UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

EDITOR DE MALHAS FERROVIÁRIAS: EMF

LUIZ RICARDO DIAS

BLUMENAU
2008

2008/1-26

LUIZ RICARDO DIAS

EDITOR DE MALHAS FERROVIÁRIAS: EMF

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. José Roque Voltolini da Silva - Orientador

BLUMENAU
2008

2008/1-26

EDITOR DE MALHAS FERROVIÁRIAS: EMF

Por

LUIZ RICARDO DIAS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. José Roque Voltolini da Silva – Orientador, FURB
Membro:	Prof. Paulo Cesar Rodacki Gomes – FURB
Membro:	Prof. Dalton Solano Dos Reis – FURB

Blumenau, 09 de julho de 2008

RESUMO

O Editor de Malhas Ferroviárias (EMF) é uma ferramenta que permite a simulação da construção de uma malha ferroviária, permitindo a criação, modificação e exclusão de trechos de uma ferrovia. Ainda, objetiva dar suporte para softwares de controle de uma malha ferroviária real, como as implementadas por Schubert (2003) e Sardo (2007), simulando suas configurações numa forma visual. Para o desenvolvimento foi utilizada a linguagem Java juntamente com a biblioteca gráfica JoGL e para a confecção dos componentes da malha foi utilizado o software Autodesk 3Ds Max (versão trial), armazenando os dados em arquivos vetoriais `wavefront obj`.

Palavras-chave: Editor gráfico. Malha ferroviária. Arquivos vetoriais `wavefront obj`. JoGL.

ABSTRACT

The Editor of Net Railway (in portuguese Editor de Malhas Ferroviárias - EMF) is a tool that allows the simulation of building a net railway, allowing the creation, modification and deletion of portions of a railroad. Still, aims to provide support for software control of a real net railway, as implemented by Schubert (2003) and Sardo (2007), simulating their settings in a visual form. In the development the Java language was used along with the JoGL graphic library and to make up components of the net was used the software Autodesk 3Ds Max (trial version), storing data in `wavefront obj` vector files.

Key-words: Graphic editor. Net Railway. `Wavefront obj` vector files. JoGL.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

EA – *Entreprise Architect*

EGMR – Editor Gráfico de Malhas Rodoviárias

EMF – Editor de Malhas Ferroviárias

FPS – *Frames* Por Segundo

GPS – *Global Position System*

GRR – Grade Regular Retangular

HTML – *HyperText Markup Language*

JoGL – *Java bindings for OpenGL*

RAM – *Random Access Memory*

RF – Requisito Funcional

RGB – *Red, Green, Blue*

RNF – Requisito Não Funcional

UML – *Unified Modeling Language*

VISM – *Visual Interactive Simulation and Modeling*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo Vilaça (2007), para o crescimento do país é necessário um eficiente escoamento de produtos industriais, minerais e agropecuários para o mercado interno e para a exportação. Isso faria com que o custo dos transportes reduzisse consideravelmente e assim um preço mais competitivo chegaria ao consumidor final. Um meio de transporte considerado eficiente para o escoamento de produtos é o transporte ferroviário. Esse tipo de transporte tem recebido um forte investimento nos últimos dez anos, quando houve a privatização do setor. Este investimento, superior a onze bilhões de reais, tem tornado o transporte ferroviário mais atrativo. Acompanhando este crescimento, serão necessários investimentos para automatizar o controle de malhas ferroviárias e assim tornar o transporte mais eficiente.

Em Schubert (2003) e Sardo (2007) é descrita uma implementação para controle de uma malha ferroviária. Para simular o controle no mundo real foi utilizada uma maquete. Schubert (2003) implementou ainda um software para visualização virtual da malha ferroviária. Porém, a malha não podia ser alterada e era simples, contendo apenas um tipo de cruzamento e um tipo de curva. Sardo (2007) não implementou a visualização virtual da malha, focando apenas no controle do hardware (trens e cruzamentos).

Visto o acima, verificou-se a necessidade da construção de um editor para construção de uma malha ferroviária virtual, objetivando simular uma malha real. Para isso foram utilizados elementos gráficos gerados por um software específico para desenho, o Autodesk 3Ds Max versão trial (AUTODESK, 2007). Esse por sua vez permite criar modelos para serem usados na confecção da malha ferroviária. Ainda, salienta-se que em Bertholdi (2004), Froeschlin (2006) e Perondi (2007) são apresentadas a implementação de um editor de uma malha rodoviária, os quais serviram como uma referência inicial para a construção do Editor de Malhas Ferroviárias (EMF).

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um editor de malhas ferroviárias.

Os objetivos específicos do trabalho são:

a) manipular arquivos com dados vetoriais, gerados por ferramentas externas como o

3Ds Max e o Blender;

b) criar uma área de edição interativa 3D, onde serão incluídos os componentes das ferrovias, componentes esses previamente criados;
c) possibilitar a realização de rotação e translação para os componentes da ferrovia e para a câmera;
d) permitir realizar aproximação e afastamento da câmera (*zoom in*, *zoom out*).

1.2 ESTRUTURA DO TRABALHO

Este trabalho é dividido em 5 partes. Neste capítulo encontram-se a introdução e os objetivos do trabalho. No segundo capítulo encontram-se a fundamentação teórica, que reúnem informações sobre interfaces e editores gráficos (enfatizando o editor tridimensional Autodesk 3Ds Max), a biblioteca Java bindings for OpenGL (JoGL), estrutura de arquivos `wavefront obj`, estrutura e controle de malhas ferroviárias e informações de trabalhos correlatos. No terceiro capítulo é demonstrada a construção dos modelos vetoriais no 3Ds Max (versão trial) usados no EMF. No quarto capítulo encontram-se informações sobre o desenvolvimento da ferramenta, contendo os requisitos principais, a especificação, a implementação, a operacionalidade da implementação e resultados e discussões que aconteceram no decorrer do desenvolvimento do trabalho. No quinto capítulo encontram-se as conclusões, limitações e sugestões para extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados informações sobre interfaces e editores gráficos, JoGL, estrutura de arquivos `wavefront obj`, estrutura e controle de malhas ferroviárias e trabalhos correlatos, citando suas principais características.

2.1 INTERFACES GRÁFICAS

O uso das interfaces gráficas expandiu com a evolução dos microcomputadores. Os sistemas eram compostos basicamente de textos e para utilizá-los o usuário tinha que decorar comandos e sintaxes, o que dificultava o aprendizado. A medida que os microcomputadores evoluíram surgiu a necessidade de criar algo que trouxesse praticidade e agilidade ao usuário, surgindo assim as interfaces gráficas (GUIA DO USUÁRIO DO CONECTIVA LINUX, 2008).

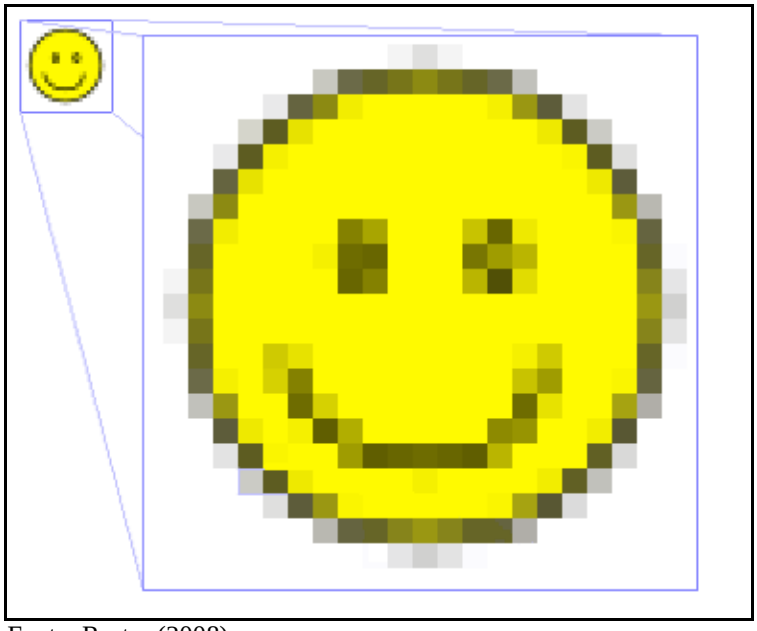
Segundo Foton (2008), a junção do elemento artístico e do lógico formam as interfaces gráficas. O elemento artístico é responsável por conferir visibilidade e conforto ao usuário, reunindo recursos em um ambiente atrativo e fazendo com que a informação possa ser acessada de uma forma simples e rápida, utilizando atalhos, *links*, botões e diversos outros recursos para tornar isso possível. Já o elemento lógico é o que confere familiaridade ao usuário, fazendo que a informação seja apresentada como uma seqüência de acontecimentos, assim como acontece no mundo real.

Borges (2008) diz que os principais elementos das interfaces gráficas são: janelas, menus, ícones, figuras geométricas, caixas de diálogo, caixas de mensagem, barra de ferramentas, controles, som e vídeo. Cada um destes elementos possui vários sub-elementos e possuem diversas funcionalidades. O controle, por exemplo, pode ser aplicado utilizando botões, barras de rolagem, caixas de edição de texto, entre outros. Os botões por sua vez podem ser de diversos tipos. Botões de comando que ao serem clicados executam determinada tarefa. Botões de rádio utilizados quando se tem várias opções, mas disponibilizando a escolha de uma só. Botões de seleção múltipla, utilizados quando pode-se escolher mais de uma opção. Enfim, são inúmeros elementos que são utilizados atualmente.

2.2 EDITORES GRÁFICOS

Segundo Brito (2007, p. 13), editores gráficos são programas que têm como objetivo facilitar a criação e alteração de imagens digitais. São classificados em três categorias: *raster*, vetoriais e tridimensionais.

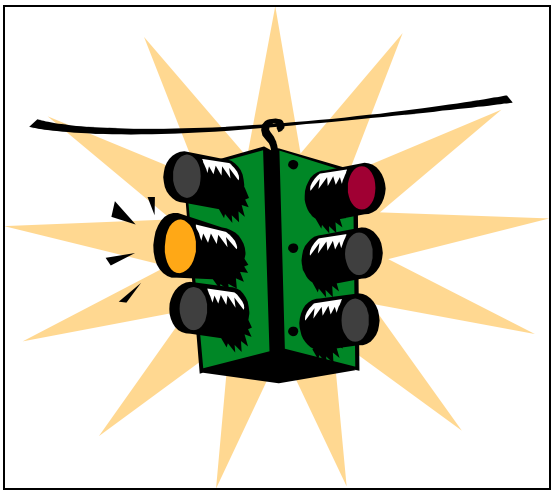
Editores *raster* são aqueles que geram imagens digitais, editam e retocam fotografias, sendo que as imagens geradas por esses são formadas por *pixels*. Um *pixel* é a menor parte de uma imagem e armazena sua informação de cor. Para gravar esta informação de cor é utilizado o padrão RGB (do inglês *Red, Green, Blue*), que é representado por três números inteiros, um para cada cor. A desvantagem deste tipo de imagem é que quando aumentada é feita a duplicação de alguns *pixels*, perdendo qualidade. O tipo de arquivo chamado `Bitmap` é um formato de imagens *raster* (Figura 1).



Fonte: Raster (2008).

Figura 1 – Exemplo de imagem raster

Editores vetoriais são aqueles nos quais é possível modificar livremente o desenho criado sem que tenha perda de qualidade. As informações são guardadas em modelos matemáticos. Em trechos de desenhos sólidos, de apenas uma cor, é feita uma repetição da cor, não sendo necessário armazenar informações da cor de cada ponto da imagem. Outra característica é o pequeno tamanho ao armazenar suas informações, visto que são armazenados apenas vértices, arestas e cores. Um exemplo de uma imagem vetorial pode ser vista na Figura 2. A imagem da Figura 2 foi obtida a partir do conjunto de imagens chamado Clip-art do editor Microsoft Word 2000 (MICROSOFT, 2008).



Fonte: Microsoft (2008).

Figura 2 – Exemplo de imagem vetorial

Editores tridimensionais são aqueles que manipulam modelos vetoriais em três dimensões. Como exemplo cita-se os sólidos primitivos (cubos, prismas, etc). Assim como nas imagens vetoriais, não há perda na qualidade ao redimensionar a imagem, visto que o desenho é formado a partir de modelos geométricos. Um exemplo de imagem tridimensional é mostrado na Figura 3. A Figura 3 apresenta a criação de um desenho no software 3Ds Max (AUTODESK, 2007) (versão trial), exibindo-o em quatro posições padrões: de cima, de lado, de frente e outra na diagonal de cima para baixo. Apenas nesta última é mostrado o desenho com sua textura. Nas demais visualizações é mostrado apenas o desenho aramado, com os vértices e arestas que o compõe. Cada uma destas visualizações pode ser configurada pelo usuário.

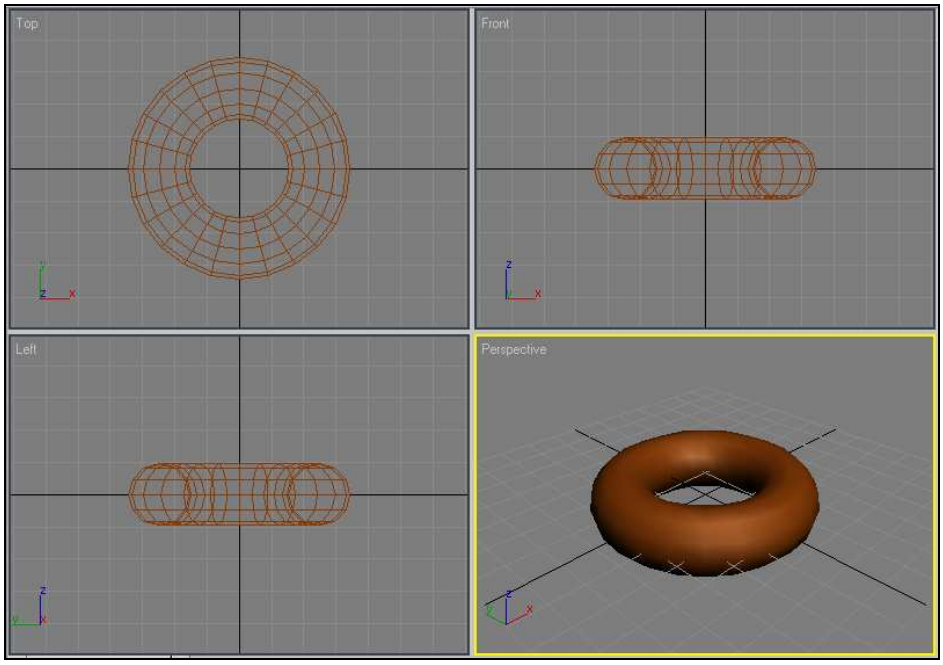


Figura 3 – Exemplo de imagem tridimensional

2.2.1 Editor tridimensional Autodesk 3Ds Max

A criação de modelos vetoriais é feita por softwares bastante comuns na área de animações e jogos. Estas ferramentas oferecem diversas funções para desenho, animação e texturização. O 3Ds Max destaca-se dentre estas ferramentas, sendo considerado um software robusto, permitindo a geração de renderizações de alta qualidade, com luzes, sombras e transparências, possibilitando assim a criação de imagens foto realísticas. Ainda, permite a confecção de *plugins* utilizando uma ferramenta interna chamada MAXScript, que possui comandos similares a uma linguagem de programação e permite criar movimentos para cada objeto da animação criada. Possui mais de dez anos de mercado (AUTODESK, 2007).

A interface do software é mostrada Figura 4. A interface possui na parte superior da tela dois menus: um menu texto que possui todos os comandos da ferramenta e outro que possui ícones de atalhos, como desfazer e refazer ações, entrar em modo de seleção, escala e rotação de objetos, renderização da cena, entre outros. À direita encontra-se o menu de edição de objetos, disponibilizando além da edição, várias opções, como definição de luzes, posicionamento de câmeras, criação de objetos primitivos como cubos, pirâmides, entre outros. O menu inferior possui a barra de *status* e atalhos para os comandos de movimentação da câmera e animação.

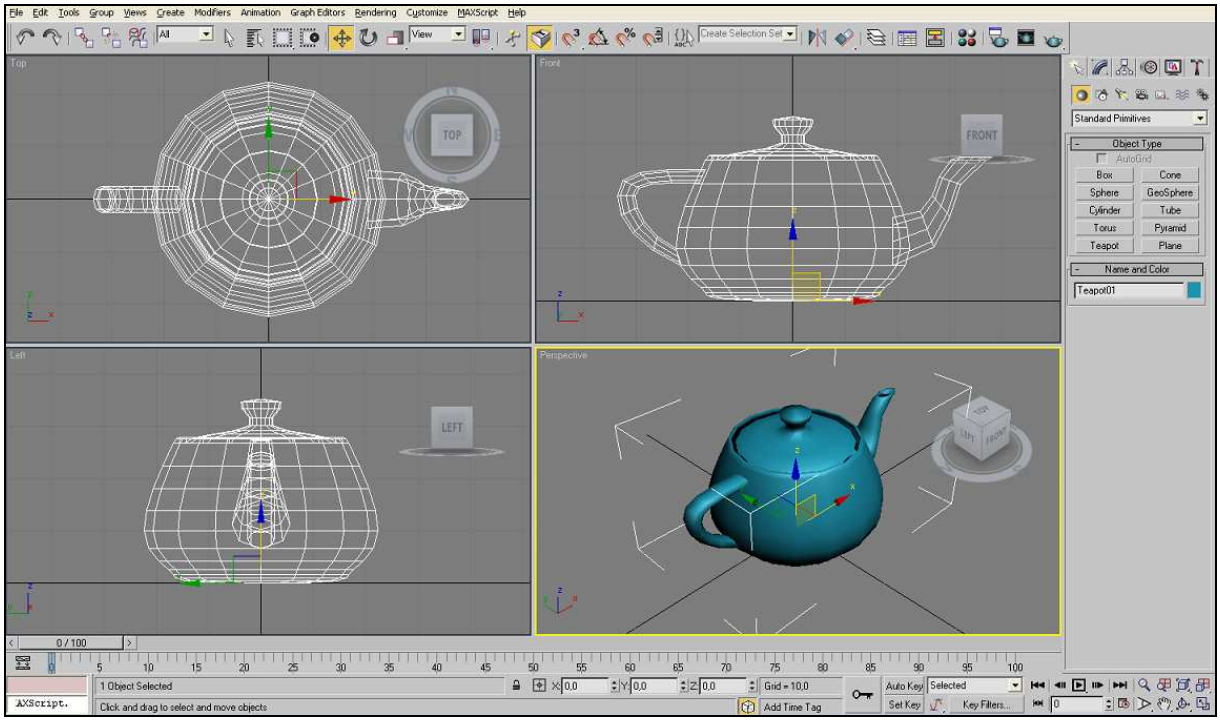


Figura 4 – Interface do software 3Ds Max

2.3 JOGL

Java.net (2007) descreve JoGL como uma biblioteca que permite a ligação da biblioteca OpenGL com a linguagem Java e seus componentes. Foi desenvolvida pelo grupo de tecnologia em jogos da Sun Microsystems e caracteriza-se por funcionar através de eventos. Para exibir o desenho na tela, o JoGL possui a interface `GLEventListener` que possui os seguinte eventos:

a) `display`: executado sempre que há a necessidade de atualizar as informações contidas na tela;
b) `displayChanged`: acionado quando há mudanças na tela;
c) `init`: executado no inicio do programa para inicializar as configurações do ambiente;
d) `reshape`: executado quando houver alteração na tela, como maximização, minimização, restauração ou redimensionamento.

Apesar da linguagem Java ser multiplataforma, a biblioteca JoGL é disponibilizada para cada sistema operacional separadamente, bastando incluí-la no projeto para que suas funcionalidades estejam disponíveis para uso.

2.4 ESTRUTURA DE ARQUIVOS `WAVEFRONT OBJ`

O modelo padrão `wavefront obj` foi especificado pela Wavefront Tecnologies e tem como objetivo armazenar informações vetoriais (REDDY, 2008). No quadro 1 é mostrado a estrutura de um cubo no formato `obj`, criado no software 3Ds Max (versão trial) e salvo no formato `obj`.

```
# colorCube.obj

mtllib colorCube.mtl

v 0.000000 1.000000 1.000000
v 0.000000 0.000000 1.000000
v 1.000000 0.000000 1.000000
v 1.000000 1.000000 1.000000
v 0.000000 1.000000 0.000000
v 0.000000 0.000000 0.000000
v 1.000000 0.000000 0.000000
v 1.000000 1.000000 0.000000
# 8 vertices

g cube_1
usemtl red
f 1 2 3 4
f 8 7 6 5
f 2 6 7 3

g cube_2
usemtl green
f 5 1 4 8
f 5 6 2 1
f 4 3 7 8
```

Quadro 1 – Estrutura de um arquivo `obj`

A estrutura apresentada Quadro 1 é composto pelas seguintes informações:

a) `#`: indica uma linha de comentários;
b) `mtllib`: fornece o nome do arquivo que armazena os materiais dos objetos (arquivo `.mtl`);
c) `v`: indica as coordenadas dos vértices;
d) `g`: indica o inicio da descrição de um grupo (a qual forma um objeto) e o seu nome;
e) `usemtl`: indica a configuração da textura utilizada pelas faces descritas abaixo do `usemtl` (a configuração das texturas `red` e `green` utilizadas no objeto está armazenada no arquivo `colorCube.mtl`);
f) `f`: indica a face de um elemento.

Além das informações apresentadas, o formato `obj` possui outras, como por exemplo, `vt` que informa coordenadas para as texturas.

O Quadro 2 apresenta uma estrutura do arquivo de materiais (texturas) utilizados no exemplo do Quadro 1. Este arquivo é armazenado separadamente do arquivo `obj` em um arquivo `mtl`.

```
#colorCube.mtl

newmtl red
Kd 1 0 0
Ks 0 0 0
Ka 0 0 0
illum 1

newmtl green
Kd 0 1 0
Ks 0 0 0
Ka 0 0 0
illum 1
```

Quadro 2 – Estrutura de um arquivo `mtl` de um cubo

O modelo mostrado no Quadro 2 é composto pelas seguintes informações:

a) `newmtl`: indica um nome para o novo material para ser usado como referência nos arquivos `obj`;

b) `kd`: indica a cor de difusão (propagação) do material;

c) `ks`: indica a cor especular (reflexão) do material;

d) `ka`: indica a cor de ambiente do material;

e) `illum`: indica o modo de iluminação do material.

O padrão `mtl` possui outras informações, porém são utilizadas para objetos mais complexos. Como exemplo cita-se a utilização texturas que utilizam imagens.

Na Figura 5 é mostrado um cubo, cuja estrutura é representada no Quadro 1 e no Quadro 2. Esta imagem foi gerada a partir do software 3Ds Max (versão trial).

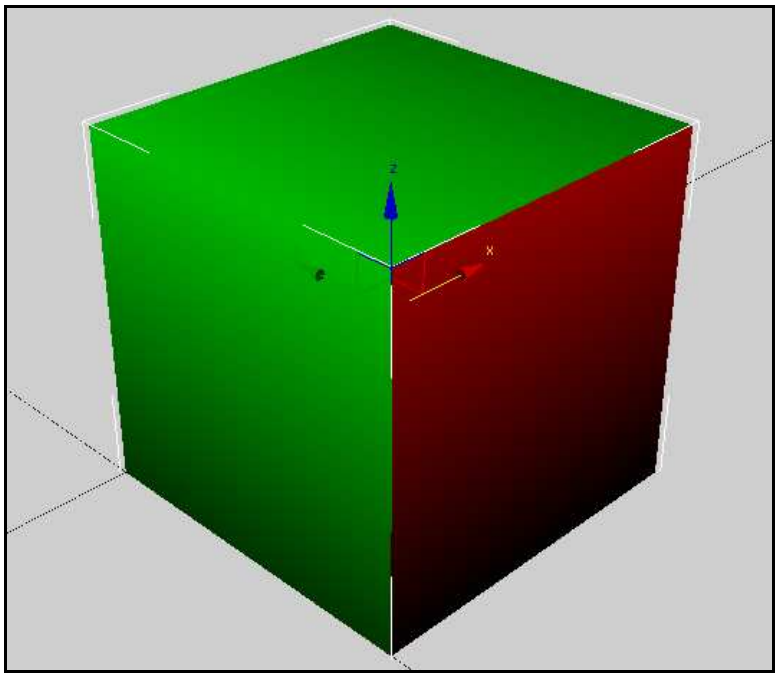


Figura 5 – Visualização de um cubo no formato `obj`

2.5 ESTRUTURA E CONTROLE DE MALHAS FERROVIÁRIAS

O transporte ferroviário é predominante em regiões altamente industrializadas, visto que é o meio de transporte terrestre com maior capacidade de carga e de passageiros. As ferrovias por sua vez são compostas por dois trilhos paralelos, chamados de *carris*, destinados esses a encaixe das rodas dos trens, dispostos perpendicularmente sobre travessas, os dormentes, podendo ser de madeira, concreto ou aço com o objetivo de manter uma distância específica constante. Esta distância constante é chamada de bitola (JUHNKE, 1968, p. 118).

Na maioria dos casos, as ferrovias estendem-se por várias cidades, podendo existir durante seu percurso desvios para mudança de trilhos, cruzamentos com ruas ou mesmo com outras ferrovias. Para que os desvios funcionem existem os aparelhos de manobra que são utilizados para determinar qual o caminho que o trem irá passar e geralmente são operados de forma manual. Para os cruzamentos é necessário tomar medidas de segurança como a utilização de sistemas de semáforos, onde somente um caminho estará liberado de cada vez (JUHNKE, 1968, p. 198).

Os primeiros semáforos utilizados nas ferrovias apresentavam vários problemas. Eram operados de forma manual e sinalizados por uma espécie de braço mecânico que indicava três estados: parar (braço na posição horizontal), atenção (braço na diagonal para baixo) e seguir em frente (braço na vertical para baixo). Como se tratava de um equipamento mecânico, havia o perigo do semáforo estar quebrado e o braço direcionado para baixo, sinalizando erroneamente seguir em frente. A noite o problema era maior, onde o estado seguir em frente era representado por uma luz amarela, podendo ser facilmente confundida pela luz de outro trem (JANCZURA, 1998, p. 7).

Segundo Janczura (1998, p. 8), as decisões tomadas nos semáforos eram baseadas em estimativas de tempo. Para evitar a colisão entre dois trens, as saídas de trens das estações eram programadas, sendo os trens liberados a cada intervalo de tempo pré-determinado.

Com a evolução da tecnologia, as estimativas de tempo deixaram de ser utilizadas e as operações dos semáforos foram automatizadas. Primeiramente utilizou-se o telégrafo, onde os trens eram liberados somente após confirmação de que ele chegou a um determinado local. Depois disso, além do telégrafo, começou-se a utilizar computadores de bordo e até o *Global Position System* (GPS) para acompanhar a posição dos trens e assim tomar as devidas decisões (AMÉRICA LATINA LOGÍSTICA, 2005).

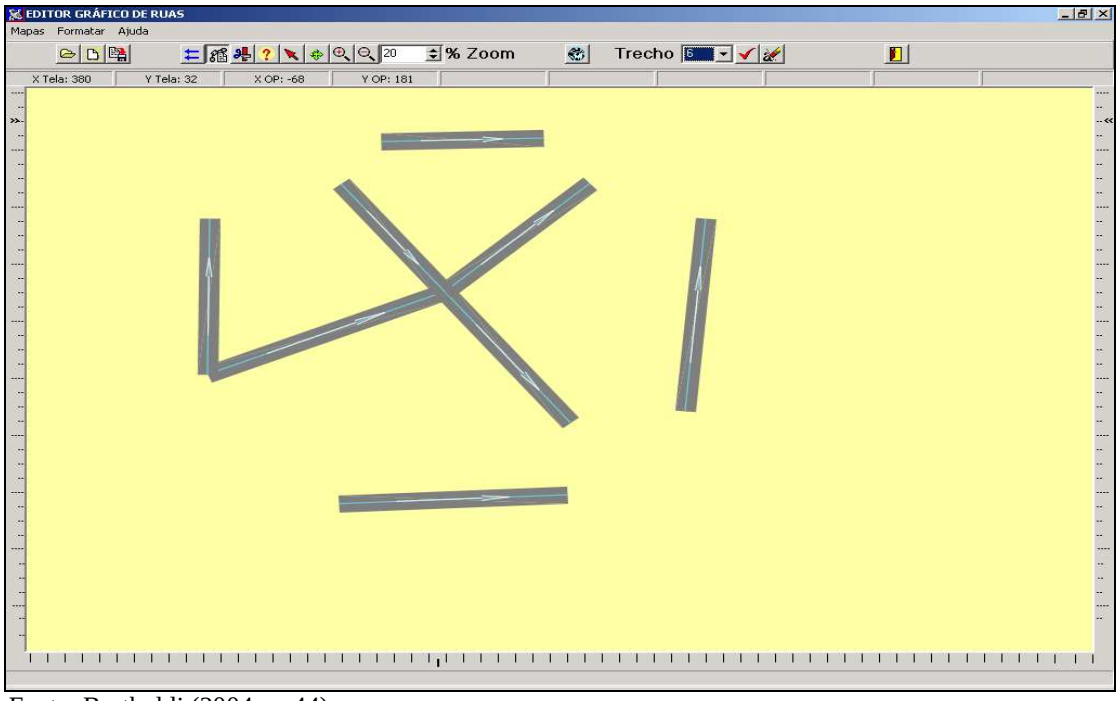
2.6 TRABALHOS CORRELATOS

Trabalhos feitos anteriormente possuem características semelhantes ao trabalho em questão. Dentre eles, cita-se os de Bertholdi (2004), Froeschlin (2006) e Perondi (2007), os quais tratam de editores de rodovias. Os trabalhos de Schubert (2003) e Sardo (2007) tratam sobre ferrovias e estão voltados ao controle da malha ferroviária e não para a sua visualização. Ainda cita-se o trabalho de Almeida (1998), o qual trata de um sistema de monitoração de ferrovias.

2.6.1 Editor Gráfico de Malhas Rodoviárias – EGMR versão 1.0

O Editor Gráfico de Malhas Rodoviárias (EGMR) - versão 1.0, apresentado em Bertholdi (2004), mostra a implementação de um editor gráfico de malhas rodoviárias urbanas. A malha é desenhada através de retas, observando-se os sentidos e intersecções. Possui ainda funções para unir e separar trechos da malha rodoviária e utiliza um mecanismo de interpretação de arquivos `txt`, onde informações sobre a malha rodoviária são persistidas. Para a implementação utilizou-se do ambiente Delphi 7 e da biblioteca OpenGL.

Uma visão da interface deste editor pode ser vista na Figura 6.



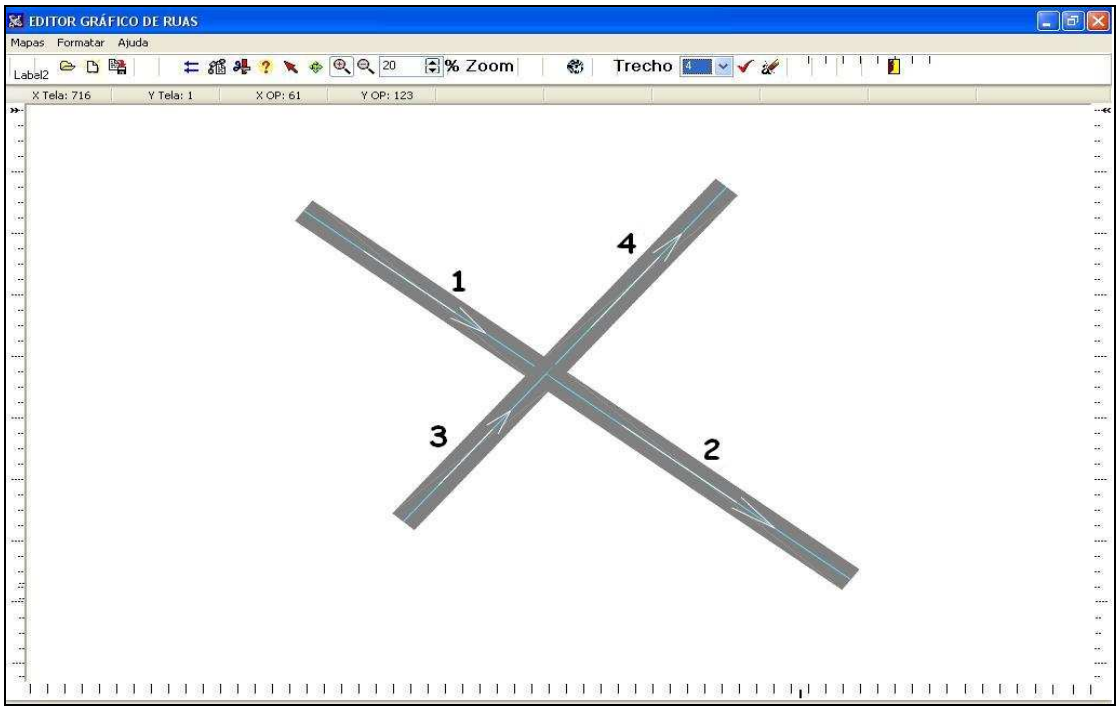
Fonte: Bertholdi (2004, p. 44).

Figura 6 – Interface do EGMR versão 1.0

2.6.2 EGMR versão 2.0

O EGMR versão 2.0, apresentado em Froeschlin (2006), é uma extensão do trabalho de Bertholdi (2004), onde mudanças foram realizadas. Entre elas cita-se a reespecificação do projeto, utilizando a técnica de orientação a objetos. A partir desta especificação, o mesmo foi reimplementado. Também foram adicionadas novas funcionalidades, como a inserção de semáforos e informações sobre as ruas criadas.

Uma visão da nova interface com o usuário pode ser vista na Figura 7.



Fonte: Froeschlin (2006, p. 37).

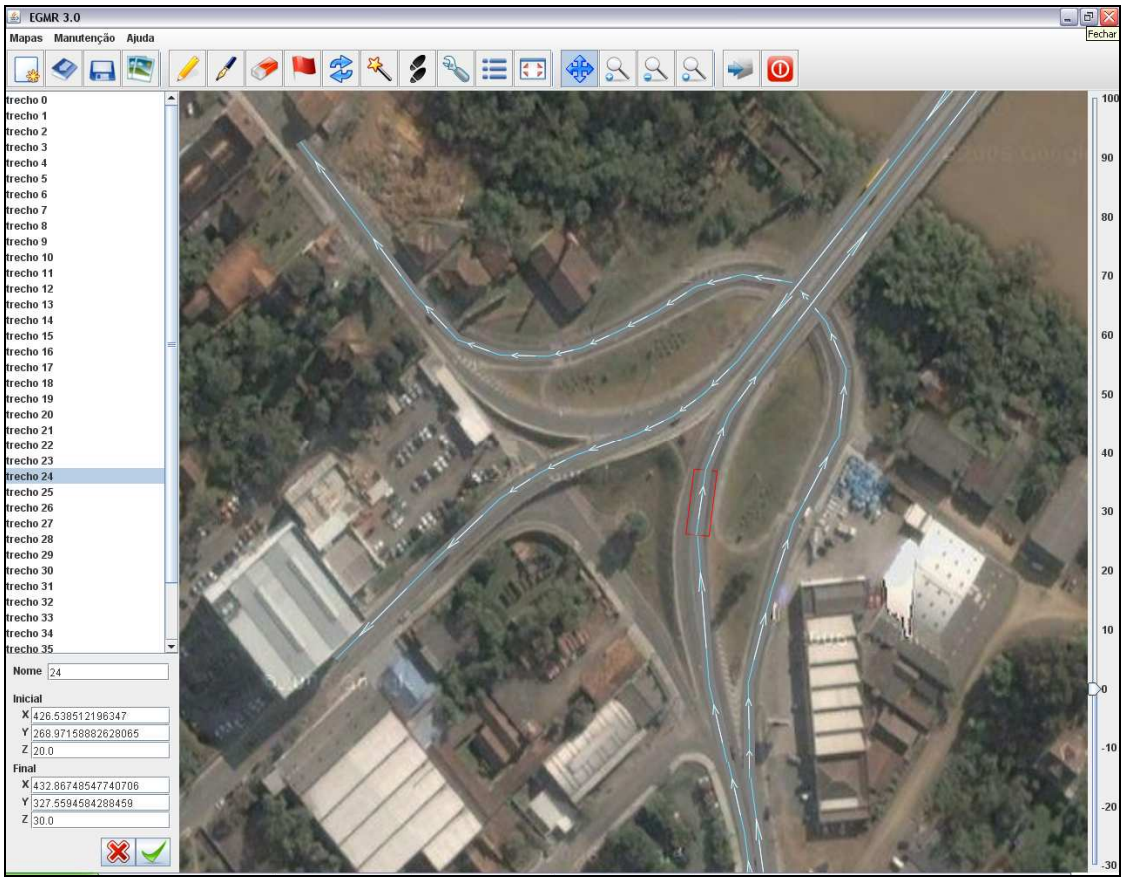
Figura 7 – Interface do EGMR versão 2.0

2.6.3 EGMR versão 3.0

O EGMR versão 3.0 apresentado em Perondi (2007) é uma extensão dos trabalhos de Bertholdi (2004) e Froeschlin (2006). Uma nova especificação foi criada, utilizando-se como base a especificação de Froeschlin (2006), incluindo mudanças e novas funcionalidades. A implementação foi realizada utilizando a linguagem Java e a biblioteca JoGL. Utilizou-se das curvas de *Bézier* para desenho das curvas das malhas rodoviárias, obtendo-se um resultado mais realista. Também foi disponibilizada a possibilidade de visualização em 3D da malha criada, a fim de auxiliar o desenho de viadutos. Ainda possibilitou a adição de imagens de

satélites como fundo para facilitar o desenho das rodovias.

Uma visão do ambiente pode ser vista na Figura 8.



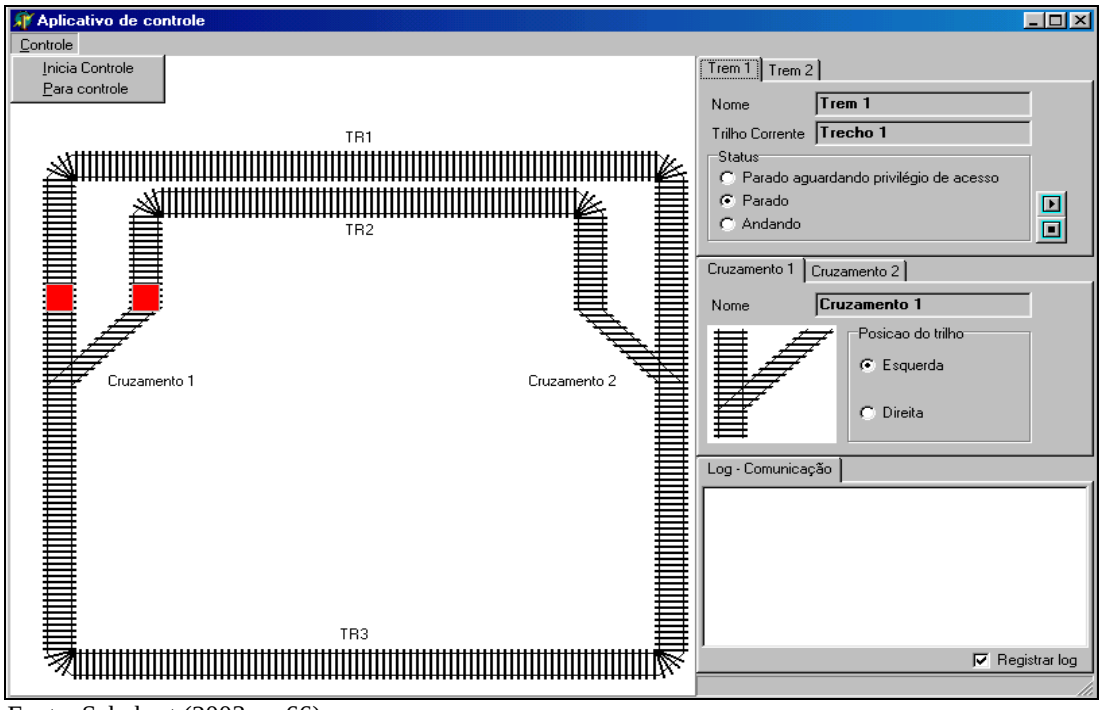
Fonte: Perondi (2007, p. 42).

Figura 8 – Interface do EGMR versão 3.0

2.6.4 Aplicativo para controle de ferrovias - versão 1.0

O “Aplicativo para Controle de Ferrovias” (SCHUBERT, 2003) tem como objetivo controlar um sistema ferroviário real. A variável tempo é considerada no aplicativo, caracterizando o uso de programação em tempo real. Ainda, uma visualização da malha ferroviária é disponibilizada num terminal, refletindo a situação real da mesma (no caso mostrando uma posição aproximada da localização dos trens). Salienta-se que a modelagem da malha (caminhos) é fixa no aplicativo. A especificação do sistema foi feita utilizando redes de Petri. Para implementação utilizou-se as linguagens Object Pascal (Delphi 7) para implementar os módulos de controle no computador central e PicBasic para a implementação do controle dos dispositivos externos (software embarcado).

Uma visão da interface do ambiente pode ser vista na Figura 9.



Fonte: Schubert (2003, p. 66).

Figura 9 – Interface utilizada no projeto de Schubert

2.6.5 Aplicativo para controle de ferrovias - versão 2.0

O “Aplicativo de Controle de Ferrovias” (SARDO, 2007) é uma extensão do projeto iniciado por Schubert (2003). O foco deste trabalho foi voltado para a parte de hardware, onde se utilizou para controle de trens e de cruzamentos o microcontrolador PIC16F628A e o componente TRF-2.4G para transmissão de dados. A visualização da malha ferroviária indicando uma localização estimada dos trens não foi implementada nesta versão. O aplicativo no computador central foi implementado na linguagem Java. Os aplicativos utilizados nos componentes externos (para controle dos cruzamentos e trens) foram implementados utilizando a linguagem Pic C.

Uma imagem da maquete utilizada por Sardo é mostrada na Figura 10.

Fonte: Sardo (2007, p. 49).

Figura 10 – Maquete utilizada por Sardo

2.6.6 Ferramenta interativa para um sistema de controle de tráfego ferroviário

A “Ferramenta Interativa para um Sistema de Controle de Trafego Ferroviário” (ALMEIDA, 1998) tem por objetivo o controle de uma malha ferroviária. Possui um editor de cenário em 2D, que possibilita a modelagem da ferrovia a ser simulada. Ainda, a ferramenta possui um ambiente em que os objetos possuem inteligência e são monitorados por um controle central. Caso ocorra falhas nos objetos, o controle central assume o controle destes objetos.

O ambiente possibilita ao usuário montar uma malha ferroviária e simular operações com trens.

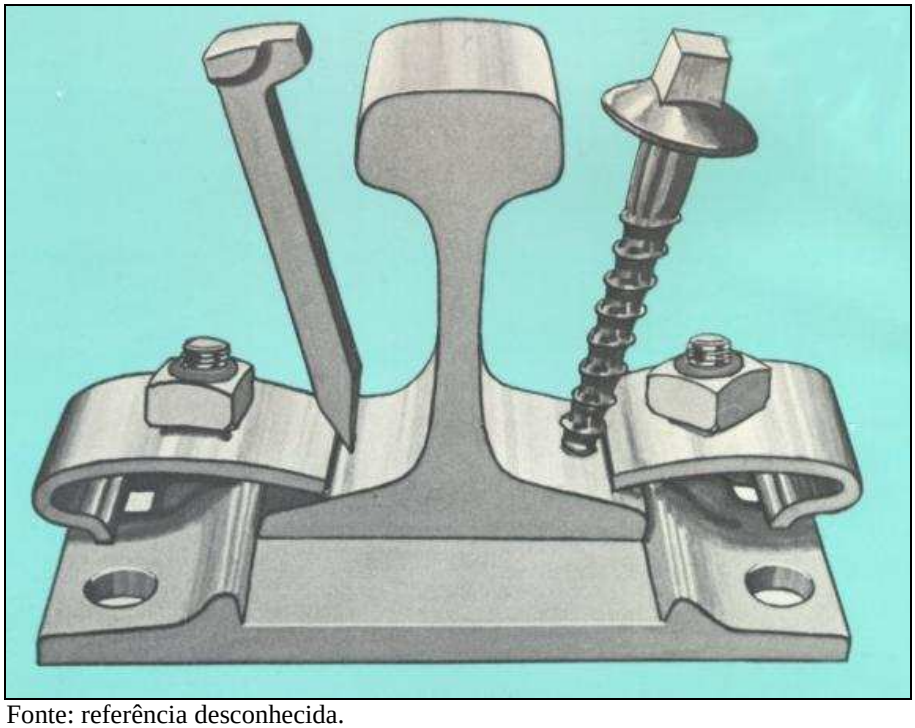
Para o desenvolvimento do ambiente utilizou-se da ferramenta *Visual Interactive Simulation and Modeling* (VISM), a qual disponibiliza recursos para representação da interação com o usuário e de simulação (RECH, 1997 apud ALMEIDA, 1998).

3 CONSTRUÇÃO DOS MODELOS VETORIAIS NO 3DS MAX

A criação dos modelos vetoriais `wavefront obj` foi feita com auxílio do software 3Ds Max (versão trial). A seguir é demonstrada a confecção dos trilhos e do trem utilizados no EMF e como se deve proceder para salvar as modelos no formato `wavefront obj` e torná-lo compatível com o EMF.

3.1 CONFECÇÃO DOS TRILHOS

Para a confecção de trilhos foram utilizadas imagens que serviram de base para a modelagem tridimensional. Na Figura 11 é mostrada a imagem utilizada como referência na modelagem do trilho e seu suporte.



Fonte: referência desconhecida.

Figura 11 – Modelo de referência para desenho de trilho

O resultado na modelagem feita a partir da Figura 11 é apresentada na Figura 12.

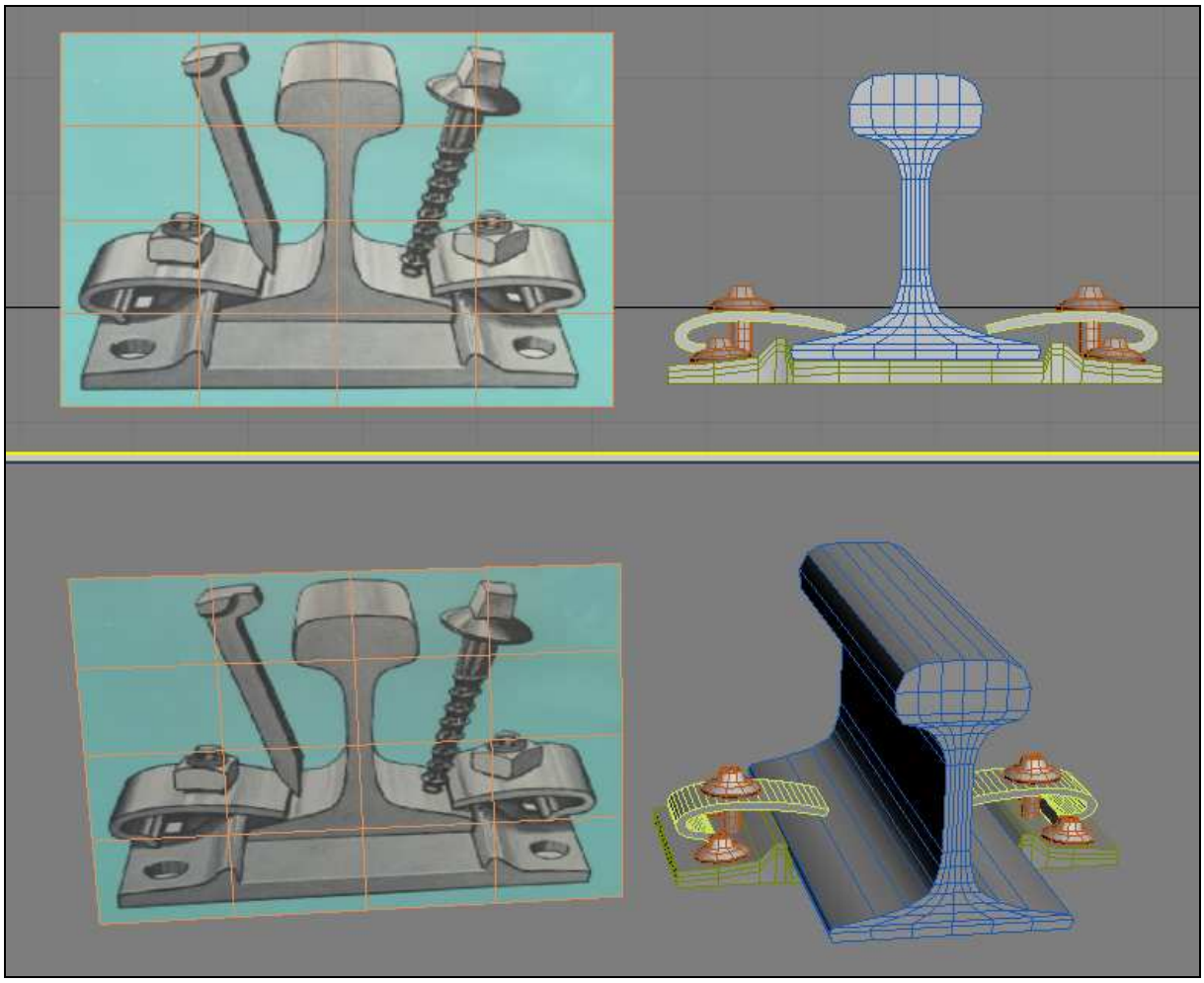


Figura 12 – Modelagem do trilho no 3Ds Max

Para modelar objetos tridimensionais foi utilizada a ferramenta `Line` do 3Ds Max. Com esta ferramenta foi desenhado o contorno de cada parte do trilho, obtendo-se um desenho bidimensional. Para transformar o desenho bidimensional em tridimensional utilizou-se a função para a modificação do objeto chamada `Extrude`. Esta modificação atribui profundidade a objetos.

Depois de modelar um trilho, foi modelada a base na qual o trilho é fixado. Após isso foram feitas cópias dos modelos e assim criados os componentes da malha ferroviária.

Um dos componentes criados é mostrado na Figura 13.

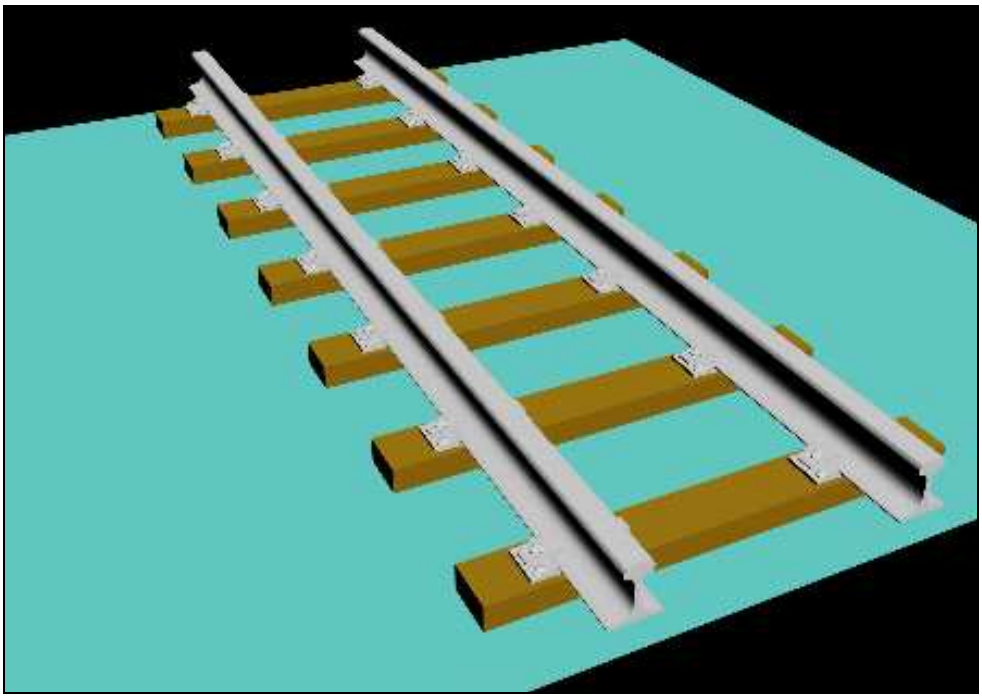
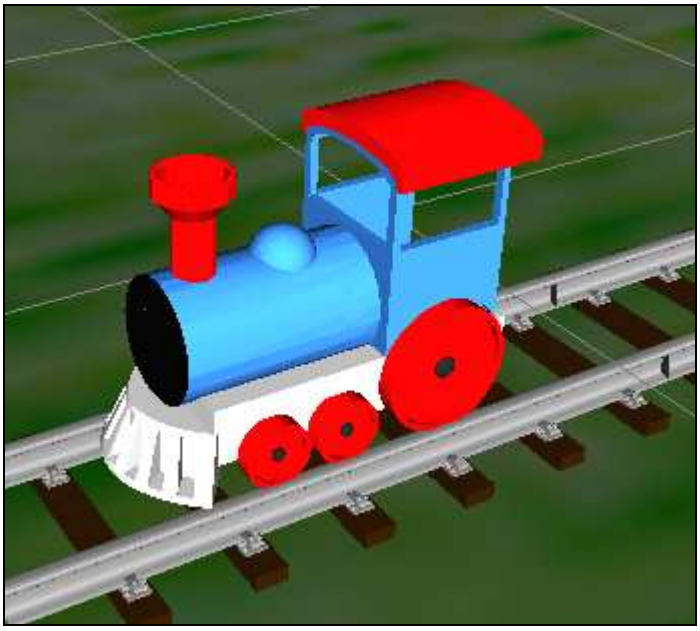


Figura 13 – Reta de trilho renderizado no 3Ds Max

3.2 CONFECÇÃO DO TREM

O trem utilizado no EMF foi obtido em Turbo Squid (2008) (Figura 14).



Fonte: Turbo Squid (2008).

Figura 14 – Trem utilizado no EMF

3.3 PROCEDIMENTOS PARA SALVAR ARQUIVOS `WAVEFRONT OBJ`

Para que o EMF possa ler e exibir arquivos `wavefront obj`, utilizou-se o conjunto de classes descritas por Davison (2007, p. 439). Os arquivos `wavefront obj` suportam diversos tipos de parâmetros. A utilização de triângulos, quadrados ou polígonos para as faces dos objetos são um exemplo. As classes descritas por Davison (2007, p. 439) possuem algumas limitações e não aceitam todos os parâmetros dos arquivos `wavefront obj`. Para gerar arquivos compatíveis com as classes de Davison é necessário configurar o ambiente do 3Ds Max, conforme é descrito a seguir.

Para salvar a modelagem feita no 3Ds Max no formato `wavefront obj` deve-se selecionar o desenho desejado e selecionar a opção "`Export Selected...`" do menu `File`. Este procedimento é mostrado na Figura 15.

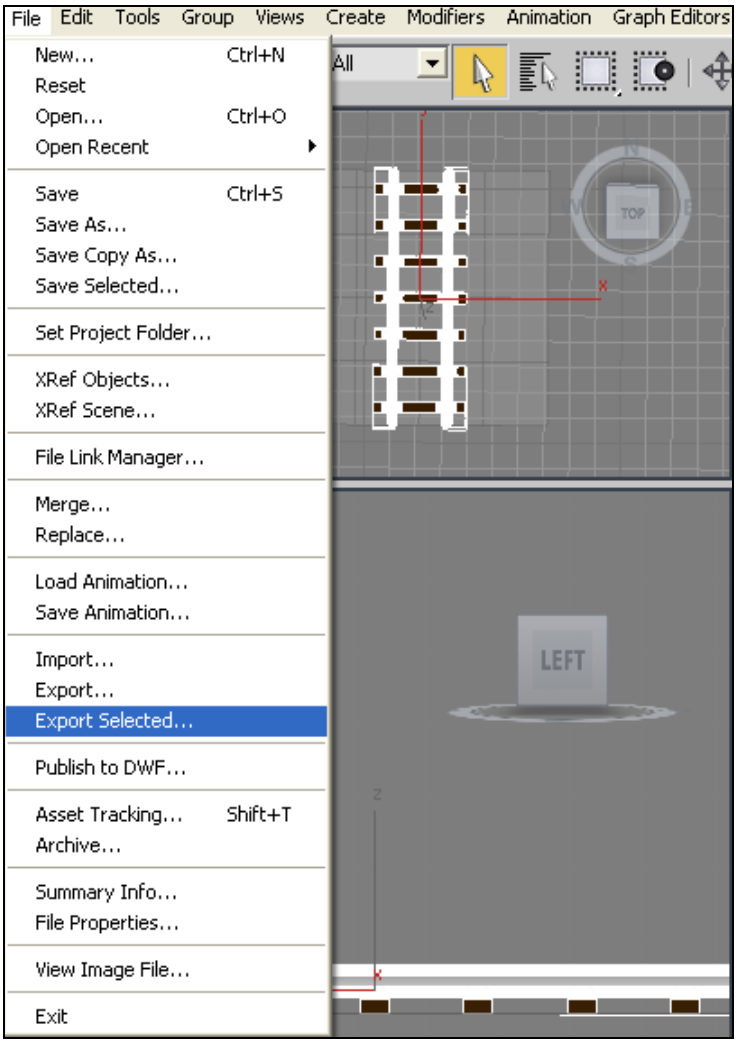


Figura 15 – Salvando modelo no formato `obj`

Em seguida deve-se escolher o tipo de arquivo a ser salvo, visto que o 3Ds Max salva

em vários formatos diferentes. Seleciona-se o tipo `gw::OBJ-Exporter (*.OBJ)` e escolhe-se o diretório a ser salvo o arquivo e seu respectivo nome, como mostrado na Figura 16.

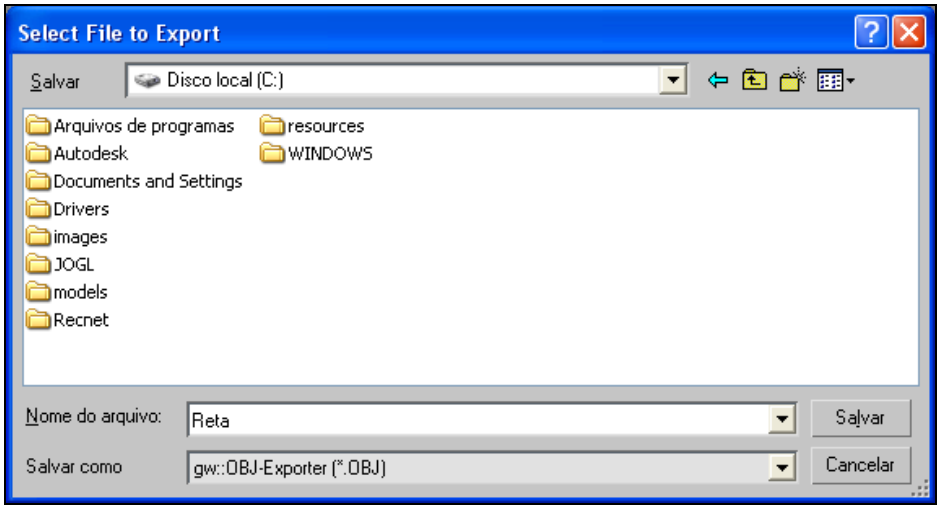


Figura 16 – Escolhendo tipo, local e nome do arquivo

Em seguida deve-se configurar os parâmetros do arquivo `wavefront obj` e pressionar o botão `Export` (Figura 17).

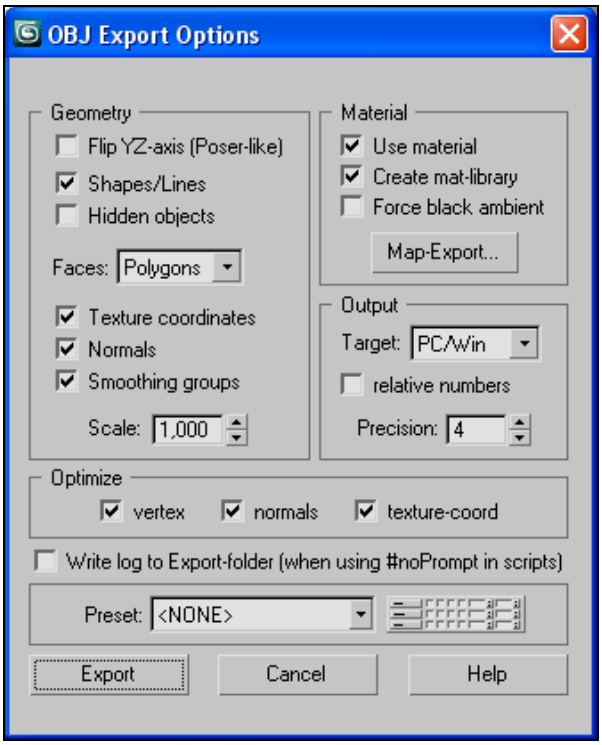


Figura 17 – Configurando parâmetros do arquivo `wavefront obj`

Caso a imagem exportada possua texturas, o 3Ds Max salvará dois arquivos: um com as coordenadas dos objetos (`.obj`) e outro com as texturas (`.mtl`).

Estes procedimentos foram realizados no 3Ds Max 2009 (versão trial).

4 DESENVOLVIMENTO

Neste capítulo são apresentados os requisitos, a especificação, a implementação, bem como a descrição dos resultados e discussões referentes ao desenvolvimento do EMF.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta deve:

a) permitir importação de modelos (vetoriais) para o editor (Requisito Funcional – RF);
b) disponibilizar um conjunto de modelos com vários tipos de cruzamentos, curvas e retas previamente definidas (RF);
c) permitir editar a malha ferroviária (RF);
d) possuir ambiente 3D para desenho e visualização das ferrovias (RF);
e) permitir salvar as ferrovias especificadas (RF);
f) permitir carregar ferrovias salvas anteriormente (RF);
g) permitir a realização de rotação, translação e aproximação da câmera (RF);
h) permitir a realização de rotação e translação dos modelos vetoriais (RF);
i) possuir um método de manipulação de arquivos vetoriais, que serão gerados por softwares como o 3Ds MAX ou Blender (Requisito Não-Funcional – RNF);
j) ser implementado na linguagem Java, utilizando a biblioteca JoGL (RNF).

4.2 ESPECIFICAÇÃO

Utilizou-se da ferramenta Entreprise Architect (EA) juntamente com os diagramas da *Unified Modeling Language* (UML) para a especificação do EMF. Os diagramas utilizados são os de casos de uso, classes, atividades e seqüência, os quais são apresentados a seguir.

4.2.1 Diagrama de casos de uso

O diagrama de casos de uso do EMF é mostrado na Figura 18.

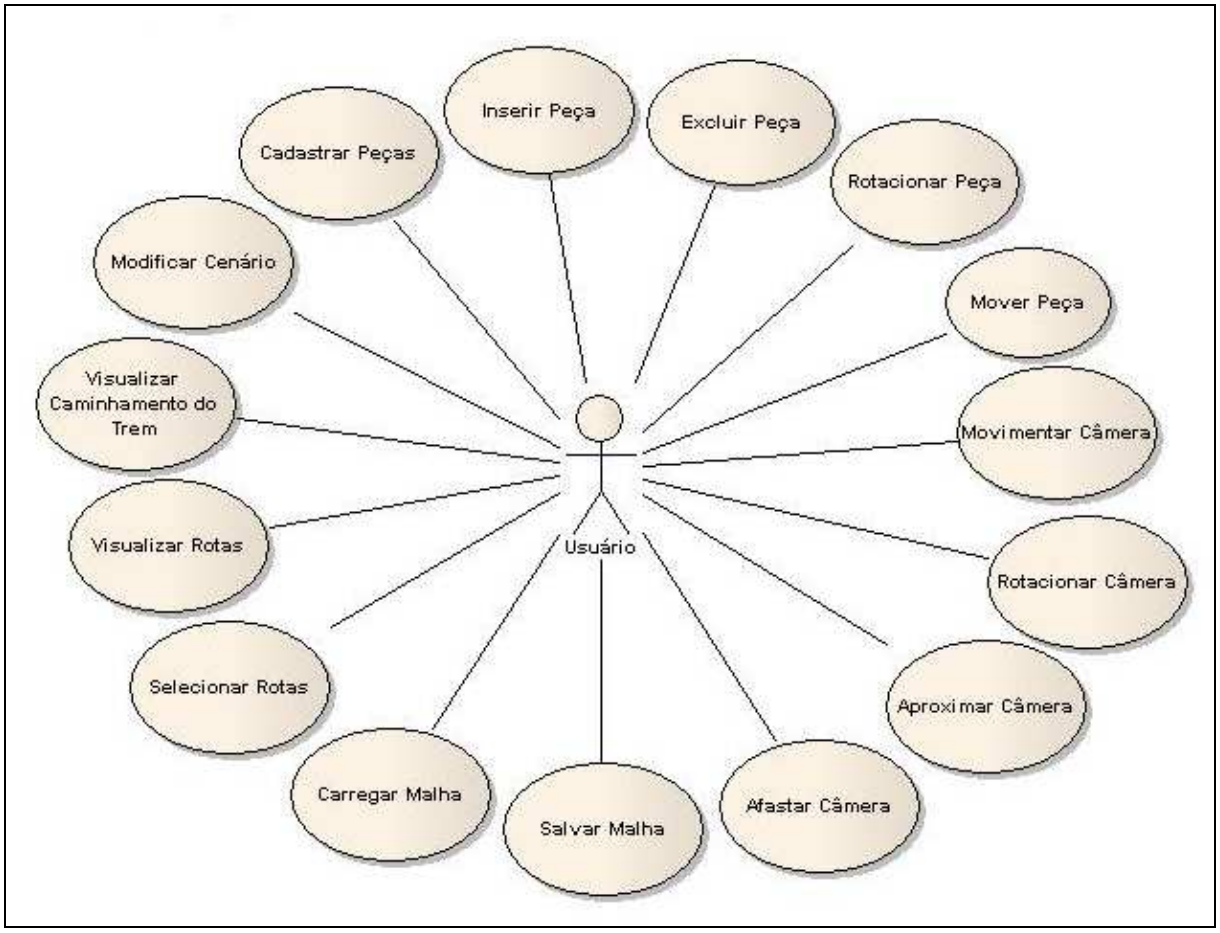


Figura 18 – Diagrama de casos de uso

Os casos de uso da ferramenta são:

a) cadastrar peças: permite ao usuário cadastrar novas peças para serem utilizadas para a confecção das malhas ferroviárias;
b) selecionar rotas: permite ao usuário definir as rotas de caminhamento dentro das peças cadastradas;
c) inserir peça: permite ao usuário inserir uma peça previamente cadastrada no cenário;
d) excluir peça: permite ao usuário excluir uma peça do cenário;
e) rotacionar peça: permite ao usuário rotacionar uma peça do cenário;
f) mover peça: permite ao usuário alterar a posição de uma peça do cenário;
g) movimentar câmera: permite ao usuário alterar a visualização do cenário;
h) rotacionar câmera: permite ao usuário rotacionar a câmera, alterando a

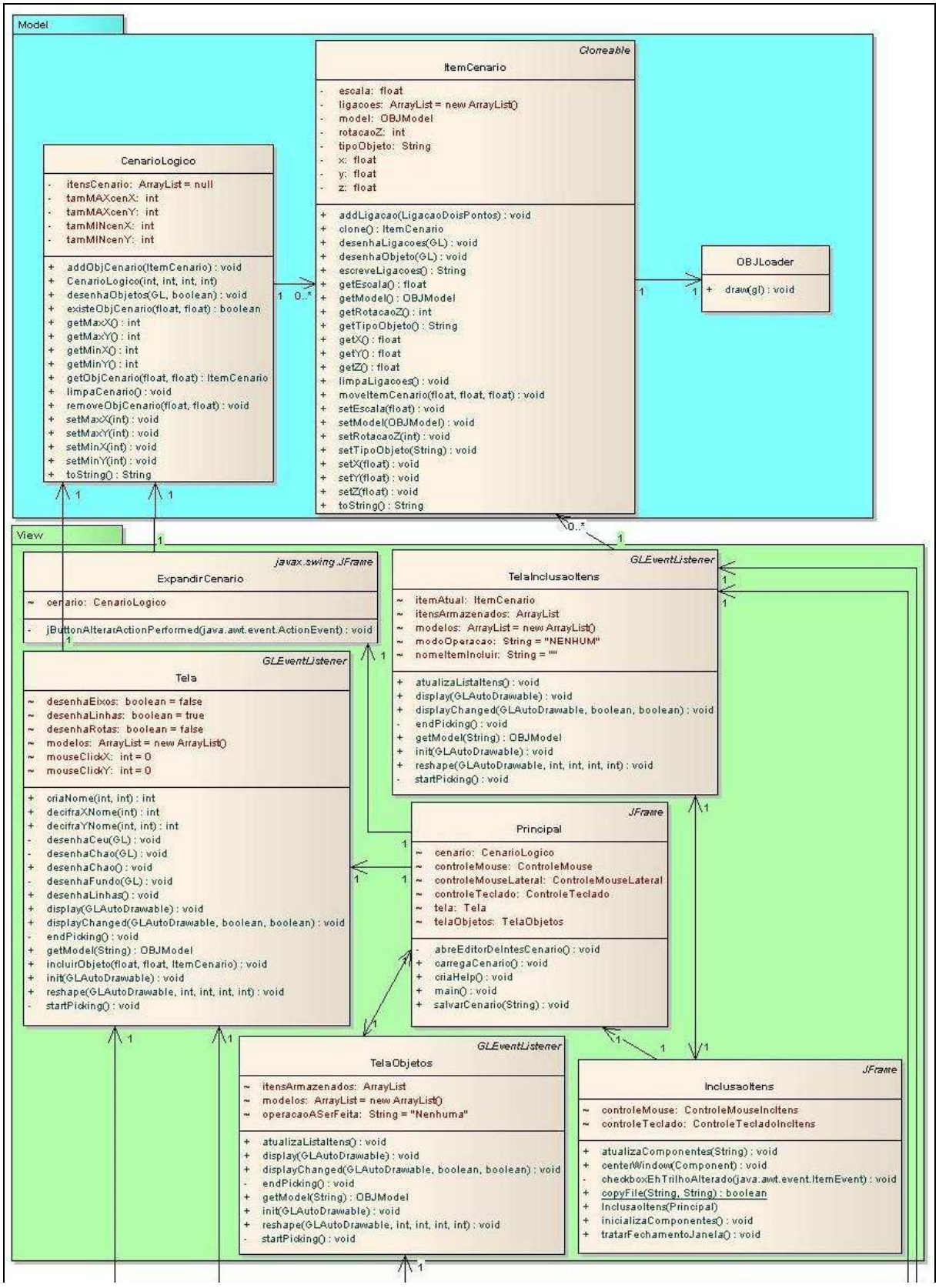
visualização;

i) aproximar / afastar câmera: permite uma visualização mais próxima (*zoom in*) ou mais abrangente (*zoom out*) do cenário;
j) salvar malha: permite ao usuário salvar a malha ferroviária que está sendo editada;
k) carregar malha: permite ao usuário recuperar as informações de uma malha ferroviária salva anteriormente;
l) visualizar rotas: permite ao usuário visualizar as rotas de caminhamento do cenário;
m) modificar cenário: permite ao usuário aumentar ou diminuir a dimensão no cenário;
n) visualizar caminhamento do trem: permite ao usuário visualizar o trem caminhando sobre os trilhos presentes no cenário.

4.2.2 Diagrama de classes

O diagrama de classes do EMF foi dividido em três pacotes: *model*, *view* e *controller*. O pacote *model* é responsável pela modelagem do ambiente, classes que armazenam dados, os quais podem ser modificados no decorrer da execução. O pacote *view* é composto por interfaces com o usuário. O pacote *controller* é composto por classes com o objetivo de tratar os eventos ocorridos durante a execução do programa, eventos esses geralmente gerados pelos usuário ao interagir com as interfaces.

O diagrama de classes é apresentado na Figura 19.


Model
Cloneable
ItemCenario
- escala: float
- ligacoes: ArrayList = new ArrayList()
- model: OBJModel
- rotacaoZ: int
- tipoObjeto: String
- x: float
- y: float
- z: float
+ addLigacao(LigacaoDoisPontos) : void
+ clone() : ItemCenario
+ desenhaLigacoes(GL) : void
+ desenhaObjeto(GL) : void
+ escreveLigacoes() : String
+ getEscala() : float
+ getModel() : OBJModel
+ getRotacaoZ() : int
+ getTipoObjeto() : String
+ getX() : float
+ getY() : float
+ getZ() : float
+ limpaLigacoes() : void
+ moveItemCenario(float, float, float) : void
+ setEscala(float) : void
+ setModel(OBJModel) : void
+ setRotacaoZ(int) : void
+ setTipoObjeto(String) : void
+ setX(float) : void
+ setY(float) : void
+ setZ(float) : void
+ toString() : String
CenarioLogico
- itensCenario: ArrayList = null
- tamMAXcenX: int
- tamMAXcenY: int
- tamMINcenX: int
- tamMINcenY: int
+ addObjCenario(ItemCenario) : void
+ CenarioLogico(int, int, int, int)
+ desenhaObjetos(GL, boolean) : void
+ existeObjCenario(float, float) : boolean
+ getMaxX() : int
+ getMaxY() : int
+ getMinX() : int
+ getMinY() : int
+ getObjCenario(float, float) : ItemCenario
+ limpaCenario() : void
+ removeObjCenario(float, float) : void
+ setMaxX(int) : void
+ setMaxY(int) : void
+ setMinX(int) : void
+ setMinY(int) : void
+ toString() : String
OBJLoader
+ draw(gl) : void
1
0..*
View
javax.swing.JFrame
ExpandirCenario
~ cenario: CenarioLogico
- jButtonAlterarActionPerformed(java.awt.event.ActionEvent) : void
GLEventListener
TelaInclusaoItens
~ itemAtual: ItemCenario
~ itensArmazenados: ArrayList
~ modelos: ArrayList = new ArrayList()
~ modoOperacao: String = "NENHUM"
~ nomeItemIncluir: String = ""
+ atualizaListaItens() : void
+ display(GLAutoDrawable) : void
+ displayChanged(GLAutoDrawable, boolean, boolean) : void
- endPicking() : void
+ getModel(String) : OBJModel
+ init(GLAutoDrawable) : void
+ reshape(GLAutoDrawable, int, int, int, int) : void
- startPicking() : void
GLEventListener
Tela
~ desenhaEixos: boolean = false
~ desenhaLinhas: boolean = true
~ desenhaRotas: boolean = false
~ modelos: ArrayList = new ArrayList()
~ mouseClickX: int = 0
~ mouseClickY: int = 0
+ criaNome(int, int) : int
+ decifraXNome(int) : int
+ decifraYNome(int, int) : int
- desenhaCeu(GL) : void
- desenhaChao(GL) : void
+ desenhaChao() : void
- desenhaFundo(GL) : void
+ desenhaLinhas() : void
+ display(GLAutoDrawable) : void
+ displayChanged(GLAutoDrawable, boolean, boolean) : void
- endPicking() : void
+ getModel(String) : OBJModel
+ incluirObjeto(float, float, ItemCenario) : void
+ init(GLAutoDrawable) : void
+ reshape(GLAutoDrawable, int, int, int, int) : void
- startPicking() : void
JFrame
Principal
~ cenario: CenarioLogico
~ controleMouse: ControleMouse
~ controleMouseLateral: ControleMouseLateral
~ controleTeclado: ControleTeclado
~ tela: Tela
~ telaObjetos: TelaObjetos
- abreEditorDeItnesCenario() : void
+ carregaCenario() : void
+ criaHelp() : void
+ main() : void
+ salvarCenario(String) : void
GLEventListener
TelaObjetos
~ itensArmazenados: ArrayList
~ modelos: ArrayList = new ArrayList()
~ operacaoASerFeita: String = "Nenhuma"
+ atualizaListaItens() : void
+ display(GLAutoDrawable) : void
+ displayChanged(GLAutoDrawable, boolean, boolean) : void
- endPicking() : void
+ getModel(String) : OBJModel
+ init(GLAutoDrawable) : void
+ reshape(GLAutoDrawable, int, int, int, int) : void
- startPicking() : void
JFrame
InclusaoItens
~ controleMouse: ControleMouseIncItens
~ controleTeclado: ControleTecladoIncItens
+ atualizaComponentes(String) : void
+ centerWindow(Component) : void
- checkboxEhTrilhoAlterado(java.awt.event.ItemEvent) : void
+ copyFile(String, String) : boolean
+ InclusaoItens(Principal)
+ inicializaComponentes() : void
+ tratarFechamentoJanela() : void

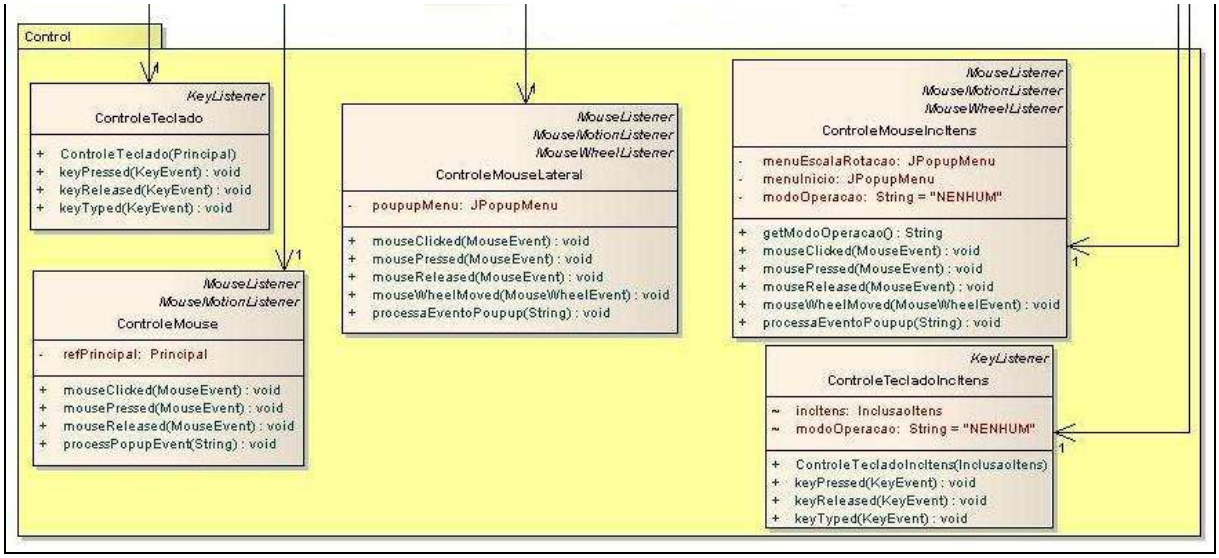


Figura 19 – Diagrama de classes

As classes desenvolvidas para a ferramenta são:

a) `Principal`: responsável pela execução do programa e a inicialização de outras classes. Possui um menu de opções e trata os eventos para cada opção deste menu;

b) `Tela`: responsável por exibir o cenário principal na tela e permitir incluir, excluir e alterar as peças incluídas no cenário;

c) `ControleMouse`: responsável pelos eventos de *mouse* na tela principal do ambiente;

d) `ControleTeclado`: responsável pelos eventos de teclado na tela principal do ambiente;

e) `CenarioLogico`: responsável por armazenar informações do cenário como tamanho e itens incluídos;

f) `ItemCenario`: responsável por armazenar informações sobre as peças do cenário;

g) `ExpandirCenario`: responsável por exibir a tela de expansão de cenário, permitindo configurar o tamanho do mesmo;

h) `TelaObjetos`: responsável por exibir a tela com as peças cadastrados para que o usuário possa selecioná-las e incluí-las no cenário;

i) `ControleMouseLateral`: responsável pelos eventos de *mouse* na tela de objetos;

j) `InclusaoItens`: responsável por exibir a tela com as opções para cadastramento de peças do cenário;

k) `TelaInclusaoItens`: responsável por exibir a tela de edição de peças, onde são definidas as rotas de caminhamento, escala e rotação das peças;

l) `ControleMouseIncItens`: responsável pelos eventos de *mouse* ocorridos na tela

de inclusão de peças;

o) `ControleTecladoIncItens`: responsável pelos eventos de teclado ocorridos na tela de inclusão de peças;

m) `ObjLoader`: conjunto de classes que tem por finalidade fazer a leitura de arquivos vetoriais `wavefront obj` e exibi-los na tela. Estas classes foram especificadas por Andrew Davison (DAVISON, 2007, p. 439).

4.2.3 Diagrama de atividades

O diagrama de atividades demonstra possíveis ações tomadas pelo usuário ao interagir com o EMF (Figura 20).

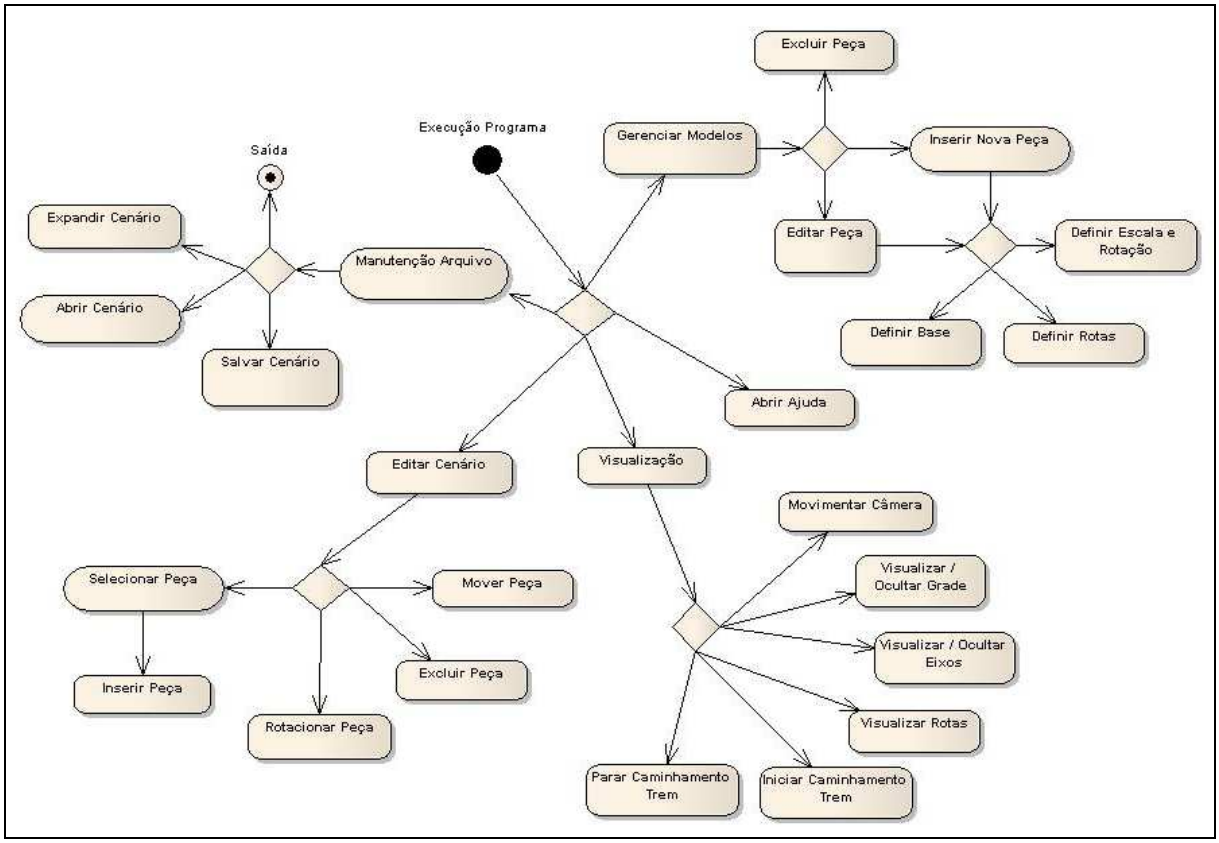


Figura 20 – Diagrama de atividades

A partir da abertura do programa as ações possíveis para o usuário são:

a) manutenção de arquivo: permite ao usuário:

- salvar cenário: armazenar o cenário atual em um arquivo,
- abrir cenário: recuperar um cenário a partir de um arquivo,
- expandir cenário: alterar o área útil do cenário, onde na qual é possível incluir

peças,
- sair: encerrar a aplicação;

b) editar o cenário: permite ao usuário:
- selecionar peça: selecionar uma peça da lista de peças,
- inserir peça: incluir no cenário a peça selecionada,
- excluir peça: excluir uma peça do cenário,
- rotacionar peça: rotacionar uma peça do cenário,
- mover peça: mover uma peça do cenário para uma posição vazia;

c) visualização: permite ao usuário:
- visualizar rotas: visualizar os possíveis caminhos para caminhamento de trens,
- visualizar / ocultar eixos: visualizar os eixos centrais do cenário,
- visualizar / ocultar grade: visualizar as grades que demarcam as posições do cenário,
- movimentar câmera: movimentar e rotacionar a câmera,
- iniciar caminhamento do trem: exibe o trem no cenário e faz com que o mesmo comece a andar sobre os trilhos,
- parar caminhamento do trem: para o trem e oculta o mesmo;

d) gerenciar modelos: permite ao usuário:
- inserir nova peça: localizar o arquivo `obj` de uma nova peça e em seguida definir base, rotas e escala,
- excluir peça: excluir uma peça cadastrada,
- editar peça: redefinir base, escala e rotas de uma peça existente,
- definir base: definir a altura base da peça,
- definir rotas: definir os caminhos para caminhamento de trens,
- definir escala e rotação: definir o tamanho e rotação inicial das peças;

e) abrir ajuda: permite ao usuário visualizar tópicos de ajuda.

4.2.4 Diagrama de seqüência

Nesta seção são apresentados os diagramas de seqüência das funcionalidades do EMF como: criação de novo cenário, salvar um cenário, abrir (carregar) um cenário, adicionar uma peça ao cenário e cadastrar uma nova peça.

Para a demonstração de laços e considerações o EA (ferramenta utilizada para

confeccionar o diagrama) utiliza a palavra `loop` quando se utiliza um laço e a palavra `consider` quando se deseja fazer alguma consideração da operação realizada. Após estas palavras pode-se dar um nome para identificar a operação e para o `loop` pode-se atribuir uma condição para que ela termine. Essas funcionalidades foram utilizadas nos diagramas de seqüência do EMF.

O diagrama de seqüência apresentado na Figura 21 demonstra a execução da criação de um novo cenário.

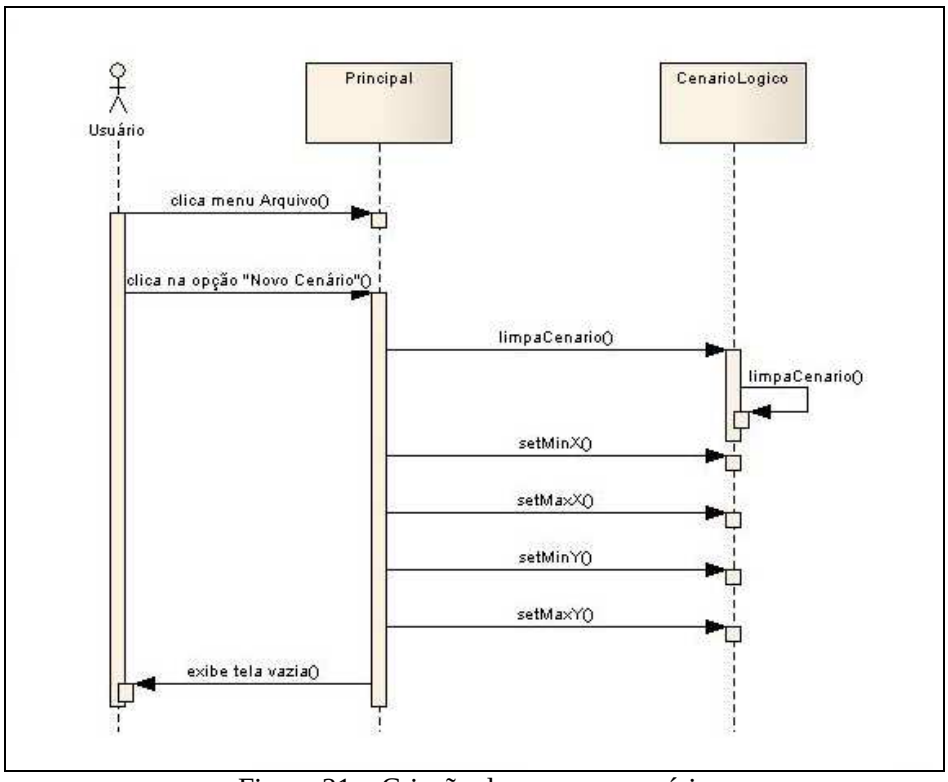


Figura 21 – Criação de um novo cenário

A operação de criação de um novo cenário (Figura 21) é iniciada quando o usuário clica com o *mouse* no menu `Arquivo` e em seguida na opção `Novo Cenário`. A função executada exclui todos os objetos presentes no cenário e retorna o tamanho do cenário para o tamanho original.

O diagrama de seqüência apresentado na Figura 22 demonstra a execução da operação para salvar o cenário que está sendo utilizado.

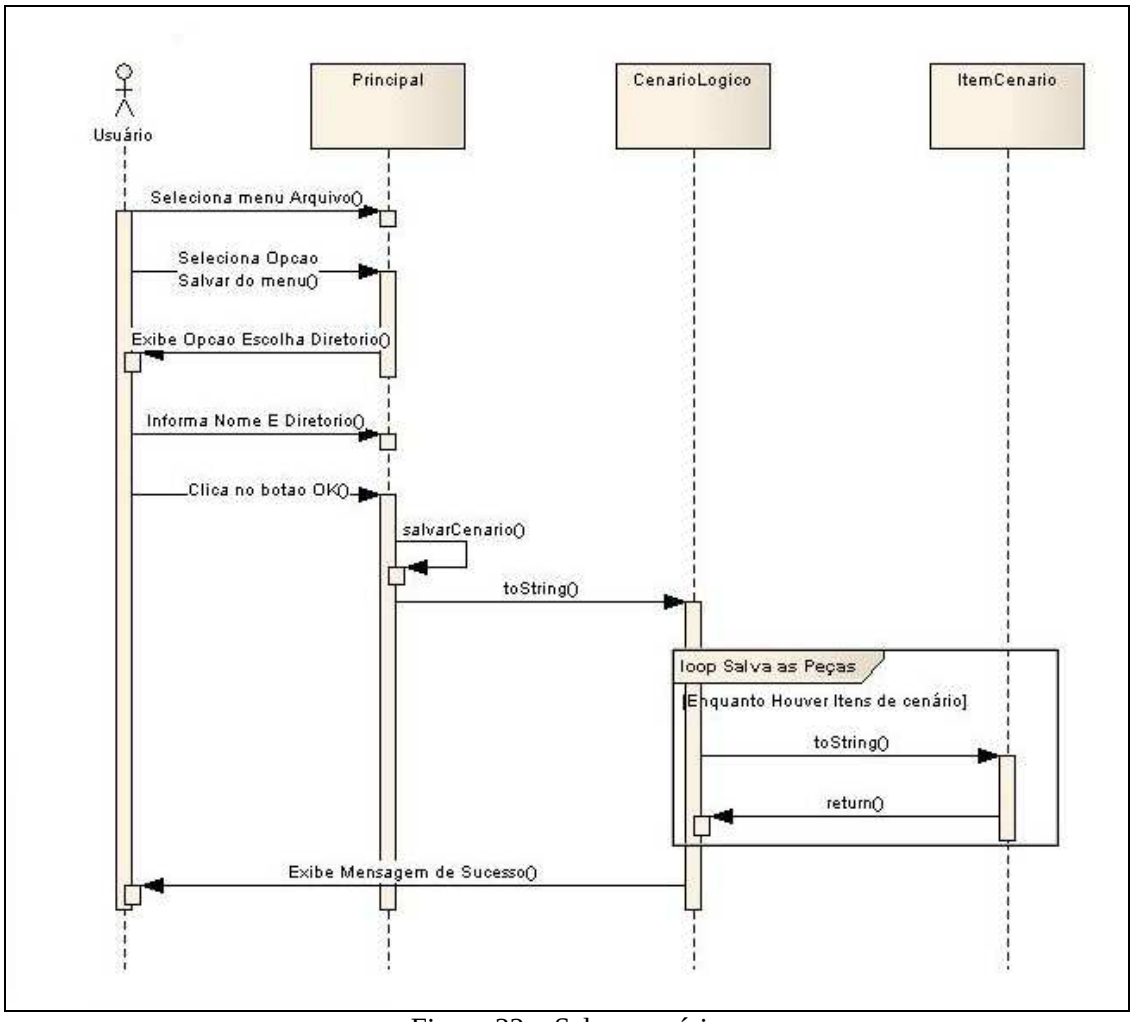


Figura 22 – Salvar cenário

A operação para salvar o cenário (Figura 22) é iniciada quando o usuário clica com o *mouse* no menu `Arquivo` e em seguida na opção `Salvar Cenário`. O sistema apresenta a interface para seleção de diretório e nome do arquivo onde o cenário será salvo. Ao informar os dados necessários o aplicativo armazena em um arquivo texto as informações do tamanho de cenário e em seguida as peças e suas localizações.

O diagrama de seqüência apresentado na Figura 23 demonstra a execução da operação para abrir um cenário salvo em um arquivo.

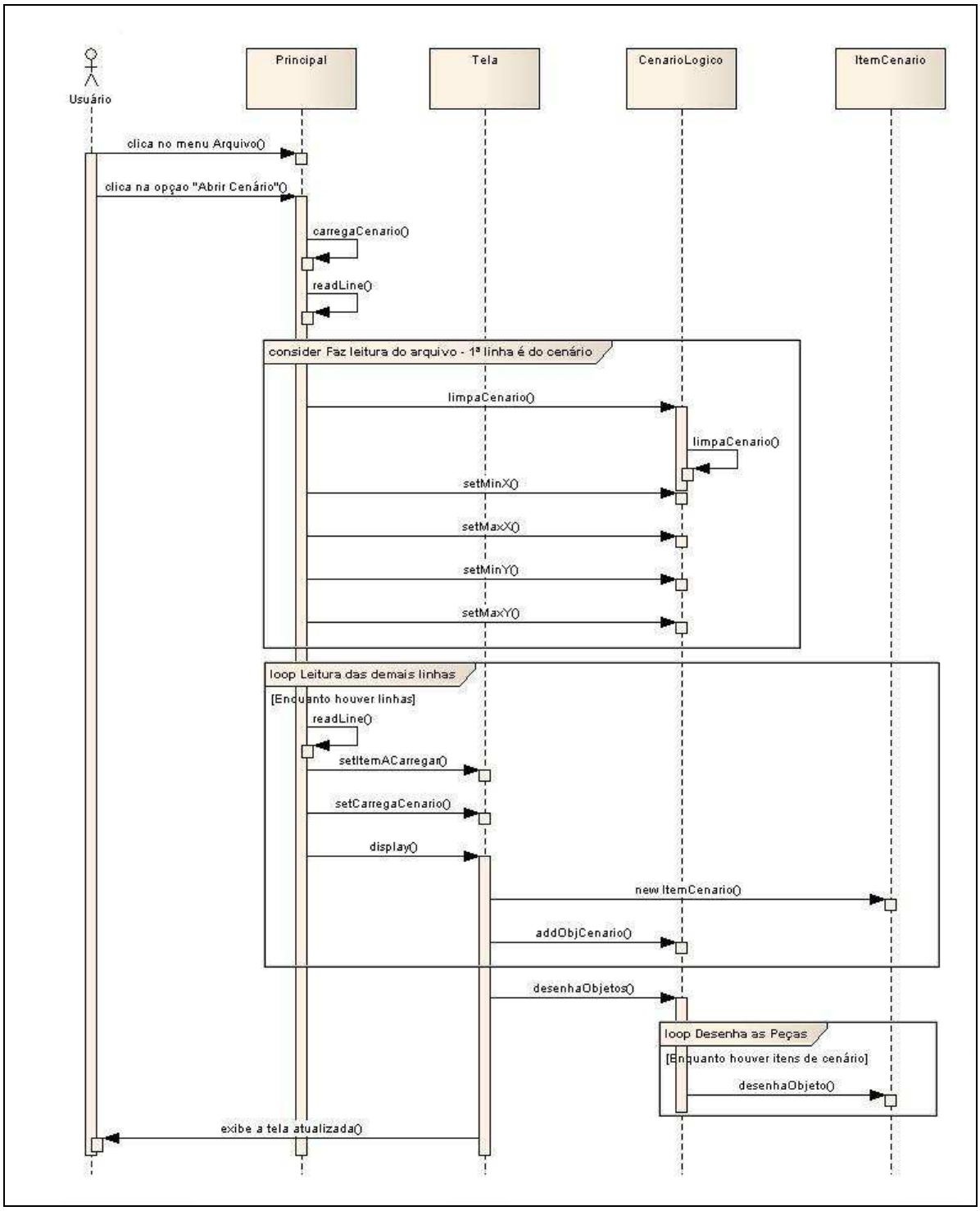


Figura 23 – Abrir cenário

A operação para abrir um cenário (Figura 23) é iniciada quando o usuário clica com o *mouse* no menu `Arquivo` e em seguida na opção `Abrir Cenário`. O sistema apresenta a interface para localização de arquivo e o usuário deve informá-lo. Ao informá-lo o aplicativo inicia a leitura do arquivo texto, sendo que a primeira linha armazena as informações de tamanho do cenário e as demais linhas informações das peças e suas localizações. Após a leitura é exibido na tela o cenário carregado.

O diagrama de seqüência apresentado na Figura 24 demonstra a execução da operação para adicionar uma peça ao cenário.

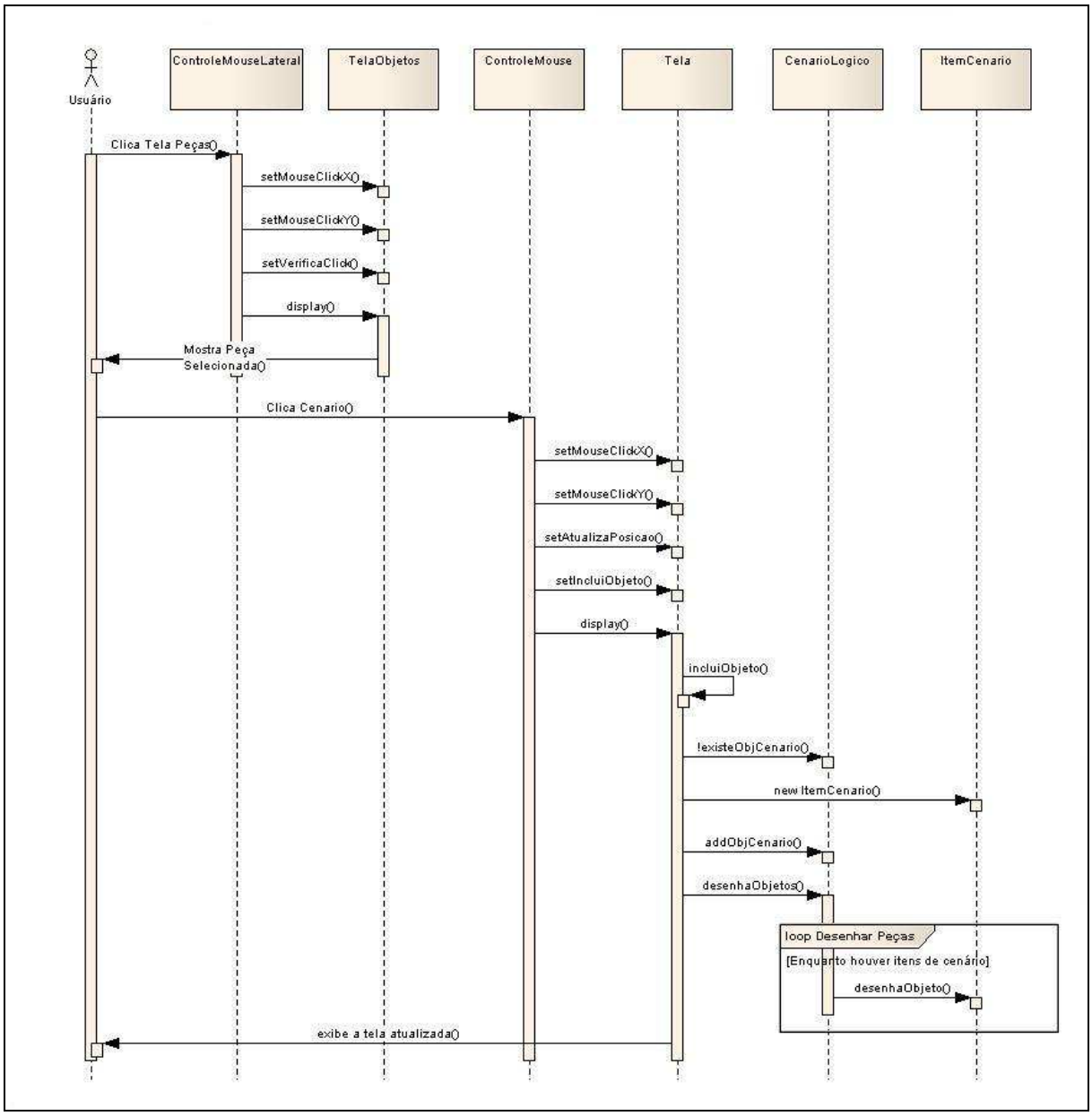


Figura 24 – Adicionar peça ao cenário

Para adicionar uma peça ao cenário (Figura 24) é necessário selecionar uma peça na tela lateral de itens, bastando clicar com o botão esquerdo do *mouse* sobre uma das peças. Tendo uma peça selecionada basta clicar em uma das áreas do cenário. Ao clicar em alguma área do cenário é realizada uma verificação da ocupação da mesma. Se a área estiver desocupada, a peça é incluída no cenário. Após a inclusão no cenário a tela é redesenhada.

O diagrama de seqüência apresentado na Figura 25 demonstra a execução da operação para cadastrar uma nova peça de cenário.

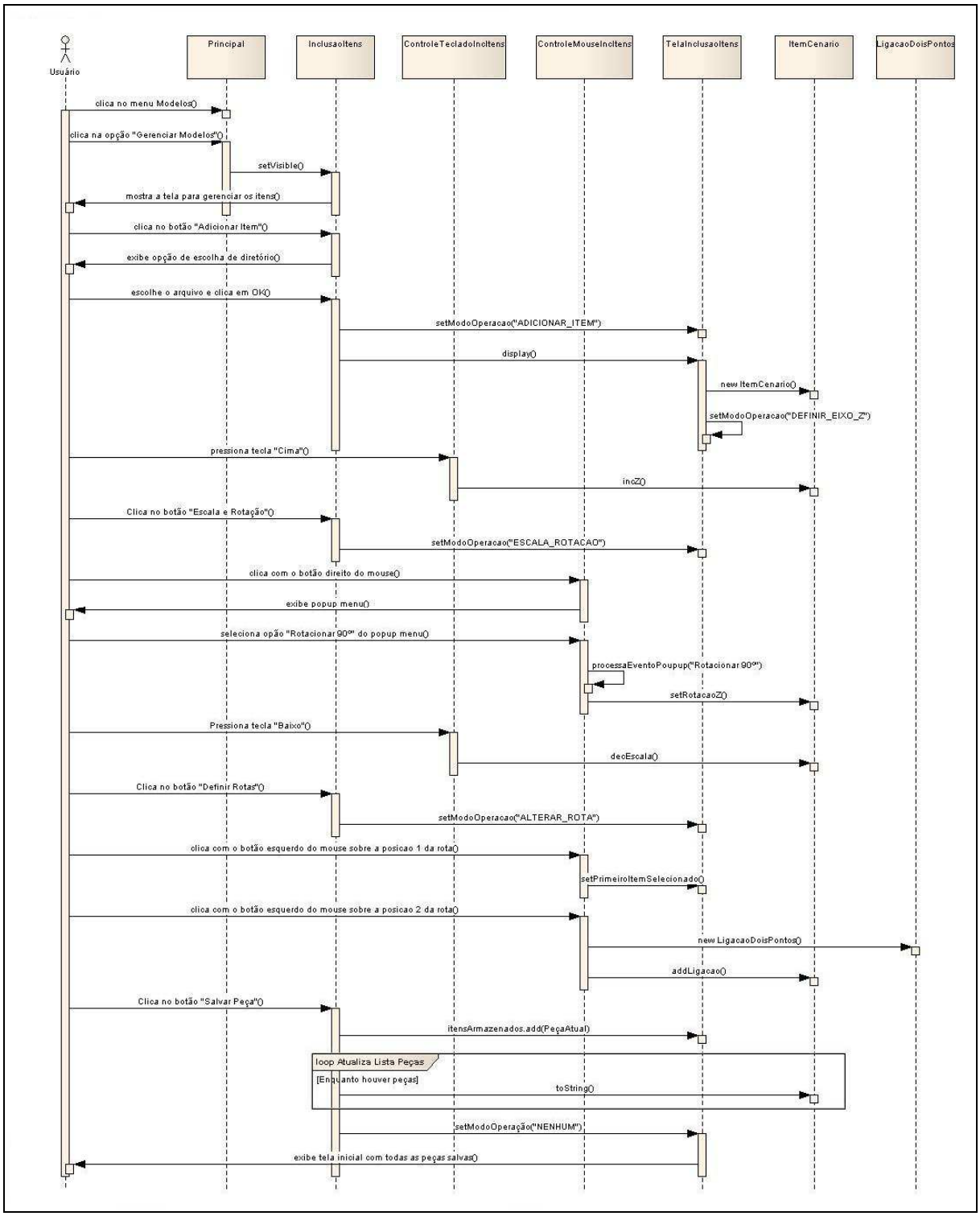


Figura 25 – Cadastrar uma nova peça de cenário

Para cadastrar uma nova peça (Figura 25) é necessário clicar com o *mouse* no menu `Modelos` e em seguida na opção `Gerenciar Modelos`. Após, a interface para gerenciamento de peças é apresentada. Nesta interface deve-se clicar no botão `Adicionar Item`. O aplicativo apresentará a interface para seleção do arquivo do modelo da peça. Após localizar o arquivo a interface apresenta a tela para definição da altura base da peça em relação ao chão, permitindo

ao usuário pressionar as teclas `cima` para subir e `baixo` para descer a peça. Este procedimento visa posicionar visualmente a peça no chão, sem que haja a necessidade de precisão. Após definir a altura o usuário deve clicar no botão `Escala e Rotação` para que o sistema exiba a interface que permite a configuração do tamanho da peça e da rotação inicial sobre o eixo z. Após definir a escala e a rotação, o usuário deve clicar no botão `Definir Rotas` para que o sistema exiba a interface para cadastramento de rotas de caminhamento. As rotas de caminhamento são utilizadas pelo trem para identificar o caminho a ser percorrido em cada peça. Ao terminar o cadastramento, o usuário deve clicar no botão `Salvar Peça` para que o sistema inclua a mesma na lista de peças cadastradas. Após o salvamento, o sistema exibe todas as peças cadastradas.

4.3 IMPLEMENTAÇÃO

Esta seção consiste na descrição das funções/procedimentos utilizados pelo EMF, que por sua vez foi implementado utilizando a linguagem Java, juntamente com o ambiente Netbeans 5.5 e as bibliotecas JoGL e JavaHelp (JAVA.NET, 2008).

A seguir são descritas algumas das funções/procedimentos utilizados na implementação, como leitura de arquivos `wavefront obj`, *display lists*, utilização de texturas, seleção e a ajuda da ferramenta.

4.3.1 Leitura de arquivos `wavefront obj`

Para que o EMF possa ler e exibir arquivos `wavefront obj` utilizou-se do conjunto de classes especificadas por Davison (2007, p. 439). A leitura e o comando de exibição de um arquivo `wavefront obj` é mostrado no Quadro 3.

```
//Indica o local onde o arquivo obj está armazenado
String nomeArquivo = "C:\\nomeArquivo.obj";

//Indica o tamanho do obj que será utilizado. Para 100% usar 1
float escala = 1;

//Se true exibe algumas informações sobre o obj na tela
Boolean exibeInformacoesOBJ = false;


//Cria uma nova instância de OBJModel
OBJModel model = new OBJModel(nomeArquivo,escala,gl,exibeInformacoesOBJ);

//Desenha o obj na tela
model.draw(gl);
```

Quadro 3 – Inicializando a leitura de um arquivo `obj`

Basicamente a classe `OBJModel` consiste em fazer a leitura do arquivo, gravar as informações lidas em `display lists` (apresentado na seção 4.3.2) e exibi-las na tela quando executado o comando `draw`.

A alteração de tamanho, rotação e a translação do objeto carregado é feita através das funções do JoGL `glScalef`, `glRotatef` e `glTranslatef`, bastando utilizar as funções `glPushMatrix` e `glPopMatrix` para isolar as transformações somente para o objeto (Quadro 4).

```
//Isola as transformações para o objeto apenas
gl.glPushMatrix();

   //Move o objeto para a posição 10,5,0 (x,y,z)
   gl.glTranslatef(10, 5, 0);

   //Rotacionar 90º sobre o eixo z
   gl.glRotatef(90, 0, 0, 1);

   //Define o tamanho do objeto, no caso dobrando o tamanho
   gl.glScalef(2, 2, 2);

   //Comando para desenha o objeto
   getModel().draw(gl);

//Termina a isolação para as transformações de objeto
gl.glPopMatrix();
```

Quadro 4 – Definindo posição, escala e rotação do objeto

4.3.2 *Display Lists*

Jouvie (2007) diz que `display lists` são usados para se obter aumento de performance na renderização de imagens. São mais usados quando utiliza-se grande

quantidade de vértices nas aplicações.

`Display lists` armazenam comandos OpenGL em uma lista e armazena esta lista na lista de execução do OpenGL. No modo normal de execução, sem a utilização de `display lists`, todos os comandos OpenGL são reenviados à lista de execução do OpenGL sempre que a tela é atualizada, havendo assim processamento desnecessário e conseqüentemente perda de performance.

Para a criação de `display lists` utiliza-se o comando `glGenLists` para criar a lista e os comandos `glNewList` e `glEndList` para iniciar e parar a gravação dos comandos OpenGL. Depois de criada a lista com os comandos OpenGL, utiliza-se o comando `glCallList` para executar os comandos armazenados na mesma.

Após a criação de uma lista, não é possível editá-la. Caso se deseje modificar a mesma é necessário deletar a lista atual e criar uma nova. Para deletar uma ou mais listas utiliza-se o comando `glDeleteLists` e para verificar se uma lista foi inicializada e possui comandos armazenados utiliza-se `glIsList`.

4.3.3 Utilização de texturas

No editor de malhas ferroviárias foram utilizadas texturas para o chão, para as laterais e para o céu. Assim como todo desenho feito no JoGL, as texturas precisam ser redesenhadas cada vez que há uma alteração na tela, utilizando a função `display`. No Quadro 5 é mostrada a forma de carregamento de imagens utilizadas no editor de malhas ferroviárias.

```
//Variáveis globais
private Texture texturaCeu;

private String localTexturaCeu = "Referencias\\texturas\\TOPO.jpg";

//Coordenadas das extremidades da área para inclusao da figura do céu
float[][] ceu; = { {-20f, -20f, 10f}, { 20f, -20f, 10f},
                  { 20f,  20f, 10f}, {-20f,  20f, 10f}   };



//Inicialização da variável – feito dentro do método init
try {
     texturaCeu = TextureIO.newTexture(new File(localTexturaCeu), true);
} catch(IOException e) {
     System.err.println("Erro ao tentar ler texturas");
}
```

Quadro 5 – Carregando textura

Com a textura carregada deve-se incluir na função `display` o comando para desenhá-

la na tela. Antes de executar o comando de desenho é necessário habilitar o modo de visualização de imagens (texturas) bidimensionais. O Quadro 6 mostra a aplicação da textura em um retângulo.

```
public void display(GLAutoDrawable drawable) {
   ...


   //Habilita o modo de visualização de texturas bidimensionais
   gl.glEnable(GL.GL_TEXTURE_2D);
   texturaCeu.bind();

   //Carrega as coordenadas limites da textura
   TextureCoords coords = texturaCeu.getImageTexCoords();
       //Informa que a forma geométrica a ser desenhada
       //a textura é um retângulo
       gl.glBegin(GL.GL_QUADS);
           //Informa as coordenadas do retângulo
           //que receberá a imagem (no caso estão guardadas nas coordenadas
           //do array céu)
           gl.glNormal3f(0,0,1.0f);
           //Indica o canto do retângulo
           gl.glTexCoord2f(coords.left(),coords.bottom());
           //Indica a posição do canto
           gl.glVertex3fv(ceu[0],0);
           gl.glTexCoord2f(coords.right(),coords.bottom());
           gl.glVertex3fv(ceu[1],0);
           gl.glTexCoord2f(coords.right(),coords.top());
           gl.glVertex3fv(ceu[2],0);
           gl.glTexCoord2f(coords.left(),coords.top());
           gl.glVertex3fv(ceu[3],0);
       gl.glEnd();
   //Desabilita o modo de visualização de texturas bidimensionais
   gl.glDisable(GL.GL_TEXTURE_2D);

   ...
}
```

Quadro 6 – Desenhando a textura

A visualização de texturas carregadas utilizando os códigos descritos no Quadro 5 e no Quadro 6 é mostrado na Figura 26.

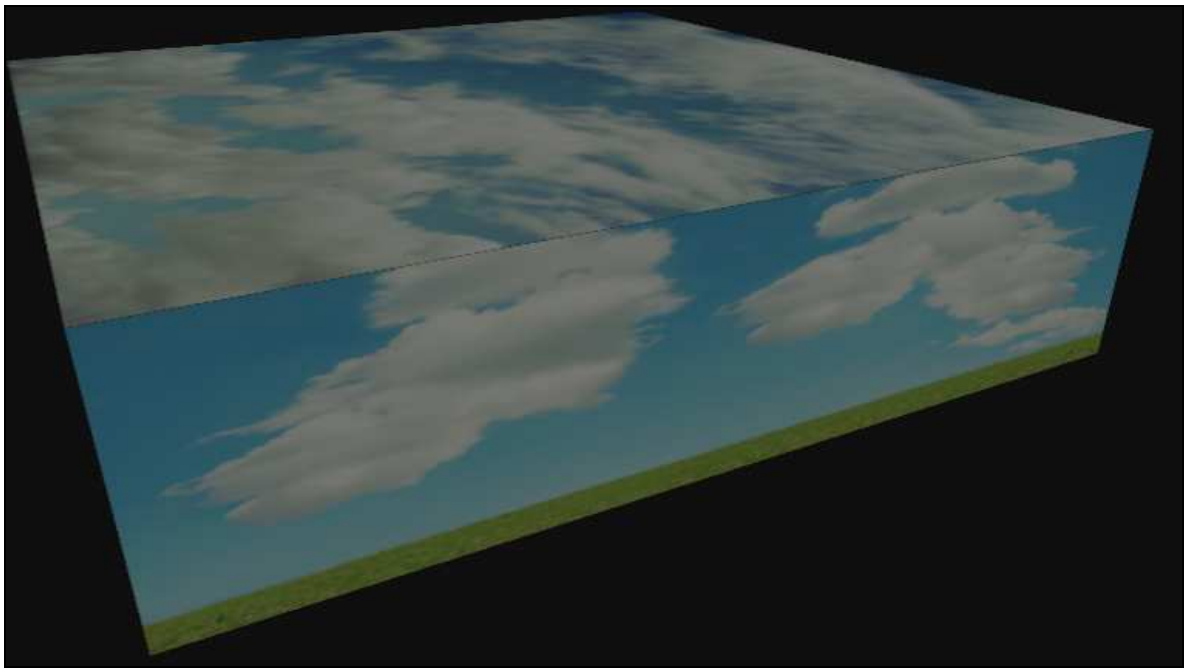


Figura 26 – Visualização de textura no EMF

4.3.4 Seleção

As rotinas padrões da biblioteca gráfica JoGL não possuem métodos de conversão de clique do *mouse* em um ponto da tela para um ponto no ambiente 3D. Quase todos os recursos do EMF são acessados através de cliques de *mouse* e para que o clique do *mouse* possa ser reconhecido no ambiente 3D utilizou-se funções para seleção de objetos.

Estas funções consistem em nomear cada objeto desenhado na tela com um número inteiro e ao clicar com o *mouse* em uma posição da tela, uma busca é feita em todos os pontos próximos ao clique o objeto nomeado, retornando o número atribuído anteriormente.

Esta técnica foi implementada por Davison (2007, p. 465) e é feita em três etapas. Primeiramente inicia-se o modo de seleção, inicializa a lista de nomes e os *buffers* (Quadro 7). Em um segundo momento atribui-se o nome aos objetos desejados utilizando as funções `glPushName` e `glPopName` do JoGL, como é mostrado no Quadro 8. Na terceira e última etapa é finalizado o modo de seleção (Quadro 9) e feito uma busca na lista de nomes (Quadro 10).

```
int BUFSIZE = 512;
//Inicializa o modo de seleção
private void startPicking() {
    //Inicializa os buffers
    int selectBuf[] = new int[BUFSIZE];
    selectBuffer = BufferUtil.newIntBuffer(BUFSIZE);

    //Carrega os buffers
    gl.glSelectBuffer(BUFSIZE, selectBuffer);

    //Ativa o modo de seleção
    gl.glRenderMode(GL.GL_SELECT);

    //Inicializa a lista de nomes
    gl.glInitNames();
    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glPushMatrix();
    gl.glLoadIdentity();
    int viewport[] = new int[4];
    //Carrega dados da tela (dimensões)
    gl.glGetIntegerv(GL.GL_VIEWPORT, viewport, 0);

    //Cria uma area de 5x5 pixels ao redor do cursor
    glu.gluPickMatrix((double) mouseClickX, (double) (viewport[3] –
                                mouseClickY), 5.0, 5.0, viewport, 0);

    float campoVisao = 45.0f;
    float proxPlanoZ = 1f;
    float longePlanoZ = 100f;
    //Configura a perspectiva
    glu.gluPerspective(campoVisao,
          (float)viewport[2]/(float)viewport[3],proxPlanoZ,longePlanoZ);
    gl.glMatrixMode(GL.GL_MODELVIEW);
}
```

Quadro 7 – Iniciando o modo de seleção

```
//Nome que será atribuído ao objeto
int nome = 1;

//Indica que iniciou o desenho a qual se quer nomear
gl.glPushName(nome);

    //Desenha algo, no caso um quadrado
    gl.glBegin(GL.GL_QUADS);
      gl.glVertex3d(x,y,0);
      gl.glVertex3d(x,y+1,0);
      gl.glVertex3d(x+1,y+1,0);
      gl.glVertex3d(x+1,y,0);
    gl.glEnd();

//Indica que o desenho está complete
gl.glPopName();
```

Quadro 8 – Nomeando os objetos

```
//Finaliza o modo de seleção
   private void endPicking() {
        gl.glMatrixMode(GL.GL_PROJECTION);
        gl.glPopMatrix();
        gl.glMatrixMode(GL.GL_MODELVIEW);
        gl.glFlush();
        int numHits = gl.glRenderMode(GL.GL_RENDER);

        //Faz a verificação dos itens
        processHits(numHits);
   }
```

Quadro 9 – Finalizando modo de seleção

```
public void processHits(int numHits)      {
        //Se não acertou em nenhum objeto desenhado não retorna nada
        if (numHits == 0)
            return;

        //Reseta o nome do objeto encontrado
        selectedNameID = -1;
        float smallestZ = -1.0f;
        boolean isFirstLoop = true;
        int offset = 0;
        for (int i=0; i < numHits; i++) {
            int numNames = selectBuffer.get(offset);
            offset++;
            float minZ = getDepth(offset);

            offset++;
            if (isFirstLoop) {
                smallestZ = minZ;
                isFirstLoop = false;
            }
            else {
                if (minZ < smallestZ)
                smallestZ = minZ;
            }
            float maxZ = getDepth(offset);
            offset++;
            int nameID;

            //Varre a lista de coordenadas próximas ao clique
            for (int j=0; j < numNames; j++){
                nameID = selectBuffer.get(offset);
                if (j == (numNames-1)) {
                    if (smallestZ == minZ)
                        //Grava o número do objeto selecionado
                        selectedNameID = nameID;
                }
                offset++;
            }
        }
   }

   private float getDepth(int offset) {
        long depth = (long) selectBuffer.get(offset);
        return (1.0f + ((float) depth / 0x7fffffff));
   }
```

Quadro 10 – Buscando possíveis objetos selecionados

A função `processHits` recupera o número inteiro, o qual serviu para nomear o objeto

selecionado e o armazena na variável `selectedNameID`. Se no local clicado não possuir nenhum objeto selecionável, o número armazenado na variável `selectedNameID` será -1, indicando que não houve seleção.

No EMF foi necessário nomear todas as posições do chão do cenário, para que essas posições pudessem tornar-se selecionáveis. Se o EMF inicia com um cenário com dez posições de largura e dez posições de comprimento, têm-se cem (100) posições. O número de posições pode ser alterado durante a execução. Para que não houvesse a necessidade de armazenar o nome de cada posição do cenário, utilizou-se uma função que gera um número inteiro baseado em suas coordenadas x e y. Para recuperar a posição é usada a função reversa utilizando o nome retornado pela função de seleção. A função utilizada para gerar o nome dos objetos é mostrada no Quadro 11 e as funções utilizadas para recuperar as coordenadas do objeto são mostradas no Quadro 12.

```
//Cria nome para objeto baseando-se em suas coordenadas x e y
public int criaNome(int x, int y) {

   //como o cenário é composto por posições positivas e
   //negativas, utilizou-se números abaixo de 500 para os
   //negativos, 500 para o 0 e acima de 500 para os positivos

   return (((x + 500) * 1000) + (y + 500));

   //O nome gerado varia de 0 a 999.999, sendo que 0 representa
   //a posição x = -500, y = -500 e 999.999
   //a posição x =  499, y =  499,
   //limitando-se a um milhão de posições.

}
```

Quadro 11 – Criando nome para um objeto

```
//Decifra a coordenada de x.
public int decifraXNome(int nome) {

    int temp = (nome / 1000);
    return (temp - 500);

}


//Decifra o a coordenada de y, porém é necessário
//decifrar x anteriormente.
public int decifraYNome(int nome, int x) {

    return ((nome - ((x + 500) * 1000)) - 500);

}
```

Quadro 12 – Recuperando coordenadas do objeto selecionado

4.3.5 Ajuda do EMF

Para a criação do sistema de ajuda do EMF utilizou-se a biblioteca Java Help 2.0. O conteúdo da ajuda foi feito utilizando a linguagem *HyperText Markup Language* (HTML) e o Java Help foi utilizado para criar a interface e o sistema de busca da ajuda.

O código fonte utilizado para executar Java Help no EMF é mostrado no Quadro 13.

```
//local do arquivo de ajuda
String helpHS = "Ajuda/MinhaAjuda.hs";
HelpSet hs;
HelpBroker hb;

try {
     //Abre o arquivo de ajuda
     hs = new HelpSet(null, this.getClass().getResource(helpHS));
} catch (Exception e) {
     //Caso nao encontre o arquivo exibe mensagem
     System.out.println("HelpSet " + e.getMessage());
     System.out.println("HelpSet " + helpHS + " not found");
     return;
}
//cria o help
hb = hs.createHelpBroker();
//Define o tamanho da janela do java help
Dimension ds = new Dimension(980,600);
//Atribui o tamanho ao help
hb.setSize(ds);
//Mostra o help
hb.setDisplayed(true);
```

Quadro 13 – Definindo parâmetros da ajuda

No Quadro 13 é visto `Ajuda/MinhaAjuda.hs` como única referência externa à aplicação. Este arquivo armazena as informações de inicialização e referências aos outros arquivos utilizados pelo Java Help. O conteúdo deste arquivo (Quadro 14) e dos demais arquivos utilizados pelo Java Help é feito em *eXtensible Markup Language* (XML).

```
<helpset version="1.0">
 <title>Ajuda do EMF </title>
 <maps>
  <homeID>
top
  </homeID>
  <mapref location="Map.jhm"/>

 </maps>
 <view>
  <name>
TOC
  </name>
  <label>
TOC
  </label>
  <type>
javax.help.TOCView
  </type>
  <data>
MinhaAjudaTOC.xml
  </data>
 </view>
 <view>
  <name>
Search
  </name>
  <label>
Busca
  </label>
  <type>
javax.help.SearchView
  </type>
  <data engine="com.sun.java.help.search.DefaultSearchEngine">
JavaHelpSearch
  </data>
 </view>
</helpset>
```

Quadro 14 – Estrutura do arquivo `MinhaAjuda.hs`

No Quadro 14 encontram-se referências para dois arquivos: `Map.jhm` e `MinhaAjudaTOC.xml`. O arquivo `Map.jhm` (Quadro 15) armazena as referências de todos arquivos HTML usados na ajuda. O arquivo `MinhaAjudaTOC.xml` (Quadro 16) armazena somente o nome exibido e a referência dos arquivos HTML que o usuário poderá acessar e efetuar buscas.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<map version="1.0">
<mapID target="top" url="index.html "/>

<mapID target="Tela de Expansao de Cenario" url="Tela de Expansao de
Cenario.html"/>

<mapID target="index" url="index.html"/>

<mapID target="Principal" url="Principal.html"/>

<mapID target="Tela de Gerenciamento de Itens" url="Tela de Gerenciamento
de Itens.html"/>

<mapID target="Tela Principal" url="Tela Principal.html"/>

</map>
```

Quadro 15 – Estrutura do arquivo `Map.jhm`

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<toc version="1.0">
 <tocitem text="Tela Principal " target="Tela Principal"/>

 <tocitem text="Interface para Expans&#227;o de Cen&#225;rio " target="Tela
de Expansao de Cenario"/>

 <tocitem text="Interface para Gerenciamento de Pe&#231;as " target="Tela
de Gerenciamento de Itens"/>

</toc>
```

Quadro 16 – Estrutura do arquivo `MinhaAjudaTOC.xml`

4.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO

O Editor de Malhas Ferroviárias apresenta um ambiente interativo em 3D, possibilitando que o usuário possa escolher a melhor forma de visualizar o cenário. A Figura 27 ilustra a interface principal da ferramenta.

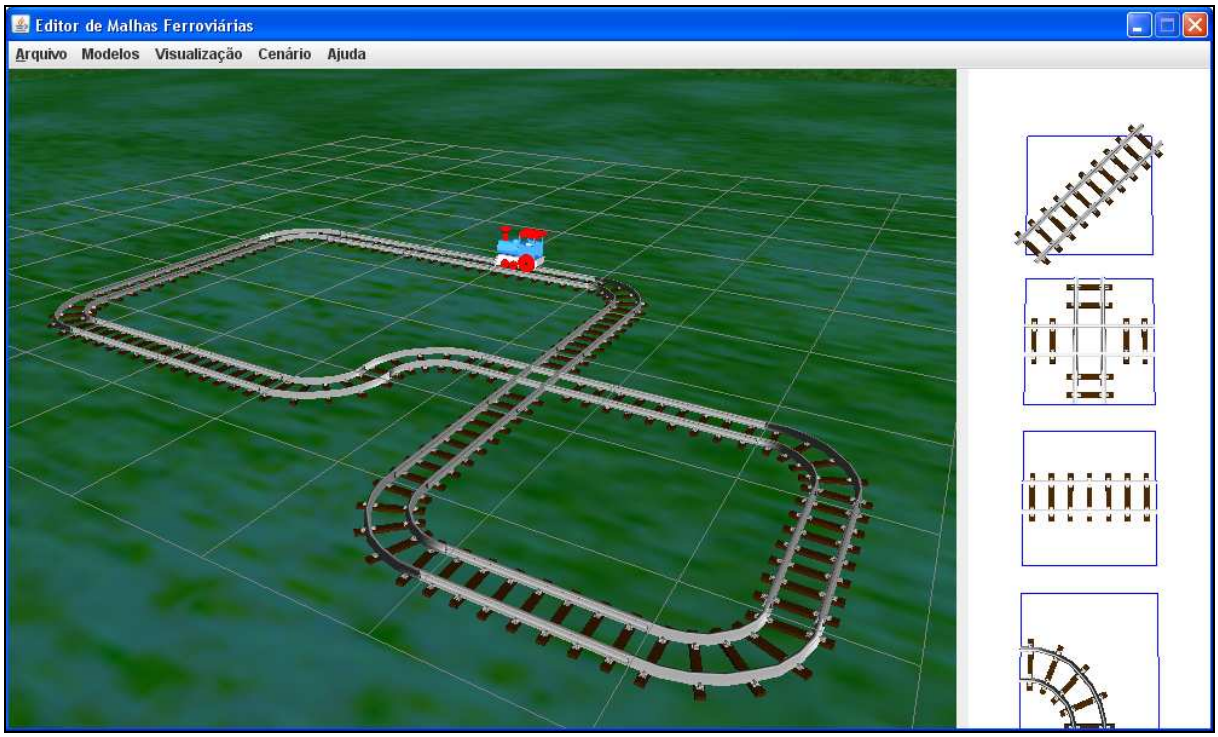


Figura 27 – Interface do EMF

Os principais componentes do ambiente do editor de malhas ferroviárias são: barra de menus (`Arquivo`, `Modelos`, `Visualização`, `Cenário` e `Ajuda`), *popup* menu, tela principal do cenário, tela lateral de itens, interface para expansão do cenário, interface para gerenciamento de itens de cenário e a ajuda da ferramenta.

4.4.1.1 Barra de menus

Com a barra de menus é possível utilizar funções da ferramenta. As funções são divididas em cinco partes: `Arquivo`, `Modelos`, `Visualização`, `Cenário` e `Ajuda`.

As opções e suas subdivisões da barra de menus são:

a) `Arquivo`: possui as seguintes sub-opções:
   - `Novo Cenário`: permite a criação de um novo cenário,
   - `Abrir Cenário`: permite recuperar um cenário salvo anteriormente,
   - `Salvar Cenário`: permite ao usuário salvar o cenário criado,
   - `Salvar Como`: permite ao usuário salvar o cenário em um local diferente do salvo anteriormente,
   - `Sair`: permite sair do sistema;

b) `Modelos`: possui a sub-opção `Gerenciar Modelos`, a qual permite a inclusão,

exclusão e alteração de peças de cenário, bem como cadastramento das rotas de caminhamento destas peças;

c) `Visualização`: possui as seguintes sub-opções:
   - `Exibir / Ocultar Eixos`: faz com que sejam exibidos ou ocultados os eixos x, y e z a partir do ponto central do cenário,
   - `Exibir / Ocultar Linhas`: faz com que seja exibida ou ocultada a grade de auxílio no chão do cenário,
   - `Visualizar Caminhos / ligações`: visualiza o grafo de caminhamento do cenário,
   - `Iniciar Caminhamento Trem`: exibe o trem em um trilho existente no cenário e faz com que o mesmo comece a andar sobre os mesmos,
   - `Parar Caminhamento Trem`: faz o trem parar de andar e oculta o mesmo;

d) `Cenário`: possui a sub-opção `Expandir Cenário`, a qual possibilita que a área para desenhos seja aumentada ou diminuída;

e) `Ajuda`: possui a sub-opção `Itens de Ajuda`, a qual exibe os tópicos de ajuda da ferramenta.

4.4.1.2 *Popup* menu

No EMF o *popup* `menu` é um menu de opções exibido quando se clica com o botão direito do *mouse* nas telas da ferramenta. Na tela principal da ferramenta o *popup* é utilizado para rotacionar e apagar os itens do cenário, bastando que o usuário selecione a opção desejada no menu exibido (Figura 28).

Figura 28 – *Popup* menu utilizado na tela principal do EMF

Na barra lateral de itens o *popup* menu é utilizado para rotacionar as peças (Figura 29) e na tela do gerenciador de itens de cenário é usado para editar e excluir peças (Figura 30).

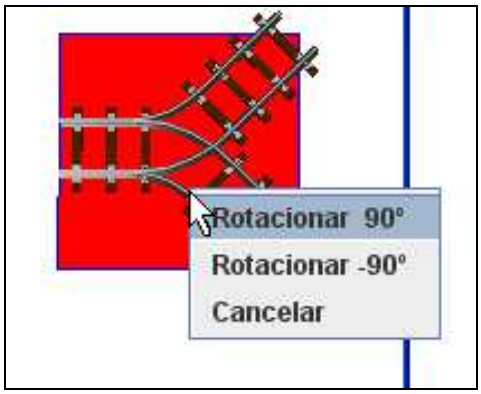


Figura 29 – *Popup* menu utilizado na tela lateral de itens do EMF

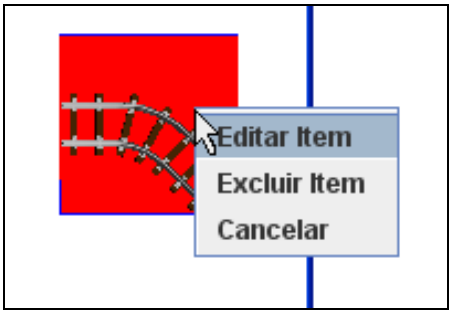


Figura 30 – *Popup* menu utilizado na tela do gerenciador de itens do EMF

4.4.1.3 Tela principal do cenário

Na tela principal do cenário (Figura 31) são exibidos o ambiente juntamente com os itens incluídos pelo usuário. Nesta tela é possível utilizar as setas do teclado para movimentar-se pelo cenário. As teclas `page up` e `page down` servem para aproximar ou afastar a câmera. Pressionando a tecla `<alt>` e a seta esquerda ou direita simultaneamente é possível rotacionar a câmera.

Para adicionar, remover, translacionar e rotacionar peças no cenário utiliza-se o *mouse*. Para adicionar peças ao cenário deve-se selecionar uma peça na tela lateral de itens e clicar com o botão esquerdo do *mouse* sobre uma posição disponível do cenário. Para excluir ou rotacionar peças do cenário basta clicar com o botão direito do *mouse* sobre a peça desejada e escolher a opção “apagar item” ou “rotacionar item” no *popup* menu exibido. Para translacionar (mover) uma peça do cenário basta clicar com o botão esquerdo do *mouse* sobre a peça desejada e arrastá-la até uma posição vazia.

Os eixos podem ser visualizados ou ocultados utilizando o item `Visualizar / Ocultar Eixos` disponível na opção `Visualização` da barra de menus. A grade de apoio pode ser visualizada ou ocultada utilizando o item `Visualizar / Ocultar Grade` disponível

na opção `Visualização` da barra de menus.

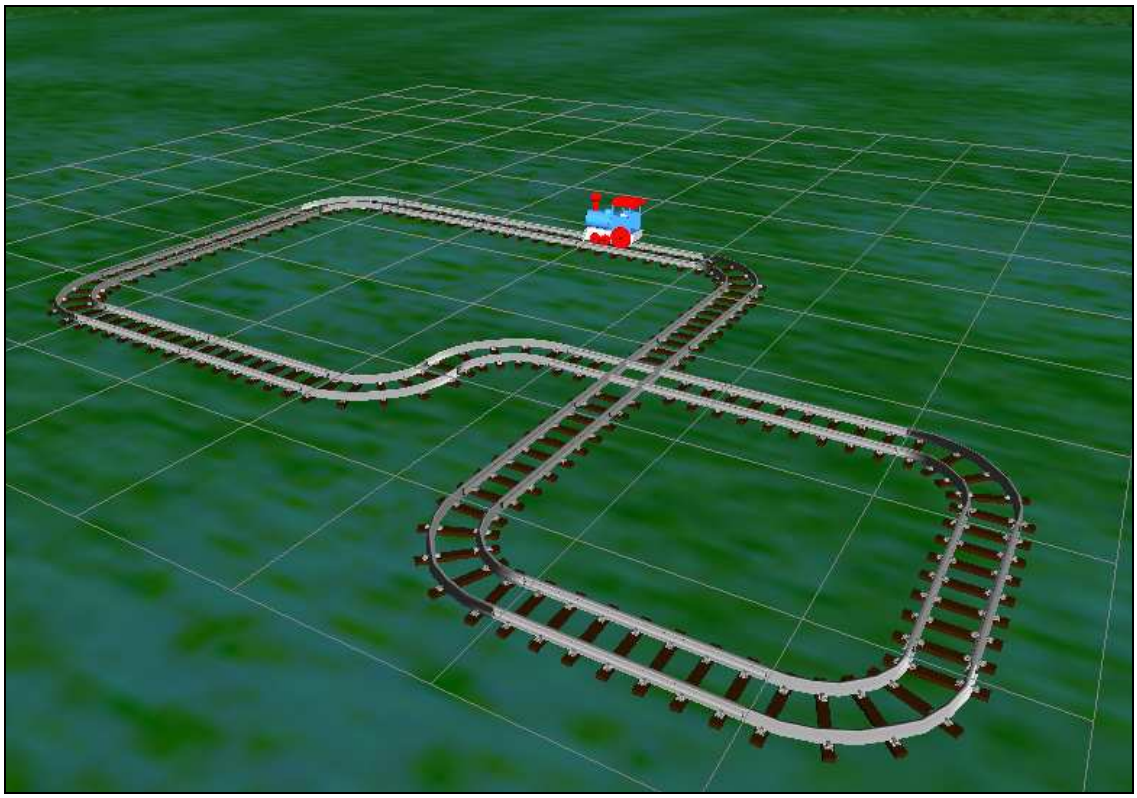


Figura 31 – Tela principal do EMF

4.4.1.4 Tela lateral de itens

A tela lateral de itens (Figura 32) é exibida à direita da tela principal e contém os itens de cenário já cadastrados pelo usuário. Para selecionar o item desejado basta clicar sobre o mesmo e para adicioná-lo ao cenário basta clicar novamente na posição desejada. Para rotacioná-lo basta clicar com o botão direito do *mouse* sobre o item de cenário e escolher a opção desejada. Ainda pode-se utilizar a rolagem do botão direito do *mouse* para subir e descer a tela, conseguindo assim localizar o item desejado.

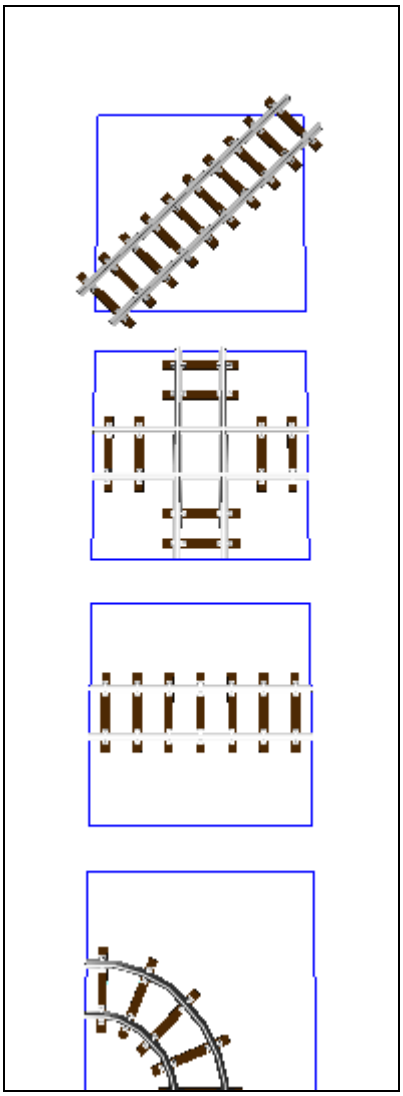


Figura 32 – Tela lateral de exibição de peças do EMF

4.4.1.5 Expansão de cenário

A expansão de cenário permite ao usuário expandir a área útil do mesmo, sendo essa a parte onde podem ser incluídas as peças. Esta opção pode ser acessada através da barra de menus, item `Cenário` e em seguida `Expandir Cenário`.

Esta opção permite expandir apenas um ou mais lados do cenário, bastando informar a quantidade de posições a ser aumentado para cada lado. A interface para expansão de cenário é apresentada na Figura 33.

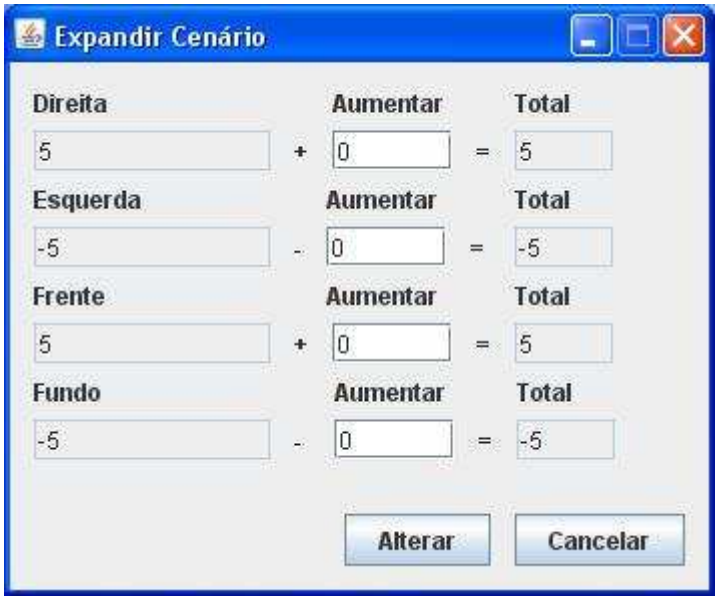


Figura 33 – Interface para expansão de cenário do EMF

4.4.1.6 Gerenciador de itens de cenário

O gerenciador de itens de cenário permite a inclusão, exclusão e alteração de itens no cenário. Para incluir um novo item o usuário deve primeiramente informar o local que está localizado o arquivo `wavefront obj`. Após é possível alterar a altura base, a rotação, escala e as rotas de caminhamento do objeto, sendo que as rotas de caminhamento são utilizadas pelo trem para identificar o caminho a ser percorrido em cada peça. Caso a peça não seja componente de ferrovia e não possua rotas de caminhamento, uma árvore, por exemplo, o usuário deve desmarcar a opção `A peça é trilho` para que o trem identifique que esta peça não é um trilho. Essa opção ainda pode ser usada quando se quer anular as rotas atuais para cadastrar outras, bastando para isso desmarcar e marcar novamente a opção `A peça é trilho`. Para concluir o cadastramento basta clicar no botão `Salvar Peça`.

Para excluir peças cadastradas basta clicar com o botão direito do *mouse* sobre a peça desejada na tela principal e selecionar a opção `Excluir peça` no *popup* menu exibido.

Para editar uma peça basta clicar com o botão direito do *mouse* sobre a mesma na tela principal e selecionar a opção `Editar peça` no *popup* menu exibido. É possível alterar a altura da peça em relação ao chão, a rotação, escala e as rotas de caminhamento. Para concluir a edição clica-se no botão `Salvar Peça`.

O gerenciador de itens de cenário (Figura 34) é acessado através do item `Gerenciar Modelos` do menu `Modelos` a partir do menu principal.

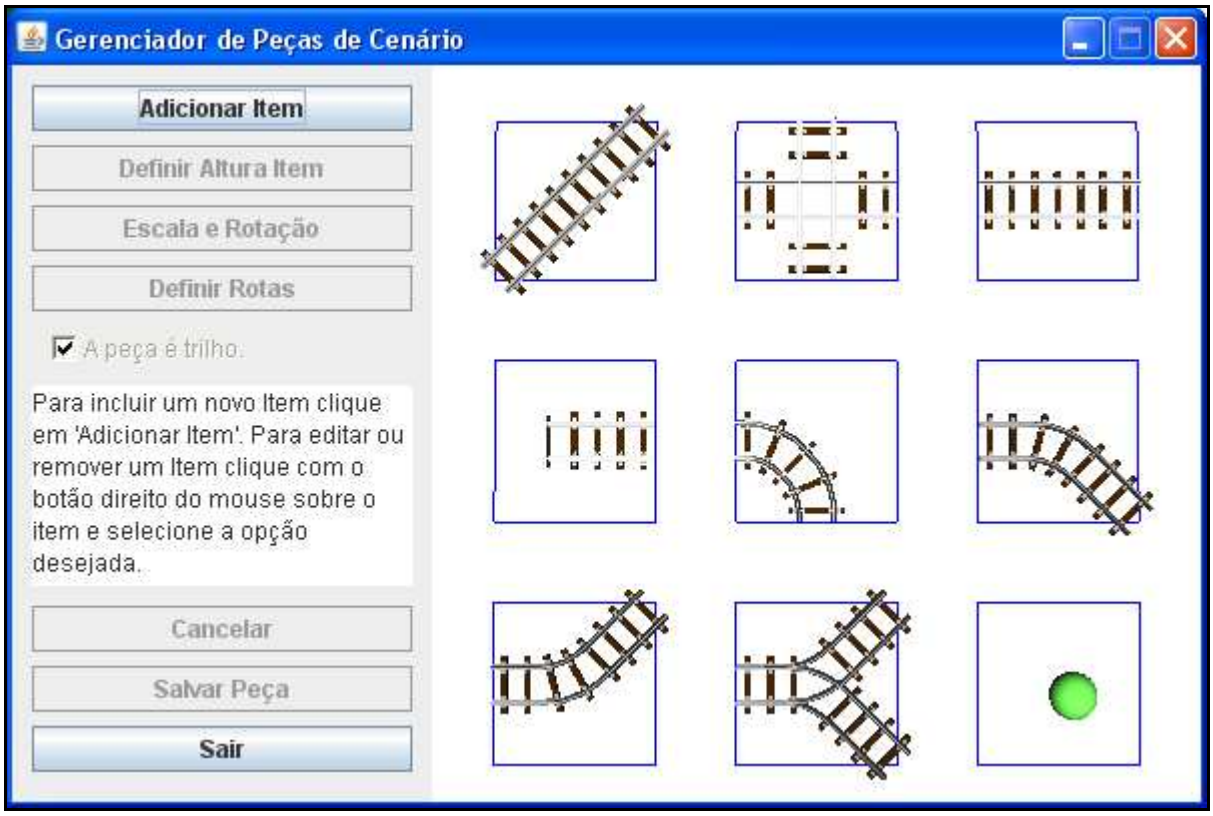


Figura 34 – Tela do gerenciador de itens do EMF

4.4.1.7 Ajuda da ferramenta

A ajuda do EMF (Figura 35) é acessada através do item `Itens de Ajuda` do menu `Ajuda` no menu principal. A ajuda contém as instruções para a utilização do EMF. Permite ao usuário realizar buscas nos conteúdos da ajuda e imprimi-las.

Ao realizar uma busca, o sistema apresenta as páginas que possuem a expressão procurada e destacam as ocorrências da expressão no texto.

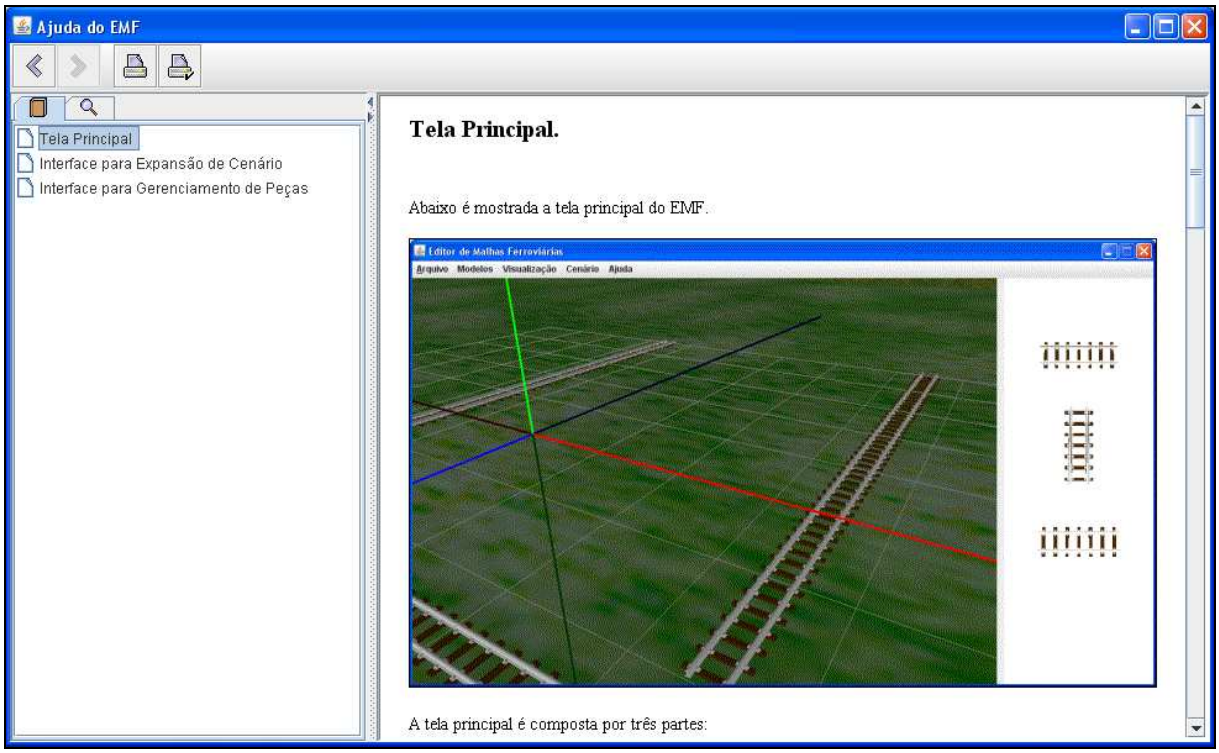


Figura 35 – Interface da ajuda do EMF

4.5 RESULTADOS E DISCUSSÃO

Durante a implementação, uma das dificuldades encontradas foi a construção da rotina de seleção de objetos em uma ambiente 3D. A solução encontrada para o problema foi nomear os objetos do cenário utilizando as funções `glPushName` e `glPopName` do JoGL. Desta forma, dividiu-se o chão do cenário em diversas partes e nomeou-se cada uma delas. Ao clicar com o *mouse*, em uma das partes nomeadas, é feita uma pesquisa e então obtido o número associado. Para criar o nome de cada parte foi utilizada uma função matemática que se baseia nos parâmetros x e y de cada parte do cenário. Para recuperar o nome utilizou-se a função reversa, sendo assim desnecessário guardar o nome de cada parte do cenário. Uma restrição deste método é que ele traz uma limitação à ferramenta, podendo utilizar-se no máximo um cenário com mil posições de largura e mil posições de comprimento, totalizando um milhão de posições, como também limita a usar um espaço discreto (não continuo), no caso uma grade regular retangular (GRR), o que pode dificultar sua utilização em terrenos irregulares (não planos, variações de elevação).

Por sua vez, as peças (trilhos e trem) utilizadas no EMF foram salvas no formato

`wavefront obj`. Para que estas peças ficassem com uma boa aparência e realistas necessitou-se dar um maior nível de detalhes para elas. Com o nível de detalhes utilizados as peças ficaram com grandes quantidades de vértices, tendo em média quinze mil vértices e chegando ao tamanho médio de 1,5 mega bytes cada uma.

Para melhorar o desempenho da aplicação utilizou-se os `display lists`, que segundo Jouvie (2007) faz a aplicação obter um melhor desempenho. No Quadro 17 é mostrado uma comparação da execução de uma cena sendo executada com a utilização de `display lists` e a utilização do modo tradicional. A comparação é feita utilizando a contagem de *Frames* Por Segundo (FPS) da aplicação de Jouvie durante sua execução. Estas comparações mostram a eficiência da utilização de `display lists` quando se deseja renderizar uma quantidade elevada de vértices.

Pirâmides com 240 vértices cada e com iluminação desabilitada:								
Número de pirâmides	4	12	20	36	100	200	500	1000
FPS com display lists	980	812	742	668	478	312	152	83
FPS no modo normal	747	486	356	233	100	52	21	10
% ganho com DL	31,2	67,1	108,4	186,7	378,0	500,0	623,8	730,0
Pirâmides com 240 vértices cada e com iluminação habilitada:								
Número de pirâmides	4	12	20	36	100	200	500	1000
FPS com display lists	908	798	694	564	322	194	88	46
FPS no modo normal	712	450	330	214	90	48	19	9
% ganho com DL	27,5	77,3	110,3	163,6	257,8	304,2	363,2	411,1
Cubos com 24 vértices cada e com iluminação desabilitada:								
Número de Cubos	1	2	3	4				
FPS com display lists	1290	1254	1190	1176				
FPS no modo normal	1172	1052	880	812				
% ganho com DL	10,1	19,2	35,2	44,8				
Cubos com 24 vértices cada e com iluminação habilitada:								
Número de Cubos	1	2	3	4				
FPS com display lists	984	983	980	978				
FPS no modo normal	984	974	964	954				
% ganho com DL	0	0,9	1,7	2,5				

Fonte: adaptado de Jouvie (2007).

Quadro 17 – Comparação de utilização e não utilização de `display lists`

Mesmo utilizando `display lists`, o EMF pode se tornar lento, principalmente em microcomputadores com pouca memória e pouca capacidade de processamento. Recomenda-se a utilização de um microcomputador com processador igual ou superior a 2 giga hertz e com memória *Random Access Memory* (RAM) igual ou superior a 1 giga bytes.

5 CONCLUSÕES

O objetivo de desenvolver um editor de malhas ferroviárias foi concluído com sucesso. Foi desenvolvido um editor com características diferentes dos editores vistos em Bertholdi (2004), Froeschlin (2006) e Perondi (2007), onde é possível interagir com o ambiente totalmente em 3D.

O editor possui algumas limitações, como a quantidade de posições de cenário, a qual é restringida a um milhão de posições, e a exigência de um microcomputador com processador igual ou superior a 2 giga hertz e com memória RAM igual ou superior a 1 giga bytes, visto o editor utiliza modelos vetoriais com grandes quantidades de vértices (quinze mil em média).

Apesar das limitações, os requisitos propostos foram alcançados. O editor é capaz de manipular arquivos vetoriais `wavefront obj`, possuindo um ambiente para manipulação em 3D. Operações para alteração na posição da câmera e nas peças (como rotação e translação) também são disponibilizados pelo EMF.

5.1 EXTENSÕES

Existem pontos que podem ser agregados ou melhorados no EMF. Como sugestão pode-se citar:

a) a criação de semáforos, para controle de tráfego nos cruzamentos;
b) a criação de ruas e o cruzamento das mesmas com as ferrovias;
c) a criação de controle e monitoramento de trens;
d) a possibilidade de translacionar e rotacionar duas ou mais peças ao mesmo tempo;
e) a confecção de novos tipos de peças (cruzamentos, curvas e retas);
f) possibilitar a inserção de mais de um trem e a configuração do local inicial dos mesmos;
g) a utilização de *splines* (curvas definidas matematicamente por dois ou mais pontos de controle) nas rotas;
h) a utilização de FPS para medição de desempenho.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, Joel Bortolotto de. Um ambiente de simulação e modelagem interativa visual orientado a objetos para avaliação de sistemas de controle de tráfego ferroviário. In: SEMANA ACADÊMICA DO CPGCC, 3., 1998, Porto Alegre. **Anais eletrônicos...** Porto Alegre: UFRGS – Programa de Pós-Graduação, [1998?]. Não paginado. Disponível em: <http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana98/joel.html>. Acesso em: 19 set. 2007.

AMÉRICA LATINA LOGÍSTICA. **Gestão voltada para segurança**. Curitiba, [2005?]. Disponível em: <http://www.fae.edu/publicacoes/pdf/revista_fae_business/n13/all.pdf>. Acesso em: 29 abr. 2008.

AUTODESK. **Autodesk 3ds Max**. [S.l.], 2007. Disponível em: <http://www.autodesk.com/fo-products-3dsmax>. Acesso em: 20 set. 2007.

BERTHOLDI, Geferson. **Editor gráfico de ruas para o sistema de controle de tráfego de automóveis em uma malha rodoviária urbana**. 2004. 55 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BORGES, Roberto Cabral de Mello. **Interface gráfica.** Porto Alegre. 2008. Disponível em: <http://www.inf.ufrgs.br/~cabral/Interf.Graf.Mai.2005.ppt>. Acesso em: 13 maio 2008.

BRITO, Allan. **Blender 3D**: guia do usuário. 2. ed. São Paulo: Novatec, 2007.

DAVISON, Andrew. **Pro Java™ 6 3D game development:** Java 3D™, JoGL, Jimput and JOAL APIs. Berheley: Apress, 2007.

FOTON. **Criação de interfaces gráficas**. [S.l.], 2008. Disponível em: <http://www.foton.com.br/midias_digitais/interfaces.html>. Acesso em: 15 abr. 2008.

FROESCHLIN, Gustavo Eduardo Grahl. **Editor gráfico de ruas para o sistema de controle de tráfego de automóveis em uma malha rodoviária urbana:** versão 2.0. 2006. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GUIA DO USUÁRIO DO CONECTIVA LINUX. **Interfaces gráficas.** [S.l.], 2008. Disponível em: <http://www.conectiva.com/doc/livros/online/9.0/usuario/interfaces.html>. Acesso em: 13 maio 2008.

JANCZURA, Chris W. **Modelling and analysis of railway network control logic using coloured Petri nets**. 1998. 246 f. Thesis (Doctor of Philosophy in Mathematics) - School of Mathematics and Institute for Telecommunications Research, University of South Australia, Australia. Disponível em: <http://citeseer.ist.psu.edu/cache/papers/cs/19121/http:zSzzSzwww.itr.unisa.edu.auzSztech_reszSzpublicationszSzcwj.pdf/janczura98modelling.pdf>. Acesso em: 29 abr. 2008.

JAVA.NET. **JoGL API project**. [S.l.], 2007. Disponível em: <https://jogl.dev.java.net/>. Acesso em: 25 set. 2007.

_____. **JavaHelp System**. [S.l.], 2008. Disponível em: <https://javahelp.dev.java.net/>. Acesso em: 13 jun. 2008.

JOUVIE, Jérôme. **Tutorial 16:** display lists. [S.l.], 2007. Disponível em: <http://jerome.jouvie.free.fr/>. Acesso em: 10 maio 2008.

JUHNKE, Klaus Jurgen. **A eficiência das ferrovias no transporte metropolitano**. São Paulo: Edgard Blucher; São Paulo: EDUSP, 1968. (Pesquisas científicas de transportes).

MICROSOFT. **Microsoft Office**. [S.l.], 2008. Disponível em: <http://office.microsoft.com/>. Acesso em: 28 abr. 2008.

PERONDI, Paulo Roberto. **Editor gráfico de ruas para o sistema de controle de tráfego de automóveis em uma malha rodoviária urbana:** versão 3.0. 2007. 54 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

RASTER. In: WIKIPEDIA, a enciclopédia livre. [S.l.]: Wikipedia Foundation, 2008. Disponível em: < http://pt.wikipedia.org/wiki/Raster>. Acesso em: 14 jul. 2008.

REDDY, Martin. **Object files (.obj)**. [S.l.], 2008. Disponível em: <http://www.martinreddy.net/gfx/3d/OBJ.spec>. Acesso em: 20 abr. 2008.

SARDO, Andrey Starke. **Controle de tráfego ferroviário utilizando microcontrolador pic16f628a**. 2007. 68 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SCHUBERT, Lucas Andreas. **Aplicativo para controle de ferrovia utilizando processamento em tempo real e redes de petri**. 2003. 76 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

TURBO SQUID. **3D models**. [S.l.], 2008. Disponível em: <http://www.turbosquid.com/3d/>. Acesso em: 15 maio 2008.

VILAÇA, Rodrigo. Infra-estrutura de transportes: cenário promissor? **Associação Nacional dos Transportes Ferroviários**, Brasília, abr. 2007. Artigos. Disponível em: <http://www.antf.org.br/cgi-bin/PageSvr.dll/Get?id_doc=2571>. Acesso em: 23 set. 2007.