**Instituto de Ciências Matemáticas e Computação da Universidade de São Paulo**

**SCC0650 – Computação Gráfica**

**Abril de 2017**

**Relatório Técnico referente ao Primeiro Trabalho da Disciplina**

**Henrique Pizzol Grando – nUSP: 8504198**

**Jéssica Bargas Aissa – nUSP: 8504250**

**Lucas Tognoli Munhoz – nUSP: 8504330**

**Introdução**

Este relatório visa esclarecer aspectos técnicos e de construção referentes ao primeiro projeto da disciplina Computação Gráfica, oferecida à Engenharia de Computação. O trabalho implementa o algoritmo de preenchimento de polígonos que utiliza coerência de arestas através de duas estruturas, *ET* (*Edge Table*) e *AET* (*Active Edge Table*). O documento está estruturado da seguinte forma: decisões de projeto, guia para compilação e execução e participação de cada integrante do grupo no trabalho.

**Decisões de Projeto**

**Linguagem de Programação**

A primeira decisão de projeto importante a ser tomada foi sobre qual linguagem de programação usar. Optamos pela linguagem *Java* devido à sua facilidade em se manipular interfaces gráficas, necessárias ao projeto. Tínhamos consciência de que outras linguagens implementavam a mesma interface, porém o grupo possui mais experiência com a linguagem *Java*.

Com relação às bibliotecas utilizadas para a criação e manutenção da *GUI* (*Graphical User Interface*)tem-se a *Java AWT* (*Abstract Window Toolkit*) e *Java Swing*. A primeira apresenta ganhos de performance com relação à última, sendo assim utilizada sempre que possível. No entanto, a segunda possui ferramentas mais poderosas, das quais se fez uso em parte do projeto.

**Estrutura de Classes**

**Polygons**: classe principal do programa, que além de conter a função *main*, cria todos os modelos utilizados na *GUI* através do método *setGUI*, configurando-a.

**DrawCanvas**: classe principal da *GUI* responsável por invocar a *API* que desenha as figuras na tela (inclusive cada pixel do nosso *grid*) e também tratar os eventos de mouse, por exemplo. Ela contém a lista de todos os polígonos adicionados pelo usuário.

**Polygon**: gerencia cada polígono desenhado na tela. Essa classe implementa o algoritmo de preenchimento e também é responsável por construir a *Edge Table* e a *Active Edge Table*.

**Edge**: implementa a estrutura de uma aresta e guarda suas informações para uso no algoritmo de preenchimento, como ymax, xmin e 1/m. Possui também o método invocado a cada rastreio de linhas de escaneamento, que atualiza o valor do x armazenado na tabela.

**EdgeTable**: implementação da *Edge Table*. Usa-se a estrutura HashTable<> do Java. Implementa a ordenação das arestas, utilizada pelo algoritmo.

**Interface Gráfica**

A interface gráfica permite que múltiplos polígonos sejam desenhados ao mesmo tempo. Há também a possibilidade de se mudar a cor de cada um. No entanto, a principal decisão de projeto em relação à *GUI* foi como seria apresentado o *canvas*, ou seja, a tela em que o usuário escolheria os pontos a serem desenhados. Optamos por apresentar um sistema de *grid* variável por questões de didática: o usuário pode escolher o tamanho do *grid* e, com a estrutura claramente quadriculada, pode-se ter a noção exata de quais pixels foram pintados pelo algoritmo.

**Funcionamento do Algoritmo**

- A função *setGUI* é invocada, criando cada elemento na *GUI* e também gerando os *listeners* para os botões.

- O usuário seleciona o tamanho do grid. O método *setGridSize* é invocado na classe *DrawCanvas*.

- Caso o usuário deseje alterar a cor do polígono, o método *setColor* é invocado na classe *DrawCanvas*.

- O usuário clica em um ou mais espaços (pixels) em branco na região quadriculada.

- Para cada clique na grade, um ponto da grade é adicionado ao polígono selecionado (se nenhum polígono foi previamente selecionado, cria um novo).

- O usuário aperta o botão de preenchimento, invocando a função de preenchimento de polígonos para cada polígono diferente.

- *fillPolygon*, por sua vez, através do algoritmo de coerência de arestas, adiciona na lista de pixels todos os pixels daquele polígono que devem ser pintados.

- Caso o polígono desenhado seja inválido (menos de 3 vértices ou 3 vértices consecutivos sobre uma mesma linha), o programa acusa um erro e não desenha o polígono.

- Se o polígono selecionado é válido, chama-se a método *repaint*, que irá chamar (internamente) *paintComponent*.

- A grade é redesenhada e é chamado *paintPolygon* para cada polígono.

- *paintPolygon* percorre as listas de vértices e pixels pintando um retângulo do tamanho dos quadrados da grade que representam um pixel.

**Compilação e Execução**

A aplicação foi desenvolvida em dois sistemas diferentes graças a característica multiplataforma do *Java*. No *Windows*, usou-se a *IDE* *Eclipse* com base no *JRE 1.8*. Os arquivos de projeto do Eclipse foram entregues juntamente com o código do trabalho, caso o corretor queira compilar no *Windows*. Usamos, também, o sistema *MacOS*. Para compilar, há um arquivo *makefile* no diretório do projeto que, além do *Mac*, deve funcionar em sistemas *Linux*.

Para compilação utilizando o *Makefile* basta digitar o comando: *make*

Para executar utilizando o *Makefile* basta digitar o comando: *make run*

**Participação dos Integrantes**

Henrique:

* Implementação da classe *EdgeTable*
* Reformulação da classe *Edge*
* Implementação do método *fillPolygon*
* Implementação do Makefile

Jéssica:

* Implementação inicial das classes: *Polygons, DrawCanvas, Polygon*
* Revisão do método *fillPolygon* e *buildEdgeTable*
* Testes

Lucas:

* Implementação da classe *Edge*
* Implementação do método *buildEdgeTable*
* Documentação.