

Capítulo

2

Programando em X3D para integração de aplicações e suporte multiplataforma

Eduardo de Lucena Falcão

LabTEVE - Laboratório de Tecnologias para o Ensino Virtual e Estatística / UFPB
eduardolfalcao@gmail.com

Liliane dos Santos Machado

Departamento de Informática - CCEN/UFPB
liliane@di.ufpb.br

Thaíse Kelly de Lima Costa

Departamento de Ciências Exatas - CCAE/UFPB
thaise@cae.ufpb.br

Abstract

This text presents the development process of applications with X3D, exploring its potential for integration and support of VR environments on different platforms. From examples, it is gradually presented the construction of a virtual environment that integrates interactions with objects and connections to applications outside the virtual environment (VE). Thus, nodes and scripts for access are defined and presented to assign behavior to objects and allow access to internal and external events. Development tools and platforms of execution are presented to illustrate the support of X3D for 3D Web content on PCs, mobile devices and projection systems.

Resumo

Este texto tem como objetivo apresentar o processo de desenvolvimento de aplicações com X3D explorando suas potencialidades quanto à integração de aplicações e suporte à sua utilização em diferentes plataformas para acesso a ambientes de RV. A partir de exemplos é apresentada, de forma gradativa, a construção de um ambiente virtual que integra tanto interações com objetos, bem como conexões a aplicações externas ao AV. Neste sentido, nós e scripts de acesso são definidos e apresentados para atribuir comportamento aos objetos e permitir acesso a eventos internos e externos. Ambientes de desenvolvimento e plataformas de execução são apresentados para exemplificar o suporte do X3D e suas potencialidades na aplicação de AVs na Web para PCs, dispositivos móveis e sistemas de projeção.

2.1. Introdução

X3D (*Extensible 3D*) é o padrão adotado internacionalmente para 3D na Web [Brutzman, 2007]. Ele é utilizado para construir ambientes virtuais tridimensionais complexos (também chamados de cenas). Ele é um padrão aberto que permite descrever em um arquivo formas e comportamentos de um ambiente virtual. As formas são descritas por figuras geométricas e os comportamentos da cena podem ser controlados internamente pelo arquivo X3D e externamente por linguagens de programação ou *script*.

O X3D evoluiu do VRML97 (Virtual Reality Modeling Language), um antigo padrão para descrição de conteúdo 3D internacionalmente aceito. Nesta evolução foi aproveitado o conceito introduzido pelo VRML, utilizando suas idéias básicas e promovendo a ampliação delas com a incorporação de novas funcionalidades. A principal mudança está no novo formato de codificação adotado: O XML. O XML (Extensible Markup Language) é um padrão mais conhecido e robusto que o antigo, e proporciona maior facilidade para integrar com a Web. Com esta mudança, o X3D possui agora uma arquitetura modularizada, permitindo uma maior extensibilidade e flexibilidade [Web3D-d, 2009]. Deste modo as aplicações podem ser mais facilmente desenvolvidas pela possibilidade de não precisarem implementar de uma vez todas as funcionalidades definidas nas especificações do X3D.

Neste capítulo serão abordados os principais conceitos relacionados ao X3D e apresentadas suas principais funcionalidades. O objetivo é permitir ao leitor compreender os elementos principais de um ambiente virtual descrito com X3D, além das características e possibilidades que a ele podem ser adicionadas.

2.1.1. Ambientes Virtuais

A concepção e disponibilização de ambientes virtuais (AVs) permitem conhecer, explorar e interagir em espaços tridimensionais que representam espaços reais ou imaginários para finalidades diversas. Tais AVs podem ter acesso local ou remoto, quando disponibilizados através da Internet. Nesse contexto, pode-se citar a construção de AVs com propósito de entretenimento, como o Second Life [Kumar, 2008], de auxílio à aprendizagem, como no projeto Campus Virtual da Universidade Federal da Paraíba [Machado, 2006], e para treinamentos [Burdea, 1994], dentre outros relacionados a diversas áreas das ciências.

No projeto Campus Virtual, a Universidade Federal da Paraíba foi modelada utilizando o X3D. A ela foram acoplados serviços voltados ao apoio do ensino e de interesse social à mesma. O projeto ficou disponível pela Internet para prover uma réplica do ambiente universitário adicionado de ferramentas computacionais de ensino [Machado, 2006] (Figura 2.1).

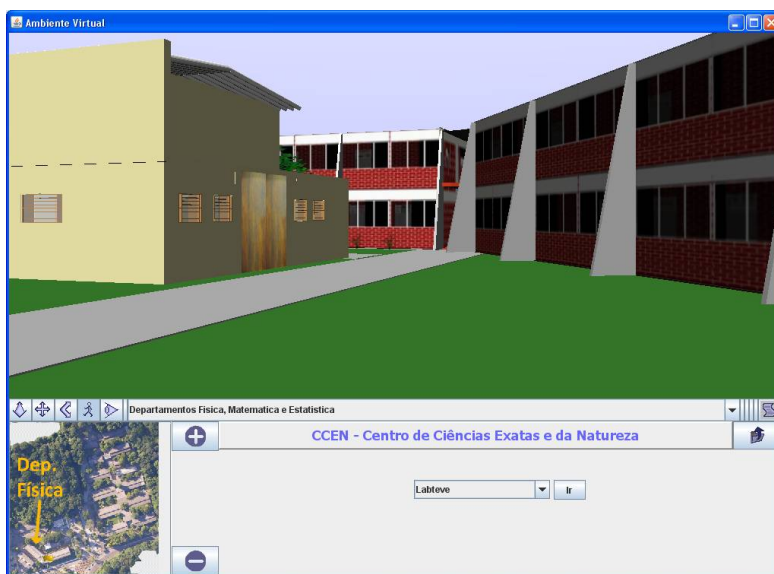


Figura 2.1. Projeto Campus Virtual da Universidade Federal da Paraíba.

Os AVs X3D também podem ser disponibilizados em dispositivos móveis e, em breve, estarão presentes na TV digital [Souza, 2010]. Essas possibilidades de disseminar os AVs X3D abrem uma gama de oportunidades para uso do mesmo, como por exemplo, fazer propagandas de um apartamento, disponibilizando-o tridimensionalmente para navegação do público alvo através do controle remoto na TV digital, ou disponibilizar o AV no celular do usuário para que, de maneira portátil, o usuário possa navegar na cena.

2.2. O X3D

O X3D possui um conjunto de características próprias para a definição dos ambientes 3D. Nesta seção será apresentada a estrutura do X3D, bem como algumas particularidades deste padrão e ferramentas de exibição de conteúdo produzido com este padrão.

2.2.1. Especificações

As especificações do X3D [Web3D-e, 2009] são uma série de documentos produzidos pelo grupo Web3D que definem e detalham geometrias e comportamentos [Brutzman, 2007]. Devido à característica de extensibilidade do X3D existem múltiplos documentos da especificação que conduzem à evolução coerente de capacidades diversas do X3D. Cada documento pode ser desenvolvido e integrado de forma independente, permitindo atualizações anuais e crescimento estável da linguagem. A produção destes documentos ocorre por meio de um processo livre e colaborativo de membros voluntários do grupo Web3D. Cada ano, as novas funcionalidades propostas são implementadas, avaliadas, especificadas formalmente e enviadas para a ISO (*International Organization for Standardization*), a fim de serem revisadas e confirmadas.

Dentre os conteúdos abordados nas especificações, existem documentos apropriados para a explicação da codificação de um arquivo X3D, incluindo vários aspectos como: tipos de campos, tipos de nós, expressões que definem rotas, etc. Um fator interessante é que as funcionalidades primitivas (como nós e campos) são especificadas de forma neutra, tornando-as independente de qualquer formato de codificação ou ligação com linguagens de programação. Portanto, diferentes formatos de codificação e as ligações com diferentes linguagens de programação mantêm-se portátil, compatível e funcionalmente equivalente entre si [Brutzman, 2007].

De forma geral, pode-se afirmar que o conjunto de especificações X3D define as funcionalidades do X3D que podem ser interpretadas pelos *browsers*, e consequentemente, de forma indireta, como os mesmos devem ser programados. Do mesmo modo, também são definidos os métodos necessários a serem implementados pelo programador que deseja desenvolver a API em uma linguagem de programação em que a mesma ainda não seja disponibilizada.

2.2.2. Grafo de Cena

O grafo de cena é uma estrutura de dados simples e intuitiva utilizada para representar a idéia de relações entre determinados objetos. Um grafo é formado por nós ou vértices, e arestas interligando-os e representado uma relação.

Os grafos de cena são utilizados para representar o AV. Tais grafos são estruturados de maneira hierárquica correspondendo semântica e espacialmente à cena

X3D. Todas as funcionalidades e características do mundo X3D (cor, geometria, transformações, etc.) são representadas por nós e expressos por um grafo acíclico e direcionado em forma de árvore. Cada nó possui um conjunto de características (translação, rotação, escalonamento, etc.) que podem ou não ser herdadas pelos nós filhos [Raposo, 2007].

Geralmente, nas aplicações em X3D o AV é renderizado percorrendo diretamente o grafo de cena. Através deste percurso de varredura do grafo, todas as informações (localização e cores dos objetos, ponto de vista do usuário, etc.) e alterações de comportamento da cena são obtidas e processadas, resultando na atualização da mesma [Brutzman, 2007]. Quando determinado nó do grafo não estiver sendo utilizado pelo usuário, por não estar em seu campo de visualização ou contexto de utilização da cena, o nó e toda sua subárvore serão descartados. Deste modo, é essencial ter uma boa noção de organização espacial para obter o melhor desempenho da aplicação [Raposo, 2007].

Como exemplo, na Figura 2.2 é apresentado o grafo de cena de um AV que representa um hotel. O grafo se encontra bem estruturado e desenhado com relação às coordenadas espaciais reais de um hotel. Este fato otimiza o desempenho, pois pode evitar a renderização desnecessárias de determinados objetos. Por exemplo, observando o grafo do hotel da Figura 2.2, nota-se que quando o usuário estiver no corredor esquerdo do “segundo andar”, o *browser* não precisará percorrer a subárvore do “corredor direito” do “segundo andar”, tampouco do “térreo” e do “primeiro andar”. Porém, se o grafo estiver mal estruturado, o *browser* percorreria subárvores desnecessárias antes de renderizar o quadro da cena.

2.2.3. Padrão de Arquivos XML

XML (*Extensible Markup Language*) é um conjunto de regras que dão suporte a gerar novos formatos de arquivo, através da criação de novas DTD's (*Document Type Definition*), que possibilitam a estruturação de quaisquer dados. Trata-se de uma linguagem modular que permite a definição de novos formatos de documentos por combinação e reutilização de outros formatos.

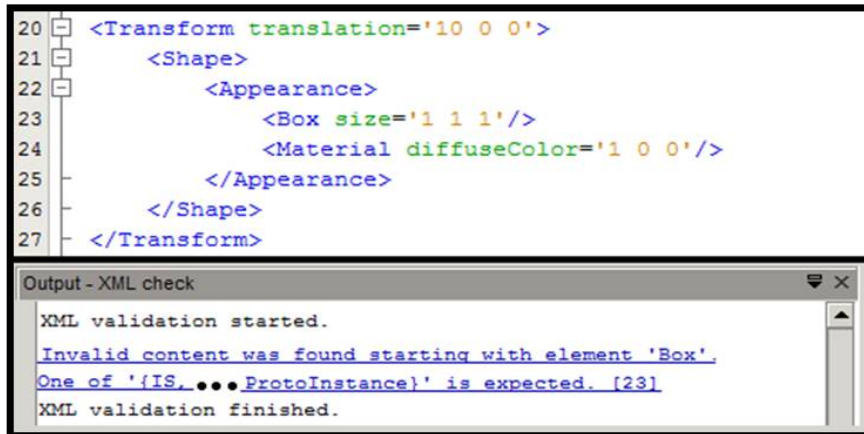
O XML possui sintaxe similar a HTML (*HyperText Markup Language*), utilizando *tags* (palavras entre ‘<’ e ‘>’) para expressar elementos e atributos na forma *nome=‘valor’*, tornando-a de fácil compreensão. Esta característica de identificação da informação de forma viável faz com que o XML possa ser considerado a base da Web Semântica [Garcia, 2009] [Boss, 1999]. Outra característica importante presente no X3D é o *XML Validation* que é representado pelo nome de “*X3D Schematron Validation and Quality Assurance*” [Web3D-a, 2009]. Trata-se de conjuntos de regras com as quais a cena X3D (arquivo codificado em XML) é submetida para detecção de erros de sintaxe dos elementos e atributos XML, considerando também erros relacionados à relação “pais-filhos” dos nós X3D. O X3D DTD e o *X3D Schematron*

são dois desses conjuntos de regras. O primeiro possibilita testes mais precisos de elementos, contudo possui testes mais fracos de atributos, pois considera-os como textos simples. O *X3D Schematron* já possibilita testes mais precisos tanto para elementos, quanto para atributos [Brutzman, 2007]. A Figura 2.3 mostra um exemplo de validação de uma cena X3D com o auxílio do *X3D Edit* mostrando o código errado e a mensagem de erro gerada. O erro se encontra na relação pai-filho do nó *Box*. Este nó, de acordo com a especificação, é filho do nó *Shape*, mas na imagem aparece como sendo filho do nó *Appearance*. A Figura 2.4 contém o código corrigido.



Figura 2.2. Grafo de cena de um AV que representa um hotel.

A Figura 2.4 apresenta parte de um arquivo X3D codificado em XML que descreve um cubo transladado dez unidades no eixo X com cor difusa vermelha, além de largura, altura e profundidade iguais a um. Pode-se notar que os nós X3D (em azul) são expressos como elementos XML e campos X3D (em verde) são expressos como atributos XML. Também pode-se observar que as *tags* podem ser finalizadas de duas maneiras: com '/' na mesma linha (caso dos nós *Box* e *Material da imagem*); ou com tag de finalização do elemento em outra linha (caso dos nós *Transform*, *Shape* e *Appearance da imagem*). Os comentários, presentes na linha 25 da Figura 2.4, são escritos entre "<!--" e "-->", fazendo com que o *parser* do *browser* ignore a interpretação do dado texto.



```

20 <Transform translation='10 0 0'>
21   <Shape>
22     <Appearance>
23       <Box size='1 1 1' />
24       <Material diffuseColor='1 0 0' />
25     </Appearance>
26   </Shape>
27 </Transform>

```

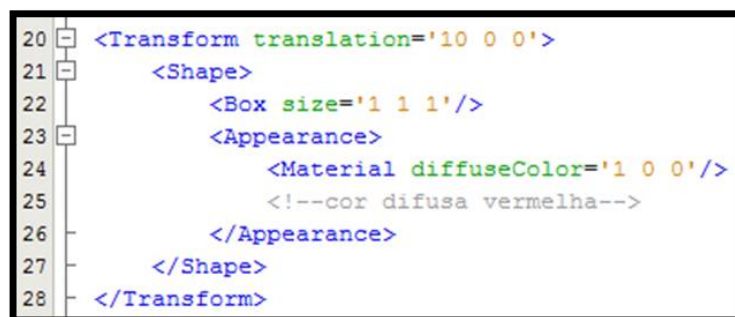
Output - XML check

```

XML validation started.
Invalid content was found starting with element 'Box'.
One of '{IS, ...ProtoInstance}' is expected. [23]
XML validation finished.

```

Figura 2.3. Validação do arquivo X3D e mensagem de erro gerada pelo X3D Schematron.



```

20 <Transform translation='10 0 0'>
21   <Shape>
22     <Box size='1 1 1' />
23     <Appearance>
24       <Material diffuseColor='1 0 0' />
25       <!--cor difusa vermelha-->
26     </Appearance>
27   </Shape>
28 </Transform>

```

Figura 2.4. Exemplo da descrição de um cubo codificado em X3D - XML.

2.2.4. Browsers X3D

A visualização de cenas gráficas X3D é realizada através de *browsers* específicos que consistem em aplicações capazes de interpretar e processar as cenas (arquivos X3D), apresentando os modelos tridimensionais, animados ou não, e permitindo interações do usuário com os objetos. Estes *browsers*, comumente chamados de navegadores ou *players*, podem se apresentar como *plugins* em navegadores Web, como o Internet Explorer ou Mozilla Firefox (Figura 2.5), ou como aplicações independentes [Web3D-d, 2009].

Os *browsers* para X3D são implementados visando interpretar funcionalidades definidas em sua especificação. Assim, desenvolvedores destes *browsers* têm autonomia para seleção de funcionalidades a serem suportadas, além das básicas, e também podem adicionar novas. Devido a este fato, é importante que os autores de ambientes X3D optem por *browsers* que ofereçam suporte a todas as funcionalidades necessárias em seu

ambiente. Uma lista de alguns *browsers* associados a Web3D e funcionalidades suportadas por eles encontra-se disponível em [Web3D-c, 2009].

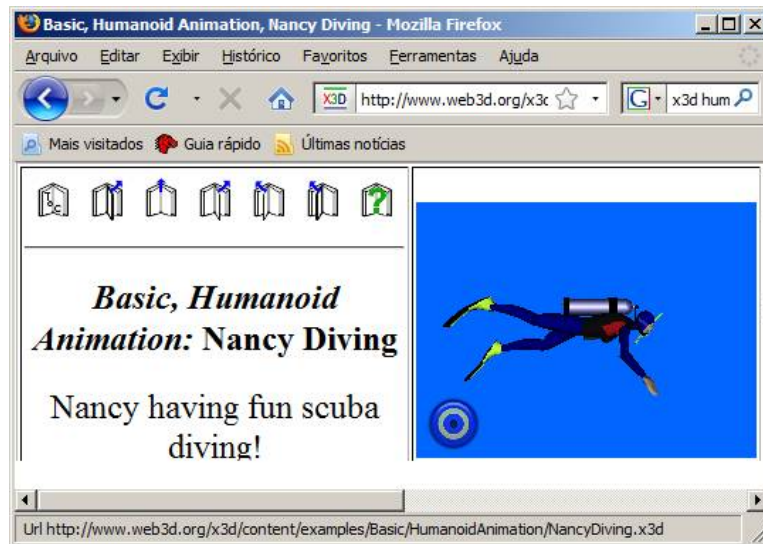


Figura 2.5. Exemplo de cena X3D disponibilizado pela Web3D [Web3D-b, 2009], visualizada no plugin BS Contact no Mozilla Firefox.

Para apresentar a cena gráfica ao usuário, os *browsers* X3D passam por diferentes etapas de processamento como identificação, interpretação, gerenciamento, etc. Na Figura 2.6 é possível observar o esquema de funcionamento de um *browser* X3D. Observa-se em ① que o navegador identifica qual formato de codificação do arquivo e direciona-o para os interpretadores (*parsers*). É importante observar que os *browsers* não são obrigados a interpretar todos os tipos de codificação, mas geralmente o formato VRML é reconhecido, além do X3D codificado em XML ou binário. Após a interpretação do arquivo, os nós são criados e enviados para o gerenciador do grafo de cena ② encarregado de gerar a árvore do grafo de cena e atualizá-la a medida em que o AV é modificado. As ferramentas de *script*, que podem ser linguagens de programação ou *script*, é um dos meios possíveis de se adicionar interatividade ao AV. A SAI (*Scene Authoring Interface*) [Web3D-g, 2009] define como essas API's (*Application Programmer Interface*) podem funcionar, permitindo aos autores criarem códigos de *script* em diferentes linguagens de programação para diferentes sistemas operacionais e navegadores ③. Os eventos podem ser ativados pelas ferramentas de *script* ou podem iniciar uma animação na cena ④, comunicando-se com o gerenciador do grafo de cena e conseqüentemente alterando-o. A árvore do grafo de cena ⑤ é percorrida rapidamente, detectando as alterações ocorridas tais como a diferente localização do usuário e as animações ativadas, redesenhando o AV em diferentes perspectivas em tempo real.

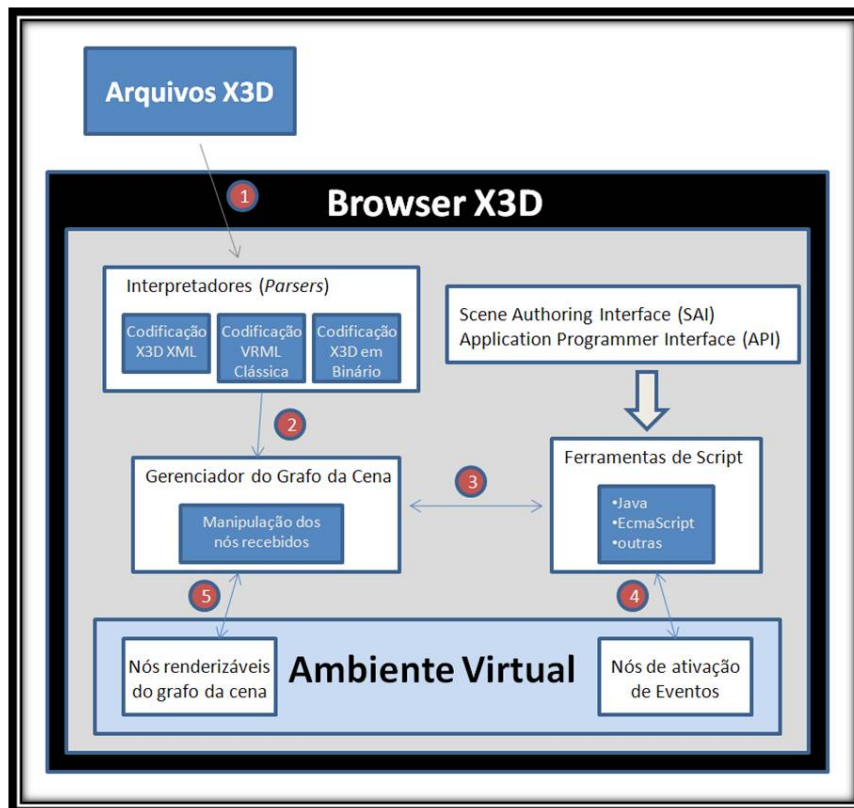


Figura 2.6. Funcionamento de um *browser X3D*. Adaptado de [Brutzman, 2007].

Exemplos de *browsers* de código aberto são: Xj3D, OpenVRML, FreeWRL. Outro visualizador *open source* interessante é o H3Dviewer [H3D, 2010], pois utiliza padrões abertos como OpenGL e X3D, para prover um grafo de cena integrado que suporte tanto gráficos X3D quanto sensibilidade háptica. Há também o Jinx [Soares, 2004] para simulação de RV imersiva, que facilita o desenvolvimento de CAVEs (*Cave Automatic Virtual Environment*) baseado em aglomerados de computadores.

Recentemente foi desenvolvida uma API JavaScript de código aberto chamada O3D que contém uma versão de *software* da OpenGL para qualquer computador (muito importante para aqueles que não possuem aceleração gráfica por *hardware*), para auxiliar no desenvolvimento de aplicações gráficas 3D para a Web. Dois pontos principais são considerados vantagens por muitos desenvolvedores: a API é toda produzida na linguagem JavaScript e existe um único *plugin* O3D de visualização (*browser*), diferente do X3D. Por outro lado, os criadores de ambiente virtuais O3D ficam restritos às funcionalidades que o *plugin* O3D pode oferecer, enquanto que desenvolvedores X3D podem expandir e implementar funcionalidades diversas utilizando o *plugin* que melhor lhe convier.

2.3. Construindo Gradativamente um Ambiente Virtual X3D

2.3.1. Estrutura de Arquivo e Extensões

O ambiente virtual X3D pode ser descrito utilizando o formato de codificação XML ou VRML Clássico, com as extensões de arquivo .x3d e .x3dv, respectivamente. Ambos os tipos de codificação geram o mesmo grafo de cena trabalhando de forma robusta e consistente [Brutzman, 2007]. Uma comparação entre a sintaxe de codificação XML e VRML Clássica pode ser feita pela observação da Figura 2.7.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN"
3 "http://www.web3d.org/specifications/x3d-3.1.dtd">
4 <X3D profile='Immersive' version='3.1'
5 xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance'
6 xsd:noNamespaceSchemaLocation='http://www.web3d.org/specifications/x3d-3.1.xsd'>
7   <head>
8     <component name='Geospatial' level='1'/>
9     <component name='H-Anim' level='1'/>
10    <meta content='exemplo.x3d' name='arquivo'/>
11    <meta content='Eduardo de Lucena Falcão' name='autor'/>
12  </head>
13  <Scene>
14    <!--Nós que descrevem a cena são inseridos aqui-->
15  </Scene>
16 </X3D>

```

```

1 #X3D V3.1 utf8
2
3 PROFILE Immersive
4
5 COMPONENT Geospatial:1
6 COMPONENT H-Anim:1
7
8 META "autor" "Eduardo de Lucena Falcão"
9 META "arquivo" "exemplo.x3d"
10
11 #Nós que descrevem a cena são inseridos aqui

```

Figura 2.7. Exemplo das codificações .x3d XML (no alto) e .x3dv VRML Clássica (embaixo).

Estes dois formatos de codificação podem ser comprimidos em um arquivo de extensão .x3db (X3D em formato binário), tendo como algumas de suas vantagens:

- assinar digitalmente o arquivo para provar ser de sua autoria;
- criptografar o conteúdo do arquivo para proteger o código;

- redução do tamanho do arquivo, e conseqüentemente um fluxo mais rápido de dados através da Internet;
- carregamento da cena mais veloz em tempo de execução.

Neste texto, o formato de codificação XML será adotado como padrão para os exemplos e explicações. Em relação à estrutura dos arquivos X3D, esta contém [Brutzman, 2007]:

- Cabeçalho XML (inexistente na codificação VRML Clássica): contém informações de configuração do XML:
- Cabeçalho X3D: contém o nó Raiz X3D (implícito na codificação VRML Clássica) com:
 - definição da versão X3D;
 - definição do Perfil a ser utilizado (ver seção 2.3.3);
 - mais informações de configuração XML (inexistente na codificação VRML Clássica).
- Definição dos Componentes (estrutura opcional e múltipla): permite adicionar componentes não especificadas pelo nível do Perfil.
- Informações META (estrutura opcional e múltipla): permite comentários adicionais em relação ao desenvolvimento do AV.
- Nó Scene (implícito na codificação VRML Clássica): seção onde são inseridos os nós que descrevem a cena.

2.3.2. Nós e Campos para Objetos e Ações no AV

Nós X3D são os elementos fundamentais que compõem o grafo de cena. Cada nó é formado por uma sequência dos campos que representa. Estes nós e seus agrupamentos descrevem as funcionalidades disponibilizadas pelo X3D utilizadas para descrição dos objetos e comportamentos nos ambientes virtuais. Alguns exemplos de nós são:

- *PointLight*: nó de iluminação que descreve uma luz pontual;
- *Box*: nó de geometria que descreve um cubo;
- *NavigationInfo*: nó de navegação que descreve os meios e características de navegação.

Na codificação X3D é possível a reutilização de nós e agrupamentos já definidos pelo autor da cena, evitando a replicação de código e aumentando o desempenho do grafo de cena. Esta reutilização pode ocorrer por meio do uso de comandos especiais: DEF e USE. Para atribuir uma referência a um nó X3D basta utilizar o comando DEF="nomeDoNó" na primeira ocorrência do mesmo. Caso o autor da cena deseje

reutilizar os mesmos atributos de um nó já definido, será necessário copiar estas características através do comando USE="nomeDoNó". Estes comandos provocam uma ligação entre o nó definido e suas réplicas, de forma que qualquer característica alterada em um deles será alterada também nos demais. Essas referências também podem ser utilizadas para a definição das rotas do AV (abordada no tópico 2.3.4). A Figura 2.8 exemplifica a utilização dos comandos DEF/USE.

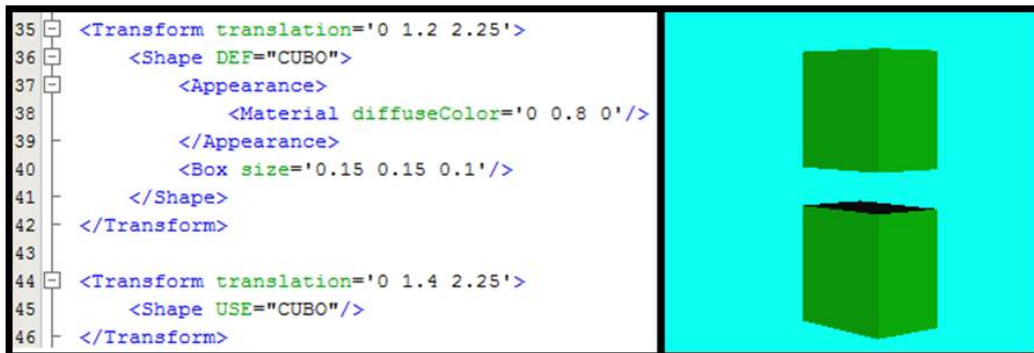


Figura 2.8. Utilização dos comandos DEF/USE para atribuição e reutilização de referências.

Os **campos** servem para especificar os atributos e características dos nós X3D. Às vezes alguns nós podem ser utilizados como campos. Os campos que possuem 'd'/'D' e 'f'/'F' ao final do nome significam respectivamente a precisão *double* e *float*. *Single* e *Multiple Field* (**SF** e **MF**) significam, respectivamente, um ou vários valores do tipo especificado. Os tipos de campos são representados conforme apresentado na Tabela 2.1.

Tabela 2.1. Tipos de campos X3D.

SF (<i>Single Field</i>) Ou MF (<i>Multiple Field</i>)	+	Bool Color, ColorRGBA Double, Float, Int32 Image Matrix3D, Matrix3F, Matrix4D, Matrix4F Node Rotation String Time Vec2d, Vec2f, Vec3d, Vec3f, Vec4d, Vec4f
--	---	---

Na Figura 2.9 encontram-se os campos e seus respectivos valores padrão dos nós *PointLight* e *Box* utilizado para exemplificar os nós. Se o autor não definir os valores dos campos, valores padrão serão assumidos. Todos os nós e campos X3D e suas respectivas descrições de como funcionam e como são codificados encontram-se em [Web3D-g, 2009].

```

PointLight : X3DLightNode {
  SFFloat [in,out] ambientIntensity 0      [0,1]
  SFVec3f [in,out] attenuation    1 0 0 [0,∞)
  SFColor  [in,out] color          1 1 1 [0,1]
  SFBool   [in,out] global         TRUE
  SFFloat  [in,out] intensity      1      [0,1]
  SFVec3f  [in,out] location       0 0 0 (-∞,∞)
  SFNode   [in,out] metadata       NULL  [X3DMetadataObject]
  SFBool   [in,out] on             TRUE
  SFFloat  [in,out] radius         100   [0,∞)
}

Box : X3DGeometryNode {
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFVec3f []      size      2 2 2 (0,∞)
  SFBool  []      solid     TRUE
}

```

Figura 2.9. Campos dos nós *PointLight* e *Box*.

2.3.3. Componentes e Perfis para Modularização do AV

Componente é um conjunto de nós com a mesma funcionalidade. Cada componente é dividido em níveis que descrevem um aumento de suporte às funcionalidades. No arquivo X3D podem-se declarar componentes que ainda não foram incluídos pelo nível do perfil. Algumas vezes, os autores de cenas X3D declaram o perfil básico (*Core Profile*), declarando um por um os componentes adicionais e seus respectivos níveis a serem utilizados, para que mais *browsers* X3D suportem a dada cena [Brutzman, 2007]. No mês de novembro de 2009 estavam presentes nas especificações do X3D 34 componentes [Web3D-f, 2009].

O perfil é um conjunto de componentes. Eles foram definidos objetivando agrupar conjuntos de funcionalidades que geralmente são utilizadas para desenvolvimento de certos tipos de cenas. Diferentes perfis provêm níveis intermediários de suporte às funcionalidades do X3D, para que os desenvolvedores de *browsers* não precisem implementar grande parte da especificação de uma vez [Brutzman, 2007]. Deste modo, alguns AVs podem ser carregados mais rapidamente de acordo com o nível do perfil especificado pelo autor da cena. Na Figura 2.10 estão representados os níveis dos perfis oferecidos pelo X3D.

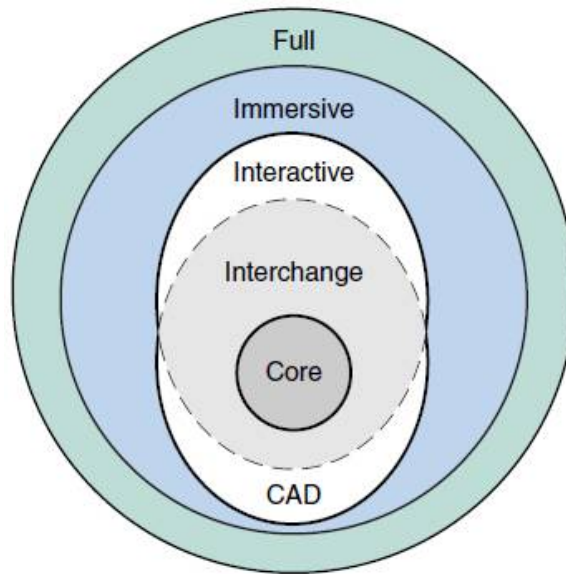


Figura 2.10. Representação gráfica dos níveis de perfil oferecido pelo X3D [Brutzman, 2007].

Os perfis do X3D são:

- *Core Profile*: possui basicamente os nós *Metadata*. Geralmente utilizado para descrever cenas pouco complexas, definindo as funcionalidades a serem utilizadas por meio de explicitação dos componentes.
- *Interchange Profile*: provê suporte a formas geométricas com os nós *Geometry* e *Appearance (material e texture)*.
- *Interactive Profile*: adiciona ao perfil *Interchange*, os nós que provêm interatividade ao AV.
- *CadInterchange Profile*: perfil que provê suporte à importação dos modelos de CAD.
- *Immersive*: adiciona ao perfil *Interactive*, os nós que dão suporte a geometria 2D, efeitos de ambiente, eventos, etc.
- *Full*: oferece funcionalidade a todos nós definidos na especificação X3D.

2.3.4. Interatividade com Eventos e Rotas

A interatividade de uma cena X3D pode ser caracterizada por mudanças de posição, orientação, tamanho, cores ou outras características pertinentes ao campo do nó X3D, resultando em uma animação. Eventos e rotas são as funcionalidades do X3D que permitem adicionar tais comportamentos a objetos do AV de forma simples. Eventos

são ativados pela interação do usuário com sensores ou outros comandos programados pelo autor da cena. Tais eventos podem ser propagados para outros nós por meio de **rotas**. Para isso é necessário ter duas informações dos campos: tipo de acesso (*inputOnly*, *outputOnly*, *inputOutput*, *initializeOnly*) ao campo e os tipos do campo de origem e destino.

- *inputOnly*: permite apenas receber (valores) eventos;
- *outputOnly*: permite apenas enviar (valores) eventos;
- *inputOutput*: permite receber e enviar (valores) eventos;
- *initializeOnly*: tal campo não pode receber nem enviar (valores) eventos, o valor deste campo pode apenas ser inicializado.

O tipo do campo do nó de origem precisa ser igual ao tipo do campo do nó de destino. Deste modo, quando o usuário ativar um sensor, o valor do campo do nó de origem será roteado (encaminhado) para o campo do nó de destino. As rotas também podem ser utilizadas para desencadear eventos em cascata. A sintaxe de uma rota é:

```
<ROUTE fromNode="nóOrigem" fromField="campoOrigem" toNode="nóDestino" toField="campoDestino"/>
```

A Figura 2.11 ilustra a utilização de rotas entre o nó *ProximitySensor* e dois nós *PointLight* para adicionar o comportamento de detecção do usuário na sala do dado AV. Nota-se que os campos roteados são (obrigatoriamente) do mesmo tipo (*SFBool*). O campo *isActive* (*outputOnly*) do nó *ProximitySensor* é ativado com a presença do usuário na sala e tem o seu valor alterado para *true*. Assim que ocorre esta mudança no valor do campo *isActive* (de origem), um evento é lançado e seu valor é roteado para os campos *on* (de destino), que passam a assumir o novo valor de *isActive*.

2.3.5. Nós tipo *Script* e Acesso a Aplicações Externas com SAI

Na maioria das vezes os mundos codificados em X3D são desenvolvidos com o objetivo de representar o mundo real. Como os objetos do mundo real mudam de maneira dinâmica e respondem a determinadas ações de seus habitantes, o mundo virtual que o representa deverá dispor das mesmas interações.

A SAI (*Scene Access Interface*) é o conjunto de serviços especificados pelo grupo Web3D para que um autor possa acessar e alterar o grafo de cena enquanto ele é executado [Web3D-g, 2009]. Com esta API é possível ler os dados dos nós da cena para manipulá-los e efetuar as alterações desejadas em tempo de execução. Este acesso pode ser de forma interna ao arquivo X3D por meio de linguagem de *script* como ECMAScript, ou externamente por linguagens de programação tais como JAVA e C++. Caso a API não esteja implementada na linguagem de programação desejada, as especificações da SAI [Web3D-g, 2009] definem como o desenvolvedor deverá proceder para implementá-la.

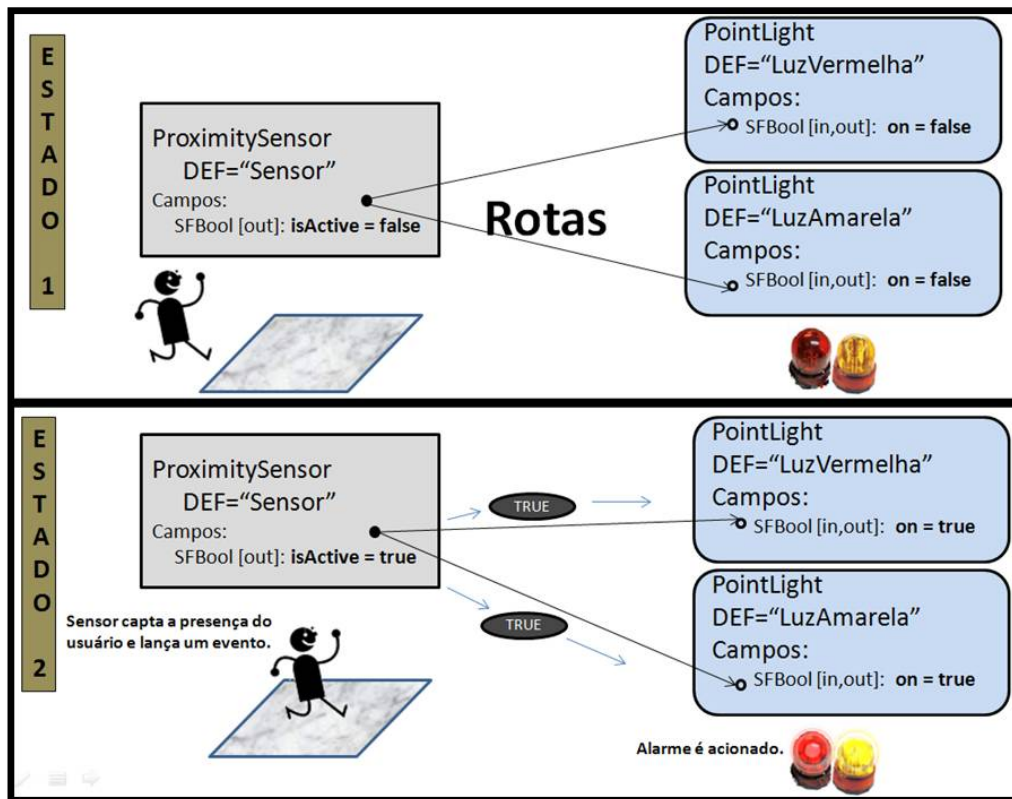


Figura 2.11. Ilustração do funcionamento das rotas.

Para intermediar a comunicação entre o grafo de cena e a linguagem de programação é utilizado o nó *Script*. O nó *Script* possui:

- o campo *url*, que especificará o caminho do código da linguagem de programação que fará as computações dos dados recebidos;
- os campos (criados pelo autor da cena) que receberão eventos de outros nós;
- os campos (criados pelo autor da cena) que enviarão eventos com os valores já computados para outros nós.

Neste texto será abordada a comunicação com o grafo de cena do ambiente virtual X3D, por meio da linguagem de programação Java. O pacote utilizado é o `org.web3d.x3d.sai` do *browser* Xj3D. Alguns *browsers* reconhecem apenas o pacote `org.web3d.x3d.sai` e outros o `org.web3d.vrml`. Por isso, antes de decidir qual pacote utilizar é importante verificar o suporte dos *browsers* ao mesmo. Para mostrar a utilização da SAI + nó *Script*, foi criada uma cena X3D que consiste de um painel branco contendo um texto e um botão verde (sensor) que quando clicado abre uma caixa de texto (*JoptionPane*) para o usuário inserir o novo texto a ser exibido pelo painel (Figura 2.12). Para a construção do painel foi utilizado os nós *Box*, *Appearance*,

Material, e a transformação de translação. Para adicionar o sensor de toque no botão verde, basta utilizar o nó *TouchSensor* dentro do escopo de nós como *Transform* ou *Group* que o conectará com a geometria definida, e para adicionar o texto é utilizado o nó *Text*. Este código está representado na Figura 2.13.

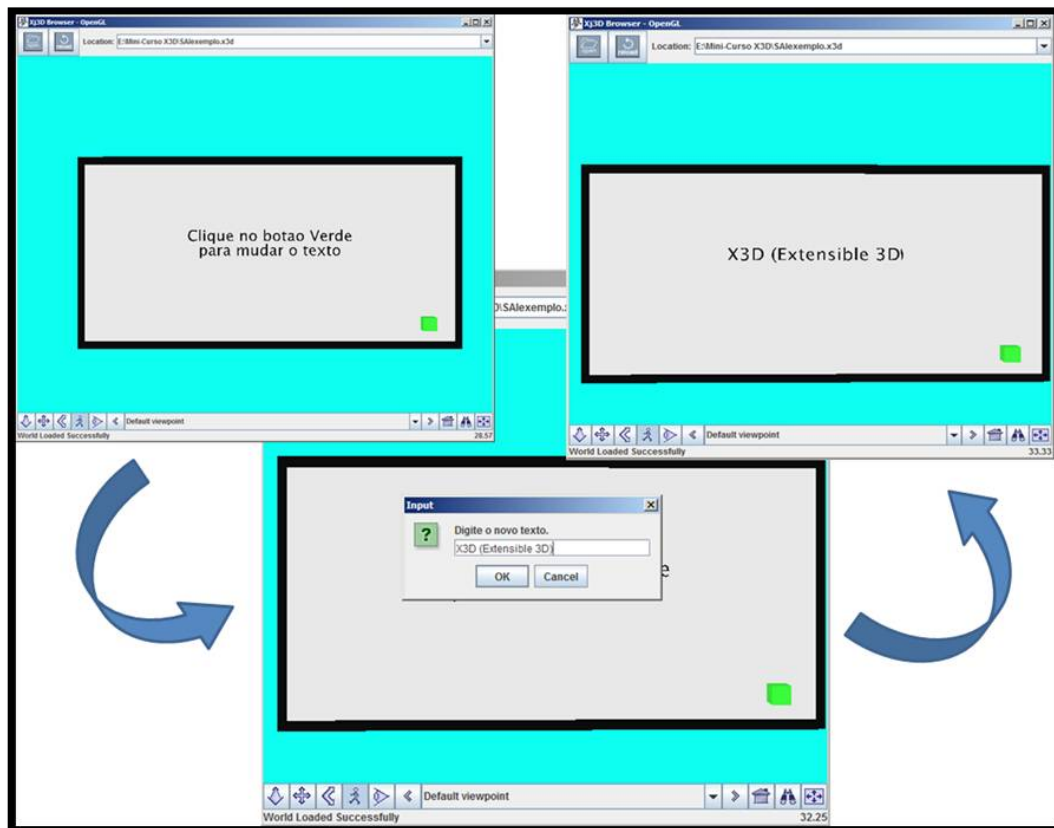


Figura 2.12. Exemplo de utilização utilização da SAI com o nó Script.

O nó *Script* é utilizado para fazer a comunicação do X3D com o JAVA. Através de rotas serão enviado os eventos lançados pelo campo *isActive* do nó *TouchSensor* para um campo “cliqueu” no nó *Script* criado pelo autor. O campo *isActive* (*SFBool*, *outputOnly*) enviará *true* quando o sensor for ativado por meio do clique, e esse valor será roteado para o campo “cliqueu” (*SFBool*, *inputOnly*). O código JAVA da classe *InsertText.class* possui o controle dos campos do nó *Script*, e as instruções necessárias para alterar os seus valores. O campo “novoTexto” (*MFString*, *outputOnly*) do nó *Script* enviará o novo texto para o campo *string* (*MFString*, *inputOutput*) do nó *Texto*. A Figura 2.14 contém este trecho do código X3D.

```

<Transform translation='1.8 1.2 2.25'>
  <TouchSensor DEF='Botao' />
  <Shape>
    <Appearance>
      <Material diffuseColor='0 0.8 0' />
    </Appearance>
    <Box size='0.15 0.15 0.1' />
  </Shape>
</Transform>

<Transform translation='0 2.3 2.26'>
  <Shape>
    <Text DEF='Texto' string='"Clique no botao Verde" para mudar o texto"'>
      <FontStyle justify='MIDDLE MIDDLE' size='0.2' />
    </Text>
    <Appearance>
      <Material diffuseColor='0 0 0' />
    </Appearance>
  </Shape>
</Transform>

```

Figura 2.13. Código X3D referente ao painel e texto do AV presente na Figura 2.12.

```

<Script DEF="scriptJava" url="InsertText.class">
  <field accessType='inputOnly' name='cliqueu' type='SFBool' />
  <field accessType='outputOnly' name='novoTexto' type='MFString' />
</Script>

<ROUTE fromNode="Botao" fromField="isActive" toNode="scriptJava" toField="cliqueu" />
<ROUTE fromNode="scriptJava" fromField="novoTexto" toNode="Texto" toField="string" />

```

Figura 2.14. Trecho do código referente à comunicação do X3D com Java.

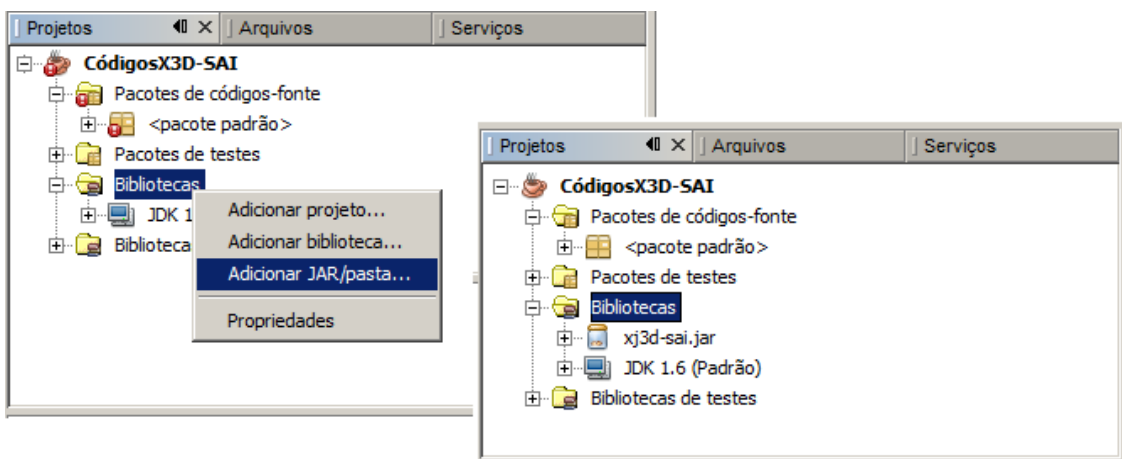


Figura 2.15. Adicionando a biblioteca xj3d-sai.jar ao projeto, utilizando a IDE NetBeans.

Para utilizar os métodos da SAI é preciso reconfigurar o *classpath* do projeto, adicionando a biblioteca *xj3d-sai.jar* presente na pasta *jars* do diretório do Xj3D (Figura 2.15). Para a computação dos valores dos campos X3D, foi criada a classe *InsertText.class* (Figura 2.16). Ela precisa implementar a interface *X3DScriptImplementation* para dizer que a classe implementada pode funcionar como um *Script* dentro do grafo de cena, e *X3DFieldEventListener* para detectar os eventos recebidos pelos campos do nó *Script*.

```
import java.util.Map;
import javax.swing.JOptionPane;
import org.web3d.x3d.sai.*;

public class InsertText implements X3DScriptImplementation, X3DFieldEventListener{

    private SFFBool clicou;
    private MFString novoTexto;

    public void setBrowser(Browser arg0) {}
    public void eventsProcessed() {}

    public void initialize() {
        clicou.addX3DEventListener(this);
    }

    public void setFields(X3DScriptNode externalView, Map fields) {
        novoTexto = (MFString) fields.get("novoTexto");
        clicou = (SFFBool) fields.get("clicou");
    }

    public void shutdown() {
        clicou.removeX3DEventListener(this);
    }

    public void readableFieldChanged(X3DFieldEvent e) {
        if(e.getSource() == clicou && clicou.getValue() == true){
            String text [] = new String [1];
            text [0] = JOptionPane.showInputDialog("Digite o novo texto.");
            novoTexto.setValue(1,text);
        }
    }
}
```

Figura 2.16. Código JAVA que adiciona interação ao ambiente virtual.

No exemplo, os métodos necessários são:

- *setBrowser()* – define a instância do *browser* a ser utilizada;
- *setFields()* - define a lista de campos que é usada pelo nó *Script* no arquivo X3D, para que possam ser usados no código JAVA;

- *initialize()* – notificação que o *Script* completou sua inicialização e deve inicializar internamente;
- *eventsProcessed()* – notificação que todos os eventos em cascata terminaram sua execução;
- *shutdown()* - notificação que este *Script* não está mais em uso e deve liberar recursos;
- *readableFieldChanged()* – método executado quando um evento é lançado. Captado pelos campos *inputOnly* ou *inputOutput* do nó *Script*, utilizado para efetuar alterações em qualquer campo *outputOnly* ou *inputOutput* do nó *Script*. É no escopo deste método que deve ser escrito o código Java para tratamento do evento.

Todos estes métodos são implementados pela interface *X3DScriptImplementation*, por isso precisam estar presente no código mas neste exemplo, os métodos *setBrowser()* e *eventsProcessed()* não precisaram ser utilizados. A documentação destes métodos e da API encontram-se em [Xj3D, 2009].

No arquivo *InsertText.class* foram criados as variáveis “*novoTexto(MFString)*” e “*clicou(SFBool)*”, para que no método *setFields()* seja adicionado uma conexão entre estas variáveis e os campos do nó *Script* (que são de mesmo nome) presentes no arquivo X3D. Tendo esta interligação, pode-se agora captar todos os eventos lançados por estes campos (presentes no nó *Script*), e manipulá-los no código JAVA da maneira que o autor desejar. Cada evento lançado pelos campos do nó *Script*, ativa o método *readableFieldChanged()*, e a partir deste método podemos verificar qual campo recebeu um evento para executar determinada instrução.

A sequência em que os eventos e ações ocorrem nos exemplos citados nesta seção, pode ser observada na Figura 2.17. Nela o campo *isActive* do nó “Botão”/*TouchSensor* (detecção do clique) refere-se ao botão verde que será alterado para *true* quando o botão do *mouse* for pressionado, e retornará ao valor *false* quando for solto. Com isso, cada vez que há a mudança de valor do campo, um evento é lançado e roteado (se houver rota) para outro campo de outro nó.

O método *readableFieldChanged()* permite detectar que um evento foi roteado a um dos campos *inputOnly* ou *inputOutput* do nó *Script*. Assim, será realizada uma comparação: se o campo em que o evento chegou é o campo “*clicou*” (*inputOnly*/campo criado no nó *Script* que recebe os eventos do campo *isActive* do nó *TouchSensor*, por meio de rota) e o valor deste evento for *true*, então aparecerá um *JOptionPane* pedindo para o usuário digitar o novo texto.

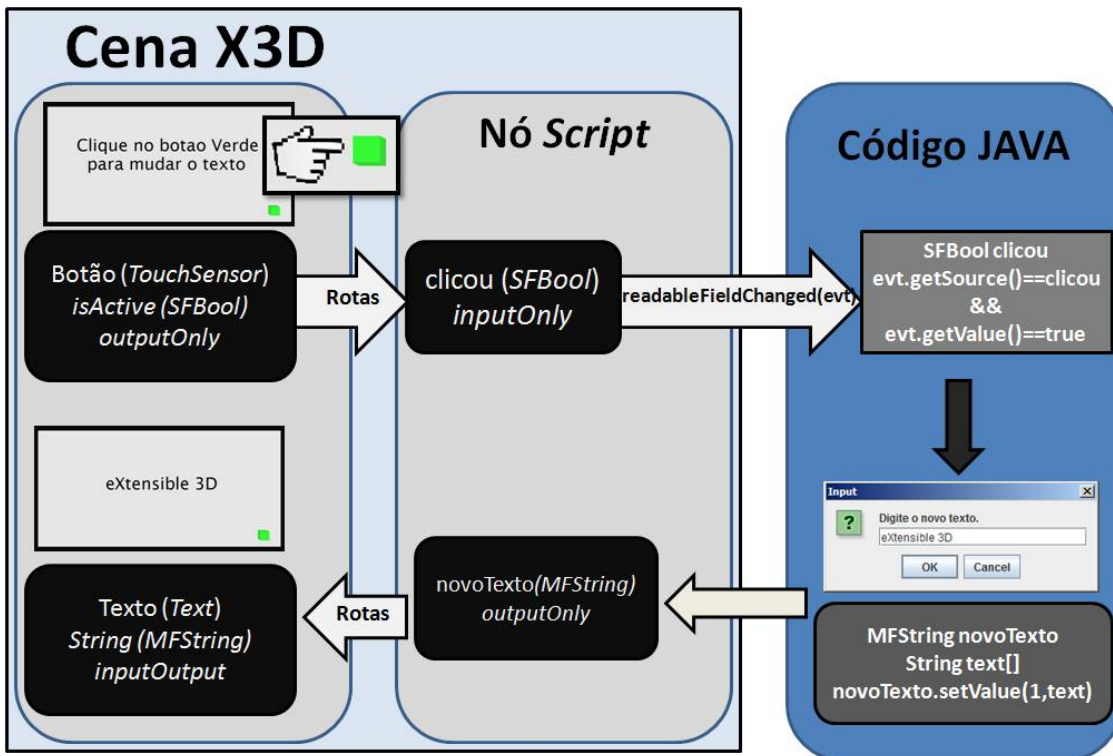


Figura 2.17. Descrição geral do exemplo utilizado nesta seção.

A mudança no valor da *string* recebida pelo usuário será atribuída à variável “novoTexto” (*outputOnly*) através do método *setValue()*. Como existe uma rota entre o campo “novoTexto” e o campo *string* (*inputOutput*) do nó “Texto”/*Text*, esse valor será propagado para o campo *string* e um novo texto aparecerá no painel.

2.4. Ferramentas de Auxílio ao Desenvolvimento

Para edição de AV, o *X3D Edit* (<http://www.web3d.org/x3d/content/README.X3D-Edit.html>) é uma ferramenta gratuita que possui um conjunto de funcionalidades para ajudar autores no desenvolvimento de ambientes virtuais tridimensionais interativos. Dentre suas funcionalidades, as principais são o *browser Xj3D* acoplado ao programa, o editor de texto que permite atribuir cor aos nós e campos X3D ajudando o autor a visualizar os efeitos, a lista de componentes X3D (com dicas de como eles funcionam) e a validação X3D, dentre outras funcionalidades (Figura 2.18). Outro editor X3D disponível é o *WhiteDune* (<http://packages.debian.org/pt/sid/whitedune>), um modelador e editor que permite a visualização e edição dos modelos tridimensionais, provendo um esboço da hierarquia do grafo de cena do mundo X3D.

Sabe-se que os ambientes virtuais são compostos por formas geométricas que precisam ser descritas pelo autor da cena. Algumas cenas X3D utilizam objetos de descrição complexa. Para facilitar a construção desses objetos são utilizados programas

de modelagem tridimensional, que possuem ferramentas que auxiliam na criação dos modelos. Exemplos destes programas são: *Blender*, *Maya*, *SketchUp*, *3D Studio Max*, *AutoCAD*, entre outros. O *Blender* é o mais indicado, por ser um programa gratuito, de código aberto, multiplataforma, e possui exportador para arquivos de extensão X3D. Após a exportação dos arquivos é possível acessá-los e editar qualquer propriedade do modelo.

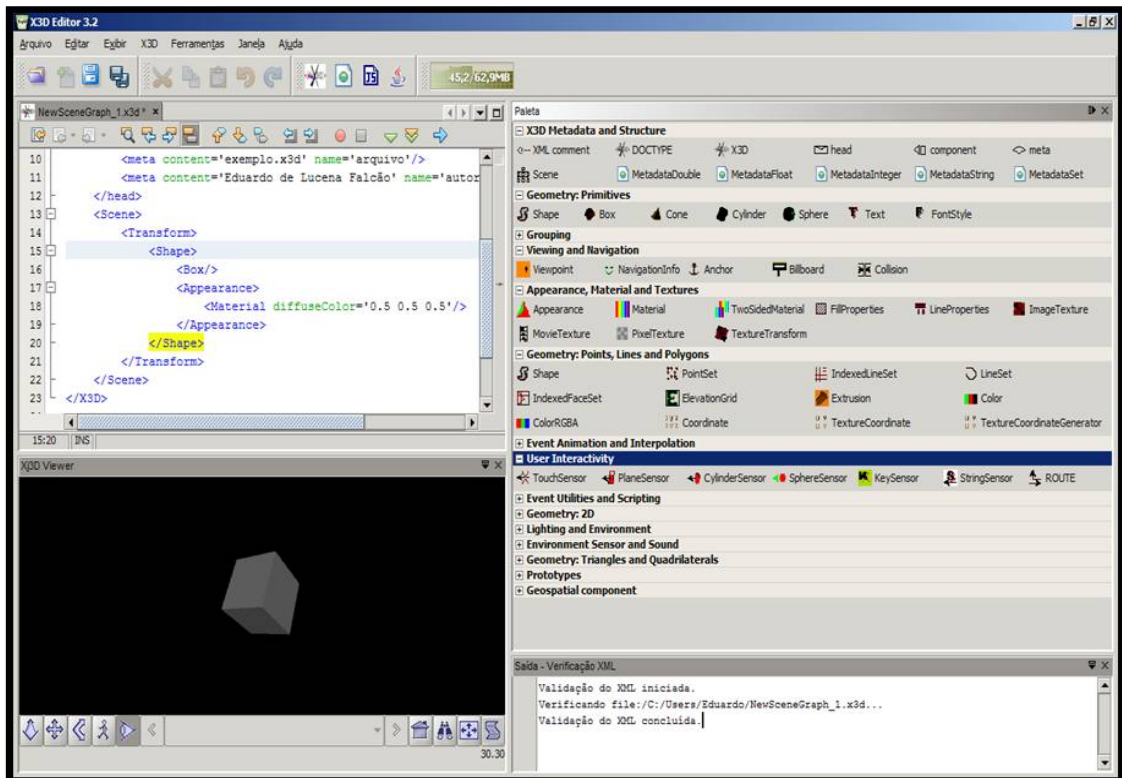


Figura 2.18. Interface do *X3D Edit 3.2*.

Os conversores de formato são programas que possuem a capacidade de converter entre os formatos de arquivo X3D (.x3d, .x3dv) e VRML (.wrl). Por meio deles, após a evolução do VRML para o X3D, os autores que possuem ambientes virtuais no formato .wrl não precisam reescrevê-los na linguagem de descrição .x3d. Neste contexto, as ferramentas de conversão tornam-se importantes pois alguns *browsers* trabalham apenas com um formato de arquivo: o .x3d ou o .wrl. Outra funcionalidade importante é a conversão de arquivos para o formato binário (.x3db). Exemplos destes programas são: *X3D Edit*, *Xj3D Converter*, *Instant Reality X3D encoding converter* [Instant Labs, 2009] e *NIST VRML to X3D Translator*, dentre outros.

2.5. Tópicos Específicos em X3D

2.5.1. Expansão de Funcionalidades

A prototipagem é considerada um sofisticado e poderoso mecanismo de extensão do X3D. Através dos nós protótipos os desenvolvedores têm a possibilidade de customizar novos nós a partir de outros nós X3D e/ou outros protótipos. Assim, este tipo de nó concede uma maior liberdade e possibilidade de expansão da própria linguagem, permitindo que o desenvolvedor possa construir, declarar e instanciar nós que melhor se adequem a necessidade do seu ambiente.

As instâncias dos nós protótipos correspondem a um tipo específico de nó X3D que é definido pelo primeiro nó do encapsulamento. Este protótipo pode ser usado em qualquer parte do grafo de cena que seja apropriado para seu tipo. Por exemplo, se o primeiro nó do encapsulamento for do tipo *Appearance*, o protótipo pode ser instanciado em um nó do tipo *Shape* ou onde mais for permitido.

Um protótipo precisa ser declarado para depois ser instanciado. A declaração de um protótipo define tanto sua interface quanto seu corpo. A interface corresponde aos campos do nó que servem para passagem de informação. O corpo do protótipo consiste no conjunto de nós X3D que são instanciados quando o protótipo é criado. Esta declaração é realizada através do nó *ProtoDeclare* e possui a sintaxe apresentada na Tabela 2.2.

Tabela 2.2. Sintaxe do nó *ProtoDeclare*.

Sintaxe XML (.x3d)

```
<ProtoDeclare>
  <ProtoInterface>
    <!-- Definição de campos-->
  </ProtoInterface>
  <ProtoBody>
    <!--Definição do corpo-->
  </ProtoBody>
</ProtoDeclare>
```

Quando for necessário reutilizar a definição de um protótipo em outros arquivos deve ser utilizado o nó *ExternProtoDeclare*. Cada definição deste nó é praticamente idêntica à definição de uma interface *ProtoDeclare*, bastando adicionar o campo url para recuperar o protótipo original. Este nó também omite a definição dos valores de cada campo, pois já foram explícitos no *ProtoDeclare*. A sintaxe do nó *ExternProtoDeclare* é apresentada na Tabela 2.3.

Tabela 2.3. Sintaxe do nó *ExternProtoDeclare*.**Sintaxe XML (.x3d)**

```
<ExternProtoDeclare name='' url=''>
    <!-- Definição de campos sem adição dos valores-->
</ExternProtoDeclare>
```

O nó *ProtoInstance* instancia um novo nó na cena X3D baseada na sua declaração. Ele possui um construtor *fieldValue* usado para sobrescrever valores de campo padrão e fornecer novos valores de inicialização, não sendo necessários quando não há substituição de valores. O tipo do nó *ProtoInstance* é definido no campo *containerField* e deve corresponder com o nome do nó pai (sintaxe na Tabela 2.4).

Tabela 2.4. Sintaxe do nó *ProtoInstance*.**Sintaxe XML (.x3d)**

```
<ProtoInstance name='' containerField=''>
    <fieldValue name='' value='' />
</ProtoInstance>
```

2.5.2. Estereoscopia

A estereoscopia em AV permite a observação das cenas com profundidade, aumentando o grau de imersão do usuário [Burdea, 1994]. Com a possibilidade de diferentes formas de visualização, existem *browsers* X3D que oferecem suporte a esta funcionalidade. Os principais são *Bs Contact Stereo* e *Instant Player*. No *Bs Contact Stereo*, pode-se adicionar a visão estereoscópica selecionando as opções *Settings* → *Stereo*, escolhendo o tipo de estereoscopia desejado e alterando seus parâmetros em tempo real.

Para gerar um ambiente virtual estereoscópico no *browser Instant Player* é necessário especificar alguns nós e componentes desenvolvidos pelo grupo do *Instant Reality* (<http://www.instantreality.org/tutorial>). Os parâmetros que geralmente são modificados para melhor adaptar a cena são a escolha dos filtros de cor e valor da paralaxe (Figura 2.19).

2.5.3. Plataformas Móveis

O X3D surgiu inicialmente como padrão para 3D na Web. Sendo utilizado por comunidades de desenvolvedores de ambientes virtuais, obteve-se a confiança e consolidação desse padrão. Deste modo, o X3D vem consolidando-se não apenas como padrão para a Internet, mas apresentando-se como uma proposta de desenvolvimento

ágil e fácil de conteúdos tridimensionais para outras plataformas. Neste contexto, inserem-se as plataformas móveis. Com a evolução das capacidades gráficas e de *hardware*, tornou-se possível o desenvolvimento de *browsers* X3D para dispositivos móveis. Graças à extensibilidade e modularização provida pelos perfis e componentes X3D, é possível aos desenvolvedores de *browsers* programarem tais visualizadores de cena, pois eles não precisam implementar todas as funcionalidades pertencentes às especificações do X3D. Atualmente foram encontrados os seguintes *browsers* X3D para celular: *X3Dl*, *BS Contact Mobile* e *Pocket Cortona (VRML)*.

```
views [
  Viewarea {
    red TRUE
    green FALSE
    blue FALSE
    lowerLeft 0 0
    upperRight 1 1
    modifier [
      ShearedStereoViewModifier {
        zeroParallaxDistance 7
        leftEye TRUE
        rightEye FALSE
      }
    ]
  }
  Viewarea {
    red FALSE
    green TRUE
    blue TRUE
    lowerLeft 0 0
    upperRight 1 1
    modifier [
      ShearedStereoViewModifier {
        zeroParallaxDistance 7
        leftEye FALSE
        rightEye TRUE
      }
    ]
  }
]

...

Scene {
  children [
    Inline { url [ "AmbienteVirtual.x3d" ]
  ]
}
```

Figura 2.19. Configuração da visualização estereoscópica de uma cena para visualização estereoscópica com óculos para anaglifo colorido.

O X3Dl é um simples visualizador de modelos X3D para celulares. Por esta razão, possui apenas as funcionalidades mais básicas, de modo a suportar a visualização

de modelos tridimensionais estáticos. Um *browser* X3D que suporta AVs mais dinâmicos é o *Bs Contact Mobile*. Ele suporta as funcionalidades necessárias para visualização de modelos 3D, assim como componentes utilizados na construção de um ambiente virtual 3D razoavelmente complexo. Além destas, implementa outras que inovam proporcionando mais interatividade ao AV. São elas: interpoladores e sensores de forma geral, alguns métodos em *JavaScript* para lançar eventos na cena, alguns tipos de texturização, iluminação por *DirectionalLight*, *Background*, entre outros. Assim como o *Bs Contact Mobile*, o *Pocket Cortona* é um *browser* que suporta AVs com um grau elevado de interação para arquivos VRML. Para que ambos funcionem, é necessário que o dispositivo móvel (*Smart Phone* ou *Pocket PC*) possua o *Windows Mobile* instalado. Os dois *browsers* são mostrados na Figura 2.20 executando exemplos.



Figura 2.20. – Visualização de AVs executados em dispositivos móveis nos *browsers BS Contact Mobile* (esquerda) e *Pocket Cortona* (direita).

2.5.4. Conexão entre Objetos do AV e Objetos Reais

Com a evolução das ferramentas de programação e plataformas de execução, os AVs X3D podem exibir informações relacionadas a objetos reais, apresentando seu estado e permitindo modificações em tempo-real [Vianna, 2009] (Figura 2.21). A atualização dos comportamentos ocorre nos dois sentidos: Ambiente Real → Ambiente Virtual e Ambiente Virtual → Ambiente Real. Qualquer mudança ocorrida em qualquer dos ambientes, deve acionar o envio de pacotes contendo as informações da corrente alteração. Para intermediar essa comunicação pode ser utilizado a SAI, que permite

acesso dos dados do ambiente virtual por uma linguagem de programação. Através desta, é possível a comunicação inteligente dos ambientes reais e virtuais através da rede.

2.5.5. Ambientes Colaborativos

A SAI proporciona ao programador um meio de controlar o comportamento dos objetos do ambiente virtual. Deste modo, é possível desenvolver um banco de dados designado para conter informações a respeito do estado destes objetos e integrá-lo aos objetos do AV por meio da SAI. Controlando tais AVs por linguagens de programação, pode-se criar ambientes virtuais tridimensionais colaborativos, ou seja, ambientes que são acessados por mais de um usuário, de maneira que as alterações produzidas pelos mesmos sejam atualizadas nos correspondentes AVs [Sales, 2008].

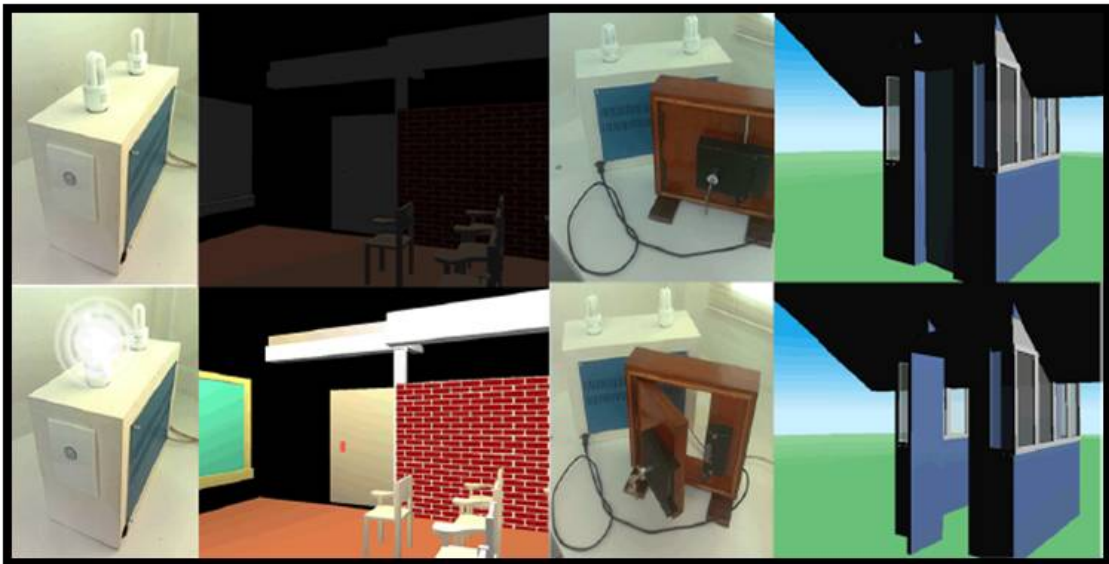


Figura 2.21. Alteração do AV pelo usuário. A alteração é repassada ao ambiente real, ou do ambiente real para o virtual. Fonte: Vianna (2009).

Referências

- [Boss, 2009] Boss, B., W3C (World Wide Web Consortium), 1999. “XML in 10 Points”. Online: <http://www.w3.org/XML/1999/XML-in-10-points>. Acesso em novembro/2009.
- [Brutzman, 2007] Brutzman, D., Daly, L., (2007) “X3D: 3D Graphics for Web Authors”. Morgan Kaufmann Publishers.

- [Burdea, 1994] Burdea, G., Coiffet, P., (1994) “Virtual Reality Technology”. John Wiley e Sons, Nova Iorque.
- [Garcia, 2009] Garcia, L. M. L. S., Dias, D. C., Brega., J. R. F., (2009) “Descrição Semântica de Componentes em Ambiente Virtual 3D”, em 6º Workshop de Realidade Virtual e Aumentada (WRVA). Santos – SP, Brasil. CD-ROM.
- [H3D, 2010] H3D.org, Open Source Haptics. Acesso em março/2010. Online: <http://www.h3dapi.org/>.
- [Instant Labs, 2009] Instant Labs, developing tomorrow technologies with today standards. “X3D encoding converter”. Online: http://instantreality.de/tools/x3d_encoding_converter/. Acesso em: dezembro/2009.
- [Kumar, 2008] Kumar, S., Chhugani, J., Kim, C., Kim, D., Nguyen, A., Dubey, P., Bienia, C., Kim, Y. (2008) “Second Life and the New Generation of Virtual Worlds”, em IEEE Computer, Volume 41, Number 8.
- [Machado, 2006] Machado, L. S., Costa, T. K. L., Moraes, R. M. (2006) “A 3D Intelligent Campus to Support Distance Learning”, em Proc. of Information Technology based Higher Education and Training (ITHET'2006), Sydney.
- [Raposo, 2007] Raposo, A. B. (2007) “Grafos de Cena”. Online em: http://www.tecgraf.puc-rio.br/~abraposo/INF1366/2007/02_GrafoDeCena.pdf.
- [Sales, 2008] Sales, B. R. A., Machado, L. S. (2008) “Um Ambiente Virtual Colaborativo e Telecomandável Baseado em X3D”, em Proc. X Symposium on Virtual and Augmented Reality (SVR). João Pessoa, Brazil. 2008 p. 327-330.
- [Soares, 2004] Soares, L. P., Zuffo, M. K. (2004) “JINX: An X3D Browser for VR Immersive Simulation”. In: Proceedings of Web3D Symposium, Monterey.
- [Souza, 2010] Souza, D, Machado, L., Tavares, T. (2010) “*Extending Brazilian DTV Middleware to Incorporate 3D Technologies*”, em Proc. XII Symposium on Virtual and Augmented Reality (SVR2010). Natal, Brazil.
- [Vianna, 2009] Vianna, A. S. G., Machado, L. S. (2009) “Controle e Gerenciamento de Ambientes Reais Educacionais Através de Ambientes Virtuais”, em Proc. International Conference on Engineering and Computer Education (ICECE2009), Buenos Aires, Argentina. CD-ROM.
- [Web3D-a, 2009] Web3D, “X3D Schematron Validation and Quality Assurance”. Online: <http://www.web3d.org/x3d/tools/schematron/X3dSchematron.html>. Acesso em outubro/2009.
- [Web3D-b, 2009] Web3D, “Basic, Humanoid Animation, Nancy Diving”. Online: http://www.web3d.org/x3d/content/examples/Basic/HumanoidAnimation/_pages/page21.html. Acesso em outubro/2009.

- [Web3D-c, 2009] Web3D Consortium – Public X3D Wiki, “Player support for X3D components – Web3D.org”. Online: http://www.web3d.org/x3d/wiki/index.php/Player_support_for_X3D_components. Acesso em outubro/2009.
- [Web3D-d, 2009] Web3D Consortium, “X3D – FAQ (Frequently Asked Questions)”. Online: <http://www.lsi.usp.br/~lsoares/x3d/faq.html>. Acesso em: agosto/2009.
- [Web3D-e, 2009] Web3D, “X3D Public Specifications”. Online: www.web3d.org/x3d/specifications. Acesso em agosto/2009.
- [Web3D-f, 2009] Web3D, “Extensible 3D (X3D). Part 1: Architecture and base components. ISO/IEC 19775-1:2008”. Online: <http://www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/index.html>. Acesso em novembro/2009.
- [Web3D-g, 2009] Web3D, “Extensible 3D (X3D). Part 2: Scene access interface (SAI). ISO/IEC 19775-2.2:2009”. Online: <http://www.web3d.org/x3d/specifications/ISO-IEC-FDIS-19775-2.2-X3D-SceneAccessInterface/index.html>. Acesso em novembro/2009.
- [Xj3D, 2009] Xj3D Project. “*Xj3D VRML/X3D Code API*”. Online: <http://circee.univ-fcomte.fr/Docs/Java/Xj3D.1.0-Doc/index.html>. Acesso em dezembro/2009.