAL possui seis sistemas que podem ser utilizados embutidos em uma aplicação, sendo eles, o de ambiente, usuário, janela, dispositivo, cena e *plugin*.

2.3.2.1.
Sistema de Cena

Uma Cena (os termos sublinhados se referem aos objetos dos sistemas da biblioteca) é uma aplicação gráfica que é carregada como um *plugin* pelo ViRAL, sendo que várias cenas podem ser carregadas e utilizadas simultaneamente.

O sistema de cenas mantém uma floresta de cenas carregadas, sendo que a raiz de cada árvore é a Cena propriamente dita e seus descendentes são chamados de Objetos de Cena. Cada Cena e Objeto de Cena possui uma interface de configuração que é carregada pelo sistema quando solicitada.

Quando o ViRAL é utilizado embutido, a aplicação principal implementará a sua interface gráfica convencional (botões, menus e janelas) e exportará para o ViRAL as Cenas e os Objetos de Cenas que serão carregados para exibição no ambiente virtual. Isso permite que a aplicação satisfaça os requisitos convencionais de um sistema, adicionando a eles novas funcionalidades que somente são úteis em um ambiente virtual.

2.3.2.2.
Sistema de Usuários

O termo Usuário no ViRAL se refere à "entidade" que está visualizando a cena, ou seja, ele é uma câmera no mundo virtual. Um Usuário pode ser a visão de uma pessoa real posicionada em uma CAVE, uma visão externa dessa pessoa, ou uma câmera móvel, por exemplo.

O Usuário possui como atributos a distância interocular (para visualização estereoscópica), a orientação da cabeça, e a posição e orientação do corpo. Outro atributo importante que o Usuário possui é a Cena da qual ele irá participar. Além disso, o Usuário pode estar acompanhado de várias superfícies de projeção.

2.3.2.3.
Sistema de Janela

Um *plugin* de Cena (ou uma aplicação mestre, no caso de uso embutido), pode ser executado em diversas configurações de Janelas. Para cada Janela está associada uma projeção. A projeção é a conexão da Janela com a Cena, pois uma projeção pertence a um Usuário e o Usuário contém uma Cena.

Uma Janela possui um atributo (canal) que define como será exibida a projeção: visão mono, do olho esquerdo, do olho direito ou em estéreo. A Figura 12 ilustra a interface de configurações de janelas no ViRAL.

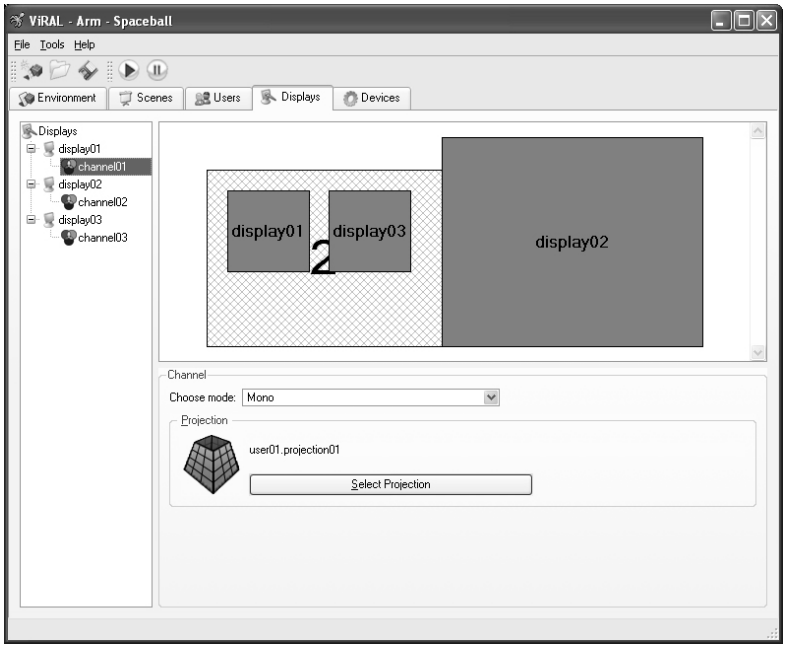


Figura 12: Interface de criação de janelas no ViRAL.

2.3.2.4. Sistema de *Plugin*

No ViRAL existem 3 tipos de *plugins*: de Dispositivo, de Cena e de Objeto de Cena. Cada um desses *plugins* herdam de uma classe específica do ViRAL e exportam seus métodos. O sistema de plugins é responsável pelo carregamento dinâmico e conexão de eventos.

Um dispositivo novo é criado como uma biblioteca de carregamento dinâmico. Nessa biblioteca, o desenvolvedor deve criar uma classe que herda sua interface da classe *vral::Device*. A Tabela 3 mostra a interface dessa classe e os principais métodos que devem ser implementados.

```
namespace vral
{
  class Device : public Object
  {
      virtual const Object * getSender() const;
      virtual const Object * getReceiver() const;
      virtual void dispatchChanges() = 0;
      virtual QWidget * loadPropertyFrame();
  };
}
```

Tabela 3: Interface da classe *vral::Device*.

De baixo para cima, o método *loadPropertyFrame* é responsável por retornar o diálogo (*Qwidget* do Qt) de configuração daquele dispositivo. A Figura 13 ilustra o diálogo de configuração de uma luva de realidade virtual.

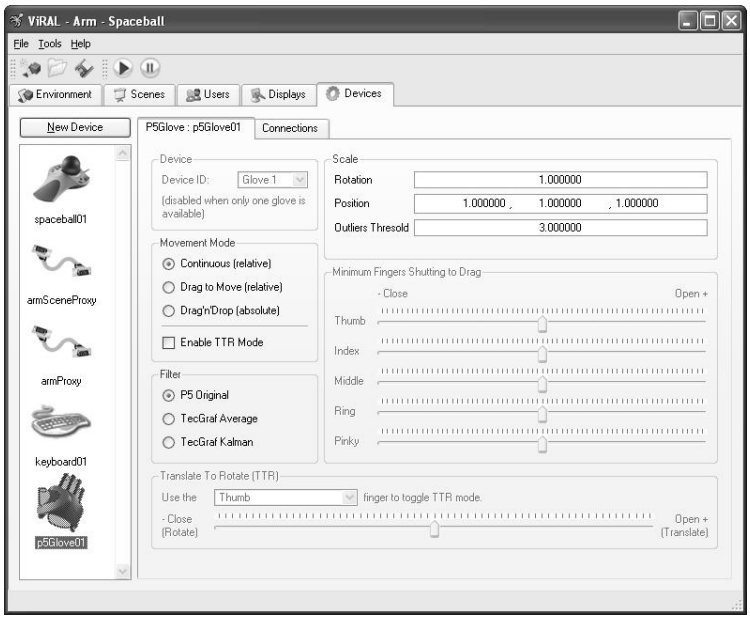


Figura 13: Configuração de uma luva de RV no ViRAL.

O método *dispatchChanges* transmite os eventos de um dispositivo, geralmente uma vez a cada passo de execução. O sistema de eventos no ViRAL utiliza o mecanismo de *signal/slot* do Qt. Esse mecanismo é utilizado para programação de componentes e comunicação entre objetos. Os sinais (*signals*) são emitidos por objetos quando os mesmo mudam de estado. As aberturas (*slots*) são utilizadas para receber sinais.

Os métodos *getSender* e *getReceiver* são utilizados para implementação de um mecanismo de redirecionamento (*proxy*). Um exemplo de uso desse mecanismo é quando se deseja transformar um Usuário do ViRAL em um Dispositivo. A partir de um dispositivo de redirecionamento, os eventos podem ser enviados a um Usuário. Dessa forma é possível modificar a orientação e a posição do usuário através de um dispositivo de RV.

Os *plugins* de Cena e de Objetos de Cena possuem o mesmo princípio de funcionamento dos *plugins* de Dispositivo, porém as interfaces desses *plugins* são definidas pelas classes *vral::Scene* e *vral::SceneObject*, como visto na Tabela 4.

```
namespace vral
{
  class SceneObject : public Object
  {
    virtual QWidget * loadPropertyFrame() = 0;
  };

  class Scene : public SceneObject
  {
    virtual bool isReady() const = 0;
    virtual bool checkTranslation() = 0;

    virtual void initializeContext() = 0;
    virtual void preDrawImplementation);
    virtual void drawImplementation();
  };
}
```

Tabela 4: Interface das classes *vral::Scene* e *vral::SceneObject*.

No ViRAL existe uma hierarquia que define quem é Cena e quem é Objeto de Cena. Uma Cena é a raiz dessa hierarquia e todos os seus filhos são objetos dessa cena. O método *loadPropertyFrame* carrega o diálogo de configuração do Objeto de Cena. *isReady* informa ao ViRAL que a Cena está pronta para ser executada. O método *checkTranslation* valida uma movimentação dentro do ViRAL.

As Cenas são responsáveis por configurar o estado do OpenGL para que o ViRAL possa utilizá-lo para renderização. Para isso existe o método

initializeContext. Os métodos *preDrawImplementation* e *drawImplementation* têm por finalidade atualizar a cena e renderizá-la, respectivamente. O ViRAL calcula automaticamente as matrizes para serem carregadas pelo OpenGL baseado nas configurações de projeção e do modo de exibição de um canal.

```
SimpleScene::SimpleScene() : Scene()
{
      mQuadricObj = gluNewQuadric();
      gluQuadricOrientation( mQuadricObj, GLU_OUTSIDE );
}
SimpleScene::~SimpleScene()
{
      gluDeleteQuadric( mQuadricObj );
}
void SimpleScene::initializeContext()
{
      glClearColor( 1.0f, 1.0f, 1.0f, 1.0f );
      glEnable( GL_CULL_FACE );
}
void SimpleScene::drawImplementation( const
                              vral::SceneDrawingData &data )
{
      glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
      glEnable( GL_LIGHTING );
      glEnable( GL_COLOR_MATERIAL );

      glMatrixMode( GL_PROJECTION );
      glLoadMatrixf( data.getProjection().get() );

      glMatrixMode( GL_MODELVIEW );
      glLoadMatrixf( data.getModelview().get() );

      glPushMatrix();
            glTranslatef( 0.0f, 0.0f, -50.0f );
            glColor3f( 1.0f, 0.0f, 0.0f );
            gluQuadricNormals( mQuadricObj, GLU_SMOOTH );
            gluSphere( mQuadricObj, 15.0, 32, 32 );
      glPopMatrix();
}
```

Tabela 5: Exemplo de uma cena simples criada no ViRAL.

A Tabela 5 mostra um exemplo de uma cena simples que implementa os métodos *initializeContext* e *drawImplementation*. A estrutura *SceneDrawingData* contém as matrizes de modelo e projeção que serão utilizadas para configurar a câmera do OpenGL.

2.3.2.5.
Sistema de Dispositivo

O sistema de dispositivo é responsável pela conexão e transmissão de eventos. Os eventos são conectados através da interface ilustrada na Figura 14. A cada quadro o ViRAL chama o método *dispatchChanges* dos seus *plugins*. Apesar dos *plugins* poderem funcionar em *threads* independentes, os dados só devem ser transmitidos mediante essa chamada de método.

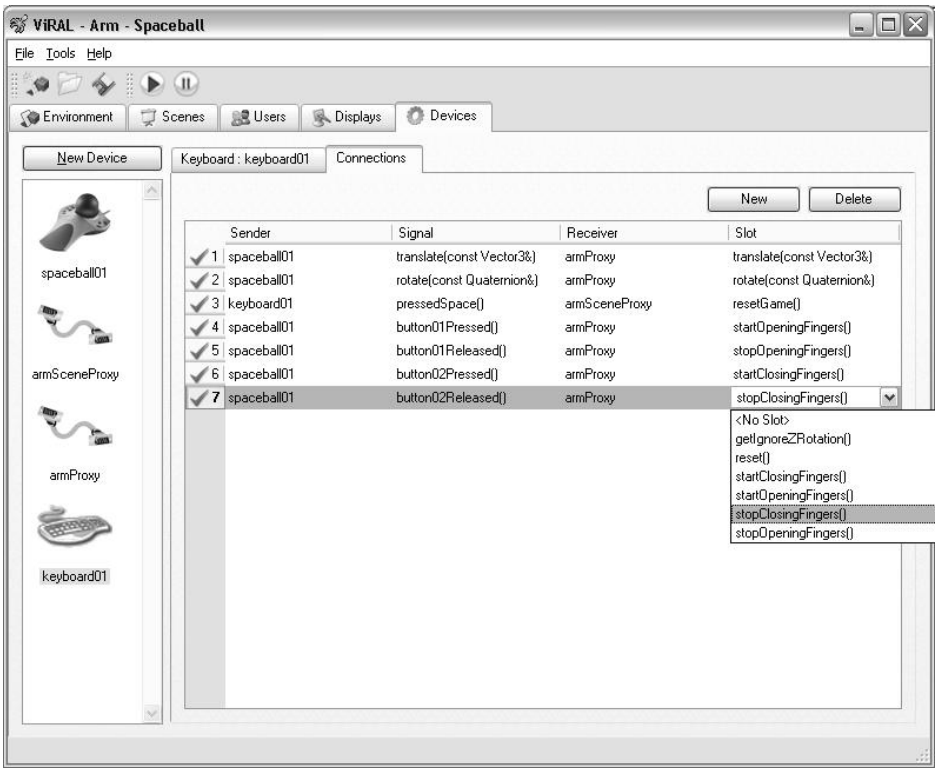


Figura 14: Interface de conexão de eventos no ViRAL.

Utilizando o mecanismo de *signal/slot* do Qt, o ViRAL permite que as conexões sejam configuradas em tempo de execução. Com isso, o usuário pode trocar, a qualquer momento, o dispositivo com o qual ele está interagindo com a aplicação.

2.3.2.6.
Sistema de Ambiente

O Sistema de Ambiente é responsável pela gravação e leitura de dados de uma configuração do ViRAL. Ele armazena os arquivos utilizando um formato XML comprimido.

Esse sistema visita todos os objetos instanciados dentro do ViRAL (dispositivos, janelas, cenas, etc.) e grava as suas informações utilizando um padrão de projeto chamado Memento [11]. Todo *plugin* criado para o ViRAL deve implementar dois métodos herdados de sua classe base. A Tabela 6 ilustra como um *plugin* de cena grava e recupera as informações de um arquivo. A classe *vral::DataNode* é a classe que implementa o padrão Memento.

```
class CaptureDevice : public vral::Device
{
   // Métodos para gravação e leitura de dados
   virtual void getMemento( vral::DataNode &node )
   {
      // inclui o memento do pai
      Object::getMemento( node );

      node.setInt( "gridSize", mGridSize );
      node.setInt( "cameraNumber", mCameraNumber );
   }
   virtual void setMemento( const vral::DataNode &node )
   {
      mGridSize = node.getInt( "gridSize" );
      mCameraNumber = node.getInt( "cameraNumber" );
   }

public:
   // Atributos que devem ser gravados
   int mCameraNumber;
   int mGridSize;
}
```

Tabela 6: *Plugin* de cena com métodos de gravação e leitura de arquivo.

2.3.2.7. Vantagens e Desvantagens

A Tabela 7 apresenta algumas vantagens e desvantagens da utilização do ViRAL, analisadas pelo perfil do desenvolvedor de novas funcionalidades. A utilização do Qt pelo ViRAL traz uma grande vantagem que é a rápida construção de interfaces gráficas, além de ser multi-plataforma.

Vantagens	Desvantagens
• Interface gráfica através do Qt; • Uso de sistema de *plugins*; • Uso embutido permite integrar aplicações *desktop* e aplicações imersivas; • Rápido desenvolvimento de novos *plugins*; • Criação de eventos arbitrários permitindo programação de componentes; • Reconfiguração em tempo de execução.	• Exige conhecimento de orientação a objeto, tornando o aprendizado mais demorado; • Novas aplicações ou *plugins* devem ser desenvolvidas com Qt, que é uma ferramenta comercial.

Tabela 7: Vantagens e desvantagens do ViRAL.