ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

VWML: Linguagem e Ambiente de Modelagem de Mundos Virtuais

Trabalho de Conclusão de Curso

Engenharia da Computação

César Augusto Mendonça de Carvalho
Orientador: Prof. Ricardo Massa Ferreira Lima

Recife, julho de 2005

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

VWML: Linguagem e Ambiente de Modelagem de Mundos Virtuais

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

César Augusto Mendonça de Carvalho
Orientador: Prof. Ricardo Massa Ferreira Lima

Recife, julho de 2005

César Augusto Mendonça de Carvalho

VWML: Linguagem e Ambiente de Modelagem de Mundos Virtuais

Resumo

Computação gráfica engloba um conjunto de tecnologias utilizadas para criar objetos tridimensionais a serem visualizados através de computadores. As suas aplicações abordam diversas áreas, incluindo arte, entretenimento e estudos científicos. Imagens tridimensionais (3D) geradas em computador requerem linguagens específicas para fornecer informações (cor, textura, localização, forma, etc) relacionadas aos objetos a serem criados. Este trabalho propõe uma nova linguagem para descrição de ambientes 3D. Este desenvolvimento foi motivado por três princípios básicos: (i) a sintaxe da linguagem não será uma tecnologia proprietária; (ii) a sintaxe será simples e intuitiva; (iii) a linguagem deve ser baseada em um formato padrão, com mecanismos automáticos para a leitura e escrita de modelos. Por esses motivos, decidimos criar uma linguagem baseada em XML. Um ambiente para escrever modelos desenvolvidos através da nova linguagem foi criado. No entanto, decidimos reutilizar ferramentas existentes para a visualização de modelos VRML. Assim, um compilador foi implementado para traduzir as descrições da linguagem nas suas equivalentes em VRML. Portanto, ferramentas comuns de exibição de informações VRML podem ser usadas. Finalmente, este trabalho apresenta um programa de assistência para inserir descrições 3D dentro de um programa Java. O objetivo é facilitar a comunicação entre o designer gráfico e o programador Java.

Abstract

Computer graphics comprises a set of technologies employed to create objects to be visualized through computers. Its applications encompasses a wide range of areas, including art, entertainment and scientific studies. Three-dimensional (3D) images generated using computers require specific languages to provide information (color, texture, location, form, etc) about the objects to be described. This work proposes a new language for 3D environment description. Its development was motivated by three basic principles: (i) the language syntax is not a proprietary technology; (ii) the syntax is simple and intuitive for graphical designers; (iii) the language is based on standardized format, with automatic mechanism for reading/writing the models. Therefore, it was decided to create a XML based language. An environment to exhibit models developed through our language was created. However, we decided to reuse existent tools for the visualization of VRML models. Thus, a compiler was implemented to translate the language descriptions into their equivalent in VRML. Then, an ordinary tool for the exhibition of VRML descriptions can be employed. Additionally, a graphical interface to assist designers is available. Finally, this work presents a wizard to plug 3D descriptions into a Java program. The aim is to facilitate the communication between graphical designers and Java programmers.

Sumário

Índice de Figuras

Tabela de Símbolos e Siglas

(Dispostos por ordem de aparição no texto)

XML - eXtensible Markup Language (linguagem de marcação extensível)

VWML - Virtual World Markup Language (linguagem de marcação de mundos virtuais)

VRML - Virtual Reality Modeling Language (linguagem de modelagem de realidade virtual)

X3D - eXtensible 3D (3D extensível)

API - Application Programming Interface (interface de programação de aplicativos)

ISO - International Organization for Standardization (organização internacional de padronização)

XSL - eXtensible Stylesheet Language (linguagem de folha de estilo extensível)

SGML - Standard Generalized Markup Language (linguagem padrão de marcação generalizada)

W3C - World Wide Web Consortium (consórcio da rede mundial de computadores)

HTML - HyperText Markup Language (linguagem de marcação de hipertexto)

XHTML - eXtensible HyperText Markup Language (linguagem de marcação de hipertexto extensível)

DTD - Document Type Definitions (definição de tipo do documento)

SAX - Simple API for XML (API simples para XML)

DOM - Document Object Model (modelo de objeto do documento)

BNF - Backus Naur Form (forma de Backus Naur)

YACC - Yet Another Compiler Compiler (ainda um outro compilador de compiladores)

JavaCC - Java Compiler Compiler (compilador de compiladores Java)

EBNF - Extended Backus Naur Form (forma de Backus Naur extensível)

CSS - Cascading Style Sheets (folha de estilo em cascata)

JDOM - Java Document Object Model (modelo de objeto do documento Java)

IDE - Integrated Development Environment (ambiente integrado de desenvolvimento)

Agradecimentos

Primeiramente a Deus, que me sustenta no ser e no poder difundir, pelos mais necessitados, um pouco do saber que Ele me permitiu adquirir;

Aos meus pais, Cezário Timóteo de Carvalho e Cleonice Mendonça de Carvalho, que me apóiam e incentivam na superação dos obstáculos que aparecem na minha vida;

Ao Prof. Ricardo Massa Ferreira Lima, que foi o meu ilustre orientador e me apoiou fortemente durante o desenvolvimento desta monografia, dando sugestões e opiniões a respeito de todo o seu conteúdo;

Ao Prof. Carlos Alexandre Barros de Mello, coordenador da cadeira de Trabalho de Conclusão de Curso, por suas orientações diretas, bem como pelas palestras, que nos proporcionou, de outros professores, ocasionando, assim, o desenvolver do presente trabalho, com requintes de boa qualidade;

Ao Prof. Márcio Lopes Cornélio, sempre presente nas adequadas soluções a problemas encontrados no decorrer desta pesquisa;

Aos demais Professores do Departamento de Sistemas Computacionais da Escola Politécnica de Pernambuco, pelos conhecimentos e experiências a nós transmitidos com abnegação e esmero;

Ao engenheiro Cláudio Cavalcanti, que teve importante participação na idealização e concretização do presente trabalho;

E, finalmente, a todos os alunos que, comigo, fizeram do Curso de Engenharia da Computação um saudável e acolhedor ambiente de estudo.

Capítulo 1

Introdução

A computação gráfica [1][2] possibilitou a criação, armazenamento e manipulação de imagens no ambiente computacional e, com isso, acelerou o desenvolvimento de representações gráficas de cenas do cotidiano e/ou imaginárias já que existem diversas ferramentas que auxiliam este propósito.

O uso de imagens em qualquer área, seja ela de entretenimento, educação ou comercial, é de grande importância devido à possibilidade de as mesmas trazerem muitas informações a respeito do tema em questão, e, se bem desenvolvidas, possibilitarem uma rápida e fácil assimilação de idéias.

A qualidade de imagens criadas no computador vem aumentando consideravelmente nos últimos anos. A prova disso está, entre outros lugares, nos filmes que contêm personagens digitais (criados no computador), que fazem os espectadores se confundirem quanto à realidade dos mesmos.

A Figura 1 mostra um exemplo de uma imagem gerada no computador. Essa imagem é uma representação em 3 dimensões de um carro. Por ser em três dimensões, a mesma pode ser utilizada para que o usuário visualize o objeto em diversos ângulos, fornecendo, assim, mais informações a respeito do mesmo.

Esse tipo de aplicação vem sendo usado em diversas áreas, principalmente científica, pois possibilita a manipulação de objetos virtuais que simulem as características de elementos reais. Tal estratégia permite estudar o comportamento e características desses elementos de forma mais rápida, flexível e com custos relativamente baixos.

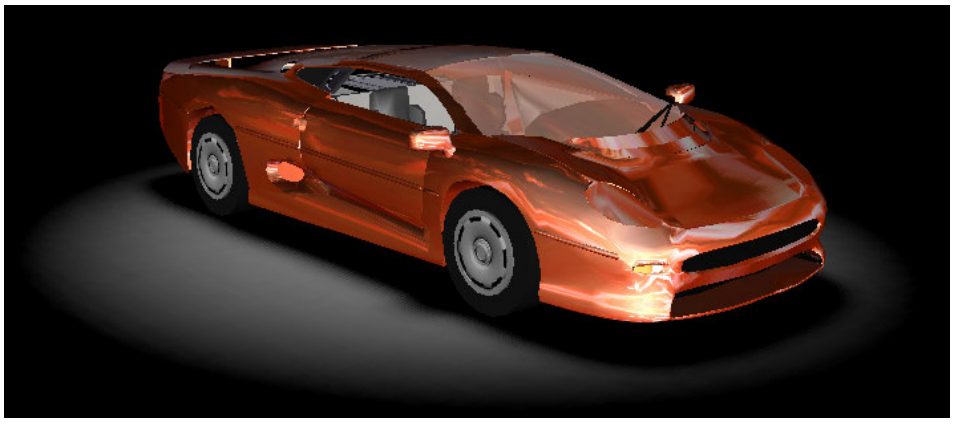

Figura 1. Exemplo de imagem gerada no computador

Para a criação de imagens 3D (tridimensionais) no computador, podem ser necessárias linguagens de descrição específicas. Tais linguagens devem ser capazes de expressar, de forma precisa, todas as características do objeto que se deseja descrever, tais como forma, textura, cor, escala, posicionamento, etc.

Existem diversas linguagens para esse propósito. Porém, observam-se dificuldades quanto ao uso de algumas, como, por exemplo, o fato de serem proprietárias, obrigando o usuário a comprar uma licença para o seu uso, ou terem uma sintaxe complexa, restringindo seu uso a poucos usuários especialistas.

Tomando por base essa realidade, o presente trabalho tem por objetivo criar uma linguagem de descrição de objetos 3D. Para tal, houve o foco em três princípios básicos: i) a linguagem será aberta permitindo, assim, o seu uso e até modificações de forma livre; (ii) a sintaxe da linguagem será simples e intuitiva para facilitar o seu uso; (iii) a linguagem será baseada em um formato padrão.

O padrão escolhido para o desenvolvimento da linguagem foi o XML (***Ex**tensible **M**arkup **L**anguage*) [25] [26]. A sua descrição, estrutura e motivações para seu uso serão abordadas neste trabalho.

O nome da linguagem desenvolvida é VWML (***V**irtual **W**orld **M**arkup **L**anguage*). Todas as funcionalidades suportadas pela mesma serão abordadas e explicadas em detalhes neste trabalho. Assim, o leitor inexperiente entenderá de forma mais rápida alguns dos diversos conceitos de modelagem 3D, além de como expressá-los através da linguagem criada. Fica fácil a ampliação das funcionalidades da linguagem VWML já que a mesma é aberta e baseada em um padrão.

Ter uma forma de visualizar as imagens criadas por esta linguagem proposta é fundamental para sua análise. Uma possibilidade de visualização está na utilização de *plugins* para navegadores, onde, além de visualizar a imagem, o usuário poderá interagir com ela.

Já existem diversos *plugins* para visualização de ambientes 3D. Muitos deles são direcionados à linguagem VRML (***V**irtual **R**eality **M**odeling **L**anguage*) [12][13]. A linguagem de descrição de mundos virtuais, VRML, tem sua origem concomitante ao surgimento e desenvolvimento de multimídia na Internet. Foi criada para que ambientes construídos em 3D pudessem ser visualizados de forma rápida e interativa por meio da rede mundial de computadores. A linguagem é aberta para o uso, porém não é intuitiva para os usuários finais, que precisam de um estudo aprofundado para iniciar o desenvolvimento de imagens 3D.

Neste sentido, como forma de agilizar o desenvolvimento do ambiente de visualização da linguagem VWML, decidiu-se por reutilizar ambientes já existentes. Para tanto, foi necessário desenvolver um compilador a fim de traduzir a descrição XML para a sua equivalente em VRML. Assim, o usuário pode trabalhar diretamente com o XML e visualizar, automaticamente, o objeto descrito. A tradução entre as duas linguagens pode ser totalmente transparente para o usuário, que não precisa saber que o *plugin* de visualização está lendo um arquivo VRML.

Visando facilitar e acelerar a escrita do arquivo VWML, uma interface gráfica é proposta. Nela, o usuário pode atribuir valores aos campos dos diversos componentes presentes na nova linguagem definida. Após a inserção desses valores, a interface gráfica gera um arquivo VWML correto, ou seja, com suas *tags* e valores escritos de forma correspondente às atribuições dadas pelo usuário.

Uma característica adicional a essa interface é a presença de uma ferramenta de assistência, cujo objetivo é gerar um programa em Java com *links* para diversos modelos em 3D,

criados pelo projetista gráfico, de forma simples. A criação dessa ferramenta visa criar um meio de comunicação entre o projetista do modelo 3D e o programador, que pode utilizar-se desses modelos para a construção de alguma aplicação para visualização dos mesmos.

É feito um estudo sobre outras linguagens para o desenvolvimento de gráficos 3D, como o OpenGL[4], OpenSceneGraph [7], Open Inventor [8][9][10] e o X3D [14], objetivando apontar as vantagens e desvantagens delas. Esta análise serve de base para o desenvolvimento da linguagem proposta neste trabalho.

A monografia encontra-se organizada em seis capítulos, incluindo esta introdução. Os capítulos estão divididos como segue:

Capítulo 2 (Conceitos Básicos e Trabalhos Relacionados): O presente capítulo aborda diversos conceitos 3D, visando dar ao leitor uma melhor compreensão de como modelar o mundo 3D. Outras informações a respeito de diversas linguagens de modelagem 3D também estão presentes nesse capítulo.

Capítulo 3 (VWML - Linguagem de Descrição 3D): Neste capítulo, encontra-se uma descrição das diversas características do XML. Sua estrutura é abordada, bem como mecanismos para validação de formatos de documentos XML. As diversas funcionalidades atualmente suportadas pelo VWML são expostas e exemplificadas. O leitor encontra, neste capítulo, uma comparação da descrição de objetos 3D, feita através do VWML, com suas correspondentes em VRML [12][13] e X3D [14], que são duas outras linguagens de descrição 3D abertas. A intenção dessa comparação é mostrar ao leitor as vantagens e desvantagens do VWML de maneira eficaz e segura.

Capítulo 4 (Compilador Construído): Abordam-se aqui conceitos básicos de compilação, dando ao leitor uma noção completa de todos os possíveis componentes de um compilador. As várias tecnologias estudadas para a construção do compilador são citadas neste capítulo. Dentre elas estão: JLex [38][39] e CUP [39][40], JavaCC [42][43], SAX [46][47], DOM [48][49] e JDOM [50][51]. As empregadas para a implementação do compilador são claramente distinguidas. O esquema de compilação da descrição VWML para sua correspondente em VRML está exposto no presente capítulo. Com o esquema, é possível entender todo o processo de tradução bem como descrever formalmente o compilador construído. Um exemplo de código e outro de compilação dão ao leitor uma maior proximidade e compreensão da implementação do compilador.

Capítulo 5 (Interface Gráfica e Programa de Assistência ao Programador): Este capítulo mostra ao leitor as diversas funcionalidades da interface gráfica desenvolvida neste trabalho. A mesma tem como objetivo facilitar e acelerar o desenvolvimento dos modelos 3D para o VWML. As tecnologias empregadas na construção da interface gráfica são abordadas. Um exemplo de modelagem e a descrição em VWML gerada pela interface são mostrados neste capítulo. Outro ponto apresentado neste capítulo é relativo ao programa de assistência ao programador. Nele detalhes de implementação são dados, além do passo-a-passo para se obter o programa gerado por este assistente.

Capítulo 6 (Conclusões e Trabalhos Futuros): São expostas as repercussões do trabalho realizado, sugerindo as possíveis aplicações e trabalhos futuros, além de uma reflexão crítica sobre os resultados obtidos.

Capítulo 2

Conceitos Básicos e Trabalhos Relacionados

Este capítulo tem como objetivo dar ao leitor uma breve noção de alguns conceitos 3D e linguagens de descrição de ambientes 3D, sendo esses de grande importância para os leitores que não tenham conhecimento na área.

2.1 Conceitos 3D

As imagens nos trazem muitas informações a respeito da cena nelas retratadas, sendo a mesma real ou imaginária. A Computação Gráfica [1][2], que é o ramo da Computação que permite a criação, armazenamento e manipulação de modelos de objetos e suas imagens pelo computador, vem avançando de forma bastante rápida. Isso pode ser comprovado através dos filmes, que se utilizam de animações feitas totalmente em computadores, que estão cada vez mais realistas, tornando difícil ao espectador distinguir quais personagens ou objetos são reais ou virtuais.

A Computação Gráfica criou diversas oportunidades de trabalho para profissionais que têm como objetivo retratar, através de imagens, o mundo real ou imaginário. Esses profissionais são designers, modeladores, animadores, iluminadores e programadores. Diversas são as aplicações da Computação Gráfica, desde entretenimento, como jogos e filmes, como também aplicações acadêmicas (simuladores de vôo, simulações médicas e em engenharias) e na indústria e comércio (treinamentos, controle de qualidade, edifícios e carros modelados em computador).

Além das aplicações anteriormente citadas, existem diversas outras que são aquelas em que os objetos de estudo só podem ser vistos através de imagens imaginárias. Diversas são as razões para que nessas áreas não possa haver imagens reais: a impossibilidade física de aproximar do objeto como, por exemplo, imagens de buracos negros; e impossibilidade de retratar imagens através de sistemas de coordenadas acima de 3 dimensões como, por exemplo, a garrafa de Klein [3].

Objetos 3D têm a vantagem de dar ao espectador uma noção mais próxima da realidade, pois possuem não apenas altura e largura, mas também profundidade. Assim, torna-se possível a visualização por diversos ângulos do item em questão. Para a representação de objetos em 3D, a Computação Gráfica utiliza-se de diversos estímulos visuais como por exemplo: perspectiva (diminuição proporcional dos tamanhos dos objetos à medida que se distanciam do observador); oclusão (objetos mais próximos ocultando os mais distantes, caso os dois estejam na mesma linha

de visão do observador); e sombras e sombreamentos (recursos que dão ao observador uma noção da posição do objeto em relação ao solo da cena).

Aplicações em 3D geralmente permitem a interação do usuário com o mundo virtual modelado, ou seja, permitem ao usuário navegar livremente entre os objetos e até ativar animações que simulam um comportamento do objeto.

2.1.1 Desenvolvimento de modelos 3D

São três as formas de criação de modelos 3D [2].

Modelagem matemática

Essa modelagem é gerada através de cálculos matemáticos sobre uma base de dados. Esse tipo de modelagem é geralmente utilizada em simulações de eventos naturais como, por exemplo, furacões e erupções vulcânicas.

Modelagem automática

Nesse tipo de modelagem, sistemas de entrada de dados como *scanners* mapeiam o objeto real no computador, permitindo-o montar um modelo em 3 dimensões do objeto em questão. Esse tipo de modelagem é o mais rápido de todos na geração de imagens 3D.

Modelagem manual

Esse tipo de modelagem é o mais barato e mais antigo utilizado na Computação Gráfica. Nele o modelador fica livre para criar um objeto através de dados por ele inseridos.

Este trabalho tem como base a modelagem manual, por ser ela a mais acessível para os usuários de computadores, em geral, e por isso é destacado a seguir.

2.1.2 Técnicas de modelagem manual

Várias são as técnicas de modelagem manual cabendo, assim, ao modelador escolher a mais conveniente para o objetivo em questão. A seguir, estão algumas das técnicas de modelagem manual e uma breve descrição.

Utilização de primitivas

Esse é um método muito simples de desenvolvimento. Baseia-se na união de diversas formas primitivas em 3D para construir objetos mais complexos.

Essas formas básicas dependem da área de aplicação em questão. As mais genéricas são: cubo, cone, cilindro e esfera. Todas essas formas podem ter as suas características ajustadas como, por exemplo, o tamanho, a posição a direção, dentre outras.

A Figura 2 foi gerada através desse método. Nela esferas e cilindros são utilizados para gerar uma estrutura de uma molécula de água (H_2O).

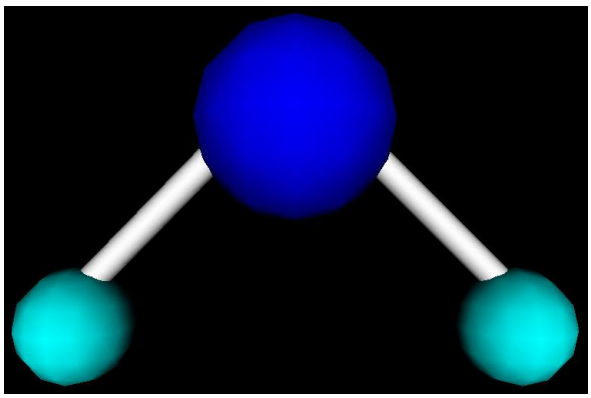

Figura 2. Molécula de água

Connect

Nesse tipo de modelagem, o usuário pode gerar novas formas através da conexão de dois ou mais sólidos existentes na cena. Essa ligação entre os sólidos preenche espaços vazios dando, assim, um aspecto diferente ao modelo em 3D.

A Figura 3 mostra um exemplo de um sólido formado com a técnica de connect.

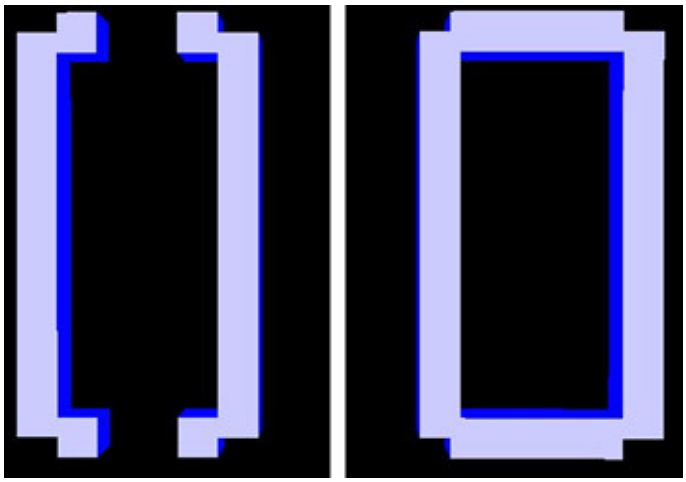

Figura 3. Connect – Sólido da direita gerado a partir da conexão dos sólidos à esquerda

Extrusão

Esse método de modelagem consiste na criação de objetos em 3D através de uma geratriz 2D.

Após a definição de uma geratriz 2D, o usuário pode determinar o percurso que a mesma fará no espaço de coordenadas 3D. Todos os pontos do espaço, por onde a geratriz passar, determinam o sólido em três dimensões.

A Figura 4 é um exemplo de um sólido criado através de extrusão.

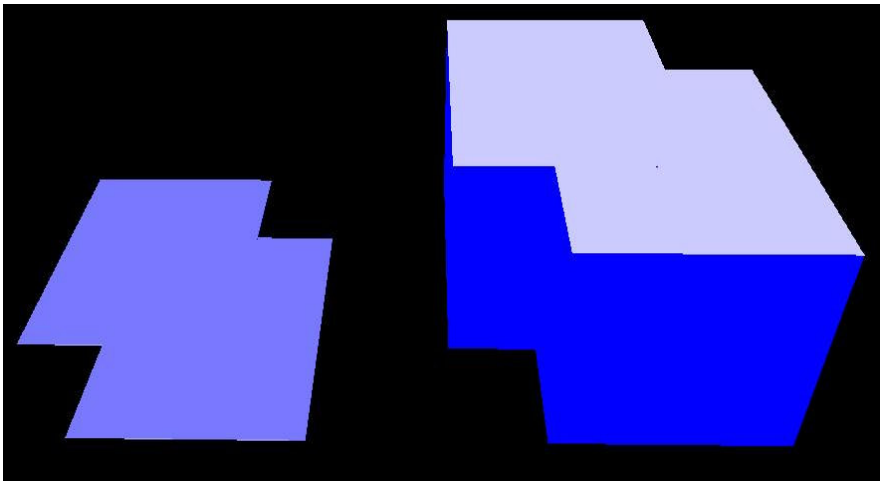

Figura 4. Extrusão – Sólido da direita criado a partir da extrusão do plano à esquerda

2.2 Linguagens de descrição de modelos 3D

As linguagens de descrição servem para fornecer informações ao computador relativas às imagens que o mesmo deve gerar. O armazenamento de imagens no computador pode ser feito através de arquivos contendo a sua descrição através dessas linguagens.

Os atributos das imagens para uma correta síntese são diversos, como por exemplo: forma, posicionamento, material utilizado para cobrir os sólidos (textura), cor, intensidade de luz, transparência, etc.

A seguir são descritas diversas linguagens desenvolvidas para aplicações da computação gráfica.

2.2.1 OpenGL

OpenGL [4] é uma biblioteca de códigos para desenvolvimento de aplicações gráficas, tanto em duas quanto em três dimensões. Inicialmente desenvolvida pela Silicon Graphics, essa biblioteca tem como importantes características a portabilidade, que se dá no fato da utilização da mesma em diversas plataformas de desenvolvimento e análise, e a rapidez na montagem de imagens.

O OpenGL pode ser utilizado por diversas linguagens de programação para o desenvolvimento de aplicações gráficas. Algumas dessas linguagens são: Ada, C++, Fortran, Java, dentre outras, além da linguagem padrão de desenvolvimento da OpenGL, que é a linguagem C.

Além da criação de primitivas gráficas como linhas e polígonos, essa biblioteca permite a manipulação de iluminação, sombreamento, animação, dentre outros artifícios gráficos.

2.2.2 OpenSceneGraph

OpenSceneGraph [7] é um conjunto de ferramentas de código aberto voltadas para o desenvolvimento de aplicações gráficas de alto desempenho, tais como: simuladores de vôo, jogos, realidade virtual ou visualizações científicas.

Esse conjunto de ferramentas tem como características: o fato de ser portável, ou seja, independência de plataforma de desenvolvimento; é um *framework* desenvolvido sobre o OpenGL, possibilitando assim o desenvolvimento de gráficos num nível mais alto de abstração.

Outra característica inerente ao OpenSceneGraph é o fato de se poder, através de uma biblioteca de banco de dados osgDB e uma variedade de *plugins*, ler e escrever dados relativos a gráficos em várias bases de dados e formatos de imagens. Dentre os formatos de base de dados estão o 3D Studio MAX (.3ds) [17] [18], Quake Character Models (.md2) [5], Direct X (.x) [6], VRML (.wrl) [12] [13], e entre os formatos de imagens estão o .gif, .jpg, .png, .bmp, dentre outros.

2.2.3 Open Inventor

Open Inventor [8][9][10], que foi inicialmente desenvolvido pela Silicon Graphics e depois passou a ser de código aberto, é um conjunto de ferramentas 3D orientadas a objeto cuja finalidade é prover soluções a problemas de criação, representação e interatividade de objetos 3D em programações gráficas.

A simplificação da programação de gráficos em 3D é dada através de um modelo de programação baseado numa base de dados de cenas 3D (*scene graph*). Existem objetos pré-definidos, que são amplamente utilizados na construção de ambientes tridimensionais, que possibilitam um mais rápido desenvolvimento e maior potencial de programação 3D.

A Mercury Computer System Inc. [11] adicionou diversas melhorias à versão *open source* do Open Inventor e passou a comercializá-la. Dentre elas estão a detecção de colisão, suporte a VRML 1.0 e VRML 97 , visualização de objetos extremamente grandes e portabilidade.

2.2.4 VRML

Virtual Reality Modeling Language (VRML) [12] surgiu a partir da idéia de visualizar ambientes 3D através da Internet. Esta linguagem contém toda a descrição do modelo 3D em questão e o arquivo contendo o código ocupa pouco espaço na memória do computador, o que possibilita um tráfego rápido pela Internet.

Nas aplicações *web*, que utilizam VRML, têm-se que o usuário pode interagir com o ambiente em 3D, ou seja, movimentar-se através do mesmo e ativar/desativar animações e sons.

A construção do código pode ser feita através de qualquer processador de texto, ou através de ferramentas que possibilitam a criação de mundos em três dimensões.

O arquivo VRML dá ao *plugin* do navegador instruções de como desenhar formas 3D. Essas instruções indicam como construir, onde colocar as formas, que cor elas terão, dentre outras [12].

O conjunto de código do VRML [13] representa uma parte do formato ASCII do Open Inventor mais um conjunto de diretivas para a navegação na web.

2.2.5 X3D

O X3D [14] é um padrão aberto para especificações de ambientes em 3 dimensões, que pode, ao mesmo tempo, definir tanto as formas dos objetos contidos numa cena, quanto os seus comportamentos.

Esse novo padrão vem sendo desenvolvido através dos retornos dados pelos usuários da especificação VRML97 e tem como objetivo incorporar avanços dos recursos gráficos atuais à sua especificação, como também melhorar a arquitetura do padrão dividindo-a em módulos, visando, assim, uma maior flexibilidade e possibilidade de extensão.

No X3D, existem três tipos de codificação de arquivo, dando assim ao usuário uma maior opção de escolha de como desenvolver seu código, o que não há no VRML97. O primeiro tipo é o próprio VRML97. Assim, quem já tem alguma experiência com esta linguagem, não será tão afetado no desenvolvimento de código para esse novo padrão. O segundo é uma codificação em XML, que possibilita uma maior integração com serviços *web* e transferência de dados entre aplicações. Por último, vem a codificação binária que ainda está em fase de desenvolvimento, mas que tem como finalidade principal permitir uma maior largura de banda na transmissão de dados.

Uma característica bastante vantajosa do X3D em relação ao VRML se dá na introdução dos perfis (*profiles*), que são um conjunto de componentes normalmente utilizados para um determinado fim. Assim, o usuário pode especificar em qual perfil se enquadra o seu arquivo e, com isso, exibi-lo em um aplicativo que não precisa suportar todas as funcionalidades do X3D e sim apenas aquelas do perfil escolhido. Informações adicionais, relativas aos perfis atualmente existentes, podem ser obtidas na parte do endereço eletrônico da W3C relativo aos X3D *profiles* [15].

Ao contrário do VRML, que tem uma API (***A****pplication* ***P****rogramming* ***I****nterface*) de script interna e outra externa, o X3D possui apenas uma API, o que possibilita uma mais confiável e robusta implementação.

Atualmente, as especificações desse padrão estão em diferentes níveis de aprovação pela ISO (***I****nternational* ***O****rganization for* ***S****tandardization*) [16].

2.3 Ambientes de desenvolvimento

Existem diversos ambientes para a criação de arquivos contendo a descrição de imagens 3D. A seguir, estão exemplos de ambientes de desenvolvimento e uma breve descrição dos mesmos.

2.3.1 Processador de texto

Nesse ambiente, o usuário descreve a imagem através da inserção manual de código dentro de algum processador de texto. Esses códigos são definidos por uma linguagem de descrição de imagens.

Esse método foi o primeiro a ser utilizado e continua até os dias atuais, sendo o mais apropriado para situações em que não se dispõe de muitos recursos. Porém, é mais difícil de utilizar, pois a forma de descrição é de mais baixo nível, já que o usuário tem de inserir

diretamente o código de descrição da imagem. O modelador fica a mercê do seu conhecimento da linguagem e não possui ferramentas auxiliares para o desenvolvimento.

2.3.2 Ferramentas de modelagem 3D

Estas ferramentas são especialmente desenvolvidas para dar o máximo de auxílio no desenvolvimento de modelos 3D. Possuem diversas funcionalidades, que permitem aos modeladores um rápido e mais fácil desenvolvimento, se comparado à modelagem nos processadores de texto.

A seguir, estão alguns exemplos dessas ferramentas.

3DS Max

O 3DS MAX [17] é um ambiente proprietário de desenvolvimento de aplicações gráficas projetado e desenvolvido pela Discreet Logic Inc. O usuário pode modelar diversos objetos em 3D e fazer animações dos mesmos através de uma interface gráfica bastante intuitiva.

Essa ferramenta é totalmente integrada. Assim, o modelador não precisa mudar de subsistema durante a modelagem 3D [18]. O usuário pode fazer ajustes de câmeras sintéticas, que permitem a gravação de vídeos em diversos ângulos; pode também utilizar-se de trilhas sonoras e ferramentas para sincronização das mesmas com as imagens geradas.

O 3DS MAX foi desenvolvido numa arquitetura orientada a objetos, que fornece uma boa capacidade de expansão e modificações. Diversas funcionalidades podem ser acrescentadas por meio de *plugins*, que também podem ser desenvolvidos pelo usuário.

Outra característica importante se dá no fato de essa ferramenta ser completamente *multi-threaded*, ou seja, para realizar suas tarefas, vários blocos de instruções são gerados e que podem ser executados paralelamente, o que é bastante útil em sistemas multi-processados já que diminui o tempo de resposta do programa.

O 3DS MAX suporta o OpenGL e o Direct3D (API de criação e manipulação de objetos tridimensionais desenvolvida pela Microsoft) que possibilitam mais rápida renderização das imagens criadas.

Blender

O Blender [19] é um programa de código aberto para modelagem em 3D, animações, renderização, criações interativas e *playback*. Disponível para Windows, Linux, Irix, Sun Solaris, FreeBSD ou Mac OS X sob a GNU (GNU's not UNIX) *Public License*.

Com o Blender, é possível desenvolver além de imagens e vídeos a partir de modelos 3D, um ambiente de interatividade em tempo real com os modelos 3D através da instalação de *plugins* para *browsers*.

Existe também o Blender X3D Exporter Project [20] que tem o propósito de desenvolver um exportador de arquivos criados pelo Blender para arquivos X3D no nível de *Interchange profile*. Os projetistas estão trabalhando na adição de suporte aos níveis *Immersive* e *Full profiles* do X3D.

Uma característica adicional dessa ferramenta é a possibilidade da geração de código em VRML. A implementação dessa funcionalidade foi relativamente fácil, devido ao uso do

X3dToVrml97.xsl *style sheet*, desenvolvido pela comunidade X3D. O X3dToVrml97.xsl é um arquivo escrito em XSL (***EX****tensible* ***S****tylesheet* ***L****anguage*), que é uma linguagem para formatar ou transformar documentos escritos em XML.

AutoCAD

O AutoCAD [21] é um conjunto de ferramentas para o desenvolvimento de modelos em 3D, que é amplamente utilizado por engenheiros, urbanistas, construtores, gestores de instalações, instituições de ensino e estudantes, além de outros setores [22].

Apesar de ser bastante utilizado na área de engenharia civil, a ferramenta não é destinada apenas a essa aplicação, podendo ser utilizada em outros setores como, por exemplo, o de entretenimento, onde cenas de filmes são parcial ou totalmente produzidas através dessa ferramenta.

O AutoCAD é uma ferramenta proprietária produzida pela Autodesk, Inc.

MilkShape 3D

Ferramenta inicialmente desenvolvida para modelar personagens do jogo Half-Life [23]. Durante o desenvolvimento do MilkShape 3D [24] novos formatos de arquivos foram adicionados.

MilkShape 3D contém uma interface gráfica de criação de modelos 3D que permite o manuseio de operações básicas como selecionar, mover, modificar a escala do objeto, dentre outras. Formas primitivas como esfera, caixa e cilindro também estão disponíveis.

A ferramenta suporta 37 diferentes formatos de arquivos de 27 jogos. Existem *plugins* que podem ser instalados no programa, dentre os quais estão o X3D Exporter e o VRML 97 Exporter, que permitem a gravação dos arquivos gerados nos formatos X3D e VRML, respectivamente.

2.4 Conclusão

Este capítulo mostrou primeiramente a importância da Computação Gráfica na atualidade, bem como exemplos de sua aplicabilidade.

Foram vistas várias formas de desenvolvimento de imagens tridimensionais, enfatizando a modelagem manual, que é o método mais acessível e barato de modelagem.

Algumas das diversas técnicas para a formação de objetos 3D, através da modelagem manual, foram exemplificadas, sendo elas: a utilização de primitivas, *connect* e *extrusão*.

Para a criação de imagens tridimensionais através da computação gráfica, utilizam-se linguagens específicas para esse fim. Algumas das linguagens de descrição de objetos 3D existentes atualmente foram apresentadas, sendo elas: OpenGL, OpenSceneGraph, Open inventor, VRML e X3D.

As interfaces gráficas para auxílio ao projetista surgiram para facilitar o desenvolvimento manual, que antes era feito apenas através de processadores de texto. Exemplos das interfaces de softwares utilizadas atualmente são: 3DSMax, Blender, Autocad e MilkShape 3D. Com elas, é possível criar e visualizar objetos 3D, de forma rápida e segura, se comparadas com os métodos tradicionais.

Capítulo 3

VWML - Linguagem de Descrição 3D

São várias as linguagens existentes para o propósito de modelagem 3D, porém existem algumas dificuldades que são impostas para o usuário. Uma das principais dificuldades se encontra no fato de a sintaxe de algumas das linguagens ser bastante complexa, dificultando o desenvolvimento manual através de um processador de texto, bem como o transporte de informações relativas ao modelo 3D entre aplicações. Outra dificuldade se encontra no fato de algumas serem proprietárias.

Este trabalho define uma linguagem aberta de descrição de ambientes 3D, visando possibilitar um desenvolvimento mais fácil e intuitivo para o usuário.

Para a definição da linguagem com este propósito foi usado XML. Uma descrição dessa tecnologia é dada a seguir.

3.1 XML

O XML [25] [26], que é derivado do SGML (***S**tandard **G**eneralized **M**arkup **L**anguage*) [27], surgiu no final da década de 90 através de um grupo de empresas e organizações autodenominada de W3C (***W**orld **W**ide **W**eb **C**onsortium*) [28], com o objetivo de criar uma estrutura de informações mais sólida e flexível para as publicações em meio eletrônico. Atualmente, o XML tem uma importância crescente tanto na *web*, quanto em outros ambientes de armazenamento e transferência de dados.

XML não deve ser compreendido apenas como uma linguagem de marcação, mas sim como um conjunto de ferramentas para a criação, modelagem e uso de linguagens de marcação.

3.1.1 Características:

As principais características do XML são listadas a seguir:

- o XML permite que qualquer tipo de dado possa ser armazenado de forma organizada em um formato que venha a facilitar o manuseio;
- o XML é um padrão aberto que possibilita o livre manuseio do mesmo sem a necessidade do pagamento de licenças. Esse padrão é extensível, permitindo que o usuário crie novos elementos estruturais para o armazenamento dos dados;

- existem diversas ferramentas que possibilitam a verificação da correta estruturação de um documento escrito em XML. Nessa verificação podem ser analisados a sintaxe e os tipos de dados usados e, até mesmo, fazer a comparação com outros documentos;
- o texto escrito no formato XML é um texto sem ambigüidade, claro e auto-descritivo, permitindo, assim, uma fácil compreensão do documento;
- devido à sua transparência e simplicidade, o XML permite uma fácil conversão de formatos de dados. Assim, o XML vem sendo bastante utilizado na migração de dados entre aplicações.

3.1.2 Motivações

Sabe-se que o volume de informações existentes na Internet hoje é imenso. Um grande problema, que existe com diversas empresas e corporações, encontra-se na transferência de dados devido à incompatibilidade de formatos. Essas empresas estão com suas informações cada vez mais disponíveis na Internet, o que vem a exigir uma estruturação de dados com facilidade de manipulação, já que os mesmos são atualizados com bastante freqüência.

O HTML (***H****yperText* ***M****arkup* ***L****anguage*) [29] é um padrão bastante utilizado para a disposição de dados na Internet, não possuindo uma estruturação bem definida. Uma versão do HTML baseada em XML está sendo desenvolvida e se chama XHTML (***Ex****tensible* ***H****yperText* ***M****arkup* ***L****anguage*) [30]. Esse novo padrão tem como objetivo utilizar-se de todos os benefícios do XML, permitindo ao seu usuário uma compatibilidade com aplicações passadas e futuras.

3.1.3 Estrutura

O XML possibilita a criação de linguagens de marcações. Essas marcações são informações adicionadas no texto com o objetivo de facilitar a compreensão e manipulação deste.

Uma analogia pode ser feita com jornais e revistas, que contêm marcações como, por exemplo, manchetes, títulos, artigos, etc. Assim, o leitor tem uma melhor noção de como as partes do texto se relacionam entre si.

No XML as marcações são representadas através das *tags*. As *tags* são facilmente reconhecidas no texto, já que são delimitadas pelos símbolos: ‘<’ e ‘/>’.

O criador do XML é livre para definir quantas *tags* forem necessárias para melhor representar o conteúdo do seu documento.

A seguir, na Figura 5, segue um exemplo de um documento escrito em XML e sua explicação.

```
1.  <Documento id=”exemplo1”>
2.     <Pessoa>
3.        <Nome>César Augusto Mendonça de Carvalho</Nome>
4.        <Idade>22</Idade>
5.        < Nível de Escolaridade > 3º Incompleto</ Nível de Escolaridade >
6.     </Pessoa>
7.  </Documento>
```

Figura 5. Exemplo de documento XML

Para esse documento XML, foram criadas *tags* para a representação de informações relativas a uma pessoa. O criador definiu as *tags*: Documento, Pessoa, Nome, Idade e Nível de Escolaridade (linhas 1, 2, 3, 4 e 5 respectivamente). De acordo com a necessidade de informações adicionais novas *tags* podem ser adicionadas no documento.

A *tag* Documento é chamada de "elemento raiz", que é o elemento mais externo e contém todos os outros elementos, que são chamados de "elementos filhos".

O leitor deve observar que existe uma hierarquia entre as *tags*, que deve ser respeitada para que uma correta construção do documento e futura análise possam ser feitas. Exemplificando, caso informações relativas a uma nova pessoa sejam adicionadas, essas devem vir dentro de outra *tag* Pessoa, externa à atualmente existente e interna ao "elemento raiz" Documento.

Na linha 1, deve ser observada a existência de um atributo chamado "id". Os atributos são informações adicionais nas *tags* que dão mais clareza ao elemento. Em geral, os atributos não trazem informações relevantes aos dados, mas sim ao programa que manipula o documento em XML.

3.1.4 Validação

É de extrema importância a existência de mecanismos para a verificação de modelos de documento. Com isto é possível determinar se um dado documento XML pertence ou não a um modelo pré-definido.

A vantagem imediata desse tipo de mecanismo é que documentos validados podem ser submetidos a aplicativos, dando ao usuário a certeza da consistência das informações nele contidas.

A seguir são apresentadas duas possíveis alternativas para validação de documentos XML.

DTD

DTD (**D**ocument **T**ype **D**efinitions) [31] serve para modelar tipos de documentos através da definição de quais elementos (*tags*) e que tipo de informação pode vir neles. A definição de um DTD pode vir dentro do próprio documento XML, ou então, pode estar definido em um arquivo externo com referência para o documento XML em seu cabeçalho.

O XML pode servir para o transporte de informações entre aplicações. Tendo um DTD definido, a aplicação ou qualquer usuário pode utilizá-lo para a validação dos dados que estão sendo transportados. Esta característica acrescenta confiabilidade às informações.

É fácil definir elementos obrigatórios a um documento XML, porém fica difícil mostrar, através de DTD, aleatoriedade na ordem de aparição desses elementos no documento.

Com DTD, atributos padrões podem ser definidos para que os mesmos sejam inseridos pelo processador de XML ao documento em caso de omissão.

Como geralmente o próprio DTD serve de documentação para ele mesmo, é necessária a aplicação de algumas políticas no momento de seu desenvolvimento. Algumas atividades dessa política devem ser: a boa organização do documento, separando-o em módulos funcionais; comentar o que for necessário para facilitar futuras modificações; e separar as diversas declarações para torná-las mais inteligíveis.

XML *Schema*

A motivação para a criação do XML *Schema* [26] foi o fato de muitos usuários do DTD questionarem sobre sua estrutura complexa, além de alegarem inflexibilidade. Outro ponto é a impossibilidade da criação de padrões de dados nos elementos e atributos.

O XML *Schema* [32] tem por finalidade definir uma classe de documentos XML através da restrição de componentes presentes nos mesmos, dos seus significados e relacionamentos. Estas restrições podem atuar sobre os tipos de dados utilizados no documento, elementos (*tags*), que devem ou não estar presentes, e, por último, sobre os atributos e seus valores.

Um uso adicional de um XML *Schema,* sobre os atributos de um documento, está na possibilidade de normalizá-los e atribuir valores-padrão para eles, caso não estejam especificados no documento. Se comparado à DTD, o *Schema* é mais poderoso quanto à verificação de tipos de dados, pois permite também a verificação de erros no conteúdo.

Assim, pode-se dizer que um *Schema* é um formalismo que expressa restrições sintáticas, estruturais e de valores. Por ser baseado em XML, a especificação do *Schema* é auto-explicativa, e pode ser facilmente desenvolvida em um aplicativo de edição de XML.

O *Schema* usa fragmentos de XML [26], que são chamados de modelos, para demonstrar como deve ser exibido um documento.

3.2 VWML

A linguagem criada, aqui denominada de VWML (***V**irtual **W**orld **M**arkup **L**anguage*), focaliza as funcionalidades básicas para a criação de objetos em três dimensões. Essas funcionalidades serão descritas nas Subseções a seguir.

Antes e durante o desenvolvimento dessa linguagem, foram feitos estudos que buscaram determinar quais os principais elementos utilizados no desenvolvimento de objetos 3D. Esses estudos foram bastante importantes, pois possibilitaram a definição do escopo da linguagem. Não seria possível a implementação de todas as funcionalidades presentes nas principais linguagens encontradas no mercado, devido ao período de tempo alocado para o seu desenvolvimento.

A seguir, são apresentadas as funcionalidades suportadas pela linguagem VWML. Saliente-se que, por ser baseada em XML, esta linguagem pode ser facilmente estendida para suportar outras funcionalidades.

3.2.1 Formas Básicas

Diversas formas básicas podem ser descritas pela linguagem desenvolvida. A utilização delas representa um dos principais e mais fáceis métodos de desenvolvimento 3D. A união dessas formas possibilita a geração de outras formas mais complexas.

Cada forma básica teve suas características bem definidas através das *tags* do XML. Assim, para diferentes formas, existem diferentes *tags*. Para ficar mais claro para o leitor segue o exemplo que mostra a definição de uma esfera e um cilindro. Devem ser observadas as *tags* que caracterizam cada uma das formas.

Definição de uma esfera.	Definição de um cilindro.
1. </sphere> 2. <radius>1.0</radius> 3. </sphere>	1. <cylinder id="nome"> 2. <radius>3.0</radius> 3. <height>2.0</height> 4. <top>TRUE</top> 5. <bottom>TRUE</bottom> 6. <side>TRUE</side> 7. </cylinder>

Figura 6. Definição de uma esfera

Figura 7. Definição de um cilindro

Como se pode observar, a esfera só precisa da informação do seu raio (linha 2 da Figura 6) para uma completa definição. Para o cilindro são necessárias mais informações, pois além do raio (linha 2 da Figura 7), é preciso ter informações sobre a altura do cilindro e quais superfícies serão visíveis como: os tampões superior e inferior (linhas 4 e 5 da Figura 7) e a lateral do cilindro (linha 6 da Figura 7).

Posicionamento

As diversas formas básicas podem ser colocadas em diferentes posições no "mundo 3D". Assim, pode-se ter formas internas, próximas ou distantes umas das outras. Essa funcionalidade possibilita uma retratação do objeto real ou imaginário com um maior grau de fidelidade.

Tags relacionadas à posição do objeto através de coordenadas em três dimensões foram adicionadas a cada descrição de forma básica. Caso o usuário queira que o objeto fique posicionado na origem do sistema de coordenadas (0, 0, 0), basta omitir a *tag* de posicionamento.

A seguir, está um exemplo de uma esfera deslocada 1 unidade no eixo X (linha 4 da Figura 8) do sistema de coordenadas.

```
1    <sphere>
2      <radius>1.0</radius>
3      <position>
4        <x>1</x>
5        <y>0</y>
6        <z>0</z>
7      </position>
8    </sphere>
```

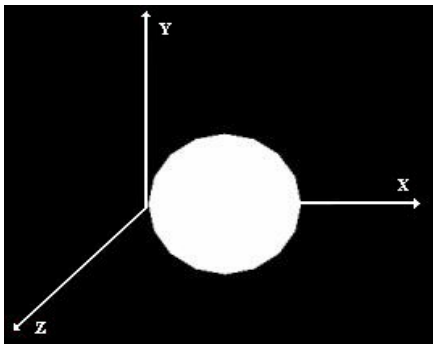

Figura 8. Definição de uma esfera deslocada no eixo X

Material

Além da definição da forma e do posicionamento, é necessária a descrição da aparência do objeto 3D para uma boa apresentação visual. É bastante importante a diferenciação de cores, brilhos e transparência dos objetos, para que seja mais fácil percebê-los na cena e para que sejam mais fiéis ao modelo original.

Para a função de definição da aparência do objeto definido no XML, foi criada a *tag* 'material', na qual podem ser definidas diversas características relativas à aparência da forma em

3D como: intensidade do ambiente, cor difusa, cor emissiva, claridade, cor especular e transparência.

Para a definição das cores, foi usado o padrão de cores RGB, que é o mais utilizado atualmente. Nesse sistema, a cor resultante é definida pela mistura de tonalidades de vermelho, verde e azul. O total de tonalidades é de 256 (valores de 0 a 255) para cada componente RGB, sendo, assim, possível representar 16.777.216 (256^3) cores.

A definição de um cone de cor vermelha difusa (linhas 4, 5 e 6 da Figura 9) e 50% de transparência (linha 8 da Figura 9) é dada a seguir.

```
1.  <cone id="Cone Vermelho">
2.      <material>
3.          <diffuseColor>
4.              <r>255</r>
5.              <g>0</g>
6.              <b>0</b>
7.          </diffuseColor>
8.          <transparency>0.5</transparency>
9.      </material>
10.     <bottomRadius>2</bottomRadius>
11.     <height>5</height>
12. </cone>
```

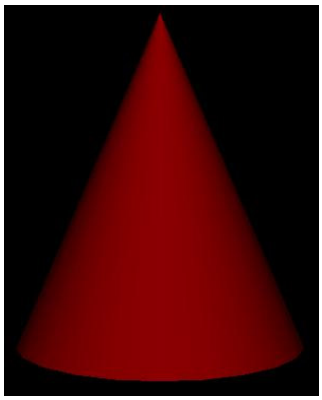

Figura 9. Definição de um cone com características de cor e transparência

Existe, ainda, a possibilidade do uso de imagens e vídeos pré-existentes como texturas a serem usadas nas formas básicas. Dessa maneira, esses elementos servirão para encobrir o objeto que está sendo definido. Informações sobre a sintaxe para inclusão de texturas podem ser obtidas no Apêndice A.

3.2.2 Group

A *tag* ‘group’ possibilita que um conjunto de objetos sejam agrupados. Esse agrupamento é muito útil quando se deseja criar formas complexas através da combinação de outras mais simples.

Assim, um objeto gerado pela união de outros objetos, união esta representada através de um ‘group’, pode ser referenciado e reutilizado apenas através do nome do grupo de objetos.

Além de formas básicas, a *tag* ‘group’ pode ter como sub-elementos outros grupos.

A Figura 10 apresenta um exemplo de utilização dessa *tag*.

```
1.  <group id="gota">
2.     <cone id="cone1">
3.        <position>
4.           <x>0</x>
5.           <y>3</y>
6.           <z>0</z>
7.        </position>
8.        <bottomRadius>1.89</bottomRadius>
9.        <height>5</height>
10.    </cone>
11.    <sphere>
12.       <radius>2.0</radius>
13.    </sphere>
14. </group>
```

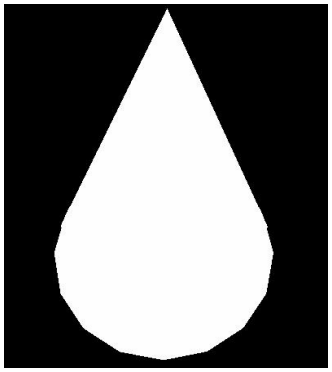

Figura 10. Exemplo de agrupamento de elementos

No exemplo da Figura 10, houve a união de duas formas básicas (cone – linhas 2 a 10 - e esfera – linhas 11 a 13) para gerar uma forma mais complexa de uma gota.

3.2.3 Transform

A *tag* 'transform'serve para criar um novo sistema de coordenadas (x,y,z) com origem deslocada em relação ao sistema de coordenadas principal. Todas as formas 3D criadas dentro dessa *tag* terão posição relativa a esse novo sistema de coordenadas.

Essa funcionalidade é bastante útil, pois permite a criação de grupos de objetos de forma independente com posições relativas entre si. Para que esses grupos se aproximem ou se distanciem, basta modificar a origem dos respectivos sistemas de coordenadas.

As *tags* internas à *transform* são: *translations* (define a posição da origem do novo sistema de coordenadas), *rotation* (define a rotação do novo sistema de coordenadas em relação aos eixos do sistema de coordenadas principal) e *scalling* (modifica a escala de unidade usada no sistema de coordenadas principal).

A Figura 11 apresenta o exemplo de um cilindro criado no sistema de coordenadas principal (linhas 1 a 11) e outro criado a partir desse com rotação e translação (linhas 12 a 27).

O atributo 'id' é de uso opcional. O uso desse atributo possibilita a reutilização de uma forma previamente definida em outros locais de um mesmo arquivo XML. Na linha 14 da Figura 11 segue um exemplo de reutilização.

Da linha 16 a 19, está a definição da origem do novo objeto resultante da translação do original. Da linha 21 a 24, está a definição da rotação do objeto transladado, sendo a linha 25 o ângulo em radianos, e as linhas 22 a 24 representando um fator de multiplicação de rotação para cada eixo de coordenada.

```
1.  <cylinder id="cilindro">
2.      <material>
3.          <diffuseColor>
4.              <r>0</r>
5.              <g>50</g>
6.              <b>255</b>
7.          </diffuseColor>
8.      </material>
9.      <radius>2</radius>
10.     <height>10</height>
11. </cylinder>
```

```
12. <transform>
13.     <use>
14.         <source>cilindro</source>
15.     </use>
16.     <translation>
17.         <x>5</x>
18.         <y>0</y>
19.         <z>0</z>
20.     </translation>
21.     <rotation>
22.         <x>2</x>
23.         <y>0</y>
24.         <z>1</z>
25.       <degree>0.6</degree>
26.     </rotation>
27. </transform>
```

Figura 11. Exemplo de uso da tag transform

Na Figura 12, encontra-se a imagem dos cilindros criados através da descrição presente na Figura 11.

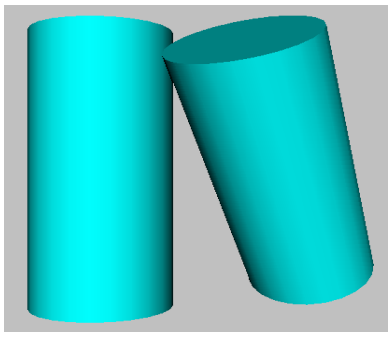

Figura 12. Cilindros resultantes do uso da *tag* transform

3.3 Comparação com o VRML

Para efeito comparativo, encontra-se na Figura 13 um código escrito em VRML, para modelar os mesmos cilindros anteriormente descritos, no qual um é resultante de rotação e translação do outro.

Como o leitor pode perceber, a quantidade de linhas para a descrição dos mesmos cilindros é maior no XML (Figura 11). Porém, cada informação presente nesta descrição é mais clara e organizada do que as presentes no VRML assim, torna-se mais fácil compreender as informações da linguagem criada.

Por exemplo, a linha 16 da Figura 13 está representada pelas linhas 21 a 26 da Figura 12. O leitor pode perceber que o tipo de informação contida nelas está mais explícito que o presente na descrição VRML.

```
1.  #VRML V2.0 utf8
2.  DEF cilindro Shape {
3.     appearance Appearance {
4.        material Material {
5.           diffuseColor 0 50 255
6.        }
7.     }
8.     geometry Cylinder {
9.        radius 2
10.       height 10
11.    }
12. }
13. Transform {
14.    children USE cilindro
15.    translation 5 0 0
16.    rotation 2 0 1 0.6
17. }
```

Figura 13. Código comparativo com o VRML

Comparando os códigos das duas linguagens, tem-se que no VRML a forma básica de um cilindro é um item do elemento "*Shape*"; assim, é necessário incluir as palavras "*geometry Cylinder*" (linha 8 da Figura 13) além do uso da palavra *Shape* (linha 2 da Figura 13). Já na linguagem proposta por este trabalho, as formas básicas possuem *tags* próprias eliminando o uso desses termos adicionais anteriormente descritos. Essa característica permite um mais rápido e fácil desenvolvimento da descrição do objeto 3D, pois evita a inserção de dados sem relevância, além de deixar o código mais claro.

O atributo "id" e a *tag* "use" presentes respectivamente, nas linhas 1 e 13 da Figura 11, estão representadas pelas palavras DEF e USE nas linhas 2 e 14 da Figura 13. O uso dessas palavras têm o mesmo objetivo que é de reutilização de formas ou materiais anteriormente definidos.

Como vantagem adicional, tem-se que o uso de XML para a descrição de modelos em 3D possibilita uma mais fácil leitura para o computador desses dados, pois são baseados em *tags* e existem diversos processadores para este tipo de documento (por exemplo, SAX e DOM, que serão explicados mais adiante no capítulo 4), que possibilitam uma fácil coleta de dados. A integração com outros programas também foi facilitada devido ao uso do XML.

3.4 Comparação com o X3D

A Figura 14 mostra a descrição de dois cilindros, com translação e rotação entre si, através do X3D com codificação em XML. Como o leitor pode ver, as informações estão estruturadas, porém a quantidade de níveis para as diversas informações é pequena sobrecarregando algumas *tags* XML. Outrossim, há a falta de informações para explicar mais

claramente os valores de alguns atributos, como pode ser visto, por exemplo, no atributo '*rotation*' na linha 11.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <X3D>
3.  <Scene>
4.      <Group >
5.          <Shape  DEF="Column">
6.              <Cylinder  height="10.0" radius="2.0"/>
7.               <Appearance >
8.                  <Material  diffuseColor="0 0.19 1"/>
9.               </Appearance>
10.         </Shape>
11.         <Transform  translation="20 0.0 0.0" rotation="0.0 0.0 1.0  3.5">
12.             <Transform  DEF="cyl1">
13.                 <Shape  USE="Column"/>
14.             </Transform>
15.         </Transform>
16.     </Group>
17. </Scene>
18. </X3D>
```

Figura 14. Descrição de dois cilindro paralelos através do X3D

3.5 Tag Raiz

Como foi visto neste capítulo, diversas são as *tags* existentes na linguagem proposta. Porém, não foi citada qual seria a *tag* raiz, que englobaria todas as outras.

A *tag* raiz foi criada visando dar a indicação da linguagem de destino (maiores explicações no próximo capítulo). Por essa razão, no caso específico deste trabalho, a *tag* <VRML> foi utilizada para determinar que a descrição XML deve ser traduzida para o formato VRML.

Esta nomenclatura será empregada em implementações futuras, nas quais o compilador fará a tradução dependendo do tipo da *tag* raiz.

3.6 Conclusão

Várias são as linguagens para descrição de ambientes 3D. Alguma delas impõem barreiras à sua utilização. Por exemplo, algumas são proprietárias, ou apresentam uma sintaxe pouco intuitiva para projetistas.

Tendo em vista essa realidade, este trabalho objetiva criar uma linguagem aberta para descrição de ambientes 3D, com uma sintaxe simples, intuitiva, baseada em formatos de amplo uso, e com facilidades para processamento automático.

Com base nestes princípios, a linguagem em questão, aqui denominada VWML (***V**irtual **W**orld **M**arkup **L**anguage*), é baseada em XML. Fato este, que trouxe vários benefícios, dentre eles, a clareza do seu código e a estruturação das informações, o que facilita o transporte de dados entre aplicações.

Quesitos como motivação, características, estrutura e validação do XML foram explorados e comentados neste capítulo. Diversas funcionalidades da linguagem, como formas básicas, agrupamento e transformação, foram detalhados e exemplificados.

Este capítulo mostra, ainda, uma comparação entre VRML e a linguagem VWRL.

Capítulo 4

Compilador Construído

Para que o usuário da nova linguagem possa analisar os resultados da sua descrição, faz-se importante uma maneira para visualizar e interagir com ambientes 3D.

Devido à existência de inúmeros *plugins* de visualização, principalmente para a linguagem VRML, e visando a acelerar o desenvolvimento do ambiente para a nova linguagem, foi criado um compilador para traduzir modelos desenvolvidos na linguagem para o VRML e, assim, reutilizar visualizadores já existentes.

4.1 Compilador

Por definição, um compilador [33][34] é um programa utilizado para traduzir (converter) um conjunto de informações descritas em uma linguagem-fonte para uma linguagem-objeto, obedecendo a um conjunto de regras e semântica.

É importante salientar que o compilador relata os erros presentes no programa-fonte ao usuário durante o processo de tradução.

No modelo descrito na Figura 15 tem-se que: um programa-fonte (P_f) descrito em uma linguagem-fonte (L_f) é inserido como entrada no compilador (C). Durante o processo de tradução, eventuais erros são informados através de mensagens. Após o processo de compilação, o compilador gera um programa-objeto (P_o) descrito na linguagem-objeto (L_o).

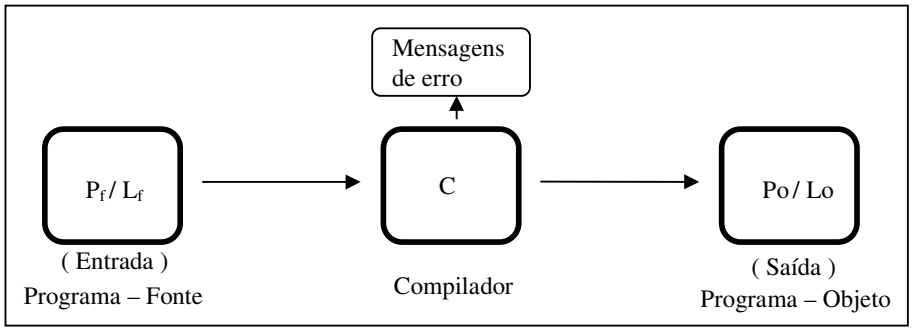

Figura 15. Modelo do processo de compilação

Os primeiros compiladores surgiram no início da década de 50. É bastante difícil precisar a data de surgimento do primeiro compilador, já que trabalhos em diversos locais foram desenvolvidos independentemente. Esses trabalhos iniciais foram de bastante valia, pois desde

então foram descobertas e desenvolvidas diversas técnicas para tratamento de tarefas envolvidas no processo de compilação, que são ainda utilizadas nos dias atuais.

John Backus [35] é um dos grandes nomes que contribuíram significativamente para o desenvolvimento das técnicas nessa área. O Fortran (**FOR**mula **TRAN**slator) [36] foi um projeto da IBM (**I**nternational **B**usiness **M**achines Corporation) liderado por John Backus, e é considerado como a primeira linguagem de programação de alto nível. Obviamente, foi necessária a construção de um compilador para traduzir as descrições das fórmulas matemáticas descritas em Fortran para a linguagem da máquina que iria executá-las. Outra grande contribuição dada por ele foi a BNF (***B**ackus **N**aur **F**orm*) que é uma linguagem para descrever gramáticas de linguagens de programação formalmente e sem ambigüidade. A BNF é ainda utilizada nos dias atuais.

4.1.1 Estrutura

De modo geral, existem duas fases durante a compilação: a fase de análise e a fase de síntese.

Na fase de análise é criada uma representação intermediária do programa-fonte. Os analisadores léxico, sintático e semântico são responsáveis por esta etapa na compilação.

Na fase de síntese, a representação intermediária é analisada e otimizada para, então, ser construído o programa-objeto correspondente. O otimizador e gerador de código formam as partes responsáveis por esta etapa de compilação.

Analisador Léxico

O analisador léxico (ou *scanning*) é a parte do compilador que lida diretamente com os caracteres presentes no código-fonte (código que descreve o programa-fonte).

A função principal desse módulo do compilador é ler, caractere a caractere, da esquerda para a direita, o código-fonte e retornar o *token* (símbolo utilizado para abstrair um conjunto de símbolos de um determinado tipo na linguagem-fonte. Ex.: número, identificador, etc.) mais adequado para a seqüência.

Para toda linguagem de programação existem conjuntos de caracteres (*tokens*) que são tidos como válidos ou inválidos. Quando uma seqüência de caracteres não é válida, o analisador léxico detecta e informa a sua presença.

Analisador sintático

O analisador sintático (ou *parser*) lida diretamente com os *tokens* fornecidos pelo analisador léxico. Os *tokens* são agrupados de forma hierárquica, de acordo como aparecem no texto, formando frases gramaticais que são usadas pelo compilador para a síntese do programa-objeto.

O analisador sintático [34] é responsável por detectar e informar erros sintáticos, que são construções de sentenças que não estão de acordo com as regras gramaticais da linguagem-fonte. Após a detecção de erro sintático, o analisador deve emitir uma mensagem de erro e continuar a análise do resto do código-fonte. Essa parte do compilador deve estar preparada para que um erro detectado interfira o mínimo possível com o resto do programa. Ou seja, que a presença de um erro não interfira no processo de análise do restante do código.

Analisador semântico

O analisador semântico é a porção do compilador responsável por verificações de semântica no programa-fonte. Nele, a estrutura hierárquica gerada pelo analisador sintático é analisada objetivando o reconhecimento dos operadores e operandos das sentenças gramaticais.

A verificação de tipos é uma tarefa realizada pelo analisador semântico. Nela, é verificado se para cada operador são atribuídos operandos com o tipo de dados permitidos pelas regras gramaticais da linguagem-fonte. Por exemplo, para um identificador de variável 'i', que foi definido com o tipo de inteiro, a atribuição 'i = 5' está correta. Porém, caso o usuário insira 'i = "texto"' haverá um erro semântico, pois o tipo de dado não corresponde ao tipo do identificador 'i'.

Gerador de código intermediário

Esse módulo existe em alguns compiladores e age após a fase de análise do código-fonte.

A finalidade do gerador de código intermediário [33] é gerar uma representação do código-fonte, que irá facilitar a tradução para o código-objeto.

Para a geração dessa representação intermediária, verificações de fluxo (ordem das ações) de controle e chamadas a procedimentos são feitas pelo compilador.

Otimizador de código

O otimizador de código [37], que é um módulo presente na maioria dos compiladores, visa à melhoria do código intermediário para que, a partir desse, seja possível gerar um código-objeto que execute no menor tempo possível e/ou utilize pouco espaço na memória.

É importante salientar que a funcionalidade do código é mantida durante a otimização, e a saída do otimizador de código é, ainda, uma representação intermediária do programa-fonte.

Como exemplo de otimização de código, pode-se ter a retirada de regiões do programa (comandos ou funções) que nunca possam ser alcançadas durante a sua execução devido à lógica usada na programação.

Gerador de código

O gerador de código, tem como funcionalidade a geração do código-objeto (escrito geralmente em código de máquina). Todo o código intermediário é traduzido em seqüências de instruções de máquina, mantendo a sua funcionalidade.

Nessa fase, o que há de mais importante é o fato de haver as alocações das variáveis aos registradores (dispositivo que armazena informações) da máquina. Essas alocações devem ser realizadas para facilitar a manipulação dos dados do programa durante a sua execução.

Essa etapa é uma das mais complexas durante a compilação, pois é difícil projetar este módulo de forma que o código-objeto seja o mais eficiente possível.

Associado ao gerador de código, geralmente existem otimizadores que estão intimamente ligados à máquina alvo de execução do programa. Esses otimizadores visam a aproveitar ao máximo os recursos disponíveis na arquitetura específica da máquina em questão.

A Figura 16 mostra todas as fases existentes durante o processo de compilação. Nela, pode-se ver quais módulos do compilador estão na fase de análise e síntese. Pode-se notar

também o que é passado como entrada para cada módulo, e o que é gerado por ele e colocado na sua saída. Por exemplo, o gerador de código intermediário recebe como entrada uma árvore sintática (que foi verificada pelo analisador semântico) e gera como saída um código intermediário, que será a entrada para o otimizador de código.

Ainda na Figura 16, pode-se verificar a existência do gerenciador da tabela de símbolos, que é responsável pelo registro de identificadores usados no código-fonte, e informações associadas a ele como: tipo, escopo de utilização e memória alocada para o seu armazenamento.

Outro módulo também existente nessa figura, é o manipulador de erros, que é chamado a cada vez que um erro é encontrado no código-fonte. Esse módulo deve informar ao usuário a existência do erro, e também ajustar o compilador para que o processo de compilação seja continuado com a menor interferência possível do erro encontrado, mesmo que não seja possível gerar o código-objeto.

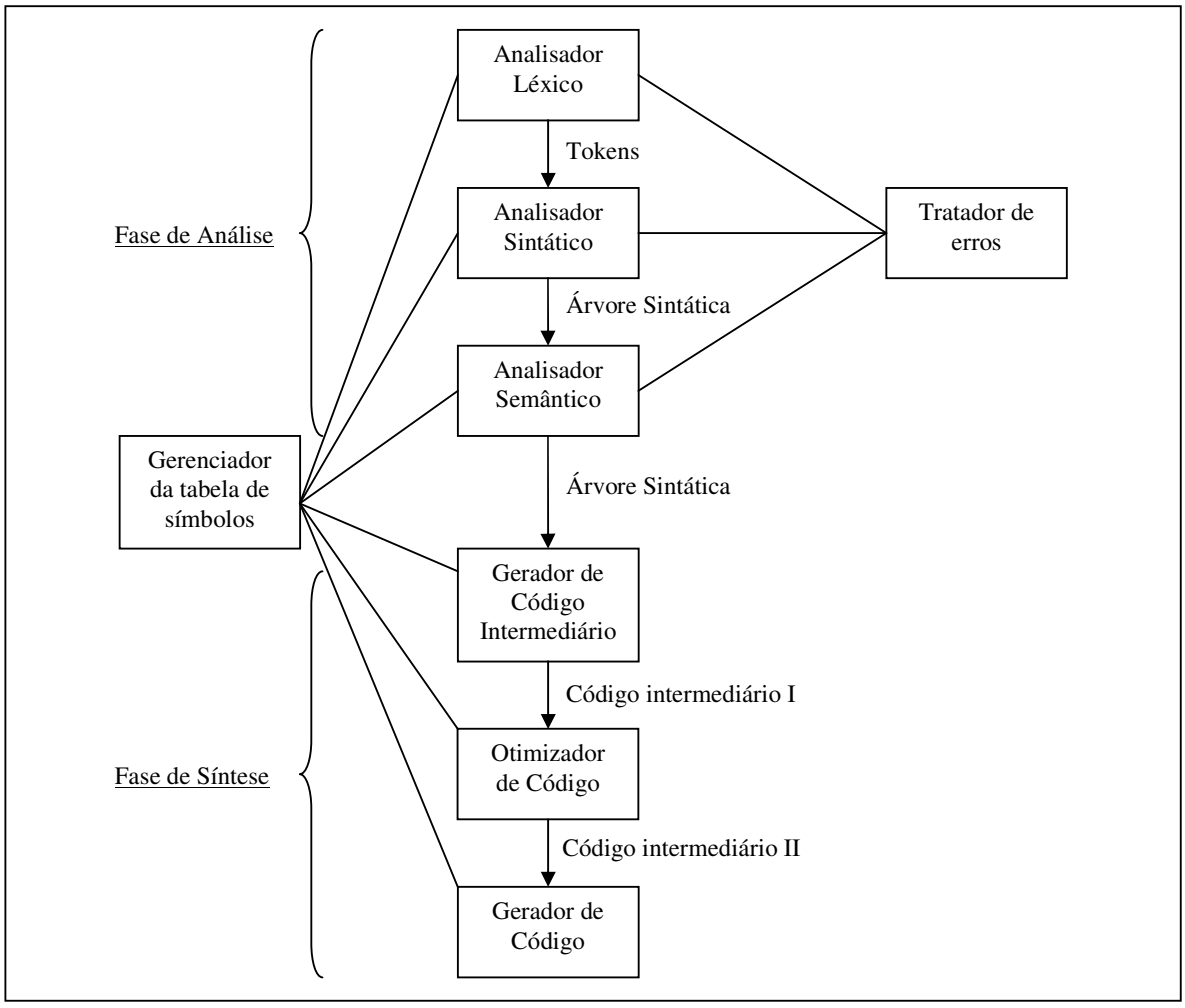

Figura 16. Fases do processo de compilação

4.2 Implementação

O compilador implementado por esse trabalho não possui todos os possíveis módulos que foram anteriormente descritos.

A razão para tal é que o compilador em questão não objetiva a geração de um programa que será executado, e sim a tradução de um código-fonte descrito em XML para a linguagem-alvo VRML, não havendo, portanto, a necessidade do analisador semântico, gerador intermediário de código e otimizador de código.

Os módulos de compilador necessários aqui, são: analisadores léxico e sintático, utilizados para a leitura do XML; e o gerador de código, que criará o documento contendo o código VRML correspondente.

A seguir, são descritas várias ferramentas/tecnologias que foram estudadas para a implementação do compilador.

4.2.1 Linguagem de programação

Primeiramente, foi feita uma análise para decidir qual linguagem de programação utilizar para a construção do compilador capaz de traduzir o código-fonte, escrito na linguagem XML, para o código-destino, escrito em VRML.

Dentre as possibilidades, a linguagem Java foi a escolhida devido ao fato de a mesma proporcionar diversas vantagens como, por exemplo, portabilidade (independência de plataforma), robustez (desenvolvimento de um código com uma alta confiabilidade, dando uma maior segurança para o programador e o usuário), API de fácil uso e disponibilidade, além de outras características.

4.2.2 Validação

Foi desenvolvido um XML Schema para a validação de todos os documentos contendo a descrição do ambiente 3D.

O propósito da validação é certificar que os documentos XML, passados para o compilador, contivessem informações corretamente escritas e válidas (sintaxe correspondente à linguagem VWML e com valores dentro dos limites impostos pela mesma).

Sabendo-se que documentos inválidos não são passados para o compilador, o desenvolvimento do mesmo pôde ser mais objetivo, pois as excessivas verificações de consistência não foram necessárias sobre as informações dos documentos que contêm as informações do(s) objeto(s) em 3D.

4.2.3 Leitura do XML

Tendo definido a linguagem de programação a ser usada para implementar o compilador, foram feitos estudos para a escolha da(s) ferramenta(s) Java para gerar os analisadores léxico e sintático capazes de ler o arquivo XML.

A seguir, temos descrições de algumas das ferramentas estudadas.

JLex e Cup

O JLex [38] [39] é um gerador de analisador léxico para Java baseado no Lex [41] (gerador de analisador léxico para a linguagem C).

A entrada do JLex deve ser um arquivo contendo um conjunto de expressões regulares e conjuntos de ações (descritas em Java) associadas a elas, que servem para descrever como devem ser verificados os *tokens* e quais ações tomar ao detectá-los. A saída do JLex será um código-fonte escrito em Java, que é a implementação do analisador léxico correspondente ao arquivo de entrada.

O JCup [39] [40] é um gerador de analisador sintático baseado no YACC [41] (***Y**et **A**nother **C**ompiler **C**ompiler*), que é para a linguagem C.

No arquivo de entrada para o JCup, devem constar todas as possíveis sentenças gramaticais da linguagem-fonte em questão. Como saída, o JCup gera um arquivo contendo o código Java, que implementa o analisador sintático para a gramática em questão.

JavaCC

JavaCC (***Java C**ompiler **C**ompiler*) [42], que foi inicialmente desenvolvido pela Sun (mesmo criador da linguagem de programação Java), é um programa gerador de analisadores léxico e sintático.

Esse programa permite que diversas linguagens sejam definidas através de um sistema de representação semelhante ao EBNF (***E**xtended **B**ackus **N**aur **F**orm*), que é uma linguagem para definição de sentenças gramaticais de linguagens.

Com essa notação, o usuário pode criar uma definição para qualquer linguagem, e colocá-la como entrada para o JavaCC. A saída será um conjunto de código-fonte de classes Java, que implementam um analisador léxico e sintático para a linguagem definida pelo usuário.

Por ser um dos mais populares parsers usados em aplicações Java, diversas ferramentas foram desenvolvidas para auxiliar e facilitar a sua utilização. Além dessas facilidades, existem diversas gramáticas de linguagens de programação definidas para serem usadas no JavaCC [43]. Dentre elas estão a própria Java, C, C++, SQL, VRML, etc.

4.2.4 Parsers XML

Visando a uma mais rápida e fácil implementação, foi feita uma pesquisa para escolher um *parser* XML que melhor atendesse ao objetivo da leitura da linguagem de descrição, que é baseada em XML, desenvolvida neste trabalho.

Parser XML [44] é uma solução bastante usada por programas que manipulam informações em documentos XML. Essa solução consiste numa biblioteca, que tem como principal função a leitura e a checagem de uma correta formatação.

Para a checagem da formatação do documento XML [45] são verificadas algumas regras como: no documento XML deve sempre existir um elemento mais externo (elemento-raiz) aos outros, caso esses existam; toda *tag* de abertura deve ter uma de fechamento; todas as *tags* devem estar corretamente aninhadas.

As aplicações-clientes utilizam-se da API do *parser* para fazer as chamadas de métodos adequadas para receber ou requisitar informações provenientes do documento XML. Com a API, o usuário fica livre de detalhes de complexidade das implementações das diversas funcionalidades do *parser*.

Existem implementações de várias APIs para a manipulação de um documento XML nas diversas linguagens de programação. Cabe ao usuário escolher a API cujas características melhor se encaixem com suas necessidades e características da aplicação [44].

É possível, em alguns casos em que o *parser* não responde corretamente, a substituição do mesmo, sem a necessidade de recompilação do código.

A seguir, tem-se a descrição de dois dos mais populares *parsers* para XML. Eles são o SAX e o DOM. Ambos possuem pontos característicos fortes e fracos. A melhor opção depende das particularidades do programa a ser desenvolvido.

SAX

O SAX (***S**imple **A**PI for **X**ML*) [46] foi originalmente desenvolvido para aplicações em Java. Posteriormente, tornou-se um padrão de API para XML e tem implementações para outras linguagens e ambientes de programação.

SAX é uma API orientada a eventos, que são basicamente de início e fim de uma *tag*. Não existe no SAX classe que modele o documento XML. O que existe é a modelagem do *parser*, fluxo de dados lidos do documento e a interface cliente para funções de *callback* que receberão os dados do *parser*.

Para a manipulação do documento, a aplicação cliente [47] precisa implementar a interface de *callback* cujos métodos são chamados pelo *parser* quando os eventos são disparados. Assim, por exemplo, a cada *tag* de abertura encontrada, o *parser* chamará o método da aplicação cliente que tratará os dados lidos do documento.

A característica de orientação a evento do SAX faz com que ele seja mais rápido e eficiente em termos de memória do que os outros *parsers* que não são orientados a eventos, já que não precisa armazenar todo o documento na memória. Outro benefício trazido por essa característica está no fato de facilitar (ou possibilitar) a verificação de documentos XML com centenas de linhas de código, que se tivessem uma estrutura guardada na memória ocupariam bastante recurso de *hardware*, que nem sempre está disponível.

Por outro lado, o não armazenamento do documento na memória faz com que o usuário tenha que desenvolver uma estrutura de dados própria para o armazenamento do conteúdo do documento. Isso dificulta e retarda o processo de desenvolvimento da aplicação cliente.

DOM

O DOM (***D**ocument **O**bject **M**odel*) [48] criado pela W3C (***W**orld**W**ide **W**eb **C**onsortium*) é uma interface neutra, criada para a leitura e atualização dinâmica de conteúdo, estrutura e estilo de documentos por programas e *scripts*. Por ser uma API de leitura e escrita, o resultado da manipulação do documento pode ser incorporado de volta ao mesmo.

As empresas são livres para gerarem suas próprias implementações. A Sun Microsystems tem suporte para DOM na sua API de processamento Java XML. Outros fornecedores de implementações são: IBM, Oracle e Apache Software Foundation.

Os principais tipos de documentos estruturados onde o DOM é utilizado para a leitura são o XML e o HTML.

A estrutura do DOM o separa em três partes: o *core,* XML, e HTML [49].

- *Core*: define uma estrutura do documento em forma de árvore que possibilita ao usuário percorrer qualquer documento estruturado de forma hierárquica. E assim, pode representar qualquer documento HTML ou XML. A interface core é compacta de pouco poder de manipulação do conteúdo do documento.
- HTML e XML: são especificações de interfaces adicionais que são usadas com a especificação *core* para uma visão mais conveniente dentro do documento. Ou seja, consistem de objetos e métodos que fornecem um acesso direto e mais fácil dentro do tipo específico de documento.

O DOM é dividido oficialmente em três módulos de níveis sucessíveis.

- nível 1: estava completo em outubro de 1998 e suporta XML 1.0 e HTML 4.0 para navegação e manipulação;
- nível 2: terminado em novembro de 2000, estende as funcionalidades do nível 1 com suporte a XML 1.0 com *namespaces*, CSS (***C**ascading **S**tyle **S**heets)*, eventos de interface ao usuário, além de métodos de manipulação em árvore e seus eventos;
- nível 3: foi finalizado em abril de 2004, e acrescenta ao nível 2 as funcionalidades de ler e salvar um documento, suporte a validação de documentos através de modelos de conteúdo (DTD e *Schemas*), além de outras;
- níveis futuros: poderão especificar alguma interface com a possibilidade de suportar sistemas de janela, *multithreading* e sincronização, segurança e repositório.

O DOM cria uma estrutura em forma de árvore para cada documento estruturado lido por ele. Para a leitura e manipulação dos dados do documento, o usuário utiliza métodos presentes no objeto armazenado na memória correspondente ao documento lido.

O fato de ter a representação do documento na memória facilita o acesso randômico a partes separadas do mesmo. Com o SAX, o usuário teria de ter um controle mais complexo para capturar as informações adequadas do documento.

Porém, o DOM impõe restrições ao tamanho do documento a ser lido, pois com a criação do objeto correspondente na memória pode gerar estouro de memória.

JDOM

JDOM (***J**ava **D**ocument **O**bject **M**odel*) [50] é uma biblioteca Java gratuita para o acesso, manipulação e escrita de documentos XML.

O JDOM, assim como o DOM, cria uma representação do documento XML na memória [51], trazendo com isso suas vantagens e desvantagens. A principal diferença entre estas duas tecnologias é que o JDOM não é neutro (independente de linguagem de programação) e sim baseado em Java.

Para o usuário Java, o uso dessa biblioteca vem facilitar bastante a manipulação do documento XML, pois a sua API é familiar à do Java.

Uma característica interessante é que para a construção do objeto na memória correspondente ao documento XML é usado (por padrão) o *SAXBuilder*, que é um *parser* baseado

em SAX. Assim, o JDOM usa o melhor dos dois *parsers*: a velocidade do SAX e a representação do documento na memória para uma mais fácil manipulação proveniente do DOM.

Devido às vantagens e facilidades encontradas no JDOM, o mesmo foi o *parser* utilizado na implementação do compilador citado neste trabalho.

4.2.5 Esquema de compilação

O compilador construído nesse trabalho traduz códigos escritos na linguagem desenvolvida, que é baseada em XML, para a linguagem VRML de acordo com o esquema de compilação apresentado a seguir.

O diagrama presente na Figura 17 apresenta de maneira global todo o processo de compilação. A seguir são descritos os significados dos símbolos utilizados na figura em questão.

Simbologia:

- As **elipses,** presentes na figura em questão, representam códigos de XML. Por exemplo, a elipse de nome "Documento XML" representa todo o documento que contém as *tags* que descrevem o ambiente 3D. Essas *tags* são traduzidas, uma a uma, de acordo com o seu tipo.
- As **setas** representam qual esquema de compilação deve ser utilizado para traduzir a *tag* atual.
- Os **retângulos,** com elipses dentro, indicam a geração de código VRML, que englobará o código gerado a partir das descrições em XML representadas pelas elipses.
- Todos os **círculos** significam que códigos puramente VRML foram gerados pelo esquema de compilação. Esse símbolo representa o fim do processo de tradução de uma determinada *tag*.

O ponto inicial do processo de tradução se encontra na elipse nomeada de Documento XML. Todas as *tags* presentes no documento, contendo a descrição do(s) objeto(s) 3D, são traduzidas a partir desse ponto. Podem existir diferentes caminhos no processo de tradução, já que os mesmos são determinados pelo tipo de *tag* presente no documento.

O que determina o fim do processo de tradução de uma determinada *tag* é a chegada a um dos dois círculos presentes na Figura 17. O fim do processo de compilação é dado quando todas as *tags* forem traduzidas. Assim, o compilador terá gerado um arquivo contendo todas as informações do mundo 3D em VRML.

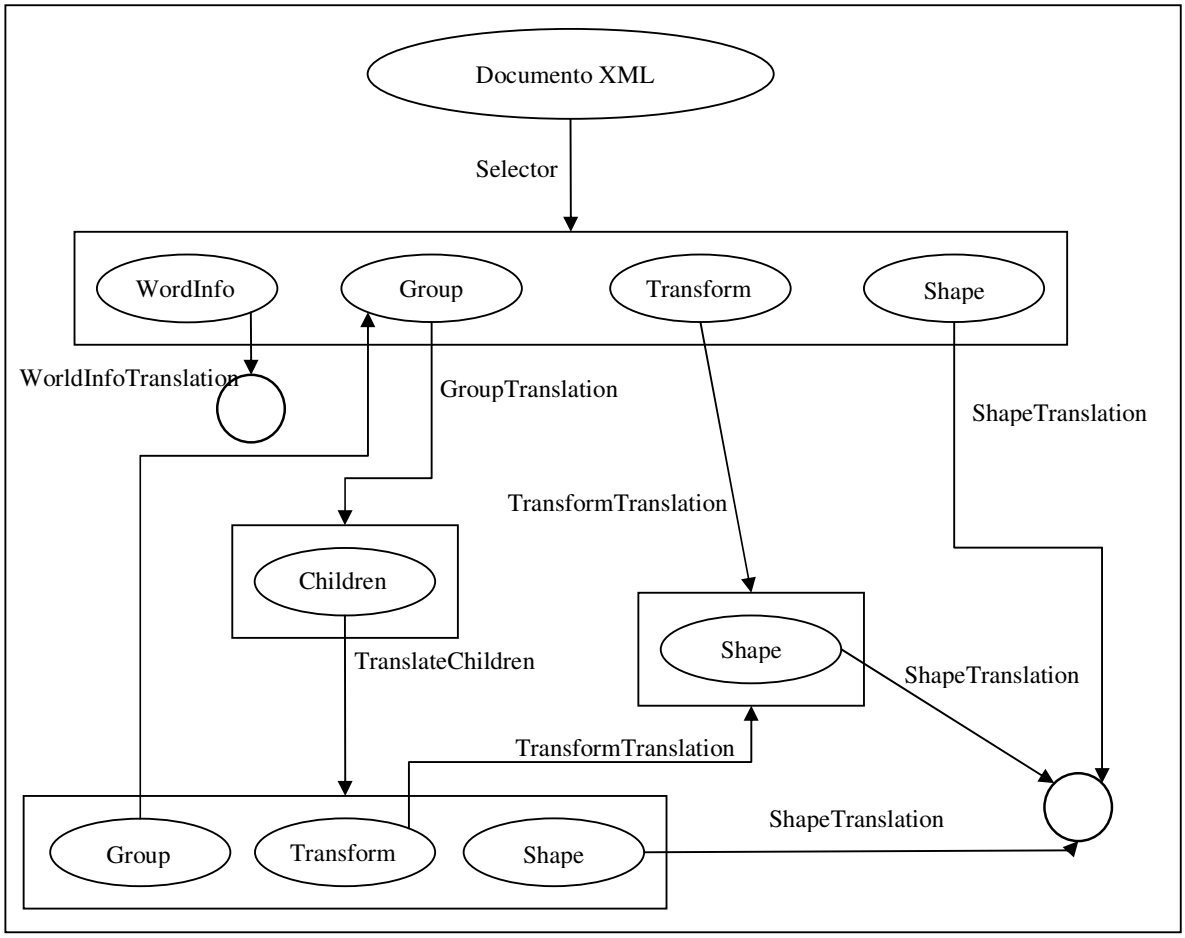

Figura 17. Diagrama do esquema de compilação

Conceitos necessários:

Para um melhor entendimento de todo o esquema de compilação apresentado a seguir, será necessário o conceito de lista e algumas operações relacionadas a ela.

- Uma **Lista** é uma estrutura de dados que contém um número finito de elementos de mesmo tipo unidos de forma seqüencial. Pode haver a inserção, remoção ou reordenação de elementos da lista.
- A **representação de uma lista** é dada pela exposição de todos os identificadores dos elementos que a compõem de forma seqüencial e entre os símbolos '[' e ']'. Por exemplo, os números naturais menores que 10 são: [0,1,2,3,4,5,6,7,8,9].
- ***Head*** é uma função que retorna o primeiro elemento de uma lista sem removê-lo da mesma.
- ***Tail*** é uma função que retorna todos os elementos da lista a partir do segundo, preservando a ordem dos mesmos.
- '**-**' representa a subtração de listas. Assim, na subtração de duas listas, o resultado será todos os elementos da primeira que não estão presentes na segunda. Por exemplo, a subtração da lista A = [0, 1, 2] com a lista B = [2, 3, 4] resulta numa lista C = [0, 1].

- '**+**' representa a união de listas. Assim, na adição de duas listas, o resultado serão todos os elementos da primeira lista acrescidos dos elementos da segunda, mantendo a ordem dos mesmos.
- **Tamanho** retorna o número de elementos que uma lista possui.
- **Proximo** retorna o elemento que está numa posição consecutiva à do elemento atual.

Esquema *Compile* – é o ponto inicial para a tradução da descrição do ambiente 3D escrito através da linguagem proposta neste trabalho. Todas as *tags* filhas do Elemento-Raiz "VRML" são procuradas e, quando localizadas, são enviadas para o esquema *Selector*. Linhas de comentários em XML são ignoradas pelo compilador.

```
Compile [<VRML> [T1,T2,…,Tn] </VRML>] →
      "#VRML V2.0 utf8\n"
      Selector[T1] Selector[T2]… Selector[Tn]
```

Esquema *Selector* – nesse módulo, os tipos das *tags* são analisados e repassados para o esquema de compilação adequado para a sua tradução. Os tipos das *tags* verificados pelo compilador são '*group*', '*transform*', *worldInfo* e *Shape*.

```
Selector [T] →
            Se T é uma tag 'group' → groupTranslation[T]
            Se T é uma tag 'transform' → transformTranslation[T]
            Se T é uma tag 'wordInfo' → worldInfoTranslation [T]
            Senão → shapeTranslation[T]
```

Esquema *groupTranslation* – esse é o esquema do compilador responsável pela tradução da *tag* '*group*', que contém informações de formas 3D que são agrupadas. Como o uso do "id" é opcional, a geração da definição do grupo (`DEF name`) também será optativa, pois depende da existência desse atributo.

```
groupTranslation [<group id=name> [C1,C2,…,Cn] </group>] →
      "DEF name\n"
      "   Group {\n"
      "          children[\n"
                        translateChildren [C1,C2,..., Cn]
      "                ]\n"
      "  }\n"
```

Esquema *translateChildren* – Nesse esquema de compilação são analisadas, em todas as possíveis *tags*, a existência da *tag* filha '*position*' (*tag* presente apenas nas formas geométricas - *cilinder*, *box*, *sphere*, *cone* - e texto plano - *tag text*).

As *tags* que possuem a definição da posição espacial são agrupadas de acordo com as coordenadas espaciais x, y e z e enviadas para o esquema de compilação *withTransform*. Esse agrupamento serve para que todos os elementos colocados no mesmo ponto do espaço sejam traduzidos juntos, ou seja, um *transform* apenas englobando todas essas formas. O objetivo disso é a diminuição de linhas de código VRML resultante do processo de compilação, já que não vai haver um *transform* para cada elemento do XML.

Para todas as outras *tags* que não foram agrupadas, o seu tipo é verificado e enviado para o esquema de compilação mais adequado: *groupTranslation, transformTranslation ou shapeTranslation*.

Para este esquema de compilação será usada uma função chamada *position* que verifica a posição espacial de uma determinada *tag*. Os valores x, y e z descrevem as coordenadas do ponto no espaço.

```
translateChildren [C1,C2,..., Cn] ->
      Para cada C de [C1,C2,..., Cn]
            Se C não contém a tag 'position'
                  listaSemPosition ^ [C]
            Senão
                  listaComPosition ^ [C]
      Fim Para

      Para cada C da listaSemPosition
            Se C  é uma tag 'group' → groupTranslation[C]
            Se C é uma tag 'transform' → transformTranslation[C]
            Senão → shapeTranslation[C]
      Fim Para

      Enquanto tamanho(listaComPosition) ≠ 0
            P = head(lista)
            listaLocal = [P]
            Se tamanho(tail(listaComPosition)) > 1
                  Para cada elemento de tail (listaComPosition)
                        C = Proximo(lista)
                        Se position(P)  = position (C)
                              Então listaLocal = listaLocal + [C]
                  Fim Para
                  ListaComPosition = ListaComPosition - listaLocal
            Fim Se
            withTransform(listaLocal, x, y, z)
      Fim Enquanto
```

Esquema *withTransform* – todas as *tags* recebidas nesse módulo são formas básicas (formas geométricas - *cilinder*, *box*, *sphere*, *cone* - e texto plano - *tag text)* e estão posicionadas no mesmo ponto de coordenada, que é descrito pelos valores de x, y e z.

Todos os elementos dessa lista são enviados para o esquema de compilação ***shapeTranslation***.

withTransform [C_1,C_2,..., Cn,x,y,z] ->

```
"Transform {\n"
"     translation x  y z \n"
```

```
"      children [\n"
       shapeTranslation[C1]shapeTranslation[C2]shapeTranslation[Cn]
"      ]\n"
"}\n"
```

Esquema *shapeTranslation* – as informações relativas às formas básicas, que são definidas através da *tag* '*shape'*, são traduzidas (compiladas) nesse esquema de compilação.

Esse é o maior dos esquemas, devido ao número de *tags* filhas que podem estar presentes em 'shape'.

Primeiro, é verificada a existência da *tag 'position'*. Caso esse esquema de compilação tenha sido chamado pelo esquema ***withTransform,*** a tradução correspondente à *tag position* não é feita, por questão de redundância.

Caso exista a *tag* 'material', algumas informações são traduzidas nesse esquema (relativas às *tags imageTexture* e *movieTexture*), e outras são enviadas para *materialTranslation.*

Em seguida, é verificada a existência de algumas *tags* e a tradução correspondente é feita. Essas *tags* são: *'radius', 'height', 'top', 'bottom', 'side', 'bottomRadius', 'sizeX', 'maxExtent', 'string'* e *'fontStyle'*.

É feita a verificação da existência da *tag 'position'* novamente, e informações adicionais escritas em VRML são colocadas.

Por último é feita a verificação da existência das *tags* '*material'* e '*movieTexture'*. E caso essa última contenha um atributo '*id'*, então informações de som serão adicionadas ao código VRML resultante.

```
shapeTranslation [C] ->
   Se C contém a tag 'position' ->
      "Transform {\n"
            "translation (valor tag filha x) (valor tag filha y) (valor tag
                 filha z)\n"
            "children [\n"
   Se C tem o atributo "id" →
      "DEF (Valor do id)"
   "Shape {\n"
   Se C contém a tag 'material' ->
      "appearance "
      Se a tag 'material' contém a tag 'use' ->
            "USE (conteúdo da tag 'use') \n"
      Senão ->
            Se 'material' tem o atributo "id" → "DEF (Valor do id)"
            "Appearance {\n"
            Se 'material' contém a tag 'imageTexture' ->
                  "texture "
                  Se contém tag id ->"DEF (Valor do id)"
                        "ImageTexture {\n"
                        "url "(conteúdo da tag 'imageTexture')\n"
                        "}\n"
                  Fim Se
                  Se 'material' contém a tag 'movieTexture->
                        "texture "
                        Se contém tag id ->"DEF (Valor do id)"
```

```
                        "MovieTexture {\n"
                        "url "(conteúdo da tag filha 'url')" \n "
                        "loop "(conteúdo da tag filha 'loop')" \n}\n "
                        Fim Se
                  Fim Se
                  materialTranslation[M]
                  "}\n"
   "geometry (valor da tag C) {\n"
   Se C contém a tag 'radius' ->
      "radius (conteúdo da tag 'radius') \n"
   Se C contém a tag 'height' ->
      "height (conteúdo da tag 'height') \n"
   Se C contém a tag 'top' ->
      "top (conteúdo da tag 'top') \n"
   Se C contém a tag 'bottom' ->
      "bottom (conteúdo da tag 'bottom') \n"
   Se C contém a tag 'side' ->
      "side (conteúdo da tag 'side') \n"
   Se C contém a tag 'bottomRadius' ->
      "bottomRadius (conteúdo da tag 'bottomRadius') \n"
   Se C contém a tag 'sizeX' ->
      "size (conteúdo da tag 'sizeX') (conteúdo da tag 'sizeY') (conteúdo da
          tag 'sizeZ')\n"
   Se C contém a tag 'maxExtent' ->
            "maxExtent (conteúdo da tag 'maxExtent') \n"
   Se C contém a tag 'string' ->
            "string ["(conteúdo da tag 'string')"] \n"
   Se C contém a tag 'fontStyle' ->
      fontStyleTranslation[F]
      "}\n"
      "}\n"
   Fim Se
   Se C contém a tag 'position' ->
      "\n] \n"
      "},\n"
   Fim Se
   Se C contém a tag 'material ->
      Se a tag 'material' contém a tag filha 'movieTexture' ->
            Se a tag 'movieTexture' contém atributo 'id'->
                  "Sound {"
                  "source USE (Valor do 'id')"
                  "minFront 30.0"
                  "minBack 30.0"
                  "maxFront 30.0"
                  "maxBack 100.0"
                  "\n"
            Fim Se
      Fim Se
   Fim Se
```

Esquema *materialTranslation* – dados relativos a materiais utilizados nas formas básicas existentes no ambiente 3D são traduzidos nesse esquema.

Nele, é verificada a existência de algumas *tags*, e sua tradução correspondente é gerada.

```
materialTranslation[M] ->
      "material Material {\n"
      Se M contém a tag 'ambientIntensity' ->
```

```
      "ambientIntensity (conteúdo da tag 'ambientIntensity') \n"
Se M contém a tag 'shininess' ->
      "shininess (conteúdo da tag filha 'shininess') \n"
Se M contém a tag 'transparency' ->
      "transparency (conteúdo da tag filha 'transparency') \n"
Se C contém a tag 'diffuseColor' ->
      "diffuseColor (conteúdo da tag filha 'r') (cont. da tag filha 'g')
          (cont. da tag filha 'b')\n"
Se C contém a tag 'emissiveColor' ->
      "emissiveColor (conteúdo da tag filha 'r') (cont. da tag filha
          'g') (cont. da tag filha 'b')\n"
Se C contém a tag 'specularColor' ->
      "specularColor (conteúdo da tag filha 'r') (cont. da tag filha
           'g') (cont. da tag filha 'b')\n"
"}\n"
```

Esquema *fontStyleTranslation* – informações dos estilos das *fonts* utilizadas da forma básica *text,* são traduzidas por este esquema.

Diversas *tags* filhas da *tag* principal 'fontStyle' são procuradas e, caso sejam achadas, é gerada a tradução correspondente. Essas *tags* são: '*family*', '*justify*', '*horizontal*', '*leftToRight*', '*size*', '*spacing*', '*style*' e '*topToBottom*'.

fontStyleTranslation[F] ->

```
"fontStyle "
Se 'F' tem o atributo "id" → "DEF (Valor do id)"
"FontStyle {\n"
Se 'F' contém a tag 'family' ->
      "family "(conteúdo da tag 'family')" \n"
Se 'F' contém a tag 'justify' ->
      "justify "(conteúdo da tag 'justify')" \n"
Se 'F' contém a tag 'horizontal' ->
      "horizontal "(conteúdo da tag 'horizontal')" \n"
Se 'F' contém a tag 'leftToRight' ->
      "leftToRight "(conteúdo da tag 'leftToRight')" \n"
Se 'F' contém a tag 'size' ->
      "size "(conteúdo da tag 'size')" \n"
Se 'F' contém a tag 'spacing' ->
      "spacing "(conteúdo da tag 'spacing')" \n"
Se 'F' contém a tag 'style' ->
      "style "(conteúdo da tag 'style')" \n"
Se 'F' contém a tag 'topToBottom' ->
      "topToBottom "(conteúdo da tag 'topToBottom')" \n"
"}\n"
```

Esquema *transformTranslation* – esse esquema é responsável pela tradução de informações relativas à translação, rotação e escala dos objetos a partir do sistema de coordenadas principal do mundo 3D.

Um destaque para esse esquema é a verificação da presença da *tag* '*use*', que visa a reutilização de formas básicas previamente definidas no documento XML.

transformTranslation [T] ->

```
Se T tem o atributo "id" → "DEF (Valor do id)"
"Transform {\n"
Se T contém a tag "def" ->
      "children [\n"
      shapeTranslation[S1] shapeTranslation[S2] shapeTranslation … [Sn]
      ]\n"
Se T contém a tag "use" ->
      Para todas as tag filhas ->
            "children USE (conteúdo da tag filha 'source')"
Se T contém a tag 'translation' ->
      "translation (conteúdo da tag filha 'x') (cont. da tag filha 'y')
          (cont. da tag filha 'z')\n"
Se T contém a tag 'scale' ->
      "scale (conteúdo da tag filha 'x') (cont. da tag filha 'y') (cont.
          da tag filha 'z')\n"
Se T contém a tag 'rotation' ->
      "rotation (conteúdo da tag filha 'x') (cont. da tag filha 'y')
          (cont. da tag filha 'z')\n"
"}\n"
```

Esquema *worldInfoTranslation* – esse esquema traduz informações contidas na *tag* 'worldInfo', que dão uma descrição do mundo 3D criado.

Essas informações são relativas ao título do mundo 3D e outras que venham a ser de interesse do autor do documento.

worldInfoTranslation [T] ->

```
Se T contém a tag 'title' ->
      "title (conteúdo da tag 'title') \n"
Se T contém a tag 'info' ->
      "info (conteúdo da tag 'info') \n"
"}\n"
```

4.2.6 Exemplo de compilação

Essa seção tem por finalidade mostrar a tradução de um documento XML (Figura 18) através do esquema de compilação mostrado anteriormente.

Deve-se observar, na Figura 18, algumas *tags* chaves. Elas são: a *tag* 'VRML', que é a *tag* raiz do XML; a *tag* '*sphere'*, que contém dados relativos à forma básica de uma esfera; a *tag* 'material', que descreve a textura usada nas formas básicas; e a *tag* '*worldInfo'*, que fornece informações a respeito do ambiente 3D descrito no documento XML.

Essas *tags* são as analisadas pelo compilador para determinar qual esquema de compilação será usado para a correta tradução das informações em XML.

Código XML contendo a descrição do ambiente 3D.	Código VRML correspondente

1. <VRML>
2. <sphere>
3. <material>
4. <emissiveColor>
5. <r>0</r>
6. <g>255</g>
7. 0
8. </emissiveColor>
9. </material>
10. <radius>10</radius>
11. </sphere>
12. <worldInfo>
13. <title>"Esfera verde de raio 10"</title>
14. </worldInfo>
15. </VRML>
	#VRML V2.0 utf8
	Shape {
	appearance Appearance {
	material Material {
	emissiveColor 0 255 0
	}
	}
	geometry Sphere {
	radius 10
	}
	}
	WorldInfo {
	title "Esfera verde de raio 10"
	}

Figura 18. Exemplo para mostrar o processo de tradução através do esquema de compilação

O passo a passo do processo de compilação do código XML exposto na Figura 18 é mostrado a seguir. O leitor deve observar a seqüência dos esquemas de compilação (cujos nomes estão em negrito) e os textos resultantes deles (informações citadas entres aspas).

Compile[<VRML> [2...11,12...14] </VRML>] ->
“#VRML V2.0 utf8\n”
Selector[2...11] **Selector**[12...14]

Selector[2...11] ->
shapeTranslation[2..11]

shapeTranslation[2..11] ->
“Shape {\n”
“ appearance Appearance {\n”
materialTranslation [3...9]
“ }\n”
“ geometry Sphere {\n”
“ radius 10\n”
“ }\n”
”}\n”

materialTranslation [3 ... 9] ->
“ material Material {\n”
“ emissiveColor 0 255 0\n”
“}\n”

Selector[12...14] ->
worldInfoTranlation [12..14]

worldInfoTranlation [12..14] ->
“WorldInfo {\n“
“ title \"Esfera verde de raio 10\"\n”
“}\n”

4.2.7 Exemplo de código

Objetivando mostrar ao leitor uma possível implementação e facilitar o entendimento do que foi anteriormente proposto está exposto, na Figura 19, um trecho do código-fonte do compilador construído, que é relativo ao esquema de compilação *worldInfoTranslation*

```
1.      public static String worldInfoTranlation(Element e){
2.          String translation=new String();
3.          Element selection;
4.          translation +=  tab + firstToUpperCase(e.getName()) + " {\n\r";
5.          incTab();
6.          if((selection=e.getChild("title"))!=null){
7.              translation+= tab + "title " + selection.getText() + "\n\r";
8.          }
9.          if((selection=e.getChild("info"))!=null){
10.             translation+= tab + "info [\n\r";
11.             List aux=selection.getChildren();
12.             incTab();
13.             for(int i=0;i<aux.size();i++){
14.                 translation+=  tab + ((Element)aux.get(i)).getText();
15.                 if(i!=(aux.size()-1)){
16.                     translation+=",";
17.                 }
18.                 translation+="\n\r";
19.             }
20.             decTab();
21.             translation+="\n\r"+ tab +"]\n\r";
22.         }
23.         decTab();
24.         translation+="}\n\r";
25.         return translation;
26.     }
```

Figura 19. Código de implementação do esquema de compilação *worldInfoTranslation*

Na linha 1 da Figura 19 está definido o nome da função que implementa o esquema de compilação *worldInfoTranslation.* O termo String, presente nessa linha, se refere ao tipo de retorno da função, ou seja, essa função irá retornar um texto relativo à tradução da *tag* de entrada, que está descrita como 'Element e'.

Internamente à função, uma variável do tipo String (texto) é criada e chamada de *translation*. Esta variável é utilizada durante todo o escopo da função e, à medida que *tags* filhas à de entrada ('e') vão sendo reconhecidas, trechos de texto traduzidos são concatenados a ela. No fim da função (linha 25), essa variável é retornada para a função que a chamou (ver esquema de compilação).

Existe uma variável global (pode ser acessada em qualquer lugar do código do compilador) que é chamada de 'tab' e serve para determinar a margem do texto. Essa variável é usada para que o texto traduzido possa ser identado, ou seja, corretamente alinhado de acordo com os níveis de profundidade da informação. Para aumentar ou diminuir o recuo da margem são usadas duas funções incTab() (linhas 5 e 12) e decTab() (linhas 20 e 23) respectivamente.

A variável local (só pode ser chamada internamente a função onde foi declarada) '*selection'* (linha 3) serve para auxiliar o processo de busca de informações da *tag* principal 'e' (linha 1). Nessa função, é verificada, de forma independente, a existência de duas *tags* filhas a 'e': a *tag* '*title'* (linha 6) e a '*info'* (linha 9). Caso as mesmas sejam encontradas, informações serão colhidas do documento XML, depois são unidas a textos-padrão da linguagem VRML, e por fim, serão concatenadas à variável *translation*. Um exemplo disto pode ser visto na linha 7, onde, caso a *tag* '*title'* seja encontrada, são inseridas informações-padrão do VRML ("*title*") e unidas a informações extraídas do XML (getText()). Por fim, há a concatenação desse trecho de texto à variável '*translation'*.

A linha 11 representa a lista de *tags* filhas da *tag* '*info'*. A tradução de todas as possíveis *tags* é feita entre as linhas 13 a 19.

4.3 Conclusão

A implementação do compilador exposto neste capítulo possibilitou a reutilização de *plugins* de visualização de ambientes 3D para VRML. A estratégia adotada envolve um processo em duas etapas. Primeiro as descrições na nova linguagem são traduzidas para VRML. Em seguida, utilizam-se *plugins* de visualização para VRML com o fim de apresentar as imagens em três dimensões.

Visando dar ao leitor um conhecimento a respeito do tema, várias características de um compilador foram descritas, além dos componentes que formam sua estrutura. Detalhes da implementação do compilador foram apresentados, salientando as tecnologias disponíveis para o seu desenvolvimento, bem como as razões pela escolha do JDOM para desenvolvimento do compilador.

O mecanismo de compilação utilizado foi descrito em detalhe através de um esquema de compilação, que cobre todos os possíveis casos da linguagem proposta. Um exemplo de tradução demonstrou a aplicação do esquema de compilação, dando ao leitor uma compreensão mais clara do funcionamento e implementação do compilador em questão.

Capítulo 5

Interface Gráfica e Programa de Assistência ao Programador

Como visto no Capítulo 2, existem duas formas, através da modelagem manual, de desenvolver arquivos que contenham descrições de ambientes 3D: através de um processador de texto, onde o usuário fica à mercê do próprio conhecimento da linguagem para desenvolvê-lo; e através de interfaces gráficas, que, com a geração automática de código, auxiliam o usuário, que precisa inserir apenas os dados essenciais para a formação dos objetos 3D.

A linguagem VWML de descrição de ambientes 3D tem uma escrita mais intuitiva, se comparada a outras com o mesmo objetivo. Porém é mais fácil e rápido desenvolver arquivos de descrição de objetos 3D através de uma interface gráfica.

A seção, a seguir, detalha a criação de uma interface gráfica de auxílio ao desenvolvimento para os usuários da linguagem exposta neste trabalho.

5.1 Interface gráfica

Paralelamente ao desenvolvimento da linguagem foi criada uma interface gráfica para auxiliar os usuários desta linguagem.

Cláudio Cavalcanti, ex-aluno do curso de Engenharia da Computação da UPE, trabalhou durante sua graduação em projetos de pesquisa focados na área de computação gráfica e processamento de imagens. Devido à sua experiência com a modelagem de mundos virtuais, Cavalcanti foi convidado a participar do projeto da nova linguagem, tendo sido o responsável direto pelo desenvolvimento da ferramenta para modelagem 3D a ser descrita neste capítulo.

5.1.1 Linguagem de programação e característica da implementação

A interface gráfica de auxílio ao usuário foi desenvolvida através da linguagem de programação Java. A subseção 4.2.1 mostra diversas características desta linguagem de programação, que determinaram a sua escolha. Dentre as características citadas, está a existência da API de fácil entendimento e acesso, o que acelerou o processo de desenvolvimento.

Um desafio encontrado durante o desenvolvimento dessa interface foi a determinação de uma maneira fácil e intuitiva para representar as diversas formas básicas da linguagem criada por este trabalho e a possibilidade de agrupamento das mesmas (característica da linguagem descrita na subseção 3.2.2).

Para a representação das formas básicas (cilindro, esfera, cubo, cone e texto) foram utilizadas janelas que contêm informações específicas para cada uma delas. Cabe ao usuário atribuir valores intrínsecos a cada objeto, para que o mesmo possa ser inserido no mundo 3D.

Com relação ao agrupamento de formas básicas, a solução se deu através do uso de JInternalFrame [52], que é uma funcionalidade da API gráfica de Java que permite a inserção de janelas gráficas internas a outras. Sendo assim, todos os objetos que forem criados internamente a uma janela que represente um agrupamento (janela com título Grupo) serão agrupados, ou seja, aparecerão no escopo de uma *tag* <*group*> no arquivo XML gerado.

A figura 19 mostra o desenvolvimento através da interface gráfica de um mundo 3D.

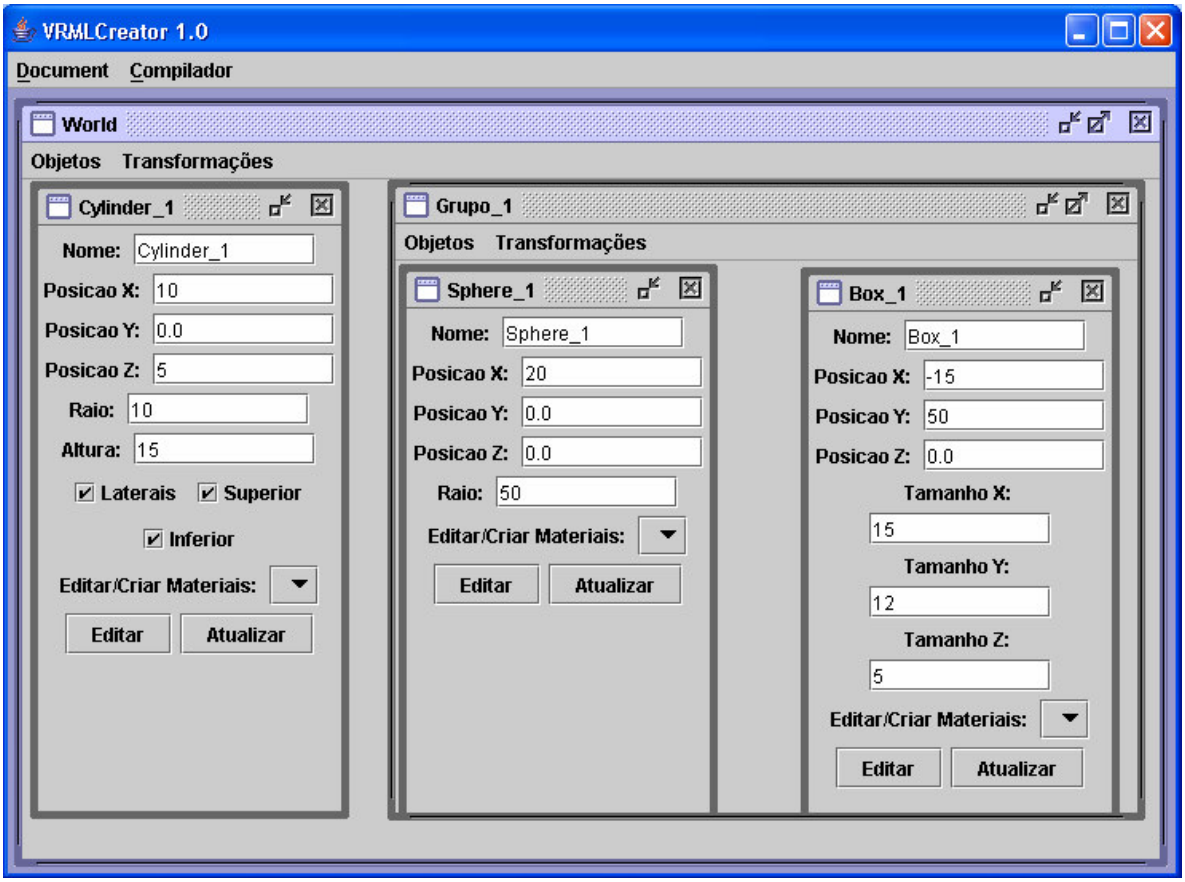

Figura 20. Interface gráfica

Como o leitor pode ver através da Figura 20, a janela mais externa é a da interface gráfica cujo título é: *VRML Creator* 1.0. A janela com o título "*World*" representa o ambiente 3D que está sendo desenvolvido pelo usuário. Esse ambiente irá conter todas as formas básicas que o usuário desejar utilizar.

Dentro do mundo que está sendo utilizado pelo usuário, existe na metade esquerda uma janela, cujo título é "*Cylinder*_1", que representa um cilindro. Na metade direita, existe outra, cujo título é "*Grupo_1"*, que representa um agrupamento de dois objetos, sendo esses uma esfera e uma caixa, que são descritos através das janelas *"Sphere_1"* e *"Box_1"*, respectivamente.

Algo importante a ser salientado quanto ao XML gerado pela interface gráfica, é que esse tem a garantia de correção. O mesmo não ocorre com os arquivos editados manualmente (através de processadores de textos) pelo usuário, que devem ser verificados pelo XML *Schema* para validação.

Ainda na Figura 20, pode-se observar que em cada janela das formas básicas, existe a opção de "Editar/Criar Materiais". Essa funcionalidade serve para que o usuário defina os tipos de materiais (cores, brilho e transparência) que podem ser utilizados para um dado objeto e também possa reutilizá-los.

No menu existente na janela de título "*World*", existe a opção "Objetos", que possibilita a inserção de todas as formas básicas definidas na linguagem VWML, bem como o agrupamento delas. Por sua vez, a opção de "Transformações" (implementa a característica "*Transform*") possibilita modificações da forma dos objetos e sistema de coordenadas espacial.

A janela relativa à interface gráfica "VRML Creator 1.0" contém um menu com a opção "*Document*", que possibilita a criação de diversos mundos virtuais; e a opção "Compilador", que permite a visualização da descrição em XML e a compilação para que haja a criação de um arquivo contendo as informações deste mundo virtual na linguagem VRML.

Na Figura 21 o leitor pode ver a descrição em VWML, relativa ao exemplo da Figura 20, gerada pela interface gráfica.

```
<VRML>
     <cylinder id='Cylinder_1'>
        <position>
            <x>10.0</x>
            <y>0.0</y>
            <z>5.0</z>
        </position>
        <radius>10.0</radius>
        <height>15.0</height>
        <side>TRUE</side>
        <top>TRUE</top>
        <bottom>TRUE</bottom>
     </cylinder>
     <group id='Grupo_1'>
          <sphere id='Sphere_1'>
               <position>
                    <x>20.0</x>
                    <y>0.0</y>
                    <z>0.0</z>
               </position>
               <radius>50.0</radius>
          </sphere>
          <box id='Box_1'>
               <position>
                    <x>-15.0</x>
                    <y>50.0</y>
                    <z>0.0</z>
               </position>
               <sizeX>15.0</sizeX>
               <sizeY>12.0</sizeY>
               <sizeZ>5.0</sizeZ>
          </box>
     </group>
</VRML>
```

Figura 21. Descrição em VWML gerada pela Interface Gráfica

5.2 Programa de assistência ao programador

Tudo que foi visto anteriormente neste trabalho possibilita a descrição de um ambiente 3D através da linguagem apresentada, que é baseada em XML. Com o compilador exposto no Capítulo 4 pode-se obter um arquivo contendo a mesma descrição escrita em VRML. Essa última descrição pode ser visualizada através de *plugin* instalado em um *browser*.

Visando a um possível uso dos arquivos gerados pelo projetista dos ambientes 3D, foi desenvolvido um programa assistente que tem como objetivo gerar um arquivo com código-fonte Java de um programa de visualização dos mundos virtuais criados pelo projetista [1].

O objetivo da construção desse assistente foi gerar um elo entre dois mundos, o de desenvolvimento gráfico e o de programação. Um ponto de partida é dado para o programador Java que deseja exibir os arquivos gerados pelo designer de ambientes 3D. Além da localização dos exemplos criados pelo projetista gráfico, o programa gerado tem todo o mecanismo gráfico Java descrito no seu código-fonte.

5.2.1 Implementação

Esta seção objetiva dar uma visão geral da implementação do assistente, bem como dos requisitos necessários para sua utilização.

Linguagem adotada

A linguagem utilizada para a implementação do programa assistente foi Java. Os motivos tais são os mesmos expostos na Subseção 4.2.1.

A implementação foi feita visando a dar ao usuário uma maneira fácil e intuitiva de obter o produção final. O programa é baseado em passos e, em cada um deles, existem explicações para auxiliar o usuário.

Requisitos mínimos para uso da ferramenta de auxílio ao programador

Existem duas formas de executar o programa de auxílio ao programador: através de arquivo executável para sistema operacional Windows; através de um interpretador (programa que interpreta e executa) Java.

Dessa forma, tem-se que os requisitos são: sistema operacional Windows; ou qualquer sistema operacional e um interpretador Java.

Para que o usuário do programa assistente consiga executar todos os passos do programa é fundamental a existência de pelo menos um arquivo VRML, cuja extensão é ".wrl". Caso este arquivo não seja encontrado, o assistente não permite que o usuário prossiga para os passos seguintes para criação da aplicação.

Para que seja possível visualizar a aplicação, o usuário necessita de um *plugin* de visualização de arquivos VRML instalado no Internet Explorer (navegador padrão do Windows).

---

1. Esta ferramenta foi totalmente desenvolvida pelo autor deste trabalho.

5.2.2 Passo-a-passo

As Figuras 22, 23 e 24 apresentam todos os três passos necessários para a criação do arquivo do programa Java de visualização dos exemplos em VRML.

O primeiro passo (Figura 22) consta na determinação da quantidade de exemplos que o designer deseja exibir no programa gerado.

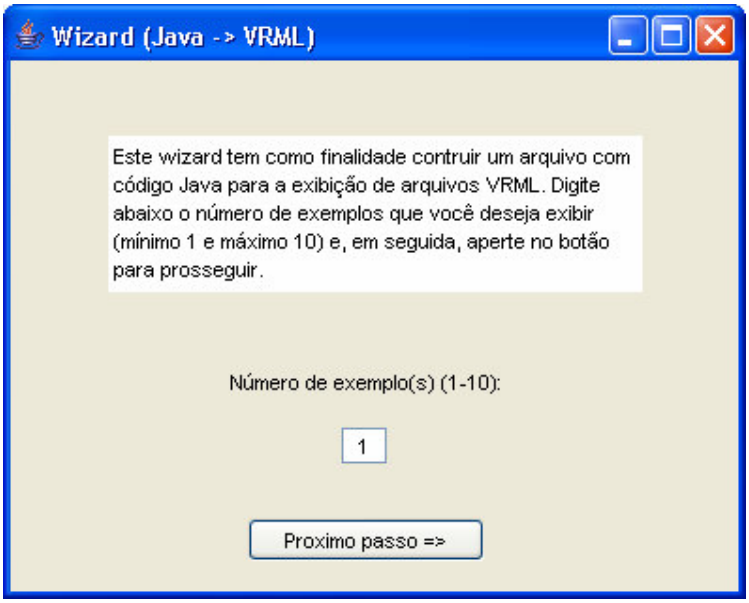

Figura 22. Passo 1

O segundo passo (Figura 24) resume-se à localização de todos os exemplos que foram determinados no passo anterior.

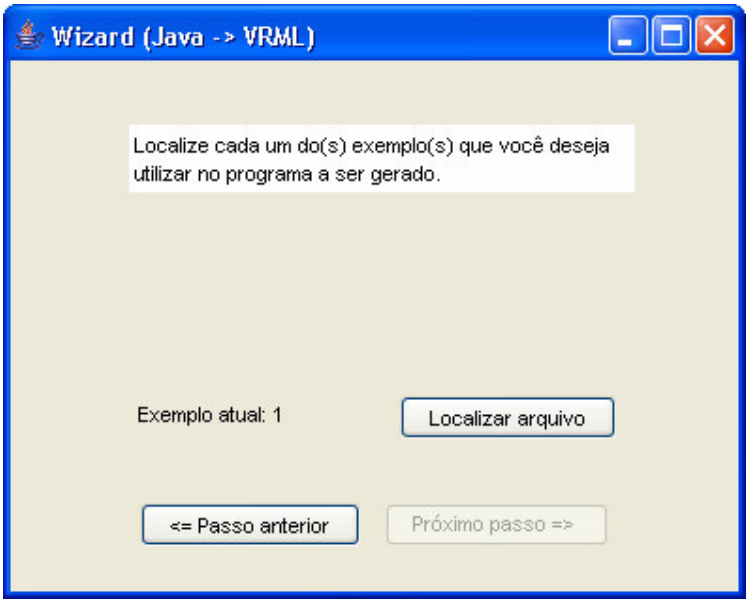

Figura 23. Passo 2

No terceiro e último passo (Figura 24) o usuário deve determinar o local onde será gerado o único arquivo Java que conterá informações relativas ao programa desejado pelo usuário do assistente.

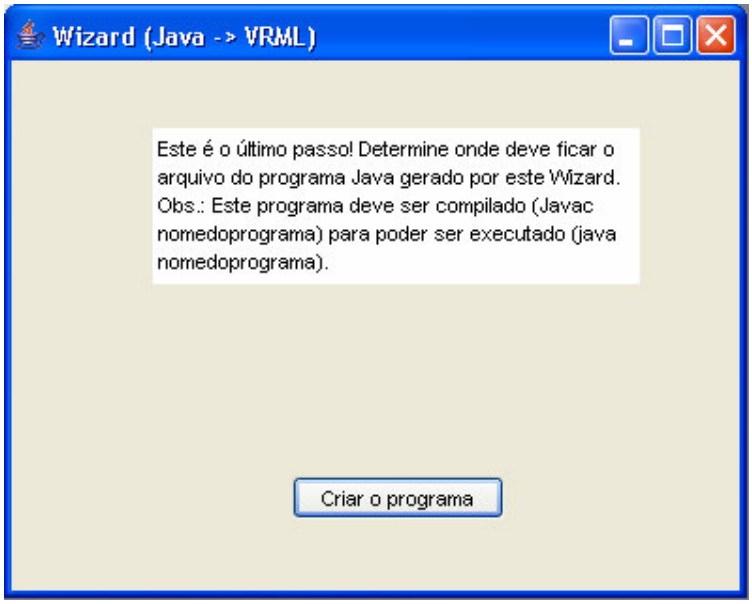

Figura 24. Passo 3

O programa exposto na Figura 25 foi gerado pelo assistente. Lembrando que para a execução desse programa, foi necessária a sua compilação. Cada um dos botões contidos nele serve para iniciar a visualização de um exemplo determinado pelo usuário do assistente.

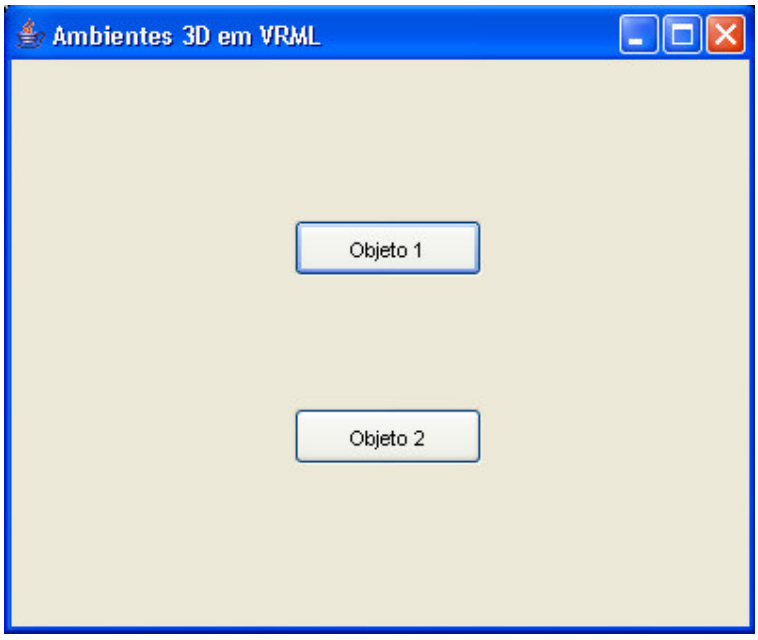

Figura 25. Programa gerado pelo assistente.

Após o pressionamento de um dos botões, relativos aos objetos criados pelo designer, o navegador *Internet Explorer* é aberto automaticamente. O *plugin* instalado nesse navegador

exibirá o modelo tridimensional correspondente. A Figura 26 mostra um modelo 3D de uma mesa e um televisão sendo visualizado através do *plugin*.

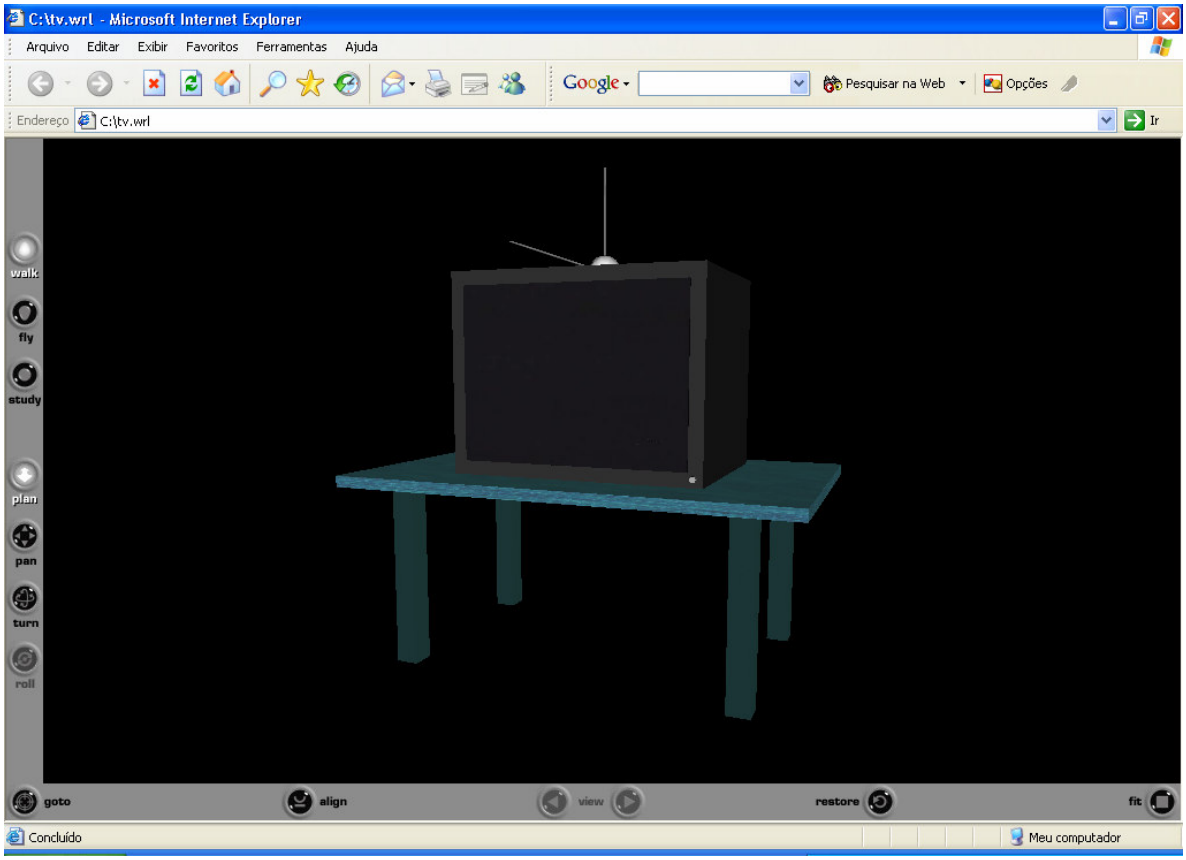

Figura 26. Modelo 3D de uma mesa e televisão

5.2.3 Código-fonte gerado

No Apêndice B, encontra-se o código-fonte gerado pelo programa de assistência ao programador. O leitor, que for familiarizado com a API gráfica de Java, não terá dificuldade em compreender o programa, que contém uma classe principal, cujo nome foi definido pelo usuário do programa assistente. Essa classe, por sua vez, contém uma referência para a classe *Frame* (classe que implementa as funcionalidades de uma janela de aplicativo), que, por sua vez, recebe um objeto da classe *Frame1*, cujo tipo é *JFrame,* que herda de *Frame*, dando suporte a componentes da arquitetura *JFC/Swing*.

A classe *Frame1* contém todos os botões, com as respectivas informações a respeito de posicionamento e ações a serem tomadas ao pressionar os botões, ou seja, a visualização de um objeto 3D.

As demais classes presentes estão relacionadas com a funcionalidade de detecção de pressionamento dos botões presentes na interface gráfica.

5.3 Conclusões

A construção da interface gráfica trouxe maior agilidade no desenvolvimento de ambientes 3D, visto que o projetista gráfico não fica à mercê do seu próprio conhecimento sobre a linguagem VWML para desenvolver imagens 3D.

Com a interface auxiliando, o projetista precisa preencher apenas poucas informações, que são necessárias para a formação dos objetos da cena. Uma vantagem adicional encontra-se no fato de que o arquivo gerado, que contém a descrição do ambiente 3D, tem a garantia de correção dada pela interface.

O programa de assistência ao programador traz, de forma fácil e rápida, um ponto de partida para a criação de aplicativos gráficos de visualização de ambientes 3D. O arquivo gerado pelo assistente contém o código-fonte escrito em Java. O mesmo pode ser modificado livremente inclusive através de IDEs (**I**ntegrated **D**evelopment **E**nvironment), fato esse que acelera o processo de desenvolvimento.

Capítulo 6

Conclusões e Trabalhos Futuros

A linguagem VWML de descrição de ambientes 3D foi desenvolvida, visando a trazer ao usuário benefícios que não são encontrados na maioria das outras linguagens. Um desses benefícios está no fato de a mesma ser baseada em XML (padrão para descrição de informações), o que torna mais fácil a transferência de informações entre aplicativos. Outro benefício está no fato da clareza que ela proporciona às informações dos ambientes 3D descritos por elas.

Um ponto negativo que existe na VWML está na sua capacidade de descrição que é bastante limitada, se comparada às consolidadas no mercado, não possibilitando que a mesma seja posta para concorrer com as demais no seu estado atual. Diversos recursos existentes nas outras linguagens de descrição de ambientes 3D não são suportados. Alguns desses recursos são: animações dos objetos, que permitem a movimentação própria dos elementos existentes no mundo virtual; extrusão, onde objetos 3D podem ser gerados através da movimentação de objetos 2D (mais detalhes na subseção 2.1.2); iluminação e sombreamento, pontos de luz podem ser inseridos no ambiente 3D, fazendo com que algumas faces dos objetos presentes numa cena fiquem mais brilhantes e sombras sejam geradas, fazendo, desta forma, com que determinadas regiões fiquem mais destacadas do que outras.

É importante salientar que, por basear-se em XML, a linguagem pode ser estendida para suportar todos esses recursos, de maneira relativamente rápida. As características já suportadas pela linguagem foram suficientes para comprovar que a mesma oferece os benefícios a que se propõe, ou seja, um rápido desenvolvimento de mundos virtuais através de uma descrição mais clara e intuitiva.

Dentre as linguagens expostas neste trabalho, a X3D é a mais próxima da VWML, pois um dos seus possíveis tipos de codificação é baseado em XML. O padrão X3D [14] não está ainda definido totalmente através da ISO, porém já vem sendo utilizado por diversas empresas e corporações, o que vem mostrando ser uma tendência comercial. Entretanto, consideramos o XML do X3D menos estruturado que o apresentado no VWML, pois a quantidade de níveis para o detalhamento das informações dos objetos é pequena (muitas informações dentro de uma única *tag*), o que sobrecarrega algumas *tags*. Outro problema é que muitos dos atributos, por si só, não explicam detalhadamente o seu conteúdo, tirando a clareza das informações correspondentes.

O uso de interface gráfica para o desenvolvimento dos objetos 3D veio facilitar e acelerar esse processo, além de garantir a correção do arquivo gerado. Portanto, torna-se bastante recomendado o seu uso. Para fins comerciais, as interfaces gráficas são primordiais, devido ao aumento considerável de produtividade.

Um avanço que pode ser feito na interface gráfica em questão é a visualização instantânea dos objetos que estão sendo criados e a possibilidade da mudança do posicionamento dos mesmos

através do mouse. Para tal, o esforço necessário é bastante alto, devido à dificuldade de desenvolvimento do programa de visualização.

O programa de assistência ao programador não deve ser mais complexo do que está atualmente. O acréscimo de funcionalidades como, por exemplo, a possibilidade de inserção de menus e outras funcionalidades gráficas, tornaria esse programa bastante próximo de IDEs de desenvolvimento. O propósito desse assistente é bem claro, o de proporcionar um ponto de partida ao programador. Porém, funcionalidades relativas à compactação e proteção dos códigos dos modelos gerados pelo designer podem ser implementados. Essas funcionalidades serviriam para facilitar o trafego de informações e garantir a não modificação dos modelos em 3D respectivamente.

Bibliografia

[1] Computer Graphics - Principles and Practice (1996) - Foley, James D. / Van Dam, A. - Addison Wesley Pub.

[2] Azevedo, Eduardo; Conci, Aura. Computação Gráfica – Teoria e prática.Rio de Janeiro: Editora Campus, 2003.

[3] Além 3D – A garrafa de Klein. Disponível em: <http://alem3d.obidos.org/pt/struik/kbottle/mov>. Acesso em 20 de fevereiro. 2005.

[4] OpenGL - The Industry's Foundation for High Performance Graphics. Disponível em: <http://www.opengl.org/>. Acesso em 15 de fevereiro. 2005.

[5] True Character truespace eXtension. Disponível em: <http://www.blackknightproductions.com/tc.htm>. Acesso em 28 de junho. 2005.

[6] Microsoft DirectX. Disponível em : <http://www.microsoft.com/windows/directx/default.aspx>. Acesso em 28 de junho. 2005.

[7] OpenSceneGraph. Disponível em: <http://www.openscenegraph.org/>. Acesso em 15 de fevereiro. 2005.

[8] The Inventor Mentor, written by Josie Wernecke, is published by Addison-Wesley and is available in technical bookstores (ISBN 0-201-62495-8).

[9] OpenInventorTM. Disponível em: <http://oss.sgi.com/projects/inventor/>. Acesso em 15 de fevereiro. 2005.

[10] OpenMOIV.Universitat Pompeu Fabra: Grupo de Tecnologias Interativas. Disponível em: <http://www.tecn.upf.es/openMOIV/relatedlibs/oiv.html>. Acesso em 15 de fevereiro. 2005.

[11] Computer System, Inc. Mercury. Disponível em: <http://www.tgs.com/>. Acesso em 16 de fevereiro. 2005.

[12] Ames, Andrea L.; Nadeau, David R; Moreland, John L.: VRML 2.0 sourcebook, John Wiley & Sons, Inc, 1997.

[13] CANAL #HTML. Disponível em: <http://www.htmlstaff.org/vrml.php>. Acesso em 08 de março. 2004.

[14] Web 3D Consortium - Open Standards for Real-Time 3D Communication. Disponível em: <http://www.web3d.org/x3d/>. Acesso em 16 de fevereiro. 2005.

[15] Extensible 3D (X3D) – Profiles. Disponível em: <http://www.web3d.org/x3d/specifications/ISO-IEC-19775-IS-X3DAbstractSpecification/Part01/profileIndex.html>. Acesso em 1 de julho. 2005.

[16] International Organization for Standardization. Disponível em: <http://www.iso.org>. Acesso em 18 de fevereiro. 2005.

[17] Autodesk 3ds Max. Disponível em: <http://www4.discreet.com/3dsmax/>. Acesso em 23 de março. 2005.

[18] Grapho Design Software. Disponível em: <http://www.grapho.com.br/3ds/3dmax.htm>. Acesso em 23 de março. 2005.

[19] Blender Foundation. Disponível em: <http://www.blender.org/>. Acesso em: 8 de dezembro. 2004.

[20] Blender X3D Exporter. Disponível em: <http://www.bitbucket.ca/~acheater/blender/>. Acesso em: 8 de dezembro. 2004.

[21] Autodesk AutoCAD. Disponível em: <http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=2704278>, acessado em 24 de março. 2005.

[22] TECAD - Soluções CAD e Projecto Colaborativo. Disponível em: <http://www.tecad.pt/subscrever/AutoCAD_2006_DocTec2_%20Perguntas_Respostas.pdf> . Acesso em 24 de março. 2004.

[23] The official Half-Life Web Site. Disponível em: <http://half-life.sierra.com/>. Acesso em 30 de junho. 2005.

[24] MilkShape 3D. Disponível em: <http://www.swissquake.ch/chumbalum-soft/index.html>. Acesso em: 8 de dezembro. 2004.

[25] Extensible Markup Language (XML). Disponível em http://www.w3.org/XML/. Acesso em 25 de outubro. 2004.

[26] Ray, Erik T.. Aprendendo XML: tradução Daniel Vieira. – Rio de Janeiro: Campus, 2001.

[27] Introdução à linguagem SGML: O SGML e suas origens. Disponível em: <http://lie-br.conectiva.com.br/godoy/sgml-1.html#ss1.1>. Acesso em 28 de junho. 2005.

[28] World Wide Web Consortium – W3C. Disponível em: <http://www.w3.org/>. Acesso em 28 de junho. 2005.

[29] Introdução à Linguagem HTML. USP - ICMP. Disponível em: <http://www.icmc.usp.br/ensino/material/html/intro.html>. Acesso em 19 de março. 2005.

[30] XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). Disponível em: <http://www.w3.org/TR/2002/REC-xhtml1-20020801/>. Acesso em 19 de março. 2005.

[31] Introduction to DTD. Disponível em: <http://www.xmlfiles.com/dtd/dtd_intro.asp>. Acesso em 4 de maio. 2005.

[32] XML Schema Part 1: Structures Second Edition. Disponível em: <http://www.w3.org/TR/xmlschema-1/>. Acesso em 4 de maio. 2005.

[33] Compiladores – Princípios, Técnicas e Ferramentas –Aho, Alfred V. / Sethi, Ravi/ Ullman, Jeffrey D. – Guanabara Koogan.

[34] A construção de um compilador – Setzer, Valdemar W. / Melo, Inês S. H.. Rio de janeiro: ed: campus, 1985.

[35] Columbia University in the city of New York – John Backus. Disponível em: <http://www.columbia.edu/acis/history/backus.html>. Acesso em 13 de abril. 2005.

[36] The University of Michigan – Dearborn. The FORTRAN Programming Language. Disponível em: <http://www.engin.umd.umich.edu/CIS/course.des/cis400/fortran/fortran.html>. Acesso em 13 de abril. 2005.

[37] Universidade Federal de Campina Grande – Departamento de Sistemas e Computação. Compiladores. Disponível em: <http://www.dsc.ufcg.edu.br/~peter/cursos/cc/material/p1.pdf>. Acesso em 9 de abril. 2005.

[38] JLex: A Lexical Analyzer Generator for Java(TM). Princeton University. Disponível em: <http://www.cs.princeton.edu/~appel/modern/java/JLex/>. Acesso em 2 de abril. 2005.

[39] MAT153 - Construção de Compiladores. Universidade Federal da Bahia. Disponível em: <http://www.im.ufba.br/mat153>. Acesso em 2 de abril. 2005.

[40] CUP Parser Generator for Java. Princeton University. Disponível em: <http://www.cs.princeton.edu/~appel/modern/java/CUP/>. Acesso em 2 de abril. 2005.

[41] Unix Programming – Lex & Yacc. John R. Levine; Tony Mason & Doug Brown. Editora O´Reilly – 1992.

[42] Java Compiler Compiler [tm] (JavaCC [tm]) - The Java Parser Generator. Disponível em: <https://javacc.dev.java.net/>. Acesso em 2 de abril. 2005.

[43] Utilização do JavaCC na construção de um compilador – UNICAMP. Disponível em: <http://www.dicas-l.unicamp.br/dicas-l/20040731.php>. Acesso em 2 de abril. 2005.

[44] Addison Wesley. Processing XML with Java. Disponível em: <http://www.awprofessional.com/articles/article.asp?p=30609&seqNum=2&rl=1>. Acesso em 1 de abril. 2005.

[45] Extensible Markup Language (XML) 1.1. Disponível em: <http://www.w3.org/TR/2004/REC-xml11-20040204/>. Acesso em 1 de abril. 2005.

[46] SAX Project. Disponível em: <http://www.saxproject.org/>. Acesso em 1 de abril. 2005.

[47] A Sax XML Parser for Allegro Common Lisp . Disponível em: <http://www.franz.com/support/documentation/7.0/doc/sax.htm>. Acesso em 1 de abril. 2005.

[48] W3C Document Object Model (DOM). Disponível em: <http://www.w3.org/DOM/>. Acesso em 1 de abril. 2005.

[49] Cover Pages. W3C Document Object Model (DOM). Disponível em: <http://xml.coverpages.org/dom.html>. Acesso em 1 de abril. 2005.

[50] DOMTM Project. Disponível em: <http://www.jdom.org>. Acesso em 15 de abril. 2005.

[51] JDOM and XML Parsing. Oracle DEVELOPER. Disponível em: <http://www.oracle.com/technology/oramag/oracle/02-sep/o52jdom.html>. Acesso em 15 de abril. 2005.

[52] How to Use Internal Frames. Java Technology. Sun. Disponível em: <http://java.sun.com/docs/books/tutorial/uiswing/components/internalframe.html>. Acesso em 21 de maio. 2005.

Apêndice A

Notação da linguagem desenvolvida neste trabalho

Este apêndice contém a definição de toda a linguagem VWML, com as funcionalidades atualmente suportadas.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<VRML>
    <group id="nome">
      <sphere id="nome">
          <material id="nome">
             <!--esse elemento use não permite que os outros elementos sejam
              definidos-->
             <!--ou seja: ou deve ser usado o elemente use, ou os outros-->
             <!--o id de material deve ser nulo caso use seja definido-->
                  <use>idDoMaterial</use>

                  <ambientIntensity>0.2</ambientIntensity>

                  <diffuseColor>
                        <!--entre 0 e 1-->
                        <r>0.8</r>
                        <g>0.8</g>
                        <b>0.8</b>
                  </diffuseColor>

                  <emissiveColor>
                        <!--entre 0 e 1-->
                        <r>0</r>
                        <g>0</g>
                        <b>0</b>
                  </emissiveColor>

                  <shininess>0.2</shininess>

                  <specularColor>
                        <!--entre 0 e 1-->
                        <r>0</r>
                        <g>0</g>
                        <b>0</b>
                  </specularColor>

                  <transparency>0</transparency>
```

```
                <!-- Os dois elementos seguintes servem para definir
                 texturas a partir de imagens e vídeos pré-existentes
                 respectivamente.-->

                <imageTexture>"imagem.jpg"</imageTexture>

                <movieTexture id"nome">
                      <url>"movie.mpg"</url>
                      <loop>TRUE/FALSE</loop>
                </movieTexture>

          </material>

          <position>
                <!--qualquer valor-->
                <x>2.5</x>
                <y>5.0</y>
                <z>-3.0</z>
          </position>
          <radius>1.0</radius>
    </sphere>

    <box id="nome">
          <material>
          ...
          </material>
          <position>
          ...
          </position>
          <!--qualquer valor-->
          <sizeX>2.0</sizeX>
          <sizeY>2.0</sizeY>
          <sizeZ>2.0</sizeZ>
    </box>

    <cone id="nome">
          <material>
          ...
          </material>
          <position>
          ...
          </position>
          <!--qualquer valor-->
          <bottomRadius>3.0</bottomRadius>
          <height>2.0</height>
          <bottom>TRUE</bottom>
          <side>TRUE</side>
    </cone>

    <cylinder id="nome">
          <material>
          ...
          </material>
          <position>
          ...
          </position>
          <!--qualquer valor-->
          <radius>3.0</radius>
          <height>2.0</height>
```

```
            <top>TRUE</top>
            <bottom>TRUE</bottom>
            <side>TRUE</side>
      </cylinder>
      <text id="nome">
            <material>
            ...
            </material>
            <position>
            ...
            </position>
            <fontStyle>
                  <family>SERIF</family>
                  <horizontal>TRUE</horizontal>
                  <justify>BEGIN</justify>
                  <leftToRight>TRUE</leftToRight>
                  <size>1</size>
                  <spacing>1</spacing>
                  <style>PLAIN</style>
                  <topToBottom>TRUE</topToBottom>
            </fontStyle>
            <maxExtent>0</maxExtent>
            <string>Este é o texto!</string>
      </text>

      <transform id="transf1">
                  <!-- Ou o usuário utiliza a tag use (para reutilizar
                  elementos), ou a def (para definir elementos) -->
                  <use>
                        <!--O source pode ser um grupo ou um objeto-->
                        <source>Group1</source>
                  </use>

                  <def>
                        <cylinder></cylinder>
                        <box></box>
                         <sphere></sphere>
                         <cone></cone>
                         <text></text>
                  <def>

                  <translation>
                        <!--qualquer valor-->
                  <x>2.5</x>
                  <y>5.0</y>
                  <z>-3.0</z>
            </translation>

            <scale>
                  <!--entre 0 e 1-->
                  <x>0.5</x>
                  <y>0.0</y>
                  <z>0.1</z>
            </scale>

            <rotation>
                  <!--entre 0 e 1-->
                  <x>0.5</x>
                  <y>0.0</y>
                  <z>0.1</z>
```

```
                    <degree>0.5</degree>
                </rotation>
            </transform>
    </group>

    <worldInfo>
        <title>"titulo"</title>
         <info>"informações sobre o ambiente 3D."</info>
    </worldInfo>

</VRML>
```

Apêndice B

Código-fonte gerado pelo programa de assistência ao programador

Apresentamos aqui o código-fonte gerado pelo programa de assistência ao programador para o exemplo mostrado na seção 5.2.2 e explicado na seção 0 desta monografia.

```
import javax.swing.UIManager;
import java.awt.*;

import java.awt.event.*;
import javax.swing.*;
public class teste {
  boolean packFrame = false;
  public teste() {
    Frame1 frame = new Frame1();
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation( (screenSize.width - frameSize.width) / 2,
                      (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
  }
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    new teste();
  }
}
```

```
class Frame1 extends JFrame {
  JPanel contentPane;
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();

  public Frame1() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  private void jbInit() throws Exception  {
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(null);
    this.setSize(new Dimension(400, 335));
    this.setTitle("Ambientes 3D em VRML");
    jButton1.setBounds(new Rectangle(150, 85, 100, 30));
    jButton1.setText("Objeto 1");
    jButton1.addActionListener(new Frame1_jButton1_actionAdapter(this));
    contentPane.add(jButton1, null);
    jButton2.setBounds(new Rectangle(150, 185, 100, 30));
    jButton2.setText("Objeto 2");
    jButton2.addActionListener(new Frame1_jButton2_actionAdapter(this));
    contentPane.add(jButton2, null);
    }
    protected void processWindowEvent(WindowEvent e) {
      super.processWindowEvent(e);
      if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
      }
    }
    void jButton1_actionPerformed(ActionEvent e) {
      try {
        Runtime.getRuntime().exec("C:\\Arquivos de programas\\Internet
Explorer\\iexplore.exe C:\\Documents and Settings\\XP\\Meus
documentos\\teste.wrl");
        Runtime.getRuntime().exec("C:\\Program Files\\Internet
Explorer\\iexplore.exe C:\\Documents and Settings\\XP\\Meus
documentos\\teste.wrl");
      } catch( Exception e2 ) {}
    }
    void jButton2_actionPerformed(ActionEvent e) {
      try {
        Runtime.getRuntime().exec("C:\\Arquivos de programas\\Internet
Explorer\\iexplore.exe C:\\Documents and Settings\\XP\\Meus
documentos\\teste.wrl");
        Runtime.getRuntime().exec("C:\\Program Files\\Internet
Explorer\\iexplore.exe C:\\Documents and Settings\\XP\\Meus
documentos\\teste.wrl");
      } catch( Exception e2 ) {}
    }
}
```

```
class Frame1_jButton1_actionAdapter implements java.awt.event.ActionListener {
  Frame1 adaptee;

  Frame1_jButton1_actionAdapter(Frame1 adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButton1_actionPerformed(e);
  }
}
class Frame1_jButton2_actionAdapter implements java.awt.event.ActionListener {
  Frame1 adaptee;

  Frame1_jButton2_actionAdapter(Frame1 adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButton2_actionPerformed(e);
  }
}
```