Capítulo 1

Tecnologias e Ferramentas para o Desenvolvimento de Sistemas de Realidade Virtual e Aumentada

Alexandre Cardoso[1], Edgard Lamounier Júnior[1], Claudio Kirner[2], Judith Kelner[3]

[1]Faculdade de Engenharia Elétrica – Universidade Federal de Uberlândia
Uberlândia - MG – Brasil

[2]Faculdade de Ciências Exatas e da Natureza – Universidade Metodista de Piracicaba (UNIMEP) – Piracicaba – SP - Brasil

[3]Grupo de Realidade Virtual – Centro de Informática– Universidade Federal de Pernambuco (UFPE) – Recife – Pernambuco – Brasil

alexandre@ufu.br, lamounier@ufu.br, ckirner@unimep.br,
jk@cin.ufpe.br

Abstract

Virtual reality and augmented reality are areas related to new user interface generation, making easy and improving the user interaction with computational applications. This chapter presents some

concepts related with virtual and augmented reality and the contents of this book.

Resumo

Realidade Virtual e Realidade Aumentada são áreas relacionadas com novas gerações de interface do usuário, facilitando e potencializando as interações do usuário com as aplicações computacionais. Este capítulo apresenta alguns conceitos de realidade virtual e aumentada e o conteúdo deste livro.

1.1. Introdução

"A interface ideal seria, naturalmente, um ambiente dentro do qual um computador pudesse controlar a existência da matéria". Esta frase dita por Ivan Sutherland em 1965 reflete o desejo dos pioneiros da Computação Gráfica Interativa.

Desde então, pesquisadores de diversos laboratórios do planeta têm concentrado esforços no sentido de alcançar esta forma fantástica de usar o computador. Hoje, temos o privilégio de vivenciar a concretização destes esforços, através do surgimento e estabelecimento da Realidade Virtual e da Realidade Aumentada.

Esta obra objetiva conjugar os conceitos e alguns dos elementos mais relevantes das tecnologias de Realidade Virtual e Aumentada, visando dar condições ao leitor de conhecer técnicas e desenvolver aplicações relacionadas com o tema.

Com esta contribuição, os editores esperam propiciar condições de melhoria de acesso às tecnologias disponíveis para o desenvolvimento de soluções de Realidade Virtual e Realidade Aumentada, preenchendo a lacuna relacionada com a falta de obras técnicas relativas ao assunto.

1.2. Realidade Virtual

Várias são as definições propostas para a Realidade Virtual, comumente chamada de RV pela comunidade científica. Porém, podemos visualizá-la como um sistema computacional usado para criar um ambiente artificial, no qual o usuário tem a impressão de não somente estar dentro deste ambiente, mas também habilitado com a capacidade de navegar no mesmo, interagindo com seus objetos de maneira intuitiva e natural. Portanto, a Realidade Virtual associa-se a uma interface homem-máquina poderosa, possibilitando ao usuário interação, navegação e imersão num ambiente tridimensional sintético, gerado pelo computador através de canais multisensoriais.

Assim, um usuário imerso em um ambiente de Realidade Virtual pode experimentar sensações nunca antes vividas por interfaces tradicionais. Por exemplo, alguém navegando em um ambiente virtual de uma sala, pode em um dado momento, naturalmente bater com uma mão em um vaso virtual que cairá e quebrará bem aos seus pés como se isso estivesse ocorrendo de verdade. Em outro momento, que não depende do vaso ter caído e quebrado, pode ligar uma TV e assistir uma programação. Esta abordagem se posta diferente de técnicas conhecidas de Computação Gráfica que tende a repetir sempre a mesma seqüência de animações.

Evolutivamente, durante os anos 90, a Realidade Virtual foi tida como uma nova e promissora forma de interface homem computador. Em paralelo, associava-se a mesma a diferentes periféricos (capacetes, luvas, rastreadores) de elevado custo e configuração. A redução de custo de hardware, o avanço das pesquisas e da demanda, têm ampliado o acesso à estas tecnologias e gerado expectativas de softwares e soluções que se apliquem às mais diferentes áreas. Áreas como Medicina, Indústria e outras têm mostrado casos de sucesso.

Vale ressaltar que o desenvolvimento de soluções de Realidade Virtual inclui aspectos como: integração de hardware, paradigmas de interação incertos, integração de vários conceitos e componentes visando sistemas de alto desempenho.

1.2.1. Classificações de Realidade Virtual

Nos últimos anos, surgiram diferentes propostas para classificação de Realidade Virtual em relação a diversos fatores. Esta seção apresenta uma classificação em relação à maneira como o usuário interage com o ambiente virtual, considerando os dispositivos multisensoriais usados ou não por ele. Nestes termos, pode-se categorizar RV essencialmente em duas frentes: Realidade Virtual Imersiva e Realidade Virtual Não-imersiva.

a) Realidade Virtual Imersiva

Tipo de Realidade Virtual que objetiva isolar o usuário por completo do mundo real. Para tanto, dispositivos especiais são usados para bloquear os sentidos do usuário (visão, audição, tato etc.) do mundo real e transferi-los para o ambiente virtual. Nestes ambientes, o usuário utiliza equipamentos como capacete de Realidade Virtual, luvas de dados, rastreadores e fones de ouvido a fim de responder somente aos estímulos gerados pelo sistema computacional.

b) Realidade Virtual Não-Imersiva

Tipo de Realidade Virtual onde o usuário tem acesso ao ambiente virtual, sem se isolar do mundo real, isto é, através de dispositivos convencionais de computador (tela e mouse). Neste contexto, alguns autores tem ainda sugerido o uso da expressão Realidade Virtual Semi-imersiva, onde o usuário é parcialmente isolado do mundo real (ou virtual).

Nestes sistemas, o controle do ambiente virtual é feito pelo usuário através de uma composição de dispositivos convencionais e não convencionais. Por exemplo, uma aplicação onde são usados óculos para compor uma imagem tridimensional a partir de duas vistas estereoscópicas, gerada numa tela convencional e sendo acessadas através de um mouse.

1.2.2. Aplicações de Realidade Virtual

Devido às potencialidades visualizadas pela Realidade Virtual, a diversidades de aplicações suportadas por esta tecnologia surge, naturalmente, em grande escala. Áreas como jogos, educação, medicina e engenharia têm sido profundamente explorados nos últimos anos. Porém, outras áreas que não apresentavam como fortes candidatos no início, tais como Psicologia, Artes etc., têm recebido uma expressiva atenção pela comunidade científica, ultimamente. Esta seção procura apresentar alguns ramos de aplicações da Realidade Virtual.

a) RV na Educação

Um grande benefício oferecido pela Realidade Virtual é que o conhecimento intuitivo do usuário a respeito do mundo físico pode ser utilizado para manipular o ambiente virtual, possibilitando ao usuário a manipulação de informações através de experiências próximas do real. Isso porque, no ambiente virtual, é possível criar a ilusão de mundo que na realidade não existe, através da representação tridimensional para o usuário.

Dessa forma, a Realidade Virtual tem potencial para propiciar uma educação como processo de exploração, descoberta, observação e construção de uma nova visão do conhecimento, oferecendo ao aprendiz a oportunidade de melhor compreensão do objeto de estudo. Essa tecnologia, portanto, tem potencial de colaborar

no processo cognitivo do aprendiz, proporcionando não apenas a teoria, mas também a experimentação prática do conteúdo em questão.

A utilização de RV com fins educativos tem merecido destaque e tem sido avaliada de forma intensiva nos últimos anos. Os resultados destas avaliações mostram ganhos, em termos de aprendizagem, superiores a diversas outras formas de interação visando educação mediada por computador.

Conclusivamente, pesquisadores apontam como principais vantagens da utilização de técnicas de RV para fins educacionais, os seguintes itens:

- Motivação de estudantes e usuários de forma geral, baseada na experiência de 1ª pessoa vivenciada pelos mesmos;
- Grande poderio de ilustrar características e processos, em relação a outros meios multimídia;
- Permite visualizações de detalhes de objetos;
- Permite visualizações de objetos que estão a grandes distâncias, como um planeta ou um satélite;
- Permite experimentos virtuais, na falta de recursos, ou para fins de educação virtual interativa;
- Permite ao aprendiz refazer experimentos de forma atemporal, fora do âmbito de uma aula clássica;
- Porque requer interação, exige que cada participante se torne ativo dentro de um processo de visualização;
- Encoraja a criatividade, catalisando a experimentação;
- Provê igual oportunidade de comunicação para estudantes de culturas diferentes, a partir de representações;
- Ensina habilidades computacionais e de domínio de perifé-

ricos.

b) RV na Medicina

A Medicina e áreas de saúde relacionadas têm se beneficiado, substancialmente, dos avanços tecnológicos apresentados pela Realidade Virtual, nos últimos anos. Pesquisadores acreditam que RV providencia um recurso ímpar para o ensino e treinamento em estruturas anatômicas. Um dos principais problemas para a educação em Medicina, em geral, é como providenciar um senso realístico da inter-relação entre estruturas anatômicas no espaço 3D. Com RV, o aprendiz pode repetidamente explorar as estruturas de interesse, separando-as ou agrupando-as com as mais diferentes formas de visualização, imersão e exploração. Isto seria, obviamente, impossível com um paciente vivo e é economicamente inviável manter com cadáveres em escolas de Medicina.

Projetos estão sendo desenvolvidos para suportar a cirurgia à distância. Os profissionais da Medicina podem, por exemplo, através de um ambiente virtual, controlar os braços de um robô para desenvolver uma cirurgia em um soldado, em um campo de batalha.

Realidade Virtual também tem sido utilizada para suportar localmente o treinamento de vários tipos de cirurgia como cirurgias endoscópicas, cirurgias de medula etc. É importante destacar, que estes aparelhos baseados em RV não só reduzem o custo de treinamento de cirurgiões, mas também reduzem os riscos cirúrgicos dos pacientes.

c) RV na Engenharia

A Realidade Virtual está se tornando a tecnologia habilitadora para a comunicação homem-máquina na indústria como suporte para uma melhor avaliação de projetos, engenharia concorrente

etc. Das diversas utilidades da Realidade Virtual em indústria e engenharia podemos destacar as facilidades providas por esta tecnologia para a criação de protótipos virtuais (*virtual prototyping*) e projetos baseados em simulação (*simulation-based design*).

Neste contexto, RV providencia para projetistas e engenheiros um ambiente onde se pode facilmente, e com velocidade, testar diversas alternativas de projeto em modelos virtuais. Isto a um custo e tempo muito mais baixos, quando comparados com os mesmos testes feitos com modelos reais. Além disso, este princípio de trabalho permite aos profissionais da indústria testar a exeqüibilidade de um projeto de uma maneira mais eficiente e, principalmente, nos primeiros estágios de um projeto. Isto é muito útil no sentido de prever mudanças em estágios finais de um modelo, caso este onde o impacto financeiro do projeto é expressivamente alto.

1.3. Realidade Aumentada

1.3.1. Conceitos e Definições

Pode-se definir Realidade Aumentada – RA – como a amplificação da percepção sensorial por meio de recursos computacionais. Assim, associando dados computacionais ao mundo real, a RA permite uma interface mais natural com dados e imagens geradas por computador. Um sistema de RA deve prover ao usuário condições de interagir com estes dados de forma natural.

Comumente, as soluções de Realidade Aumentada envolvem a geração de elementos virtuais que são inseridos no ambiente real, de tal forma que o usuário crê que os mesmos são partes do meio na qual está inserido.

Uma das características mais importantes da RA é a modificação no foco da interação homem computador. Com uso de RA, a

interação não se dá com um único componente e/ou elemento localizado, mas, com o ambiente que circunda aquele que interage. Neste sentido, Realidade Aumentada faz uso da combinação de Realidade Virtual e Mundo Real, propiciando a melhoria da percepção do usuário e sua interação.

São características básicas de sistemas de RA:

- Processamento em tempo real;
- Combinação de elementos virtuais com o ambiente real;
- Uso de elementos virtuais concebidos em 3D.

Por tais características, a concepção de soluções de Realidade Aumentada necessita de componentes que permitam avaliar a posição de quem interage (registro do usuário), o ponto de vista e, gerar os elementos virtuais para, finalmente, combina-los com o mundo real, por meio de um sistema de projeção. Para tanto, os elementos reais e virtuais necessitam ser alinhados corretamente, um em relação ao outro (Azuma, 2001). Em alguns casos, de forma complicadora, tal alinhamento requer grande precisão de cálculos.

Assim, sistemas de Realidade Aumentada demandam hardware de captura da informações do meio onde está o usuário, software para geração, em tempo real, de elementos virtuais e hardware para mapear tais elementos no mundo real.

1.3.2. Sistemas de Realidade Aumentada

Os sistemas de realidade aumentada, relacionados com a percepção de imagens, podem ser classificados conforme o tipo de display utilizado [Azuma, 2001], dando origem a quatro tipos de sistemas:

1. Sistema de visão ótica direta;

2. Sistema de visão direta por vídeo;
3. Sistema de visão por vídeo baseado em monitor;
4. Sistema de visão ótica por projeção.

O sistema de visão ótica direta utiliza óculos ou capacetes com lentes que permitem o recebimento direto da imagem real, ao mesmo tempo em que possibilitam a projeção de imagens virtuais devidamente ajustadas com a cena real. Para conseguir tal estratégia, pode-se, por exemplo, usar uma lente inclinada que permita a visão direta e que reflita a projeção de imagens geradas por computador diretamente nos olhos do usuário

O sistema de visão direta por vídeo utiliza capacetes com microcâmeras de vídeo acopladas. A cena real, capturada pela microcâmera, é misturada com os elementos virtuais gerados por computador e apresentadas diretamente nos olhos do usuário, através de pequenos monitores montados no capacete. Neste caso, o sistema de projeção obtura a imagem real.

O sistema de visão por vídeo baseado em monitor utiliza uma webcam para capturar a cena real. Depois de capturada, a cena real é misturada com os objetos virtuais gerados por computador e apresentada em um monitor convencional.

O sistema de visão ótica por projeção utiliza superfícies do ambiente real, onde são projetadas imagens dos objetos virtuais, cujo conjunto é apresentado ao usuário que o visualiza sem a necessidade de nenhum equipamento auxiliar. Embora interessante, esse sistema é muito restrito às condições do espaço real, em função da necessidade de superfícies de projeção.

Os sistemas de visão direta são apropriados para situações onde a perda da imagem pode ser perigosa, como é o caso de uma pessoa andando pela rua, dirigindo um carro ou pilotando um avião.

Em locais fechados, onde o usuário tem controle da situação, o uso da visão por vídeo é adequado e não oferece perigo, pois em caso de perda da imagem, pode-se retirar o capacete com segurança, se for o caso. O sistema com visão por vídeo é mais barato e mais fácil de ser ajustado (Realidade Aumentada, 2007).

1.3.3. Aplicações de Realidade Aumentada

As aplicações de Realidade Aumentada têm, nos últimos anos, apresentado forte crescimento. A título de ilustração, podemos destacar algumas áreas de utilização:

1. Treinamento e apoio a tarefas complexas, como manutenção de máquinas, assistência em treinamento de manutenção e visualização de elementos escondidos;
2. Visualização de elementos construtivos, objetos ocultos, sinalização de ambientes e outras tarefas relativas a inserção de informações complementares em ambientes reais que possam auxiliar, por exemplo, a engenharia e a arquitetura;
3. Prospecção e mapeamento de dados por estimativa em ambientes reais;
4. Visualização de dados, de forma a permitir aprimoramento da interação e da análise dos mesmos;
5. Simulação;
6. Conferência com participantes remotos;
7. Entretenimento, como jogos apoiados por computador;
8. Arqueologia, provendo condições de visualização das condições de elementos danificados ou incompletos em condições relacionadas com
9. Educação, possibilitando a inserção de informações complementares e/ou relevantes ao cenário real.

1.4. Considerações sobre Realidade Virtual e Aumentada

Pode-se dizer que a Realidade Virtual encontra-se num estágio bem mais maduro (e mais barato) quando comparado com a idéia produzida pelos seus primeiros visionários na década de 60. Porém, apesar destes avanços, existem muitos desafios para RV que precisam ser considerados e vencidos:

- A simulação de ambientes mais realísticos. Por exemplo, existem cidades simuladas em RV onde se tem a sensação de que todas as casas acabaram de ser pintadas e as ruas estão todas bem limpinhas. Sabe-se que isto não é real! Em medicina, a simulação de órgãos humanos ainda está muito "poligonal" e neste, como em muitos outros, o realismo precisa se bastante fiel.
- Resposta mais imediata para os dados de entrada do usuário. A fim de providenciar a sensação de "estar lá" para o usuário, o tempo de resposta precisa ser melhorado. Ainda enfrenta-se o dilema que quanto melhor o objeto virtual trabalhado (mais polígonos e bons modelos de iluminação) mais tempo de máquina é necessário para refazer o modelo.
- Alto custo dos dispositivos multisensoriais. É fato que se vive um barateamento destes dispositivos. Porém, a realidade de alcance mundial, tanto para indústria como e principalmente para as universidades está longe do ideal mínimo de trabalho.
- Simulações de todos sentidos humanos em um ambiente virtual. Essencialmente, os sentidos de visão e audição estão bem avançados nos projetos de Realidade Virtual contemplados até agora. Entretanto, outros importantes sentidos tais como cheiro, tato (e por que não dizer paladar?) precisam ainda de profunda pesquisa no sentido de alcançar ambientes virtuais mais realísticos.

É improvável que venhamos a assistir a construção de ambientes virtuais com a máxima fidelidade em nosso tempo. Porém, pesquisadores estão trabalhando para criar ambientes virtuais que apresentem uma realidade virtual cada vez mais real. Para tanto, a qualidade de dispositivos visuais está cada vez mais aumentando em contraste com seu tamanho e peso.

Assim, espera-se que num futuro não tão distante a Realidade Virtual seja uma ferramenta de fácil acesso e de larga escala, transportando seres humanos para lugares nunca imaginados ou visitados antes.

Considerando Realidade Aumentada, vislumbra-se a possibilidade da interação homem computador dar se por meio de amplificação do ambiente real com elementos virtuais. A redução de custos de equipamentos de rastreamento, sistemas de comunicação móvel, sistemas de projeção entre outros possibilitará, em breve, as condições de obtenção da hiperrealidade.

Os principais desafios que se apresentam relacionam-se com:

- Captura e rastreamento do usuário, de seu ponto de vista e de interesse. Equipamentos como os GPS podem vir a ser fortes aliados neste processo. Ademais, com recursos da computação móvel e a redução do tamanho e peso dos computadores darão condições de acesso a informações, via redes GPRS que podem facilitar tal problema;
- Melhoria das soluções de erros do sistema de rastreamento: muitos sistemas utilizam pontos de vista estático, objetos estáticos ou ambos para equacionar tal problema. Assim, será necessário considerar um número maior de variáveis do ambiente real, visando gerar elementos virtuais com muito mais precisão, viabilizando o uso de RA para tarefas complexas, como as exigidas pela medicina;

- Soluções dos problemas de oclusão: um pequeno obstáculo que se apresenta ou aparece ao usuário pode dificultar sobremaneira a o rastreamento e o aumento da realidade.

Em seguida, o leitor deverá ter uma visão geral sobre as seções deste livro, envolvendo os aspectos de tecnologia e aplicações.

1.5. Aspectos Tecnológicos de Realidade Virtual e Aumentada

Considerando o avanço do desenvolvimento de sistemas de Realidade Virtual e Aumentada, a redução de custos dos equipamentos e o aumento significativo de pesquisadores nas diversas instituições de pesquisa brasileiras e internacionais, este livro, elaborado por diversos pesquisadores reconhecidos por suas atuações nas áreas afins tenciona elucidar as principais tecnologias disponíveis para o desenvolvimento de soluções de RV, RA (incluindo o desenvolvimento de jogos e aplicações interativas tridimensionais).

Para atingir tais objetivos, o mesmo foi dividido em quatro seções e em 10 capítulos. Em todos os capítulos, os autores procuraram, por meio de uma abordagem didática e prática, apontar os caminhos a serem trilhados para utilizar cada uma das tecnologias no desenvolvimento de sistemas de Realidade Virtual e Aumentada.

1.5.1. Fundamentos

A seção 1 apresenta fundamentos e é composta por este capítulo, onde os desafios que se oferecem para a solução de sistemas de RV e RA são apontados, bem como, de que forma tais soluções podem ser adequadas.

1.5.2. Linguagens e Ferramentas para Realidade Virtual

A seção 2 apresenta as linguagens e ferramentas para o desenvolvimento de sistemas de Realidade Virtual, contendo três capítulos.

No capítulo 2, sobre OpenGL (Open Graphics Library), os autores apresentam uma tecnologia multiplataforma e multi linguagem, de reconhecida importância, gratuita. OpenGL pode ser aplicada no desenvolvimento de soluções de RV, CAD, apresentação de dados em 3D, aplicações gráficas interativas 2D e 3D, imagens médicas e realidade virtual de forma geral.

Desde sua primeira versão em 1992, OpenGL tem sido usado intensivamente pela indústria e pela academia para concepção de aplicações relacionadas com as mais diferentes plataformas.

No capítulo 3, sobre VRML e X3D, os autores introduzem as vantagens de utilização de linguagens de modelagem de formas para construção de ambientes virtuais compatíveis com a Web. Uma apresentação do passo a passo de construção dos grafos de cena e um do conjuntos de nós disponíveis permite uma avaliação do potencial das linguagens.

A modelagem dos ambientes virtuais, usando tais linguagens permite, ao usuário, visualizar ambientes tridimensionais, movimentar-se dentro deles e manipular seus objetos virtuais.

Os objetos virtuais podem ser animados, apresentando comportamentos autônomos ou disparados por eventos (Cardoso, 2002).

No capítulo 4, é introduzida a linguagem Java3D. Por meio do paradigma de orientação a objetos e o uso da estrutura de grafos de cena, a API Java3D permite elaboração de ambientes virtuais

associados ao potencial da linguagem Java. Os autores apresentam a estratégia de desenvolvimento destas elaborações, por meio de exemplos simples e com uso de classes exemplificadas e explicadas.

Ainda são apresentadas a estrutura básica de um programa elaborada com uso da API, bem como aspectos da elaboração de programas mais complexos e elaborados, tudo isto com uso de exemplos simples e de fácil entendimento.

1.5.3. Ferramentas para Realidade Aumentada

A seção 3 apresenta as ferramentas para o desenvolvimento de sistemas de Realidade Aumentada (RA). Composta por dois capítulos, ARToolkit e OpenCV, apresenta os fundamentos, funcionamento e exemplos das duas tecnologias.

O capítulo 5 aborda ARToolkit, que é uma biblioteca amplamente utilizada na concepção de aplicações de RA de baixo custo. Por meio do rastreamento óptico, com uso de webcam, permite experimentar RA e aplicar a mesma na construção de soluções simples.

Trata-se de um recurso gratuito e livre que pode ser utilizado mesmo em computadores mais simples, dispensado pesado processamento e/ou configuração.

No capítulo 6, é visto o OpenCV (Intel Open Source Computer Vision Library), que é uma coleção de funções em C (e umas poucas classes em C++) que implementam diversos algoritmos de processamento de imagens e visão computacional. Trata-se de uma biblioteca gratuita de código aberto. Suas principais vantagens relacionam-se com o desenvolvimento de aplicações de tempo real,

sendo uma alternativa interessante para implementação de interfaces para Realidade Aumentada.

Nesse capítulo, os autores fornecem o conhecimento básico, com abordagem prática, para permitir que o leitor possa começar a usar e explorar essa API com facilidade. A título de ilustração são apresentados alguns exemplos de aplicações do OpenCV, incluindo APIs dele derivadas

1.5.4. Ferramentas para Jogos e Aplicações Interativas em 3D

A seção 4 apresenta ferramentas para desenvolvimento de Jogos e aplicações interativas em 3D. A mesma é composta por 4 capítulos.

No capítulo 7, sobre OGRE3D, os autores apresentam a tecnologia por meio de exemplos práticos, dando uma visão geral das potencialidades do engine de desenvolvimento.

O OGRE (Object-Oriented Graphics Rendering Engine) é um engine gráfico 3D de código aberto orientado a objetos. Desenvolvido em C++ e com o propósito de tornar a implementação de aplicações 3D mais fácil e intuitiva tem como diferencial a utilização de aceleração gráfica do hardware, podendo ser utilizada para desenvolvimento de aplicações 3D interativas, inclusive para dispositivos móveis.

O capítulo 8 apresenta um engine de jogos gratuito (de código aberto) desenvolvido com uso de C++ com binding para a linguagem Python, o Panda3D.

O processo de utilização do Panda, relacionado com a escrita de um programa em Python que acessa e controla os recursos do engine é apresentado por meio de um conjunto de módulos, do pi-

peline de renderização, hierarquia de classes, ferramentas existentes para facilitar o desenvolvimento.Além disto, um tutorial exemplifica a criação de um jogo 3D simples a partir do Panda3D.

No capítulo 9, sobre Enjine, apresenta-se um software livre, usado como engine didático para concepção de jogos, com utilização de Java. O projeto, resultado de pesquisas no laboratório de Tecnologias Interativas da Escola Politécnica da USP (Interlab) tem como objetivo permitir o uso livre de um engine para a criação de jogos educacionais.

O capítulo 10 do livro discorre sobre o PhysX, um engine, que embora proprietário, é livre para uso não comercial, capaz de permitir o desenvolvimento de aplicações gráficas altamente realistas por incorporarem aspectos do comportamento físico, que pode ser muito útil no desenvolvimento de jogos, por exemplo.

O objetivo deste capítulo é apresentar essa tecnologia por meio de uma abordagem teórica associada a exemplos práticos. São apresentados introduzidos os principais componentes do PhysX e suas funcionalidades. O capítulo também introduz diversas aplicações desenvolvidas com uso de PhysX e exemplos de programas implementados.

1.6. Conclusões

Este capítulo apresentou os fundamentos de realidade virtual e aumentada e a estrutura comentada do livro, indicando os aspectos tecnológicos importantes de cada capítulo.

Assim, os editores esperam ter dado, aos leitores, melhores condições de assimilar as abordagens tecnológicas, apresentadas a seguir, envolvendo linguagens, bibliotecas e ferramentas usadas no desenvolvimento de aplicações de Realidade Virtual e Aumentada.

Referências

Azuma, R. et al. (2001) "Recent Advances in Augmented Reality." IEEE Computer Graphics and Applications, v .21, n.6, p. 34-47.

Burdea, G., Coiffet,P. (1994) "Virtual RealityTechnology", John Wiley & Sons.

Cardoso A. (2002) – Uma Arquitetura para Elaboração de Experimentos Virtuais Interativos suportados por Realidade Virtual não-imersiva - tese de doutorado – Escola Politécnica da Universidade de São Paulo.

Kirner, C., Pinho, M.S. (1996) "Introdução a Realidade Virtual". Mini-Curso, JAI/SBC, Recife, PE.

Realidade Aumentada (2007) – site sobre conceitos de Realidade Aumentada com textos, referências e dicas – disponível em http://www.realidadeaumentada.com.br - acesso em março de 2007.

Vince, J. (1995) "Virtual Reality Systems", Addison-Wesley.

Vince, J. (2004) "Introduction to Virtual Reality", Springer-Verlag, 2nd edition.

Capítulo 2

Ambientes de Realidade Virtual Usando OpenGL

Márcio Serrolli Pinho[1] e Marcos Wagner de Souza Ribeiro[2]

[1]Faculdade de Informática - PUCRS

[2]Institututo Luterano de Ensino Superior de Itumbiara

pinho@pucrs.br, marcos.ribeiro@ulbra.br

Abstract

This chapter describes the use of OpenGL graphics libray in virtual reality applications.

Resumo

Este capítulo aborda o uso da biblioteca gráfica OpenGL para a construção de aplicações de realidade virtual.

2.1. Introdução

OpenGL foi portada para PC, por volta do ano de 1992. Foi derivada da biblioteca GL usada para acesso ao hardware gráfico das máquinas Silicon Graphics. Apesar dos problemas iniciais, alternativas foram criadas e atualmente OpenGL tem sido a platafor-

ma gráfica de maior preferência para desenvolvedores da área de computação gráfica.

Cabe sempre ressaltar que OpenGL não é uma linguagem de programação, e sim uma robusta biblioteca que possibilita a configuração e o acesso a um determinado hardware. É necessário hardware e software para implementar a arquitetura OpenGL. Mais especificamente a biblioteca implementa chamadas para o driver do hardware. No entanto existe uma biblioteca que implementa as funções da OpenGL via software (MESA). Por ser uma biblioteca e não uma linguagem de programação, OpenGL é dependente das últimas (C/C++, Java e Delphi) para realizar o acesso ao hardware.

2.2. Bibliotecas

OpenGL compõe-se de duas bibliotecas:

- Opengl32.lib ou **libGL.so:** contém todos os dados de acesso ao driver da placa de vídeo.

- OpenGL Utility Library (glu32.lib, libGLU.so ou libMesaGLU.so): Esta biblioteca implementa diversas funções úteis para o desenvolvimento de aplicações.

Existe ainda uma outra biblioteca necessária, que varia entre os Sistemas Operacionais, que é responsável por enviar os dados gráficos dos buffers para as aplicações. Em sistemas Windows, a biblioteca base é WGL, enquanto no Linux a biblioteca é a GLX, no MAC tem-se a GLA e o OS/2 a PGL.

Considera-se uma tarefa difícil, fazer um programa portável para todas essas bibliotecas, para isso existem duas soluções para resolver esse problema: SDL e GLUT.

SDL (Simple DirectMedia Layer): biblioteca que implementa todas as funcionalidades necessárias a uma aplicação multimídia.

GLUT (OpenGL Utility Toolkit): conjunto de rotinas que são independentes de sistema operacional..

2.3. Tipos de Dados

Devido à diversidade de tipos de dados em cada linguagem de programação (int, float...), em função do sistema operacional e da arquitetura de computadores, OpenGL possui um conjunto de enumerações mapeando os tipos de dados de cada linguagem para tipos de dados próprios da biblioteca. A tabela abaixo mostra os principais tipos de dados em OPenGL e seus equivalentes em C/C++, assim como os sufixos usados:

OpenGL	**C/C++**	**Sufixo**
GLbyte	char	b
Glshort	short	s
GLint ou Glsizei	int	i
GLfloat ou GLclampf	float	f
GLdouble ou GLclampd	double	d
GLybyte ou GLboolean	unsigned char	ub
Glushort	unsigned short	us
GLuint ou GLenum ou GLbitfield	unsigned integer	ui

2.4. Primitivas

OpenGL oferece 10 primitivas que compõe a função glBegin():

- GL_POINTS: pinta o pixel na posição do vértice apresentado numa cor corrente.

- GL_LINES: desenha uma linha numa cor corrente a cada dois vértices. Se for informado 4 vértices (v1, v2, v3 e v4), duas linhas serão desenhadas, uma ligando o v1 ao v2 e outra do v3 ao v4.

- GL_LINE_STRIP: desenha uma linha numa cor corrente a cada dois vértices. De acordo com o exemplo do comando anterior, a diferença seria o desenho de uma linha do v2 ao v3.

- GL_LINE_LOOP: idêntico ao comando anterior, porém liga o primeiro vértice ao último vértice.

- GL_TRIANGLES: desenha um triângulo a cada três vértices informados.

- GL_TRIANGLE_STRIP: desenha um triângulo a cada três vértices, porém com o aumento dos vértices informados, despreza o primeiro, mantendo sempre um grupo de três vértices atuais.

- GL_TRIANGLE_FAN: desenha triângulos com vértice em comum.

- GL_QUADS: desenha um quadrilátero, preenchido ou não a cada quatro vértices informados.

- GL_QUAD_STRIP: desenha um quadrilátero com o uso de quatro vértices, seguindo o mesmo princípio do GL_TRIANGLE_STRIP.

- GL_POLYGON: desenha uma linha entre os vértices de um conjunto e liga o último ao primeiro. Uma restrição da GL_POLYGON é o fato de que o polígono deve ser obrigatoriamente Convexo.

2.5. Máquina de Estados

OpenGL é estruturada de forma que a maioria das configurações permanecem intactas até que essa configuração seja alterada, fazendo com que todas as funções obedeçam a uma configuração inicial. O melhor exemplo para essa estrutura é o uso da cor com o comando **glColor**, que ao ser configurada faz com que todas as primitivas processem aquela cor até que haja alteração. Essa estrutura é chamada de **Máquina de Estados.**

OpenGL possui dois tipos de configuração de estado:

1 – Habilitação de processamento de estado: passa por um estado e recupera os resultados processados por ele.

2 – Configuração de produção do resultado gerado pelo estado: passagem e produção de resultados obrigatórios.

2.5.1. Habilitação de processamento

Os estados dessa categoria são habilitados ou desabilitados com os comandos **glEnable()** ou **glDisable().**

Os parâmetros desses comandos podem ser:

- **GL_ALPHA_TEST:** fragmento alfa para cálculo de transparência.

- GL_AUTO_NORMAL: gera vetores normais à superfície.

- GL_BLEND: combina RGBA com a cor já armazenada para efeito de transparência.

- GL_COLOR_LOGIC_OP: processamento lógico de cores.

- GL_CLIP_PLANEi: processamento de plano de recorte para o plano i.

- GL_COLOR_MATERIAL: processamento da saída de resultado do material ao objeto.

- GL_COLOR_TABLE: realize busca em uma tabela indexada de cores para produzir a cor RGBA.

- GL_CONVOLUTIN_nD: realiza uma operação de processamento de imagens chamada Convolução, ela pode ser 1 ou 2D.

- GL_CULL_FACE: elimina superficies escondidas de acordo com a normal da superfície.

- GL_DEPTH_TEST: realize o teste de profundidade do pixel para remoção de superfícies escondidas.

- GL_FOG: habilita o processamento de geração de neblina.

- GL_LIGHTi: habilita o processamento da luz i configurada e calcula se e como a luz modifica a cor de um pixel.

- GL_LIGHTING: habilita o processamento de iluminação da OpenGL.

- GL_LINE_SMOOTH: processamento antialiasing nas linhas desenhadas.

- GL_LINE_STIPPLE: usa a forma corrente para desenhar as linhas.

- GL_NORMALIZE: os vetores normais gerados por meio do commando glNormal são normalizados.

- GL_POINT_SMOOTH: suaviza os pontos desenhados usando antialiasing.

- GL_POLYGON_SMOOTH: suaviza os polígonos usando antialiasing.

- GL_POLYGON_STIPPLE: preenche o polígono com um padrão configurado.

Ainda existem outros estados que podem ser habilitados por esse procedimento.

2.5.2. Configuração de Saída

- GLColor: configura a cor.

- **GLClearColor:** configura cor para limpar a tela.

- glBindTexture(): configura a textura a ser aplicada.

- glNormal: configura a normal a ser usada.

Também existem outros estados de configuração de saída.

2.6. Cenários para Ambientes de Realidade Virtual

Partindo para o uso da OpenGL na construção de uma aplicação, a primeira tarefa é iniciar o sistema com as configurações iniciais. Essas configurações são importantes, porém não é o foco desse capítulo, pois permanecem inalteráveis para a maioria dos sistemas desenvolvidos. Generalizando, essas configurações são importantes para:

1 - Inicialização

a) Ajustar o formato de *pixel* na arquitetura OpenGL (RGB ou RGBA).

b) Indicar se a OpenGL utilizará ou não buffer de profundidade.

c) Indicar se a OpenGL utilizará ou não dois buffers.

2 – Projeção

a) Configurar a cor usada para apagar a tela.

b) Configurar o modo como os dados serão desenhados, indicando o modelo de matriz (projeção, modelo/visão ou textura).

c) Configurar a matriz escolhida com a matriz identidade, ou seja ajustar o valor da matriz para permitir um controle.

d) Mapear as coordenadas de tela configurando a matriz de projeção.

3 – Interação

a) Iniciar, configurar e posicionar a janela no sistema operacional.

b) Tratar os eventos do teclado.

Após essas configurações a função principal é a criação de cenários ou desenho de objetos, geralmente essa função é chamada de "*draw*". Essa rotina é a principal no programa e o foco desse capítulo. A seguir são apresentados os principais procedimentos na construção de cenários que podem usados na construção de ambientes virtuais.

Para modelagem de cenários, há duas alternativas básicas, quando se usa uma biblioteca gráficas como OpenGL. Na primeira, modela-se "à mão" os objetos gráficos por meio de primitivas OpenGL, e na segunda utilizam-se modeladores gráficos capazes de exportar arquivos de objetos tridimensionais que posteriormente são lidos e exibidos com comandos OpenGL.

Na Figura 1 pode-se observar um retângulo modelado em OpenGL.

```
glBegin(GL_POLYGON); // desenha um polígono
      // define os vértices da face
    glVertex3f(-10.0, -10.0, -10.0);
    glVertex3f(10.0, -10.0, -10.0);
    glVertex3f(10.0, -10.0, 10.0);
    glVertex3f(-10.0, -10.0, 10.0);
glEnd; // fim do desenho
```

Figura 1 – Exemplo de Objeto modelado em OpenGL

Para a leitura de modelos tridimensionais uma alternativa é a biblioteca SmallVR, disponível em http://www.smallvr.org ou o OBJ File Reader, disponível em http://www.inf.pucrs.br/~pinho/CG/Aulas/LeitorDeObj.

2.6.1. Transformações Geométricas

A biblioteca gráfica OpenGL é capaz de executar transformações de translação, rotação e escala.

A idéia central das transformações em OpenGL é que elas são cumulativas, ou seja, podem ser aplicadas umas sobre as outras.

Uma transformação geométrica de OpenGL é armazenada internamente em uma **matriz**. A cada transformação, esta matriz é alterada e usada para desenhar os objetos a partir daquele momento, até que seja novamente alterada.

2.6.1.1. Translação

Para efetuar uma translação há o comando glTranslatef(tx, ty, tz) que move todas as coordenadas dos objetos ao longo dos eixos coordenados. Na Figura 2 pode-se observar do uso do comando de translação.

```
DesenhaObjeto();// Desenha o objeto na posição
                // correspondente às suas
                 // coordenadas originais
```

```
glTranslatef(10,10,10);
DesenhaObjeto(); // Desenha o objeto descolado de 10
                 // unidades em cada eixo
glTranslatef(10,10,10);
DesenhaObjeto(); // Desenha o objeto descolado de 20
                 // unidades em cada eixo
```

Figura 2 - Exemplos de Translação

2.6.1.2. Rotação

Para efetuar uma rotação há o comando glRotatef(Angulo, x, y, z) que gira o objeto ao redor do vetor (x,y,z). O giro é de Angulo graus, no sentido anti-horário. Na Figura 3 pode-se observar do uso do comando de rotação.

```
DesenhaObjeto();// Desenha o objeto na posição
                // correspondente às suas
                // coordenadas originais

glRotatef(20,1,0,0);
DesenhaObjeto(); // Desenha outro objeto rotacionado
                 // 20 graus ao redor do eixo X

glRotatef(30,1,0,0);
DesenhaObjeto(); // Desenha mais um objeto,
                 // rotacionado
                 // 30 graus ao redor do eixo X
                 // LEMBRE-SE, AS TRANSFORMAÇÕES SÃO
                 // CUMULATIVAS. Neste caso o giro
                 // total será de 50
                 // graus
```

Figura 3 - Exemplos de Rotação

2.6.1.3. Escala

Para efetuar uma escala há o comando glScalef(ex, ey, ez) que altera a escala do objeto ao logo dos eixos coordenados (Figura 4).

```
DesenhaObjeto();// Desenha o objeto na posição
                // correspondente às suas
                // coordenadas originais

glScalef(0,0.5,0);
DesenhaObjeto(); // Desenha outro objeto com  a
                 // altura
                 // diminuída pela metade
glScalef(0,2.0,0);
DesenhaObjeto(); // Desenha  mais um objeto no
                 // tamanho original
                 // LEMBRE-SE, AS TRANSFORMAÇÕES SÃO
                 // CUMULATIVAS.
```

Figura 4 - Exemplos de Escala

2.6.2. Reinicializando as Transformações

Para permitir que a transformação atual seja reinicializada há o comando `glLoadIdentity()`. Este comando anula todas as transformações geométricas setadas anteriormente (Figura 5).

```
DesenhaObjeto();// Desenha o objeto na posição
                // correspondente às suas
                // coordenadas originais

glScalef(0,0.5,0);
DesenhaObjeto(); // Desenha outro objeto com  a
                 // altura
                 // diminuída pela metade
glLoadIdentity();// reinicializa as transformações
glScalef(0,2.0,0);
DesenhaObjeto(); // Desenha o outro objeto com o
                 // dobro do tamanho original
                 // LEMBRE-SE, AS TRANSFORMAÇÕES SÃO
                 // CUMULATIVAS.
```

Figura 5 - Exemplos de Re-inicialização de Transformações

2.6.3. Limitando o Escopo das Transformações

Para permitir que uma transformação valha somente em um certo trecho de programa e assim não altere o que está sendo desenhado depois, há os comados `glPushMatrix()` e `glPopMatrix()`.

A idéia é que o `glPushMatrix()` armazene as transformações atuais em um pilha interna do OpenGL e que estas transformações possam ser retiradas depois por um `glPopMatrix()`.

```
DesenhaObjeto();// Desenha o objeto na posição
                // correspondente às suas
                //  coordenadas originais
glPushMatrix(); // salva as transformações atuais na
                // pilha
                // Diminui a altura do objeto
                //  à metade do original
      glScalef(0,0.5,0);
      DesenhaObjeto();
glPopMatrix(); // restaura as transformações
               // anteriores

glPushMatrix(); // salva as transformações atuais na
                // pilha
// Desenha o outro objeto com o dobro do tamanho
// original
      glScalef(0,2.0,0);
      DesenhaObjeto();
glPopMatrix(); // restaura as transformações
anteriores
```

Figura 6 - Exemplos de Limitação das Transformações Geométricas

2.7. Navegação em ambientes de Realidade Virtual

A navegação em OpenGL ou, em outras palavras, o posicionamento do observador, é feito normalmente com a função glu-

LookAt. Nesta função define-se a posição do observador e o seu ponto de interesse, além do vetor que aponta para o "lado de cima" do cenário 3D.

2.7.1. Andando "para frente" com a gluLookAt

Para permitir que um observador ande na direção em que ele está olhando é necessário somar à posição atual do observador um vetor que tenha a mesma direção que vetor que vai do observador até o alvo. A mesma soma deve ser feita ao ponto alvo. É interessante que este vetor seja unitário pois assim se pode controlar a velocidade do deslocamento. A fórmula da Figura 7 demonstra como pode ser feito o deslocamento descrito acima.

```
PosicaoNova = PosicaoAtual + VetorAlvoUnitario *
TamanhoDoPasso
AlvoNovo = AlvoAtual + VetorAlvoUnitario *
TamanhoDoPasso
```

Figura 7 – Movimento do observador “pra frente”

2.7.2. Olhando "para os lados" com a gluLookAt

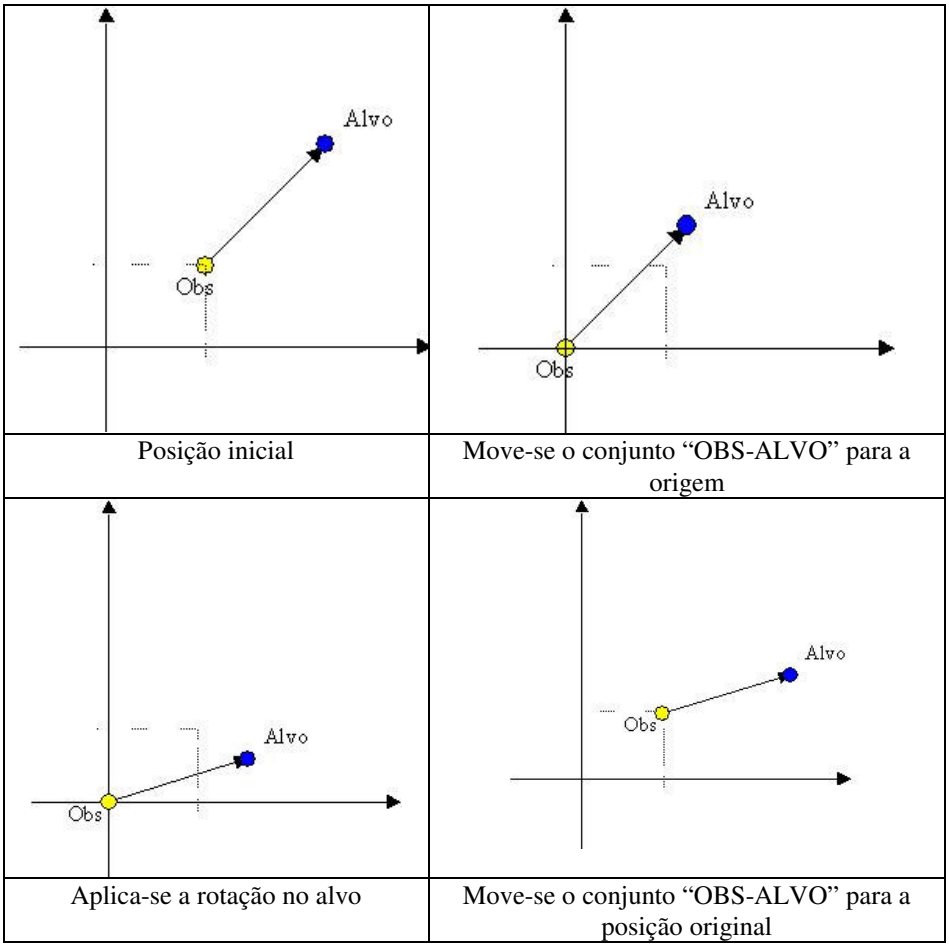
Para permitir que um observador "olhe para os lados" é preciso recalcular o ponto Alvo, aplicando sobre ele uma rotação ao redor do eixo Y.

Para rotacionar um ponto ao redor deste eixo usa-se a fórmula da Figura 8.

```
AlvoNovo.X = AlvoAtual.X*cos(alfa) +
AlvoAtual.Z*sen(alfa)
AlvoNovo.Y =  AlvoAtual.Y
AlvoNovo.Z = -AlvoAtual.X*sen(alfa) +
AlvoAtual.Z*cos(alfa)
```

Figura 8 – Rotação do observador na Origem

Note, entretanto que esta fórmula rotaciona o ponto ao redor do ponto (0,0,0). Como fazer para rotacioná-lo corretamente ? Uma alternativa é executar os passos da Figura 9.



Posição inicial	Move-se o conjunto "OBS-ALVO" para a origem
Aplica-se a rotação no alvo	Move-se o conjunto "OBS-ALVO" para a posição original

Figura 9 – Rotação do Genérica do Observador

2.8. Geração de Visão Estereoscópica

2.8.1. Posicionamento do Observador

A idéia básica de criação de imagens estereoscópicas em OpenGL é a geração de imagem diferente para cada olho a partir da mudança da posição do observador.

Uma forma simples de efetuar esta mudança de posição é apresentada na Figura 10 onde o observador é movido para a esquerda ou para a direita, conforme o olho para o qual se deseja gerar a cena.

```
//*************************************************
//  void PosicUser()
//  esta função define a posicão de cada um dos
//  olhos do observador
//
//  Variáveis usadas:
//        DistOlhos: distância entre os olhos do
//  usuário
//
//*************************************************
void PosicUser(int Olho)
{
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(80,ratio,0.01,200);
  switch (Olho)
  {
    case ESQUERDO: // seta posição para o olho
                   // esquerdo
     gluLookAt(-DistOlhos/2, 0, 5, 0, 0, 0,
0.0f,1.0f,0.0f);
     break;
  case DIREITO: // seta posição para o olho direito
     gluLookAt(DistOlhos/2, 0, 5, 0, 0, 0,
0.0f,1.0f,0.0f);
     break;
  }
  glMatrixMode(GL_MODELVIEW);
```

```
}
```

Figura 10 – Posicionamento do Observador para geração de Imagens Estereoscópicas

Além de gerar duas imagens diferentes, é preciso exibi-las uma para cada olho. A forma de fazer esta exibição depende do tipo de dispositivo que está sendo usado. Nas seções a seguir são apresentadas duas formas de realizar esta tarefa, cada uma vinculada a um tipo de dispositivo.

2.8.2. Imagens Entrelaçadas: técnicas e dispositivos

No caso do uso de Óculos do tipo I-Glasses, a separação entre a imagem do olho esquerdo e do olho direito é feita pelo *hardware* do óculos que põe no visor do olho direito as imagens das linhas pares da tela, e no visor do olho esquerdo a imagem das linhas ímpares.

Maiores informações sobre este tipo de óculos podem ser obtidas no *site* http://www.i-glassesstore.com. Atente para o fato de que nem todos os modelos disponíveis nesta página têm capacidade de exibir imagens estereoscópicas.

Para fazer a separação das imagens na tela, usa-se o ***stencil buffer***, com o seguinte algoritmo:

- Bloqueia-se, através do *stencil buffer*, a exibição das imagens nas linhas impares;
- Exibe-se a imagem referente ao olho direito;
- Libera-se, através do *stencil buffer*, a exibição das imagens nas linhas impares;
- Bloqueia-se, através do *stencil buffer*, a exibição das imagens nas linhas pares;
- Exibe-se a imagem referente ao olho esquerdo

O bloqueio (ou desbloqueio) das linhas pares e ímpares neste caso é feito através da montagem de uma "máscara" no *stencil*, conforme a Figura 11. Esta máscara é, na verdade, um conjunto de linhas horizontais desenhadas no *stencil*, somente nas linhas pares da tela.

```
//*************************************************
//  void DesenhaFundo()
//      Inicializa o stencil com 1 nas linhas pares
//
//*************************************************
void DesenhaFundo(int Largura, int Altura )
{

 // Habilita o uso do Stencil
 glEnable(GL_STENCIL_TEST);
 // Define que "0" será usado para limpar o Stencil
 glClearStencil(0);
 // limpa o Stencil
 glClear(GL_STENCIL_BUFFER_BIT);
 glStencilFunc(GL_ALWAYS, 1, 1);
 glStencilOp(GL_REPLACE,GL_REPLACE,GL_REPLACE);

// desenha a máscara nas linhas pares
 glMatrixMode(GL_PROJECTION); //
 glLoadIdentity ();
 gluOrtho2D(0, Largura, 0, Altura);
 glMatrixMode(GL_MODELVIEW);
 glBegin(GL_LINES);
  // inicializa apenas as linhas pares no stencil
  for (int y= 0; y < Altura; y += 2)
  {
   DrawLine(0,y, Largura, y);
  }
 glEnd();
 // volta à matriz de objetos
 //glMatrixMode(GL_MODELVIEW);
}
```

Figura 11 - Marcação das linhas pares no stencil buffer

No momento da exibição dos objetos do cenário, o programador deve habilitar e desabilitar o desenho sobre as linhas marcadas no *stencil*. A Figura 12 demonstra este procedimento.

```
//*************************************************
//  void HabilitaOlhoEsquerdo()
//
//*************************************************
void HabilitaOlhoEsquerdo()
{
      glStencilFunc(GL_NOTEQUAL, 1, 1);
      glStencilOp(GL_KEEP,GL_KEEP, GL_KEEP);
}
//*************************************************
//  void HabilitaOlhoDireito()
//
//
//*************************************************

void HabilitaOlhoDireito()
{
      glStencilFunc(GL_EQUAL, 1, 1);
      glStencilOp(GL_KEEP,GL_KEEP, GL_KEEP);
}
//*************************************************
//
//
//*************************************************
void display( void )
{
 // Limpa a tela
 glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 DefineLuz();
 PosicUser(ESQUERDO);  // posiciona o observador no
olho esquerdo
  HabilitaOlhoEsquerdo();
  DesenhaCena();
 PosicUser(DIREITO);   // posiciona o observador no
olho direito
  HabilitaOlhoDireito();
  DesenhaCena();
 glutSwapBuffers();
}
```

Figura 12 – Exibição de Cenas Estereoscópicas com Stencil Buffer

2.8.3. Imagens Alternadas: técnicas e dispositivos

O uso de imagens alternadas depende do uso de um óculos do tipo shutter-glasses e de um placa de vídeo que suporte a geração de estéreo. O processo de posicionamento do observador continua sendo o mesmo descrito na seção 0.

Primeiramente o programador deve inicializar a biblioteca GLUT com a opção de estereoscopia. Caso a placa de vídeo não suporte esta característica, ocorrerá um erro neste momento. O comando para esta inicialização é:

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
GLUT_STEREO);
```

No momento de exibir o cenário, o programador deverá definir para que olho está gerando a imagem, através da escolha do buffer de desenho. Na Figura 2.13 apresenta-se um trecho de código que faz esta tarefa.

```
void display( void )
{
 // Limpa a tela
 DefineLuz();
 PosicUser(ESQUERDO);  // posiciona o observador no
                       // olho esquerdo
 glDrawBuffer(GL_BACK_LEFT); // ativa BUFFER
                             // ESQUERDO
       glClearColor(1.0, 0.0, 0.0, 1.0); // verm.
       glClear( GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
       DesenhaCena();
 PosicUser(DIREITO);   // posiciona o observador no
                       // olho direito
 glDrawBuffer(GL_BACK_RIGHT); // ativa BUFFER DIREITO
       glClearColor(0.0, 0.0, 1.0, 1.0);// azul
       glClear(GL_COLOR_BUFFER_BIT);
       DesenhaCena();
 glutSwapBuffers();
}
```

Figura 2.13 – Exibição de Cenas Estereoscópicas para shutter-glasses

2.9. Apontamento de Objetos

Todas as técnicas de interação apresentadas na seção anterior exigem a existência de rotinas de cálculo de interseção entre o apontador do usuário e o objeto a ser selecionado. Para tanto, existem diversas bibliotecas e técnicas disponíveis.

Em todas é necessário obter as coordenadas dos vértices do objeto aplicando a estes pontos todas as transformações geométricas que afetam o objeto.

Em OpenGL, para obter estas coordenadas, e deve-se multiplicar a matriz de transformação atual pelas coordenadas dos vértices do objeto. No exemplo da Figura 14, apresenta-se uma das formas de realizar esta multiplicação.

No caso de **apontamento de objetos** esta função é útil pois permite calcular as coordenadas dos vértices dos triângulos que formam este apontador e a partir destas determinar se há ou não intersecção entre os objetos e o apontador.

```
#include <windows.h>
#include <gl\gl.h>
typedef struct
{
 GLfloat x,y,z;
} Ponto3D;
// ***********************************************
//  void calcula_ponto(Ponto3D *p, Ponto3D *out)
//
//  Esta função calcula as coordenadas de um ponto
//  no sistema de referência do universo (SRU), ou
// seja, aplica as rotações, escalas e translações a
// um ponto no sistema de referência do objeto
// (SRO).
// ***********************************************
void calcula_ponto(Ponto3D *p, Ponto3D *out)
{
   GLfloat ponto_novo[4];
```

```
   GLfloat matriz_gl[4][4];
   int  i;
   glGetFloatv(GL_MODELVIEW_MATRIX,&matriz_gl[0][0]);
   for(i=0;i<4;i++)
   {
      ponto_novo[i]= matriz_gl[0][i] * p->x+
                     matriz_gl[1][i] * p->y+
                     matriz_gl[2][i] * p->z+
                     matriz_gl[3][i];
   }
   out->x=ponto_novo[0];
   out->y=ponto_novo[1];
   out->z=ponto_novo[2];
}

Ponto3D p1, p2;
// ********************************************
//  void Desenha ()
//
//
// ********************************************
void Desenha ()
{
     Ponto3D p1_new, p2_new;
     p1.x = 10;   p1.y = 10;   p1.z = 20;
     p2.x = 20;   p2.y = 10;   p2.z = 20;
     glRotate3f(10,0,0,1);
     glBegin(GL_LINES);
         //desenha a linha
         glVertex3f(p1.x, p1.y, p1.z);
         glVertex3f(p2.x, p2.y, p2.z);
         // aplica as transformações geométricas
         // aos pontos da linha
         calcula_ponto(&p1, &p1_new);
         calcula_ponto(&p2, &p2_new);
         //imprime p1_new e p2_new
         // .......
     glEnd();
}
```

Figura 14 – Aplicação das Transformações geométricas a um ponto

2.10. Movimentação de objetos – Transformações hierárquicas

Outro aspecto importante na manipulação de um objeto é a necessidade de fazê-lo mover-se junto com o apontador que o selecionou. Para tanto, usa-se a idéia de montar uma hierarquia de objetos em que tem-se "objetos-filhos" que acompanham todas as transformações de seus "pais".

As seções a seguir apresentam os conceitos necessários para trabalhar com hierarquias diretamente em OpenGL.

2.10.1. Representando Transformações Geométricas por Matrizes

As transformações geométricas aplicadas usando comandos OpenGL do tipo `glTranslate`, `glRotate` ou `glScale`, são sempre armazenadas em uma matriz chamada MODELVIEW.

A cada chamada de uma destas funções, OpenGL cria uma matriz de transformação específica para a função e a seguir multiplica esta matriz pela matriz MODELVIEW atual.

Por exemplo, na chamada da função `glTranslatef(-3, 2, -8)` é criada a seguinte matriz:

```
 1.00   0.00   0.00   0.00
 0.00   1.00   0.00   0.00
 0.00   0.00   1.00   0.00
-3.00   2.00  -8.00   1.00
```

Note que na última linha da matriz são colocados os valores passados como parâmetros para a função.

No momento de exibir um objeto o que OpenGL faz é a multiplicação desta matriz pelos vértices do objeto, obtendo assim a posição final do objeto. Por exemplo, no caso de exibir o ponto (10, 15, 20) seria obtido o ponto (7,17, 12) conforme a Figura 15.

$$\begin{bmatrix}10 & 15 & 20 & 1\end{bmatrix} * \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ -3 & 2 & -8 & 1\end{bmatrix} = \begin{bmatrix}7 & 17 & 12 & 1\end{bmatrix}$$

Figura 15 – Aplicação de uma Translação a um ponto

Caso seja desejável ou necessário, é possível setar "à mão" esta matriz de transformação. Esta operação manual pode ser feita de três formas:

- Usando a função `glLoadIdentity()`: com esta função a matriz de transformação converte-se na matriz identidade;
- Usando a função `glLoadMatrixf(matrix)`: com esta função é possível definir uma nova matriz de transformação destruindo-se a matrix anterior;
- Usando a função `glMultMatrixf(matrix)`: com esta função é possível multiplicar a matriz de transformação atual por uma segunda matriz.

2.10.2. Aplicando novas Matrizes de Transformação

Caso se tenha uma nova transformação (também representada em forma de matriz) é possível aplicá-la a matriz atual, bastando multiplicar a matriz atual pela nova matriz.

Por exemplo, caso tenhamos a matriz de transformação para uma translação (5, 7, 2) a equação a seguir calcula a nova matriz.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 2 & -8 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & 7 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 9 & -6 & 1 \end{bmatrix}$$

Figura 16 – Aplicação de uma Nova Translação a um Ponto

2.10.3. Armazenando as Transformações de um objeto em uma matriz

Para armazenar as transformações geométricas de um objeto em uma matriz deve-se chamar a função glGetFloatv imediatamente após terem sido "*setadas"* todas estas transformações do objeto (Figura 17).

```
GLfloat MatObj [4][4]; // matriz de transformações do
objeto
//*************************************************
************
//  void RecalculaMatrizObjeto()
//*************************************************
************
void RecalculaMatrizObjeto()
{
 glPushMatrix();
      glLoadIdentity();
      glTranslatef (PosObj.X, PosObj.Y, PosObj.Z);
  // guarda a matriz de transformação do objeto para
  // um futuro uso
      glGetFloatv(GL_MODELVIEW_MATRIX, &MatObj[0][0]);
 glPopMatrix();
}
```

Figura 17 – Obtenção da matriz de Transformação após uma translação

2.10.4. Alterando a Matriz de Transformação antes de desenhar um objeto

Uma vez que se tem a matriz de transformações de um objeto é possível aplicar estas transformações multiplicando-se esta matriz pela matriz atual. Isto é feito conforme a Figura 18.

```
GLfloat MatObj [4][4]; // matriz de transformações do
objeto
// ************************************************
//  void DesenhaObjeto()
// ************************************************

void DesenhaObjeto()
{
 glColor3f(0.0f,0.0f,0.6f);
 glPushMatrix();
      // multiplica a matriz de transformação atual
      // pela transformação do objeto-filho a ser
      // desenhado
      glMultMatrixf(&MatObj[0][0]);
      DesenhaCubo();
 glPopMatrix();
}
```

Figura 18 – Aplicação de uma matriz de Transformações a um Objeto

2.10.5. Montando Transformações Hierárquicas

A montagem de uma hierarquia de transformações serve para que se possa definir um objeto como **filho** de outro.

Neste caso, o objeto filho sofrerá todas as transformações geométricas aplicadas a seu pai e mais as suas próprias transformações. Para tanto, deve-se apenas desenhar o objeto logo após o desenho de seu pai. O exemplo da Figura 19 ilustra este procedimento.

```
// ************************************************
//  void DesenhaPAI()
// ************************************************

void DesenhaPAI()
```

```
{
   glColor3f(0.7f,0.7f,0.0f);
   glPushMatrix();
      //aplica as transformações do pai
     glMultMatrixf(&Mat_Pai[0][0]);
     DesenhaCubo();
     if (TemFilho)
        DesenhaObjeto();  // desenha Filho aplicando
                          // suas transformações
   glPopMatrix();
}
```

Figura 19 – Aplicação de uma Transformação Hierárquica

Neste caso, entretanto, no instante em um objeto torna-se filho de outro, este filho irá sofrer as transformações geométricas de seu pai **adicionadas às suas**. Por exemplo, se o pai tem uma translação em X igual a 40 e o filho outra igual a 30, então o objeto-filho irá ser transladado de 70 unidades. Isto irá ocorrer no exato momento em que o objeto filho for desenhado pela primeira vez após o pai, causando uma translação indesejada.

2.10.6. Montando o vínculo de hierarquia

Para contornar este efeito colateral é necessário que no momento do estabelecimento da relação pai-filho, a matriz do filho desfaça as transformações existentes no pai.

Para tanto deve-se multiplicar a matriz do filho pela **inversa da matriz** do pai, conforme ilustra o exemplo da Figura 20.

Note que isto deve ser feito somente no instante em que se estabelece o vínculo entre pai e filho. Isto porque só deve-se desfazer as transformações do pai que existem no momento do estabelecimento do vínculo. A partir deste momento, se uma nova transformação for aplicada ao pai, ela deverá ser também aplicada ao filho.

```
//
**************************************************
```

```
*********
//  void CriaVinculoPai_Filho()
//
*****************************************************
*********
void CriaVinculoPai_Filho()
{
    GLfloat MatInv[4][4];
        TemFilho = TRUE;
    // calcula a inversa da matriz do Pai
    CopyMatrix(MatInv_Pai, Mat_Pai);
    M_invert(MatInv_Pai);
      // Multiplica a Matriz do filho pela inversa da
      // matriz do pai
    M_mult(MatObj, MatInv_Pai); // MatObj = MatObj *
                                // MatInv_Pai
}
```

Figura 20 – Cálculo da Transformação do Objeto-Filho

2.10.7. Desfazendo o vínculo de hierarquia

Para desfazer o vínculo de hierarquia e deixar o objeto no mesmo lugar, deve-se multiplicar sua matriz pela matriz atual de seu pai. Se isto não for feito ocorrerá uma translação indesejada no objeto filho, pois este deixará de sofrer as translações do pai após o término do vínculo.

```
// ***********************************************
//  void DesfazVinculoPai_Filho()
//
// ***********************************************
void DesfazVinculoPai_Filho()
{
    TemFilho = FALSE;
    M_mult(MatObj, MatPai); // MatObj = MatObj *
                            // Mat_Pai
}
```

Figura 21 – Novo cálculo da Transformação do Objeto-Filho

A partir deste momento o objeto filho não mais deverá ser desenhado após o pai.

2.11. Exibição de Cenários em Múltiplas Telas

Diversas aplicações de realidade virtual necessitam exibir cenários em um conjunto de telas colocadas lado a lado. Este é o caso das *CAVEs* e dos *Workbenches*. Um exemplo de uma cena exibida em várias telas é apresentada na Figura 22.

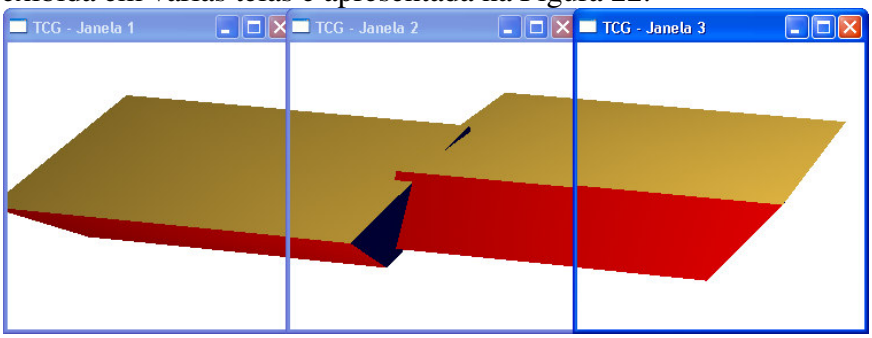


Figura 22 – Cena em Múltiplas telas

Em OpenGL, para gerar este tipo de imagem usa-se a função `glFrustum` que define o *View Frustum* ou "volume de visualização" a ser usado para projetar uma cena da tela. Na Figura 23 pode-se observar o "volume de visualização" e sua relação com a posição do observador e do alvo.

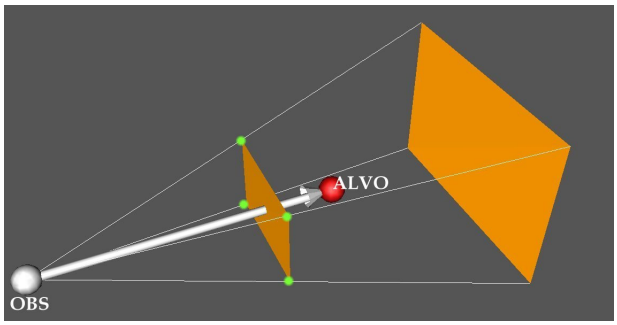


Figura 23 – Volume de Visualização

Na Figura 24 este mesmo "volume de visualização" é apresentado com suas respectivas dimensões.

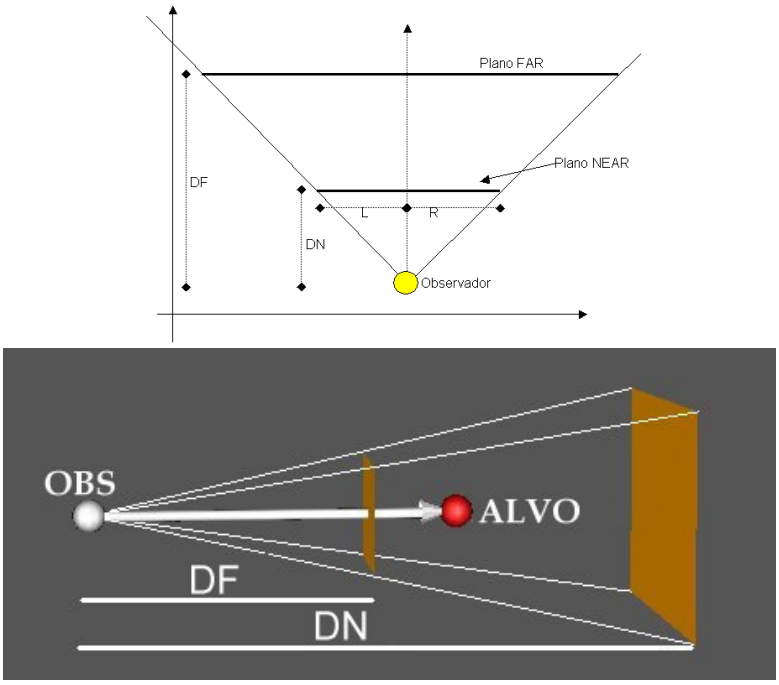


Figura 24 – Volume de Visualização (com dimensões)

Na Figura 24, os valores DF e DN são, respectivamente as distâncias máximas e mínimas do volume visível pelo observador e L (de *Left*) e R (de *Right*) definem o quanto o observador vê à esquerda e à direita de seu ponto central de visão, exatamente na distância do plano NEAR. Para definir um volume de visualização é necessário ainda definir os limites verticais (inferior e superior) da visão do observador na distância do plano NEAR. Para efeito de exemplo estas medidas serão chamadas B (de Bottom) e T (de Top), respectivamente.

Os valores de NEAR e FAR são fornecidos, em geral, pela aplicação. Os valores de L e R podem ser calculados a partir do ângulo de visão pelas fórmulas:

```
L = -Near * tan(ViewAngleRad/2)
R = Near * tan(ViewAngleRad/2)
```

Se considerarmos o Plano NEAR como um quadrado, teremos B igual a L e T igual a R.

A partir destes valores é possível definir o volume de visualização, através da função `glFrustum`. O código da Figura 25 apresenta um exemplo deste procedimento.

2.11.1. Dividindo o View Frustum em Múltiplas Telas

Para exibir uma cena em múltiplas telas, deve-se dividir o "volume de visualização" de forma que cada janela exiba uma parte dele. Na Figura 26 pode-se observar 3 volumes de visualização distintos.

```
// ************************************************
//  void SetViewFrustum()
// ************************************************
void SetViewFrustum()
{
      float Near, Left, Right, Bottom, Top;
      float ViewAngleRad;

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();

      ViewAngleRad = ViewAngle * 3.14f/180;

      Near = 1; // definido na aplicação
      Far = 100; // definido na aplicação

      // Calcula os limites horizontais do View
      // Frustum
      Left = -Near * tan(ViewAngleRad/2);
      Right = Near * tan(ViewAngleRad/2);

      // Calcula os limites verticais do View
      // Frustum
      Bottom = -Near * tan(ViewAngleRad/2);
      Top = Near * tan(ViewAngleRad/2);
```

```
        // Seta o Frustum
        glFrustum(Left, Right, Bottom, Top, Near, Far);

        glMatrixMode(GL_MODELVIEW);
        SetObs(); // posiciona o observador
}
```

Figura 25 – Setando o Volume de Visualização

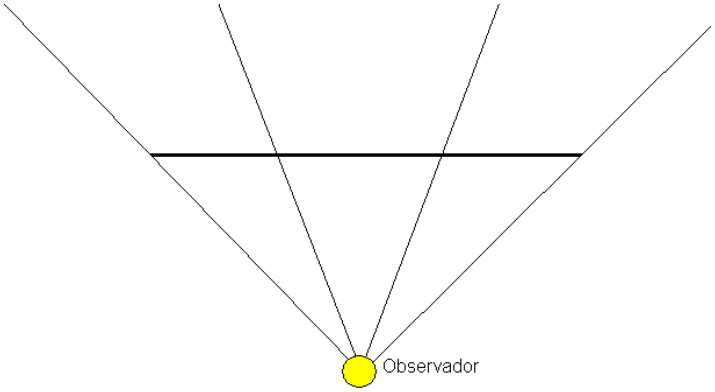


Figura 26 – Dividindo o Volume de Visualização

Em OpenGL, para recalcular o *View Frustum*, toma-se o ângulo de visão e divide-se pelo número de janelas nas quais se deseja exibir a imagem. Note que se deve dividir separadamente o volume na horizontal e na vertical pois o número de telas nestas duas direções pode ser diferente.

No código apresentado na Figura pode-se observar um exemplo que um "volume de visualização" é dividido horizontalmente em 3 janelas e verticalmente em duas. A Figura 28 apresenta o resultado visual deste programa.

```
// ************************************************
//  void SetViewFrustum()
// ************************************************
```

```
void SetViewFrustum()
{
      float Near, Left, Bottom, Top, Right, TileWidth,
TileHeight;
      float ViewAngleRad;

      glMatrixMode(GL_PROJECTION); glLoadIdentity();

      Near = 1; // definido na aplicação
      Far = 100; // definido na aplicação

      // Calcula os limites horizontais do View
      // Frustum
      Left = -Near * tan(ViewAngleRad/2);
      Right = Near * tan(ViewAngleRad/2);

      // Calcula os limites verticais do View Frustum
      Bottom = -Near * tan(ViewAngleRad/2);
      Top = Near * tan(ViewAngleRad/2);

      // Calcula a largura e a altura de cada janela
      TileWidth =  (Right - Left)/3;
      TileHeight = (Top - Bottom)/2;

      // Seta o Frustum de cada janela
      if (glutGetWindow()==jan1)// Inferior Esquerda
            glFrustum(Left, Left+TileWidth,
            Bottom, Bottom+TileHeight,  Near, Far);

      if (glutGetWindow()==jan2) // Inferior Central
            glFrustum(Left+TileWidth,
            Left+TileWidth*2,
            Bottom, Bottom+TileHeight, Near, Far);

      if (glutGetWindow()==jan3) // Inferior Direita
            glFrustum(Left+TileWidth*2,
            Left+TileWidth*3,
            Bottom, Bottom+TileHeight, Near, Far);

      if (glutGetWindow()==jan4) // Superior Esquerda
            glFrustum(Left, Left+TileWidth,
            Bottom+TileHeight, Top, Near, Far);

      if (glutGetWindow() == jan5) // Superior Central
```

```
            glFrustum(Left+TileWidth,
            Left+TileWidth*2,
             Bottom+TileHeight, Top, Near, Far);

      if (glutGetWindow() == jan6) // Superior Direita
            glFrustum(Left+TileWidth*2,
            Left+TileWidth*3,
            Bottom+TileHeight, Top, Near, 100);

      glMatrixMode(GL_MODELVIEW);

      SetObs();
}
```

Figura 27 – Código de Divisão do View Frustum

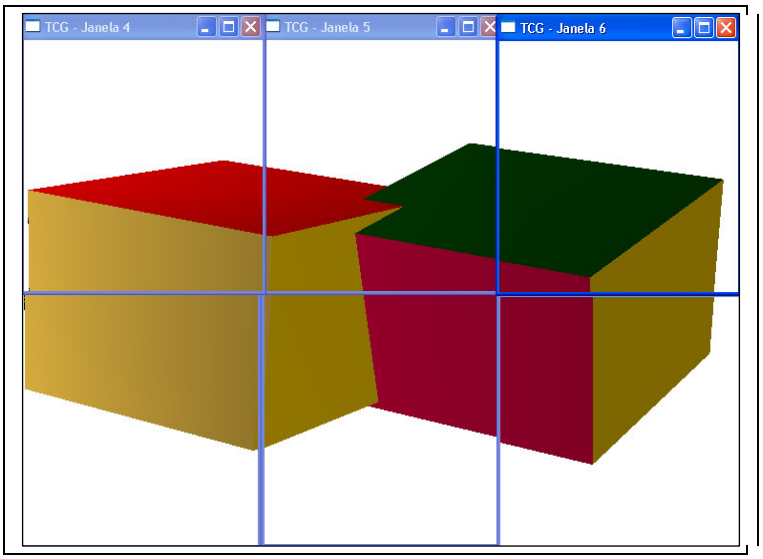


Figura 28 – "Telão 3 x 2"

Referências

BALDIS, J. Effects of Spatial Audio on Communication During Desktop Conferencing. Unpublished masters thesis, University of Washington.1998. Documento disponível no endereço eletrônico http://www.hitl.washington.edu/publications/r-98-23/

BASDOGAN, C., Srinivasan, M.A., 2001, "Haptic Rendering In Virtual Environments (PDF)" ,"Virtual Environments Hand-Book", pp. 117-134. http://network.ku.edu.tr/~Ecbasdogan/Papers/VRbookChapter.pdf

BOLT, R., Put-that-there: voice and gesture at the graphics interface. In:ACM SIGGRAPH CONFERENCE. SIGGRAPH'80 Los Angeles, 1980, **Proceedings**... Los Angeles: ACM Press, 1980. p. 262-270.

BOWMAN, D., & HODGES, L. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. **The Journal of Visual Languages and Computing**, New York, v. 10, n.1, p. 37–53. Jun. 1999.

BURDEA, Grigore;. Force and Touch Feedback for Virtual Reality. Prentice Hall, 2002.

CHURCHILL, E.F.; SNOWDOWN, D. Collaboratorative Virtual Environments: An Introductory Review of Issues and Systems. **Virtual Reality**, Londres, v. 3, n. 1, p 5-9, Jan. 1998.

DURLACH, N., SLATER M. Presence in Shared Virtual Environments and Virtual Togetherness.. Disponível em <http://www.cs.ucl.ac.uk/staff/m.slater/BTWorkshop/durlach.html> Acesso em: 20 de mar. 2002).

FORSBERG, A., HERNDON, K., ZELEZNIK, R. Aperture based selection for immersive virtual environment. In:ACM USER INTERFACE SOFTWARE AND TECHNOLOGY, UIST'96,

1996, Seattle. **Proceedings**…, Los Angeles:ACM Press, 1996. pp. 95-96.

FRASER, M., GLOVER, T., VAGHI, I., BENFORD, S., GREENHALGH, C., HINDMARSH, J., HEATH, C. Revealing the realities of collaborative virtual reality. In: ACM Collaborative Virtual Enviroment'2000. ACM CVE, 2000, San Francisco, CA. **Proceedings**… Los Angeles: ACM Press, 2000. p. 29-37.

JACOBY, R., FERNEAU, M., HUMPHRIES, J. Gestural Interaction in a Virtual Environment. In:STEREOSCOPIC DISPLAYS AND VIRTUAL REALITY SYSTEMS CONFERENCE, 1994, [s.l.]

LECUYER, A.; MEGARD, Christine; BUCKHARDT, Jean-Marie; LIM, Taegi; COQUILLART, Sabine; COIFFET, Phillipe, GRAUX, Ludovic. **The Effect of Haptic, Visual and Auditory Feedback on an Insertion Task on a 2-Screen Workbench**. Extraído em agosto de 2002. Online. Disponível na Internet:

http://www-rocq.inria.fr/i3d/i3d/publications/lecuyeript.pdf

LIANG, J., GREEN, M., JDCAD: A highly interactive 3Dmodeling system. **Computer and Graphics**, New York, v. 4, n. 18, p. 499-506. Apr. 1994.

MACLEAN, K. **Design of Haptic Interfaces**. Extraído em agosto de 2002. Online. Disponível na Internet: http://www.cs.ubc.ca/~maclean/publics/uist01-hapticsSurv.pdf

MINE, M. **Virtual environment interaction techniques**. Chapel Hill: UNC CS Dept., 1995. (Technical Report TR95-018)

MINE, M., ISAAC: A Meta-CAD System for Virtual Environments. **Computer-Aided Design**, [s.l.]

PINHO, M. S. **SmallVR Uma Ferramenta Orientada a Objetos para o Desenvolvimento de Aplicações de Realidade Virtual**.

Proceedings of the 5th SVR – Symposium on Virtual Reality, Fortaleza, Brasil, Outubro 2003, p. 329-340.

POUPYREV, I., M. BILLINGHURST, S. WEGHORST AND T. ICHIKAWA. The Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR. In: ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, 1996, Seattle, WA. **Proceedings…**, New York, NY: ACM Press, 1996, p.79-80.

SCHNEIDERMAN, B. Design the user Interface: strategies for effective human-computer Interaction. Addison-Wesley. 3rd ed., 1998.

STOAKLEY, R., CONWAY, M., PAUSCH, R., Virtual reality on a WIM: interactive worlds in miniature. In:ACM CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS., CHI'95, 1995, Denver, Colorado. **Proceedings…** New York, NY: ACM Press, 1995. p. 265-272.

VAN DAM, A. Post-WIMP User Interfaces. Communications of the ACM. Vol 40. No. 2, Feb 1997. P 63-67.

Capítulo 3

VRML e X3D

Alexandre Cardoso[1], José Gustavo Paiva[1], Luciano Pereira Soares[2]

[1]Faculdade de Engenharia Elétrica – Universidade Federal de Uberlândia
Campus Sta Mônica- 38400.902 – Uberlândia - MG – Brazil

[2]Universidade de São Paulo, POLI, Laboratório de Sistemas Integráveis.
Av. Professor Luciano gualberto n 158 trav.3 -Cidade Universitária
05508-900 - Sao Paulo, SP - Brasil

alexandre@ufu.br, gustavo@ufu.br, lpsoares@gmail.com

3.1. VRML97

Arquivos que simulam ambientes tridimensionais em VRML, são na verdade uma descrição textual, na forma de textos AScii. Por meio de qualquer processador de textos, o desenvolvedor pode conceber tais arquivos, salvá-los e visualizar resultados no navegador de Internet associado a um *plug-in* interpretador de VRML. Estes arquivos definem como estão as formas geométricas, as cores, as associações, os movimentos, enfim, todos os aspectos relacionados com a idéia do autor [AMES 1997]. Quando um dado navegador - *browser* - lê um arquivo com a extensão *.wrl*, o mesmo constrói o cenário descrito, usando os recursos do *plug-in* compatível.

De forma simplificada, um arquivo VRML se caracteriza por quatro elementos principais:

- *Header* – obrigatório;

- *Prototypes* – usado para descrever classes de objetos que podem ser usados na descrição do cenário;
- *Shapes*, *Interpolators*, *Sensors*, *Scripts* – usados para descreverem o cenário que se deseja construir;
- *Routes* – rotas: essenciais na definição de comportamento dos objetos que estão descritos no cenário.

Nem todos os arquivos contêm todos estes componentes. Na verdade o único item obrigatório em qualquer arquivo VRML é o cabeçalho (*header)*. Porém, sem pelo menos uma figura, o navegador não exibirá nada ao ler o arquivo. Outros componentes que um arquivo VRML também pode conter:

- *Comments;*
- *Nodes;*
- *Fields, field values;*
- *Defined node names;*
- *Used node names;*

O cabeçalho (*header*) é composto pela instrução "#VRML V2.0 utf8" e sua omissão impossibilita o *plug-in* do navegador de ler o arquivo em questão. Os protótipos (*proto*) contém a definição de novos nós que podem ser usados no arquivo em definição. A seção de descrição de formas (*shapes* etc) apresenta a descrição das formas que serão exibidas no navegador e a seção de rotas (*routes*) contém a definição das trocas de mensagens entre os nós de descrição de formas, interpoladores, sensores e *scripts*.

A concepção de cenários tridimensionais, usando VRML, se baseia na elaboração de uma grafo direcionado acíclico, contendo diferentes ramos - nós - que, associados de forma correta podem ser agrupados ou estarem independentes uns dos outros. A grande diversidade destes nós (54 pré-definidos), incluindo primitivas geométricas, propriedades de aparência, sons (e propriedades) e vários tipos de nós de agrupamentos, é uma das principais características e qualidades da linguagem.

É permitida reutilização de código através da prototipação, baseada na definição de novos nós (*protos*) que podem ser utilizados por outros arquivos e ativados dentro de um arquivo como um nó externo, sem duplicação de códigos. Ou senão, ser reconhecido internamente pelo navegador utilizado.

A concepção de formas se dá através da associação de elementos 3D geométricos pré-definidos, tais como Cones, Cilindros, Esferas, Caixas, etc que possuem atributos variáveis e que podem estar associados a texturas.

A modificação de fundos de ambiente está possibilitada pelo uso de nós específicos - *backgrounds*, - que permitem simular ambientes diferenciados que se assemelham a condições que variam de um lindo dia de sol, um dia nublado ou com muita neblina até a noites.

É possível o controle de aparência de elementos do cenário, bem como a inserção de diferentes formas de fontes de luz (pontuais, direcionais, ambiente), visando dar mais realismo ao cenário concebido. Recursos de acrescentar sons e filmes também estão disponíveis por utilização de nós específicos e são compatíveis com os principais formatos de áudio e vídeo: .mpeg, .mpg, .mid., .wav.

Podem ser elaborados *scripts* que facilitam as animações utilizando-se *Java* ou *JavaScript* de forma a complementar a troca de informações entre os elementos do mundo virtual. Esta propriedade provê possibilidade de animações e de dinamismo às formas concebidas e inseridas no cenário. O código em *JavaScript* pode fazer parte do arquivo original.

3.2. X3D

O X3D é um padrão aberto para distribuir conteúdos de realidade virtual em 3D, em especial pela Internet. Ele combina tanto dados geométricos como descrições de comportamentos instantâ-

neos em um simples arquivo que tem inúmeros formatos de transmissão, sendo que o padrão de codificação ideal é o XML (Extensible Markup Language). O X3D é a próxima revisão da especificação ISO VRML97, incorporando os avanços dos recursos disponíveis nos últimos dispositivos gráficos comerciais tanto quanto melhoras na sua arquitetura baseado nos anos de retorno da comunidade de desenvolvimento do VRML97. Este padrão esta sendo desenvolvido por um consórcio internacional, conhecido como Web3D, que tem o objetivo de propor e manter o padrão, e manter ele como um sistema aberto para a comunidade.

O X3D pode ser utilizado para Modelagem e animação, interação, interoperabilidade com outras tecnologias da web, treinamento, visualização científica, suporte a vendas, etc.

O XML foi adotado como sintaxe para o X3D para resolver um grande número de problemas reais, A sintaxe do VRML 97 é estranha para todos com exceção da comunidade do VRML. Ela é similar a sintaxe do grafo de cena do Open Inventor, na qual foi baseada e algumas notações de objetos. Todavia, a sintaxe dominante na Web atualmente é o XML. Linguagens de marcações tem provado ser a melhor solução para o problema de um longo ciclo de vida de arquivamento de dados.

Integração baseada em páginas XML vai direto ao problema de manter o sistema mais simples, assim mais pessoas podem desenvolver páginas web, tanto em conteúdo como implementações. Extensível suporte ao XML é esperado na última versão do Mozilla e do Internet Explorer. O X3D, como um formato que define informações visuais, é tipicamente o último estágio em uma linha de produção. Usando um conjunto de ferramentas disponíveis, como stylesheets, você pode trabalhar em qualquer formato nativo XML que você queira, e ver que uma representação 3D é tão trivial quanto um processo de transformação.

Existe uma Document Type Definition (DTD) para X3D, definida como parte do padrão. A URL do DTD é: http://www.web3d.org/specifications/x3d-3.0.dtd. Também existe um Schema para X3D aceito. Ele é definido como parte do padrão. A URL do Schema é: http://www.web3d.org/specifications/x3d-3.0.xsd.

O X3D é um padrão aberto que não tem royalties associados com ele - você pode usar livremente aonde você quiser. Não existe nenhum política de limite de Propriedade Intelectual para a tecnologia, além de existir um acordo com a ISO para publicar a especificação para o público sem nenhum custo.

Existe um esforço para um programa de conformidade para a especificação do X3D que tem o objetivo de prover consistência e confiabilidade para as implementações que suportem X3D pelos diversos vendedores das múltiplas plataformas, e para criar uma definição objetiva de conformidade com o padrão X3D. Somente produtos que passe pela conformidade poderão usar a marca registrada X3D.

Profiles e components são a nova forma do X3D de definir ambas, extensibilidade e o conjunto de serviços que o conteúdo dos usuários necessita. Um componente (*component*) define uma específica coleção de nós. Tipicamente esta coleção tem em comum um conjunto de funcionalidades - por exemplo a estruturas NURBS e as habilidades de texturização. Um componente (component) consiste da definição dos nós e um conjunto de níveis que prove um cada vez maior conjunto de implementações. Um nível simples requer apenas poucos nós e talvez uma seleção de campos a serem suportados, enquanto que níveis maiores requerem todos os níveis mais simples, mais nós mais complexos.

Por exemplo, o nível 1 de NURBS requer apenas linhas e curvas 2D básicas, enquanto que o nível 4 requer, costura, junção e superfícies de revolução. Um perfil (*profile)* é uma coleção de

componentes para um específico nível de suporte. Um perfil (profile) pode não conter outro perfil, porém ele necessita todos os mesmos componentes (components) e níveis como outro perfil (profile), e ainda mais. Todos os arquivos X3D requerem a definição do perfil (profile) que está em uso, na qual pode ser suprida com a requisição de componentes adicionais pelo usuário - ou por níveis maiores que aqueles providos pelo perfil (profile) ou que ainda não foram definidos no perfile (profile). Empresas podem criar novos componentes com suporte para seus produtos e submetê-los para a diretoria do X3D para sua aprovação.

Quando um componente é submetido, este contém um prefixo para a empresa que submete o componente similar como as extensões OpenGL que têm o prefixo da empresa que criou a extensão. Componentes vão sofrer os testes e revisões pela diretoria do X3D, o Consórcio Web3D e a comunidade como um todo. Uma vez que um componente seja aceito e implementado por mais de uma empresa, o prefixo muda para EXT_. Se o componente é classificado pela diretoria, ele então recebe o prefixo X3D_. A diretoria pode julgar que certos componentes são tão largamente adotados e importantes que deveriam ser incluídos em um conjunto de modificações para a especificação ISO oficial. Uma vez que um grupo de componentes (components) e/ou perfils (profiles) são julgados importantes para a inclusão através das várias aplicações, uma nova versão do X3D pode ser criada incluindo, por padrão, um conjunto de perfis (profiles).

Uma nova versão implica mais funcionalidade que as versões anteriores. Empresas podem criar navegadores, ferramentas, importadores e exportadores X3D na qual suportam diferentes versões, perfils (profiles) e componentes (components). Por exemplo, um exportador para o 3DSMax que é usado nos ambientes de produção de video-games pode suportar somente o perfil Interchange,

enquanto que um plug-in para navegador da Internet pode suportar o perfil Immersive ou Full.

As empresas não precisam suportar todos os Profiles, Components e Levels. Perfis e componentes existem para que as empresas precisem somente suportar perfis e componentes para suas necessidades. Por ter perfis, seus produtos podem ter certeza que o conteúdo que eles lerem irá funcionar nas suas aplicações, e que aquele conteúdo que eles criaram irá funcionar em outras aplicações que suportem seus componentes ou perfis. Muitas empresas não vão querer suportar uma especificação grande e complexa como o VRML97. Porém a estrutura modular do X3D significa que eles podem começar suportando perfis (profiles) e componentes (components) simples, e gradualmente adicionar perfis adicionais conforme eles se sentirem preparados. Por haver empresas capazes de desenvolver componentes e submetê-los, o X3D força os avanços da indústria a serem rápidos e eficientes. Isto também garante que o X3D crescerá e florescerá, e não se tornará tecnicamente obsoleto, como o padrão anterior se tornou - Figura 27.

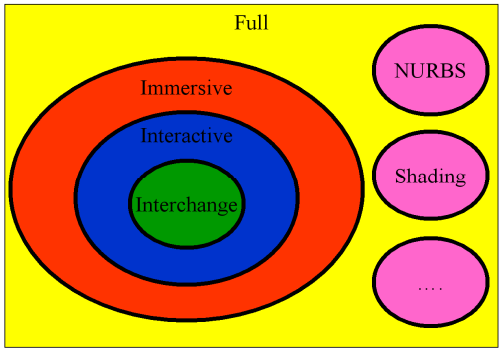


Figura 27 - Diagrama de Profiles

3.3. Grafo de Cena

3.3.1. Definições básicas

A concepção de ambientes 3D deve começar pela definição de uma hierarquia e de seus componentes. Como exemplo, para a construção de um avião, suas partes devem ser definidas e agrupadas, hierarquicamente, através de um grafo de cena, como pode ser visto na Figura 28.

Um nó é um componente construtivo do grafo de cena. No caso da Figura 28, cada asa será um nó. Os nós podem ser outros grafos de cena e, sucessivamente, definirem como cada instância primitiva será usada para montar o cenário final.

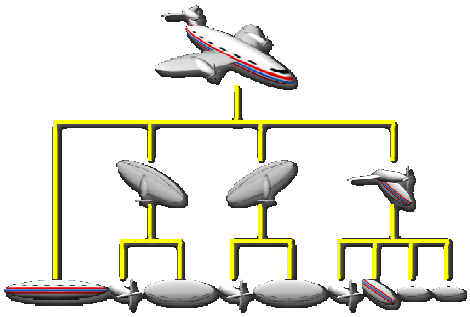


Figura 28 - Grafo de Cena para uma aeronave simples

Assim, a construção dos grafos de cena requer a união, usando a estrutura de árvores ou de grafos direcionados acíclicos – DAG – de primitivas gráficas. Cada componente do cenário final será uma associação de primitivas, que, no grafo de cena representa um nó. Neste contexto, é muito importante o conhecimento das primitivas disponibilizadas pela linguagem.

Cada primitiva Shape está associada a dois nós (necessariamente):

- nó aparência: que descreve a aparência final da forma;
- nó geometria: usado para definir a geometria da forma.

Estruturalmente, a sintaxe dos nós em VRML e X3D que permite acomodar a definição das primitivas tem, portanto, o aspecto:

Forma Geométrica {

Definição da aparência {}

Definição da geometria{}}

3.3.2. Atributos

Os atributos declarados em cada nós podem conter um valor simples ou múltiplos, isto é definido pelas duas primeiras letras. Caso o tipo do atributo comece com SF (Single Field) somente suportará um valor, caso seja MF (Multiple Fields), este suportara vários valores. Estes também podém se referenciar outros nós com o tipo abstrato básico SFNode and MFNode.

Dentre os tipos básicos de atributos temos:

- Bool - para valores booleanos
- Color - para a descrição de RGB de uma dada cor
- ColorRGBA- para a descrição de RGBA de uma dada cor
- Double – para valores reais com altíssima precisão
- Image – para imagem
- Int32 - para valores inteiros

- FLoat - para valores reais
- Node - para a descrição de um nó
- Rotation – para os valores de uma rotação
- String – para um texto
- Time – para um valor de tempo
- Vec2d, Vec2f – para um vetor de 2 elementos de alta e menor precisão
- Vec3d, Vec3f – para um vetor de 3 elementos de alta e menor precisão

3.3.4. Codificação VRML97

Usando a idéia básica apresentada na seção anterior, serão detalhados, nesta seção, o formato de codificação em VRML 97.

3.3.4.1. Formas Geométricas Básicas e alguns exemplos

Para começar a discutir as formas geométricas, será iniciada a definição de um cone. Tal definição teria a descrição textual abaixo como arquivo VRML:

```
Shape { appearance Appearance {}
        geometry Cone {
                height 2.0
                bottomRadius 1.0 }}
```

Observe que não foram feitas quaisquer definições de aparência, levando a forma gerada, de acordo com o texto acima, a ter aparência branca.

3.3.4.2. Textos

É possível inserir textos nos ambientes 3D concebidos através do nó *Text*, cuja sintaxe foi apresentada no grafo de cena.

A apresentação do texto pode ser formatada com uso do nó *FontStyle*, que permite definir a fonte que será utilizada, seu tamanho, espaçamento de caracteres, como o texto será justificado etc.

3.3.4.3. Transformações Geométricas

O nó VRML que permite transformações geométricas (translação, rotação e escala) sobre as formas definidas é o nó *'Transform'*. *'Transform'* é, de fato, um nó de agrupamento, que, para todas as formas de seus filhos (*'children'*) aplica as transformações definidas nos campos *'scale'*, *'translation'* e *'rotation'*:

- 'Scale': escala em X,Y,Z, respectivamente;

- 'Translation': translação nos eixos X,Y,Z respectivamente;

- 'rotation': rotação em X, Y, Z pelo argumento (em rad) do quarto campo;

- 'children': lista dos elementos a qual devem ser aplicadas as transformações.

Até meados do ano de 2000, a edição de textos que descreviam formas em VRML não dispunha de um editor de textos exclusivo para este fim. De olho neste nicho de mercado, a empresa *Parallel Graphics* lançou o VrmlPad®, que facilita sobremaneira a edição de tais textos.

3.3.4.4. Alterando Aparência - nó appearance

De maneira a acomodar a definição adequada da aparência de uma forma, o nó *appearance* permite a definição de variáveis relativas à cor e textura de uma dada forma. É importante ressalatar que a inserção de uma textura elimina qualquer definição de cor aplicada ao componente.

No caso de aplicação de uma textura a uma dada forma, podem ser usadas imagens do tipo JPEG, GIF e PNG, com uso do nó

'Material'. São campos possíveis de serem utilizados: *ImageTexture*, *PixelTexture* e *MovieTexture*.

Há possibilidade de controlar o mapeamento de texturas com utilização dos nós *TextureCoordinate* e *TextureTransform*. O nó *TextureTransform* pode ser usado como um valor do campo de valor no nó *Appearance*.

3.3.4.5. Reutilizando Formas, Elementos e Definições

Quando se faz necessário reutilizar uma dada forma, são usados os termos DEF e USE. DEF define o nome de um dado elemento, forma ou definição. A reutilização deste nome a partir de sua chamada por USE dispensa a redefinição.

Podem ser inseridos elementos definidos em um dado arquivo dentro de um cenário 3D, com uso do nó *Inline*. A sintaxe de *Inline* é dada por:

```
Inline{ url []
        bboxCenter -1.0 -1.0 -1.0
        bboxSize 0.0 0.0 0.0 }
```

3.3.4.6. Compondo formas como conjunto de Faces

A utilização do nó *IndexedFaceSet* permite construir conjuntos de faces interligadas, de forma que tal combinação gere efeitos de formas complexas. Na descrição de um conjunto de faces, uma lista de pontos coordenados deve ser explicitada, além das conexões entre estes pontos.

Supondo que se deseje a construção de um cubo seriam necessárias as coordenadas da parte superior do cubo e da parte inferior, além das descrições de ligações entre estes pontos.

3.3.4.7. Fundos e Cenários

O nó *Background* permite criar fundos para os mundos virtuais usando uma caixa de textura, elementos de piso e elementos do céu. Há possibilidade de aplicação de uma textura ao background.

3.3.4.8. Iluminação

VRML provê diversos nós para uma iluminação adequada de cenário. Todos estes nós apresentam os seguintes campos: *color*, *ambientIntensity* e *intensity.* As funções dos nós podem ser:

- iluminação do tipo direcional, com os raios emanando de forma radial em todas as direções: *PointLight;*
- iluminação do tipo direcional, com os raios pertencem a um pincel de luz paralela: *DirectionalLight*;
- iluminação do tipo 'spot', com os raios em um pincel na forma de cone, com a fonte de luz no ápice do cone: *SpotLight*;

Em todos os casos, a contribuição de uma fonte de luz é definida pelo campo *intensityColor*, enquanto que a contribuição da luz ambiente é dade em função do valor do campo *ambientIntensity*. Objetos com textura não são afetados pela fontes de luz. Pode-se desligar a luz na cabeça do usuário (navegador) através da definição a *headlight FALSE* no nó *NavigationInfo*, de acordo com a sintaxe:

```
NavigationInfo {headlight FALSE}
```

3.3.4.8.1. DirectionalLight

O nó *DirectionalLight* gera um pincel de luz paralelo, que tem a orientação dada pelo campo *direction*, a cor definida no campo *color* e a intensidade definida em *intensity*.

3.3.4.8.2. PointLight

O pincel de luz, equivalente à PointLigth, puntiforme tem a localização no ambiente 3D dada pelo campo *locaion,* o raio da esfera de iluminação dado pelo campos *radius,* a intensidade definida em *intensity*. Os campos *intensity* e *color* combinam-se para definir o nível de cor provida pela fonte de luz em definição. Assim, a equação que define a cor da luz é:

lightColor = *color* X *intensity.*

A intensidade da luz ambiente é dada por:

lightAmbientColor = *color* X *intensity* X *ambientIntensity*.

3.3.4.8.3. SpotLight

O nó *SpotLight* é capaz de gerar uma fonte de luz na forma de um spot que pode ser usada para iluminar determinado(s) elemento(s) de cena. Este "cone" de iluminação permite controle de abertura, intensidade e cor.

3.3.4.9. Controlando Detalhamento de Elementos - nó LOD

A técnica de controlar o detalhamento de elementos aparece na Computação Gráfica quando simuladores de vôo foram usados para treinamento de pilotos. Em tais simuladores, ao apresentar o cenário de um terreno, onde uma dada aeronave deveria pousar, o realismo é muito importante.

Para dar mais realismo a tais cenas, os terrenos necessitavam de grande detalhamento, associados à presença de edificações, árvores, carros etc. Para melhorar o desempenho de tais simuladores, observa-se que não é necessária a definição de muitos detalhes quando o elemento está situado a grande distância do observador e o inverso é válido para elementos muito próximos.

A técnica de controlar o detalhamento dos elementos está associada à criação de diferentes versões do mesmo que serão apresentadas ao navegante à partir de sua distância à forma em questão. O nó *LOD* ativa a chamada de cada forma em função de tais distâncias.

3.3.4.10. Animando as formas

3.3.4.10.1. Animações básicas

Para tornar o mundo mais dinâmico, podem ser animadas as posições, orientações e escalas das formas que estão definidas. Uma animação é uma mudança de algo em função de um intervalo de tempo. Uma animação requer dois elementos:

- Um elemento de tempo (relógio) que controla a animação;
- A descrição de como alguma coisa altera-se ao longo do tempo da animação.

A animação de algum elemento deve ser feita imaginando que o mesmo pode sofrer alterações de posição, orientação e escala em função de um dado tempo (fração do tempo em questão). O nó responsável pelo controle de tempos (e frações) é o nó *TimeSensor.*

O nó *TimeSensor* por si é incapaz de promover a animação. A associação do mesmo com nós que controlam posição, orientação e escala é essencial. O nó que controla a posição de uma forma em função de uma fração de tempo é o nó

O nó *PositionInterpolator* necessita de receber uma informação de tempo, enviada por um elemento de tempo (geralmente um *TimeSensor*). Com tal informação, compatibiliza a posição da forma (*keyValue*) com a fração de tempo (*key*). A associação é sempre feita por uma rota que combina a saída de tempo do *TimeSensor* com a entrada do nó *PositionInterpolator*.

Há necessidade de inserir rotas, de forma que valores obtidos em um nó sejam mapeados para outro nó, possibilitando a animação.

Se, ao invés de controlar posições, fosse necessário controlar a orientação, o nó adequado seria o nó *OrientationInterpolator*, que permite que sejam aplicadas rotações às formas desejadas.

Onde o campo *key* define as frações de intervalos de tempo e o campo *keyValue* define as frações da animação (na forma de rotação).

A última forma de animação relativa às transformações geométricas é a mudança de escala. O nó que permite mudança de escala é o nó *PositionInterpolator* (o mesmo que permite a animação de posição). Neste caso, no entanto, devem ser colocados valores de escala no campo *keyValue* e a rota de saída de valores deve ser entrada para valores de escala ao invés de valores de posição.

3.3.4.10.2 Controlando as animações

De maneira a controlar as animações, podem ser inseridos nós sensores de ações dos usuários. Há três formas de ações que podem ser percebidas:

- Movimento (*Move*): sem pressionar o mouse, o usuário move o cursor sobre um item qualquer;
- Clique: quando o cursor está sobre um elemento, o botão do mouse é pressionado e liberado, sem movimento do mesmo;
- Arraste (*Drag*): com o cursor sobre um item, o botão do mouse é pressionado e, mantido pressionado, o mouse é arrastado.

O nó capaz relativo à captura de toque é o nó *TouchSensor*. A sintaxe do nó *TouchSensor* é:

```
TouchSensor  { enabled       TRUE }
```

O nó *TouchSensor* tem sensibilidade ao toque (campo *touchTime*), ao cursor estar sobre uma forma (*isOver*) e pressionado (*isActive*).

De maneira a permitir que um dado elemento gráfico seja manipulado pelos movimentos do usuário, existem três nós distintos;

- *PlaneSensor*: converte as ações do usuário em movimentos em um plano 2D;
- *SphereSensor*: converte as ações do usuário em movimentos ao longo de uma esfera que envolve a forma;
- *CylinderSensor*: converte as ações do usuário em movimentos ao longo de um cilindro definido sobre um dos eixos.

Os campos maxPosition e minPosition definem os limites da translação que será aplicada à forma.

3.3.4.11. Controlando o Ponto de Vista e a Navegação

A navegação pode ser controlada pelo nó *NavigatonInfo*.

A velocidade de movimentação do *avatar* é dada pelo campo *speed*. Os tipos de navegação, relativos ao campo *type* são:

- "WALK": quando o usuário caminha no mundo e é afetado pela gravidade;
- "FLY": quando o usuário pode se mover sem ser afetado pela gravidade;
- "EXAMINE": quando o usuário fica estático, mas pode se mover ao redor do mundo em diversos ângulos;
- "NONE", quando o usuário não pode controlar os movimentos.

O campo *headlight* define se o *avatar* terá ou não uma fonte de luz na sua cabeça. Se TRUE, uma fonte de luz estará ligada na cabeça do *avatar*.

Um ponto de vista (*Viewpoint*) é uma posição pré-definida com uma dada orientação no cenário. Podem ser alteradas as formas de navegação do usuário e suas posições de visualização.

O campo *orientation* define um eixo de rotação e um ângulo de rotação. O campo *position* define a posição de visualização do *avatar* em relação ao cenário apresentado.

3.3.4.12. Adicionando Sons e Filmes ao Cenário

De maneira a permitir que o cenário seja mais realístico, podem ser adicionados sons, na forma de sons ambientes ou de sons controlados por animações. Os nós relativos a adição de sons são:

- *AudioClip*: nó que suporta alguns tipos de sons digitais (não é permitido o mp3): MIDI e WAV;
- *Sound*: nó que cria um emissor de som que pode ser ouvido dentro de uma região elipsoidal;
- *MovieTexture*: permite a inserção de filmes (movie).

3.3.4.13. Sentindo a proximidade do usuário

Como visto, pode se usar o nó *TouchSensor* para detectar quando o usuário toca uma forma do cenário. Há, no entanto, três outros nós capazes de identificar ações do usuário, que podem ser utilizados para controlar animações, da mesma forma que o nó *TouchSensor*:

- *VisibilitySensor*: usado para identificar a visibilidade de um observador, através de uma região que tem o formato de uma caixa (região de visibilidade);
- *ProximitySensor*: nó que permite detectar quando o observador entra e/ou se move em uma região fechada (em formato de uma caixa);
- *Collision*: permite detectar quando o usuário colide com alguma forma, sendo também um nó de agrupamento

(como *Group* ou *Transform*). Este nó gera o tempo absoluto em que o choque ocorreu ou alerta o navegador o fato;

3.3.4.14. Unindo Cenários Virtuais e Links

Usando VRML é possível simular que, ao toque em uma porta, um novo cenário será carregado. Tal ativação pode carregar um novo cenário 3D, uma página Web ou uma animação. O nó adequado à tais simulações é o nó *Anchor*.

É importante destacar o campo *url* que conterá a informação da página com a qual se deseja ativar ao tocar os elementos descritos em *children*.

3.3.5. Combinando VRML e JavaScript

3.3.5.1. Introdução

Os nós já apresentados para animação, tais como *TouchSensor*, *TimeSensor*, *PlaneSensor* etc são muito limitados quando se desejam animações mais complexas ou controle de elementos de cena a partir de equações (matemáticas, físicas, químicas etc). Em tais situações é imperativa a necessidade de uso dos nós *Script*, que associam VRML com JavaScript ou Java.

Um nó Script pode ser entendido como uma forma particular de controle ou de sensor. Como qualquer nó de controle ou de sensor, este nó requer uma lista de campos (*field*), eventos de entrada (*eventIns*) e eventos de saída (*eventOuts*). A descrição do nó deve complementar a definição destes campos, dando a eles uma dada finalidade . Um nó *Script* deve ser entendido como um nó que recebe informações de outros nós através dos eventos de entrada, processa-as e envia informações na forma de eventos de saída.

Logicamente, não serão encontrados nós *Script* independentes da descrição de rotas, que mapeiam a troca de informações entre

nós externos e o nó *Script*. Uma exigência importante é a necessidade de equivalência de tipos de elementos entre os nós que trocam informações.

3.3.6. Codificação X3D

Os dois padrões VRML97 e X3D são muito semelhantes o X3D aproveita o trabalho desenvolvido pelo VRML97 solucionando diversos problemas e pontos em aberto. Dentre as modificações temos a maior precisão com a iluminação e modelo de eventos e a troca de certos nomes de campos para uma maior consistência.

As maiores mudanças podem ser sumarizadas da seguinte forma:

- Capacidades do grafo de cena expandidas
- Modelo de programação de aplicações revisado e unificado
- Multiplos formatos de codificação, descrevendo o mesmo modelo abstrato, incluindo XML
- Arquitetura modular permitindo uma faixa de níveis para serem adotados e suportados por diversos tipos de mercados
- Estrutura da especificação expandida

O grafo de cena X3D - o coração de uma aplicação X3D - é quase idêntico a do grafo de cena VRML97. O projeto original da estrutura e tipos de nós do grafo de cena VRML97 foi baseado no OpenInventor. Modificações foram feitas no grafo de cena X3D, primariamente em função de incorporar os avanços nos dispositivos gráficos comerciais, via a introdução de novos nós e campos dos dados. Adicionalmente mudanças menores foram feitas de forma a esclarecer, como as de ser mais preciso no sistema de iluminação e modelo de eventos, e prover acesso aos valores de transparência nos campos de cores.

O X3D tem uma única interface de programação de aplicações (API). Esta difere do VRML97 na qual tem uma API de script

interna mais uma API externa. A API unificada do X3D esclarece e resolve diversos problemas que existiram no VRML97 resultando em uma implementação mais robusta e confiável. Conexões por ECMAScript e Java são definidas e ECMAScript é um requisito para a conformidade de uma aplicação. O ECMAScript na verdade é um sistema Java interpretado porém devido a restrições da Sun o nome Java não pode ser usado livremente.

O X3D suporta múltiplas codificações de arquivos, a própria VRML97 e adicionalmente Extensible Markup Language (XML), além disso esta sendo estudada uma codificação binária, que seria muito mais comprimida e eficiente para a leitura e transferência de arquivos, contudo esta ainda não esta consolidada. A codificação XML possui uma vantagem pois permite uma suave integração com serviços de web (web services) e arquivos e transferência de dados entre plataformas inter-aplicações (cross-platform inter-application). Cada codificação tem suas vantagens para diferentes usos. Todas as codificações suportam todo o conjunto de características do X3D.

O XML é um padrão definido para a troca de dados em multiplataformas. Existe uma série de bibliografias que podem ser estudadas. Basicamente o XML consistem em uma série de nós que são organizados pelos sinais de menor e maior (“<” e “>”) e também um sinal de barra (“/”) para informar o final de um nós

Vamos apresentar aqui dois exemplos de arquivos, um no formato clássico, que pode ser facilmente reconhecido pelos sinais de chaves e colchetes. Percebam que ele tem a mesma estrutura do VRML97, porém com diferenças no cabeçalho. Logo a seguir é apresentado um na codificação XML, nele é possível verificar a codificação XML, que é mais comum para usuários em geral, que lidam com formatos de codificação.

- Exemplo de um arquivo em X3D Classico

```
#X3D V3.0 utf8
```

```
Profile Immersive
NavigationInfo {
       type [ "ANY" ]}
Transform {
       children [
              Shape {
                     appearance Appearance {
                            material Material {
                                   diffuseColor 1.0 1.0 1.0}}
                     geometry Sphere {}}]}
```

- Exemplo de um arquivo em X3D codificado em XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE                    X3D                    PUBLIC
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile="Immersive">
       <Scene>
              <NavigationInfo type="ANY"/>
              <Transform>
                     <Shape>
                            <Appearance>
                                   <Material  diffuseColor=”1.0
                            1.0 1.0”/>
                            </Appearance>
                            <Sphere/>
                     </Shape>
              </Transform>
       </Scene>
</X3D>
```

O X3D emprega uma arquitetura modular para prover uma maior estensibilidade e flexibilidade. A maioria dos domínios de aplicações não necessitam de todos os recursos do X3D, tampouco

todas as plataforma suportam a gama de funcionalidades definidas na especificação. Os recursos do X3D são agrupados em componentes (components) que podem ser suportados pela implementação em uma mistura de capacidades para atingir as necessidades de um mercado ou plataforma em particular. O X3D também introduz o conceito de perfis (profiles) - uma pré-definida coleção de componentes comumente encontrados em certos domínios de aplicações , plataformas, ou um cenário de uso, por exemplo trocas de geometrias entre ferramentas de modelagem. Diferente do VRML97, na qual requer um completo suporte de suas funcionalidades para estar em conformidade, o X3D permite vários graus de suporte do padrão para atingir uma variedade de necessidades. O mecanismo de componentes do X3D também permite desenvolvedores a implementar suas próprias extensões de acordo com um rigoroso conjunto de regras, ajudando a evitar a "torre de babel" das extensões dos desenvolvedores que surgiram no VRML97 nos últimos anos.

Finalmente, a especificação do X3D por si mesma tem sido reestruturada, permitindo um ciclo de vida mais flexível para acomodar as evoluções dos padrões. O padrão X3D é divido em três especificações separadas lidando com conceitos abstratos de arquitetura, codificação de arquivo e acesso de linguagem de programação. Este enfoque permite à especificação mudar durante o tempo e facilita sua adoção através da ISO para especificar partes da especificação conforme necessário.

3.3.6.1. Transição do VRML para o X3D

A mudança básica que deve ser efetuada caso para a conversão entre VRML97 e o “X3D clássico” é mudar o cabeçalho de:

#VRML V2.0 utf8 para

#X3D V3.0 utf8

Profile Immersive

Existe agora a necessidade de se definir o profile para que o navegador saiba o grau de recursos a serem utilizados, uma outra diferença é a forma de se chamar scripts no X3D. No VRML97 as chamadas eram feitas com 'javascript' ou 'vrmlscript'. O X3D usar o chamado 'ecmascript'. ECMAScript é a versão padronizada do JavaScript.

Existem diversas ferramentas disponíveis que podem fazer a conversão entre VRML97 e X3D. Se você tem um arquivo relativamente simples que não contem scripts ou externprotos então você pode editar manualmente o texto para trocar o cabeçalho e inserir a nova declaração PROFILE que você estará com o documento pronto. Qualquer coisa mais complexa vai requerer uma ferramenta e idealmente um razoável trabalho de conhecimento com a nova especificação (diversas áreas funcionais foram mudadas para serem mais rigorosas ou apenas são simplesmente diferentes).

Existem algumas ferramentas para a conversão de conteúdo, uma delas é o X3D-Edit, porém é possível também a utilização de algumas ferramentas por linha de código. Usualmente basta um programa capaz de converter os nós de um formato de codificação para outro.

Um cuidado que se deve tomar é que um arquivo VRML97 não é exatamente um arquivo X3D. Um navegador puramente X3D poderá não será capaz de ler um arquivo VRML97 logo que existem mudanças na sintaxe que levam os dois a incompatibilidades. Entretanto, a maioria dos navegadores que suportam o padrão X3D também suportam o padrão VRML97. Provavelmente no futuro a maioria dos navegadores ira suportar arquivos X3D, e alguns ainda suportarão o formato VRML97.

Existem mudanças entre o VRML97 e o X3D sutis e não tão sutis. Algumas são:

- Os arquivos estão agora estruturados para definir as capacidades necessárias como parte do cabeçalho. Isto necessita que ao menos seja definido o perfil (profile) e qualquer componente (component) extra.
- Externprotos agora são usados somente para definir conteúdo externo ao arquivo X3D. Eles não podem ser usados para prover mecanismos de extensão para o navegador. A forma de prover extensões específicas para o navegador é através de componentes (components) adaptados.
- Nomes de acesso para campos mudaram de eventIn, eventOut, field e exposedField, para inputOnly, outputOnly, initializeOnly e inputOutput, respectivamente.
- Scripts podem ter campos inputOutput (exposedFields) definidos.
- Um nome DEF não pode ser multi definido mais.
- Todo conteúdo de leitura é desacoplado. Onde no VRML97 necessitaria que scripts fossem carregados antes da execução começar, X3D remove esta necessidade e define que o arquivo começa a executar primeiro e depois de certo tempo, recursos (scripts, texturas, sons, inlines, externprotos) são carregados.
- O modelo de execução entre o conteúdo de um script e o grafo de cena é rigorosamente definido e precisamente controlado. Onde o VRML97 permite um script multi-threaded arbitrariamente trocar o grafo de cena em qualquer ponto, o X3D define somente certos ponto onde estas alterações podem ser feitas.
- A execução e modelo de programação para scripts é consistente entre linguagens de programação e aonde quer que você esteja, dentro ou fora do navegador - uma definição da API é regra geral.
- Rigorosa definição de conjunto de definições de tipos abstratos para nós.

Os arquivos X3D devem seguir as seguintes regras:

Codificação X3D	Extensão de arquivo	Extensões Compactadas	Tipo MIME
Clássico VRML	.x3dv	.x3dvz, .x3dv.gz	model/x3d+vrml
XML	.x3d	.x3dz, .x3d.gz	model/x3d+xml
Binário	.x3db	.x3dbz, .x3db.gz	model/x3d+binary

- os tipos MIME ainda estão sendo aprovados

3.3.6.2. Interação com Usuário

A interação do usuário com o ambiente virtual utilizando VRML pode se tornar um pouco limitada, pois não é possível definir ações diretamente associadas às teclas do teclado, ou outro dispositivo externo. Nestes ambientes, o usuário fica dependente das teclas de atalho dos *plug-ins* de geração do ambiente virtual. Este problema foi solucionado no X3D, através da utilização dos nós que compõem o componente *KeyDeviceSensor*.

Estes nós são responsáveis por gerar eventos quando o usuário pressiona alguma tecla ou conjunto de teclas do teclado. Eles representam nós extremamente úteis, pois permitem que o construtor do ambiente virtual possibilite o uso de todas as teclas do teclado para realizar ações específicas dentro do ambiente virtual.

Referências

AMES, L. A.; NADEAU, R.D.; MORELAND D. **VRML Sourcebook - Second Edition**, John Wisley & Sons, Inc - USA, 1997.

CARDOSO, A.; LAMOUNIER E.; TORI R. Sistema de Criação de Experiências de Física em Realidade Virtual para Educação a Distância. In: II WORKSHOP BRASILEIRO DE REALIDADE VIRTUAL, **WRV´99**, Marília, São Paulo, Brasil, 1999, p. 174-181.

CARDOSO, A. Alexandre Cardoso - Página do Pesquisador. Contém informações sobre aplicações de Realidade Virtual, pesquisa e publicações do pesquisador, tutoriais sobre VRML e artigos indicados para leitura. Disponível em: <http://www.alexandre.eletrica.ufu.br> Acesso em Mar./2007

CHEN, S.; MYERS, R.; PASSETO, R. The Out of Box Experience - Lessons Learned Creating Compelling VRML 2.0 Content. In: Proceedings of the Second Symposium on Virtual Reality Modeling Language, p. 83-92, 1997.

EcmaScript Specification (JavaScript), disponível em http://www.ecma.com, Search, EcmaScript, EC 262

ELLIS R. S. What are Virtual Environments**. IEEE Computer Graphics and Applications**, p. 17-22, Jan. 1994.

LEMAY, MURDOCK, COUCH **3D Graphics and VRML 2** Sams.Net, Indiana - USA – 1996

Introducing X3D by Leonard Daly, Don Brutzman, Nicholas Polys, Joseph D. Williams, SIGGRAPH 2002. Curso introdutório de X3D, disponível em http://realism.com/Web3D/x3d/s2002, acessado em Agosto 2004

MATSUBA, N.; STEPHEN; ROEHL, B. Bottom, Thou Art Translated: The Making of VRML Dream. **IEEE Computer Graphics and Applications**, v. 19, n. 2, p. 45-51, Mar./Apr. 1999.

NSS X3D Modeler. Nature Simulation Systems. O sítio disponibiliza o software NSS X3D Modeler. Disponível em: <http://www.xmodeler.com/>. Acesso em: 12 mar. 2007.

Scenario Authoring and Visualization for Advanced Graphical Environments (Savage) project at http://web.nps.navy.mil/~brutzman/Savage/contents.html

Vapor Tutorial VRML - Tutorial de VRML com exemplos e fontes. Disponível em http://web3d.vapourtech.com/. Acesso em Set./2003

VRMLPAD. Parallellgraphis. O sítio disponibiliza diversos programas computacionais de grande utilidade para desenvolvimento de ambientes virtuais em VRML, inclusive o programa VRMLPad. Disponível em: <http://www.parallellgraphics.com>. Acesso em: 02 nov. 2000.

WEB3D, "Web 3D Consortium", WebSite que disponibiliza informações a respeito do padrão X3D, Disponível em <http://www.web3d.org>, visitado em Janeiro/2006

Capítulo 4

Java 3D

José Remo Ferreira Brega[1,2], Antônio Carlos Sementille[1,2], Ildeberto Aparecido Rodello[1]

[1]UNIVEM - Centro Universitário Eurípides de Marília - Marília – SP

[2]UNESP – Universidade Estadual Paulista – Campus de Bauru – SP

{remo, semente, rodello}@univem.edu.br

Abstract

This Chapter presents the API Java 3D, starts with the basic structure of a program, until the more complex and elaborated environment implementation. Its classes and methods are presented for the creation of forms and support for diverse devices of entrance and exit, illustrating the form of implementation by means of examples.

Resumo

Este Capítulo apresenta a API Java 3D, discutindo desde a estrutura básica de um programa, até a implementação de ambientes mais complexos e elaborados. São apresentados e discutidos as principais classes e métodos para a criação de formas e suporte para diversos dispositivos de entrada e saída, ilustrando a forma de implementação por meio de exemplos.

4.1. Introdução

Java 3D é um conjunto de classes desenvolvido pela Sun Microsystems (SUN, 2002) com o objetivo de ser uma interface para o desenvolvimento e apresentação de programas em Java com conteúdo tridimensional, otimizando renderizações.

Java 3D segue o paradigma de escreva uma vez e rode em qualquer lugar (*Write once, run anywhere*), fornecendo suporte para várias plataformas, vários dispositivos de visualização e vários dispositivos de entrada.

4.2. Vantagens e desvantagens

Como principais vantagens de Java 3D [SELMAN, 2002]: Implementação somente em Java, oferecendo facilidades para a portabilidade; Suporte para aplicações distribuídas, desde o uso de Java RMI e para a Internet com *Applets* e *Servlets* desenvolvidas facilmente em Java [SUN, 2006]; o emprego de grafo de cena; Suporte para alguns dispositivos não convencionais A Descrição da cena; e possuir mecanismos para otimização da renderização dos grafos de cena.

Java 3D também oferece algumas desvantagens, dentre as quais [SELMAN, 2002]: Não conseguir atender a todas as características oferecidas por linguagens gráficas mais apuradas como *OpenGL*; Não permitir aos desenvolvedores um controle maior sobre o grafo de cena; A existência do coletor automático de lixo de Java; e A dificuldade na distribuição das aplicações para os usuários, pela necessidade da correta instalação da API Java 3D associada com a aplicação.

4.3. Estrutura da API Java 3D

Um programa Java 3D cria instâncias de objetos, colocados em uma estrutura hierárquica chamada de grafo de cena (*scene gra-*

ph). Esta estrutura, segue um modelo que especifica o conteúdo de um universo virtual e como este conteúdo será renderizado.

Em um grafo de cena são definidos a geometria, os sons, as luzes, a localização, a orientação e a aparência dos objetos Java 3D, os quais são organizados de acordo com o modelo pai-filhos de relacionamento.

Como toda estrutura baseada em grafos, o grafo de cena é composto por nós (*nodes)* e arcos (*arcs*). Um nó representa um elemento e um arco representa o relacionamento entre os elementos. A Figura 4.1 apresenta os nós e os arcos, presentes em um grafo de cena Java 3D.

Os primeiros dois símbolos, *Virtual Universe* e *Locale*, representam objetos de classes específicas. Os outros três símbolos, *Group, Leaf* e *Node Component,* indicam subclasses de objetos específicos. Por fim, o último símbolo é usado para representar um objeto de qualquer outra classe.

As linhas em formato de setas indicam os relacionamentos entre os objetos. As linhas contínuas representam um relacionamento pai-filho entre dois objetos e as linhas pontilhadas, por sua vez, indicam uma referência para um objeto.

Em cada grafo de cena existe um objeto *VirtualUniverse*, que tem uma lista dos objetos *Locale.* Normalmente existe um universo por aplicação. Ele é a base sobre a qual o grafo de cena é montado.

Os objetos *Locale*, por sua vez, fornecem um ponto de referência em termos de localização dos objetos no universo virtual, ou seja, uma posição no universo na qual irão ser colocados os grafos de cena. Normalmente existe um *Locale* por universo.

Um grafo de cena é construído de nós em relacionamentos pai-filho, formando uma árvore, que possui um nó raiz. Outros nós

são acessíveis seguindo os arcos a partir da raiz. Um grafo de cena é formado de árvores originadas em um objeto *Locale* e só existe um caminho da raiz de uma árvore a cada uma de suas folhas.

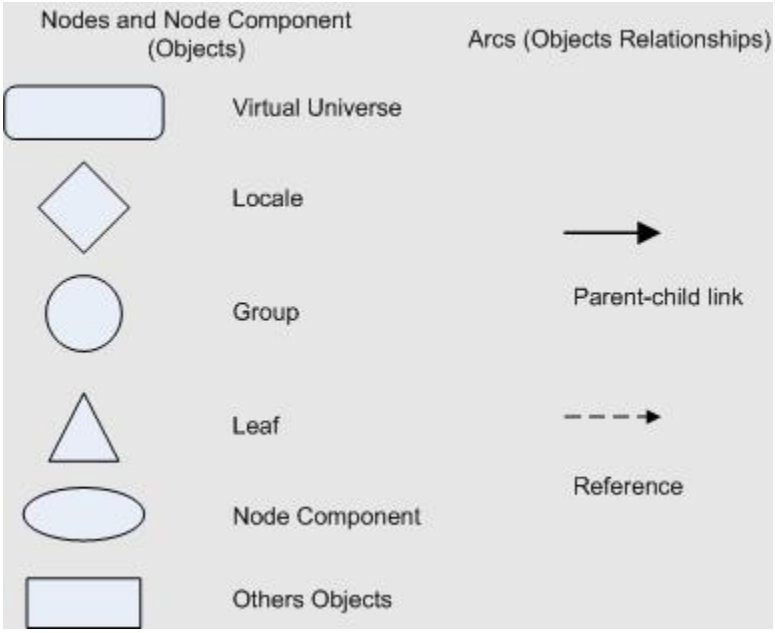


Figura 4.1 – Símbolos para representação de objetos no grafo de cena [BOUVIER, 2002].

Cada caminho especifica o estado e informações sobre determinada folha, como local, orientação, e tamanho de um objeto visual. Assim os atributos visuais de cada objeto dependem apenas de seu caminho no grafo. Java 3D aproveita esta característica e renderiza os nós na ordem que determinar ser mais eficiente. O programador geralmente não tem controle sobre a ordem de renderização dos objetos. Além disso, as representações gráficas de um grafo de cena podem servir como documentação para programas escritos em Java 3D.

A raiz de cada subgrafo do grafo de cena (ou *branch graph*) é representado por um *BranchGroup*. Normalmente existem diversos *branch graphs* por *locale*. Existem duas categorias de subgrafo: *view branch graph* e *content branch graph*. Normalmente existem vários *branches* por *locale.*

O *content branch graph* especifica o conteúdo do universo virtual (geometria, aparência, comportamento, localização, som e iluminação), enquanto o *view branch graph* especifica os parâmetros de visualização tais como localização e direção.

A Figura 4.2 mostra um exemplo de um grafo de cena. Neste grafo observa-se a presença de um *virtual universe* e um *locale*. Abaixo do *locale* pode-se observar a presença de dois subgrafos.

As principais vantagens na estruturação de ambiente em grafo de cena são as facilidades de identificar e corrigir erros antecipadamente e a possibilidade de buscar otimizações na renderização da cena.

Os objetos discutidos até o momento fazem parte de uma hierarquia de classes. Além dos itens já discutidos, destaca-se:

View: descreve os parâmetros e as políticas de visualização. Está ligado a uma classe *ViewPlatform*, que define um ponto de visão dentro de um *locale*. Em conjunto com um nó *Transfom3D*, o *ViewPlatform* define um ponto de referência para o posicionamento e orientação do usuário dentro do mundo virtual.

PhysicalBody: recebe parâmetros para ajuste do ponto de visão do usuário como por exemplo, localização dos olhos.

PhysicalUniverse: descreve o ambiente do usuário. Esta descrição serve de suporte para a classe View. Descreve dispositivos de visualização (monitores, óculos), sensores de entrada, e dispositivos de áudio, entre outros.

Screen3D: descreve o suporte para o dispositivo de visualização. É sempre ligado a um Canvas 3D.

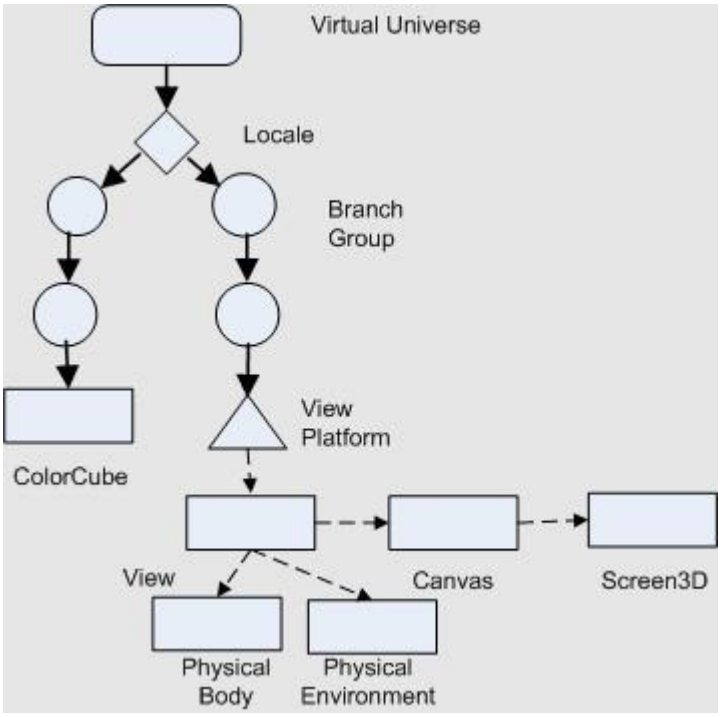


Figura 4.2 – Exemplo de grafo de cena.

Canvas3D: é derivada da classe *Canvas* do *Abstract Windowing Toolkit* (AWT), nativo do Java. Cria base para exibição do grafo de cena, e pode ser entendido como o plano de imagem.

ObjectGraphObject: tem duas subclasses: *Node* e *NodeComponent* que compõem o grafo de cena.

Node: é uma superclasse abstrata das classes *Group* e *Leaf*, representando um item do grafo de cena

Group: é uma superclasse usada na especificação da localização e orientação dos objetos visuais no universo virtual. Duas subclasses da classe *group* são *BranchGroup* e *Transform Group*, representados no grafo de cena por BG e TG, respectivamente.

Leaf: é uma superclasse usada para especificação da forma, som e comportamento (*behavior*) dos objetos visuais no universo virtual. Esses objetos não têm filhos.

NodeComponent: é uma superclasse usada para especificação de geometria, aparência, textura e propriedades do material de um nó *shape3D*, ou seja, um conjunto de atributos para os *nodes*. Os *NodeComponents* não são parte do grafo de cena, sendo somente referenciados.

Transform3D: os objetos dessa classe representam transformações de geometria 3D, como translação, rotação e escala. Geralmente estes objetos são usados em conjunto com um objeto *TransformGroup*.

Alpha: objetos desta classe são usados para criar uma função variante de tempo. Uma classe *Alpha* produz um valor entre zero e um, dependente do tempo e parâmetros do objeto. São comumente usados com objetos interpoladores, para fornecer animações de objetos visuais.

Uma vez definidos e apresentados os principais componentes, pode considerar que o desenvolvimento básico de um programa em Java 3D consiste nos seguintes passos (BOUVIER, 1999): a) criar um objeto *Canvas* 3D; b) criar um objeto *Virtual Universe*; c) criar um objeto *Locale* e relacioná-lo ao objeto *Virtual Universe*; d) construir um *View branch graph*; e) criar um objeto *View*; f) criar um objeto *ViewPlatform*; g) *c*riar um objeto *PhysicalBody*; h) criar um objeto *PhysicalEnvironment*; i) relacionar os objetos *ViewPlatform*, *PhysicalBody*, *PhysicalEnvironment* ao objeto *View*; j) cons-

truir o(s*) Content Branch Graph*(s); k) compilar o(s) *Branch Graph*(s); l) iInserir os subgrafos em um objeto *Locale*.

4.4. Geração de Conteúdo

Existe em Java 3D um rico conjunto de características, que permite a construção de conteúdos 3D complexos. Vale ressaltar que não existe um formato próprio para conteúdo em Java 3D. Existem, basicamente, duas formas para a geração desse conteúdo: usando *loaders* para carregar um conteúdo previamente construído e a construção do conteúdo dentro da própria aplicação Java, usando as classes Java 3D para criação de geometrias.

Para o caso do uso de *loaders*, são necessárias classes *File loader* que habilitam a leitura de conteúdos a partir de arquivos em outros formatos, tais como: VRML (*Virtual Reality Modeling Language*), OBJ (*Alias/Wavefront object*), LW3D (*Lightwave 3D scene*) e ainda pode-se construir um *File loader* próprio para a aplicação que se está desenvolvendo.

Já no caso da criação de conteúdo dentro da própria aplicação, pode-se usar um nó *leaf Shape3D* que constrói um objeto 3D baseado na geometria (*Geometr*y) que se refere à forma ou estrutura da forma e, na aparência (*Appearanc*e), que se refere à coloração, transparência e sombreamento da forma.

Existem 14 diferentes tipos de geometria, agrupados em: *Simple geometry* (PointArray , LineArray , TriangleArray e QuadArray); *Strip geometry* (LineStripArray , TriangleStripArray e TriangleFanArray); *Indexed simple geometry* (IndexedPointArray , IndexedLineArray , IndexedTriangleArray e IndexedQuadArray) ; e *Indexed stripped geometry* (IndexedLineStripArray , IndexedTriangleStripArray e IndexedTriangleFanArray).

4.5. Manipulação de Capabilities

O grafo de cena construído por um programa Java 3D poderia ser usado diretamente para renderização. Esta representação, entretanto, não é muito eficiente. A flexibilidade construída em cada objeto de um grafo de cena torna a representação do universo virtual sub-otimizada. Portanto uma representação mais eficiente é usada, para melhorar a performance de renderização.

Java 3D tem uma representação interna para um universo virtual e métodos para fazer a conversão. Há dois modos para que o sistema faça a conversão para a representação interna. Um deles é compilar cada grafo. O outro é inserir o grafo em um universo virtual para torná-lo vivo. A compilação de um *branch graph* otimiza-o para uma renderização mais rápida.

Essa conversão, ocasiona alguns efeitos. Um deles é fixar o valor das transformações e outros objetos no grafo. A menos que especificado, o programa não tem a permissão para alterar os valores dos objetos do grafo depois que foram tornados vivos.

Há casos quando um programa precisa alterar valores de um objeto, depois que este se torna vivo, como por exemplo, para alterar o valor de um objeto *TransformGroup*, quando tem-se animações. Para isso acontecer, a transformação deve ser capaz de mudar, depois que o objeto esteja vivo.

Para tanto, cada *SceneGraphObject* tem um conjunto de bits de permissão (*capabilities*). Os valores desses bits determinam quais as permissões existem para o objeto depois que este é compilado ou se torna vivo. O conjunto de permissões varia por classe.

Tais permissões controlam o acesso aos atributos de um nó após já ter se tornado vivo ou sido compilado. É interessante observar que quanto menor a quantidade de permissões, mais otimizações poderão ser feitas.

Vale ressaltar ainda, que cada nó tem suas próprias permissões para leitura e escrita, que podem ser herdadas de sua classe pai

4.6. Interações

Em Java 3D, interações e animações são determinadas com objetos da classe *Behavior*. A classe *Behavior* é uma classe abstrata, que fornece mecanismos para mudar o grafo de cena.

O propósito de um objeto *Behavior* em um grafo de cena é alterá-lo, ou alterar seus objetos, em resposta a alguns estímulos. Um estímulo pode ser o pressionar de uma tecla, um movimento de mouse, a colisão de objetos, a passagem do tempo, algum outro evento, ou uma combinação de todos esses, entre outros. As alterações produzidas podem ser: adição ou remoção de objetos ao grafo de cena, mudança dos atributos desses objetos e rearranjo deles. Estas possibilidades são limitadas apenas pelas permissões dos objetos do grafo.

A Tabela 4.1 abrange algumas das possibilidades de *Behavior* (um estímulo resulta em uma mudança).

Por meio de *behaviors*, Java 3D fornece suporte aos dispositivos convencionais (*mouse,* teclado) e não convencionais (*joysticks,* luvas). Java 3D provê acesso a teclado e *mouse* usando a API padrão para suporte a esses dispositivos por meio dos *behaviors KeyNavigatorBehavior* e *MouseBehavior*, respectivamente.

Por meio de *behaviors*, Java 3D fornece suporte aos dispositivos convencionais (*mouse,* teclado) e não convencionais (*joysticks,* luvas). Java 3D provê acesso a teclado e *mouse* usando a API padrão para suporte a esses dispositivos por meio dos *behaviors KeyNavigatorBehavior* e *MouseBehavior*, respectivamente.

	Objeto de mudança

Estímulo (razão de mudança)	TransformGroup (objetos visuais mudam de orientação ou local)	Geometria (objetos visuais mudam de forma ou cor)	Grafo de Cena (adicionar ou remover objetos)	Vista (mudança de lugar ou direção da vista)
Usuário	Interação	Específico da aplicação	Específico da aplicação	Navegação
Colisões	Objetos visuais mudam de orientação ou lugar	Objetos visuais mudam aparência quando colidem	Objetos visuais desaparecem com a coilsão	Vista muda com a colisão
Tempo	Animação	Animação	Animação	Animação
Lugar de visão	"Billboard"	Nível de detalhe (LOD)	Específico da aplicação	Específico da aplicação

Tabela 4.1 – Algumas possibilidades de Behavior

Adicionalmente, Java 3D provê acesso a uma variedade de dispositivos de entrada, como *trackers* e *joysticks*. Os dispositivos

possuem valores de entrada contínuos e bem definidos. *Joysticks*, por exemplo, produzem dois valores contínuos entre –1.0 e 1.0 que Java 3D armazena internamente como uma matriz de transformação, com um dos valores como sendo a translação do eixo X e outro como sendo a translação do eixo Y.

Uma classe *behavior* pode ser ainda personalizada. Para tanto é necessário, no mínimo, a implementação dos métodos *initialization* e *processStimulus*, da classe abstrata *Behavior*. Essa classe personalizada, tem um construtor e pode ter outros métodos.

A classe *behavior* precisa de uma referência a seu objeto de mudança para poder executar as mudanças. O construtor pode ser usado para isso. Caso contrário, é necessário que outro método guarde essa informação.

O método *initialization* é executado quando o grafo de cena contendo a classe *behavior* se torna vivo. Ele é responsável por ajustar o evento inicial e a condição inicial das variáveis que fazem parte da classe.

O método *processStimulus,* por sua vez, é invocado quando o evento de disparo especificado para o *behavior* ocorre. Ele é o responsável a responder ao evento de disparo.

Outro método que pode ser implementado é o *WakeupOn*. Ele é usado nos métodos *initialize* e *processStimulus* para ativar o disparo para o *behavior*.

Vale ressaltar que os mecanismos de implementação de uma classe *Behavior* são simples. Entretanto, deve-se saber que uma classe implementada de forma errada pode degradar a performance do sistema. Dois aspectos devem ser evitados: consumo demasiado de memória e condições de disparo desnecessárias.

4.7. Picking

É comum em ambientes virtuais, existir mais de um objeto visual presente. Para podermos manipular esses objetos individualmente, utiliza-se de uma técnica chamada *Picking*.

A classe *Picking* é implementada por um *behavior* tipicamente disparado por eventos causados pelos botões do mouse. Neste caso, o usuário posiciona o indicador do mouse em cima do objeto visual de sua escolha e pressiona um botão, disparando a "operação" *picking*.

Essa operação consiste no seguinte: um raio é projetado no mundo virtual com origem na posição do ponteiro do mouse. A intersecção desse raio com os objetos do mundo virtual é computada. O objeto visual mais perto do plano de imagem que faz intersecção com o raio será o escolhido para interação.

O teste de intersecção requer uma grande demanda de processamento. A técnica de *picking* consome grande quantidade de recursos, e depende da complexidade da cena.

4.8. Suporte para Dispositivos

As principais formas de interação com o AV são o mouse e a barra de movimentação. Por meio delas o participante pode atuar na molécula em seis graus de liberdade (6DOF - *Degrees Of Freedom*), efetuando rotações e translações em qualquer sentido dos eixos X, Y e Z.

A finalidade da barra de movimentação é mudar o ponto de vista do participante em relação ao AV como um todo, enquanto o mouse muda o ponto de vista do participante em relação ao objeto.

Além do mouse e da barra de movimentação, existe também a possibilidade do usuário utilizar *joystick* ou luvas como dispositivos de entrada. Para tanto, é necessária a implementação das clas-

ses *joystick* e *gloves* com o respectivo suporte para o dispositivo utilizado [RODELLO, 2003].

4.9. Exemplo

Para os mundos mais simples, e que não requerem o tratamento com os valores de coordenadas, existe a possibilidade de utilização da classe *SimpleUniverse* [PALMER, 2001]. Ela permite uma construção mais simples e rápida com a fusão de alguns nós a serem criados no caso do *VirtualUniverse*. Na Figura 4.3 a classe *ConeTest*, e o resultado da sua interpretação na Figura 4.4.

```
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.geometry.Cone;
import javax.media.j3d.*;
import javax.vecmath.*;
public class ConeTest {
public ConeTest() {
   SimpleUniverse universe = new SimpleUniverse();
   BranchGroup group = new BranchGroup();
   group.addChild(createLight(0.0f, 0.0f, -1.0f));
   group.addChild(createLight(0.3f, -0.3f, -0.3f));
   group.addChild(new Cone( 0.3f, 0.9f, createAppearance()));
   universe.getViewingPlatform().setNominalViewingTransform();
   universe.addBranchGraph(group);
}
private Light createLight(float x, float y, float z) {
    DirectionalLight light =
      new DirectionalLight( true,
                        new Color3f(1.0f, 1.0f, 1.0f),
                        new Vector3f(x, y, z));
    light.setInfluencingBounds(new      BoundingSphere(new      Point3d(),
100.0));

    return light;
}
private Appearance createAppearance() {
    Appearance app = new Appearance();
    Material mat = new Material();
    mat.setDiffuseColor(0.5f, 1.0f, 0.5f);
    app.setMaterial(mat);
    return app;
  }
public static void main( String[] args ) {
   new ConeTest();
}
                  }
```

Figura 4.3 – Exemplo a Classe ConeTest

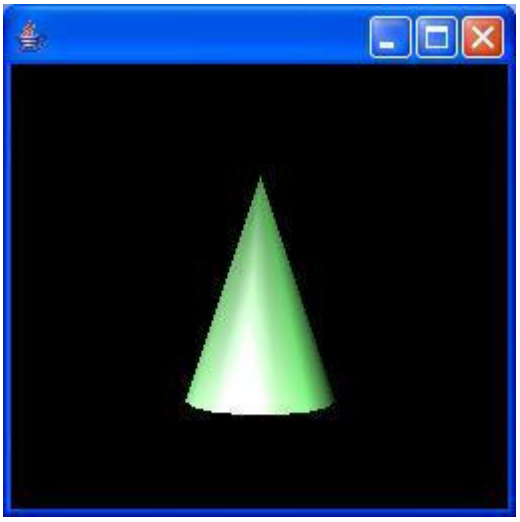


Figura 4.4 – O cone gerado

Referências

Bouvier, J. D. (2002), "Getting Started with the Java 3D™ API - A Tutorial for Beginners", http://java.sun.com/products/java-media/3D/collateral/index.html.

Palmer, I. (2001) "Essential Java 3D Fast: Developing 3D Graphics Applications in Java", Springer.

Rodello, I. A.; Brega, J. R. F.; Sementille, A. C. (2003) "Interação com dispositivos de entrada não convencionais em ambientes virtuais desenvolvidos com Java 3D", Proccedings SVR 2003 – VI Symposium on Virtual Reality. 15-18 October. Ribeirão Preto, SP – Brasil.

Selman, D. (2002) "Java 3D Programming", Manning Publications Co..

Sun (2006) Sun Microsystems Inc. Disponível em <http://www.sun.com.br> . Acesso em 13 de Maio de 2006.

Capítulo 5

ARToolKit: Biblioteca para Desenvolvimento de Aplicações de Realidade Aumentada

Rafael Santin e Claudio Kirner

Curso de Mestrado em Ciência da Computação
Universidade Metodista de Piracicaba - UNIMEP
Rodovia do Açúcar, Km 156, 13400-911 Piracicaba – SP, Brasil

{rafael, ckirner}@unimep.br

Abstract

ARToolKit is a free and open library used in the development of augmented reality applications. The augmented environment is obtained by mixing video with registered virtual objects, showed on head mounted display or monitor. This chapter presents the concepts, structure and behavior of ARToolKit, as well as configuration examples and programming of functions.

Resumo

ARToolKit é uma biblioteca gratuita e aberta usada para o desenvolvimento de aplicações de realidade aumentada. O ambiente aumentado é obtido pela mistura de vídeo com objetos virtuais, sendo

mostrado em capacete ou monitor. Este capítulo apresenta conceitos, estrutura e funcionamento do ARToolKit e exemplos de configuração e programação de funcionalidades.

5.1. Realidade Aumentada e ARToolKit

A Realidade Aumentada (RA) consiste numa interface avançada de computador, que promove em tempo real a exibição de elementos virtuais sobre a visualização de determinadas cenas do mundo real, oferecendo um forte potencial a aplicações industriais e educacionais, devido o alto grau de interatividade [Kirner e Tori 2004]. Para desenvolver aplicações de RA, são utilizadas bibliotecas computacionais que implementam a captura de vídeo, técnicas de rastreamento, interação em tempo real e os ajustes visuais das cenas do mundo real e virtual.

O ARToolKit é uma biblioteca de programação multiplataforma considerada um kit de ferramenta de RA, bastante utilizada e discutida por desenvolvedores e pesquisadores da comunidade de Realidade Aumentada. O ARToolKit possui o seu código livre para modificações e uso no desenvolvimento de aplicações não comerciais sob licença GPL [GNU 2007], enquanto que a versão proprietária para a comercialização é oferecida pela incorporação ARToolworks [ARToolworks 2007].

5.1.1. Fundamentos do ARToolKit

A biblioteca ARToolKit implementada em C e C++ oferece suporte a programadores para o desenvolvimento de aplicações de RA. Essa biblioteca utiliza o rastreamento óptico, que implementa técnicas de visão computacional para identificar e estimar em tempo real a posição e a orientação de um marcador (moldura quadrada desenhada eu um papel) em relação ao dispositivo de captura de vídeo. Assim, o cálculo da correlação entre os dados estimados do

marcador real e a sua imagem, possibilita posicionar objetos virtuais alinhados à imagem do marcador [Kato et al 2000].

A saída de aplicações desenvolvidas com ARTooKit pode ser visualizada através de dispositivos de visão indireta, não imersivos, ou visão direta, imersivos. A diferença de imersão entre as duas formas de visão, está relacionada à autenticidade impelida ao sentido visual do usuário. Os dispositivos de visão indireta, como os monitores, não imersivos, promovem a visualização conjugada das cenas virtual e real fora do espaço alvo de atuação. Já a visualização direta, imersiva, pode ser obtida por meio dos dispositivos como o HMD (Head Mounted Display), que fornecem a visualização combinada das cenas diretamente no espaço alvo de atuação da aplicação [Kirner e Tori 2006].

Os objetos virtuais visualizados em aplicações desenvolvidas com as distribuições do ARToolKit podem ser implementados com OPENGL e/ou com VRML. A visualização desses objetos virtuais é realizada no momento da inserção de seus respectivos marcadores no campo de captura da câmera de vídeo.

5.1.1.1. Marcadores

O rastreamento implementado no ARToolKit estima a pose de marcadores, tornando possível desenvolver aplicações que necessitem conhecer a posição e orientação de elementos ou ações reais, que são representados na cena por marcadores. Como exemplo, tem-se as aplicações de RA, que utilizam o marcador para posicionar e orientar elementos virtuais na cena do mundo real, tornando um meio de interação do usuário com essas aplicações.

Os marcadores reconhecidos pelo ARToolKit consistem em figuras geométricas quadradas, que contém no seu interior símbolos para identificá-los. A figura 6.1 mostra um exemplo de marcador, com símbolos para a sua identificação.

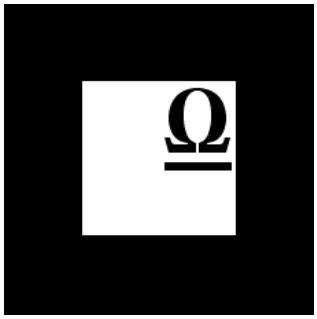
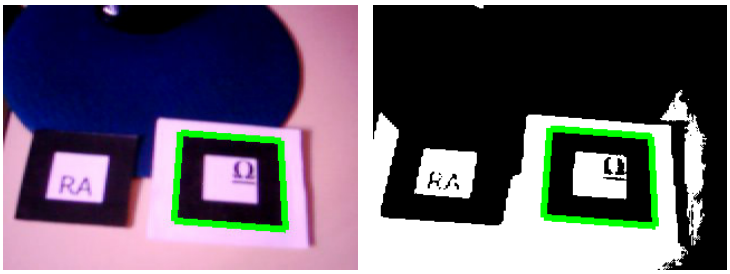


Figura 5.1. Exemplo de marcador.

Como o ARToolKit extrai da imagem de vídeo limiarizada (em preto e branco) as bordas do quadrado em preto, utiliza-se uma moldura em branco antecedendo esse quadrado para promover o contraste no próprio marcador, viabilizando o seu reconhecimento sobre superfícies de cores escuras. A Figura 6.2a demonstra dois marcadores dispostos sobre uma superfície escura. A diferença entre os marcadores consiste no fato do marcador com o símbolo RA não possuir a moldura em branco e outro conter essa moldura. O marcador com o símbolo RA não pode ser identificado sobre a superfície escura, pois não é possível extrair as suas bordas na imagem limiarizada (Figura 6.2b).



(a) (b)

Figura 5.2 Utilidade da moldura branca do marcador

O reconhecimento de padrões identifica os quatro vértices de regiões quadradas, contidas na imagem de vídeo, e compara os

símbolos do seu interior com os gabaritos dos marcadores cadastrados pelo usuário [Claus Fitzgibbon 2005]. Caso o retângulo extraído seja semelhante com algum marcador cadastrado, o sistema passa a calcular a sua orientação e posição.

5.1.1.2. Rastreamento

O rastreamento no ARToolKit é responsável pelo processamento da imagem, que extrai algumas informações com relação a detecção, e pela identificação de características dos marcadores, além de estimar sua posição e orientação. Nesse caso, a obtenção da posição e orientação do marcador é realizada através da análise da imagem de vídeo, que estabelece o relacionamento entre as coordenadas do marcador e as coordenadas da câmera, como demonstrado na figura 6.3 [Kato et al 1999].

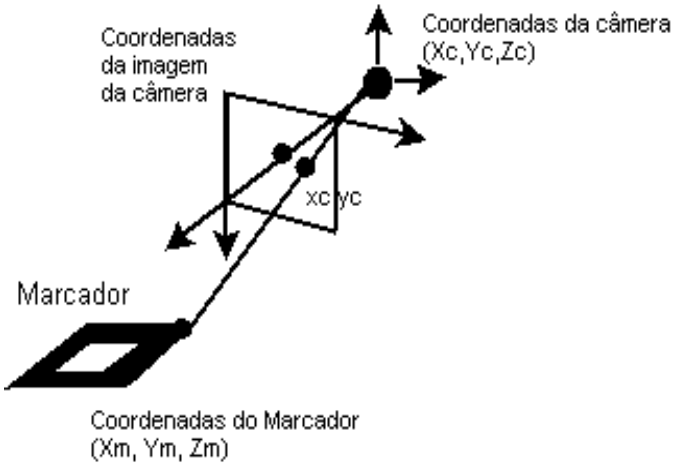


Figura 5.3. Relacionamento entre os sistemas de coordenadas do marcador e da câmera.

O relacionamento entre as coordenadas do marcador e as coordenadas da câmera é realizado por intermédio de uma matriz 3x4, denominada "matriz transformação". A Figura 6.4 mostra a multiplicação de uma matriz transformação "T" por um ponto 3D

no marcador (*Xm,Ym,Zm*), obtendo o ponto correspondente no sistema de coordenadas da câmera (*Xc,Yc,Zc*)

$$\begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Xm \\ Ym \\ Zm \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} Xm \\ Ym \\ Zm \\ 1 \end{bmatrix}$$

Figura 5.4. Multiplicação de matrizes

Para estimar a posição e orientação do marcador, através da análise da imagem de câmera, torna-se necessário utilizar os parâmetros de câmera, a fim de corrigir as distorções inerentes a câmera [Lepetit Fua 2005]. Assim, é possível estimar, com certa precisão, o relacionamento entre as coordenadas 3D do mundo e as coordenadas 2D da imagem [Abdullah e Martinez 2002]. O ARToolKit utiliza esses parâmetros de câmera no cálculo da matriz transformação [Kato Billinghurst 1999].

As funções, responsáveis pelo cálculo da matriz transformação no ARToolKit, são "arGetTransMat" e "arGetTransMatCont". A função "arGetTransMat" é utilizada no momento em que o marcador é detectado, enquanto a "arGetTransMatCont" deverá ser chamada posteriormente, caso o marcador permaneça visível nos quadros de vídeos subseqüentes, o que possibilita o uso de informações obtidas anteriormente para agilizar o cálculo da matriz [Kato 2002].

Estimada a matriz transformação, a API "OPENGL" é utilizada para ajustar a câmera virtual, posicionar e desenhar o objeto virtual alinhado na visualização do marcador real [Piekarski Thomas 2002].

5.1.1.3. Funcionamento do ARToolKit

A biblioteca de programação ARToolKit disponibiliza um conjunto de funções que oferecem suporte ao desenvolvimento de aplicações de Realidade Aumentada. Para implementar uma aplicação simples de RA, o programador necessita conhecer as funcionalidades de algumas funções dessa biblioteca e seguir os seguintes passos:

1. Iniciar a configuração do vídeo; ler o arquivo de cadastramento dos marcadores; ler os parâmetros da câmera.

2. Capturar um quadro do vídeo.

3. Detectar e identificar os marcadores

4. Calcular a transformação do marcador relativa à câmera.

5. Desenhar o objeto virtual referente ao marcador.

6. Encerrar a captura de vídeo.

Os passos de número 2 a 5 são repetidos continuamente até a aplicação ser finalizada, enquanto os passos 1 e 6 fazem respectivamente a inicialização e o término da aplicação [Kato et al 2000].

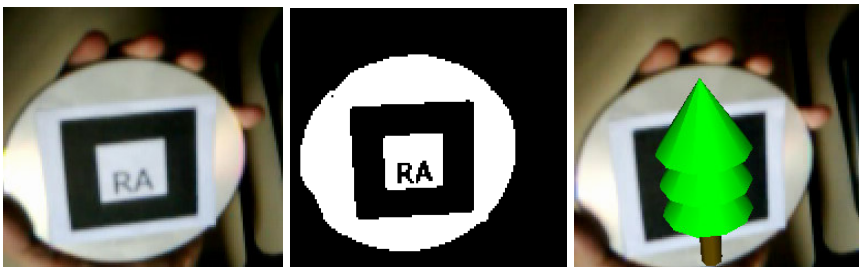
O ARToolKit é distribuído com aplicações exemplos, que implementam os passos citados anteriormente, servindo de modelo aos programadores, tanto para o conhecimento de funções da biblioteca, quanto para o auxílio no desenvolvimento de novas aplicações de RA. A tabela 5.1 mostra os passos e as respectivas funções executadas na aplicação "simple" disponibilizada nas distribuições do ARToolKit.

Passos	Função
1. Inicializa a aplicação	Init
2.Captura do quadro de vídeo.	ArVideoGetImage

3. Detecta os marcadores	ArDetectMarker
4.Calcula a transformação da câmera	ArGetTransMat
5. Desenha o objeto virtual.	Draw
6. Fecha a captura de vídeo	Cleanup

Tabela 5.1 Passos e as funções implementadas num exemplo de aplicação distribuída com o ARToolKit .

O funcionamento de aplicações de RA, como alguns dos exemplos inclusos na distribuição do ARToolKit, executa várias etapas relacionadas as ilustrações da Figura 5.5. Na primeiro etapa, a imagem de vídeo capturada, conforme a figura 5.5a, é convertida em uma imagem binária (preta e branca), de acordo com o valor do limiar ou *threshold*, resultando numa imagem como a da figura 5.5b. Por conseguinte, as regiões quadradas, nessa imagem binária, são detectadas e comparadas com gabaritos de marcadores cadastrados no sistema pelo usuário. Caso haja a identidade entre supostos marcadores e os marcadores conhecidos pelo sistema, a aplicação considera que encontrou um marcador na imagem. A próxima etapa, então, consiste na obtenção da posição e orientação de marcadores [Kato et al 2000]. Assim, é possível desenhar o objeto virtual sobreposto a seu respectivo marcador, como mostra a figura 5.5c.



(a) (b) (c)

Figura 5.5 Etapas envolvidas no funcionamento de uma aplicação de RA : a) imagem da cena com o marcador, b) a

imagem limiarizada e c) objeto virtual sobrepondo o marcador.

A Figura 5.6 mostra um diagrama, detalhando as principais etapas realizadas no funcionamento da aplicação.

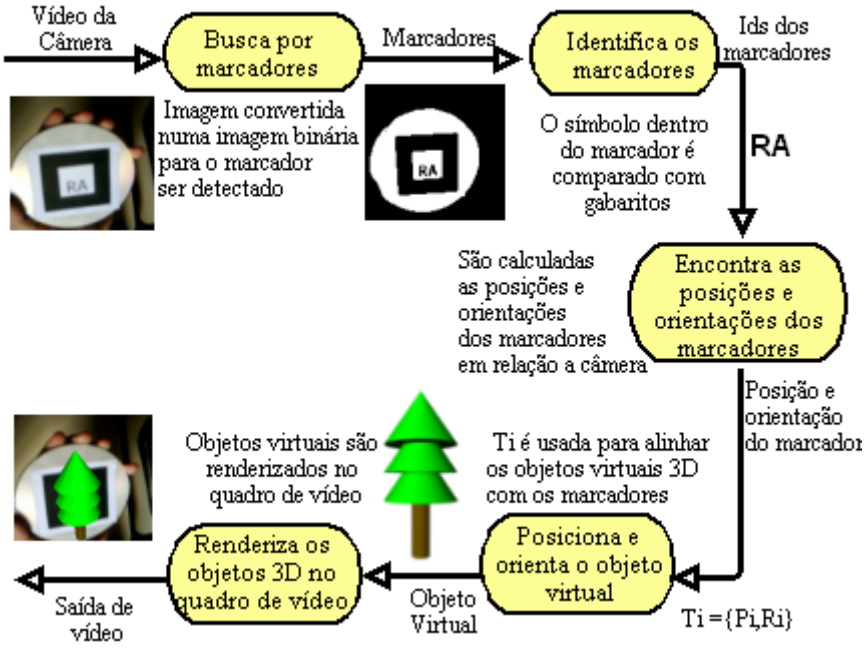


Figura 5.6. Funcionamento de uma aplicação de RA

5.2. Instalação e configuração do ARToolKit

O site do ARToolKit no laboratório HITL da universidade de Washington [ARToolKit 2007], disponibiliza o ARToolKit, a sua documentação, projetos e artigos relacionados a biblioteca. A área download desse site, além de manter os links de versões mais antigas e algumas outras contribuições, também, disponibiliza um link para o site, onde estão disponíveis as versões mais recentes.

5.2.1. Instalação

Para a instalação do ARToolKit deve-se, primeiramente, baixar a versão desejada. O próximo passo é descompactar o arquivo no local de conveniência. Esse local será referenciado abaixo como {ARToolKit} A Figura 5.7 mostra a estrutura de diretórios da versão 2.72.1 do ARToolKit, após descompactação.

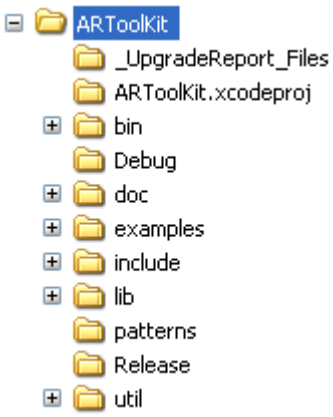


Figura 5.7. Estrutura do diretório da versão 2.72.1 do ARToolKit após a instalação.

5.2.2. Configurações na plataforma Windows

Realizada a descompactação, o próximo passo consiste em configurar o sistema, instalando os pré-requisitos exigidos para a compilação do ARToolKit. Os pré-requisitos necessários para a compilação no Windows, Linux/SGI Irix e Mac OS X são descritos no arquivo README, contido na pasta "ARToolKit".

Para compilar as bibliotecas e aplicações da versão 2.72.1 do ARToolKit no Windows, por exemplo, são pré-requisitos:

- Microsoft Visual Studio .NET 2003 ou Visual Studio 6, ou Cygwin.

- DSVideoLib-0.0.8b-win32
- GLUT
- OpenVRML-0.16.1-bin-win32 ou OpenVRML-0.14.3-win32 (opcional)

No caso dessa versão 2.72.1 na plataforma Windows, é necessário seguir os seguintes passos para compilar os seus projetos:

1. Instalar um dos compiladores da Microsoft (Visual Studio .NET 2003 ou Visual Studio 6) ou o CygWin.

2. Baixar e descompactar a biblioteca "DSVideoLib-0.0.8b-win32" [DSVideoLib 2007] dentro do diretório raiz "{ARToolKit}", certificando-se que o diretório seja denominado "DSVL". Copiar os arquivos "DSVL.dll" e "DSVLd.dll" para a pasta "bin" do ARToolKit ("{ARToolKit}\bin") .

3. Baixar o GLUT [GLUT 2007], descompactar e copiar o arquivo "glut32.dll" para pasta "system" do Windows, copiar o arquivo "glut32.lib" para a pasta "lib" do ARToolkit ("{ARToolKit}\lib") e criar uma pasta "GL" no diretório "include" do ARToolKit ("{ARToolKit}\include \GL"), copiando para essa nova pasta o arquivo "glut.h".

4. Executar o script "Configure.win32.bat" que está em "{ARToolKit}\Configure.win32.bat" para criar o arquivo "{ARToolKit}\include\AR\config.h".

5. Abrir o arquivo "ARToolKit.sln" no VisualStudio .NET ou o arquivo "ARToolkit.dsw" no Visual Studio 6.

6. Compilar os projetos.

7. Executar as aplicações geradas em "{ARToolKit}\bin"

   Caso o usuário deseje compilar a aplicação "Simple VRML", que renderiza objetos virtuais implementados em VRML, deve-se :

8. Baixar a biblioteca "OpenVRML" (OpenVRML-0.16.1-bin-win32 ou OpenVRML-0.14.3-win32 [OpenVRML 2007]) e descompactá-la no diretório raiz do ARToolKit.

9. Copiar o arquivo "js32.dll" da pasta "{ARToolKit}\OpenVRML\bin" para a pasta "{ARToolKit}\bin".

10. Habilitar e compilar, no VisualStudio .NET, os projetos "libARvrml" e "simpleVRML". Já, no VisualStudio 6, é necessário criar um projeto para compilar a biblioteca estática gerada pelo "libARvrml" e um projeto de aplicação win32, com as devidas dependências ajustadas para gerar o executável "simpleVRML.exe".

No momento de execução das aplicações, podem surgir mensagens de erro, acusando a falta de algumas dlls da Microsoft. Nesse caso, é necessário baixá-las de sites na Internet e copiá-las para a pasta "system32", existente no diretório Windows.

5.3. Aplicações inclusas no ARToolKit

O ARToolKit é distribuído com diversas aplicações. A versão 2.72.1, por exemplo, após a compilação disponibiliza várias aplicações executáveis na pasta "bin". Essas aplicações possuem diferentes funcionalidades, permitindo que os usuários as utilizem tanto na configuração, quanto no auxílio no desenvolvimento de

suas próprias aplicações. No total são vinte e duas aplicações, das quais seis são denominadas aplicações utilitárias e as demais são exemplos de aplicações de RA.

5.3.1. Aplicações Utilitárias

As aplicações utilitárias são responsáveis pela configuração e teste do sistema. Os utilitários de configuração são o "mkpatt, calib_camera2, calib_cparam e calib_distortion". Já os utilitários de teste do sistema são o "graphicTest" e "videoTest". Os códigos desses programas estão em "{ARToolKit}\util".

O "mkpatt" é um programa usado na geração dos arquivos bitmaps (mapa de bits), que relacionam os marcadores aos objetos virtuais. Cada arquivo bitmap contém um conjunto de exemplo de imagens do marcador. Esse conjunto é conhecido, como treinamento do marcador. Uma vez executado o "mkpatt", será pedido para entrar com o arquivo contendo os parâmetros de câmera. Escreva o caminho para o arquivo, caso esse seja diferentes da configuração default ("Data/câmera_para"), senão basta teclar "enter". A Figura 6.8 demonstra o início do programa.

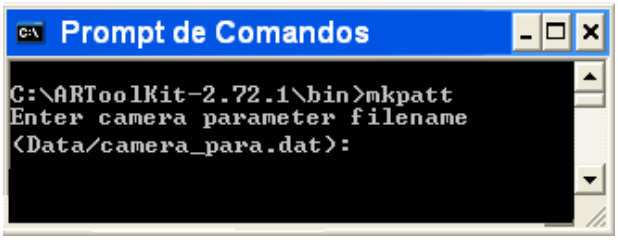


Figura 5.8. Início da execução do mkpatt

O programa exibirá uma janela com a imagem do vídeo. Enquadre o marcador nessa imagem, de modo a aparecer um retângulo com lados vermelhos, à esquerda e acima, e verdes, à direita e abaixo, nas bordas do marcador, como mostra Figura 5.9.

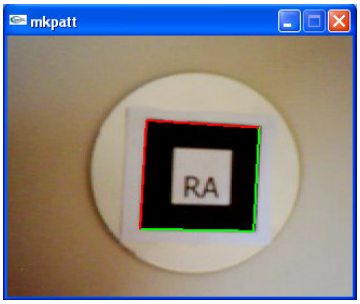


Figura 5.9. Marcador identificado pelo "mkpatt"

Pressione o botão esquerdo no mouse e será pedida a entrada de um nome para o arquivo. Realizado esse passo, um arquivo será criado na pasta "bin". Como os arquivos, com dados de configurações das aplicações, geralmente estão localizados na pasta "Data", deve-se transferir o arquivo gerado para esta pasta.

Os programas "calib_camera2, calib_cparam e calib_distortion" são responsáveis pela calibração de câmera. As etapas relacionadas à calibração de câmera são detalhadas em [Consularo et al 2004].

O utilitário "graphicTest" exibe uma janela com um objeto 3D desenhado em seu interior, enquanto o "videoTest" mostra a uma janela com a imagem capturada pela câmera. Os resultados dos testes que esses programas realizam são as próprias execuções. Caso ocorra algum erro durante a execução de um desses utilitários, provavelmente ocorrerá o mesmo erro na execução dos outros programas fornecidos pela biblioteca.

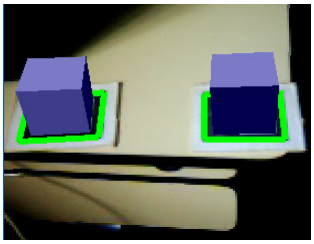
5.3.2. Exemplo de aplicações de RA

As aplicações de RA fornecidas junto ao ARToolKit são exemplos que não só viabilizam o entendimento no funcionamento das funções do ARToolKit, mas também servem como modelo para a produção de novas aplicações.

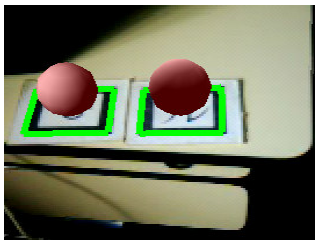
Os exemplos executáveis contidos na pasta "bin" são: "collide, exview, loadmultiple, modetest, multi, optical, paddle, paddledemo, paddleinteraction, range, relation, simple, simple2, simplelite, simplevrml e twoview". Cada exemplo disponibiliza diferentes funções relacionadas à interação da aplicação, servindo como base aos usuários que desejem implementar novas funcionalidades em suas aplicações. Os códigos desses programas estão localizado em "{ARToolKit}\examples". A próxima seção demonstrará algumas funcionalidades desses exemplos.

5.3.2.1. Teste de Colisão

A aplicação "collideTest" possui como característica principal a verificação de colisão entre dois marcadores. A função responsável em verificar a colisão é denominada "checkCollisions". Essa função recebe as estruturas relacionadas às informações de cada um dos dois marcadores, além e um número representante do fator de colisão. O retorno dessa função é um número inteiro "1", caso os marcadores esteja em colisão e "0", caso contrário. A Figura 6.10 mostra a aplicação em execução, quando a distância entre os marcadores supera o limite do fator de colisão, sendo desenhado um cubo sobre os marcadores, conforme a Figura 5.10a. Quando a distância entre os marcadores for inferior ao limite, será desenhada uma esfera sobre os marcadores, conforme a Figura 5.10b.



(a)



(b)

Figura 5.10. Execução da aplicação "collideTest": a) cubos desenhados sobre os marcadores distantes e b) esferas desenhadas sobre marcadores próximos.

O exemplo "collide" utiliza os marcadores cadastrados no arquivo "object_data2" contido em "{ARToolKit}\bin\Data". Dessa forma, o usuário pode associar os seus próprios marcadores, utilizando para isso o "mk_patt" na geração do arquivo bitmaps (mapa de bits) dos marcadores, os quais deverão substituir os marcadores configurados no "object_data2".

5.3.2.2. Uso da Pá

O "paddleDemo" (uso da pá) é um dos exemplos que insere um marcador com funcionalidades especializadas a técnicas tangíveis. Esse marcador será denominado pá. A pá é utilizada para interagir com objetos virtuais atrelados a um cartão contendo vários marcadores, denominado de cartão base. Essa aplicação disponibiliza técnicas que permitem identificar a inclinação da pá em relação ao cartão base e, assim, atribuir algumas ações, como no caso despejar uma esfera no cartão base ou a pegá-la. As funções que implementam essas técnicas são a "check_incline" e a "check_pickup". Essas funções encontram-se no arquivo "command_sub.c" em "{ARToolKit}\exemple\paddleDemo". A Figura 6.11 mostra a sequência de execução do "paddleDemo".

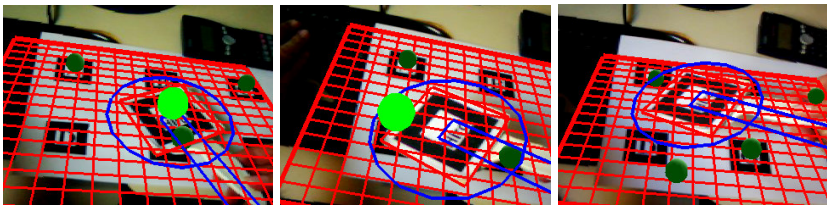


Figura 5.11. Execução do exemplo "paddleDemo", mostrando a esfera sobre a pá, a inclinação para despejar a esfera na base e a esfera fixada na base.

O arquivo que relaciona a pá na aplicação é o "paddle_data", contido na pasta data em "{ARToolKit}\bin\Data". Já o cartão base configurado pelo arquivo "marker.dat" está contido na pasta "multi" localizada em "{ARToolKit}\bin\Data\multi".

5.3.2.3. SimpleVRML

O "simpleVRML" é um programa que possibilita visualizar objetos virtuais, escritos na linguagem VRML, sobrepostos aos marcadores. Esse exemplo utiliza a biblioteca OpenVRML para a renderização dos objetos VRML. A figura 6.12 exibe o resultado da execução do "simpleVRML".

Figura 5.12. Resultado da execução do simpleVRML

Essa aplicação utiliza o arquivo de configuração "object_data_vrml" para atrelar um marcador a um objeto virtual. Localizado em "{ARToolKit}\bin\Data", esse arquivo armazena o relacionamento entre o arquivo bitmap do marcador, disposto no diretório "Data", e o arquivo de referência ao objeto VRML, localizado em "{ARToolKit}\bin\wrl". A Figura 5.13 mostra o arquivo "object_data_vrml", que contém dois marcadores cadastrados: o "patt.hiro" e o "patt.kanji". O marcador "patt.hiro" está associado a um arquivo de referência VRML, denominado "bud_B.dat", enquanto o marcador "patt.kanji" encontra-se associado ao arquivo "snoman.dat".

Em virtude dessa associação de arquivos, é possível adicionar objetos virtuais de maneira muito simples, bastando editar o

arquivo "object_data_vrml". Os demais exemplos disponibilizados no ARToolKit possuem os objetos virtuais implementados diretamente em seu código, de forma que, para adicionar novos elementos virtuais, é necessário incluir o código do objeto virtual em openGL na aplicação e a compilar novamente.

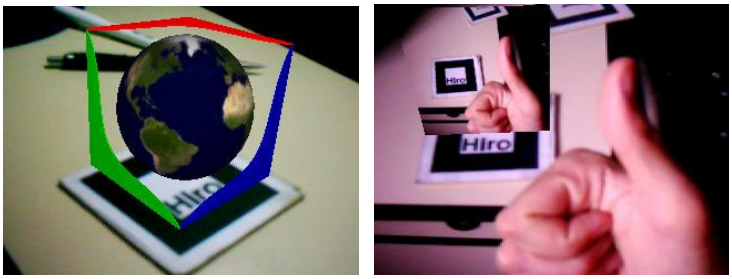
```
#the number of patterns to be recognized
2

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

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

Figura 5.13. Referência de marcadores e objetos virtuais VRML

5.4. Softwares baseados no ARToolKit.

Existem Softwares de Realidade Aumentada que estendem algumas técnicas de rastreamento do ARToolKit a diferentes abordagens funcionais e estruturais, no que tange os aspectos da diversidade computacional. Exemplos desses softwares são: O ARToolKitPlus considerado uma biblioteca de RA otimizada para uso em dispositivos portáteis como PDAs e alguns telefones celulares [Wagner e Schmalstieg 2003]; o jARToolKit que possibilita escrever aplicações de RA em java, acessando funções da biblioteca ARToolKit através da interface JNI [GEIGER et al 2002]; o ARToolKit Phyton, um "bind" Python que encapsula as funções do ARToolKit, permitindo a exploração das vantagens do Python nas implementações, que podem ser feitas sem a compilação de código

[Kirner 2007]; osgART framework baseado no ARToolKit que implementa a biblioteca gráfica OpenSceneGraph, apresentando alta qualidade na renderização de objetos virtuais e possibilidade de importar e exportar arquivos 3D gerados pelo 3D Studio Max e Maya [Looser et al 2006]. A Figura 5.14 mostra a execução de exemplos do osgART.



(a) (b)

Figura 5.14. Resultado da execução de exemplos do osgART: a) Exemplo osgARTsimpleNPR e b) Exemplo osgARTvideoPlane.

5.5. Conclusões

O desenvolvimento de novas formas de interações em sistemas computacionais busca fornecer mecanismos adaptativos dinâmicos às necessidades dos usuários, aproximando as técnicas de manipulação do computador à intuição sensorial humana [Kirner e Tori 2006]. Nesse contexto, está a Realidade Aumentada que promove a visualização e a manipulação de objetos modelados por computador no mundo real. Assim, é possível desenvolver aplicações altamente interativas e estimulantes associadas ao sentido visual do usuário.

O ARToolKit é uma biblioteca de desenvolvimento de aplicações de Realidade Aumentada, bastante popular na comunidade de RA. Isto acontece pelo fato da biblioteca fornecer soluções de rastreamento 3D, em tempo real, com baixo custo computacional

[Lepetit Fua 2005]. Além disso, o ARToolKit é amplamente utilizado por ser distribuído livremente para fins não comerciais, incentivando a liberdade para os usuários executarem, estudarem e modificarem os códigos disponíveis na biblioteca de acordo com as suas necessidades.

Este capítulo apresentou uma abordagem sobre o ARToolKit, ressaltando os aspectos relacionados a seu funcionamento interno e detalhes para a sua instalação e configuração. Foram discutidos também, alguns exemplos inclusos nas versões mais recentes, que não só contribuem com o conhecimento prático de funções do ARToolKit, mas também, como modelo de partida para os usuários desenvolverem as suas próprias aplicações. Para finalizar, foram apresentados outros Softwares de RA, incluindo variações da biblioteca ARToolKit.

Referências

Abdullah, J.; Martinez, K. (2002) "Camera Self-Calibration for the ARToolki"t. In Proceedings of First International Augmented Reality Toolkit Workshop, pp. 84-88, Darmstadt, Germany.

ARToolKit. Human Interface Technology Laboratory < http://www.hitl.washington.edu/artoolkit/> acesso em fev, 2007.

ARToolworks, Inc.< http://www.artoolworks.com/>acesso em fev, 2007.

Claus, David.; Fitzgibbon, A. W. "Reliable Automatic Calibration of a Marker-Based Position Tracking System," *wacv-motion*, pp. 300-305, Seventh IEEE Workshops on Application of Computer Vision (WACV/MOTION'05) - Volume 1, 2005.

Consularo, L.A.; Calonego Jr, N.; Dainese, C.A.; Garbin, T. R.; Kirner, C.;Trindade, J.; Fiolhais, C.(2004) "ARToolKit: Aspectos Técnicos e Aplicações Educacionais". In: Cardoso, A.; Lamounier Jr, E. editores. Realidade Virtual: Uma Abordagem Prá-

tica. Livro dos Minicursos do SVR2004, SBC, São Paulo, 2004, p. 141-183.

DSVideo-Lib,<http://sourceforge.net/project/showfiles.php?group_id=75107&package_id=75491> acesso em fev, 2007.

GEIGER, C.; REIMANN, C.; STICKLEIN, J.; PAELKE, V. (2002) "JARToolKit- A Java binding for ARToolKit." In: The First IEEE Augmented Reality ToolKit International Workshop, p124, 2002.

GLUT, The OpenGL Utility Toolkit.< http://www.xmission.com/ ~ nate/glut.html> acesso em fev, 2007

GNU General Public License< http://www.gnu.org/licenses/gpl .html #TOC1 >acesso em fev, 2007

Kato, H.; Billinghurst, M.(1999) Marker Tracking and HMD Calibration for a Videobased Augmented Reality Conferencing System. In: Proceedings of the 2nd IEEE and ACM Internationall Workshop on Augmented Reality, San Francisco, CA,USA, p85-94, 1999.

Kato, H.; Billinghurst, M.; Poupyrev, I. (2000) "ARToolKit Version 2.33", Human Interface Lab, Universidade de Washington.

Kato, H. Inside ARToolKit.(2002).< http://www.hitl.washington .edu/artoolkit/Papers/ART02-Tutorial.pdf> acesso em fev, 2007.

Kirner, C.; Tori, R.(2004) "Introdução à Realidade Virtual, Realidade Misturada e Hiper-realidade". In: Claudio Kirner; Romero Tori. (Ed.). Realidade Virtual: Conceitos, Tecnologia e Tendências. São Paulo, 2004, v. 1, p. 3-20.

Kirner, C.; Tori, R.(2006) "Fundamentos de Realidade Aumentada". In: Claudio Kirner; Romero Tori; Robson Siscoutto. (Ed.).Fundamentos e Tecnologia de Realidade Virtual e Aumentada. Pré Simpósio SVR 2006, SBC, Belém, 2006, pp. 22-37.

Kirner, C .(2007) pyARToolKit <www.ckirner.com>

Lepetit, V.; Fua, P.(2005) "Monocular Model-Based 3D Tracking of Rigid Objects: A Survey". Foundations and Trends in Computer Graphics and Vision, Vol.1, N.1, pp. 1-89, Out 2005.

Looser, J.; Grasset,R.; Seichter, H,; Billinghurst, B.(2006) "OSGART - A pragmatic approach to MR".Proceedings of the Fifth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR06).

OpenVRML,< http://www.openvrml.org/> acesso em fev, 2007.

Piekarski, W.; Thomas, B. H.(2002) Using ARToolKit for 3D Hand Position Tracking in Mobile Outdoor Environments, In 2nd International Augmented Reality Toolkit Workshop, Darmstadt, Set,2002.

WAGNER, D.; SCHMALSTIEG, D.(2003) "ARToolKit on the PocketPC Platform". In:Proceedings of the 2nd Augmented Reality Toolkit Workshop, p14-15, 2003.

Capítulo 6

OpenCV

João Bernardes, Fábio Miranda, Daniel Calife, Lucas Trias, Romero Tori.

6.1. Introdução a Processamento de Imagens e Visão Computacional

A OpenCV, ou Open Computer Vision library, é, como seu nome já indica, uma biblioteca para visão computacional. Visão Computacional (ou CV, do termo em inglês) é o estudo e as técnicas de obtenção automática de informação a partir de imagens e pode ser classificada como uma sub-área da Computação Gráfica ou de Inteligência Artificial. Conceituar informação foge do escopo desse trabalho, mas intuitivamente pode-se dizer que são dados com algum significado, que podem inclusive ser usados num processo de tomada de decisão. A análise de imagens, em contrapartida, normalmente é definida como a área da computação gráfica que permite a extração de dados (não necessariamente informação) de uma imagem, ou sua conversão para dados. Visão de máquina (*machine vision*) é um termo muitas vezes usado como sinônimo de CV, ou como uma área correlata voltada à aplicação da visão a máquinas, como robôs ou veículos autônomos.

Já o processamento de imagens pode ser considerado uma área distinta da visão computacional, embora seja usado por ela. Trata-se da conversão de uma imagem em outra, por exemplo para eliminar ruído, aumentar o contraste etc. Tanto é importante para a

visão computacional que técnicas de processamento estão presente na OpenCV e são discutidas brevemente na seção 6.3.1.

Mas qual a aplicação de visão computacional em RV ou RA? Uma das principais aplicações em RA é na composição por vídeo das imagens real e virtual. Muitas vezes é necessário, por exemplo, decidir que partes da imagem real eqüivalem a objetos que devem ser inseridos no ambiente virtual (por exemplo, a imagem do usuário como um vídeo-avatar) para poder separá-las do resto da imagem. A obtenção dessa informação (o que é objeto de interesse e o que é "o resto") é uma das aplicações mais importantes de CV, normalmente chamada de *feature extraction*. Pode-se dizer que um caso particular de *feature extraction* é a remoção de fundo (nesse caso a *feature* desejada é tudo que se encontra no *foreground* da imagem).

Outras aplicações, tanto para RA quanto para RV que têm crescido nos últimos anos com o barateamento do hardware para aquisição de imagens e para operar sobre elas em tempo real, é o uso de CV como um dispositivo de entrada do usuário e como um sensor para detectar posição e orientação de objetos ou partes de objeto com um custo menor que o uso de sensores ativos (por exemplo os magnéticos). O ARToolKit, por exemplo, é uma aplicação bastante popular do uso de técnicas de CV para obter a posição e orientação de marcadores fiduciais, que nesse caso funcionam como sensores passivos. Como um dispositivo de entrada a câmera pode utilizar a posição, postura, direção do olhar, expressões faciais, gestos ou a própria interação do usuário com objetos reais ou inclusive diretamente com os virtuais, "tocando-os", como formas de interação. Algumas das vantagens do uso de câmeras e CV para essas funções, além do custo mais baixo, são a "liberdade" do usuário, que não precisa carregar ou vestir dispositivos ou sensores, e o fato de que muitas vezes a câmera e o processamento de imagens já fazem parte do sistema (como no caso da combinação por vídeo de

elementos reais e virtuais em RA, já citado), de forma que é natural usá-los também com essas finalidades. Algumas das principais desvantagens de seu uso são o custo computacional normalmente mais elevado, inclusive podendo ser necessárias técnicas complicadas estatísticas ou de IA, a precisão normalmente mais baixa se comparada ao uso de sensores ativos e problemas oriundos da oclusão total ou parcial dos objetos de interesse.

O restante desse capítulo discute especificamente a OpenCV. Ao contrário de outras APIs utilizadas em RA, essa não oferece uma única solução *ready-to-use* (como o uso de marcadores fiduciais), mas sim uma gama de diversas opções de solução. O preço que se paga por essa flexibilidade é uma complexidade maior em seu uso e a necessidade de conhecimentos básicos de processamento de imagens e visão computacional. Esse capítulo busca fornecer justamente esse conhecimento básico, principalmente de ordem prática, para permitir que o leitor possa começar a usar e explorar essa API com mais facilidade. Não há a pretensão, no entanto, de se cobrir todas as técnicas e soluções existentes na OpenCV.

6.2. Começando a usar o OpenCV

6.2.1. Visão geral da API, Licensa, Download e Instalação

Seus criadores definem a OpenCV como "uma coleção de funções em C (e umas poucas classes em C++) que implementam diversos algoritmos de processamento de imagens e visão computacional". Além disso, o OpenCV tem um foco explícito em aplicações de tempo real, tornando-o interessante para uso em RV e RA. A biblioteca é gratuita, inclusive para uso comercial, e tem código aberto (a licensa pode ser vista em detalhe em http://www.intel.com/technology/computing/opencv/license.htm). A biblioteca está armazenada no sourceforge e o *download* do último *release* pode ser feito em http://sourceforge.net/projects/opencvlibrary/. Além de *releases* para MS Windows e Linux, pode ser feito o *download* da

documentação da ferramenta (que também está disponível *online* no sourceforge como uma wiki) e de cursos sobre a ferramenta, em inglês. No windows a instalação é bastante simples, visto que a distribuição é feita através de um executável de instalação. A configuração de um projeto para que utilize o OpenCV, no entanto, não é tão simples e será tratada em 6.2.3.

O OpenCV divide-se em 5 partes principais. CXCORE contém as estruturas de dados e funcionalidades para armazenamento e manipulação dos diversos tipos de dados usados na biblioteca, incluindo funções matemáticas e até de desenho. CV tem os principais algoritmos de visão computacional e processamento de imagens que não usam aprendizado de máquina. Esses últimos se encontram na MLL (*Machine Learning Library*). HighGUI contém funções para abertura, gravação e exibição de imagens estáticas, captura de e saída para *streams* de vídeo e manipulação de janelas. Por fim, CVCam é um módulo multiplataforma para processamento das streams de vídeo de câmeras digitais. Normalmente HighGUI, CXCore e CV são as mais usadas explicitamente.

6.2.2. Linguagens e plataformas

As versões da OpenCV publicadas pela comunidade de desenvolvedores atualmente têm suporte oficial para as plataformas Windows, Linux e Mac OS X. Em cada uma destas plataformas, foi construída de forma a fazer um bom trabalho de se aproveitar de recursos e APIs nativas para acesso a dispositivos de captura de imagem, interfaces com biblioteca de interface gráfica e invocação de codecs específicos para apresentação de determinados arquivos de vídeo.

Apesar de atualmente suportada apenas nas plataformas mencionadas, a OpenCV está programada de forma bastante modular em ANSI C, e é possível adaptá-la a novos sistemas operacionais ou novas arquiteturas com relativa simplicidade, especialmente

se os sistemas para os quais se pretende portá-la forem baseados no Unix. Os processadores ARM, encontrados em alguns celulares e PDAs, são capazes de executar a OpenCV através de uma outra biblioteca da Intel chamada IPP – *Integrated Performance Primitives* – que emula processamento em ponto flutuante com instruções de aritmética inteira.

A linguagem de programação que consegue acesso mais fácil e direto aos recursos da OpenCV é a própria linguagem C em que foi programada, e que está disponível em todas as plataformas. A distribuição oficial da OpenCV também acompanha algum suporte a duas outras linguagens, Ch e Python. Ch é uma linguagem de *scripting*, ou seja, interpretada dinamicamente, baseada na sintaxe da linguagem C, desenvolvida pela companhia Soft Integration, e que está disponível gratuitamente para uso pessoal ou acadêmico. No código-fonte dos exemplos que acompanham a OpenCV, há definições de métodos e inclusões de bibliotecas de formas específicas para quando estes fontes forem interpretados via Ch, mas é necessário fazer o download de um pacote complementar chamado ch-opencv disponível na página do projeto da OpenCV no repositório Sourceforge.

O suporte da OpenCV para Python é bem recente e é desenvolvido por membros da comunidade que trabalham com a biblioteca. Para que este suporte funcione, é preciso recompilar a OpenCV no Linux e passar o *flag* –with-python para o *script* de configuração. No Windows é necessário apenas compilar a ponte entre o Python e a OpenCV, através da execução de um script chamado setup.py que fica no diretório OpenCV\interfaces\swig\python. Para que este *script* funcione, é importante que o Python tenha sido compilado com o mesmo compilador C/C++ presente no sistema e é necessário ter instaladas as extensões PyWin32.

Por conta de a OpenCV se apresentar, em última análise, como uma série de operações dentro de bibliotecas carregadas di-

namicamente (dlls no Windows, arquivos só no Linux e dylibs no Mac OS X), é relativamente simples invocar suas funções a partir de linguagens disponíveis em cada uma das plataformas. Especialmente quando se tem uma necessidade específica de apenas algumas funções da OpenCV e não se quer realizar um porte completo como os casos do Python e da Ch mencionados acima. Particularmente, existe um gerador de vínculos entre bibliotecas dinâmicas escritas em C e outras linguagens chamado Swig (http://swig.org) (o mesmo usado na criação dos *bindings* Python) e permite exportar funções para diversas linguagens: C# , Java, Perl, PHP e Ruby, entre outros.

Há ainda funções da OpenCV nas bibliotecas prontas para executar SharperCV (para C#, em http://www.cs.ru.ac.za/research/groups/SharperCV/) e Scilab Image and Vídeo Processing Toolbox (http://sivp.sourceforge.net/).

6.2.3. Setup Básico e construindo seu "Hello World"

Nesta seção é apresentada a configuração básica para trabalhar com o OpenCV, como por exemplo os includes, bibliotecas e diretórios necessários e onde encontrá-los.

Como demonstração prática e teste da configuração é desenvolvido e explicado o código de um programa simples utilizando o OpenCV, que utiliza suas estruturas básicas para carregar uma imagem utilizando uma webcam, modificá-la (mais especificamente, inverter suas cores) e exibi-la ou salvá-la.

Setup Básico

Após a instalação é necessário configurar sua IDE para utilizar os arquivos e diretórios do OpenCV necessários para compilar seu projeto. Esta configuração foi realizada utilizando o Visual Studio 2005 da Microsoft, mas pode ser adaptada para qualquer outro ambiente de programação em C++ (OpenCV, 2007). É consi-

derado nessa configuração que o OpenCV foi instalado em: "C:\Program Files\OpenCV".

Nas configurações de diretórios do projeto os arquivos de bibliotecas devem ser localizados em:

C:\Program Files\OpenCV\lib

Cada módulo do OpenCV possuí um diretório próprio de arquivos cabeçalhos, é necessário apontá-los como diretórios de include:

C:\Program Files\OpenCV\cv\include

C:\Program Files\OpenCV\cxcore\include

C:\Program Files\OpenCV\otherlibs\highgui

C:\Program Files\OpenCV\cvaux\include

C:\Program Files\OpenCV\otherlibs\cvcam\include

Assim como os cabeçalhos os arquivos fonte possuem diretórios próprios e estão localizados em:

C:\Program Files\OpenCV\cv\src

C:\Program Files\OpenCV\cxcore\src

C:\Program Files\OpenCV\cvaux\src

C:\Program Files\OpenCV\otherlibs\highgui

C:\Program Files\OpenCV\otherlibs\cvcam\src\windows

No Visual Studio ainda é necessário apontar todos os arquivos de biblioteca (.lib) como entrada para o Linker, estes arquivos são:

C:\Program Files\OpenCV\lib\cv.lib

C:\Program Files\OpenCV\lib\cxcore.lib

C:\Program Files\OpenCV\lib\highgui.lib

C:\Program Files\OpenCV\lib\cvaux.lib

C:\Program Files\OpenCV\lib\cvcam.lib

Em seu arquivo fonte do projeto (.cpp) devem ser relacionados os seguintes arquivos cabeçalhos:

```
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
```

Se tudo estiver correto o programa exemplo na seção seguinte deve rodar sem problemas (o programa requer uma webcam).

Criando o "Hello World"

O objetivo deste programa é testar a configuração do OpenCV e apresentar seus principais comandos para iniciar a captura de uma webcam, tratar a estrutura básica de uma imagem, modificar esta imagem e salvá-la.

O programa deve iniciar com as diretivas de #include e com a função principal:

```
#include "stdafx.h"
#include <cv.h>
#include <highgui.h>
#include <cxcore.h>

int main(int argc, char* argv[])
{
```

Na função main é criado um ponteiro para `IplImage`, que é a estrutura básica de uma imagem no OpenCV. Também é criado um ponteiro para acessar diretamente os dados da imagem e variá-

veis que guardam as informações desta imagem, como altura e largura.

```
IplImage* img = 0; //estrutura basica de imagens
uchar *data;                  //ponteiro para os dados da imagem
int height,width,step,channels; //informações da img
```

Para acessar a webcam é necessário definir a estrutura que trata os dispositivos de captura, e selecionar o tipo de captura e o dispositivo selecionado. Nas linhas seguintes a estrutura `CvCapture` é criada e definida como o qualquer dispositivo de câmera que o Sistema Operacional encontrar (é necessário que a webcam esteja instalada). Logo após é feita uma verificação do dispositivo de captura.

```
CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
if( !capture ) {
    fprintf(stderr, "ERRO: dispositivo de captura não encontrado \n" );
    getchar();
    return -1;
}
```

Para exibir a imagem é criada uma janela padrão do Windows com tamanho automático, o nome da janela será “helloworld”.

```
// Cria uma Janela onde a imagem será exibida
cvNamedWindow( "helloworld", CV_WINDOW_AUTOSIZE );
```

Uma das formas de capturar uma imagem da webcam é através da função `cvQueryFrame`, que retorna uma imagem na estrutura `IplImage`. Obs.: caso você não possua uma webcam ou precise trabalhar com imagens estáticas é possível trocar a função

cvQueryFrame pela função cvLoadImage que carrega uma imagem. Por exemplo: `cvLoadImage("imagem.bmp")` também retorna uma estrutura `IplImage`. Nesse caso não há necessidade do trecho de código que procura a câmera, mostrado acima.

```
// captura um frame
img = cvQueryFrame( capture );
if( !img ) {
    fprintf(stderr, "ERRO: imagem não carregada.\n" );
    getchar();
    return -1;
}
```

Com a imagem carregada é necessário obter as suas informações, estas informações são: altura e largura, step (é o número de elementos em cada linha da imagem, normalmente a largura da imagem multiplicada pelo número de canais da imagem), o número de canais de cor (3 canais se a imagem for RGB) e um ponteiro para acesso direto aos dados da imagem.

```
// guarda as informações da imagem
height    = img->height;
width     = img->width;
step      = img->widthStep;
channels  = img->nChannels;
data      = (uchar *)img->imageData;
```

Estas informações são necessárias pois os dados da imagem, apontados por data, são guardados como um único vetor, e não em uma matriz como a estrutura sugere (Altura x Largura x Canais). Para acessar corretamente um pixel é necessário realizar a seguinte operação: L*step + C*canais + CH, onde:

L = linha do pixel desejado;

C = coluna do pixel desejado;

CH = canal do pixel desejado, por exemplo 1 = canal Green, em uma imagem RGB;

Step = intuitivamente seria a largura da imagem multiplicada pelo número de canais, mas como é possível que existam hiatos entre as linhas (por exemplo para que as dimensões da imagem sejam potências de 2, o que acelera alguns algoritmos), é mais seguro utilizar esta variável;

Canais = número de canais da imagem, por exemplo 3 canais em uma imagem RGB.

```
//inverte a imagem
for(int i=0;i<height;i++)
    for(int j=0;j<width;j++)  {
        // canal Blue
        data[i*step+j*channels]=255-
                          data[i*step+j*channels];
        // canal Green
        data[i*step+j*channels+1]=255-
                          data[i*step+j*channels+1];
        // canal Red
        data[i*step+j*channels+2]=255-
                          data[i*step+j*channels+2];
    }
```

O trecho de código acima percorre toda imagem e inverte as cores de cada pixel. Note que os canais são armazenados na ordem BGR e não RGB como seria o mais natural.

As próximas linhas de código utiliza uma função para mostrar a imagem modificada na janela criada, e usa a função cvWait-

Key() para que o usuário possa ver a janela até pressionar uma tecla.

```
// mostra a imagem na janela
cvShowImage( "helloworld", img );
// espera por uma tecla
cvWaitKey();
```

Para salvar a imagem em um arquivo, no mesmo diretório do projeto, é utilizada a função na linha de código seguinte.

```
// salva a imagem em um arquivo
cvSaveImage("imagemnova.bmp", img);
```

A última etapa deste programa libera o dispositivo de captura e destrói a janela criada, depois envia o retorno da função main() e termina o programa.

```
// libera o dispositivo de captura e destroi a janela
cvReleaseCapture( &capture );
cvDestroyWindow( "helloworld" );
return 0;
}
```

6.3. Algumas técnicas simples de visão no OpenCV

6.3.1. Processamento de imagens

O OpenCV dispõe de uma vasta coleção de ferramentas para processamento de imagens. Algoritmos de transformações geométricas, conversão de espaços de cor, operações morfológicas e cômputo e análise e histogramas, entre outros, são implementados em funções de chamada simples.

As funções `cvGetAffineTransform`, `cv2DRotationMatrix`, `cvWarpAffine` e `cvResize` implementam transformações afins, ou seja, aquelas que preservam a

colinearidade e a razão entre as distâncias. A primeira computa a matriz de transformação afim a partir de seis pontos correspondentes, por exemplo, os vértices de dois triângulos em planos distintos. Deste modo obtém-se uma transformação que mapeia pontos do primeiro plano para o segundo plano. A segunda também retorna uma matriz de transformação, porém baseando-se em um ponto que representa o centro de uma rotação e na magnitude desta. A função `cvWarpAffine` aplica a transformação contida nas matrizes parâmetro a uma imagem. A quarta função, `cvResize` permite redimensionar imagens. Ela recebe uma imagem de origem e uma de destino, e as dimensões da primeira são alteradas para que se encaixem na segunda.

Também é possível aplicar transformações perspectivas, através das funções `cvGetPerspectiveTransform` e `cvWarpPerspective`. Elas funcionam de modo análogo às funções para transformações afins, ou seja, a primeira provê uma matriz de transformação baseada em um conjunto de pontos equivalentes(quatro pontos neste caso) que é usada pela segunda para transformar imagens.

Existem ainda as funções `cvRemap` e `cvLogPolar`, que também implementam transformações. A primeira aplica uma transformação geométrica genérica, representada por imagens que são usadas como mapas para coordenadas X e Y da imagem original. Já a função `cvLogPolar` mapeia a imagem do espaço euclidiano para o espaço log-polar, que tem um comportamento mais próximo com o sistema de visão humano do que o euclidiano. Este comportamento é especialmente útil quando deseja-se simular a percepção visual que uma pessoa tem de uma cena.

As operações morfológicas são tratadas em dois níveis. Os algoritmos de dilatação e erosão, bases para outras operações, têm chamadas próprias via `cvDilate` e `cvErode` respectivamente. Já os algoritmos de abertura, fechamento, gradiente, *top hat e black*

hat, baseados em dilatações e erosões, são todos executados por meio da função `cvMorphologyEx`, sendo selecionados através de um parâmetro. O algoritmo de *watershed*, que não é necessariamente dependente de dilatações e erosões, é implementado em uma função específica, a `cvWatershed`. A transformada de distância, algoritmo que associa à cor de um pixel não nulo o valor da sua distância até o píxel nulo mais próximo, é executado pela função `cvDistTransform`.

Uma questão importante quando se fala de transformações geométricas em imagens é a interpolação. O tipo de interpolação utilizado é crucial e é ele que permite optar entre grande qualidade ou grande desempenho. O OpenCV implementa quatro métodos de interpolação: vizinho mais próximo, bilinear, por área e bicúbica. Por padrão é utilizada a interpolação bilinear, que permite uma qualidade aceitável na interpolação sem ter alto custo computacional, e os outros tipos de interpolação podem ser selecionados através de parâmetros nas funções de transformação. A interpolação por vizinho mais próximo é a mais rápida, e a que provê menor qualidade, ela é selecionada pela constante CV_INTER_NN passada como parâmetro para as funções. A bilinear não precisa ser explicitamente chamada, pois é usada sempre que nenhum tipo é informado e sua constante é CV_INTER_LINEAR. A interpolação por área tem usos específicos, por exemplo quando se deseja quantizar uma imagem sem sofrer o efeito Moiré. Em aplicações como a escala de imagens o resultado é bastante semelhante ao da interpolação por vizinho mais próximo ao passo que o custo computacional é maior. A seleção da interpolação por área é feita pela constante CV_INTER_AREA. Finalmente, a interpolação bicúbica, selecionada por CV_INTER_CUBIC, é a que provê maior qualidade para a imagem, e a que mais consome processamento.

Os histogramas são tratados por uma série de funções que permitem, além do cômputo (`cvCalcHist`) e acesso a seus valo-

res, operações como normalização (`cvNormalizeHist`), limiarização (`cvThreholdHist`), comparação (`cvCompareHist`) e equalização (`cvEqualizeHist`). É importante destacar que todas estas funcionalidades são disponíveis não só para imagens, mas para quaisquer populações de valores numéricos, o que dá maior flexibilidade à biblioteca.

Outros recursos poderosos que o OpenCV implementa são os filtros de diversos tipos. Através da função `cvSmooth`, é possível aplicar suavizações às imagens. Pode-se optar por suavização sem escala, pela soma simples da vizinhança, gaussiana (usada como padrão), pela média e bilateral através das constantes CV_BLUR_NO_SCALE, CV_BLUR, CV_GAUSSIAN, CV_MEDIAN, e CV_BILATERAL respectivamente, passando-as como parâmetro à função.

Convoluções com *kernels* genéricos podem ser aplicadas por meio da função cvFilter2D. Ela efetua filtragem linear na imagem de acordo com o kernel recebido, e trata automaticamente as bordas da imagem pelos vizinhos mais próximos, para que não ocorram tentativas de acesso a pixels que na verdade não existem.

Outra alternativa para evitar este tipo perigoso de tentativa de acesso é a função `cvCopyMakeBorder`. Ela faz uma cópia da imagem original, adicionando uma borda a esta. A cor da borda pode ser escolhida de duas formas, através das constantes IPL_BORDER_CONSTANT e IPL_BORDER_REPLICATE. A primeira usa uma cor constante que deve ser informada, e é útil, por exemplo, quando a evolução de um algoritmo é controlada por informação tonal. Já a segunda replica os tons dos píxeis das linhas superior e inferior e das colunas mais à esquerda e mais à direita para preencher a borda.

A conversão de espaços de cor também é uma filtragem de grande importância. Através dela é possível efetuar modificações em uma imagem, obtendo uma cujos canais representam valores

diferentes do BGR usado como padrão. São possíveis conversões de entre BGR e escala de cinzas, CIE XYZ, YCrCb JPEG, HSV, HLS, CIE L*a*b*, CIE L*u*v*, e Bayer(muito usado em sensores CCD e CMOS) . Estas tranformações podem ser aplicadas através da função `cvCvtColor`, passando a imagem original, a que receberá o resultado da conversão e uma constante da forma CV_<espaço original>2<espaço de destino>, que informa qual a conversão que deve ser feita.

6.3.2. Subtração de fundo

O problema da subtração de fundo pode ser resumido da seguinte forma: dada uma imagem atual da cena, quais são os objetos de nosso interesse e quais podemos classificar como fundo? Em geral este problema não é formulado de maneira tão aberta, e no contexto de aplicações interativas simples pode ser bastante simplificado quando o fundo não é composto de objetos móveis.

A abordagem mais simples que se pode adotar para este problema é comparar a imagem atual com uma imagem da estática da cena, antes de os objetos móveis poderem ser vistos. O objetivo final desta comparação é em geral obter uma imagem em preto e branco que possa ser usada como máscara – multiplicar a imagem que vem da câmera para que se possa ter uma imagem resultante em que aparecem só os objetos de interesse, sem o fundo.

Figura 1 - Imagem original (a), subtração de fundo (b), limiarização (c) e abertura morfológica (d). *Fonte: Projeto Câmera Kombat (http://camerakombat.googlepages.com)*

Na Figura 1, pode-se observar na parte (a) uma imagem assim como é captada pela câmera, em seguida executa-se a subtra-

ção simples (b) entre este quadro atual e uma imagem da cena estática, antes que aparecesse o objeto de interesse. É interessante observar que naqueles pixels da imagem em que o fundo permaneceu inalterado, esta subtração retorna valores próximos de zero, e consequentemente a parte estática da cena fica praticamente preta. Para segmentar o objeto de interesse, é necessário aplicar um limiar (c) na imagem obtida até então para separar completamente o sujeito do fundo, o valor deste limiar depende muito das condições particulares da aplicação e da iluminação. Na figura limiarizada, observa-se algumas imperfeições e regiões dentro do objeto de interesse erroneamente classificadas como fundo. Em geral operações morfológicas de abertura e fechamento têm a propriedade de suavizar o contorno de objetos de interesse e remover pequenos buracos e saliências – foi aplicada a operação morfológica de abertura e o resultado foi (d).

A programação desta idéia através da OpenCV é bastante simples e pode ser vista no trecho de código a seguir, em que imagemAtual é a imagem capturada pela câmera num determinado momento, frente é a imagem binarizada com os elementos de interesse e fundo é um frame que representa o fundo da imagem. A função `cvAbsDiff` calcula o módulo da diferença entre as duas imagens.

```
/* Recebe uma imagem em grayscale e devolve uma nova
imagem, limpa, na saída */

void limiariza_e_limpa(IplImage* entrada, IplConvKer-
        nel* estruturante, IplImage* saida){

    // Funcao que calcula o limiar

    cvThreshold(entrada, saida, limiar, 255,
         CV_THRESH_BINARY);

    // Funcao que aplica a abertura
```

```
    cvMorphologyEx(saida, saida, tempImage, estrutu-
        rante, CV_MOP_CLOSE, 2);
}
/* Converte para imagem em tons de cinza */
IplImage* paraTonsDeCinza(IplImage * entrada)
{
    IplImage* cinza = cvCreateImage(cvGetSize(frame),
        IPL_DEPTH_8U, 1);
    cinza->origin = entrada->origin;
    cvCvtColor(entrada, cinza, CV_RGB2GRAY);
    return gray;
}
```

Após realizada a diferença entre imagens, a imagem de frente pode conter pequenos artefatos resultantes de variações de iluminação, sombras e pequenas movimentações da câmera. Para a resolução destes problemas recomenda-se que sejam aplicadas operações morfológicas, discutidas na seção 8.3.1. O trecho de código anterior faz uso de das funções `paraTonsDeCinza` e `limia-riza_e_limpa` que, respectivamente, alocam uma imagem com apenas um canal de 8 bits e convertem a imagem original capturada de uma fonte para tons de cinza e aplicam a operação de limiar e operações morfológicas para limpar a imagem resultante. Estas funções são exemplificadas a seguir (reticências representam trechos omitidos).

```
[ . . . ]
while(1) {
frame = 0;
// Captura imagem da fonte
frame = cvQueryFrame(capture);
[ . . . ]
image = cvCreateImage(cvGetSize(frame), 8, 3);
```

```
imagemAtual = cvCreateImage(cvGetSize(frame), 8, 1);
image->origin = frame->origin;
cvCopy(frame, image);
// Converte a imagem para tons de cinza
imagemAtual = paraTonsDeCinza(image);
// Subtrai a imagem do fundo da atual
cvAbsDiff(imagemAtual, fundo, frente);
frente->origin = fundo->origin;
// Aplica o limiar e a abertura
limiariza_e_limpa(frente, ee, frente);
[ . . . ]
```

Em algumas situações, como por exemplo casos em que a luminosidade do fundo varia ao longo do tempo, pode ser insuficiente apenas subtrair o quadro atual de um quadro que representa o fundo. Uma abordagem que se costuma pode adotar neste caso é considerar com fundo a média de n quadros anteriores, de modo que variações nas condições no ambiente possam ser incorporadas ao fundo. A OpenCV provê funções interessantes para que se possa usar uma abordagem como essa, por exemplo `cvAcc`, que acumula uma imagem em um *buffer* (que deve ser do tipo `IplImage`), ou então `cvRunningAvg`, que pode realizar uma soma ponderada de duas imagens, e é interessante quando se quer gradualmente atualizar a imagem de fundo com as imagens mais novas.

Convém ressaltar que a detecção e subtração de fundo é um problema aberto e objeto de muita pesquisa, anualmente, na conferência internacional de multimídia (ACM Multimedia 2006), há uma competição específica para tratar do problema, em que equipes das instituições mais renomadas procuram enfrentar o problema, longe de estar resolvido de forma definitiva. Algumas técnicas mais flexíveis e avançadas encontradas recentemente (Han et al, 1999; Stauffer et al, 2004) usam princípios estatísticos e representam o

fundo da cena como um conjunto de curvas gaussianas, com o resultado de obter mais robustez face a variações nas condições da cena ou mesmo movimentos da câmera.

6.3.3. Feature Extraction

Além das funções de transformação e conversão o OpenCV provê funções para análise de imagens e extração de características importantes.

Os algoritmos de detecção de bordas mais comuns são implementados em funções de chamada simples, recebendo basicamente como parâmetros a imagem de origem, a imagem de destino e valores específicos de cada algoritmo, por exemplo, limiares ou a ordem desejada da derivada da imagem.

A função `cvSobel` permite aplicar o algoritmo de Sobel com derivadas de até terceira ordem. É possível selecionar de forma independente a ordem da derivada aplicada na horizontal e na vertical, assim como o tamanho da máscara utilizada.

O algoritmo de Laplace, implementado na função `cvLaplace`, também permite que o tamanho da máscara utilizada seja escolhido, assim como o de Canny, `cvCanny`, que além disso permite escolher os limiares que definem a detecção e a conectividade das bordas identificadas.

A detecção e rotulação de componentes conexos também possui um tratamento prático e eficiente. Existe uma estrutura de dados, `CvConnectedComp`, específica para representá-los. Ela armazena sua área, seu matiz de cinza e o menor retângulo que o inscreve, numa representação própria do OpenCV.

É possível rotular, ou seja, pintar com uma dada cor, um componente conexo através da função `cvFloodFill`. Ela recebe como parâmetros a imagem a ser rotulada, as coordenadas do pixel onde a rotulação deve começar e a cor que deve ser atribuída ao

componente. Além disso é possível usar máscaras para a rotulação, evitando que a imagem seja alterada, definir a diferença mínima e máxima entre dois pixels para que eles sejam considerados do mesmo componente.

O OpenCV permite também a eficiente detecção e acesso a contornos de imagens binárias. A função `cvFindContours` detecta contornos em imagens em preto e branco, e recebe como parâmetros a imagem, uma estrutura de armazenamento do tipo `CvStorage` que vai conter os contornos detectados e um ponteiro que receberá o endereço do primeiro contorno detectado, ou seja, do contorno mais externo. Também é possível optar, por meio de constantes, como vai ser o modo de operação da função e qual vai ser o método usado para fazer aproximações durante a detecção.

A constantes CV_RETR_EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP e CV_RETR_TREE são responsáveis por controlar o modo de operação. A primeira faz com que a função retorne somente os contornos externos dos objetos. Com a segunda, a função retornará todos os contornos encontrados em uma lista. A terceira constante dia à função que esta deve retornar todos os contornos encontrados organizados em hierarquias de dois níveis, onde o nível superior representa a fronteira dos objetos e o inferior, o contorno dos furos. Finalmente, a quarta constante faz com que a função retorne todos os contornos detectados em uma estrutura hierárquica de árvore representando a estrutura de encadeamento de contornos da imagem.

Já as constantes CV_CHAIN_CODE, CV_CHAIN_APPROX_NONE, CV_CHAIN_APPROX_SIMPLE, CV_CHAIN_APPROX_TC_89_L1 e CV_LINK_RUNS controlam o método de aproximação usado dentro do algoritmo. Com primeira, os contornos returnados serão aproximados pela codificação de cadeia de Freeman. A segunda faz com que os pontos codificados da cadeia de Freeman sejam transladados para pontos comuns. Com

a terceira constante, é aplicada uma compressão nos segmentos verticais, horizontais e diagonais de forma que estes são aproximados para seus extremos somente. A quarta constante usa aproximações do algoritmo de Teh-Chin. A última constante aplica uma técnica diferente, ligando segmentos horizontais de valores 1 da imagem, e deve ser usada em conjunto com a constante CV_RETR_LIST.

Além da função `cvFindContours` é possível detectar contornos de forma seqüencial, usando funções que operam sobre uma estrutura responsável por buscar contornos chamada `CvContourScanner`. A criação de uma estrutura deste tipo se dá pela chamada da função `cvStartFindContours`, que recebe parâmetros e constantes de configuração semelhantes aos da função `cvFindContours` (uma vez que ela é usada para efetivamente achar os contornos na imagem). A função `cvFindNextContour` recebe um `CvContourScanner` e retorna um ponteiro para o próximo contorno obtido. É importante ressaltar que a noção de ordenação de contornos que permite a relação de ordenação entre contornos é dependente da configuração do modo de operação usada. Por exemplo, o próximo contorno de uma lista com todos os contornos encontrados não é necessariamente o próximo contorno de uma árvore de contornos hierarquicamente classificados.

Ainda de acordo com o modo de operação selecionado, a função `cvSubstituteContour` permite substituir o último contorno retornado pela função `cvFindNextContour` por um que é passado como parâmetro, dentro do armazenamento interno da estrutura `CvContourScanner` que está sendo utilizada. Esta função pode ser utilizada, por exemplo, para substituir um ramo completo de uma árvore de contornos de modo prático e eficiente. Quando não for mais necessário um escaneador de contornos, a função `cvEndFindContours` se encarrega de terminar o processo de busca por contornos liberar as estruturas internas utilizadas, e

retorna um ponteiro para o primeiro contorno do nível mais alto da estrutura utilizada de acordo como o modo de operação escolhido.

Os contornos ativos, também conhecidos como snakes, são ferramentas muito utilizadas no rastreamento de objetos. O OpenCV possui a função `cvSnakeImage`, que procura reposicionar um contorno de forma a minimizar a soma de sua energia interna e externa. A energia interna de um contorno é dependente de sua forma, e quanto mais suave menor é a energia. Já a energia externa se relaciona com a imagem, e é minimizada quando o contorno se aproxima das bordas detectáveis, que podem ser destacadas com funções como `cvCanny`, `cvSobel` ou `cvLaplace`. A função `cvSnakeImage` recebe como parâmetros a imagem, uma seqüência de pontos que representam um contorno e vetores que representam os pesos da energia associada à continuidade, os pesos da energia associada à curvatura do contorno e os pesos associados à energia da imagem, para cada um dos pontos que compõem o contorno. Além destes, são passados parâmetros específicos da operação interna da função e critérios de parada dentro de uma estrutura própria do OpenCV, a `CvTermCriteria`. Usualmente são usados como critério de parada o número de iterações e uma precisão mínima desejada na minimização da energia total do contorno.

6.4. Aprendizado de Máquina e Reconhecimento de Padrões

Como já foi mencionado é comum o uso de técnicas de IA, estatística e reconhecimento de padrões na visão computacional. As principais aplicações dessas técnicas são o reconhecimento de formas ou de padrões (por exemplo, mãos, expressões faciais, gestos, tumores, queimadas, obstáculos num caminho...) em áreas tão diversas quanto medicina, geografia, interação homem-máquina ou robótica. O OpenCV possui uma biblioteca especializada no uso de estatística para a classificação, regressão e agrupamento (clustering) de dados, a Machine Learning Library (MLL). A MLL tem foco nas técnicas de aprendizado estatístico através da análise de grandes

volumes de dados de treinamento. Embora essa biblioteca implemente um grande número de técnicas bastante diferentes, há uma funcionalidade padrão, comum a essas técnicas. A maioria está implementada como classes de C++ e conta com as seguintes funcionalidades:

- Um construtor default e um construtor que já recebe os dados de treinamento, além do destrutor;
- Funcionalidade para gravar e ler o modelo treinado em arquivo (`save, load, write, read, clear`);
- Uma função de treinamento (`train`);
- Uma função `predict` que, uma vez que o modelo esteja treinado, prevê sua resposta para uma nova amostra (a classe nos algoritmos de classificação ou um valor no caso de regressão).

Discutir todas as técnicas implementadas na MLL vai muito além do escopo desse livro (de fato, cada uma dessas técnicas pode ser discutida em um ou mais livros dessa dimensão), mas segue uma rápida descrição de algumas das técnicas mais populares.

- Classificadores de Bayes normais assumem que os vetores de *features* (os dados usados na classificação) tem uma distribuição normal, mas não necessariamente independente. O classificador estima o vetor médio e a matriz de covariância para cada classe com base nos dados de treinamento e os usa para a classificação de novas amostras.
- A técnica de vizinho mais próximo (também conhecida como aprendizado por exemplo) simplesmente prevê a resposta de uma amostra com base nas K amostras de treinamento mais próximas dela no espaço multidimensional dos vetores de *features*, por exemplo calculando uma média ponderada desses vizinhos.

- Árvores de decisão são árvores binárias construídas a partir da raiz dos dados de treinamento que podem ser usadas tanto para classificação quanto para regressão. No primeiro caso, cada folha eqüivale a uma classe, enquanto no segundo é uma constante que aproxima a função de interesse naquele ponto.
- O algoritmo de expectativa-maximização assume uma função multidimensional de densidade de probabilidades como um específico de combinações de distribuições Gaussianas. Trata-se de um algoritmo iterativo onde cada interação tem dois passos. No passo de expectativa, calcula-se as probabilidades da amostra pertencer à cada combinação. Com base nessas probabilidades, o passo de maximização refina os parâmetros das gaussianas, aproximando-se dos parâmetros de máxima semelhança. Pode-se provar que esse algoritmo converge para uma estimativa de máxima semelhança.
- Redes neurais: a MLL implementa *Perceptrons* de múltiplas camadas (MLPs), um caso particular de redes *feedforward* muito usado. Cada neurônio da rede pode ter várias entradas e saídas e essas entradas tem pesos, que são resultados do treinamento.

6.5. Exemplos de uso do OpenCV

6.5.1. APIs

A OpenCV, por conta da vasta gama de funcionalidade e ampla disponibilidade, despertou muito interesse da comunidade acadêmica, que a adota como base em diversas pesquisas e no desenvolvimento de novas bibliotecas. Dois notáveis exemplos que se apóiam fortemente na OpenCV para provar funcionalidade de alto nível são as bibliotecas de código aberto AVCSR e HandVU.

A AVCSR (Luhong Liang et al, 2002) - Audio-Visual Computer Speech Recognition – é uma biblioteca para reconhecimento de fala que junta as informações vindas do áudio com imagens do rosto do usuário através de modelos ocultos de Markov acoplados para melhorar a qualidade do reconhecimento, e está disponível na página do projeto da OpenCV junto com seqüências de vídeo de testes.

A HandVU (Kölsch et al, 2004) é uma biblioteca para rastreamento de imagens de mãos e deteção de gestos estáticos numa câmera em primeira pessoa. Com esta biblioteca é possível associar ações à realização de determinadas posturas das mãos do usuário e experimentar com interfaces inovadoras para o controle de aplicações sem o uso de dispositivos convencionais como mouses e teclados. A deteção da HandVU é baseada numa cadeia de classificadores que leva em conta o tom da pele do usuário, modelos construídos dinamicamente para representar as mãos e o fundo e o histórico de movimento de features observadas nas mãos.

6.5.2. Aplicações

Nesta seção serão apresentados dois exemplos de aplicações desenvolvidas pelos autores usando o OpenCV, o Interlab Robot ARena e o SENAC ARHockey.

Robot ARena (Calife et al, 2006) é uma infraestrutura para o desenvolvimento de aplicações interativas em um ambiente com Realidade Aumentada Espacial, cujo principal elemento de interação é um robô.

Como parte desta infraestrutura o OpenCV é utilizado para desenvolver um módulo de software que resolve um dos principais problemas da Realidade Aumentada, o registro e o rastreamento. Nesta infraestrutura é necessário registrar o "campo" real onde o robô se locomove e o ambiente virtual que é projetado neste campo, e também, determinar a posição e direção do robô real neste ambi-

ente, em uma taxa rápida o suficiente para garantir a interação com os elementos virtuais.

Com o OpenCV foi desenvolvida uma classe em C++ que captura as imagens de uma webcam e segmenta os diversos marcadores coloridos, duas etiquetas vermelhas para registrar o campo e um marcador em forma de "T", azul e verde, para rastrear o robô. Foram utilizadas as facilidades do OpenCV para iniciar a câmera e capturar seus frames, fazer conversões entre sistemas de cores, separar os canais de cores das imagens e aplicar limiarizações e filtros.

Uma das principais preocupações durante o desenvolvimento foi a integração desta classe em C++ com a base do sistema desenvolvida em Java. Para esta integração foi utilizado o JNI (Java Native Interface), que a partir de uma classe interface em Java, com as chamadas para os métodos que são necessários acessar da classe em C++, cria um arquivo cabeçalho (.h) que deve ser utilizado para a criação de uma dll. Esta dll é carregada na classe de interface em Java, que chama suas funções em tempo de execução do programa.

Uma outra maneira de utilizar o OpenCV com Java é utilizar um processo similiar, mas fazendo chamadas diretamente às funções do OpenCV.

O ARHockey (Vieira et al, 2006) - Augmented Reality Air Hockey - é um projeto em que foi criado um jogo multijogador, relacionado ao ambiente físico, em que se pode participar sem empecilhos e que explora a inserção de elementos virtuais para criar novas possiblidades de jogo. O desenvolvimento do trabalho se baseia fortemente em conceitos de realidade aumentada espacial e no jogo Air Hockey original.

Além do jogo semelhante ao original, foram criadas inovações na dinâmica do jogo e em suas regras, como adição de barreiras dinâmicas e estáticas, portais que teletransferem o disco e mini-

jogos dentro das partidas. Os únicos componentes reais do jogo são os batedores que cada jogador usa para interagir com o disco, que assim como a mesa, é virtual e projetado sobre um plano. Estes batedores emitem luz infravermelha, que é rastreada pelo sistema de visão, implementado usando OpenCV.

A câmera que filma o ambiente de jogo é equipada com um filtro passa-infravermelho, que remove toda a luz visível permitindo uma visão mais nítida dos batedores. Quando uma imagem é capturada, ela passa por uma limiarização(cvThreshold), e por uma detecção de contornos (cvFindContours). Após isso é computada a média aritmética das posições dos pixels que compõem os contornos dos batedores e o valor encontrado representa a posição do batedor no sistema de coordenadas da câmera. Finalmente é feita uma mudança de sistema de coordenadas, para coordenadas do mundo real, usando a função cvWarpPerspective. Esta função usa uma matriz de homografia, que mapeia posições do sistema de coordenadas do mundo real para o sistema de coordenadas da câmera e vice-versa. Ela é obtida numa etapa inicial de calibração do sistema por meio da função cvFindHomography.

Referências

Calife, D., Tomoyose, A., Spinola, D. and Tori, R. (2006), "Controle e Rastreamento de um Robô Real para um Ambiente de Realidade Misturada", In: II Workshop de Aplicações de Realidade Virtual (WARV 2006), p. 1-4.

OpenCV Library Wiki. http://opencvlibrary.sourceforge.net/ VisualC++, acessado em Fevereiro de 2007.

Vieira, B., Trias, L., Theodoro, C., Miranda, F. e Tori, R. (2006), "ARHockey: Um Jogo em Realidade Aumentada Baseada em Projetores". V Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2006).

ACM Multimedia 2006. "Open source algorithm competition". Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks, International Multimedia Conference. 2006, p. 211-214.

Stauffer, C. e Grimson, W.E.L (1999) "Adaptive background mixture models for real-time tracking", IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR99), 1999

Han, B., Comaniciu, D. e David, L. (2004) Sequential Kernel Density Approximation Through Mode Propagation: Applications to Background Modeling. In proceeding of Asian Conference on Computer Vision (ACCV), Jeju Island, Korea, 2004

Luhong Liang , Xiaoxing Liu , Yibao Zhao, Xiaobo Pi e Nefian, A.V.(2002). "Speaker independent audio-visual continuous speech recognition." Proceedings of the 2002 IEEE International Conference onMultimedia and Expo (ICME '02). v.2, pp 25-28, 2002

Kölsch, M e Turk, M. (2004). "Robust Hand Detection". Proc. IEEE Intl. Conference on Automatic Face and Gesture Recognition, May 2004.

Capítulo 7

O Engine Gráfico OGRE

Veronica Teichrieb[1], Judith Kelner[1], Thiago Farias[1]

[1]Grupo de Pesquisa em Realidade Virtual e Multimídia, Centro de Informática – Universidade Federal de Pernambuco (GRVM CIn UFPE)
Av. Prof. Moraes Rego, S/N – Prédio da Positiva – 1º Andar – Cidade Universitária – 50670-901 – Recife – PE – Brasil

{vt,jk,tsmcf}@cin.ufpe.br

Abstract

OGRE is an open source and object oriented 3D graphics rendering engine. It was conceived with the purpose of making 3D applications development easier and more intuitive for developers. This chapter aims to present this technology, approaching briefly OGRE's main concepts, and giving an overview of its potentialities as graphics applications development engine.

Resumo

O OGRE é um engine de renderização gráfica 3D open source e orientado a objetos. Foi concebido com o propósito de tornar a implementação de aplicações 3D mais fácil e intuitiva para os desenvolvedores. O objetivo deste capítulo é apresentar essa tecnologia, abordando sucintamente os principais conceitos do OGRE, e dando uma visão geral das suas potencialidades como engine de desenvolvimento de aplicações gráficas.

7.1. Introdução ao *Engine* Gráfico OGRE

OGRE (*Object-oriented Graphics Rendering Engine*) é um *engine* gráfico *open-source* que funciona na maioria das plataformas existentes. Ele provê uma vasta gama de *plugins*, ferramentas e *add-ons* que favorecem a criação de vários tipos de aplicações gráficas, empregando diversos conceitos inerentes ao desenvolvimento destas.

O *engine* funciona em várias configurações de *hardware* 3D (GPUs – *Graphics Processing Units*) disponíveis no mercado, desde as obsoletas às mais sofisticadas. Através da interface provida pelo OGRE, o desenvolvedor pode verificar se o *hardware* oferece suporte aos requisitos mínimos para execução da aplicação, e conseqüentemente pode programar técnicas alternativas para requisitos mais sofisticados. Este conceito de técnicas alternativas pode ser aplicado também aos materiais e efeitos, incluindo versões de *shaders* e texturas multicamada.

A interface de programação oferecida nativamente pelo OGRE é escrita em C++. Atualmente, existem alguns *wrappers* para o OGRE em Java, .NET e Python, mas eles ainda encontram-se em desenvolvimento.

O propósito do OGRE não é ser um *game engine*; ele é um *rendering engine* genérico que pode ser incorporado a bibliotecas de tratamento de entradas, de processamento de som e a plataformas com algoritmos de inteligência artificial, compondo um *kit* de desenvolvimento mais completo que oferece suporte ao desenvolvimento de aplicações e jogos 3D [OGRE 2007].

7.1.1. Licença e Instalação

O OGRE é licenciado sob os termos da GNU *Lesser Public License* (LGPL). Mais informações relativas à licença podem ser encontradas em [LGPL 2007].

A instalação do *engine* pode ser feita de duas formas: através da versão pré-compilada do SDK e a partir do código fonte. Detalhes sobre como instalar o OGRE podem ser encontrados em [Farias et al. 2006].

7.2. A Arquitetura do OGRE

A arquitetura adotada pelo OGRE utiliza vários conceitos de "Padrões de Projeto" (*Factories*, *Singletons*, *Observers*, etc.) que a tornam mais compacta e fácil de usar. Os principais componentes da arquitetura são: Root, SceneManager, RenderSystem, Entity, Mesh, SceneNode e Material. O relacionamento entre estes componentes pode ser observado na Figura 7.1 7.1

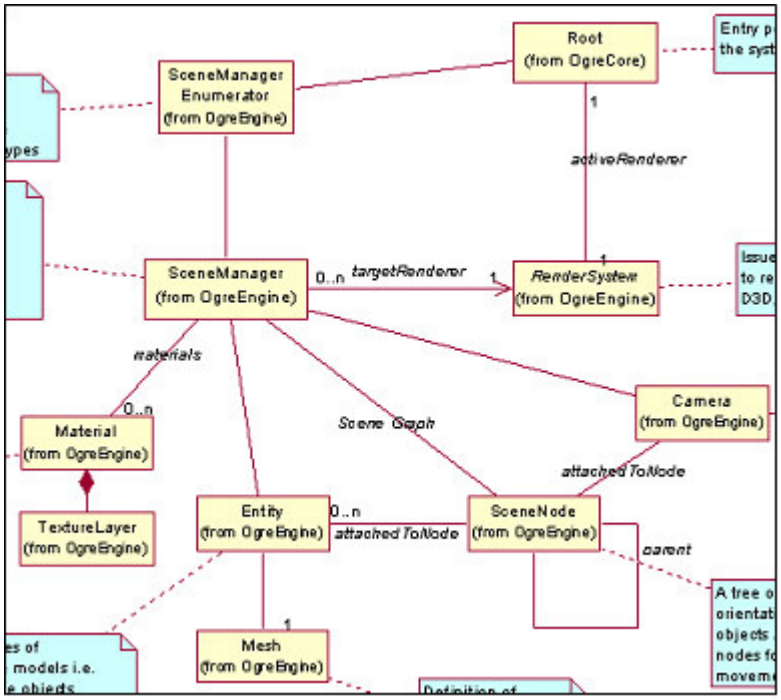


Figura 7.1 Principais componentes da arquitetura do OGRE

7.2.1. O Objeto Root

O objeto Root é o ponto de entrada do OGRE. Ele deve ser instanciado antes de todos os outros objetos, pois é responsável por dar origem a todo o sistema e administrar o acesso ao mesmo. Desta forma, ele também deve ser o último a ser destruído, pois sua destruição significa a finalização do uso da biblioteca OGRE.

A configuração do sistema pode ser feita através do objeto Root, chamando o método `Root::showConfigDialog`, que detecta todas as opções disponibilizadas pelos sistemas de renderização (OpenGL [OpenGL 2007] e DirectX [DirectX 2007]) e exibe uma caixa de diálogo para que o usuário configure a resolução da tela, se a exibição será em tela cheia, entre outras opções (ver Figura 7.2).

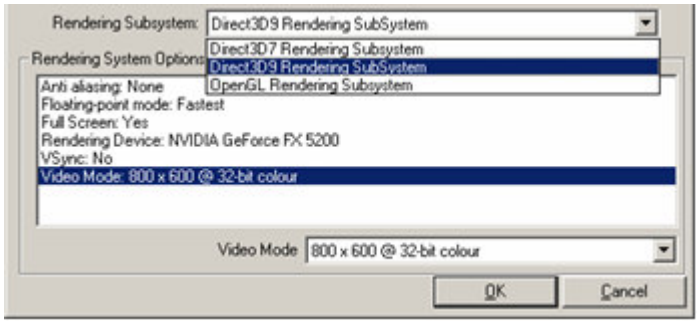


Figura 7.2 Janela de configuração inicial do OGRE

A partir do Root pode-se obter a referência para outros objetos importantes do sistema, tais como o SceneManager, o RenderSystem e outros gerenciadores de recursos, que serão detalhados nas próximas subseções. Fazem parte também das tarefas desempenhadas pelo Root habilitar o modo de renderização contínua (modo utilizado em jogos) e o uso de FrameListeners. FrameListener é uma interface fornecida pelo OGRE, implementada pela classe ExampleFrameListener, que possui recursos de interação com a câmera e exibição de estatísticas [Farias et al. 2006].

7.2.2. O Objeto SceneManager

O SceneManager é o objeto que reúne e organiza todo o conteúdo da cena que será renderizada pelo *engine*. Ele é responsável por aplicar algoritmos para otimizar a exibição da cena, criar e gerenciar todas as câmeras, objetos móveis (Entities, descritas na Subseção 0), luzes, malha que representa o mundo e partes estáticas da cena.

A partir do SceneManager pode-se criar e acessar nós e objetos móveis, a partir de nomes atribuídos a eles. Por exemplo, o método `SceneManager::createSceneNode/getSceneNode` fornece acesso a nós de controle da cena.

O SceneManager também é responsável por enviar toda a cena para o RenderSystem (ver Subseção 7.2.3). Isso é feito automaticamente através do método `SceneManager::_renderScene`, chamado pelo *engine* a cada quadro.

A maior parte da interação com o SceneManager se dá ao longo da criação da cena. Durante este processo uma grande quantidade de objetos é criada, havendo a possibilidade de modificação dinâmica durante o ciclo de renderização, através de um FrameListener.

Diferentes tipos de cenas (*indoor* e *outdoor*) requerem diferentes algoritmos para decidir que objeto pode ser enviado para o RenderSystem (ver Subseção 7.2.3) de forma a manter um bom desempenho na renderização. Por causa disso, o SceneManager foi projetado para ser especializado (através de subclasses) em um determinado tipo de cena. O SceneManager padrão tem um bom desempenho em cenas relativamente pequenas, pois faz pouca ou nenhuma otimização nelas. O ideal é que seja criado um SceneManager para cada tipo de cena, que faça as devidas otimizações na sua organização e seleção de objetos a serem renderizados, assumindo algumas características existentes de acordo com cada tipo. Um e-

xemplo de especialização do SceneManager é o BspSceneManager, que otimiza cenas contendo grandes ambientes fechados baseando-se numa árvore BSP (*Binary Space Partition*). Este tipo de cena é comumente encontrado em jogos, como Quake 3. Outro exemplo de especialização é o PagingLandscapeSceneManager, utilizado em grandes cenas abertas e terrenos extensos. Ele é capaz de otimizar a memória e a velocidade do programa através de técnicas de compressão de vértices e compartilhamento de coordenadas de textura.

As aplicações que utilizam o OGRE não necessitam saber quais subclasses do SceneManager estão disponíveis para utilização. A aplicação pode simplesmente chamar o método `Root::getSceneManager` passando o tipo da cena, que pode assumir valores representados pelas constantes `ST_GENERIC`, `ST_EXTERIOR_CLOSE`, `ST_EXTERIOR_FAR`, `ST_EXTERIOR_REAL_FAR` e `ST_INTERIOR`. Durante a chamada do método o OGRE se encarregará de procurar a subclasse mais próxima da sugerida no tipo, ou retornar a subclasse genérica caso a especialista não esteja disponível. Isto permite ao desenvolvedor criar posteriormente um SceneManager especializado que otimizará um tipo de cena antes não otimizado, eliminando a necessidade de compilar novamente a aplicação.

7.2.3. O Objeto RenderSystem

O RenderSystem é uma classe abstrata que define a interface entre o OGRE e a API (*Application Programming Interface*) gráfica utilizada. Ele é responsável por executar comandos para renderização e configurar opções próprias da API gráfica. Esta classe precisa ser abstrata porque todos os comandos realizados por ela dependem da API de renderização utilizada, fazendo-se necessária a presença de subclasses específicas para cada uma delas (D3D9RenderSystem, GLRenderSystem, etc.).

Após a inicialização do sistema, através do método `Root::initialise`, o objeto RenderSystem referente ao tipo escolhido na configuração pode ser acessado a partir do método `Root::getRenderSystem`.

Mesmo sendo um dos componentes mais importantes do *engine*, o RenderSystem não deve ser acessado diretamente. Tudo que for preciso para renderizar objetos e definir opções estará disponível a partir do SceneManager, do Material (ver Subseção 0) e de outras classes provenientes da cena. O acesso ao RenderSystem é somente justificado quando se deseja criar múltiplas janelas de renderização (janelas totalmente separadas, excluindo alguns casos como múltiplos *viewports* para dividir uma mesma janela, que podem ser criados através da classe RenderWindow) ou acessar outros recursos avançados presentes somente no RenderSystem.

O Objeto Entity

Uma entidade é uma instância de um objeto móvel na cena. Por exemplo, um carro, um personagem, um avião ou qualquer objeto relativamente pequeno e móvel na cena.

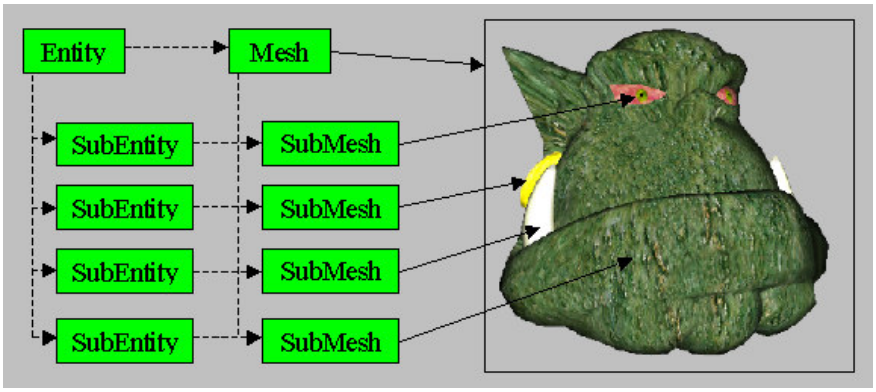


Figura 7.3 Relação entre Entity, SubEntity, Mesh, SubMesh e Material

Toda entidade possui uma malha associada, ou seja, uma geometria que descreve o objeto graficamente e que é representada pelo objeto Mesh (ilustrado na Figura7.3). Uma malha é composta por um conjunto de vértices e uma série de outras informações associadas a cada uma delas, tais como normais, mapeamento de coordenadas de textura e índices de posicionamento das faces.

Várias entidades podem referenciar o mesmo objeto Mesh, no caso de existir mais de uma cópia de um objeto na cena. Desta forma, mesmo com vários objetos do mesmo tipo sendo exibidos na cena, existirá apenas uma cópia do objeto Mesh para todos eles, otimizando o tempo de carregamento ou replicação das malhas e a memória utilizada pela aplicação.

As entidades são criadas através do método `SceneManager::createEntity`, fornecendo-se um nome para a entidade e o nome do arquivo `.mesh` que contém as informações da malha. O SceneManager não carregará diretamente o arquivo, pois tal operação será executada pelo MeshManager, que é responsável por carregar a malha apenas uma vez, mesmo que ele seja requisitado inúmeras vezes.

Para que uma entidade seja renderizada, é necessário associá-la a um SceneNode, que será descrito na próxima subseção. Quaisquer informações relativas à posição, à orientação e à escala não serão armazenadas diretamente na entidade, mas sim no SceneNode ao qual ela está associada.

Quando uma Mesh é inicializada, ela automaticamente carrega os materiais que foram definidos no arquivo `.mesh`. É possível se ter mais de um material associado a uma Mesh, sendo cada um deles associado a um subconjunto da malha (SubMesh). Normalmente uma Mesh é formada por várias SubMeshes e cada uma destas recebe originalmente um material diferente. Sendo assim, se uma Mesh possuir apenas um material, ela será composta por uma única SubMesh.

Uma entidade é composta por uma ou mais subentidades (SubEntity), dependendo apenas da quantidade de SubMeshes contidas na Mesh a qual ela referencia. Cada SubMesh terá uma SubEntity relacionada, que armazenará informações adicionais como o Material que pode ser modificado através do método `Material::setMaterialName` (detalhes na Subseção 0 e ver Figura7.3). Desta forma, pode-se modificar a entidade dando-se uma aparência diferente da Mesh que a originou.

7.2.4. O Objeto SceneNode

SceneNode é um objeto utilizado para agrupar entidades e armazenar informações de posição, orientação e escala destas.

Toda entidade, para tornar-se visível, deve estar associada a um SceneNode, que deve fazer parte da estrutura hierárquica que compõe a cena. Esta estrutura é composta inicialmente por um nó raiz, disponibilizado pelo SceneManager através do método `SceneManager::getRootSceneNode`, e deve ser expandida criando-se nós que serão filhos deste nó raiz. Cada nó pode ter mais de um filho, podendo assim compor um ambiente hierárquico qualquer. Nós, como câmeras e luzes, também podem fazer parte desta hierarquia. Estes, particularmente, não precisam estar associados a SceneNodes, uma vez que também armazenam informações de orientação e posicionamento.

Para criar uma hierarquia de nós deve-se utilizar o método `SceneNode::createChildSceneNode` que cria nós filhos.

Pode-se utilizar o SceneNode também para aplicar transformações espaciais às entidades associadas. Estas transformações são aplicadas hierarquicamente a todos os nós descendentes, facilitando a construção e animação de objetos hierárquicos.

Tais conjuntos de transformações podem ser traduzidos em operações, como translações e rotações, sendo estas últimas em re-

lação a um determinado eixo de coordenadas (*row*, *pitch*, *yaw*) ou a um eixo qualquer (utilizando quatérnios).

Outros comandos também estão disponíveis através do SceneNode. Entre eles estão o `setVisible`, que alterna a visibilidade do grupo de entidades associado ao nó, e o `showBoundingBox`, que mostra a caixa que delimita o volume das entidades.

Cada SceneNode pode ser identificado através de um nome, dado no momento de sua criação. Sendo assim, buscas podem ser realizadas utilizando como parâmetro o nome do SceneNode desejado, a partir do objeto SceneManager.

7.2.5. O Objeto Camera & Viewport

Camera representa um ponto de vista da cena, simbolizado por um nó com propriedades de Frustum. O conceito de Frustum está ligado a uma área restrita de visualização, que possui atributos como campo de vista (ângulo de abertura) ou FOV (*Field Of View*), aspecto (relação entre largura e altura) ou *aspect*, e os planos que delimitam o volume de visualização (*near plane* e *far plane*) [Foley et al. 2005].

A câmera possui ainda uma posição e uma orientação, podendo ser movimentada dinamicamente. Sua criação deve ser feita sempre através do SceneManager, o qual guarda referências para as câmeras e pode retorná-las a partir de um nome associado a elas no momento de sua criação.

Figura 7.4 Viewports diferentes em uma mesma janela

Uma câmera só deverá ser utilizada como ponto de vista se associada a um alvo de renderização (RenderTarget), como uma janela ou uma textura. No caso mais comum adiciona-se uma câmera a uma RenderWindow e cria-se uma área de visualização dentro da janela chamada Viewport. O Viewport tem informações como altura, largura, cor do fundo e posicionamento no alvo de renderização, que podem ser utilizadas para criar mecanismos como visualizações diferentes em uma mesma janela (recurso bastante utilizado em jogos *multiplayer* para *consoles* como visto na Figura 7.4).

A partir de Cameras e Viewports pode-se também criar efeitos de reflexão do ambiente utilizando outro tipo de RenderTarget, chamado RenderTexture, descrito na Seção 7.5.2.

O Objeto Material

O objeto Material altera a aparência dos objetos da cena e controla como os mesmos serão renderizados, o que reflete no modo como serão visualizados posteriormente. No objeto Material são especificadas quais as propriedades da superfície do material, como os componentes do modelo de iluminação (ambiente, difuso, especular e emissivo) [Foley et al. 2005].

Fazem parte também do Material as camadas de texturas associadas a ele, suas imagens correspondentes, e como elas devem interagir para formar a superfície final (*blending*, composição aditiva, composição modulativa, etc.). Além disso, alguns efeitos podem ser adicionados através do Material, como Mapeamento de Ambiente (*Environment Mapping*), filtros de textura, animações e modo de *culling* (se o material deve ser aplicado nas duas faces dos triângulos da malha). Os Materials são identificados unicamente por um nome e podem ser atribuídos a qualquer Entity ou SubEntity através dos métodos `Entity::setMaterialName` ou `SubEntity::setMaterialName`.

Um Material é composto de um conjunto de `techniques` e `passes` que podem ser alteradas através do código C++ ou carregadas em um *script* de configuração de material. Esta característica torna a edição e modificação do material independente do código fonte, podendo inclusive haver uma modificação sem que seja necessária uma recompilação. Os *scripts* e seus atributos e facilidades serão melhor explorados na Seção 7.4.

7.3. Geração de Malhas

Por padrão, o OGRE oferece suporte ao carregamento de malhas através dos arquivos `.mesh`. Estes arquivos podem ser facilmente exportados através de diversos *softwares* de modelagem 3D, como 3DS Max [3DS 2007], Maya [Maya 2007], Blender [Blender 2007], entre outros. Além dos arquivos `.mesh`, existem os `.xml.mesh` para armazenar dados da malha, sendo esses diferenciados dos primeiros apenas por não serem binários.

O processo de exportação das malhas é realizado em dois passos. Primeiramente um arquivo `.xml.mesh` é gerado pela ferramenta de modelagem 3D. Este arquivo, por não ser binário, pode ser lido, entendido e editado diretamente em qualquer editor de texto, possibilitando assim que pequenas alterações possam ser realizadas diretamente no mesmo. Em seguida, o `.xml.mesh` deve ser traduzido para um `.mesh`, sendo este último o arquivo final que será carregado pelo *engine*. Caso o modelo tenha sido exportado pelo 3DS Max este passo será abstraído, pois a ferramenta o executa implicitamente (o 3DS chama automaticamente o tradutor).

A tradução dos arquivos `.xml.mesh` para os `.mesh` é feita pela ferramenta OgreXMLConverter. Esta ferramenta não possui uma interface gráfica, logo todas as funções devem ser chamadas através de linhas de comando. Ela está disponível para *download* gratuito no *site* oficial do OGRE. Outra ferramenta disponibilizada

no *site* é o OgreMeshViewer, que possibilita que os arquivos sejam visualizados antes de serem carregados pelo *engine*.

7.4. Scripts

Scripts são simples arquivos de texto que podem ser criados e alterados em qualquer editor de texto, e que têm o propósito de agilizar o processo de desenvolvimento. Eles servem como um caminho alternativo ao código fonte para criação de alguns recursos suportados pelo *engine*. Os *scripts* suportados pelo OGRE são: Material Scripts, Particle Scripts, Overlay Scripts e Font Definition Scripts. Nesta seção serão brevemente apresentados os *scripts* de inicialização e os Material Scripts; maiores informações sobre os demais *scripts* suportados podem ser encontradas em [Farias et al. 2006]

Os *scripts*, além de tornarem o código fonte da aplicação mais enxuto, também descartam a necessidade de recompilação do código quando houver a necessidade de alteração de parâmetros dos recursos. Esta característica torna o processo de experimentação mais rápido e menos penoso. Outra característica positiva dos *scripts* é que eles podem ser reusados facilmente por outras aplicações, já que são independentes do código fonte.

7.4.1. Inicialização

Fazem parte dos *scripts* de inicialização os arquivos `plugins.cfg`, `ogre.cfg` e `resources.cfg`. Eles são responsáveis, respectivamente, pelo carregamento dos *plugins*, preenchimento automático das opções de configuração e definição de caminhos para os recursos utilizados pela aplicação. Este último é utilizado obrigatoriamente apenas para aplicações que derivam da classe `ExampleApplication`, fornecida pelo modelo de aplicação do OGRE [Farias et al. 2006].

Cada arquivo tem seu formato específico. No caso do arquivo `plugins.cfg` existe uma configuração chamada `PluginFolder`, que aponta para a pasta onde estarão localizados todos os *plugins* listados no arquivo. As demais configurações são do tipo `Plugin` e devem ser seguidas de nomes de *plugins* a serem carregados (os nomes das bibliotecas sem a extensão `dll`). O arquivo `resources.cfg` é formado de diretivas `Zip` e `FileSystem`, que apontam respectivamente para arquivos compactados no formato zip e para pastas onde o OGRE procurará por mais arquivos de *scripts* e recursos utilizados na aplicação. O arquivo `ogre.cfg` é dispensável, pois é gerado sempre que a tela de configurações exibida na inicialização da aplicação é fechada. Ele contém os últimos valores configurados pelo usuário.

7.4.2. Materiais

Material Scripts possibilitam a definição de materiais complexos de forma fácil e rápida, os quais podem ser reusados posteriormente. Na definição de materiais através de *scripts*, o desenvolvedor pode escrever seus *scripts* e carregá-los quando necessário.

Arquivos contendo *scripts* de materiais geralmente são identificados pela extensão `.material`. Esses arquivos são analisados (*parsed*) automaticamente no momento da inicialização da aplicação. Por padrão, a aplicação busca em todos os diretórios de recursos comuns pelos arquivos `.material`, e analisa a estrutura de cada *script*. Ao final da análise, os *scripts* que possuírem erro na sintaxe não serão carregados, e os objetos que utilizem este material aparecerão brancos.

É possível definir inúmeros materiais em um único arquivo `.material`. Essa é uma opção bastante utilizada quando se deseja agrupar materiais relativos a uma única entidade da cena. Por exemplo, todos os materiais aplicados em um personagem de um

jogo podem ser agrupados em um único arquivo, facilitando a busca dos mesmos caso seja necessária alguma alteração.

A estrutura de um *script* de material é relativamente simples; ela segue uma estrutura hierárquica de seções. As seções de um *script* de material são: `material`, `technique`, `pass` e `texture_unit`. Cada seção possui um conjunto específico de atributos e, caso um atributo não apareça na sua respectiva seção, a este será atribuído um valor padrão.

O valor padrão dos atributos pode variar de atributo para atributo; por exemplo, o valor padrão do atributo `ambient` é `1.0 1.0 1.0 1.0`, enquanto o do atributo `cull_hardware` é `clockwise`. A descrição dos atributos e seu valor padrão para todas as seções de um *script* de material pode ser vista em [Farias et al. 2006].

O identificador de um material é definido na declaração da seção `material`. Todo material deve ser identificado por uma *string* globalmente única e, caso contrário, a aplicação será finalizada devido ao conflito entre nomes. Os identificadores podem conter caracteres do tipo '/', de modo que a noção de árvore possa ser expressa.

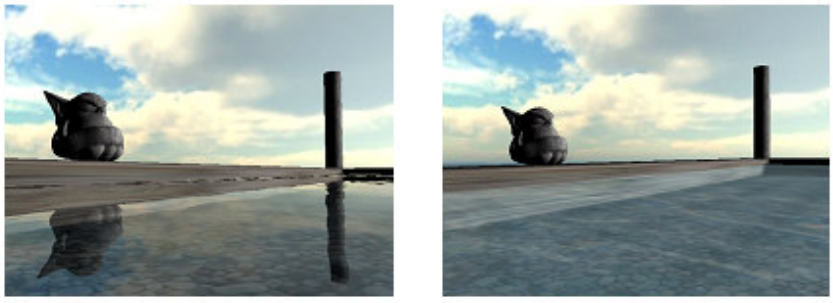


Figura 7.5. Cena com efeitos de reflexão e refração, em dois LODs

As técnicas são definidas através da palavra `technique` e diferentemente de um `material`, não requerem identificador.

Uma `technique` é usada quando se deseja criar um efeito específico (ver Figura 7.5); desta forma, um `material` pode ser composto por várias `techniques`. `Materials` com múltiplas `techniques` são bastante úteis nos casos em que a aplicação será executada em diferentes plataformas e configurações de *hardware*.

O *engine* escolherá (em tempo de execução) uma das `techniques` definidas no *script*, dando maior prioridade as que forem primeiramente declaradas. Assim, é recomendado que se disponha ordenadamente as `techniques` das mais sofisticadas as menos sofisticadas. Múltiplas `techniques` podem ser utilizadas quando se deseja um efeito menos preciso em objetos que, por exemplo, estejam distantes do observador (*Level Of Detail* - LOD).

A seção `pass` representa um passo da renderização (uma chamada a API). O custo computacional de uma `technique` é diretamente proporcional à quantidade de passos. `texture_unit` é usada para definir os atributos relativos a textura.

7.5. Facilidades do *Engine*

Geração de sombras, Render-to-Texture e Mouse Picking são facilidades oferecidas pelo OGRE que possibilitam a implementação de aplicações com características que as tornam mais agradáveis para o usuário, tanto do ponto de vista visual quanto interativo.

7.5.1. Geração de Sombras

Geração de sombras é um aspecto importante na busca pelo realismo de uma cena, pois ajuda na percepção da distância entre os objetos. Existem diversas formas de gerar sombras, cada uma com suas vantagens e desvantagens. De uma forma geral, quanto mais próxima da realidade for a simulação das sombras, mais ela será

computacionalmente custosa. O OGRE oferece várias implementações de sombra, de modo que o desenvolvedor possa escolher qual implementação é a mais apropriada para a sua aplicação. O OGRE suporta três tipos de cálculos de sombra; a Figura 7.6 ilustra seus resultados:

- *Texture Modulative* – é a menos custosa das três, porém não produz efeitos tão bons. Ela funciona criando um Render-to-Texture (ver detalhes na próxima subseção), que gera os mapas de textura aplicados à cena simulando as sombras;

- *Stencil Modulative* – não é tão precisa quanto a *Stencil Additive*, mas é mais rápida. Ela funciona renderizando os volumes das sombras após a renderização dos objetos não transparentes;

- *Stencil Additive* – esta técnica renderiza cada luz como um passo aditivo separado na cena. É a mais custosa das três, pois para cada luz adicional ela requer um novo passo de renderização.

Para se ter sombras em uma cena, primeiramente o cálculo das mesmas deve ser habilitado através do método `SceneManager::setShadowTechnique`. É importante que este passo seja executado antes de qualquer outro, pois a técnica escolhida pode alterar a forma como as malhas são carregadas. Em seguida, uma ou mais luzes devem ser criadas, pois sem elas as sombras não podem ser geradas. Após isso, desabilita-se a projeção de sombra nos objetos que não devem projetá-las. Este passo é realizado pelo método `Entity::setCastShadows`, que recebe como parâmetro um valor booleano. E, por final, define-se quais objetos receberão ou não sombras. Tal definição (diferentemente da anterior) é realizada em cada Material, podendo ser feita através do *script* (no atri-

buto `receive_shadows`), ou diretamente no código (`Material::setReceiveShadows`).

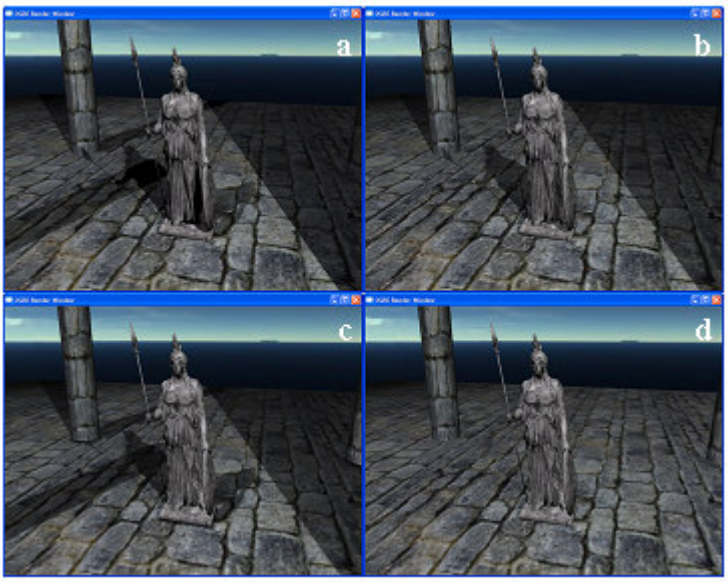


Figura 7.6. Geração de sombras: (a) *Stencil Additive*; (b) *Stencil Modulative*; (c) *Texture Modulative*; (d) sem sombra

7.5.2. Render-to-Texture

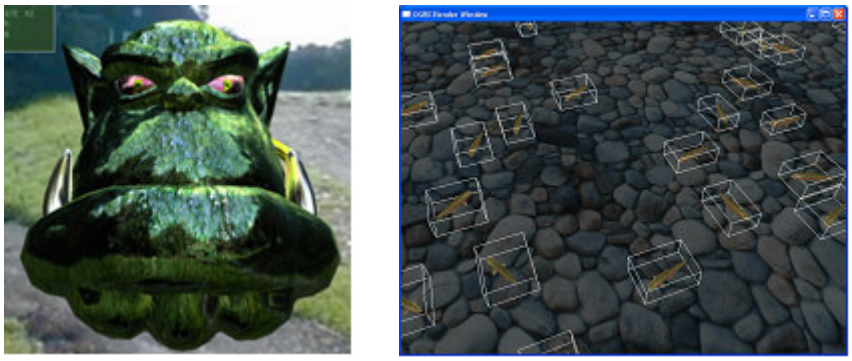
Render-to-Texture é uma técnica para renderizar cenas em texturas que são utilizadas em materiais para torná-los reflexivos. Ela pode ser usada, por exemplo, para compor o material da carroceria de um carro, que tem uma cor base e também reflete o ambiente.

Para utilizar esta técnica no OGRE tem-se que entender o conceito de RenderTargets. Estes servem como "alvos" de renderização para uma determinada câmera, que é associada a um *Viewport* (porção do alvo que será utilizada para renderização) deste alvo. Desta forma, pode-se classificar como RenderTargets os objetos

RenderWindow e RenderTexture, sendo o último o alvo utilizado na renderização para uma textura.

Os passos necessários para aplicar um material que contenha uma RenderTexture são os seguintes:

1. Criação do *script* de material;
2. Adição do nome da RenderTexture como `texture_unit` de um `material`;
3. Criação (no código) de um RenderTexture com o mesmo nome utilizado no *script* de material;
4. Adição de um *Viewport* através de uma câmera que vai definir que parte da cena será renderizada no RenderTexture;
5. Aplicação do material em alguma entidade.



a **b**

Figura 7.7. a) Material reflexivo criado com Render-to-Texture; b) AxisAlignedBoxes das entidades da cena para Mouse Picking

Pode-se, a partir da seqüência de comandos descrita acima, obter resultados significativos, do ponto de vista de mapeamento do

ambiente em objetos reflexivos, como ilustrado na Figura 7.7a, onde percebe-se a reflexão do ambiente (grama, árvores e céu) na cabeça do ogro.

7.5.3. Mouse Picking

Uma operação bastante comum em aplicativos 3D é a seleção de objetos da cena utilizando o *mouse*, a qual é denominada Picking. Tal rotina é suportada nativamente pelo OGRE, de modo que aplicações (principalmente editores 3D) que necessitem desse tipo de requisito possam ser desenvolvidas.

O Picking é realizado através do objeto RaySceneQuery. Tal objeto é criado através do método `SceneManager::createRaySceneQuery` recebendo como parâmetro um raio (*ray*). Este raio pode ser inicialmente nulo, pois ele poderá ser redefinido posteriormente. Após a definição do raio, o RaySceneQuery deverá ser executado, retornando assim o conjunto de objetos interceptados pelo mesmo. Este raio é usualmente computado utilizando a função `getCameraToViewportRay` do objeto Camera, que recebe como parâmetro a posição relativa do *mouse* à janela.

O Picking padrão implementado no OGRE detecta os objetos com base no AxisAlignedBox dos objetos, o que faz com que os resultados sejam imprecisos, porém rápidos. Um AxisAlignedBox é um tipo de BoudingBox que sempre está alinhado com o eixo de coordenadas do mundo; assim, mesmo que os objetos sejam rotacionados seus BoudingBoxes continuarão alinhados. Se a aplicação requisitar uma forma mais acurada de Picking, esta deverá ser implementada pelo desenvolvedor. A implementação padrão do Picking no OGRE, por ser mais rápida, pode ser utilizada como um filtro inicial, diminuindo assim a área de busca na qual um algoritmo mais complexo atuará. A Figura 7.7b mostra os AxisAlignedBo-

xes (observe que eles sempre estão alinhados com o mundo) dos peixes nadando.

7.6. Considerações Finais

A utilização de *engines* de renderização genéricos, *open-source*, multiplataforma, com uma vasta gama de ferramentas compatíveis para o desenvolvimento de vários tipos de aplicações gráficas e com suporte a programação de técnicas alternativas de acordo com a infra-estrutura de *hardware* disponível e os requisitos da aplicação, que funcionam em várias configurações de GPUs, que oferecem *wrappers* para outras plataformas de *software*, para desenvolvimento de aplicações gráficas e jogos está se tornando cada vez mais comum. Isto se deve as funções sofisticadas que os mesmos oferecem para o desenvolvedor, que também usufrui das facilidades de implementação que ferramentas que utilizam o paradigma de orientação a objetos oferecem. O OGRE, apresentado neste capítulo, é um destes *engines*.

Referências

(2007) "OGRE", http://www.ogre3d.org, Fevereiro.

(2007) "LGPL", http://www.gnu.org/copyleft/lgpl.html, Fevereiro.

Farias, T., Silva, D., Teichrieb, V. e Kelner, J. (2006) "Minicurso SBGames 2006: O Engine Gráfico OGRE", http://www.cin.ufpe.br/~grvm, Fevereiro.

(2007) "OpenGL", http://www.opengl.org, Fevereiro.

(2007) "Microsoft DirectX", http://www.microsoft.com/windows/directx/default.mspx, Fevereiro.

Foley, J., Van Dam, A., Feiner, S. e Hughes, J., Computer Graphics: Principles and Practice, Addison Wesley, 2005.

(2007) "Autodesk 3ds Max", http://usa.autodesk.com/adsk/servlet /index?id=5659302&siteID=123112, Fevereiro.

(2007) "Autodesk Maya", http://usa.autodesk.com/adsk/servlet/ index?siteID=123112&id=7635018, Fevereiro.

(2007) "Blender", http://www.blender.org/, Fevereiro.

Capítulo 8

Panda3D

Roberto Cezar Bianchini[1], Alexandre Tomoyose[1], João Bernardes Jr[1]

[1]Laboratório de Tecnologias Interativas - Escola Politécnica – Universidade de São Paulo (USP)
Caixa Postal 05508-900 – São Paulo – SP – Brazil

roberto-bianchini@hotmail.com, alextomoyose@gmail.com, joao_l_Bernardes@yahoo.com.br

Abstract

Panda3D is a free game engine written in C++ with Python biding. It was developed by Walt Disney Company to create its Massive Multiplayer Online Game, Toontown, and now it is developed in association with Carnegie Mellon University's Entertainment Technology Center. The game developer writes a program in Python that acesses the engine in C++. This chapter presents the main features of Panda3D, its modules, the rendering pipeline, the hierarchy of classes and a small tutorial of how to create a simple 3D game with Panda3D.

Resumo

Panda3D é um engine de jogos gratuito escrito em linguagem C++ com binding *para a linguagem Python. Foi desenvolvido inicialmente pela Disney para a criação de seu jogo massivo multiplayer*

Toontown*, e atualmente é desenvolvida pela parceria Disney-Carnegie Mellon University's Entertainment Technology Center. O desenvolvedor escreve um programa em Python que acessa e controla os recursos do engine. Este capítulo apresenta as características do Panda3D, seus módulos principais, a* pipeline *de renderização, hierarquia de classes, ferramentas existentes para facilitar o desenvolvimento, e um pequeno tutorial que mostra a criação de um jogo 3D simples a partir do Panda3D.*

8.1. Introdução

Em meados da década de 1990, em um processo de mutualismo entre as empresas que desenvolvem jogos e a indústria de hardware gráfico, foram surgindo jogos cada vez mais complexos e visualmente realistas que demandavam um grande processamento gráfico. O surgimento destes jogos acabou gerando um grande número de adeptos e fãs com desejo de modificar os seus jogos favoritos ou criarem outros inteiramente novos, muitas vezes apenas como *hobby*. Neste sentido, algumas empresas produtoras de jogos liberavam parte ou a totalidade do software desenvolvido para a criação do jogo para que os fãs pudessem usá-lo da melhor forma que encontrassem. Este software para o desenvolvimento de jogos ficou conhecido como *Game Engine* ou simplesmente *Engine*, e este capítulo trata de um destes engines: Panda3D.

Panda3D [ETC 2007] é um anagrama e quer dizer: *Plataform Agnostic Networked Display Architecture*. Panda3D foi desenvolvido pela *Walt Disney Company* para a criação do seu jogo massivo on line – ou simplesmente MMO da sigla em inglês – *ToonTown* (ver figura 8.1), e foi liberado como software livre em 2002. Atualmente é desenvolvido em conjunto pela Disney e pelo *Entertainment Technology Center* da *Carnegie Mellon University*. Este é um engine 3D: um conjunto de subrotinas para renderização de cenas 3D e desenvolvimento de jogos. As subrotinas são escritas

em linguagem C++. Seguindo a tendência da maioria dos engines [DALMAU 2003], Panda3D proporciona acesso às subrotinas através de uma linguagem de script: Python. Desenvolver um jogo com Panda3D consiste em escrever um programa em Python que controla as subrotinas em C++. Como o engine está sob a licença de software livre, também é possível modificar as subrotinas diretamente e criar *binding* para uma outra linguagem de script, como Lua ou Pearl, mas este capítulo ficará restrito à programação no Panda3D utilizando Python apenas.

Figure 8.1. ToonTown, MMO desenvolvido com o Panda3D.

Uma das principais características do Panda3D foi o seu desenvolvimento tendo como principal requisito uma curva de aprendizado curta e rápida. Projetos de jogos com prazos curtos podem se beneficiar da utilização deste engine. Outras características do Panda3D incluem:

- Possibilidade de uso das principais APIs gráficas: OpenGL/DirectX

- Interface de programação script via linguagem Python e programação em tempo de execução utilizando um shell interativo em Python.

- Formato próprio de modelos 3D, Egg, com exportadores para 3D Studio Max e Maya.

- Animação por soft skin interface de animação de personagens.

- Som via biblioteca FMOD.

- Módulos para rastreamento magnético para aplicações em Realidade Virtual.

As próximas seções apresentam detalhes mais técnicos da Panda3D e um estudo de caso da criação de um jogo simples utilizando o engine.

8.2. Programando com o Panda3D

O processo de instalação, programação e utilização do engine são discutidos em maiores detalhes nesta seção. Assume-se que o leitor possui o mínimo de conhecimentos da linguagem Python para a compreensão desta seção.

8.2.1. Instalando e Configurando o Panda3D

Panda3D é um engine multiplataforma, podendo ser instalado tanto em PCs com o sistema MS-Windows quanto em PCs com GNU-Linux. A sua instalação requer que a biblioteca Python [PYTHON, 2007]. Em sistemas MS-Windows é só baixar um arquivo auto instalável do site oficial; em sistemas GNU-Linux, existe a possibilidade de instalação via arquivos RPM ou DEB, caso eles existam para a versão do sistema, ou pode-se baixar o código fonte do site oficial e compilá-lo para a sua versão.

Uma vez instalado, Panda3D vem com um “Greeting Card” para testar se tudo ocorreu bem. Na instalação em MS-Windows, o

instalador executa o teste automaticamente. Em GNU-Linux, é necessário mudar para o diretório `samples/GreetingCard` e executar o comando `ppython GreetingCard.py`. Teste manual também funciona em MS-Windows.

Panda3D utiliza o padrão Unix para descrição de caminhos no sistema de arquivos, mesmo na plataforma MS-Windows. O separador de diretórios é a barra, /, e não existe uma letra de prefixo. Por exemplo, para se carregar um arquivo contendo um modelo 3D no padrão aceito pelo Panda3D, .egg, armazenado na plataforma MS-Windows no caminho "`c:\Projetos\model.egg`", o correto seria utilizar o caminho "`/c/Projetos/model.egg`". A classe `Filename` possui um método, `fromOsSpecific(string)` que converte de um dado padrão de caminho para o utilizado no Panda3D.

Ainda em configuração, no subdiretório etc da instalação do Panda3D, existe um arquivo chamado `Config.prc` que permite personalizar uma boa parte das variáveis que definem as características de renderização, tanto utilizando OpenGL quanto DirectX. Características como renderização em tela cheia, esconder o cursor, exibição do número de frames por segundo, entre outras, podem ser definidas neste arquivo. Durante a execução, uma lista completa das variáveis pode ser conseguida com o seguinte comando Python: `cvMgr.listVariables()`. É necessário importar o módulo `DirectStart` antes de invocar este método.

8.2.2. Principais Módulos

Panda3D funciona com um Grafo de Cena. Objetos só são visíveis após serem inseridos no grafo de cena. No Panda3D o grafo é uma árvore com objetos do tipo `PandaNode`, como `ModelNode`, `GeomNode`, `LightNode`, entre outros. A raiz do grafo de cena é um nó especial chamado `render`. Podem existir nós a-

dicionais como raízes para alguns propósitos específicos. A parte de renderização será explicada em maiores detalhes na seção 8.2.3.

Um dos principais módulos do Panda3D é de `Actor` e `Characters`. Eles definem como os avatáres controlados tanto por jogadores quanto pelo jogo se comportam, como animação por esqueleto e por morph. `Actor` define a interface de alto nível para a programação das estruturas de baixo nível definidas na classe `Character`.

A funcionalidade de câmera no Panda3D é feita pelo módulo `ShowBase` que possui uma classe `Camera` derivada de `PandaNode`. Por default, Panda3D executa uma tarefa que permite controlar a câmera com o mouse. Para escrever código que controla a câmera é necessário desabilitar esta tarefa primeiro. A classe Camera define visualização em perspectiva e ortogonal, permitindo controlar todos os parâmetros destas visualizações como ângulo de abertura, plano próximo e distante, entre outros.

Tarefas, no módulo `task`, são subrotinas escritas pelo usuário que são chamadas a cada frame pelo Panda. Eventos são as subrotinas chamadas pelo engine quando certas condições especiais são satisfeitas. Tarefas podem ser adicionadas para chamada pelo engine pela função `taskMgr.add(subrotina, "Nome")`, e removidas com o método `taskMgr.remove( "Nome")`. É possível também configurar uma tarefa para executar após um intervalo de tempo com o método `taskMgr.doMethodLater()`. O mecanismo de eventos no Panda3D é tanto complexo. Envolve criar uma classe derivada de `DirectObject` do módulo showbase, e registrar esta classe em um mecanismo chamado "Messenger". Uma explicação mais detalhada sobre o mecanismo foge do escopo deste capítulo.

Panda3D também possui um mecanismo de colisão por bounding box, como esferas, e por geometrias variadas. Existe um

módulo de física bem simples que permite aplicar forças e torques nos objetos do jogo, assim como mudar alguns parâmetros que influenciam no movimento como a viscosidade. Um sistema simples de simulação de partículas também existe no módulo de partículas.

Som no Panda3D é feito com a biblioteca comercial FMOD [FMOD 2007]. Para aplicações não comerciais, a biblioteca pode ser utilizada sob uma licença gratuita. Caso o usuário do Panda3D não deseje utilizar a biblioteca, basta remover `fmod.dll` e `libfmod_audio.dll` do diretório /bin da instalação do Panda. É possível utilizar sons no Panda3D através do

8.2.3. Processo de Renderização no Panda3D

O processo de renderização do Panda3D é bastante flexível e utiliza o conceito de grafo de cena, por isso é interessante discutilo com mais detalhe nessa seção, não só para que o seu grafo de cena seja propriamente compreendido, mas também para que se possa tirar vantagem de toda essa flexibilidade.

Um grafo de cena é uma forma de organizar hierarquicamente os objetos de uma cena em uma estrutura de árvore direcionada (um grafo conectado direcionado acíclico). Os elementos desse grafo são chamados nós. Um nó imediatamente superior a outro na hierarquia e conectado a ele é chamado de pai. Embora um pai possa ter vários filhos, cada nó pode ter somente um pai. O nó mais alto na hierarquia, de qual todos os outros nós são descendentes, é a raiz do grafo. Nós sem descendentes são chamados de folhas.

No Panda 3D, a raiz de um grafo de cena indica sua função. Dois grafos, por exemplo, são criados por *default* pelo *engine* com raízes chamadas `render`, o mais usado, e `render2d`. Esses grafos são responsáveis pela renderização de elementos 3D e 2D (normalmente elementos de interface) da cena, respectivamente.

Além disso, tanto as transformações geométricas aplicadas sobre um nó, que definem sua posição e orientação, como as propriedades de renderização (como cor, efeito de *fog* ou planos de *clipping*) são aplicadas também a todos os seus filhos. Dessa forma, se a cabeça de um modelo deve sempre estar X unidades acima de seu pescoço, não importa o quanto ele se desloque, a cabeça pode ser incluída como um nó filho do pescoço na posição (0, 0, X). E se foi definida uma cor para a pele do pescoço, a cabeça terá a mesma cor por *default*. As propriedades de renderização podem ser modificadas em cada filho, se necessário.

Todas as funções que criam nós no Panda 3D retornam um objeto da classe `NodePath`. Esse objeto é um *handle* para o nó e não um ponteiro para ele. A partir de um NodePath é possível obter um ponteiro para o nó usando o método `node()`, mas a partir de um nó não se obtém um `NodePath`. Como ambos são necessários (por exemplo como parâmetros de funções), os NodePaths é que devem ser armazenados.

Para estabelecer o pai de um nó usa-se o método `reparentTo`, que recebe o novo pai como parâmetro. Essa função pode ser usada em qualquer momento para alterar o pai de um nó no grafo de cena (consequentemente alterando todo o galho da árvore abaixo daquele nó) e deve ser usada ao menos uma vez depois que o nó é criado para conectá-lo ao grafo de cena (por exemplo, passando o objeto render como pai). Como os nós herdam as transformações de seus pais, pode ser conveniente usar o método `wrtReparentTo` para modificar o pai de um nó de forma que sua posição e orientação sejam recalculados em relação ao novo pai para que o objeto permaneça na mesma posição global. Outra forma de fornecer um pai a um nó, já em sua criação, é usar o método `attachNewNode`, de `NodePath`.

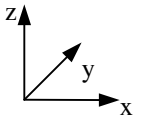


Figura 8.2. Sistema de Coordenadas

A figura 8.2 mostra o sistema de coordenadas do Panda 3D. As transformações geométricas sobre um nó (sempre em relação a seu pai) podem ser estabelecida utilizando métodos de `NodePath`. `setPos(X, Y, Z)` e `setScale(S)` ou `setScale(SX, SY, SZ)` são auto-explicativas. Se for passado para o método `setPos` um *handle* para um nó antes das 3 coordenadas, o posicionamento será feito em relação a esse nó. Para controlar a orientação, pode-se usar o método `setHpr(H, P, R)` que recebe como parâmetro os ângulos de Euler (Heading ou Yaw, Pitch e Roll) ou `setQuat`, que especifica a rotação como um quaternio. O método `lookAt(refNode)` faz o eixo Y do nó apontar para refNode. Existem também métodos que modificam cada componente da posição ou rotação individualmente ou para fazer todas as transformações de uma só vez. Uma forma de controlar automaticamente a orientação de um nó é fazê-lo um *billboard*, ou seja, fazer com que sempre aponte para um objeto (normalmente a câmera, nesse caso usa-se o método `setBillboardPointEye`). Para modificar qualquer um dos parâmetros de renderização de um nó (como cor, fog ou textura) usa-se o método `setAttrib`, um método de `PandaNode`, não de `NodePath`.

O Panda3D utiliza quatro tipos de fonte de luz; pontual, direcional, *spot* e ambiente; que nada mais são que nós do grafo de cena. Esses nós são construídos, respectivamente, com as funções `PointLight`, `DirectionalLight`, `SpotLight` e `AmbientLight`.. O método `setColor` modifica a cor da luz. Luzes *spot* podem ter também como parâmetro um objetoO nó a ser iluminado

por uma luz qualquer (todos os seus filhos também o serão), deve usar o método `setLight` passando como parâmetro essa luz. Para iluminar a cena inteira com a luz `light`, por exemplo, usa-se `render.setLight(light)`. Para que uma luz não ilumine mais um nó, existe o método `clearLight`. De forma semelhante, pode-se usar as funções `setFog` e `clearFog` para aplicar efeitos de *fog* a nós específicos ou à cena toda. *Fogs* também tem com, manipulada com `setColor`, e tem densidade definida pelo método `setExpDensity`.

As principais classes que controlam o processo de renderização do Panda3D são `GraphicsPipe`, `GraphisEngine`, `GraphicsStateGaurdian` e `GraphicsOutput`. Normalmente não é necessário utilizar essas interfaces em aplicações simples, já que uma janela *default* é criada no começo da aplicação e essas classes são instanciados no objeto `base`. `GraphicsPipe` faz a interface do Panda3D com APIs 3D (DirectX ou OpenGL). `GraphicsEngine` é o coração do processo de renderização, responsável por todas as operações de *culling* e desenho, seja para janelas na tela ou para buffers na memória (objetos `GraphicsWindow` ou `GraphicsBuffer`, respectivamente, ambos derivados da classe abstrata `GraphicsOutput`). `GraphicsEngine` pode ser usado para criar esses objetos, mas existem formas mais simples de criar uma janela ou buffer para renderização: o método `openWindow` de base e o `makeTextureBuffer` de window. GraphicsStateGuardian equivale a um contexto de OpenGL, ou seja, contém todos os objetos a serem renderizados. Objetos `GraphicsBuffer` podem ser associados a diferentes câmeras usados para renderização em múltiplas passadas. Cada câmera, nesse caso, pode inclusive modificar os parâmetros de renderização de objetos da cena através dos métodos `setInitialState` ou `setTagStateKey` e `setTagState`.

Embora isso normalmente não seja necessário, o Panda3D permite que se estabeleça uma ordem para a renderização dos objetos de uma cena. Isso é útil para otimizar a renderização, mandando sequências de objetos com estados (como cor ou textura) semelhantes para o hardware gráfico para minimizar as mudanças de seu estado ou mandando os objetos mais próximos primeiro para que não seja necessário fazer cálculos para pixels que estão oclusos. Outra utilidade de ordenar a renderização dos objetos é para mostrar objetos transparentes. No Panda3D, essa ordenação se dá colocando os objetos nos *bins* apropriados. Os objetos são associados por *default* aos *bins* "opaque" (que agrupa os objetos por estado) ou "transparent" (que renderiza do mais próximo para o mais distante), dependendo de sua transparência. O método `setBin` de `NodePath` pode ser usado para colocar os objetos em outros *bins*, como o "background", o "unsorted" e o "fixed" (nesse caso a ordem de renderização dos objetos deve ser fornecida). Outros *bins* podem inclusive ser criados e associados a diferentes formas de ordenação através do método `add_bin` de `CullBinManager`.

8.2.4. Ferramentas do Panda3D

Algumas ferramentas interessantes acompanham o Panda3D:

- O *Scene Graph Browser*, ou navegador de grafo de cena, permite a visualização do grafo que tem `render` como raíz de uma forma similar a como a árvore de diretórios é visualizada no gerenciador de arquivos do MS Windows. Cada nó pode ser manipulado interativamente com um clique do botão direito, por exemplo permitindo que se mude seu pai ou sua posição.
- O *Scene Editor* (sceneEditor.py) ou editor de cena permite que uma cena seja montada visualmente, permitindo a importação e manipulação de objetos e animações; configura-

ção da iluminação, de sistemas de partículas e de detecção de colisão e geração de caminhos para movimentação de objetos.

- Pview, um visualizador de modelos .egg ou .bam. O modelo a ser visualizado é especificado na linha de comando. Permite rotação e translação interativas do modelo, sua visualização como wireframe e com ou sem texturas ou iluminação.

Além disso, diversos plugins ou conversores de linha de comando acompanham o Panda 3D para que modelos feitos em programas como 3D Studio, Maya, Blender e Milkshape possam ser exportados ou convertidos no formato .egg. O uso do formato .x do DirectX como formato intermediário para conversão é comum.

8.3. Exemplo de um jogo no Panda3D

Um exemplo clássico de jogo utilizando o Panda3D é o Airblade [ETC 2007]. Este jogo foi desenvolvido por alunos do *Entertainment Technology Center da Carnegie Mellon* e sempre é atualizado para versões recém lançadas do Panda3D.

Devido à complexidade do Airblade, neste tutorial será apresentada uma aplicação mais simples, utilizando os recursos básicos do Panda3D que foram apresentados nos tópicos anteriores. Para maiores detalhes sobre o Airblade, sugere-se ao leitor a versão completa na página oficial do engine.

8.3.1. Descrição do Jogo

O jogo proposto é um trecho do projeto de formatura desenvolvido por alunos da Escola Politécnica da Universidade de São Paulo (figura 8.3). O projeto consistia no desenvolvimento de uma engine de mais alto nível integrada ao Panda, com o intuito de facilitar a criação de jogos do tipo RPG (Role-Playing Game) baseados em gráficos 3D.

Figure 8.3. Jogo desenvolvido com o Panda3D para projeto de formatura.

O trabalho original visava à utilização como uma plataforma para facilitar o desenvolvimento de jogos, por causa disso, diversos parâmetros podiam ser configurados externamente através de arquivos XML, aumentando a flexibilidade do sistema. Devido ao espaço reduzido disponível para este tutorial, práticas mais elaboradas como esta não serão abordadas, promovendo uma implementação sucinta e objetiva.

Serão apresentados os passos básicos para a criação de um jogo, ou outro tipo de aplicação. Os elementos chaves são a criação do ambiente no qual será inserido o avatar virtual, o posicionamento do ponto de vista da câmera e a interação do usuário com o mundo virtual através de teclado e mouse.

8.3.2. Implementação no Panda3D

Na implementação serão utilizados alguns recursos básicos, como modelos e sons. Os recursos necessários são listados em cada etapa.

Para executar cada etapa, copie o código para um arquivo com extensão .py, por exemplo tutorial.py, e realize o comando: ppython tutorial.py.

Inicialização

Para iniciar uma instância do Panda são necessárias duas linhas de código:

```
import direct.directbase.DirectStart

run()
```

A primeira linha carrega os principais módulos do Panda3D e a segunda inicia a simulação.

No decorrer do tutorial outros módulos e classes do Panda3D serão carregados, mais informações podem ser encontradas na sessão Documentation do site do Panda.

Tela Inicial

Recursos: Imagem de dimensões 2^n x 2^n pixels

Como primeiro passo, vamos exibir uma imagem de apresentação para o usuário. Quando o usuário clicar na imagem, esta será removida da cena.

```
import direct.directbase.DirectStart
from pandac.PandaModules import Vec3,Vec4
from direct.gui.DirectButton import DirectButton
from direct.showbase.DirectObject import DirectObject

class Tutorial(DirectObject):
    def __init__(self):
        self.button = DirectButton(
                    image = "models/imagem.jpg",
                    frameColor = (1, 1, 1, 0),
                    rolloverSound = None, clickSound = None,
                    command = self.inicia)

    def inicia(self):
        self.button.destroy()

tut = Tutorial()
run()
```

Nesta etapa definimos a classe Tutorial e inserimos em seu construtor uma instância de DirectButton que, quando clicado, dispara o método `self.inicia()`.

As linhas from X import Y significam importar a classe Y que se encontra dentro do módulo X.

Carregando um ambiente

Recursos: Modelo do ambiente no formato .egg e arquivo de música no formato MP3

Neste passo, vamos carregar o modelo do ambiente e uma música de fundo, além de acrescentar algumas luzes à cena.

Para carregar o ambiente após o usuário clicar no botão, adicionamos os seguintes `imports`:

```
from pandac.PandaModules import LightAttrib
from pandac.PandaModules import AmbientLight
from pandac.PandaModules import DirectionalLight
from direct.interval.SoundInterval import SoundInterval
```

E modificações à classe `Tutorial`:

```
    def inicia(self):
        self.button.destroy()
        self.carregaLuz()
        self.carregaAmbiente()
        self.carregaSom()

    def carregaAmbiente(self):
        base.setBackgroundColor(Vec3(0.2, 0.2, 0.6))
        ambiente = loader.loadModel("models/ambiente.egg")
        ambiente.setScale(0.5,0.5,0.5)
        ambiente.setPos(0,77,0)
        ambiente.reparentTo(render)

    def carregaLuz(self):
        lightAttributes = LightAttrib.makeAllOff()
        ambientLight = AmbientLight("ambientLight")
        ambientLight.setColor(Vec4(0.6, 0.6, 0.6, 1 ))
        lightAttributes = lightAttributes.addLight(ambientLight)
        directionalLight = DirectionalLight("directionalLight")
        directionalLight.setColor(Vec4(0.6, 0.6, 0.6, 1))
        directionalLight.setDirection(Vec3(1, 1, -2))
```

```
        lightAttributes = lightAttri-
butes.addLight(directionalLight)
        render.node().setAttrib(lightAttributes)

    def carregaSom(self):
        musica = base.loadSfx("sounds/musica.mp3")
        musicaInterval = SoundInterval(musica, na-
me="musicaInterval")
        musicaInterval.loop()
```

No `carregaAmbiente()` o comando `setBackgroundColor()` configura a cor de fundo seguindo o padrão RGB. Nas outras linhas estamos carregando o modelo do ambiente e anexando-o ao render.

No `carregaLuz()` são criados dois tipos de luzes, ambiente e direcionada, que são adicionadas a uma `LightAttrib`. A última linha anexa a `LightAttrib` ao render, afetando assim toda a cena.

No `carregaSom()` criamos um `SoundInterval` carregando a música de um arquivo e colocamos para tocar indefinidamente com o `loop()`.

Avatar e Movimentação

Recursos: Modelo do avatar no formato .egg

A seguir as modificações para carregar um avatar com uma câmera focada nele, executar animações e movimenta-lo com o auxílio de uma `Task`.

As setas esquerda e direita serão utilizadas para girar o avatar, enquanto as setas cima e baixo farão ele andar para frente e para trás, respectivamente. Precisaremos dos seguintes módulos do Panda3D:

```
from direct.actor.Actor import Actor
from direct.task.Task import Task
```

E as modificações abaixo:

```
    def inicia(self):
```

```
        self.button.destroy()
        self.avatar = render.attachNewNode("avatar")
        self.avatarActor = Actor()
        self.avatarAndando = 0
        self.carregaLuz()
        self.carregaAmbiente()
        self.carregaAvatar()
        self.carregaSom()
        self.configuraCamera(self.avatar)
        self.configuraMovimento()
        taskMgr.add(self.movimento, "movimento")

    def carregaAvatar(self):
        self.avatarActor = Actor("models/tritao-model.egg",
                    {"parado":"models/tritao-parado.egg",
                    "andando":"models/tritao-andando.egg"})
        self.avatarActor.setPos(0, 0, 0)
        self.avatarActor.setScale(0.06, 0.06, 0.06)
        self.avatarActor.reparentTo(self.avatar)
        self.avatar.setPos(-10, 0 , 0)
        self.avatarParar()

    def configuraCamera(self, referencia):
        base.disableMouse()
        cameraNode = referencia.attachNewNode('cameraNode')
        camera.reparentTo(cameraNode)
        base.camLens.setNear(10)
        base.camLens.setFar(1000)
        cameraNode.setPos(-15, -80, 30)
        cameraNode.setHpr(-10, -15, 0)

    def configuraMovimento(self):
        self.accept('arrow_up', self.avatarAndar, ["frente"])
        self.accept('arrow_up-up', self.avatarParar)
        self.accept('arrow_down', self.avatarAndar, ["tras"])
        self.accept('arrow_down-up', self.avatarParar)
        self.accept('arrow_left', self.avatarVirar, ["esquerda"])
        self.accept('arrow_right', self.avatarVirar, ["direita"])

    def avatarAndar(self, sentido):
        if sentido == "frente" :
            self.avatarAndando += 1
        elif sentido == "tras" :
            self.avatarAndando += -1
        if self.avatarAndando <> 0 :
            animacao = self.avatarActor.actorInterval("andando")
            animacao.loop()
        else :
            self.avatarParar()

    def avatarVirar(self, sentido):
        angulo = self.avatarActor.getH()
        if sentido == "esquerda" :
```

```
            angulo += 90
            if angulo >= 360 :
                angulo -= 360
            animacao = self.avatarActor.hprInterval(0,
                        Vec3(angulo, 0, 0))
            animacao.start()
        elif sentido == "direita" :
            angulo -= 90
            if angulo < 0 :
                angulo += 360
            animacao = self.avatarActor.hprInterval(0,
                        Vec3(angulo, 0, 0))
            animacao.start()

    def avatarParar(self):
        self.avatarAndando = 0
        animacao = self.avatarActor.actorInterval("parado")
        animacao.loop()

    def avatarMovimenta(self, direcao, valor):
        if direcao == "X" :
            self.avatar.setX(self.avatar.getX() + valor)
        elif direcao == "Y" :
            self.avatar.setY(self.avatar.getY() + valor)

    def movimento(self, task):
        if self.avatarAndando <> 0 :
            direcao = "X"
            valor = 0.1
            if self.avatarActor.getH() == 0 :
                direcao = "Y"
                valor *= -1
            elif self.avatarActor.getH() == 90 :
                direcao = "X"
            elif self.avatarActor.getH() == 180 :
                direcao = "Y"
            elif self.avatarActor.getH() == 270 :
                direcao = "X"
                valor *= -1
            if self.avatarAndando == -1 :
                valor *= -1
            self.avatarMovimenta(direcao, valor)
        return Task.cont
```

No `inicia()` são instanciadas algumas variáveis relacionadas ao avatar que serão usadas em outros métodos. Na última linha, `taskMgr.add(self.movimento, "movimento")`, estamos incluindo uma tarefa, o método `movimento()`, no gerenciador de tarefas do Panda.

O `carregaAvatar()` anexa um `Actor` ao nó `self.avatar`, carregando o modelo tritao-model e as animações tritao-parado e tritao-andando de arquivos .egg.

No `configuraCamera()`, a primeira linha desabilita a navegação via mouse, em seguida criamos um nó para a câmera e anexamos a uma referência, nesse caso o próprio avatar.

No `configuraMovimento()` estão sendo criadas as associações de comandos do teclado a chamadas de métodos da classe `Tutorial`.

Os métodos `avatarAndar()`, `avatarParar()`, `avatarVirar()` e `avatarMovimenta()` são responsáveis pela animação do avatar. Os dois primeiros executam as animações carregadas de arquivos .egg, através do comando `actorInterval`, o terceiro realiza a rotação do `Actor`, com o comando `hprInterval` e o último realiza a traslação do avatar, alterando as coordenadas X e Y.

Finalmente, o método `movimento()` que na verdade trata-se de uma `Task`. O Panda3D chamará este método em todos os quadros, portanto é um recurso muito útil para operações que precisam ser executadas continuamente, como a translação do avatar em pequenos passos. A linha `return Task.cont` indica para o Panda que esta tarefa deve continuar sendo executada.

8.4. Conclusão

Este capítulo apresentou uma introdução ao Panda3D, discutindo seus principais aspectos, módulos, ferramentas e um jogo simples desenvolvido utilizando este engine. Panda3D é um engine que tem como ênfase a velocidade no aprendizado, simplicidade no código via os diversos módulos já implementados e um formato para inclusão de conteúdo artístico bem completo. Suas ferramentas também tornam os testes e decisões sobre o jogo bem mais rápidas,

sem necessidade de se programar uma grande quantidade do jogo. Para maiores informações e exemplos de como utilizar as diversas funcionalidades do Panda3D, consultar a página oficial do engine.

Referências

ETC, Panda3D. Disponível em: <http://www.panda3d.org/>. Data de acesso em: 31 de jan. 2007.

PYTHON. Disponível em: <http://www.python.org>. Data de acesso em: 31 de jan. de 2007.

DALMAU, Daniel S-C. (2003) "Scpriting", capítulo 9. In: Core Techniques and Algorithms in Game Programming", New Riders Publishing.

FMOD. Disponível em: <http://www.fmod.org>. Data de acesso em: 15 de fev. de 2007.7

Capítulo 9

enJine: game engine didático

Ricardo Nakamura, Romero Tori

INTERLAB - Laboratório de Tecnologias Interativas
Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo (USP)
Av. Prof. Luciano Gualberto, travessa 3 no. 158 - CEP 05508-970 - São Paulo, SP

{ricardo.Nakamura, romero.tori}@poli.usp.br

Abstract

EnJine is a didactic game engine, developed in Java and Java 3D and available under the GPL license. It can also be integrated with libraries such as ARToolkit. In this chapter, the architecture and main features of enJine will be presented, exploring ways of developing augmented and virtual reality systems with it.

Resumo

O enJine é um engine didático para jogos, desenvolvido em Java e Java 3D e disponível através da licença GPL. Ele também pode ser integrado com bibliotecas como o ARToolkit. Neste capítulo, a arquitetura e principais características do enJine serão apresentadas, explorando-se maneiras de desenvolver sistemas de realidade virtual e aumentada com ele.

9.1. Introdução

O enJine é um engine didático para jogos computacionais, desenvolvido em Java e utilizando-se da biblioteca Java 3D. Ele é um projeto mantido pelo Laboratório de Tecnologias Interativas (INTERLAB) da Escola Politécnica da Universidade de São Paulo e disponibilizado como software livre através da licença GPL. Desta forma, o software está aberto a melhorias e contribuições. Para baixá-lo e participar de sua comunidade de usuários e desenvolvedores deve ser acessado o endereço http://enjine.iv.fapesp.br.

Como engine didático, o principal objetivo do enJine é servir como ferramenta de apoio no ensino de diferentes disciplinas de Engenharia e Ciência da Computação nas quais a criação de jogos computacionais possa ser utilizada como atividade motivadora ou exemplo de aplicação. Este software tem sido utilizado com sucesso no ensino de Computação Gráfica, mas pode-se vislumbrar seu uso em assuntos como Engenharia de Software, Inteligência Artificial, Programação Orientada a Objetos, além de disciplinas de cursos voltados especificamente para o desenvolvimento de jogos.

O objetivo secundário do enJine é servir como plataforma para o desenvolvimento de protótipos e testes de novas tecnologias. Neste aspecto, a orientação a objetos, a arquitetura bem organizada e a adoção de boas práticas de Engenharia de Software no projeto do enJine contribuem para sua estabilidade, um elemento importante quando se procura avaliar novas tecnologias. É neste contexto que se pode contemplar a construção de ambientes de Realidade Virtual e Aumentada com o uso desta ferramenta.

Este capítulo se inicia com uma visão geral do enJine, incluindo suas características, arquitetura e os principais serviços fornecidos por ele. Em seguida, serão apresentadas formas de utilizar o software para construir ambientes de Realidade Virtual e Aumentada. O capítulo é concluído com observações sobre as expansões atualmente em desenvolvimento para o enJine.

9.2. Arquitetura do enJine

O enJine procura fornecer um conjunto de serviços úteis para o desenvolvimento de jogos, bem como uma estrutura de classes que auxilia a organização do jogo. Suas principais características são:

- Contém um conjunto de classes Java que permitem a representação de diferentes entidades do jogo tais como personagens, jogadores e elementos de cenário de forma flexível;
- Traz a infraestrutura básica para a implementação de diferentes jogos, permitindo interações complexas entre entidades do jogo através de um sistema de troca de mensagens;
- Inclui um sistema de detecção e notificação de colisões entre entidades do ambiente virtual, baseado em múltiplos estágios para melhor desempenho;
- Implementa a renderização de cenas tridimensionais através da API Java 3D, de forma a utilizar grafos de cena e os recursos de placas aceleradoras de vídeo;
- Implementa a renderização de modelos animados através da técnica "skin-and-bones" no formato .X;
- Contém recursos básicos para reprodução de áudio e tratamento de entradas do usuário, que podem ser expandidos por aplicações usuárias do enJine.

A figura 9.1 apresenta a organização do enJine em três camadas: na base encontram-se as APIs do Java, tais como o Java 3D, que são utilizadas pelos serviços do enJine. Cada serviço é implementado como um pacote de classes. Por fim, a camada de "framework" traz um conjunto de classes que simplificam o uso do enJine em alguns cenários comuns. O restante desta seção será dedicado à apresentação dos principais serviços do enJine para aplicações de Realidade Virtual e Aumentada.

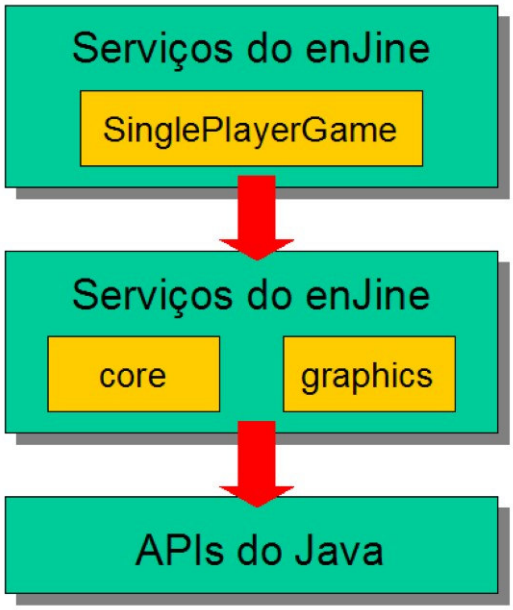


Figura 9.1 – Organização do enJine em camadas

9.2.1. Pacote "`core`"

O pacote "core" contém as classes utilizadas para representar o jogo e seus elementos. As três classes fundamentais deste pacote são Game, GameState e GameObject.

A classe Game representa o corpo principal do jogo e implementa um laço de execução flexível baseado no padrão de projeto "template method". Os detalhes da execução são implementados através de subclasses de GameState. Num dado momento durante a execução de um programa desenvolvido com o enJine, existe sempre um e somente um objeto de uma subclasse de GameState ativo. Os métodos deste objeto são executados a partir da classe Game, conforme mostrado na figura 9.2. Subclasses de GameState podem ser utilizadas para implementar diferentes modos de operação do ambiente virtual, como por exemplo telas de configuração.

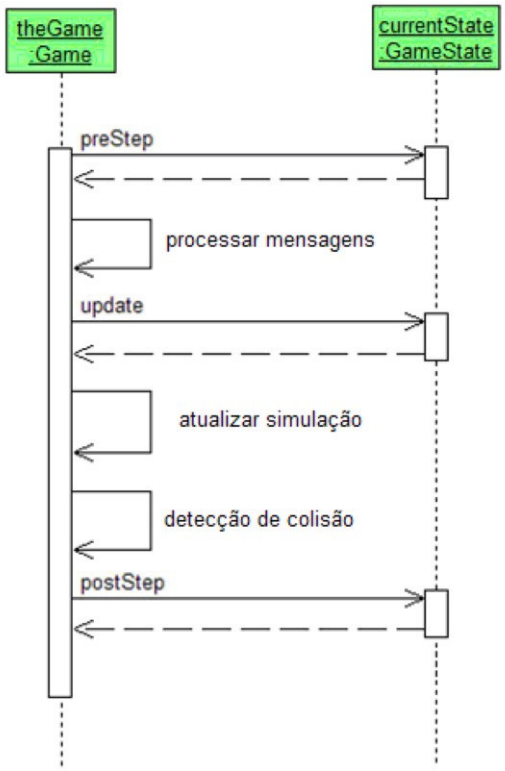


Figura 9.2 – modelo de execução de GameStates a partir do Game

A classe GameObject é utilizada para representar todo tipo de entidade no jogo. Para isso, uma subclasse de GameObject pode agregar instâncias de diferentes classes provedoras de serviços. Atualmente existem quatro serviços que podem ser agregados a um GameObject: Viewable, Updater, Collidable e Messenger, conforme ilustrado na figura 9.3. A classe Viewable deve ser agregada a entidades que possuem uma representação visual na aplicação. A classe Updater deve ser incluída em GameObjects que possuem algum tipo de comportamento dinâmico ou simulação. A classe Collidable indica que a entidade deve ser notificada sobre colisões com outros

elementos do ambiente. Por fim, a classe Messenger é utilizada para entidades que desejam se comunicar com outras através do mecanismo de mensagens do enJine.

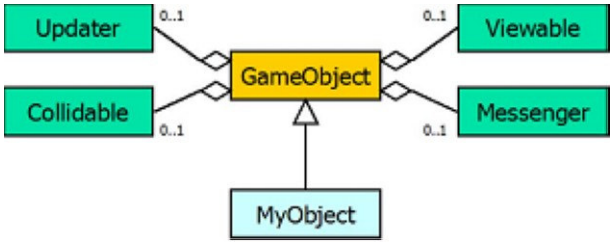


Figura 9.3 – Agregação de serviços no GameObject

9.2.2. Pacote "`graphics`"

O pacote "graphics" contém as classes responsáveis pela geração da visualização do jogo ou ambiente virtual. Estas classes fazem a interface com a biblioteca Java 3D e os recursos dela podem ser acessados diretamente, se necessário.

A principal classe do pacote "graphics" é chamada GameDisplay. Ela é responsável por configurar o ambiente de execução do Java 3D e construir a janela da aplicação. Outras classes importantes deste pacote são a classe Camera, que armazena as configurações de visualização do Java 3D e a classe GameWindow, que representa a janela de visualização. Por fim, o enJine também conta com um sistema de "overlays" para sobrepor informações 2D sobre a cena 3D. Esta funcionalidade é implementada na classe OverlayCanvas e na interface Overlay.

9.2.3. Pacote "`collision`"

O pacote "collision" contém as classes que implementam o sistema de detecção de colisões em múltiplas etapas no enJine. Num primeiro nível, o espaço virtual é particionado em volumes, representados por objetos da classe Subspace. O sistema de colisão

constrói listas de entidades que se encontram em um mesmo Subspace e constrói todas as combinações de dois elementos que poderiam colidir, conforme ilustrado na figura 9.4.

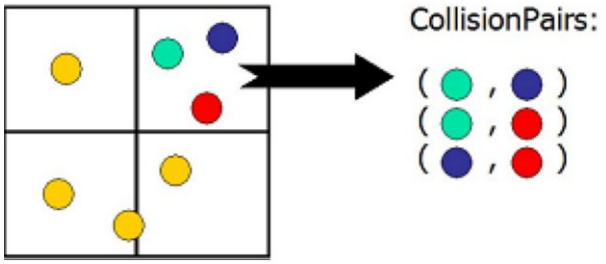


Figura 9.4 – Construção dos pares candidatos para colisão

Em seguida, os pares de entidades são submetidos a uma cadeia de "filtros de colisão" cuja responsabilidade é descartar rapidamente os pares que não podem colidir. O processo continua ao longo dos diferentes filtros, progressivamente mais complexos. Os pares que não foram descartados até passar pelo último filtro são considerados em colisão. Desta forma, é possível ajustar o nível de precisão do sistema de colisão através do encadeamento dos filtros desejados. Atualmente, o enJine implementa um filtro baseado em volumes envoltórios, mas é possível se desenvolverem outros filtros mais precisos.

9.3. Criando Ambientes Virtuais com o enJine

Esta seção traz roteiros que mostram como escrever uma aplicação que configure um ambiente de Realidade Virtual ou Aumentada com o enJine e como usar seus principais componentes para este objetivo.

9.3.1. Configuração de um ambiente mínimo

Uma aplicação mínima que utilize o enJine pode ser construída utilizando-se a classe SinglePlayerGame do framework do

enJine, e criando uma subclasse de GameState que corresponde ao ambiente virtual. A listagem 9.1 ilustra estes conceitos.

```
public class MinimalApplication {
    private SinglePlayerGame game;
    public MinimalApplication() {
    }
    public void run() {
        game = new SinglePlayerGame();
        game.setWindowParameters(640, 480, "Hello
World");
        game.initialize();
        game.mainLoop(25, new MyState());
    }
    public static void main(String[] args) {
        MinimalApplication game = new MinimalApplica-
tion();
        game.run();
    }
}

public class MyState extends GameState {
       …
}
```

Listagem 9.1 –Ambiente mínimo com o enJine

9.3.2. Construção de elementos do ambiente virtual

Uma vez que a estrutura básica do ambiente está pronta, é preciso criar as entidades que compõem o ambiente virtual. Isso é feito criando-se uma subclasse de GameObject e agregando-se subclasses dos provedores de serviços, conforme a necessidade. A listagem 9.2 mostra um exemplo em que uma classe seguindo o padrão de projeto "Object Factory" foi implementada para ser responsável por essas tarefas.

```
public class GameObjectFactory {
    private static GameObjectFactory theInstance;
    private GameObjectFactory() {
    }
    public static GameObjectFactory getInstance() {
```

```
        if (theInstance == null) {
            theInstance = new GameObjectFactory();
        }
        return theInstance;
    }
    public GameObject createMyEntity() {
        GameObject obj = new MyObject();
        Viewable view = new MyViewable();
        obj.setViewable(view);
        Updater upd = new MyUpdater();
        obj.setUpdater(upd);
        return obj;
    }
}

public class MyObject extends GameObject {
    private Point3f position;
    . . .
}

public class MyViewable extends Viewable {
    public void updateView(long delta) {
    // acrescente aqui código para atualizar
    // a representação visual do GameObject
    }
    public BranchGroup getView() {
    // acrescente aqui código para construir e
    // retornar a representação visual
    }
}

public class MyUpdate extends Updater {
    public void update(Game g, float delta) {
    // acrescente aqui código para implementar
    // o comportamento dinâmico da entidade
    }
}
```

Listagem 9.2 – Construção de uma entidade simples

Deve-se observar que, para entidades do ambiente virtual que não possuem comportamento dinâmico (por exemplo, elementos de cenário), basta agregar uma subclasse de Viewable e, conforme o caso, Collidable.

9.3.3. Carregamento de modelos

Para a representação visual dos elementos do ambiente, em geral deseja-se carregar um modelo 3D previamente criado em uma ferramenta apropriada. Isso pode ser feito utilizando-se os "loaders" do Java 3D e construindo-se um grafo de cena que contenha o modelo no Viewable. A listagem 9.3 traz um exemplo de Viewable que carrega um modelo no formato OBJ, utilizando o "loader" deste formato que faz parte da biblioteca Java 3D.

```
import java.io.FileNotFoundException;

import javax.media.j3d.*;
import javax.vecmath.*;

import com.sun.j3d.loaders.*;
import com.sun.j3d.loaders.objectfile.ObjectFile;

import interlab.engine.core.Viewable;

public class ModelView extends Viewable {
    private BranchGroup root;
    private TransformGroup placement;

    public void updateView(long delta) {
    // acrescente aqui código para atualizar
    // a representação visual do GameObject
    }
    public BranchGroup getView() {
        root = new BranchGroup();

        placement = new TransformGroup();
        placement.setCapability(
                  Trans-
formGroup.ALLOW_TRANSFORM_WRITE);
        root.addChild(placement);
        try {
            ObjectFile loader = new ObjectFile();
            Scene scene = loader.load("model.obj");
            placement.addChild(scene.getSceneGroup());
        }
        catch(FileNotFoundException ex) {
```

```
            ex.printStackTrace();
        }
        return root;
    }
}
```

Listagem 9.3 – Carregando um modelo em um Viewable

9.3.4. Interações entre elementos do ambiente

Existem dois tipos de interação que podem ocorrer entre entidades do ambiente virtual em uma aplicação implementada com o enJine: troca de mensages e colisão.

Para troca de mensagens, deve-se implementar uma subclasse de Messenger, incluindo código no método processMessage para tratar as mensagens recebidas. Para enviar uma mensagem, deve-se criar um objeto da classe InteractionMessage (ou uma subclasse especializada) e chamar o método sendMessage do objeto MessageManager, que é responsável por transmitir as mensagens. A listagem 9.4 ilustra estes procedimentos.

```
public class MyMessenger extends Messenger {
    private MessageManager mm;
    public void setMessageManager(MessageManager mm) {
        this.mm = mm;
    }
    public void processMessage(InteractionMessage m) {
    // para enviar uma mensagem, pode-se fazer:
    InteractionMessage im = new InteractionMessage();
        mm.sendMessage(im);
    }
}
```

Listagem 9.4 – Troca de mensagens

Para implementar a detecção de colisão, é preciso primeiramente configurar os Subspaces que particionam o ambiente virtual. Isto pode ser feito com código semelhante ao da listagem 11.5, durante a inicialização de um GameState. No caso deste exemplo, dois pa-

ralelepípedos são definidos como os únicos subespaços do ambiente. Isto significa que somente as entidades que se encontrem dentro destes volumes serão testadas para colisão.

```
SinglePlayerGame g = (SinglePlayerGame)game;

SimpleCollisionManager scm = g.getCollider();
BoundingBox sub1 = new BoundingBox(
        new Point3d(-2, -2, -2),
        new Point3d(2, 2, 0));
Subspace sp1 = new Subspace(sub1);
scm.addSubspace(sp1);
BoundingBox sub2 = new BoundingBox(
        new Point3d(-2, -2, 0),
        new Point3d(2, 2, 2));
Subspace sp2 = new Subspace(sub2);
scm.addSubspace(sp2);
```

Listagem 9.5 – Definição de subspaces

Em seguida, os elementos que podem colidir precisam agregar uma subclasse de Collidable que estabelece qual o comportamento em caso de uma colisão. Cada Collidable também possui um volume envoltório que é utilizado pelos filtros de colisão. Sempre que a posição lógica da entidade for alterada, este volume envoltório também deve ser atualizado. A listagem 9.6 traz um exemplo destes procedimentos.

```
public class MyCollidable extends Collidable {
    public MyCollidable() {
        BoundingBox bb = new BoundingBox(
                new Point3d(-1, -1, -1),
                new Point3d(1, 1, 1));
                  ObjectBounds bounds = new Object-
Bounds(bb);
        setBounds(bounds);
    }
    public void handleCollision(Collidable cl) {
    // acrescentar aqui código para tratar a
    // ocorrência de colisões
```

```
    }
}
```

Listagem 9.6 – Criação de um Collidable

9.3.5. Interações entre o usuário e o ambiente

A interação entre o usuário e o ambiente no enJine é feita por meio do tratamento de dispositivos de entrada. Cada dispositivo é representado por uma subclasse de InputDevice e contém um ou mais objetos da classe InputSensor, que representam os sinais que aquele dispositivo pode enviar. Por exemplo, a classe Keyboard contém um InputSensor para cada tecla.

Os dispositivos de entrada devem ser registrados em um objeto da classe InputManager, que é responsável pelo gerenciamento destes dispositivos. Além disso, através desta classe os InputSensors são associados a objetos da classe InputAction, que representam comandos dentro do ambiente. Desta forma, cada comando (por exemplo, andar para frente) não fica acoplado fortemente a um dispositivo, facilitando a configuração.

Uma vez que o dispositivo de entrada está inicializado e seus sensores estão ligados a ações, o tratamento dos sinais de entrada do usuário pode ser feito analisando-se o estado do InputAction correspondente, através do método getIntensity desta classe. Um padrão de implementação comum é armazenar os InputActions em objetos de subclasses de Updater e tratar estes InputActions a partir do método update( ).

A listagem 9.7 mostra como inicializar um dispositivo e associar sensores a ações. Este código normalmente será inserido no método de inicialização do jogo ou de um GameState.

```
InputManager input = g.getInput();
Keyboard key = Keyboard.getInstance();
```

```
// neste caso, updater.moveForward é um InputAction
// armazenado dentro de uma subclasse de Updater
input.bind(key.getSensor(KeyEvent.VK_W),
    updater.moveForward);
```

Listagem 9.7 – Inicializando o tratamento de dispositivos de entrada

A listagem 9.8 mostra um exemplo de subclasse de Updater com o tratamento de InputActions.

```
public class InputTreatment extends Updater {
    public InputAction moveForward;

    public InputTreatment() {
        moveForward = new InputAction(1, 0, “walk”);
    }
    public void update(Game g, float delta) {
        if (moveForward.getIntensity() > 0) {
        // acrescentar codigo de resposta ao
        // comando “mover para frente”
        }
    }
}
```

Listagem 9.8 – Tratamento de sinais de entrada

O enJine possui implementações de InputDevices para teclado e mouse, mas este modelo de dispositivos de entrada pode ser expandido para suportar outras tecnologias. Por exemplo, uma nova subclasse de InputDevice pode ser criada para encapsular o rastreamento de um marcador do ARToolkit. Esta classe forneceria, então, seis InputSensors correspondentes à posição e orientação do marcador a cada momento.

9.4. Conclusões

Este capítulo procurou apresentar as características principais do enJine e como utiliza-lo para a construção de ambientes de Realidade Virtual e Aumentada. O código-fonte do enJine, bem

como distribuições pré-compiladas, tutoriais e exemplos podem ser encontrados no site oficial, que é hospedado na Incubadora Virtual de Conteúdos da FAPESP: http://enjine.iv.fapesp.br

Além dos pacotes centrais do enJine apresentados neste texto, existem diversos trabalhos em andamento que procuram expandir ou adicionar novas funcionalidades para este software. Um pacote de Inteligência Artificial (IA) está sendo desenvolvido no próprio INTERLAB e deve ser incorporado ao enJine em breve. Este pacote deverá conter a infraestrutura para a representação de agentes inteligentes, bem como a implementação de comportamentos baseados em máquinas de estados, que são uma das formas mais freqüentes de simulação de inteligência encontrada em jogos computacionais. Este pacote também poderá ser expandido futuramente para incorporar técnicas mais elaboradas de IA.

Atualmente, também estão em desenvolvimento expansões do enJine para comunicação em redes e para integração com bibliotecas de Realidade Aumentada como o ARToolkit. A disponibilidade destes novos módulos deve facilitar a criação de novos ambientes virtuais colaborativos e aplicações de realidade aumentada.

9.4.1. Para saber mais

Os links a seguir contêm mais informações relacionadas ao enJine e às tecnologias utilizadas por ele.

http://enjine.iv.fapesp.br - Página oficial do enJine, contendo downloads, tutoriais e documentação.

http://www.interlab.pcs.poli.usp.br - Página do INTERLAB, contendo informações sobre o enJine, além de artigos sobre o uso de jogos em educação, entre outros.

http://java.sun.com - Página oficial da linguagem Java.

https://java3d.dev.java.net - Página oficial da API Java 3D.

http://java3d.j3d.org - Página da comunidade de usuários do Java 3D, contendo tutoriais e exemplos.

Capítulo 10

O Engine de Física AGEIA PhysX

Thiago Farias[1], Judith Kelner[1], Veronica Teichrieb[1], Daliton da Silva[1], Guilherme Moura[1], Márcio Bueno[1]

[1]Grupo de Pesquisa em Realidade Virtual e Multimídia, Centro de Informática – Universidade Federal de Pernambuco (GRVM CIn UFPE)
Av. Prof. Moraes Rego, S/N – Prédio da Positiva – 1º Andar – Cidade Universitária – 50670-901 – Recife – PE – Brasil

{tsmcf,jk,vt,ds2,gsm,masb}@cin.ufpe.br

Abstract

The PhysX engine allows the development of highly realistic graphics applications by simulating physics behaviors. This chapter aims to present this technology applying a theoretical approach. It will be introduced PhysX's main components, giving an overview of its functionalities. Features such as collision detection, rigid bodies and joints will be explained.

Resumo

O engine de física PhysX permite o desenvolvimento de aplicações gráficas altamente realistas pela simulação de comportamentos físicos. O objetivo deste capítulo é apresentar essa tecnologia explorando uma abordagem teórica. Serão introduzidos os principais

componentes do PhysX, dando uma visão geral das suas funcionalidades. Aspectos como colisão, corpos rígidos e juntas serão abordados

10.1. Introdução

No mundo real, objetos estão sempre interagindo entre si, cada um apresentando um comportamento diferente por causa das diferenças das matérias. Por exemplo, uma pessoa ao cair no chão, experimenta uma sensação diferente do que ao cair na água. Então, adicionar propriedades como atrito, colisão e torque em um jogo de corrida é essencial para fazer com que os jogadores se sintam dirigindo um carro real. Portanto, quanto mais uma aplicação incorpora física, mais ela se tornará interessante e realista [Eberly 2006].

Uma aplicação que possui simulação física requer muitos cálculos de alta complexidade, que podem ser embutidos na aplicação usando um *engine* de física. Para aliviar a carga da CPU (*Central Processing Unit*) é possível ter um processador dedicado para esta tarefa, assim como a unidade de processamento gráfico (*Graphical Processing Unit* - GPU) realiza a renderização gráfica.

As duas abordagens atualmente disponíveis são: usar a unidade de processamento físico (*Physics Processing Unit* - PPU) da AGEIA [PhysX 2007] ou GPUs de última geração.

10.1.1. *Engines* de Física

Um *engine* de física é um programa de computador que simula modelos físicos Newtonianos com o objetivo de prever o que aconteceria no mundo real em alguma situação específica. Existem dois tipos de *engines* de física, os de tempo real e os de alta precisão. Os *engines* físicos de alta precisão requerem muitos cálculos para refletir de forma confiável a vida real. Eles são necessários em aplicações científicas e em filmes animados por computador. Para

jogos, a simulação física precisa ser calculada em tempo real, portanto os modelos físicos são simplificados.

Tradicionalmente, os *engines* de física disponíveis tratam de detecção de colisão e de dinâmica de corpos rígidos. O tratamento de comportamentos como fluidos, roupas, partículas e dinâmica de corpos deformáveis requer excessivo poder de processamento, tornando a execução em tempo real inviável.

10.1.2. AGEIA PhysX SDK

O AGEIA PhysX SDK é um poderoso *middleware* de *engine* de física baseado em *software* para a criação de ambientes físicos dinâmicos [PhysX 2007]. O PhysX suporta as principais plataformas para jogos e aplicações gráficas, como mostra a Figura 10.10.1.

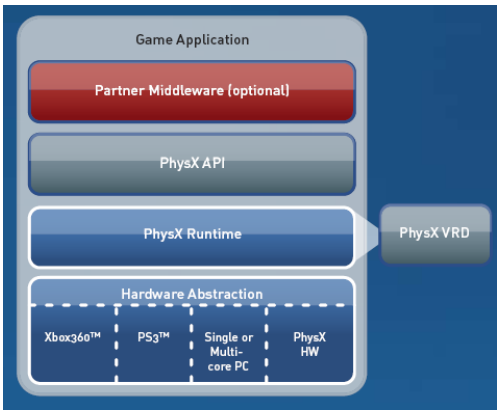


Figura 10.1. Arquitetura de uma aplicação desenvolvida utilizando o PhysX

A API do PhysX possui uma avançada tecnologia de simulação para jogos e aplicações:

- Dinâmica de objetos com corpos rígidos complexos e detecção de colisão, para movimentação e interação realista de

personagens;

- Juntas e molas são ferramentas úteis para modelagem de mecanismos complexos, como a movimentação de personagens e a criação de colisões integradas a efeitos;

- Criação e simulação de fluidos volumétricos pode melhorar os efeitos visuais de cenas, e tornar possível o uso de armas de fluidos ou mangueiras de incêndio;

- O sistema de partículas inteligentes do PhysX FX torna possível simular de forma realista fogo, fumaça e neblina;

- Roupas aumentam o realismo de cenas contendo cortinas trêmulas e roupas que se moldam aos corpos.

10.1.3. Processador AGEIA PhysX

Atualmente, o processador AGEIA PhysX é a primeira e única PPU dedicada construída para melhorar a física contida nos jogos [PhysX 2007]. Através do uso desta PPU é possível ter ações e comportamentos mais realistas em jogos e aplicações gráficas sofisticadas, sem diminuir seu desempenho. Uma foto da PPU AGEIA PhysX pode ser vista na **Figura 10.2**10.2.

Figura 10.2. PPU AGEIA PhysX

Para fazer uso dos poderosos melhoramentos físicos é necessário ter uma boa CPU e uma GPU de alto desempenho, pois a física demanda a renderização de objetos mais detalhados como o movimento de roupas, além da grande quantidade de destroços numa cena simulando explosões altamente realistas.

10.2. Arquitetura e Componentes

As classes mais importantes que compõem a arquitetura do *engine* PhysX são a NxPhysicsSDK, NxScene, NxActor, NxShape, NxJoint e NxMaterial. Estes componentes são sintetizados na seqüência.

10.2.1. O Mundo

O mundo é representado pelo PhysX SDK, que está implementado em C++, e é internamente organizado como uma hierarquia de classes. Cada classe que contém alguma funcionalidade acessível ao usuário implementa uma interface, que é na verdade uma classe base abstrata. Além disso, algumas funções de utilidade sem estados são exportadas como funções em C.

Todas as classes de interface têm um arquivo de cabeçalho com o mesmo nome do arquivo da classe e começam sempre com "Nx". Além disso, tipos e classes começam com letra maiúscula, enquanto métodos e variáveis começam com letra minúscula.

A NxPhysicsSDK é uma classe abstrata (*abstract singleton factory class*) usada para instanciar objetos e ajustar parâmetros globais que afetarão todas as cenas. Para obter uma instância desta classe, o método `NxCreatePhysicsSDK()` deve ser chamado. Maiores detalhes podem ser encontrados em [Farias et al. 2006].

10.2.2. A Cena

Uma cena é uma coleção de corpos, restrições e efetuadores, capazes de interagir entre si. O comportamento destes objetos ao longo do tempo é simulado pela cena.

Várias cenas podem ser criadas ao mesmo tempo, porém seus corpos e restrições não podem ser compartilhados. Por exemplo, o desenvolvedor não pode criar uma junta em uma cena e depois usá-la para conectar corpos em uma cena diferente (juntas são apresentadas na Subseção 10.2.5).

Uma cena pode manipular elementos do ambiente (como gravidade, corpos, juntas, materiais, etc.) e é representada internamente pela classe NxScene.

Para criar uma cena, primeiramente é necessário inicializar o mundo como visto na seção anterior. A partir da instância do mundo, a cena é criada a partir do descritor de cena (NxSceneDesc). É possível ajustar a gravidade (através do parâmetro `gravity` da classe NxSceneDesc) e material padrão da cena. Depois, resta apenas criar os objetos que serão inseridos na cena, como por exemplo um chão (plano). Criar um chão para a cena permitirá que os objetos, mesmo que sejam atraídos, não cairão e desaparecerão por causa da força da gravidade. Maiores detalhes sobre a criação de uma cena podem ser encontrados em [Farias et al. 2006].

10.2.3. O Ator

Um ator (NxActor) é o principal objeto de simulação no PhysX SDK. O ator é criado por uma cena específica, que o possui. Ele pode ser dinâmico, ou estático fixo no mundo. Atores dinâmicos/estáticos serão detalhados na Subseção 10.4.1.

Uma explicação sobre como criar um ator pode ser encontrada em [Farias et al. 2006].

10.2.4. As Formas

Formas são usadas para detectar e controlar colisões. A classe abstrata NxShape encapsula as subclasses de formas suportadas pelo PhysX, que são NxBoxShape, NxCapsuleShape, NxConvexShape, NxHeightFieldShape, NxPlaneShape, NxSphereShape, NxTriangleMeshShape e NxWheelShape. Nas subseções seguintes as subclasses mais importantes serão brevemente descritas; maiores informações sobre estas formas e as demais não descritas aqui podem ser encontradas em [Farias et al. 2006].

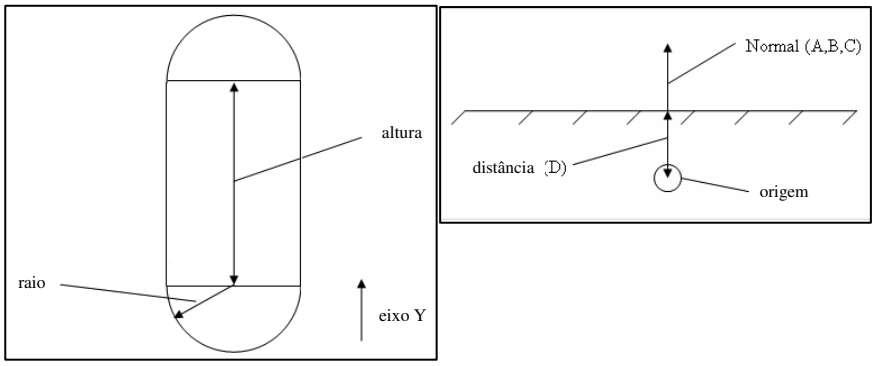
10.2.4.1. Forma de Caixa

A NxBoxShape é a classe que implementa a primitiva de detecção de colisões em forma de caixa. Cada forma está associada ao ator ao qual foi anexada.

Uma instância da classe pode ser criada através do método `createShape()` do objeto NxActor que a possui. As propriedades da forma estão definidas em um objeto descritor NxBoxShapeDesc, que contém toda a informação sobre a forma; este objeto é passado como parâmetro no método `createShape()`. A forma é excluída através do método `NxActor::releaseShape()` no ator proprietário.

10.2.4.2. Forma de Cápsula

A NxCapsuleShape é definida por um raio, que é o tamanho da extremidade hemisférica da cápsula, e por uma altura, que é a distância entre os dois hemisférios, conforme ilustrado na **Figura 10.3**a.



a **b**

Figura 10.3. Formas: a) cápsula; b) plano

10.2.4.3. Forma de Plano

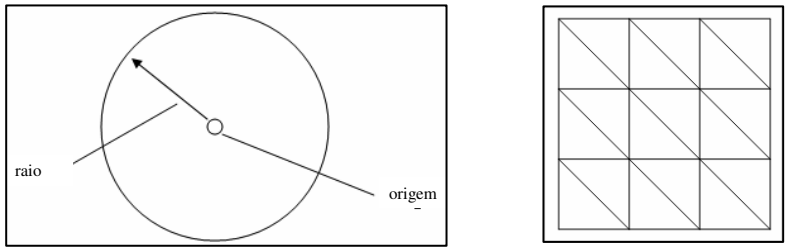
Uma forma de plano, ilustrada na **Figura 10.3**b, é uma primitiva de detecção de colisão planar. Por padrão, ela é configurada para ser o plano $y == 0$. O desenvolvedor pode então customizar a normal e uma variável d para especificar um plano arbitrário. d é a distância do plano para a origem ao longo da normal, assumindo que esta está normalizada. A forma deve usar o tipo NxPlaneShape.

10.2.4.4. Forma de Esfera

Uma primitiva de detecção de colisão esférica é implementada pela classe NxSphereShape, e definida pelo atributo raio (**Figura 10.4**a).

10.2.4.5. Forma de Malha Triangularizada

A classe NxTriangleMeshShape é uma forma instanciada de um objeto triangularizado, conforme ilustrado na **Figura 10.4**b.



a **b**

Figura 10.4. Formas: a) esfera; b) malha triangularizada

10.2.5. As Juntas

Uma junta é um ponto que conecta dois atores. Na descrição de uma junta, é definido como os atores podem mover-se um em relação ao outro sobre este ponto, restringindo o movimento tanto com relação à rotação quanto à translação. Um exemplo de uma junta típica pode ser visto na **Figura 10.5**.

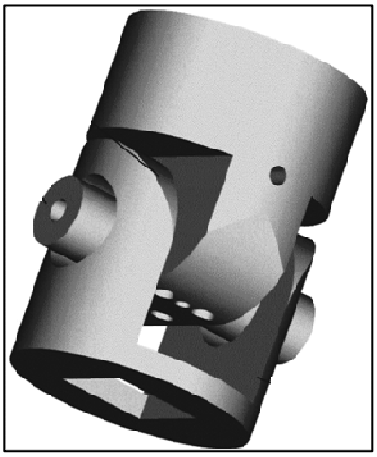


Figura 10.5. Um exemplo de uma junta típica

O uso de juntas é uma característica bastante útil e poderosa do PhysX SDK. Na Seção 10.6 serão detalhados tipos de juntas e suas propriedades, assim como as suas restrições, molas e motores.

10.2.6. Os Materiais

NxMaterial é uma classe que descreve as propriedades da superfície da forma. Um material padrão pode ser criado ajustando os valores de atrito e restituição. Além disto, pode-se criar um material (anisotrópico) que possui coeficientes diferentes de atrito, dependendo da direção na qual o corpo em contato está sendo movido. Este tipo de atrito é chamado anisotrópico, e é muito útil na modelagem de objetos como trenós ou esquis.

Uma explicação sobre como criar um material, padrão ou anisotrópico, pode ser encontrada em [Farias et. al. 2006].

10.3. Suporte Matemático

Algumas classes providas pelo PhysX para dar suporte a operações matemáticas e ajudar os programados a passar informações para o núcleo do *engine* de física. Estas classes incluem operações matemáticas sem estado, como relações trigonométricas e funções auxiliares (por exemplo, seno, tangente, valor absoluto, etc.), operações sobre matrizes (transposição, inversão, multiplicação, etc.), operações sobre vetores (produto escalar, produto interno, etc.) e funções relacionadas a quatérnios.

Para encapsular estas operações, o PhysX possui algumas classes que englobam funções relacionadas: NxMath, NxMat33, NxMat34, NxVec3 e NxQuat. NxMath é uma classe abstrata que contem métodos para tratar funções matemáticas escalares de ponto flutuante e define algumas constantes matemáticas. NxMat33 é uma abstração para uma matriz 3×3, usualmente utilizada para representar rotações ou tensores de inércia. NxMat34 é uma classe composta por um NxMat33 e um NxVec3, que permite realizar transformações afins. NxVec3 é uma classe que agrupa coordenadas x, y e z, implementando várias funções usadas em operações sobre vetores e funções de acesso. NxQuat é a representação de um quatérnio usada pelo PhysX, a fim de representar e simplificar rota-

ções 3D. Maiores detalhes sobre estas classes podem ser encontrados em [Farias et al. 2006].

10.4. Atores

Os atores são a base para qualquer contexto de simulação. Eles representam os objetos físicos no espaço. Muitas propriedades estão relacionadas com os atores, como posição global, orientação e interação com os outros atores que compartilham o mesmo cenário. Em relação ao comportamento, atores podem ser classificados como estáticos, dinâmicos ou cinemáticos, dependendo de como eles reagem a forças ou interação de contato com as outras partes do mundo.

10.4.1. Atores Estáticos Versus Dinâmicos

Atores estáticos são aqueles que fazem parte da paisagem, como construções ou qualquer elemento que seja imóvel no cenário. O principal propósito dos atores estáticos é prover apenas detecção de colisão. Eles não necessitam possuir massa, momento, inércia, ou qualquer atributo físico, pois a posição destes objetos não é modificada durante a simulação.

Os atores dinâmicos, que também podem ser chamados de "corpos" (*bodies*), devem conter um conjunto de informações relativas às propriedades do corpo, que é encapsulado pela classe NxBodyDesc.

Estáticos ou dinâmicos, os atores possuem um conjunto de formas, que pode estar vazio, para representar o controle de colisão. Estas formas podem ser usadas também como *triggers*, como pode ser visto na Subseção 10.7.2..

A criação de atores no PhysX consiste de alguns passos. Primeiro, deve-se criar e preencher a descrição do ator, através da classe NxActorDesc. Após isso, devem-se criar as formas que farão parte do modelo de colisão do ator. Atores podem possuir muitas

formas associadas a eles. Para adicionar mais formas, novas descrições têm que ser adicionadas no atributo `shapes` da classe NxActorDesc. Exemplos de formas foram dados na Subseção 10.2.4.

Um passo opcional é o ajuste das opções na descrição do corpo. Estas opções podem modificar o comportamento do ator em alguns aspectos, desabilitando a gravidade para ele, ou modificando sua interação com o mundo. Para desabilitar a gravidade em um ator, a opção `NX_BF_DISABLE_GRAVITY` deve ser habilitada. Uma opção importante é a `NX_BF_KINEMATIC`, que é responsável por desabilitar a interação de força e torque, incluindo a gravidade, sobre o ator. Para mover um ator cinemático, o programador tem que modificar sua posição global. O ator cinemático é um tipo intermediário entre os atores estáticos e dinâmicos, que pode se tornar um ator dinâmico em tempo de execução, de acordo com a vontade do usuário. O passo final é a criação real do ator, através da instância da classe NxScene.

Alguns parâmetros são necessários para uma simulação realística e são classificados como propriedades de corpos rígidos. Estes parâmetros englobam configurações como massa, posição do centro de massa, velocidade linear (do centro de massa), força resultante exercida no ator, tensor de inércia (descreve a distribuição de massa no corpo), orientação, velocidade angular e torque. Todos estes parâmetros podem ser encontrados na classe NxBodyDesc.

10.4.2. Os Materiais

O conceito de material está associado à reação com a colisão exercida entre as formas dos atores. Através da descrição de um material, o *engine* irá decidir se duas superfícies irão quicar quando se colidirem, quanta força irá ser aplicada para mover estas superfícies quando elas estão juntas, ou quão forte deve ser um impulso externo para vencer a inércia de um objeto estático.

Todas as formas têm um material associado a elas. Quando o programador não configura um material específico, o *engine* aplica o material padrão. Materiais são guardados internamente pelo PhysX, e todo material criado tem um índice que é unicamente mapeado para cada um deles. Desta forma, o processo de criação de uma forma que reage utilizando propriedades diferentes de material é feito utilizando o índice de um material já criado. Durante o processo de criação, alguns parâmetros são ajustados:

- Restituição, relativa a quão elástica será a colisão entre este material e outro. Esse valor deve permanecer no intervalo entre 0 e 1, oferecendo maior elasticidade quando perto de 1;

- Fricção estática, relativa ao coeficiente de fricção estática. O valor deve estar entre 0 e $+\infty$. Esta propriedade só é utilizada enquanto o corpo está em repouso;

- Fricção dinâmica, relativa ao coeficiente de fricção dinâmica. O valor deve ficar no intervalo de 0 a 1, e é utilizado quando o corpo está em movimento e em contato com outro corpo.

10.5. Interação entre Elementos

No PhysX, pode-se interagir com elementos (atores), aplicando forças e/ou torques, utilizando diversos modos de força.

10.5.1 Força e Torque

A maneira mais natural de interagir com um corpo é aplicando uma força externa que diretamente controla a sua aceleração e indiretamente controla a sua velocidade e posição. Geralmente, quando se aplicam forças, a simulação será capaz de manter todas

as restrições (juntas e contatos) satisfeitas, mas não obterá uma resposta imediata.

As forças aplicadas a um corpo são acumuladas antes de cada *frame* de simulação, então utilizadas na simulação e depois são canceladas na preparação para o próximo *frame*.

10.5.2. Aplicando Força e Torque

Para aplicar força e torque é necessário usar os métodos de NxActor. O método `addForce()` aplica uma força ao ator utilizando coordenadas globais, enquanto que o método `addLocalForce()` usa as coordenadas locais do *frame*. O mesmo acontece com os métodos `addTorque()` e `addLocalTorque()`. Estes métodos aplicam forças e torques ao centro de massa.

O primeiro parâmetro destes métodos é um vetor 3D que define o módulo e a orientação da força ou do torque. O segundo é o modo da força, que será detalhado na próxima subseção.

As mesmas considerações sobre coordenadas locais e globais dos *frames* podem ser aplicadas a estes métodos, isto é, o vetor de força pode referenciar as coordenadas locais ou globais, assim como as posições onde a força será aplicada podem ser em relação às coordenadas globais ou locais.

10.5.3. Modos de Força

O SDK divide cada *time step* em um número de *substeps* que são integrados pela física, e normalmente quando uma força é aplicada a um ator ela será aplicada toda de uma vez no primeiro *substep*.

A enumeração NxForceMode define um modo de força que será aplicado utilizando os métodos `addForce()`/`addTorque()` e seus derivados; estes modos são detalhados a seguir:

- NX_FORCE – para aplicar uma força (massa * distância / tempo^2) a um objeto no primeiro *substep* do *time step*;

- NX_IMPULSE – para aplicar um impulso, que é uma mudança de *momentum* (massa * distância / tempo), ao ator no primeiro *substep* do *time step*;

- NX_VELOCITY_CHANGE – para ajustar a velocidade (distância / tempo) de um objeto diretamente no primeiro *substep* do *time step*;

- NX_SMOOTH_IMPULSE – para aplicar um impulso ao objeto em cada *substep* do *time step*;

- NX_SMOOTH_VELOCITY_CHANGE – para mudar a velocidade de um objeto em cada *substep* do *time step*;

- NX_ACCELERATION – para aplicar aceleração (distância / tempo^2) a um objeto no primeiro *substep* do *time step*.

10.6. Juntas

Assim como foi introduzido na 0, juntas são conexões entre corpos rígidos. A seguir, alguns tipos de juntas predefinidas no PhysX e seus parâmetros serão mais detalhadamente descritos.

10.6.1 Tipos

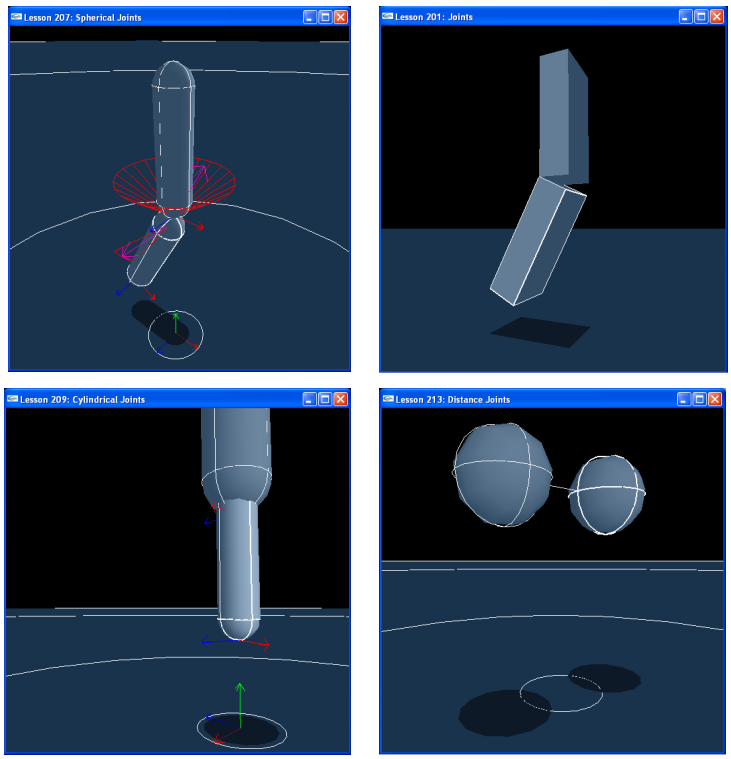


Figura 10.6. Juntas: a) esférica; b) de revolução; c) cilíndrica; d) de distância

No PhysX as juntas são classificadas em diferentes tipos: *spherical*, *revolute*, *prismatic*, *cylindrical*, *fixed*, *distance*, *point in plane*, *point on line* e *pulley*. Os principais tipos de juntas (ver Figura 10.6) serão descritos nas subseções a seguir; maiores informações sobre os demais tipos podem ser encontradas em [Farias et al. 2006].

10.6.1.1. Junta Esférica

Juntas esféricas (*spherical joints*) possuem três graus de liberdade, e dessa forma podem rotacionar ao redor dos eixos de outras juntas enquanto estão fixas em uma posição.

Juntas esféricas são comumente utilizadas para simular articulações semelhantes ao ombro (*shoulder joints*) e outras juntas do tipo bola-*socket (ball-and-socket joints*). As juntas são livres para girar ao redor de qualquer eixo, mas podem ser limitadas no quanto elas podem girar em torno de cada eixo. Na Figura 10.6a podem ser vistos dois objetos unidos por uma junta esférica, limitados pela área vermelha.

Para criar juntas esféricas o desenvolvedor precisa de referências para os objetos de dois atores. Adicionalmente, o desenvolvedor precisa definir um ponto que é a âncora global, que indica onde a junta será posicionada, e um eixo global, que servirá para definir o limite dos eixos. Ainda é preciso configurar os limites oscilação (*swing*) e torção (*twist*). O limite oscilação serve para não permitir que o eixo global definido ultrapasse um determinado ângulo, e o limite torção não permite que o eixo rotacione livremente.

10.6.1.2. Junta de Revolução

A junta de revolução (*revolute joint*) é freqüentemente chamada de "*hinge joint*" (dobradiça), porque pode ser utilizada para representar a dobradiça de uma porta (ver Figura 10.6b).

Para criar uma junta dobradiça o desenvolvedor precisa definir a âncora global e o eixo global, da mesma forma que na junta esférica. Se a *flag* colisão for habilitada, os atores colidirão entre si.

10.6.1.3. Junta Cilíndrica

Juntas cilíndricas (*cylindrical joints*) são livres para rotacionar em torno do eixo primário, como uma junta de revolução, bem

como transladar em torno do eixo primário, como uma junta prismática. A junta prismática (*prismatic joint*), exemplificada por um amortecedor, é puramente de translação e apenas pode mover-se ao longo do seu eixo primário. A Figura 10.6c mostra um exemplo.

A configuração da junta cilíndrica é idêntica à da junta prismática, assim como a função de criação.

10.6.1.4. Junta de Distância

A junta de distância (*distance joint*) conecta dois atores por uma haste. Cada extremidade da haste é unida a um ponto âncora em cada ator (ver Figura 10.6d).

Assim, diferentemente das outras juntas mencionadas até agora, esta junta particular tem dois pontos âncora. Estas âncoras são ajustadas localmente.

Com o objetivo de ajustar as distâncias máximas e mínimas entre os objetos conectados pela junta, o desenvolvedor pode calcular a distância inicial entre os objetos e multiplicá-la por um parâmetro. Pode-se também adicionar um fator de mola à junção, se for desejado.

10.6.2. Parâmetros

A fim de obter uma simulação próxima da realidade, quando se está tratando de juntas, o desenvolvedor deve modificar alguns parâmetros para um ajuste fino do comportamento da junta. Alguns desses parâmetros são descritos nas subseções seguintes.

10.6.2.1. Limites da Junta

Restrições de movimentos podem ser aplicadas em juntas, simulando alguns aspectos do objeto em questão. Estas restrições são chamadas de limites, no contexto do PhysX. Limites podem ser aplicados de duas formas: fornecendo um intervalo para o ângulo

relativo entre os corpos que compõem a junta, ou criando planos limitadores que restringem a junta. O PhysX não limita o número de planos limites. Estes tipos de limites podem ainda ser utilizados conjuntamente, para obter o efeito desejado.

Para definir os limites angulares de uma junta, primeiramente é setada uma *flag* para habilitar o uso de limites na junta. Posteriormente, os limites da junta são atribuídos, um superior e um inferior. O parâmetro de restituição, em cada limite, é um valor entre 0 e 1, que define a elasticidade do limite, sendo responsável por modelar a reação dos corpos no caso de um limite ser alcançado.

10.6.2.2. Juntas Quebráveis

Há situações em que são necessárias juntas que possam ser quebradas. Para quebrar uma junta, uma força ou torque tem que ser aplicado no conjunto de corpos. Um exemplo de junta quebrável é uma porta que pode ser derrubada dependendo da força do pontapé.

Depois da criação da junta, são definidos dois valores máximos para limitar as forças e torques que podem ser aplicadas no conjunto de corpos, sem causar sua ruptura. Em seqüência, o método `setBreakable()` é chamado, recebendo como parâmetros os valores definidos.

10.6.2.3. Motor da Junta

O motor da junta é usado para fornecer um torque relativo entre dois corpos, conectados por uma junta de revolução ou esférica.

Primeiramente, o motor da junta é habilitado ajustando a *flag* `NX_RJF_MOTOR_ENABLED` no descritor da junta. Então, um

NxMotorDesc é criado para ajustar os parâmetros do motor. Os parâmetros que podem ser ajustados são:

- `velTarget` - o valor máximo da velocidade;
- `maxForce` - a força máxima que será aplicada para conseguir o `velTarget`;
- `freeSpin` - caso o motor esteja girando livremente (ponto morto), o corpo girará livremente (inércia)? Se a resposta for verdadeira, isto acontecerá quando o motor for desligado. Para tanto se deve selecionar o valor zero para o parâmetro `maxForce`.

10.6.2.4. Mola da Junta

As molas podem ser adicionadas nas juntas para criar efeitos no seu eixo ativo. O efeito gerado por uma mola é similar a uma mola real, com forças esticando e comprimindo, empurrando e afastando as partes da junta. Esta característica pode ser adicionada para aumentar a fidelidade do modelo ao mundo real, em alguns casos.

10.7. Detecção de Colisão

Detecção de colisão é uma característica esperada em *engines* físicos. No PhysX, as colisões são processadas levando em consideração atributos associados aos atores envolvidos, de acordo com a elasticidade na colisão, tensores de inércia, momento e outras propriedades. Os desenvolvedores podem ignorar a manipulação da colisão, deixando esta tarefa para o *engine*. Em alguns casos, eventos e informações sobre colisões seriam de grande utilidade para a criação de efeitos, mudança do estado da simulação (contato entre objetos inflamáveis e partículas de fogo) ou detecção de aspectos importantes no contexto da simulação.

O conceito de colisão pode ser utilizado para criar "zonas de alerta", que podem lançar eventos passíveis de manipulação através de módulos especializados de relatório do usuário (*User Reports*). Nestes relatórios o ator relacionado ao evento de colisão diz: "Ei, eu sou um ator e estou atravessando esta zona! Faça alguma coisa comigo ou modifique o ambiente, por favor!".

Para visualizar ou manipular eventos de colisão, o PhysX oferece algumas técnicas usadas através de relatórios fornecidos pelo *engine*. Estas técnicas são explicadas na seqüência.

10.7.1. Raycasting

Para explicar esta técnica, uma frase simples pode ser dita: "*Raycasting* é detecção de colisão com raios.". Este procedimento pode ser aplicado para selecionar objetos com o *mouse*, checar a existência de uma linha de visão entre dois objetos, calcular distâncias, simular ações de tiros, entre outros.

Como sua denominação em inglês diz, *raycasting* consiste em lançar raios e inspecionar o que e onde eles acertaram. Uma típica técnica de "*point-and-shoot*" (apontar e atirar) recupera informações sobre os objetos interceptados pelo raio, de acordo com alguns parâmetros ajustados antes da execução do método.

No PhysX, *raycasting* pode ser feito através de três modalidades, variando apenas a granularidade de informações retornada. Na primeira forma, apenas a informação de que o raio colidiu ou não é retornada pelo método `raycast` (`raycastAnyShape()`). Na segunda forma, o método retorna apenas o objeto mais próximo atingido pelo raio e a distância entre o emissor e o objeto (`raycastClosestShape()`). O último método retorna o número total de objetos atravessados pelo raio e enumera todos estes através de um NxUserRaycastReport, que deve ser fornecido pelo usuário, a fim de que ele seja avisado sempre que o raio atinja algum objeto (`raycastAllShapes()`). Todos os métodos citados têm

como parâmetro um objeto do tipo NxRay, que representa o raio. Ele tem dois atributos, que representam a origem do raio e a sua direção, que deve sempre ser representada por um vetor normalizado. Outro parâmetro comum destes métodos é o tipo dos objetos que podem ser atingidos, podendo assumir valores como `NX_STATIC_SHAPES`, `NX_DYNAMIC_SHAPES` e `NX_ALL_SHAPES`, representando respectivamente objetos estáticos, dinâmicos e todos os tipos de objetos. Maiores informações sobre como implementar e utilizar os métodos podem ser encontradas em [Farias et al. 2006].

10.7.2. Triggers

Um *trigger* é um mecanismo para detectar a presença de formas em áreas específicas da cena. Este recurso é implementado com uma forma (pode ser qualquer tipo de forma disponível no PhysX) que permite que outras formas atravessem-na e avisa quando ocorreu alguma interseção. É importante saber que formas construídas através de malhas de triângulos são interpretadas pelo gerador de eventos como superfícies ocas e não como volumes. O ato de atravessar um *trigger* pode gerar muitos eventos, incluindo eventos de entrada, de saída ou simplesmente eventos de permanência na zona do *trigger*. O tratamento de eventos do *trigger* deve ser feito pelo programador utilizando o relatório (NxUserTriggerReport) fornecido pelo PhysX.

O conceito de *trigger* pode ser usado para simular sensores de presença, áreas de monitoramento, portas automáticas, etc. Notas sobre a implementação e utilização do conceito de *trigger* na prática podem ser encontradas em [Farias et al. 2006].

10.7.3. Contact Reports

Contact reports ou relatórios de contato são relatórios definidos pelo usuário que são chamados quando uma colisão ocorre.

Somente colisões envolvendo atores previamente identificados pelo desenvolvedor serão tratadas pelo relatório de contato. Este relatório pode ser usado para manipular colisões genéricas, adicionar efeitos de som quando ocorrer um contato, aplicar deformações a malhas, etc. Para usar relatórios de contato, um NxUserContactReport deve ser implementado.

Os relatórios de contato podem lançar eventos de início de contato, fim de contato ou permanência de contato.

Para informar ao *engine* quais elementos devem gerar eventos para o relatório de contato, o programador deve informar quais formas, atores ou grupos de formas e atores estão envolvidos neste procedimento. Isto pode ser feito apenas atribuindo grupos aos atores e formas, ou ainda habilitando o contato específico com um par de atores. Em qualquer destas formas é necessário dizer ao *engine* que tipo de eventos estes atores ou formas poderão gerar. Mais informações sobre implementação e exemplos podem ser encontradas em [Farias et al. 2006].

10.8. Considerações Finais

Fornecer física em jogos e aplicações gráficas não é uma tarefa trivial. Estes ambientes requerem um processamento extremamente intensivo baseado em um conjunto único de algoritmos de simulação física. Estes algoritmos demandam quantidades enormes de cálculos matemáticos e lógicos, com suporte de uma massiva largura de banda de memória. Desta forma, o PhysX ajuda a resolver estas questões fornecendo um conjunto de funções que encapsulam todos os algoritmos e cálculos mais complexos, acelerando o processo de desenvolvimento.

Referências

Eberly, D., Game Physics, Interactive 3D Technology Series, Morgan Kaufmann, 2006.

(2007) "AGEIA PhysX", http://www.ageia.com/physx/, Fevereiro.

Farias, T., Silva, D., Moura, G., Bueno, M., Teichrieb, V. e Kelner, J. (2006) "Minicurso SBGames 2006: AGEIA PhysX", http://www.cin.ufpe.br/~grvm, Fevereiro.