SCC0250 COMPUTAÇÃO GRÁFICA – TRABALHO 1

Professora Agma Juci Machado Traina

Bruna Zamith Santos (11383109)
bruna.zamith@hotmail.com

Introdução

Este relatório visa apresentar o primeiro trabalho desenvolvido para a disciplina de Computação Gráfica, ministrada pela Prof. Dra. Agma Luci Machado Traina, ao longo do segundo semestre de 2019 da Universidade de São Paulo.

Trata-se da Implementação do algoritmo de Preenchimento de Polígonos (base do ScanLine), em C++ utilizando o Qt Framework, que implementa as seguintes funcionalidades:

1. Definição dos vértices do polígono ao clicar na tela;
2. Seleção da cor de preenchimento;
3. Preenchimento do polígono;
4. Alteração da cor;
5. Alteração de vértices do polígono;
6. Limpeza da tela e definição de novos vértices.

O trabalho foi desenvolvido no Qt framework sob plataforma Linux (Ubuntu 19.04) e está também disponível no repositório [1] do Github.

O restante do documento está definido como segue: Distribuição do Trabalho (descrição dos passos realizados para desenvolvimento do trabalho); Funcionamento da Aplicação (como utilizar a aplicação); Arquitetura da Aplicação (diagrama de classes, descrição das principais classes e descrição sucinta do método que implementa o ScanLine); Capturas de Tela (imagens do sistema funcionando); Compilação e Execução (instruções para compilação e execução do projeto); Informações de Ambiente (descrição do ambiente onde o projeto foi desenvolvido e testado).

Distribuição do Trabalho

O trabalho foi distribuído da seguinte forma:

1. Inicialmente foi implementada a interface da aplicação. Isto é:
    a. A criação dos widgets da janela principal;
    b. Captura da cor definida pelo usuário;
    c. Captura do clique nos botões de controle ("Iniciar", "Encerrar", "Limpar");
    d. Captura dos vértices a partir de cliques na janela;
    e. Mudança de vértice do polígono ("Alterar");
2. Então, foram criadas as classes ETNode, ET e AET, com suas principais funcionalidades;
3. Por fim, foi implementado o algoritmo de preenchimentos de polígonos ScanLine, que faz uso das classes ETNode, ET e AET.

É importante salientar que apesar de ser possível identificar o método "ScanLine", ele é quebrado em outros diferentes métodos para melhor compreensão do código e para possibilitar a identificação das etapas de desenvolvimento.

Funcionamento da Aplicação

A aplicação segue a seguinte lógica:

1. Clique no botão "Iniciar";
2. Clique na tela para selecionar as posições desejadas para os vértices do polígono;
3. Clique no botão "Encerrar" quando terminar de selecionar os vértices;
4. O poligono será desenhado;
5. Caso deseje alterar a cor, selecione uma nova cor no Menu de Cores;
6. Caso deseje alterar a posição de um vértice, digite o valor no vértice (primeiro vértice tem valor 1, segundo vértice tem valor 2, … N-ésimo vértice tem valor N) no campo de "Valor do Vértice" e clique em "Alterar". Selecione a nova posição daquele vértice clicando na tela;
7. Caso deseje limpar a tela e começar novamente, selecione "Limpar".

Interface da Aplicação

A interface da aplicação é composta por uma janela principal (classe MainWindow) e uma subjanela que nela está contida (classe AreaPoligono). Ela é representada na Figura 1.

Figura 1 - Representação da interface da aplicação

Selecionando a cor

Para selecionar a cor, basta usar o *Menu de Cores* drop-down. As opções são: Black, Red, Blue, Cyan, Yellow, Red, Magenta, Gray ou Green. A cor também pode ser alterada depois que o polígono foi desenhado.

Criação dos Vértices

Para começar a criar os vértices, clique no botão Iniciar e dê cliques na janela *AreaPoligono* indicando as posições desejadas. Quando terminar de selecionar todos os vértices, aperte o botão Encerrar. O polígono será desenhado na tela.

Alterando Vértices

Para alterar um vértice após o desenho do polígono, digite o número do vértice (primeiro vértice tem valor 1, segundo vértice tem valor 2, … N-ésimo vértice tem valor N). Não serão aceitas entradas:

1. Que não são números inteiros;
2. Iguais ou menores que zero;
3. Maiores que o número de vertices.

Após inserir o número do vértice, clique no botão Alterar. O vértice selecionado aparecerá na tela. Dê um clique na janela *AreaPoligono* indicando a nova posição desejada daquele vértice.

O polígono será desenhado na tela.

Limpando a tela

Para limpar a tela após o polígono ter sido desenhado, clique no botão Limpar. A tela será limpa e o botão Iniciar voltará a ficar disponível.

Arquitetura da aplicação

A Figura 2 apresenta um diagrama de classes simplificado da aplicação.

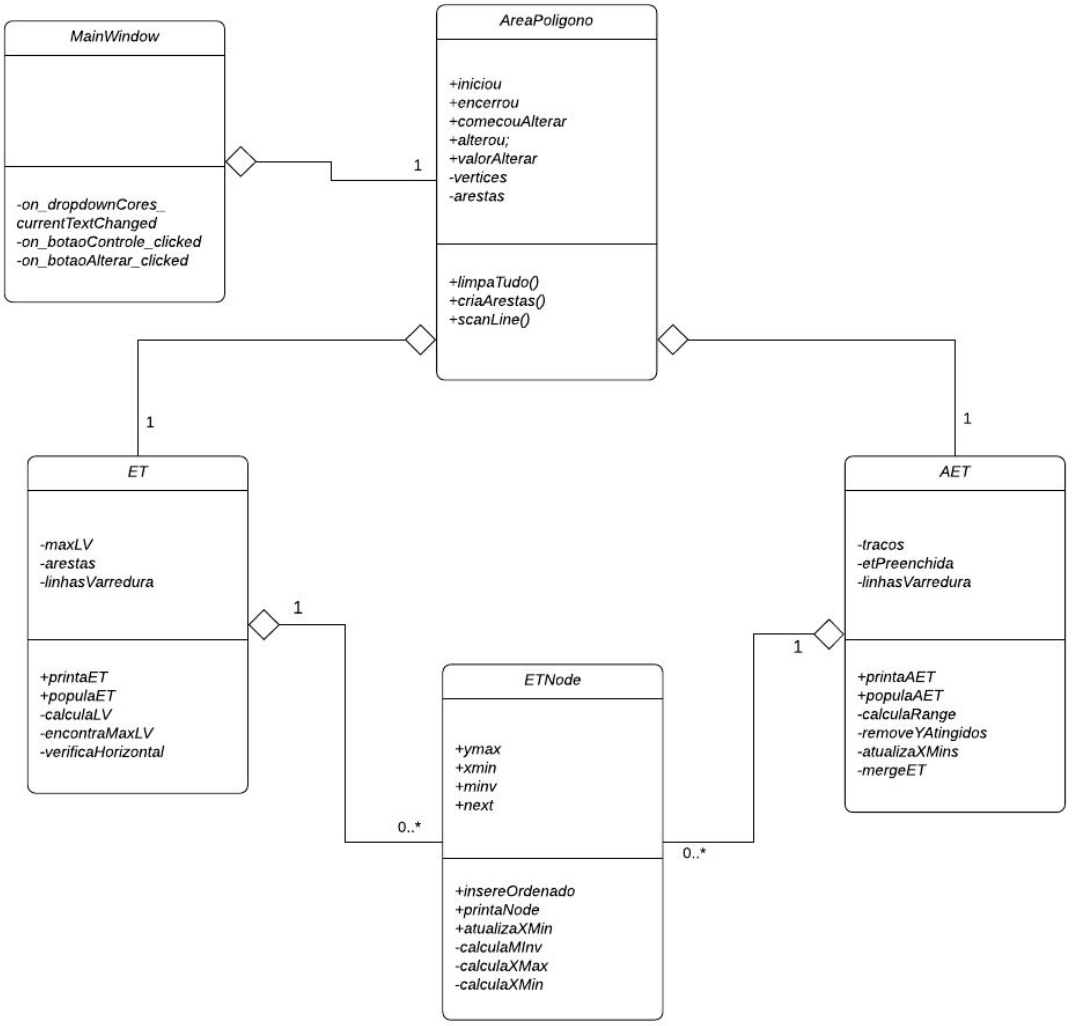


Figura 2 - Diagrama de classes da aplicação

ETNode

A classe ETNode é a classe mais interna do sistema. Ela contém os atributos ***ymax***, ***xmin***, ***minv*** (1/m) e um ponteiro para outra ETNode, chamado ***next***.
Devido ao ponteiro ***next***, ela pode ser interpretada como uma lista ligada.
Seu construtor recebe uma ***aresta*** e, a partir dela, calcula ***ymax***, ***xmin*** e 1/m (***mInv***) (métodos **calculaYMax**, **calculaXMin**, **calculaMInv**, respectivamente). O ponteiro ***next*** é setado como nulo;
O método **insereOrdenado** é equivalente à função de inserir um nó numa LinkedList, mas o faz de maneira ordenada (pelo valor de ***xmin***).

O método **printaNode** também é equivalente à função de percorrer uma LinkedList, mas printa informações de cada nó enquanto a percorre.
O método **atualizaXMin** faz a soma de ***xmin*** atual com ***minv*** sempre que é invocado.

ET

A classe ET é, resumidamente, um vetor de ponteiros para ETNodes (atributo ***linhasVarredura***).
Seu construtor recebe um vetor de ***arestas***, encontra a linha de varredura máxima (método **encontraMaxLV**) e inicializa a ET com a quantidade de posições igual à linha de varredura máxima calculada. Cada posição recebe um ponteiro nulo.
O método **populaET**:

1. Para cada aresta do vetor de ***arestas***, verifica se a aresta em questão é horizontal a partir do método **verificaHorizontal**. Em caso negativo, prossegue;
2. Cria um objeto do tipo ETNode;
3. Invoca o método **calculaLV**, para encontrar qual a linha de varredura que primeiro toca aquela aresta. Armazena o valor na variável ***level***;
    a. Se o valor correspondente de ***level*** na ET for nulo, ele passa a apontar para o ETNode criado;
    b. Se não for nulo, é chamado o método **insereOrdenado** da classe ETNode. Cada posição da ET funciona como uma lista ligada de objetos ETNode;
4. Retorna quando todas as arestas do vetor de **arestas** tiverem sido percorridas.

AET

A classe AET se baseia na ET (atributo ***etPreenchida***) para analisar as linhas de varredura (atributo ***linhasVarredura***) e determinar os intervalos de preenchimento (atributo ***tracos***).
O método **populaAET**:

1. Copia a primeira linha de varredura da ET na AET;
2. Remove a linha da ET;
3. Chama o método **calculaRange**, que encontra o intervalo de pontos a ser desenhado e o inclui no vetor de ***tracos*** da classe AET;
4. Enquanto a ET e a AET não estiverem vazias:
    a. Remove da AET os ETNodes cujo ***ymax*** é igual à linha de varredura atual (método **removeYAtingidos**);
    b. Atualiza os valores de ***xmin*** da AET. Importante salientar que o método **atualizaXMins** faz uso das funções **ceil** e **floor** para respeitar a regra do algoritmo ScanLine em relação às arestas que pertencem ao polígono (inferior e esquerda);
    c. Junta a linha de varredura (LV) da AET atual com a linha de varredura correspondente na ET ao invocar o método **mergeET**:
        i. Se aquela LV da ET não é um ponteiro nulo, prossegue;
        ii. Se aquela LV da AET está vazia, simplesmente copia a LV correspondente da ET;
        iii. Senão, junta as duas com o método **insereOrdenado** explicado anteriormente;

    d. Chama o método **calculaRange**, que encontra o intervalo de pontos a ser desenhado e o inclui no vetor de ***tracos***.
5. Retorna o vetor de ***tracos***.

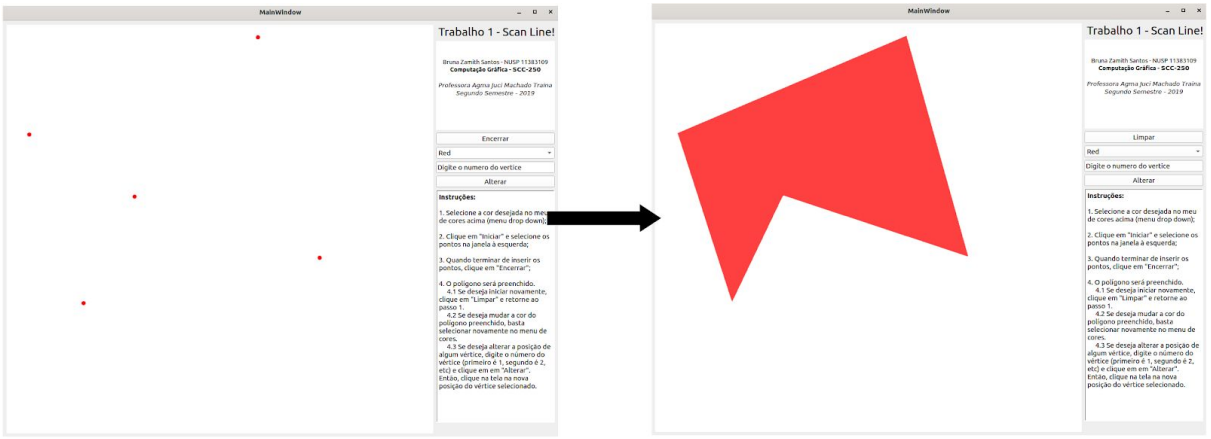
Scan Line

O método **scanLine** pertence à classe AreaPoligono e funciona da seguinte forma:

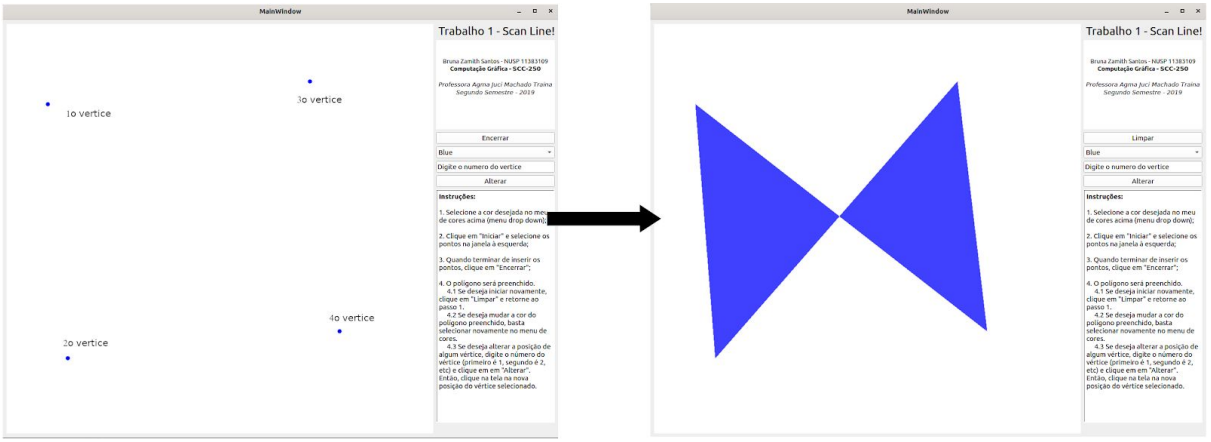
1. Verifica se existe mais de uma aresta no vetor de ***arestas*** da classe AreaPoligono e só prossegue em caso afirmativo;
2. Cria um objeto do tipo ET.
3. Invoca o método **populaET** da classe ET (descrito anteriormente):
4. Cria um objeto do tipo AET;
5. Invoca o método **populaAET** da classe AET (descrito anteriormente):
6. O vetor de ***arestas*** agora é o vetor de ***tracos*** retornado;
7. Chama o método **update**, responsável por chamar o método **paintEvent:**
    a. O pincel de desenho QPainter recebe a ***cor*** definida no *Menu de Cores*;
    b. Se a flag ***comecouAlterar*** está desativada (ou seja, o usuário não está fazendo nenhuma alteração):
        i. Se os vetor de ***vertices*** não está vazio e a flag ***encerrou*** está desativada, chama o método nativo do Qt **drawPoints**, passando o vetor de ***vertices*** como parâmetro;
        ii. Se os vetor de ***arestas*** não está vazio e a flag ***encerrou*** está ativada, chama o método nativo do Qt **drawLines**, passando o vetor de ***arestas*** como parâmetro.

Capturas de Tela

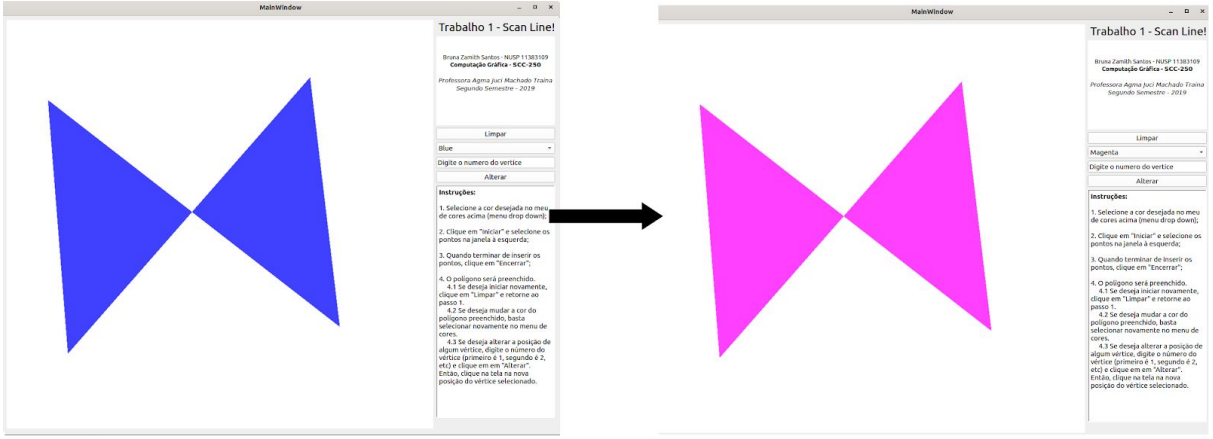
A seguir, algumas imagens do sistema em funcionamento:



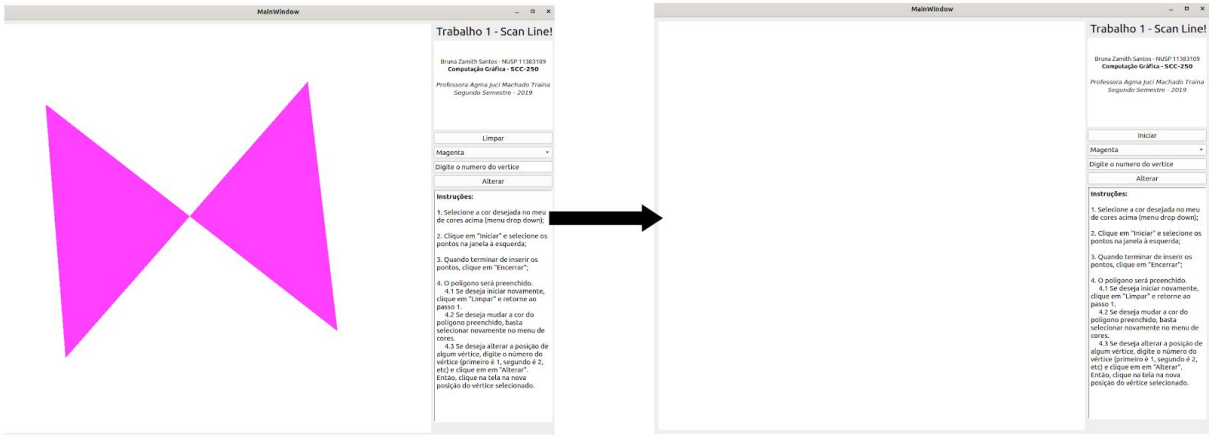
Captura 1 - Desenho de um polígono sem auto intersecção.



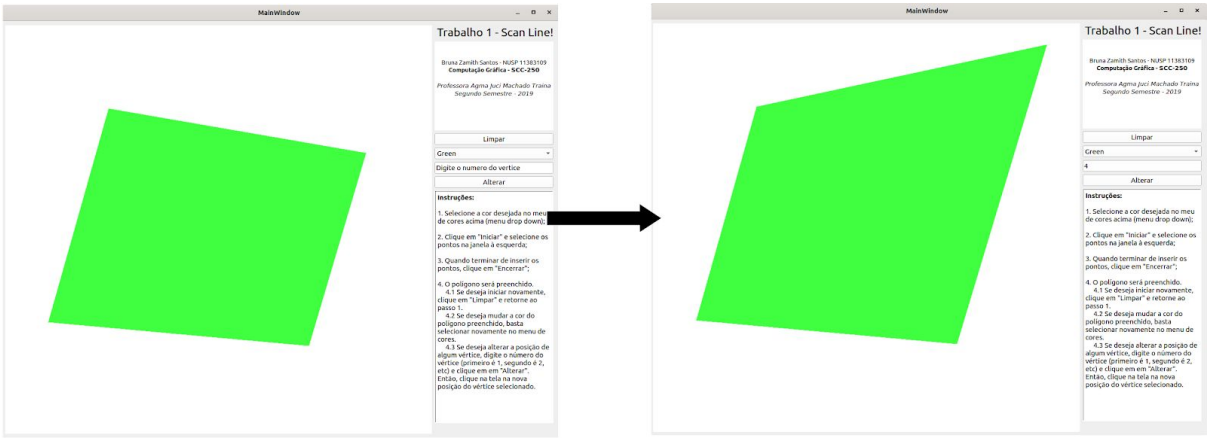
Captura 2 - Desenho de um polígono com auto intersecção.



Captura 3 - Demonstração da mudança de cor do polígono já desenhado.



Captura 4 - Demonstração da limpeza do polígono desenhado.



Captura 5 - Demonstração da alteração do quarto vértice do polígono.

Compilação e Execução

Faça o download do projeto disponibilizado no escaninho do Tidia da disciplina ou no repositório [1] do Github.
Dentro do diretório do projeto, executar:
$ qmake Trabalho1.pro
$ make Trabalho1
$ ./Trabalho1

O sistema então será executado.

Informações de Ambiente

O projeto foi desenvolvido e testado num ambiente com as seguintes configurações:

Qt 5.13.1 (x86_64-little_endian-lp64 shared (dynamic) release build; by GCC 5.3.1 20160406 (Red Hat 5.3.1-6)) on "xcb"
OS: Ubuntu 19.04 [linux version 5.0.0-29-generic]

Architecture: x86_64; features: SSE2 SSE3 SSSE3 SSE4.1 SSE4.2 AVX AVX2

Compiler: GCC (C++), x86 64bit

Environment:
QT4_IM_MODULE="xim"
QT_ACCESSIBILITY="1"
QT_DEVICE_PIXEL_RATIO="auto"
QT_IM_MODULE="ibus"

Features: QT_NO_EXCEPTIONS

Platform capabilities: ThreadedPixmaps OpenGL ThreadedOpenGL WindowMasks MultipleWindows ForeignWindows NonFullScreenWindows NativeWidgets WindowManagement SyncState RasterGLSurface SwitchableWidgetComposition

LibGL Vendor: Intel Open Source Technology Center
Renderer: Mesa DRI Intel(R) HD Graphics 630 (Kaby Lake GT2)
Version: 3.0 Mesa 19.0.2
Shading language: 1.30
Format: Version: 3.0 Profile: 0 Swap behavior: 0 Buffer size (RGB): 8,8,8
Profile: None (QOpenGLFunctions_3_0)

Qt Creator 4.10.0
Based on Qt 5.13.1 (GCC 5.3.1 20160406 (Red Hat 5.3.1-6), 64 bit)
From revision 9b7bab7d35
Built on Sep 4 2019 04:49:18