UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

EDITOR GRÁFICO DE RUAS PARA O SISTEMA DE CONTROLE DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA URBANA: VERSÃO 3.0

PAULO ROBERTO PERONDI

BLUMENAU
2007

2007/1-35

PAULO ROBERTO PERONDI

EDITOR GRÁFICO DE RUAS PARA O SISTEMA DE CONTROLE DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA URBANA: VERSÃO 3.0

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. Paulo César Rodacki Gomes, Doutor – Orientador.

BLUMENAU
2007

2007/1-35

EDITOR GRÁFICO DE RUAS PARA O SISTEMA DE CONTROLE DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA URBANA: VERSÃO 3.0

Por

PAULO ROBERTO PERONDI

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

Presidente:	Prof. Paulo César Rodacki, Doutor – Orientador, FURB
Membro:	Prof. José Roque Voltolini da Silva – FURB
Membro:	Prof. Antônio Carlos Tavares – FURB

Blumenau, 11 de julho de 2007

Dedico este trabalho a minha família, ao orientador e em especial a minha mulher.

AGRADECIMENTOS

À minha mulher Anita, pelas cobranças e por estar sempre ao meu lado.

À minha família, Vanir, Doralice, Marcos e Marcelos, que sempre estiveram presentes, mesmo que distantes.

Ao meu amigo Andrey, pelos auxílios, pela força e pelos risos.

Ao meu orientador, Paulo César Rodacki Gomes, pela oportunidade e pela paciência.

A Empresa Baumgarten, pelo apoio financeiro.

A todas as pessoas que me incentivaram e acreditaram que eu chegaria até aqui.

A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original.

Albert Einstein

RESUMO

O Editor Gráfico de Malhas Rodoviárias (EGMR) é um software que tem como principal objetivo a criação e manutenção de malhas viárias, as quais tem a finalidade de servir como cenário para o Simulador de Tráfego Rodoviário desenvolvido por Freire (2004). Este editor foi desenvolvido por Bertoldi (2005) e continuado por Froeschlin (2006). O programa oferece ferramentas para a criação, edição e manutenção de malhas rodoviárias. Neste trabalho são apresentadas as alterações efetuadas no EGMR, com o intuito de torná-lo mais ágil e fácil. Entre as alterações, destacam-se a conversão do código para a linguagem Java, e conseqüentemente, a necessidade da utilização da biblioteca JoGL. Também foram realizadas modificações no formato do arquivo para a persistência da malha, que passou para XML. Além disso, para facilitar o desenho e manutenção de trechos sinuosos, foi inclusa uma ferramenta para o desenho de curvas de Bézier. Foi disponibilizada a possibilidade de visualização em 3D da malha criada, a fim de auxiliar o desenho de viadutos. Para uma melhor exatidão na representação de cenas reais, foi inclusa também a possibilidade de utilizar uma imagem de fundo.

Palavras-chave: Ciência da computação. JoGL. OpenGL. Editor gráfico. Curvas de Bézier.

ABSTRACT

The Road Meshes Graphical Editor (EGMR) is a software with the main objective the creation and maintenance of road meshes, which has the purpose to serve as scene for the Simulator of Road Traffic developed by Freire (2004). This publisher was developed by Bertoldi (2005) and continued by Froeschlin (2006). The program offers tools for the creation, edition and maintenance of road meshes. In this work the alterations performed in the EGMR are presented, with intention to become it more agile and easy. Among the alterations, they are distinguished it rewrite of the code for the Java language, and consequently, the necessity of the use of the JoGL library. Also modifications in the format of the archive for the persistence of the mesh had been carried through, that passed to XML. Moreover, to facilitate to the drawing and maintenance of winding stretches, a tool for the drawing of curves of Bézier was included. The possibility of visualization in 3D of the created mesh was available, in order to assist the drawing of viaducts. For one better exactness in the representation of real scenes, was included also the possibility of use of an image of deep.

Key-words: Computer science. JoGL. OpenGL. Graphic editor. Bézier curves.

LISTA DE ILUSTRAÇÕES

Quadro 1 – Exemplo de método `display()` da interface `GLEventListener` em JoGL.... 16
Quadro 2 - Exemplo de documento XML 17
Quadro 3 – Utilização de XStream para a geração de documentos XML com Java..... 18
Quadro 4 – Resultado do programa apresentado no quadro 3..... 19
Figura 1 – Curva de Bézier..... 20
Quadro 5 – Implementação de curva de Bézier..... 20
Figura 2 – Curva de Bézier com mais (esquerda) e menos (direita) pontos..... 21
Figura 3 – Grafo direcionado..... 21
Figura 4 – Grafo não-direcionado 22
Figura 5 – Simulador do tráfego de automóveis em uma malha rodoviária..... 23
Figura 6 – Tela principal da versão 1.0 do EGMR e suas funções..... 24
Figura 7 – Representando um trecho criado..... 25
Figura 8 – Tela principal do EGMR 2.0..... 26
Figura 9 – Trecho com um semáforo..... 27
Figura 10 – Trechos selecionados 27
Quadro 6 – Relação das funcionalidades e seus atalhos..... 28
Figura 11 – Tela para manutenção de ruas da malha 29
Figura 12 – TransModeler..... 30
Figura 13 – Módulos do SINCMobil..... 30
Figura 14 – Diagrama de casos de uso 33
Figura 15 – Diagrama de atividades..... 34
Figura 16 – Diagrama de classes..... 35
Figura 17 - Diagrama seqüência para criação de um trecho..... 37
Figura 18 - Diagrama de seqüência para inserção de uma imagem de fundo 38
Figura 19 – Tela Principal 39
Figura 20 – Funções dos botões da barra de ferramentas do EGMR 3.0 40
Figura 21 – Desenho tendo como fundo uma imagem..... 42
Figura 22 – Trecho criado com a ferramenta Bézier..... 43
Quadro 7 - Cálculo do pontos intermediários das curvas de Bézier..... 43
Figura 23 – Visualização da malha em 3D..... 44
Quadro 8 - Código para a visualização em 3D..... 44

Figura 24 – Seleção de um trecho para edição 45
Figura 25 – Ferramenta de edição de trechos 46
Figura 26 – Estrutura dos trechos 46
Figura 27 – Movimentação de um trecho 47
Figura 28 – EGMR no Sistema Operacional Linux 48
Figura 29 – EGMR no Sistema Operacional MacOS 48
Figura 30 – EGMR no Sistema Operacional Windows 49
Quadro 9 - Comentários especiais para a geração do JavaDoc 49
Figura 31 – Documentação HTML da classe Ponto 50
Quadro 10 - Comparação entre versões do EGMR 51

LISTA DE SIGLAS

API – *Application Programming Interface*

AWT - *Abstract Window Toolkit*

EGMR– Editor Gráfico de Malhas Rodoviárias

GUI – *Graphical User Interface*

HTML – *Hyper Text Markup Language*

IDE – *Integrated Development Environment*

JoGL – *Java Bindings for OpenGL*

OpenGL – *Open Graphics Library*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS DO TRABALHO 14
1.2 ESTRUTURA DO TRABALHO 14
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 JOGL 15
2.1.1 GLJPanel e GLCanvas 15
2.1.2 GLEventListener 15
2.2 XML 16
2.2.1 XStream 17
2.3 EDITORES GRÁFICOS 19
2.3.1 Curvas de Bézier 20
2.4 GRAFOS 21
2.4.1 Grafos direcionados 21
2.4.2 Grafos não-direcionados 22
2.5 SIMULAÇÃO DO CONTROLE DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA URBANA 22
2.6 EGMR VERSÃO 1.0 23
2.7 EGMR VERSÃO 2.0 25
2.7.1 Tela principal 26
2.7.2 Semáforos 26
2.7.3 Identificação de trechos 27
2.7.4 Agilidade na criação de Malhas 27
2.7.5 Ruas 28
2.8 TRANSMODELER 29
2.9 SINCMOBIL 30
3 DESENVOLVIMENTO DO PROTÓTIPO 32
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 32
3.2 ESPECIFICAÇÃO 32
3.2.1 Diagrama de casos de uso 32
3.2.2 Diagrama de atividades 33
3.2.3 Diagrama de classes 35

3.2.4 Diagrama de Seqüência 37
3.3 IMPLEMENTAÇÃO 38
3.3.1 Técnicas e ferramentas utilizadas 38
3.3.2 Operacionalidade 39
3.3.3 Conversão de código 40
3.3.4 Desenho de viadutos 41
3.3.5 Imagem de fundo 41
3.3.6 Ferramenta de Bézier 42
3.3.7 Visualização da malha em 3D 43
3.3.8 Editar Trechos 45
3.3.9 Grafos 46
3.3.10 Multiplataforma 47
3.3.11 Documentação 49
3.4 RESULTADOS E DISCUSSÃO 50
4 CONCLUSÕES 52
4.1 EXTENSÕES 53
REFERÊNCIAS BIBLIOGRÁFICAS 54

1 INTRODUÇÃO

A tecnologia avança rapidamente, trazendo consigo as facilidades do mundo moderno. Entretanto, quando em demasia, pode dar origem a problemas bastante conhecidos do cotidiano. Um exemplo disso é o constante aumento do número de veículos automotivos registrado nas estradas, principalmente no perímetro urbano das grandes cidades. Este aumento é resultado, em parte, da grande oferta de mercado, o baixo custo dos veículos e as condições facilitadas oferecidas aos consumidores. Um agravante desta situação é o mau planejamento das vias urbanas, o que contribui para os engarrafamentos e altos índices de poluição sonora e ambiental.

Conforme crescem estes números, surge a necessidade de planejamento urbano e das suas vias. Este tende a ser mais eficaz quando utiliza ferramentas que permitam o controle e visualização das variáveis envolvidas. O uso da informática nestas situações já é bastante comum e também tem se desenvolvido neste segmento, como pode ser verificado em Caliper Corporation (2006).

Considerando estes fatos, Freire (2004) desenvolveu um simulador de tráfego viário, que realiza simulações do trânsito em relação a uma malha previamente definida. Esta, por sua vez, é representada na forma de um arquivo texto contendo as coordenadas das vias. Em decorrência a esta situação, surgiu a necessidade do desenvolvimento de um Editor Gráfico de Malhas Rodoviárias (EGMR) a fim de facilitar a criação dos mapas a serem simulados.

Bertoldi (2005) apresenta a versão 1.0 deste editor e Froeschlin (2006) descreve a versão 2.0 com algumas melhorias e novas funcionalidades. Este editor proporciona um desenho mais fácil e intuitivo das vias e, posteriormente, salva a malha desenvolvida em um arquivo texto, seguindo o formato definido por Freire (2004).

A ampla gama de possibilidades oferecidas neste contexto permite a continuidade dos trabalhos a fim de adicionar novas funcionalidades ao software em questão, facilitando a representação de cenários do mudo real.

Este trabalho apresenta o desenvolvimento da terceira versão do EGMR, demonstrando as novas características e ferramentas incorporadas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é dar continuidade ao EGMR, convertendo o código fonte, que hoje encontra-se em Delphi, para Java. Além disso, adicionar novas funcionalidades ao projeto, com o intuito de facilitar a criação de malhas mais complexas, que representarão com mais facilidade o mundo real.

Os objetivos específicos do trabalho são:

a) converter o código existente para a linguagem Java;
b) incluir funcionalidades para o desenho de viadutos, possibilitando a passagem de uma rua sob a outra;
c) adicionar uma ferramenta de desenho de curvas, utilizando as curvas de Bézier;
d) converter o atual formato do arquivo gerado para persistir o desenho em XML;
e) adicionar a possibilidade do desenho ser realizado tendo como fundo uma imagem de satélite ou aerofotogrametria;
f) oferecer a possibilidade de visualização da malha em 3D.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está divido em quatro capítulos. Inicialmente realiza uma introdução ao contexto do trabalho e demonstra os objetivos principais. Em seguida, traz uma fundamentação teórica dos conhecimentos necessários para a realização do trabalho, bem como das principais ferramentas e técnicas utilizadas. Depois é apresentada a especificação e a forma como foi realizado o desenvolvimento. Por fim, o quarto e último capítulo apresenta as conclusões e resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

No presente capítulo são abordados os principais conceitos referentes a realização do trabalho e as ferramentas e tecnologias utilizadas.

2.1 JOGL

Em Java.net (2006), JoGL é descrita como uma API que faz a ligação entre a linguagem Java e a OpenGL. Segundo Cohen e Manssour (2006, p. 364) existem várias soluções com este propósito, porém o projeto JoGL é a implementação de referência neste seguimento. A API fornece meios para que as rotinas da OpenGL possam ser facilmente executadas juntamente com componentes da interface com usuário do ambiente Java.

A seguir são comentadas algumas das principais características e ferramentas disponíveis nesta biblioteca. Apesar de ser multiplataforma, existem bibliotecas específicas para cada sistema operacional, disponíveis no *site* Java.net (2006), onde também são apresentadas os requisitos para a execução dos programas que utilizam JoGL nas principais plataformas.

2.1.1 GLJPanel e GLCanvas

GLJPanel e GLCanvas são duas classes que implementam um canvas OpenGL. Foram projetadas para fornecer compatibilidade com as duas bibliotecas para confecção de GUIs de Java, Swing e AWT. Entretanto, apesar de ser 100% compatível com Swing, o GLJPanel ainda não oferece aceleração por hardware.

2.1.2 GLEventListener

`GLEventListener` é uma interface que define 4 métodos que devem ser

implementados: `display()`, `reshape()`, `init()`, `displayChange()`. Deve ser atribuído a um `GLCanvas` ou `GLJPanel`.

O método `display()` é responsável por redesenhar o canvas a que está associado. É nele que devem ser chamadas as rotinas de desenho de OpenGL. `Reshape()`, por sua vez, é invocado quando o tamanho do canvas sofre alguma alteração. `Init()`é chamado quando o OpenGL é inicializado e `displayChanged()` ainda não está implementado. Este último seria invocado quando existisse algum tipo de alteração na forma de visualização do monitor, como quantidade de cores ou passar o canvas de um monitor para outro.

Os 4 métodos que devem ser implementados recebem como parâmetro um objeto `GLAutoDrawable`, de onde é extraído o objeto `GL`. A partir deste objeto são realizadas as operações OpenGL, como no exemplo do Quadro 1.

```
public void display(GLAutoDrawable drawable){
        //obter o objeto GL
        GL gl = drawable.getGL();

        //limpar a janela de visualização com
        //a cor de fundo previamente definida
        gl.glClear(gl.GL_COLOR_BUFFER_BIT);

        //Altera a cor de desenho para preto
        gl.glColor3f(0.0f,0.0f,0.0f);

        //Função para desenho de um cubo
        GLUT glut = new GLUT();
        glut.glutWireCube(50);

        //Excecuta os comandos OpenGL
        gl.glFlush();
}
```

Quadro 1 – Exemplo de método `display()` da interface `GLEventListener` em JoGL

2.2 XML

Veloso (2003, p. 5) define XML como uma metalinguagem de marcação que permite a criação de linguagens personalizadas, baseadas em marcações, seguindo regras definidas em um determinado contexto. Segundo ele, uma das principais vantagens de XML é promover a interoperabilidade ou troca de dados entre os diversos tipos de computadores ou sistemas. Isto é possível porque um arquivo XML é baseado em formato texto, o que também contribui para a leitura e o entendimento humano. Um exemplo de documento XML é apresentado no Quadro 2.

```
<?xml version = “1.0” encoding = “UTF-8”?>
<LIVRO>
   <TITULO>
      Java e XML
   </TITULO>
   <ASSUNTO>
      Processamento de documentos XML com Java
   </ASSUNTO>
</LIVRO>
```

Fonte: Veloso (2003, p.7).

Quadro 2 - Exemplo de documento XML

Existem diversas ferramentas para auxiliar a utilização de XML com a linguagem Java, entre elas destacam-se Dom e Sax, que são as mais conhecidas. Será apresentada a seguir a XStream, outra forma de utilização de XML em Java, que fornece meios mais simples para a manipulação destes documentos.

2.2.1 XStream

XStream é uma alternativa para facilitar o trabalho com documentos XML e Java.

> Existem diversas formas de se trabalhar com XML e Java, dentre elas, SAX, DOM, Digester. Cada uma serve ao seu propósito. Por exemplo, SAX é extremamente eficiente na leitura de arquivos. DOM, por sua vez oferece um melhor controle da estrutura hierárquica dos documentos XML, sendo mais adequado para a escrita de documentos para os quais não se tem total domínio da estrutura (esquema). Finalmente, Digester oferece uma forma interessante de mapear documentos XML para objetos Java e vice-versa. Entretanto, exige que o mapeamento seja manualmente configurado. (Amorim, 2007).

Amorim (2007) cita algumas vantagens da utilização de XStream:

a) facilidade de uso: geralmente só se precisa conhecer uma classe;
b) não é necessário configurar o mapeamento manualmente;
c) XML é legível e mais compacto que a serialização nativa de Java;
d) não exige que os objetos sigam alguma regra em particular;
e) suporta referências duplicadas e circulares dos objetos;
f) pode ser integrada com outras APIs;
g) a estratégia de conversão pode ser configurada;
h) realiza uma avaliação prévia se o documento XML não está mal-formado, exibindo eventuais mensagens de erros.

O Quadro 3 exemplifica a utilização de XStream. Primeiramente é criado um

`ArrayList` com alguns contatos (Objetos `Pessoa()`). Posteriormente este *array* é convertido para XML e vice-versa e o resultado é impresso na saída padrão.

```
import java.util.ArrayList;
import java.util.List;

import com.thoughtworks.xstream.XStream;

public class TesteXStream {
    public static void main(String[] args) {
        // Criando um objeto XStream
        XStream xstream = new XStream();

        // Criando alguns dados
        Pessoa vinci = new Pessoa();
        vinci.setNome("Vinci Pegoretti Amorim");
        vinci.setEmail("vinci_amorim@yahoo.com.br");

        Telefone foneDoVinci = new Telefone();
        foneDoVinci.setDdd(55);
        foneDoVinci.setNumero("5555 5555");

        vinci.setFoneComercial(foneDoVinci);
        List contatos = new ArrayList(1);
        contatos.add(vinci);

        // Passando os dados de Objetos Java para XML
        String contatosEmXML = xstream.toXML(contatos);

        System.out.println("\nContatos em XML:");
        System.out.println(contatosEmXML);

        // Passando os dados de XML para Objetos Java
        List amigos = (List) xstream.fromXML(contatosEmXML);
        Pessoa amigo = (Pessoa) amigos.get(0);
        Telefone foneDoAmigo = amigo.getFoneComercial();

        System.out.println("\nAmigo como Objeto Java:");
        System.out.println("Nome: " + amigo.getNome());
        System.out.println(
            "Fone Comercial: ("
                + foneDoAmigo.getDdd()
                + ") "
                + foneDoAmigo.getNumero());
    }
}
```

Fonte: Amorim (2007).

Quadro 3 – Utilização de XStream para a geração de documentos XML com Java

O programa do quadro 3 resulta no documento XML conforme o Quadro 4.

```
Contatos em XML:
<list>
<Pessoa>
<email>vinci_amorim@yahoo.com.br</email>
<foneComercial class="Telefone">
<ddd>55</ddd>
<numero>5555 5555</numero>
</foneComercial>
<nome>Vinci Pegoretti Amorim</nome>
</Pessoa>
</list>

Amigo como Objeto Java:
Nome: Vinci Pegoretti Amorim
Fone Comercial: (55) 5555 5555
```

Fonte: Amorim (2007).

Quadro 4 – Resultado do programa apresentado no quadro 3

2.3 EDITORES GRÁFICOS

Com a evolução da informática e do avanço no desenvolvimento de equipamentos destinados a produção de imagens digitais, a interface gráfica torna-se uma ferramenta indispensável, devido à facilidade com que permite ao usuário realizar tarefas praticamente impossíveis em sistemas modo texto.

Conforme Paula Filho (2000, p. 144), a interface gráfica constitui um fator importante do software, que deve oferecer ao usuário respostas rápidas e facilidade de uso, acelerando o processo de aprendizagem, tornando mais difícil a ocorrência de erros por parte do usuário e aumentando a produtividade. Diz ainda que os editores gráficos bidimensionais atingem muitas áreas da tecnologia, desempenhando um papel fundamental na criação de estruturas gráficas, permitindo diversas transformações e a sua correta visualização.

Da mesma forma, é notável a importância destes conceitos no desenvolvimento de um editor para a construção de malhas viárias, que exigem uma boa representação da realidade a qual o simulador de trânsito, neste caso, o desenvolvido por Freire (2004), destina-se. Isso reforça a idéia das ferramentas que o editor deve oferecer para representar a realidade.

A maioria dos editores de desenhos gráficos apresenta uma ferramenta chamada curva de Bézier, que tem a finalidade de auxiliar o desenho de curvas. Os conceitos que envolvem o desenvolvimento e geração de uma curva de Bézier são apresentados a seguir.

2.3.1 Curvas de Bézier

A curva de Bézier é uma estrutura gráfica composta basicamente por dois pontos de início e fim, e os pontos de controle, que combinados com os primeiros definem as tangentes à curva nas duas extremidades. Por definição, as curvas de Bézier podem ser geradas com infinitos pontos, porém, tratando-se de curvas para desenho e edição interativa, são utilizados 4 pontos para cada segmento: o inicial e final que pertencem a curva e mais dois de controles. Um exemplo de curva de Bézier é mostrado na Figura 1.

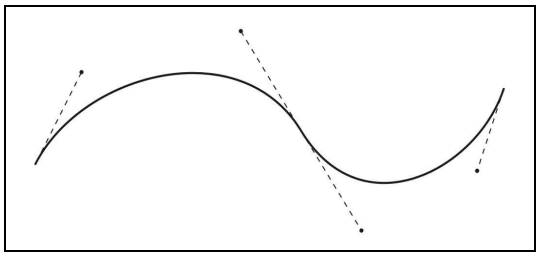


Figura 1 – Curva de Bézier

Os pontos são uma referência para a construção da curva, portanto, esta pode ser facilmente editada mudando a posição dos controles.

A união de várias curvas de Bézier é uma tarefa simples uma vez que os pontos de controlo final e inicial pertencem sempre à curva. Para que a curva não perca a sua "suavidade", na junção com a seguinte, basta que o segmento final do polígono de Bézier de uma e o segmento inicial da próxima, sejam colineares.

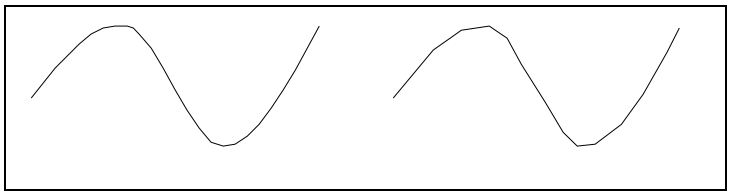
A JoGL, através da OpenGL, implementa diretamente este tipo de curva, através de funções que calculam valores intermediário, a partir de uma seqüência de pontos de controle. O Quadro 5 ilustra a criação de um seguimento de uma curva de Bézier utilizando JoGL.

```
//Função que passa um array com os pontos da curva
gl.glMap1d(gl.GL_MAP1_VERTEX_3,0.0,1.0,3,4,pontos,0);
//Ativa a geração de coordenadas
gl.glEnable(gl.GL_MAP1_VERTEX_3);
//Traça a curva
gl.glBegin(gl.GL_LINE_STRIP);
for(double f=0;f<=1.01;f+=0.01)
    gl.glEvalCoord1d(f);
gl.glEnd();
```

Fonte: adaptado de Cohen e Manssour (2006, p. 108).

Quadro 5 – Implementação de curva de Bézier

A função `gl.glEvalCoord1d()` é responsável por calcular os pontos intermediários da curva. Quanto maior a quantidade de pontos, mais suave é o desenho da curva, entretanto, aumenta também o tempo de processamento. Na Figura 2 é mostrada uma curva com mais e menos pontos intermediários.



Fonte: adaptado de Cohen e Manssour (2006, p. 112).
Figura 2 – Curva de Bézier com mais (esquerda) e menos (direita) pontos

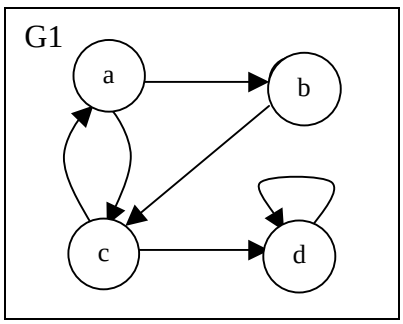
2.4 GRAFOS

Preiss (2001, p. 478) define um grafo como um conjunto de linhas que conectam vários pontos. Segundo ele, muitos problemas reais podem ser traduzidos a problemas em grafos. Existem vários tipos de grafos definidos pela área de estudos conhecida como Teoria dos Grafos. A seguir são apresentados dois tipos, grafos direcionados e grafos não-direcionados.

2.4.1 Grafos direcionados

Grafo direcionado é um par ordenado `G=(V,E)`. `V` é um conjunto finito não vazio e seus elementos são chamados de Vértices. `E` é um conjunto finito de pares ordenado de vértices. Seus elementos são chamados de arestas de `G`.

A Figura 3 mostra um exemplo de um grafo direcionado. Os vértices estão representados por círculos e as arestas estão representadas pelos arcos que conectam vértices associados. Os grafos também podem ser representados através de conjuntos matemáticos.



Fonte: Preiss (2006).
Figura 3 – Grafo direcionado

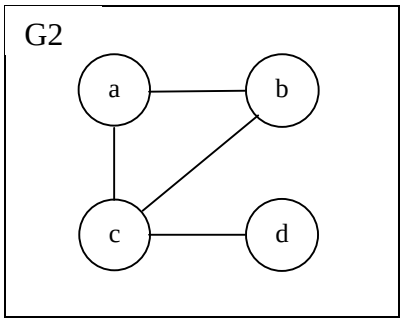
O grafo da Figura 3 pode ser representado através de dois conjuntos de pares

ordenados, o de vértices, `V={a,b,c,d}` e o de arestas, `E={(a, b), (a, c), (b, c), (c, a), (c, d), (d, d)}`. Como as arestas são ordenadas, os pares `(a, c)` e `(c, a)` são distintos e uma vez que `E` é um conjunto matemático, ele não pode conter mais de uma instância da mesma aresta. Além disso, uma aresta como `(d, d)` conecta um nó a ele mesmo (PREISS, 2006. p. 479).

2.4.2 Grafos não-direcionados

De acordo com Preiss (2006, p. 482), grafo não-direcionado é um grafo no qual os nós estão conectados por arcos não-direcionados, que são arestas que não possuem uma seta.

A Figura 4 mostra o grafo da Figura 3, como se este fosse não-direcionado.



Fonte: Preiss (2006).
Figura 4 – Grafo não-direcionado

Matematicamente, `V=(a, b, c, d)` e `E={{a, b},{a, c},{b, c},{c, d}}`. Em um grafo não direcionado, uma aresta é um conjunto, portanto `{a,b}` é equivalente a `{b,a}`. Como ele também é um conjunto, ele não pode conter mais de uma instância da mesma aresta.

2.5 SIMULAÇÃO DO CONTROLE DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA URBANA

Freire (2004) apresenta um protótipo para a simulação de tráfego em uma malha viária urbana. Este software lê um arquivo texto com as coordenadas que definem a malha e a partir de alguns dados fornecidos pelo usuário, simula o comportamento do tráfego na mesma, dando uma visão geral do trânsito sob aquelas condições. As funcionalidades do software são:

a) configuração da malha rodoviária;

b) configuração da quantidade de veículos que circulam sobre esta malha;

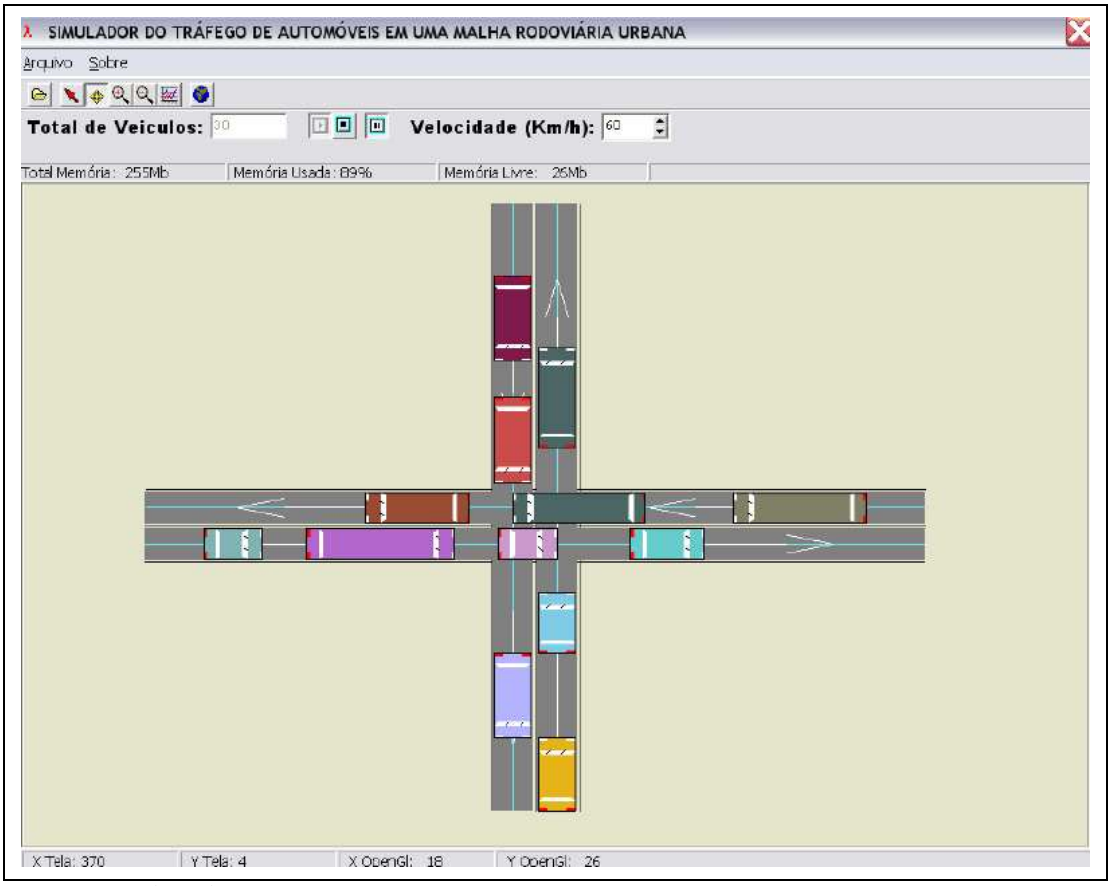
c) configuração da velocidade dos veículos;

d) efetua o cálculo da distância entre dois pontos sobre a representação gráfica;

e) oferece a visualização do mapa com os veículos em movimento;

f) controla o tráfego em uma malha rodoviária sem semáforos.

A Figura 5 demonstra uma simulação feita a partir do software.



Fonte: Freire (2004).

Figura 5 – Simulador do tráfego de automóveis em uma malha rodoviária

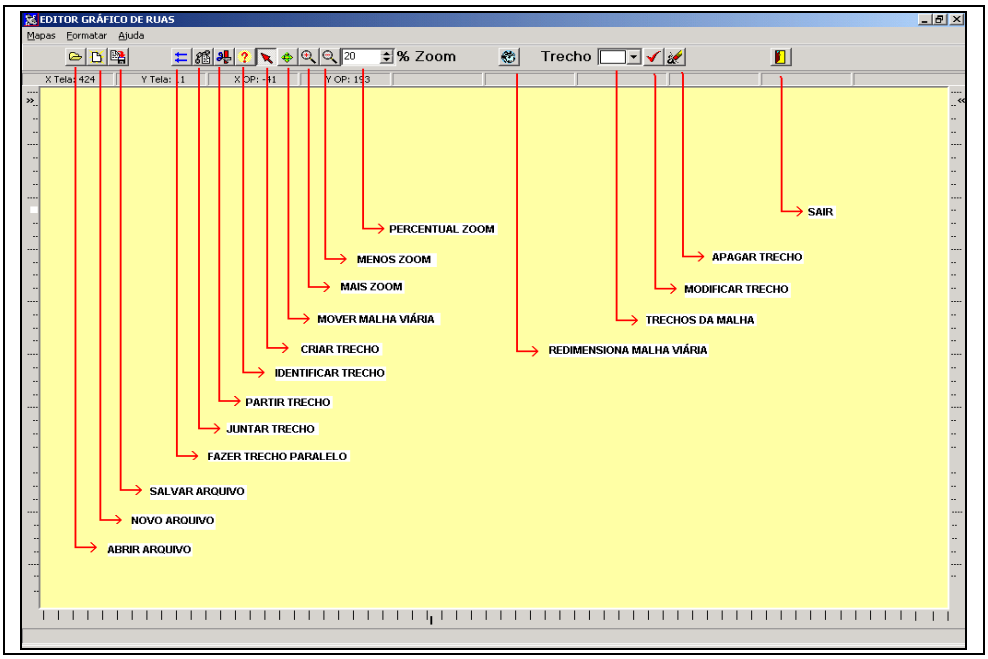
2.6 EGMR VERSÃO 1.0

Bertoldi (2005) desenvolveu a primeira versão do EGMR, com várias funcionalidades para o desenho de malhas viárias, a partir da necessidade do simulador criado por Freire (2004).

A seguir são apresentadas as principais características desta primeira versão.

A tela principal de edição de malhas rodoviárias está representada na Figura 6. É

composta de menus no estilo *windows* e de uma barra de ferramentas composta de ícones que identificam a funcionalidade desejada pelo usuário. Existe um campo para que o usuário informe a proporção de *zoom* que deseja aplicar e outro campo do tipo lista de seleção, que identifica o número do trecho criado e/ou selecionado.

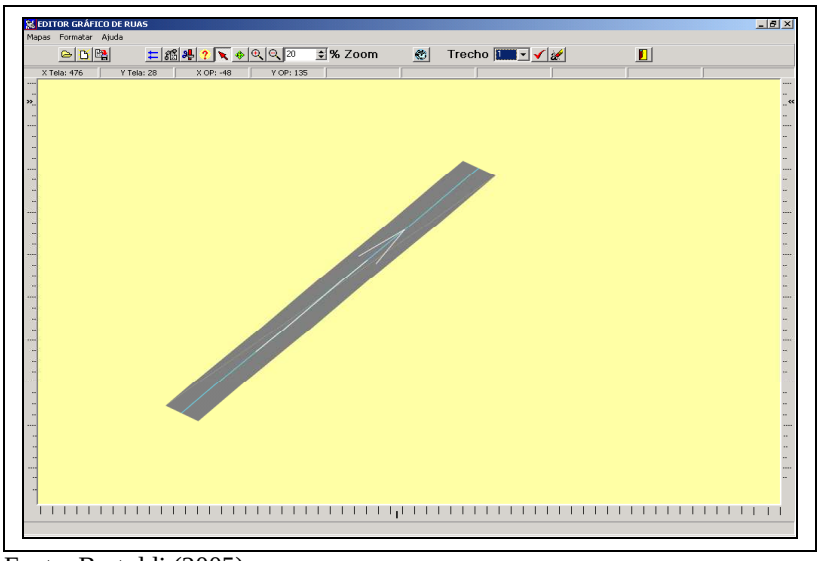


Fonte: Bertoldi (2005).

Figura 6 – Tela principal da versão 1.0 do EGMR e suas funções

Na parte mais esquerda da barra de ferramentas (Figura 6) estão os botões para manipulação de arquivo. O usuário pode escolher entre criar um novo arquivo, abrir um arquivo pré-definido e salvar o arquivo com a malha editada.

O segundo conjunto de ferramentas, situado no centro da barra, é relacionado com a criação e manipulação dos trechos. Cada trecho é representado por um segmento de reta, com um traço no meio e uma seta indicando o sentido do trecho. A Figura 7 mostra um trecho criado.



Fonte: Bertoldi (2005).

Figura 7 – Representando um trecho criado

A cada novo trecho criado, o sistema chama a rotina para verificar a existência de interseções entre trechos. Quando uma interseção é detectada, o sistema automaticamente divide os trechos, criando mais dois trechos por interseção.

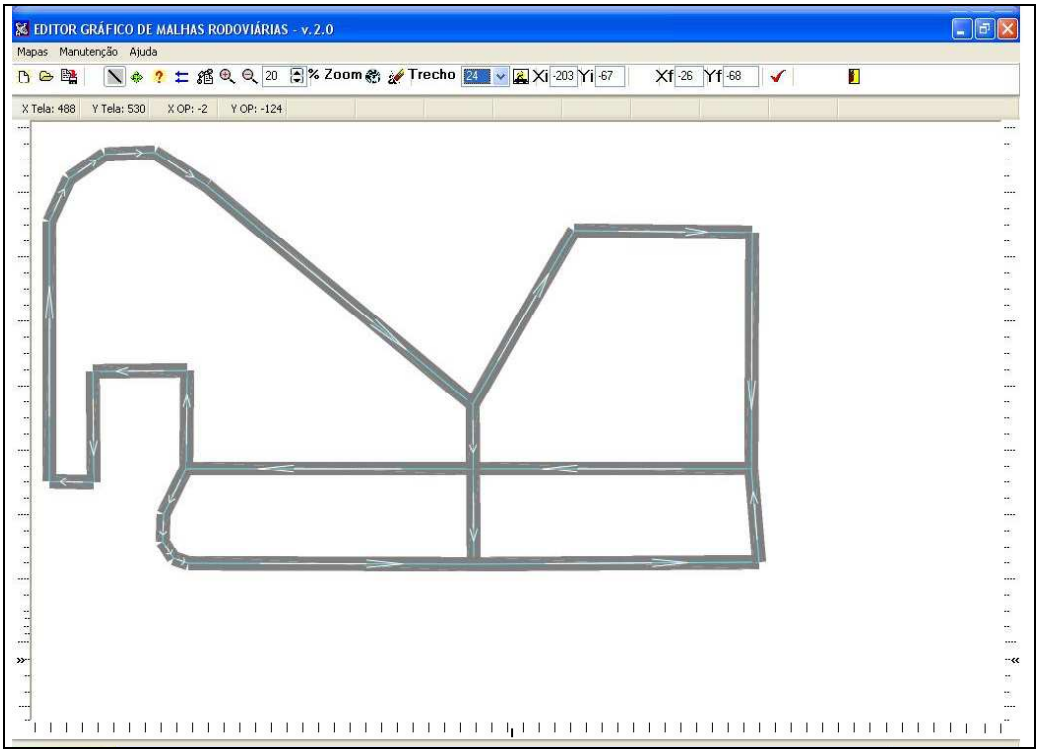
Os outros botões referem-se às funções de partir trecho, tornar trecho paralelo, juntar trecho, identificar trecho, mais *zoom* e menos *zoom* e o botão mundo, responsável por mostrar todo o desenho na tela. Na seqüência, existe uma lista de seleção de trechos, onde o usuário tem uma visão de quantos e quais são os trechos atuais na tela, podendo selecionar um para manipulação. O penúltimo botão da barra de ferramentas chama uma tela para modificar manualmente cada atributo do trecho. O último botão desta barra, trata da saída ou encerramento do sistema. Caso a malha não tenha sido salva, o usuário é questionado da necessidade ou não de salvá-la.

2.7 EGMR VERSÃO 2.0

Partindo do trabalho de Bertoldi (2005), Froeschlin (2006) implementou a segunda versão do software EGMR com várias alterações e melhorias. Nesta versão, Froeschlin (2006) redefine a tela e a posição dos botões para uma melhor apresentação das funções e a redução das possibilidades de erros por parte do usuário. Na seqüência são apresentadas suas principais funcionalidades e novas ferramentas incorporadas: tela principal, semáforos, identificação de trechos, agilidade na criação de malhas e o cadastro de ruas.

2.7.1 Tela principal

A tela principal do sistema foi remodelada no estilo windows, para facilitar a utilização. A Figura 8 ilustra a tela principal do EGMR 2.0, com alguns trechos desenhados.



Fonte: Froeschlin (2006).

Figura 8 – Tela principal do EGMR 2.0

2.7.2 Semáforos

Os semáforos são criados mantendo a tecla “s” pressionada e clicando no trecho onde ele deve ser inserido. A Figura 9 mostra um trecho com um semáforo, que é apresentado na forma de um pequeno quadrado, de cor intermitente entre verde e vermelha, sobre o trecho.

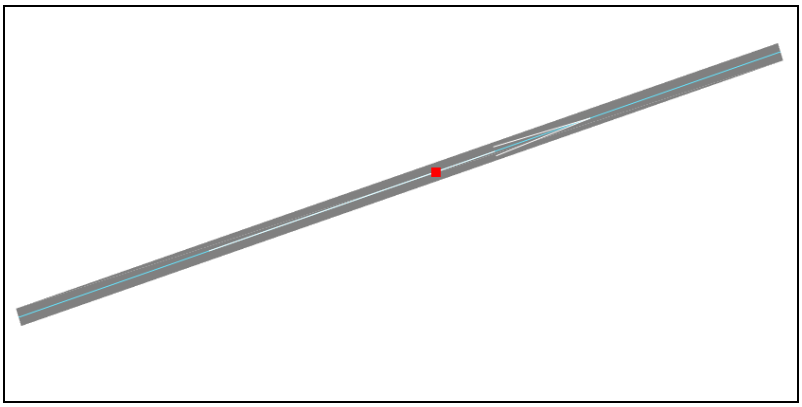


Figura 9 – Trecho com um semáforo

2.7.3 Identificação de trechos

A identificação de trechos com a finalidade de edição é realizada, selecionando a ferramenta correspondente e clicando no trecho desejado. Se for necessária a seleção de vários trechos, deve-se manter a tecla *shift* pressionada.

Froeschlin (2006) efetuou algumas correções no comportamento desta função, tanto no funcionamento do algoritmo de detecção do trecho como na forma de representação da seleção para o usuário. A Figura 10 ilustra a seleção de dois (2) trechos.

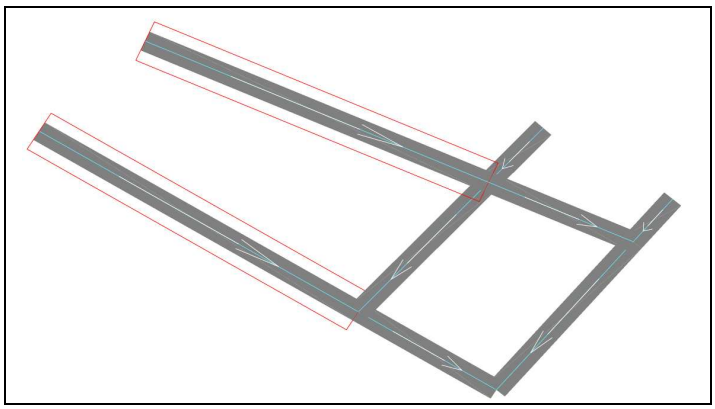


Figura 10 – Trechos selecionados

2.7.4 Agilidade na criação de Malhas

No Quadro 6 são apresentadas as funcionalidades e suas teclas de atalho correspondentes, que oferecem maior agilidade no desenho da malha, entretanto, espera-se que o usuário conheça algumas funções que não estão presentes na barra de ferramentas.

Funcionalidade	Atalho / Comando
Criar semáforos	Função reta selecionada + tecla “S” pressionada + *click* dentro dos limites do trecho
Desenhar trecho já partindo de uma extremidade de outro trecho	Função reta selecionada + tecla “J” pressionada + *click* dentro do trecho inicial desejado
Excluir trecho	Menu do botão direito do *mouse* ou pressionar tecla “D” quando trecho estiver selecionado
Finalizar trecho num ponto extremo de outro trecho	Durante o desenho do trecho, manter pressionada a tecla “J” e mover o cursor para dentro dos limites do trecho, para que o sistema faça a sugestão da extremidade
Juntar trechos	Função reta selecionada + tecla “L” pressionada + movimentos do *mouse*
Modificar trecho	Clique duplo com o botão esquerdo no trecho
Mover trechos	Função identificação selecionada + tecla “M” pressionada + movimentos do *mouse*
Mudar direção do trecho	Menu do botão direito do *mouse* ao clicar num trecho
Partir trecho	Função reta selecionada + tecla “P” pressionada + *click* num ponto do trecho a partir
Selecionar mais de um trecho	Função identificação selecionada + tecla “SHIFT” pressionada + *click* nos trechos
Copiar / Colar trecho(s)	Com o(s) trecho(s) selecionado(s) pressionar CTRL+C para copiar e CTRL+V para colar
Identificar os trechos que compõem uma rua	Função identificação selecionada + manter pressionada a tecla “R” + clicar sobre trecho

Fonte: Froeschlin (2006).

Quadro 6 – Relação das funcionalidades e seus atalhos

2.7.5 Ruas

Com o intuito de fornecer mais dados sobre a malha representada, foram adicionadas ferramentas para o cadastro e a identificação de ruas. Ao ser ativada, a função mostra a tela apresentada na Figura 11, onde é possível criar novas ruas, excluí-las e adicionar-lhes trechos.

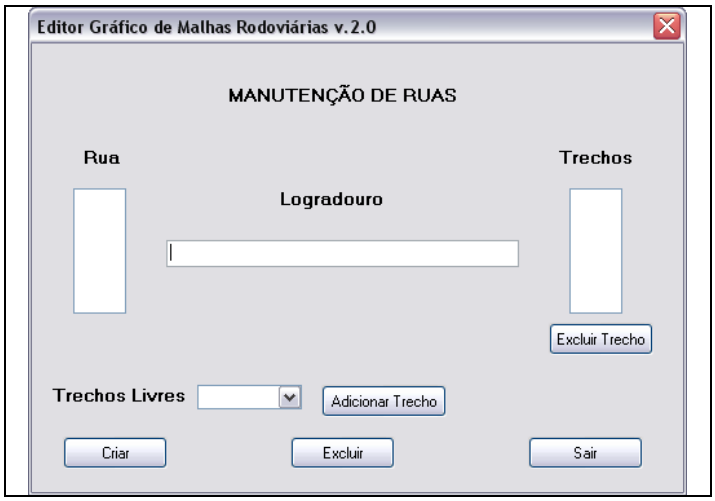
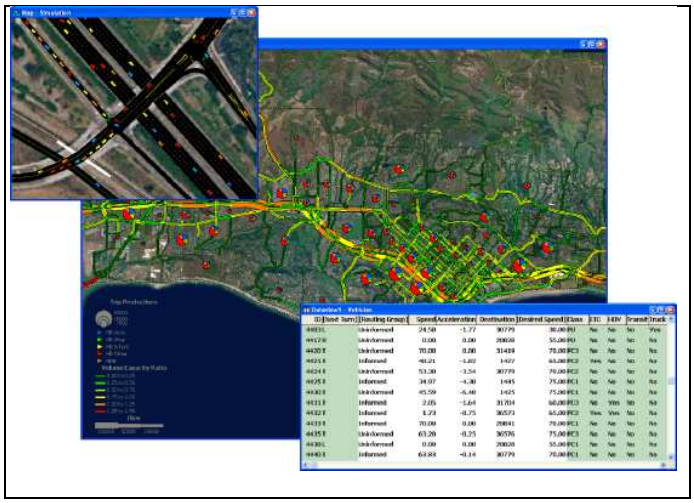


Figura 11 – Tela para manutenção de ruas da malha

2.8 TRANSMODELER

O *TransModeler*, desenvolvido pela empresa Caliper Corporation (2006), é uma ferramenta comercial destinada à simulação e controle de tráfego viário, que tem como público alvo planejadores e engenheiros de trânsito. Fornece meios para a projeção e representação de grandes malhas em todos os tipos de tráfego, bem como mantém uma base com os dados obtidos, a fim de prever os efeitos das modificações que se pretende realizar na malha viária de uma determinada região. Este software tem como característica marcante o fato de representar cenas bem próximas à realidade, pois possibilita o desenho a partir de imagens obtidas de satélite. Com isso, é possível o desenvolvimento de um modelo seguindo as vias e representado-as de maneira concisa e coerente às situações a qual o simulador se destina. Na Figura 12 observa-se um exemplo de utilização do *TransModeler*.

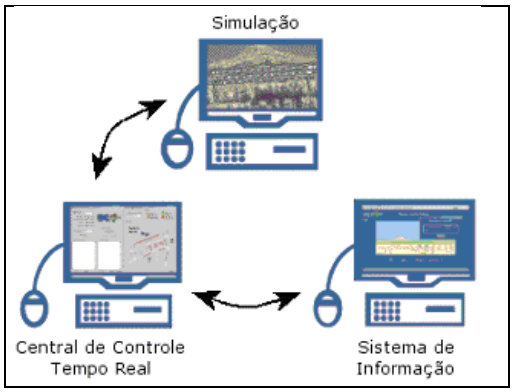


Fonte: Caliper Corporation (2006).

Figura 12 – TransModeler

2.9 SINCMOBIL

Em Farines et al. (2003) são apresentados os resultados de um projeto realizado na Universidade Federal de Santa Catarina (UFSC), o Sistema de Informação e Controle para Mobilidade Urbana (SINCMobil), que durou quarenta e oito (48) meses. Foram desenvolvidos três módulos (Figura 13), com a finalidade de encontrar maneiras para melhorar o tráfego das grandes cidades.



Fonte: Farines et al. (2003).

Figura 13 – Módulos do SINCMobil

O ambiente de simulação, juntamente com controladores virtuais, têm a função de simular os semáforos, fornecendo os dados necessários à Central de Controle de Tráfego em Tempo Real que os analisa e alimenta funções de otimização a fim de determinar quais serão as futuras temporizações dos semáforos. O Sistema de Informação, por sua vez, traduz os dados fornecidos pela Central de Controle, de modo que os usuários do sistema possam

interpretar e acompanhar as condições do tráfego.

Esta estrutura permite que o Sistema de Informação e a Central de Controle Tráfego em Tempo Real operem em um cenário real, substituindo o micro-simulador por uma malha viária e os controladores virtuais por controladores reais.

3 DESENVOLVIMENTO DO PROTÓTIPO

A seguir serão apresentados os requisitos, a especificação e a implementação do EGMR 3.0.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos funcionais do software são:

a) incluir meios para a representação de viadutos;
b) incluir a função para o desenho tendo como fundo uma imagem;
c) converter o formato do arquivo gerado para XML;
d) melhorar o desenho de curvas;
e) incluir a possibilidade de visualização em 3D da malha desenvolvida.

Os requisitos não-funcionais do software incluem:

a) adaptar a modelagem atual orientada a objetos, representada através da UML para refletir as novas funcionalidades;
b) converter o software para a linguagem Java, utilizando a API JoGL.

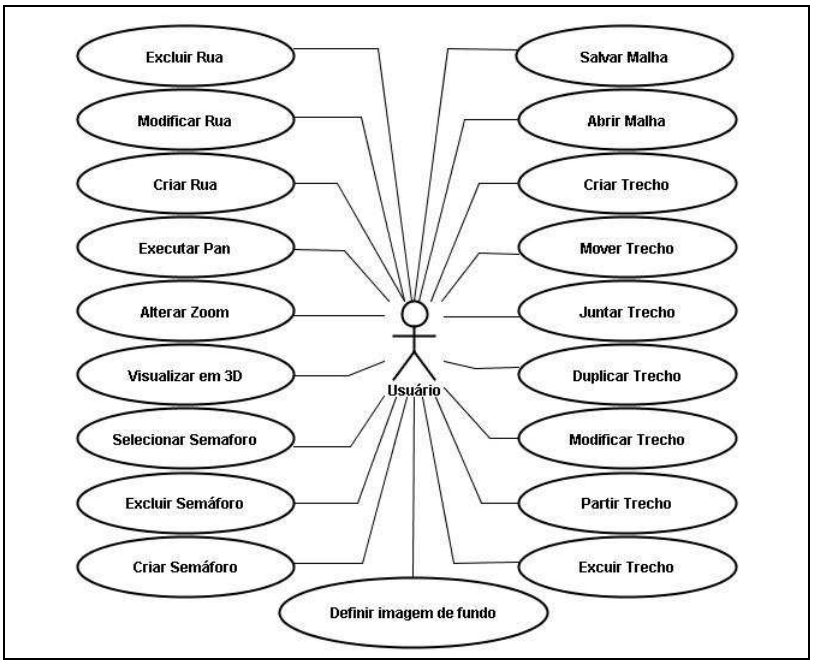
3.2 ESPECIFICAÇÃO

Para a especificação foi utilizada a *Unified Modeling Language* (UML), com auxílio da ferramenta JUDE, adaptando os diagramas de casos de uso, de atividades e de classes, já existentes na versão anterior do software. Posteriormente são apresentados os diagramas de seqüência desenvolvidos para a versão 3.0 do EGMR.

3.2.1 Diagrama de casos de uso

Este diagrama foi refeito para refletir as modificações dos casos de uso desta versão.

Foram incluídos os casos de uso Visualizar em 3D e Definir imagem de fundo. Na Figura 14 é apresentado o diagrama de casos de uso conforme os requisitos apresentados anteriormente.



Fonte: adaptado de Froeschlin (2006).

Figura 14 – Diagrama de casos de uso

3.2.2 Diagrama de atividades

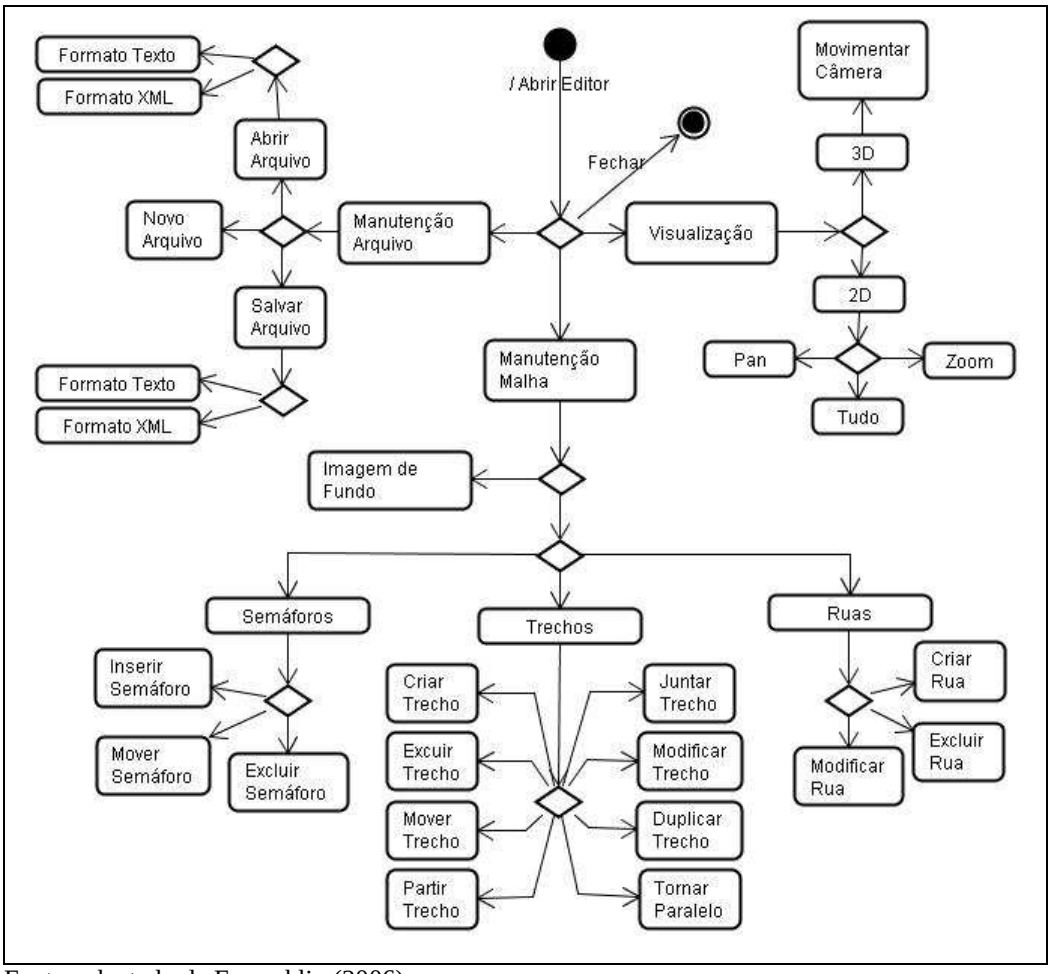
Este diagrama (Figura 15) também foi refeito, sofrendo algumas alterações. As modificações realizadas são:

a) visualização da malha: a partir desta versão, também poderá ser em 3D, com possibilidade de movimentação de uma câmera virtual no espaço de desenho da malha;

b) manutenção da malha: inclusa a atividade de definição de uma imagem de fundo, obtida através de satélite ou aerofotogrametria, para a representação de cenas reais. As funções para criar trecho, juntar, duplicar, partir, tornar trecho paralelo, excluir trecho, mover trecho, modificar propriedades de trecho, incluir semáforo, mover semáforo, excluir semáforo, criar rua, modificar rua e excluir rua foram mantidas;

c) Manutenção de arquivos: com a finalidade de manter a compatibilidade com as malhas criadas em versões anteriores do EGMR (1.0 e 2.0), as operações com arquivos no formato texto foram mantidas, porém foram inclusas funcionalidades

para a manipulação de documentos no formato XML;

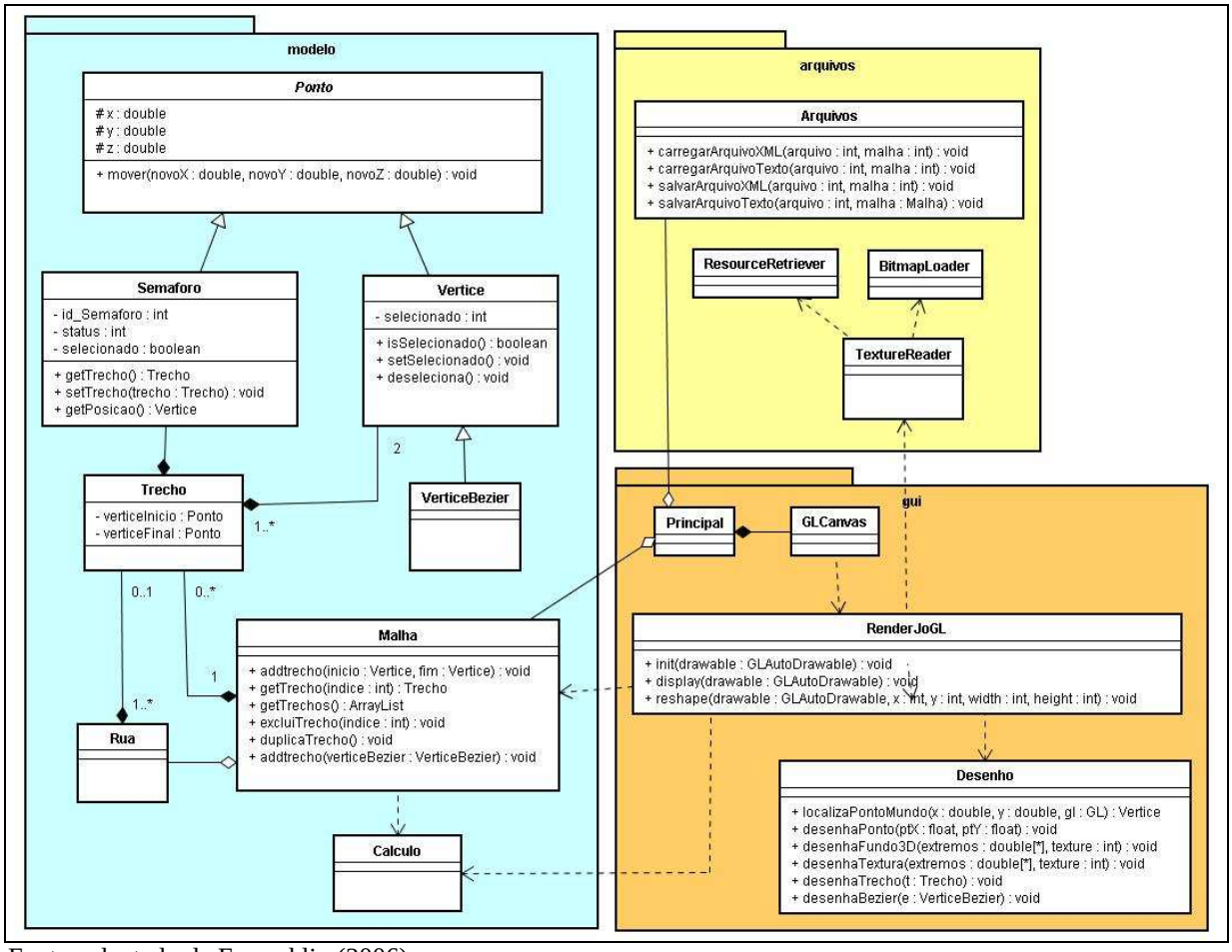


Fonte: adaptado de Froeschlin (2006).

Figura 15 – Diagrama de atividades

3.2.3 Diagrama de classes

A Figura 16 mostra o diagrama de classes do protótipo, também refeito para representar as novas classes definidas.



Fonte: adaptado de Froeschlin (2006).

Figura 16 – Diagrama de classes

Neste diagrama estão representados apenas os principais atributos e métodos de cada classe. As classes e suas principais funções são:

a) `Ponto`: classe abstrata que define características para a construção de um ponto de 3 coordenadas;
b) `Semáforo`: herda as características de `Ponto`. Responsável pela inclusão de semáforos nas vias;
c) `Vertice`: classe derivada de `Ponto`, utilizada na criação de trechos. Define os pontos inicial e final. Criada para representar os vértices da estrutura da malha, a qual pode pertencer á vários trechos, tornando-os interligados. Quando um vértice é movido, por exemplo, todos os trechos que estão ligados a ele são

automaticamente movimentados e as ligações são mantidas;

d) `Trecho`: Responsável pela criação e manutenção dos trechos da malha. Como a estrutura representa um grafo, cada objeto desta classe é formado por dois vértices, um inicial e um final;

e) `Malha`: promove a manutenção da malha viária. É formada principalmente por duas listas, uma de vértices, que armazena os pontos de cada trecho, e uma de trechos (arestas), que são referências informando a interconexão dos vértices que formam a malha. Esta classe ainda outra lista, para comportar os trechos criados com a ferramenta de Bézier, onde são armazenados os pontos da curva que são informados para a JoGL, que calcula os pontos intermediários e desenha a curva na tela;

f) `Principal`: promove a interação com o usuário. Implementa funcionalidades para responder a eventos do *mouse* e de teclado;

g) `Desenho`: implementa funções para o desenho de trechos, linhas, curvas, etc. Por utilizarem funções de OpenGL, seus métodos só podem ser invocados a partir da classe `RenderJoGL`, segundo as limitações citadas no guia da *API* JoGL (JAVA.NET, 2006);

h) `Rua`: mantém informações referentes as ruas da malha, formada por um ou mais trechos;

i) `Calculo`: realiza cálculos diversos, como a intersecção de trechos;

j) `VerticeBezier`: herda as características de `Vértice`, permitindo a criação e manutenção de curvas. O diferencial desta classe é possuir um ponto de controle, que indica a sinuosidade do trecho em questão.

k) `Arquivos`: classe responsável pelas operações de persistência de dados no formato XML. Além disso, manipula arquivos no formato texto, padrão das versões anteriores, entretanto, não salva os trechos criados com a ferramenta de Bézier, pois não são suportadas.

l) Existem ainda as classes `GLCanvas` e `RenderJoGL`, proveniente da biblioteca JoGL, que oferece o canvas para a interação com o usuário e as classes `TextureReader`, `BitmapLoader` e `ResourceRetriever`, obtidas do portal da Neon Helio Game Development (2007), que realizam o mapeamento de um arquivo de imagem para uma textura JoGL, para que possa ser visualizada tanto em 2D como em 3D.

3.2.4 Diagrama de Seqüência

Na Figura 17 é mostrado o diagrama de seqüência para criação de um trecho. Foram representadas as principais operações entre as classes, para a ação de inserir um novo trecho na malha, realizada pelo usuário.

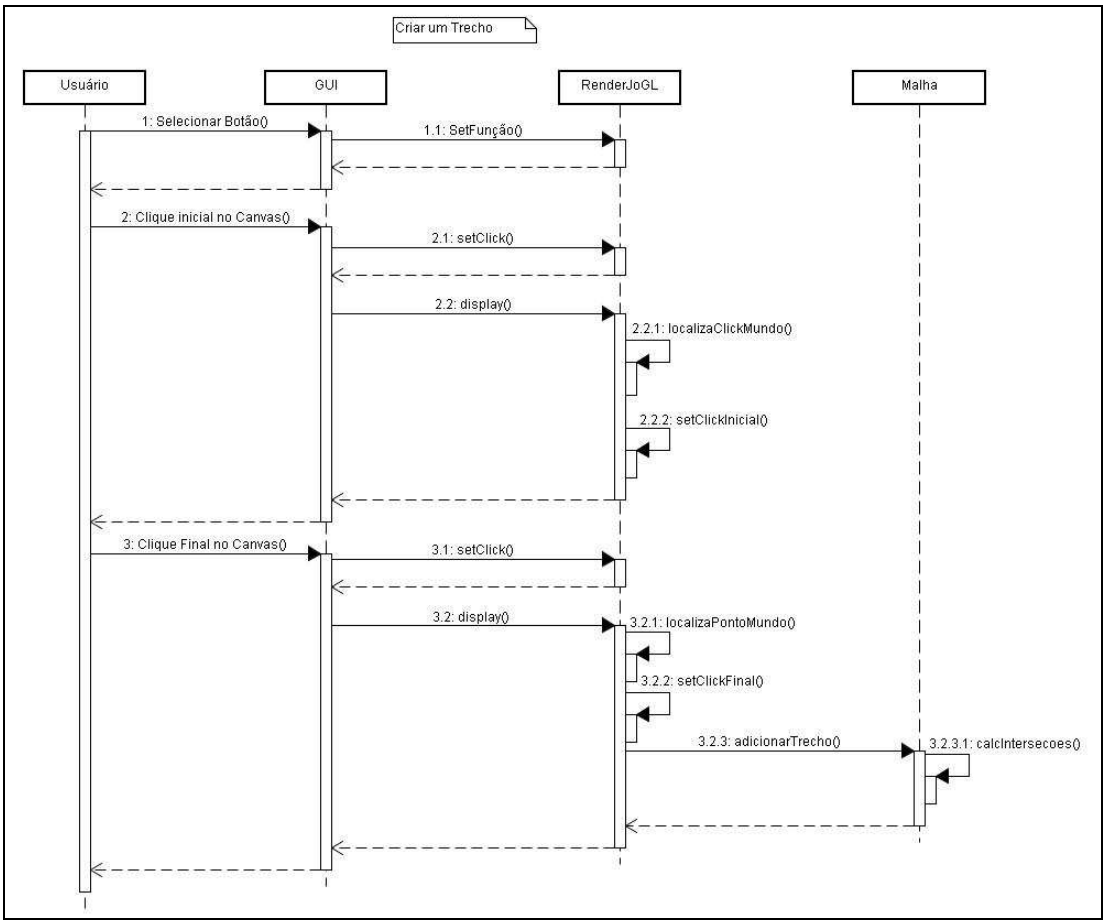


Figura 17 - Diagrama seqüência para criação de um trecho

A seqüência para inserir uma imagem de fundo é representada na Figura 18.

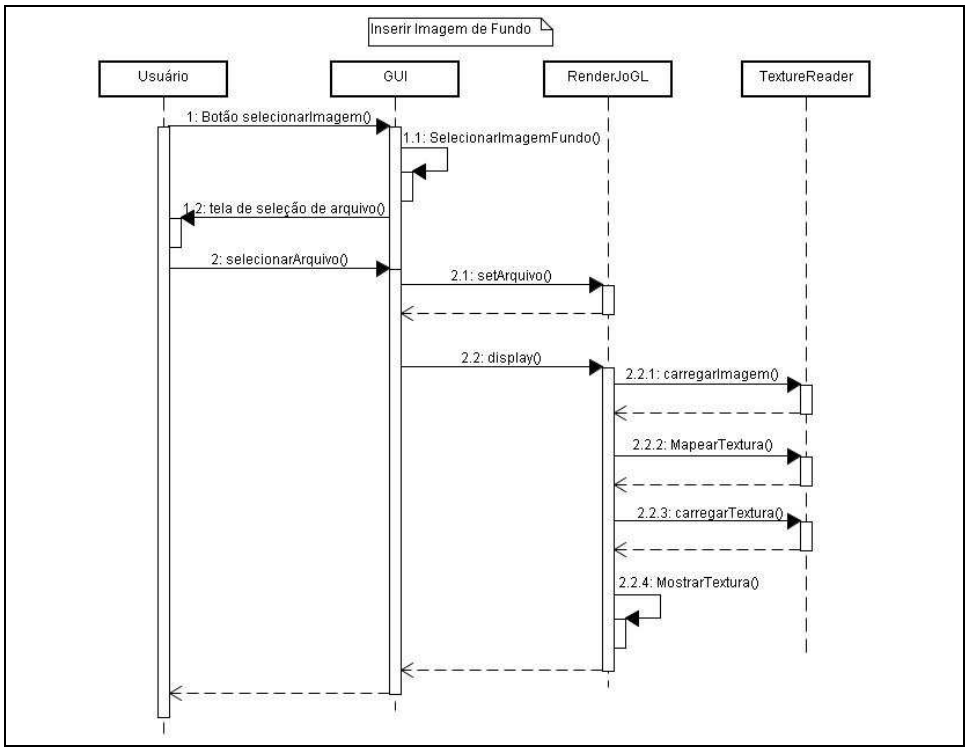


Figura 18 - Diagrama de seqüência para inserção de uma imagem de fundo

Nos dois diagramas acima, é importante perceber a utilização de JoGL com Java, que é sempre realizada a partir da classe JoGL. Um exemplo é o clique do *mouse*, que é armazenado em uma variável global da classe RenderJoGL e somente é convertido, de coordenadas de tela para coordenadas JoGL, quando é invocado o método `display()` da classe RenderJoGL.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O código fonte da versão 2.0 do software foi convertido para a linguagem Java com a *IDE* Netbeans. Para a utilização da biblioteca OpenGL nesta linguagem foi utilizada a *API* JoGL. Já para a persistência dos dados em disco no formato *XML* utilizou-se a *API* XStream.

Nas seções seguintes são apresentados os detalhes da implementação da versão 3.0 do

EGMR. Primeiramente é apresentada a interface gráfica com o usuário do EGMR 3.0, uma breve explanação referente a conversão do código de Delphi para Java e em seguida os detalhes da implementação.

3.3.2 Operacionalidade

Para facilitar a utilização do EGMR a interface gráfica com o usuário foi refeita e remodelada e a barra de ferramentas apresenta todas as funcionalidades do *software*.

A Figura 19 ilustra a tela principal do sistema.

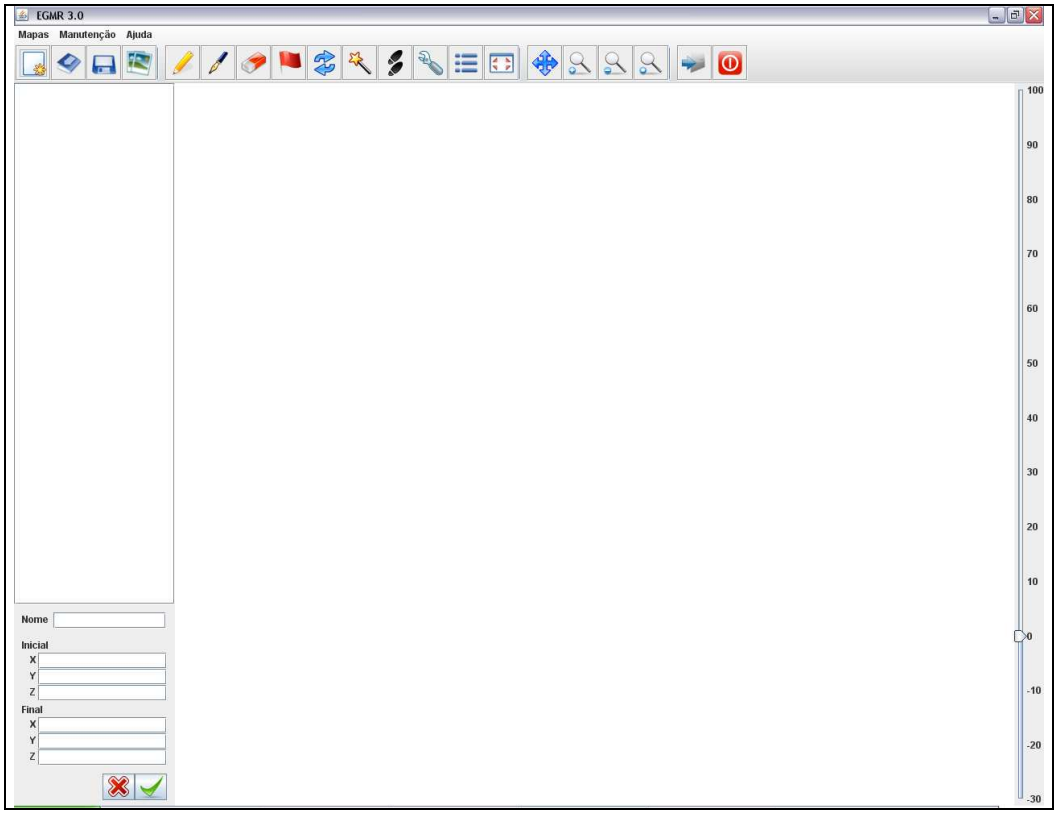


Figura 19 – Tela Principal

Na versão 2.0 do EGMR, muitas funções estavam disponíveis somente através de atalhos do teclado, dificultando a utilização pelo usuário. Foram adicionados alguns botões para realizar as tarefas. Na Figura 20 é apresentada a função de cada botão da barra de ferramentas.

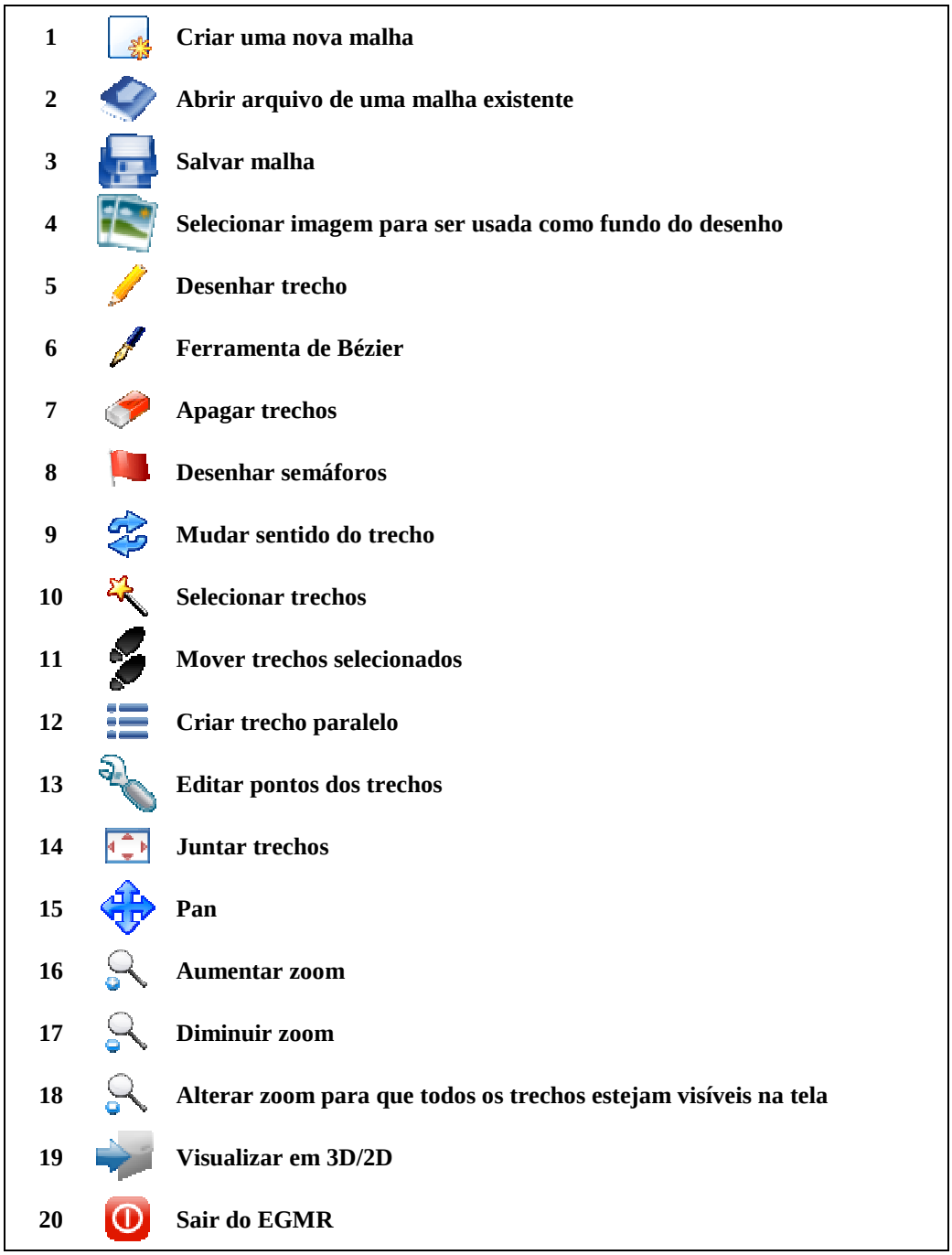


Figura 20 – Funções dos botões da barra de ferramentas do EGMR 3.0

3.3.3 Conversão de código

O código fonte, que nas versões anteriores encontra-se em Delphi, foi convertido para a linguagem Java, visando tornar o software multiplataforma e facilitar o desenvolvimento de uma versão para a internet.

Algumas partes do código, como cálculos específicos para a colisão entre trechos e

seleção de trechos foram facilmente convertidos, simplesmente modificando detalhes da sintaxe. Porém, em partes mais específicas, como na utilização de OpenGL, o código foi totalmente refeito, pois não era compatível diretamente com a linguagem Java.

Um exemplo disto é que em Delphi, as funções OpenGL estão disponíveis em qualquer ponto do código, já em Java, estas funções só podem ser acessadas a partir de um objeto `GL`, disponível somente na classe `GLEventListener`, forçando a conversão e reestruturação do código fonte.

3.3.4 Desenho de viadutos

Com a finalidade de possibilitar o desenho em vários níveis, foi adicionada uma barra lateral que possibilita que seja modificada a altura do ponto a ser desenhado. Esta barra está posicionada à direita da janela de desenho, como pode ser observado na Figura 19. Trata-se de uma escala que varia de –30 á 100. Para fazer um trecho mais alto ou mais baixo, altera-se o valor da escala. É recomendado que esta barra seja associada ao botão 14 (Figura 20), para que o trecho seja contínuo, o que possibilita o desenho de trechos inclinados, com o vértice de início em uma determinada altura e o vértice de final em outra altura.

3.3.5 Imagem de fundo

Outra funcionalidade disponível no EGMR 3.0 é a inserção de uma imagem de fundo para o desenho de malhas mais próximas a realidade. Para selecionar esta opção, utiliza-se o botão 4 (Figura 20). Após selecionado, uma janela para a seleção do arquivo da imagem aparecerá. A Figura 21 mostra um desenho de uma malha tendo como imagem de fundo uma cena do mudo real.

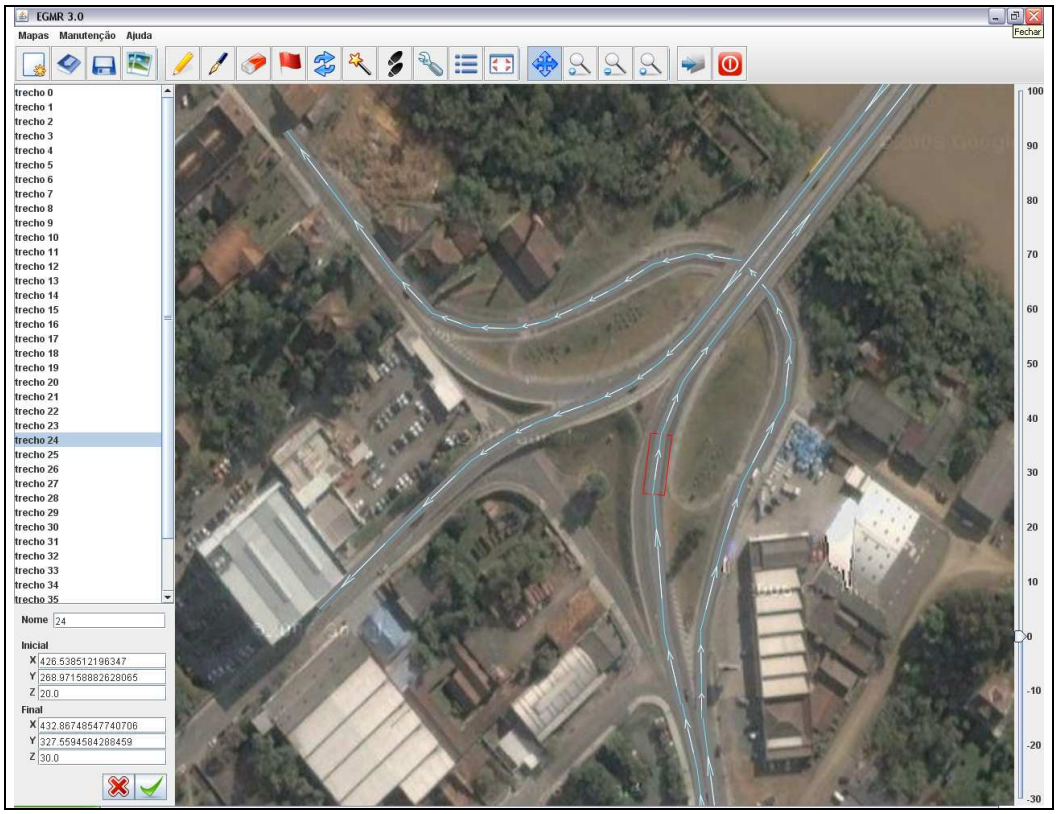


Figura 21 – Desenho tendo como fundo uma imagem

3.3.6 Ferramenta de Bézier

Para o desenho de curvas de Bézier, foi incluso o botão 6 na barra de ferramentas (Figura 20). Para desenhar uma curva de Bézier, seleciona-se esta ferramenta, arrasta-se o *mouse*, definindo as posições dos pontos de controle. Após, repete-se esta operação para cada ponto do trecho. Cada ponto pertencente ao trecho é desenhado na cor azul e os pontos de controle na cor azul clara. A Figura 22 exemplifica um trecho desenhado com a ferramenta de Bézier.

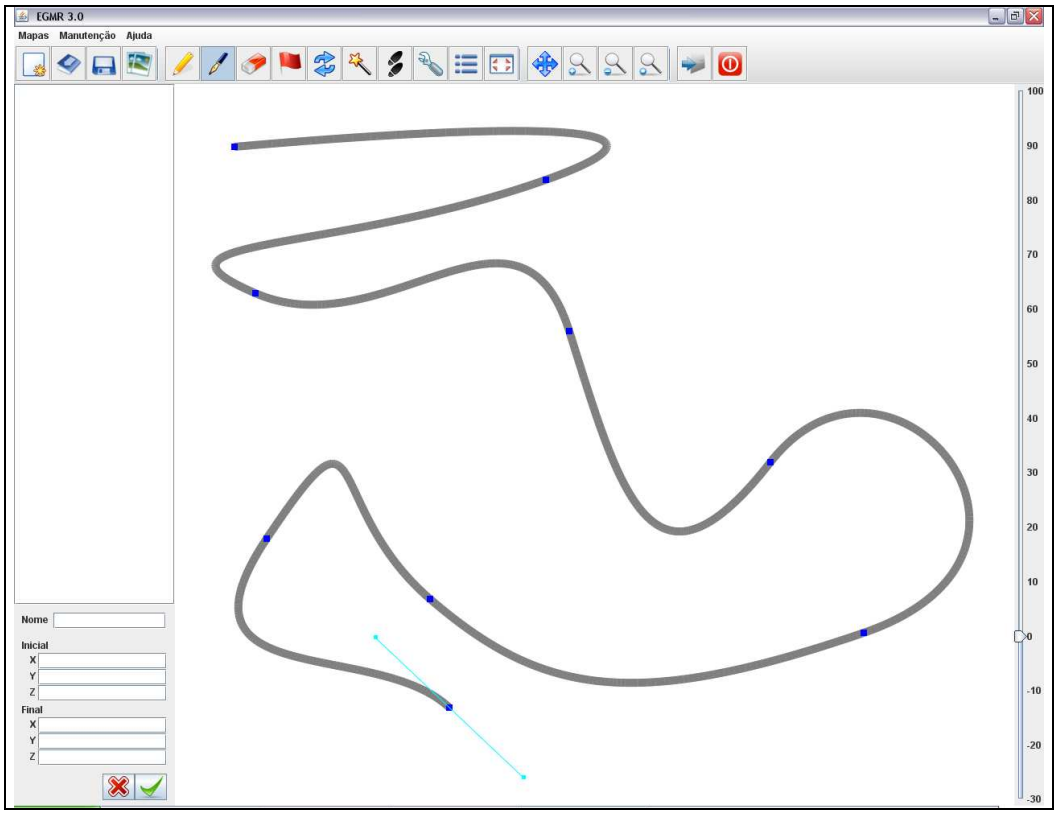


Figura 22 – Trecho criado com a ferramenta Bézier

A OpenGL gera e calcula automaticamente as curvas de Bézier através dos pontos de controle. O código necessário para o cálculo dos pontos intermediários está apresentado no Quadro 7.

```
1. gl.glMap1d(gl.GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, db.array(),0);
2. gl.glEnable(gl.GL_MAP1_VERTEX_3);
3. gl.glLineWidth(15);
4. gl.glBegin(gl.GL_LINE_STRIP);
5.       for(double f = 0;f <= 1.01;f += 0.01)
6.             gl.glEvalCoord1d(f);
7. gl.glEnd();
```

Quadro 7 - Cálculo do pontos intermediários das curvas de Bézier

Nas linhas 5 e 6 é realizado o laço que calcula os pontos intermediários de cada trecho da curva. Na linha 6 são calculados os pontos intermediários da curva, quanto maior o número destes, isto é, quanto menor o incremento da variável `f`, mais pontos são calculados, suavizando o desenho da curva.

3.3.7 Visualização da malha em 3D

A visualização da malha em 3D é possível a partir da utilização do botão 19 na barra de ferramentas (Figura 20). Após a seleção deste, é apresentada uma tela com um plano

constituído de linhas ou a própria imagem de fundo, a fim de proporcionar uma melhor percepção de espaço. A visualização do usuário é realizada através de uma câmera virtual. A movimentação da câmera é realizada com as setas de direção do teclado. Também é possível aproximar ou distanciar a visualização através das teclas + e -, respectivamente. Todos os movimentos de câmera são feitos em relação ao centro da malha. A Figura 23 demonstra a visualização da malha em 3D.

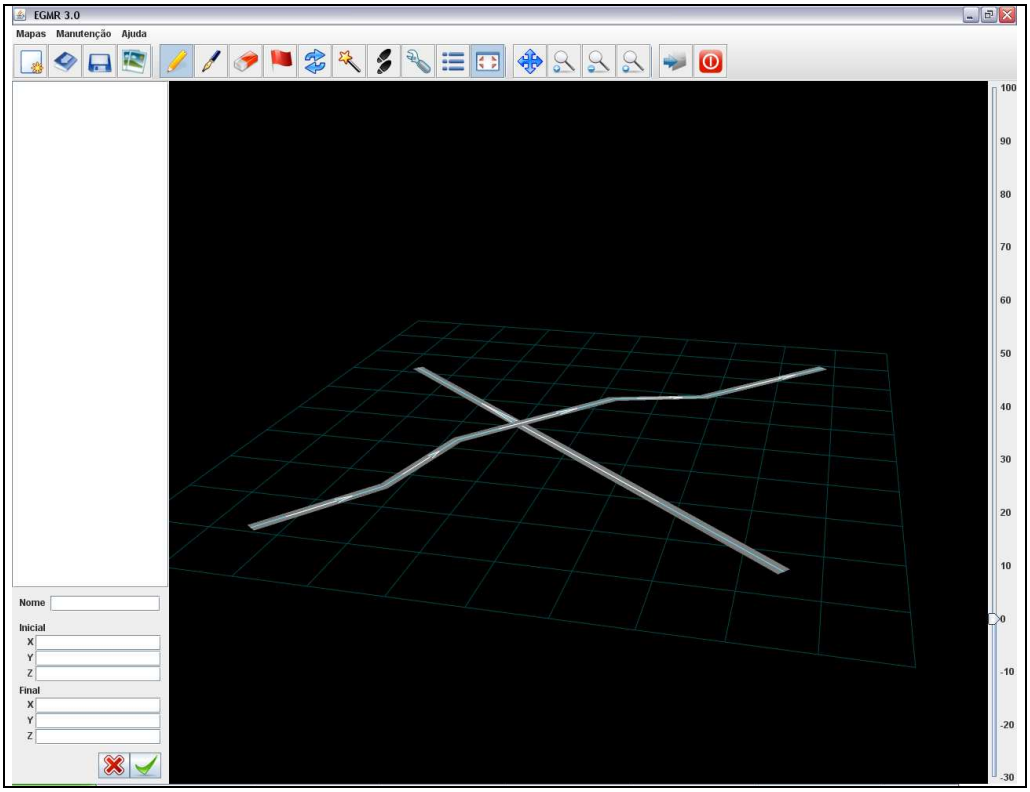


Figura 23 – Visualização da malha em 3D

O Quadro 8 mostra o código necessário para a visualização tridimensional da malha utilizando JoGL.

```
1. if (perspectiva == true){
2.       calculaAlvoCamera();
3.      gl.glClearColor(0f,0f,0f,0f);
4.      glu.gluPerspective(fovy,aspect,zNear,zFar);
5.      gl.glMatrixMode(gl.GL_MODELVIEW);
6.      gl.glLoadIdentity();
7. glu.gluLookAt(obsX,obsY,obsZ,alvoX,alvoY,alvoZ,upX,upY,upZ);}
```

Quadro 8 - Código para a visualização em 3D

No código fonte, nas linhas de 1 à 6, são realizadas algumas configurações e na linha 7, os parâmetros são enviados para a JoGL, configurando a posição da câmera (obsX, obsY e obsZ), o local para onde a câmera está voltada (alvoX, alvoY e alvoZ) e o seu ângulo de rotação (upX, upY e upZ).

3.3.8 Editar Trechos

A edição de trechos pode ser feita com ferramentas como mover ou apagar trechos ou através da lista de trechos situada a esquerda do canvas de desenho. Ao selecionar um trecho da lista, suas coordenadas são apresentadas na parte inferior, onde podem ser modificadas. Dessa forma, os trechos podem também ser editados na visualização em 3D. Na Figura 24 um trecho foi selecionado, logo suas coordenadas são apresentadas nos campos inferiores.

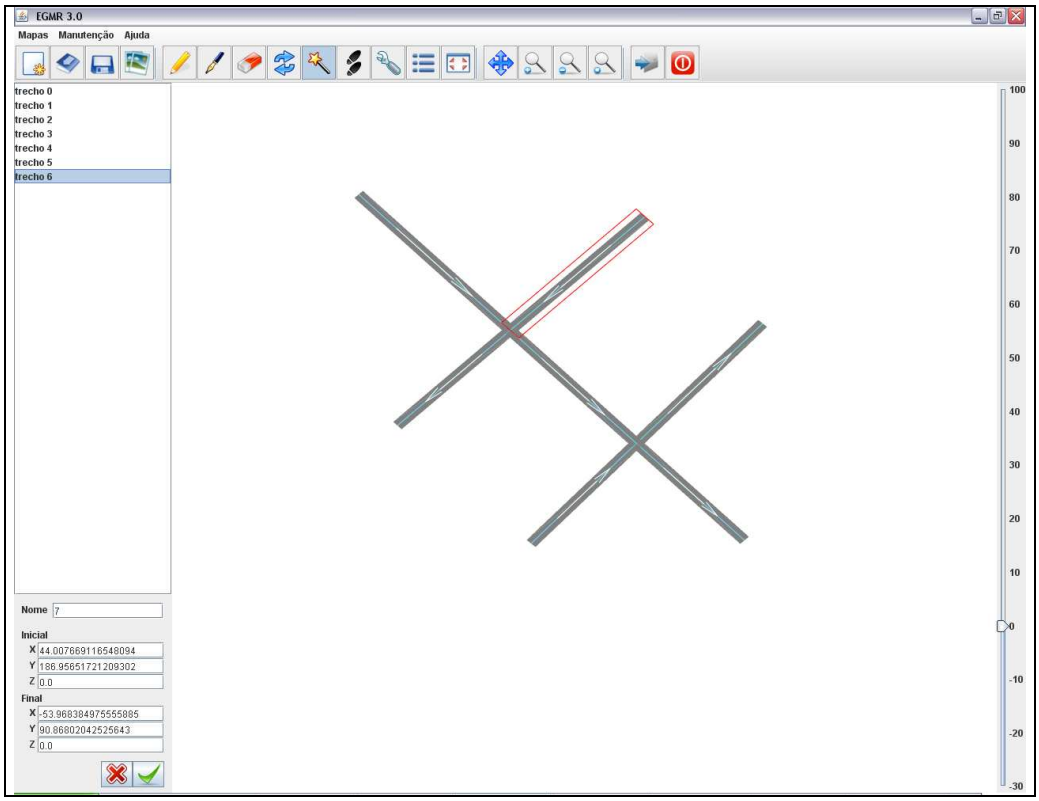


Figura 24 – Seleção de um trecho para edição

Também foi criada uma ferramenta específica para a edição dos vértices dos trechos. Quando esta ferramenta é selecionada, através do botão 13 (Figura 20), os vértices e os controles Bezier são desenhados, possibilitando a edição dos mesmos através do *mouse* (Figura 25).

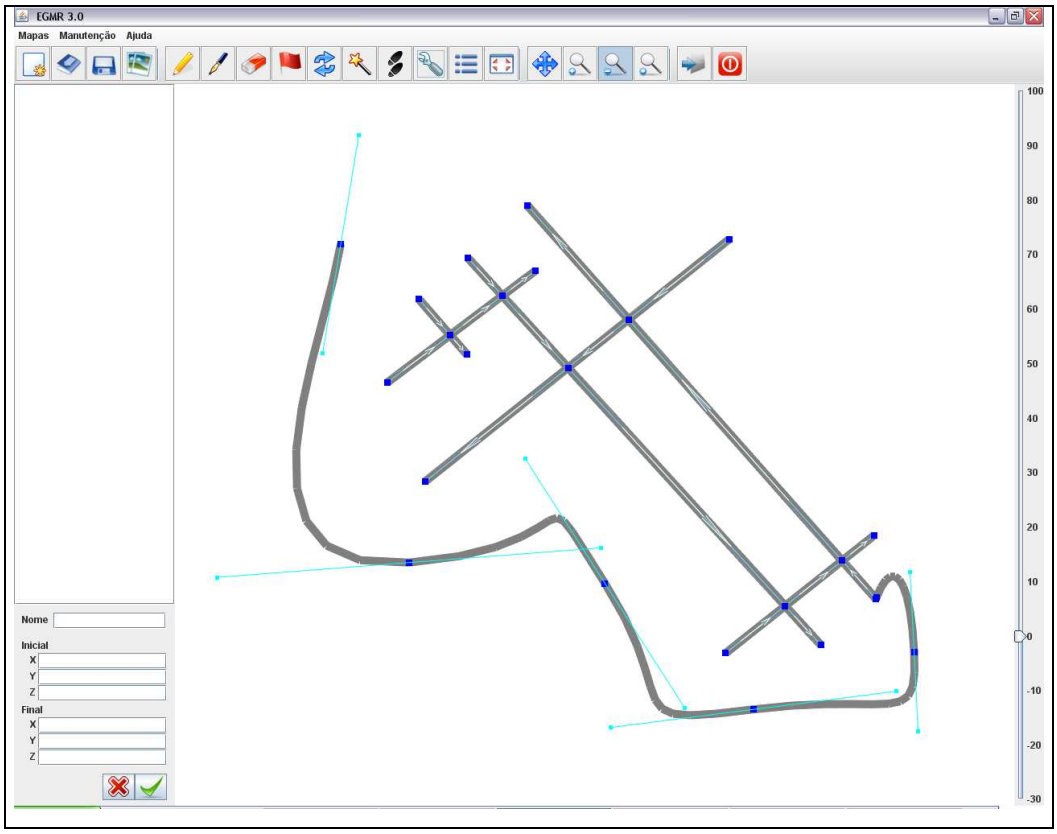


Figura 25 – Ferramenta de edição de trechos

3.3.9 Grafos

Para um melhor controle dos trechos e principalmente dos pontos de encontro de dois ou mais trechos, a estrutura utilizada para a manipulação dos pontos pertinentes a malha foi adaptada para refletir as características de um grafo. Foram criadas duas estruturas, uma para guardar os pontos (vértices) e outra para guardar os trechos (arestas) (Figura 26). Dessa forma, quando um trecho é movido, todos os outros pontos ligados a ele são movidos também, mantendo a ligação entre os mesmos.

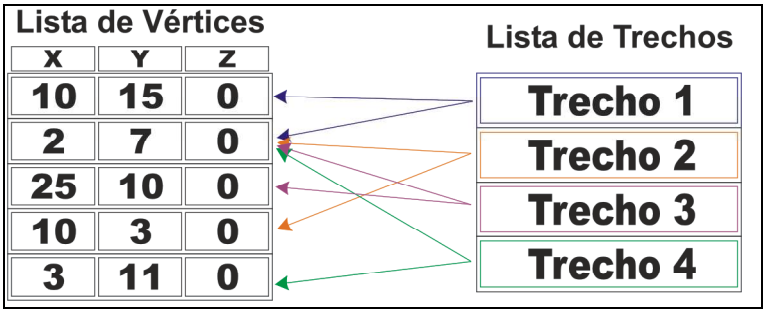


Figura 26 – Estrutura dos trechos

A Figura 27 mostra a movimentação de um trecho da malha. Nota-se que todos os trechos que compartilham os vértices movimentados também são alterados.

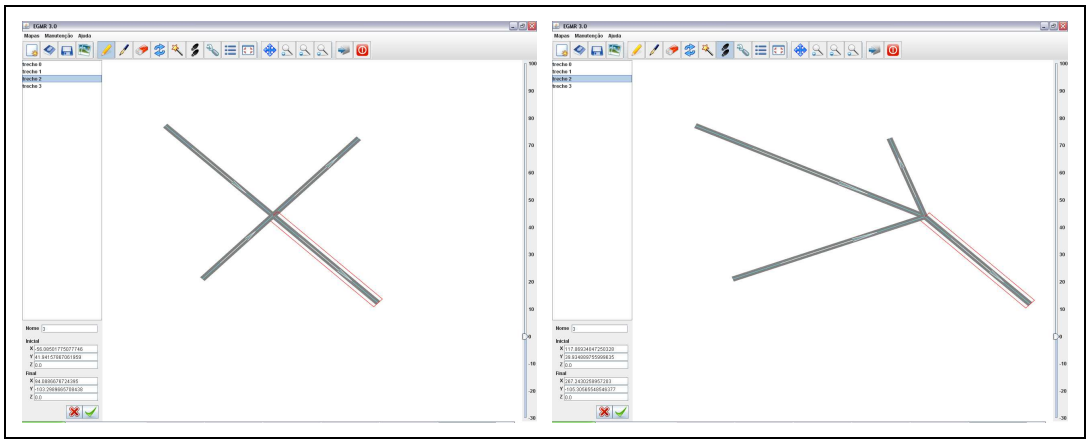


Figura 27 – Movimentação de um trecho

Esta estrutura também foi crucial para a criação de viadutos, permitindo que somente um dos vértices seja elevado, formando um trecho inclinado, sem perder a conexão com os demais trechos da malha.

3.3.10 Multiplataforma

O código fonte do EGMR foi convertido para a linguagem Java, o que entre outras vantagens, também oferece a possibilidade de execução em vários sistemas operacionais, entre eles Linux (Figura 28), MacOS (Figura 29) e Windows (Figura 30). Para tanto, a versão de Java do sistema deve ser, no mínimo, a 5.0 e a JoGL deve estar corretamente configurada.

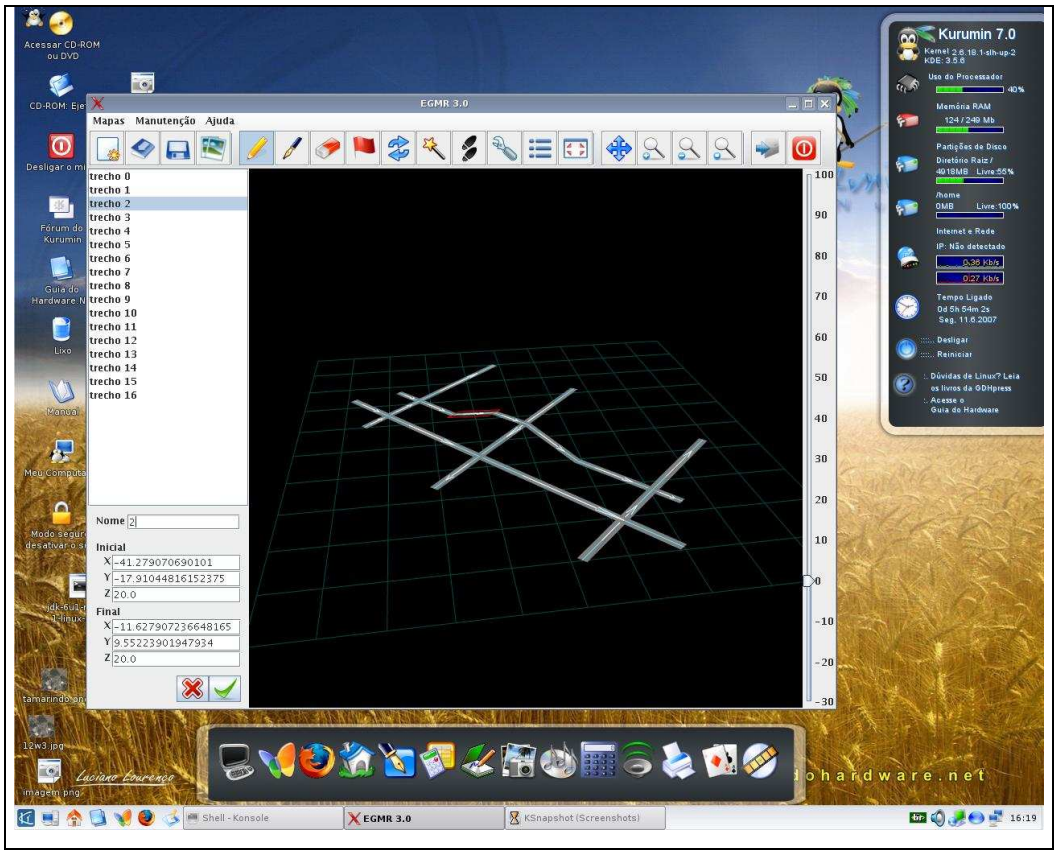


Figura 28 – EGMR no Sistema Operacional Linux

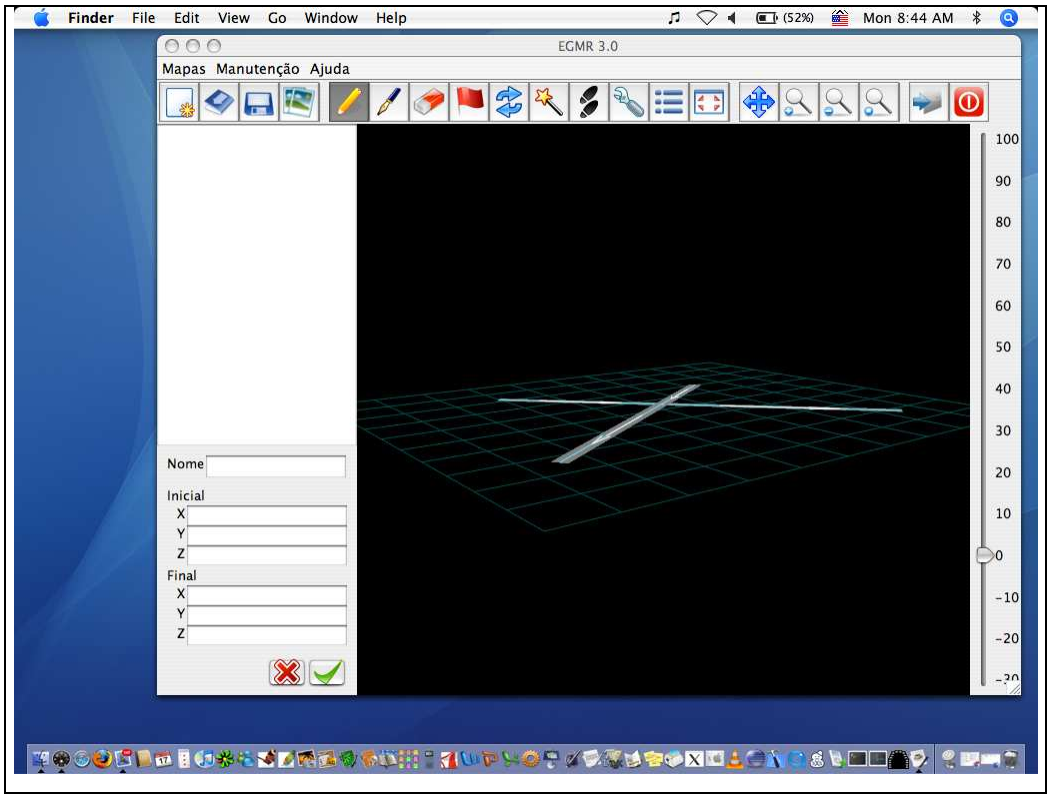


Figura 29 – EGMR no Sistema Operacional MacOS

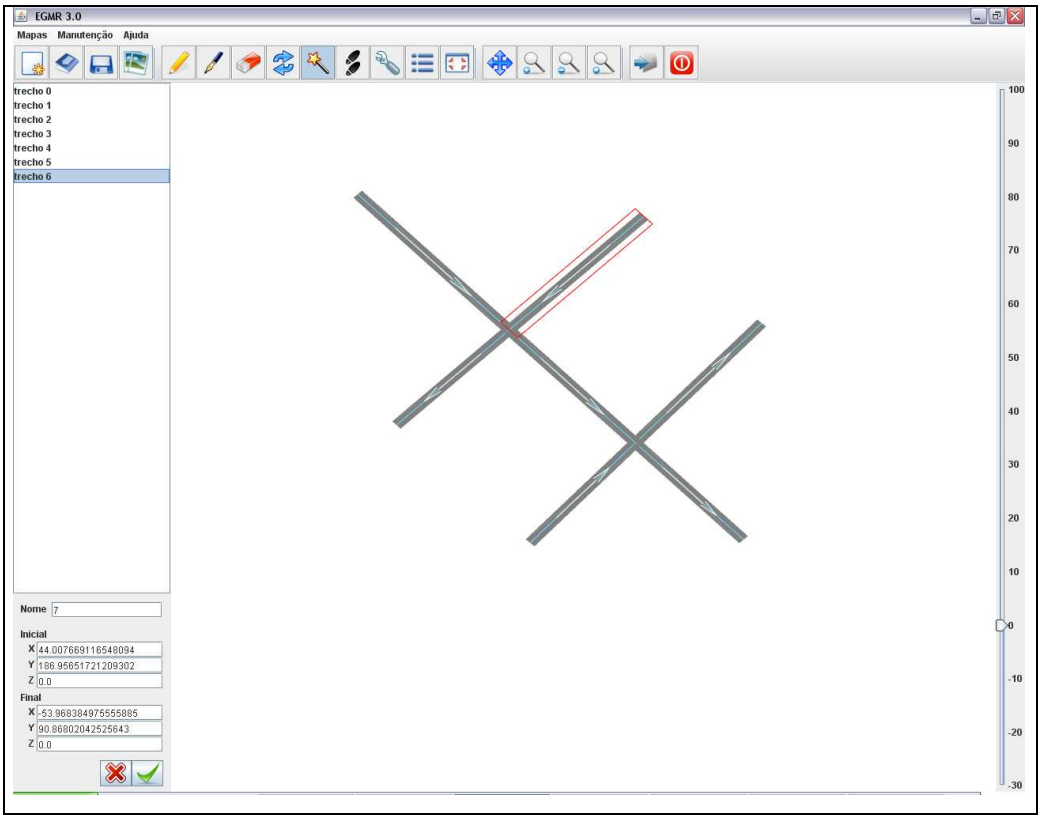


Figura 30 – EGMR no Sistema Operacional Windows

3.3.11 Documentação

Com a finalidade de agilizar e facilitar o entendimento do código em futuras versões do EGMR, foram utilizados os recursos de Java para a documentação do código, o JavaDoc, juntamente com ferramentas do NetBeans para a criação do mesmo.

Esta documentação é adicionada ao código fonte através de comentários especiais e referências a algumas variáveis como o retorno e os atributos dos métodos (Quadro 9).

```
/**
  * Criar novas instâncias de Ponto com coordenadas predefinidas
  * @param x coordenada X do Ponto a ser criado
  * @param y coordenada Y do Ponto a ser criado
  * @param z coordenada Z do Ponto a ser criado
  */
 public Ponto(double x, double y, double z){
     this.x = x;
     this.y = y;
     this.z = z;
 }
```

Quadro 9 - Comentários especiais para a geração do JavaDoc

Posteriormente, são gerados automaticamente vários arquivos em HTML, que podem ser visualizados em qualquer navegador (Figura 31).

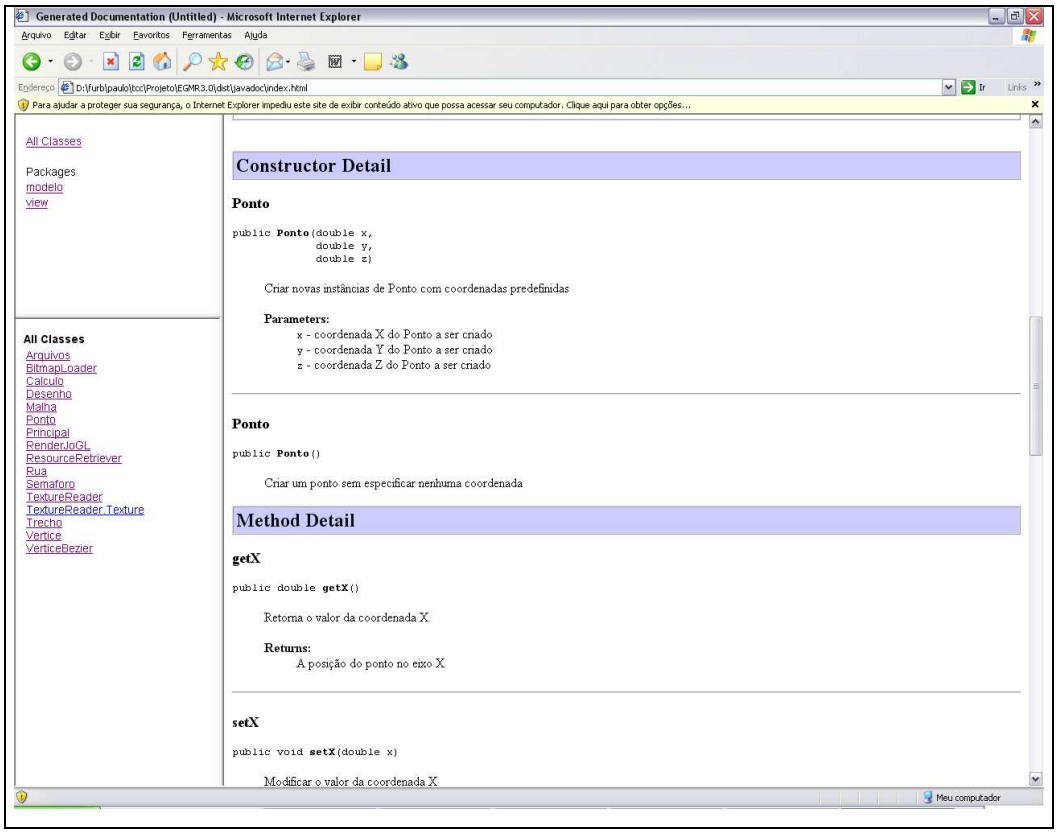


Figura 31 – Documentação HTML da classe Ponto

3.4 RESULTADOS E DISCUSSÃO

Com as novas ferramentas e modificações realizadas no EGMR, as possibilidades de representação aumentaram significativamente. A conversão do código para a linguagem Java, além de permitir sua utilização em diversos sistemas operacionais, oferece uma maior proximidade com uma versão para a internet.

A inclusão de uma imagem para a representação de cenas reais faz com que o software torne-se mais eficaz e próximo da realidade, permitindo uma maior acurácia na criação da malha.

O desenho e a manutenção da malha foram melhorados com a inclusão das ferramentas de Bézier e a de modificação de trechos. Com isso, pode-se fazer o reaproveitamento do desenho, modificando os trechos a fim de criar malhas derivadas e aumentar o poder de representação do software.

O formato do arquivo em XML permite uma melhor leitura e apresentação dos dados, favorecendo o entendimento das informações tanto por parte do usuário como por outros

softwares que venham a fazer uso do EGMR como editor de malhas viárias.

No quesito usabilidade, o programa ganhou uma melhor distribuição das ferramentas, com a possibilidade de identificar todos os trechos da malha a partir da tela principal e rapidamente atualizar suas coordenadas. Além disso, todas as funções, antes só disponíveis através de atalhos do teclado, receberam um botão específico na barra de ferramentas.

Através da inclusão do desenho em 3 dimensões é possível a representação de viadutos, ainda que somente identificados na visualização em 3D.

O Quadro 10 apresenta um comparativo entre as funcionalidades já existentes e aquelas adicionadas no presente trabalho.

Característica/Ferramenta	**EGMR 2.0**	**EGMR 3.0**
Zoom	Sim	Sim
Pan	Sim	Sim
Visualização 3D	Não	Sim
Arquivo da malha em XML	Não	Sim
Ferramenta de Bézier	Não	Sim
Imagem de Fundo	Não	Sim
Multiplataforma	Não	Sim
Grafos	Não	Sim

Quadro 10 - Comparação entre versões do EGMR

De uma forma geral, percebe-se um avanço significativo na usabilidade e abrangência do programa, visto as grandes possibilidades que as ferramentas oferecem.

4 CONCLUSÕES

No presente trabalho foi apresentado o desenvolvimento do EGMR 3.0, com algumas melhorias e adaptações, a fim de torná-lo mais dinâmico e facilitar o desenho de malhas.

O código fonte, além de bastante documentado, foi totalmente convertido para a linguagem Java, com o auxílio da *IDE* NetBeans, sendo que agora sua execução pode ser feita também em outros sistemas operacionais. Para tanto foi necessária a utilização da *API* JoGL, que fornece meios para utilizar as rotinas de *OpenGL* com Java.

A JoGL mostrou-se bastante eficaz e completa, entretanto, as funções do OpenGL somente são disponíveis a partir dos métodos `init, reshape e display.` Por isso, muitas funções, como a localização do clique do *mouse* no contexto da OpenGL, só podem ser realizadas dentro da classe que implementa `GLEventListener`, dificultando a estruturação e a leitura do código.

O formato do arquivo da malha passou a ser *XML*, o que é feito através da biblioteca *XStream*, a qual se mostrou bastante útil e eficaz.

A estrutura da malha passou a representar um grafo, permitindo que trechos sejam movidos e suas ligações com o restante da malha não sejam perdidas.

Os trechos desenhados com a ferramenta de Bézier não são compatíveis com as versões anteriores do EGMR, portanto, quando o arquivo for salvo em formato texto, estas serão perdidas.

Os objetivos do trabalho foram alcançados, entretanto algumas limitações apareceram, e não houve tempo hábil para a sua correta solução. São elas:

a) as curvas de Bézier são trechos independentes e não são conectadas com os trechos normais;
b) não foram obtidos resultados satisfatórios na representação desta curva em 3D, afinal, tratando-se de uma linha, ela não é corretamente visualizada em perspectiva;
c) o cálculo de interseção de um ponto um uma curva também não foi implementado devido a sua complexidade. Com isso não são realizadas quebras de trechos nestes pontos;
d) também não são calculadas interseções entre trechos inclinados, pois muitas verificações precisam ser realizadas. A simples quebra do trecho não é suficiente para corrigir este problema.

O software tem se mostrado bastante eficiente nos testes, permitindo que sejam representados muitas cenas reais, entretanto, para melhorar seu desempenho, é aconselhável que seja utilizado em computadores equipados com uma placa para aceleração gráfica.

4.1 EXTENSÕES

São colocados como sugestões de extensões para este protótipo, os seguintes trabalhos:

a) melhora a integração entre as curvas de Bézier com o restante da malha, bem como a sua visualização em 3D;
b) inserir funcionalidades para o desenho de trechos como várias pistas;
c) incluir a possibilidade de desenho de rotatórias;
d) criar uma versão para a internet;
e) oferecer a possibilidade de desenho sob uma superfície não plana como um terreno com relevo.

REFERÊNCIAS BIBLIOGRÁFICAS

AMORIM, Vinci P. **XStream:** Trabalhando com facilmente XML em Java. [S.l.], [mar. 2007]. Disponível em: <http:// http://www.guj.com.br/java.tutorial.artigo.144.1.guj>. Acesso em: 30 abr. 2007.

BERTOLDI, Gefferson. **Editor gráfico de ruas para o sistema de controle de tráfego de em uma malha rodoviária urbana**. 2005. 56 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

CALIPER CORPORATION. **TransModeler**. Newton, [2006?]. Disponível em: <http://www.caliper.com/transmodeler/default.htm>. Acesso em: 07 set. 2006.

COHEN, Marcelo; MANSSOUR, Isabel H. **OpenGL**: uma abordagem prática e objetiva. São Paulo: Novatec, 2006.

FARINES, Jean-Marie et al. **Projeto SINCMobil**: sistema de informação e controle para mobilidade urbana. Florianópolis, 2003. Disponível em: <http://www.das.ufsc.br/sincmobil/>. Acesso em: 11 set. 2006.

FREIRE, Jocemar J. **Simulação do controle de tráfego de automóveis em uma malha rodoviária urbana**. 2004. 47 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Universidade Regional de Blumenau, Blumenau.

FROESCHLIN, Gustavo E. G. **Editor gráfico de ruas para o sistema de controle de tráfego de em uma malha rodoviária urbana:** versão 2.0. 2006. 73 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

NEON HELIO GAME DEVELOPMENT. **Lesson 06**: textures. [S.l.], [2007?]. Disponível em: <http://nehe.gamedev.net/lesson.asp?index=02>. Acesso em: 28 maio 2007.

JAVA.NET: JoGL API Project. [S.l.], 2006. Disponível em: <https://jogl.dev.java.net/>. Acesso em: 15 set 2006.

PAULA FILHO, Wilson de P. **Multimídia**: conceitos e aplicações. Rio de Janeiro: LTC, 2000.

PREISS, Bruno R. **Estruturas de dados e algoritmos**: padrões de projetos orientados a objetos em Java. 3. ed. Tradução Elizabeth Ferreira Gouvêa. Rio de Janeiro: Campus, 2001.

VELOSO, Renê Rodrigues. **Java e XML**: processamento de documentos. São Paulo: Novatec, 2003.