RELATÓRIO

Nome: Daniela Rigoli

Tema: Inclusão de Ponto em Polígonos

Enunciado: O programa deverá ler um polígono descrito em um arquivo e exibi-lo na tela. A partir disto deverá ser gerado um conjunto de pontos que serão classificados como dentro ou fora do polígono lido.

- o **Convex Hull** do polígono. O polígono gerado deverá ser exibido na tela com uma cor diferente do polígono lido;
- um conjunto de 10 faixas dividindo o espaço ao longo do eixo Y e cadastrando, em cada faixa, o número das arestas que passam naquela faixa. As faixas deverão ser exibidas na tela.

Os pontos que estiverem dentro do polígono deverão ser exibidos em **azul** e os que estiverem fora do polígono, mas dentro do Convex Hull em **amarelo** e os que estiverem fora do Convex Hull, em **vermelho**. Deverão ser implementados os seguintes algoritmos de teste de inclusão:

- Força bruta, calculando o número de de interseções;
- Teste de inclusão em polígono convexo, usando o Convex Hull;
- Teste de força bruta, considerando apenas as arestas que estão na faixa onde fica o ponto.

Para a entrega do trabalho deverá ser gerado um relatório que demonstre o comportamento dos algoritmos com 200, 2000 e 20.000 pontos. Este comportamento deverá ser descrito em termos de :

- o tempo de execução de cada algoritmo
- número de vezes que a função **HaIntersecao** foi chamada;
- número de vezes que a função **ProdVetorial** foi chamada.

Resposta: Neste trabalho foi utilizado um algoritmo para verificar quando há colisão com as arestas de um polígono. Conforme solicitado foi implementado um método de força bruta no qual é verificado para todas as arestas de um polígono passado por parâmetro e para evitar redundância o mesmo método pode ser usado para o teste de inclusão da convex hull, apenas alterando o polígono passado.

O algoritmo de convex hull, localizado na classe Poligono, busca os limites do polígono e a partir disso encontrar o ponto inicial no limite do menor valor de y, começa a comparar qual o maior produto escalar entre as possíveis próximas arestas e o vetor direita até atingir o limite do maior x. Ao atingir o limite do maior x continuará verificando o maior produto escalar até atingir o limite do maior y e depois do menor x. Esse método retorna um novo polígono.

O último algoritmo para gerar as faixas é chamado no init fazendo o cadastro de todas as arestas nas suas respectivas faixas. Para isso se passa por todas as arestas verificando em quais faixas ela se está. Sendo verificado se um dos pontos está na faixa ou se a aresta atravessa a faixa, para fins de auxiliar no processo de identificar se um ponto está dentro do polígono essas faixas foram feitas de forma inclusiva em relação a vértices em cima da linha da faixa. Para usar as faixas quando se recebe um ponto foi feito um método para calcular em qual faixa está esse

ponto, após isso é passado por todas arestas na faixa verificando se há intersecção entre o ponto considerando uma linha para esquerda em relação às arestas dessa faixa.

Para fins de facilitar na geração dos dados para o relatório foi adicionado no teclado que quando o usuário digitar ‘0’ será definido o algoritmo de força bruta, ‘1’ será utilizado a convex hull e ‘2’ será utilizado as faixas para o polígono original, para que quando clicar em um ponto ou quando for gerar o relatório de tempo do algoritmo seja utilizado esse algoritmo. O mesmo foi feito para o número de pontos que será gerado aleatoriamente, sendo ‘a’ para 200, ‘b’ para 2000 e ‘c’ para 20000. Caso não seja selecionado, é definido como padrão o algoritmo de força bruta e 200 pontos gerados.

Além de um método que faz a integração desses algoritmos, recebendo um ponto e verificando se está dentro da convex hull e caso positivo verificando se está de fato dentro do polígono utilizando as faixas para ser mais otimizado e passando para um polígono que irá exibir na tela esse ponto com a cor desejada.

Código desenvolvido:

// Classe poligono.cpp

```
Poligono Poligono::GeraConvexHull()
{
    Poligono convex;
    Ponto PMin, PMax;
    obtemLimites(PMin, PMax);

    // localiza o ponto inicial para gerar o convex
    Ponto Start;
    // procura o menor ponto em y e começa com produto escalar pro vetor (1,0)
    for(int i = 0; i < Vertices.size(); i++)
    {
        if(PMin.y == getVertice(i).y){
            Start = getVertice(i);
            break;
        }
    }
    Ponto VetorComparacao = Ponto (1.0f, 0.0f, 0.0f); // direita
    Ponto VetorTemp;
    Ponto Temp = Start;
    Ponto ProxPonto;
    do{
        if(Temp.x == PMin.x){
```

```
            VetorComparacao = Ponto (0.0f, -1.0f, 0.0f); // pra baixo -terceiro que deve
acontecer
        } else if(Temp.y == PMax.y){
              VetorComparacao = Ponto (-1.0f, 0.0f, 0.0f); // pra esquerda -segundo que
deve acontecer
        }
        else if(Temp.x == PMax.x){
             VetorComparacao = Ponto (0.0f, 1.0f, 0.0f); // pra cima -primeiro que deve
acontecer
        }
        double ProdEsc = 0;
        for(int i = 0; i < Vertices.size(); i++)
        {
            // calcular o vetor unitario
            // 1. gerar um vetor do ponto inicial ate os outro:
            if( PontosIguais(Temp, getVertice(i)))// evitar divisão por zero
                exit;
            VetorTemp = operator-(Temp, getVertice(i));
            VetorTemp = VetorUnitario(VetorTemp);
            double TempProdEsc = ProdEscalar(VetorTemp, VetorComparacao);
            //cout << "ProdEsc: " << TempProdEsc << " | "<< ProdEsc << endl;
            if(TempProdEsc > ProdEsc){
                ProdEsc = TempProdEsc;
                ProxPonto = getVertice(i);
                //cout << "ProxPont: " << ProxPonto.x << " | "<< ProxPonto.y << endl;
            }



        }
        convex.insereVertice(Temp);
        Temp = ProxPonto;
    } while( PontosIguais(Temp, Start) == FALSE); // enquanto nao der a volta
    return convex;
}
```

// Classe ExibePoligonos.cpp

```
int NumFaixas = 10;
// ************************************************************************
//  Cadastra todas as arestas nas faixas necessárias
//  Percorre todas as arestas vendo em quais faixas ele se encontra
// ************************************************************************

void CadastraTodasArestas(Poligono &Pol, Ponto &Max, Ponto &Min){

   Ponto P1, P2;
```

```
    double tamFaixa = (Max.y - Min.y)/NumFaixas;
    cout << "Cadastra " << Min.y << " | " << Max.y << " | " << tamFaixa << endl;
    for (int i=0; i < Pol.getNVertices();i++)
    {
        Pol.getAresta(i, P1, P2);
        int faixa = 0;
        for(double j = Min.y; j < Max.y; j+= tamFaixa){
            if(P1.y >= j && P1.y <= j + tamFaixa){ // P1 dentro da linha
                EspacoDividido.CadastraArestaNaFaixa(faixa ,i);
            } else if(P2.y >= j && P2.y <= j + tamFaixa){ // P2 dentro da linha
                EspacoDividido.CadastraArestaNaFaixa(faixa ,i);
            } else if(P1.y <= j && P2.y >= j + tamFaixa){ // quando os pontos atravessam a faixa
                EspacoDividido.CadastraArestaNaFaixa(faixa ,i);
            } else if(P2.y <= j && P1.y >= j + tamFaixa){
                EspacoDividido.CadastraArestaNaFaixa(faixa ,i);
            }
            faixa++;
        }
    }
}
Ponto P1, P2;
bool NoSegmento; // variavel para ver se o ponto esta em cima da aresta
// ************************************************************************
//  Algoritmo que passa por todas os vertices do poligono vefiricando suas colisoes com o vetor gerado
//  a partir Ponto Inicial (clicado ou chamado) indo para a esquerda
//  Tambem conta em quantos vertices bateu para tratar alguns casos especiais
// ************************************************************************
void ForcaBruta(int &Colisoes, int &ColidVertice, Ponto &PI, Ponto &Esq, Poligono &Pol){
    for (int i=0; i < Pol.getNVertices();i++)
    {
        Pol.getAresta(i, P1, P2);
        if(P1.y != P2.y){
            if (HaInterseccao(PI,Esq, P1, P2)){
                Colisoes++;
                if(ProdEscalar(P1-P2, PI-P2)==0)
                    NoSegmento = true;
                if(PI.x == P1.x && PI.y == P1.y){
                    ColidVertice++;
                } else if(PI.y == P2.y){
                    ColidVertice++;
                }
```

```
        }
      }
   }
}
// ponto da dentro do poligono naquela faixa
void PontoNaFaixa(int &Colisoes,int &ColidVertice, Ponto &PI, Ponto &Esq, Faixa &F){
   for (int i=0; i < F.getNroDeArestas();i++)
   {
      Mapa.getAresta(F.getAresta(i), P1, P2);
      if(P1.y != P2.y){
         if (HaInterseccao(PI,Esq, P1, P2)){
            Colisoes = Colisoes + 1;
            if(ProdEscalar(P1-P2, PI-P2)==0)
               NoSegmento = true;
            if(PI.y == P1.y){
               ColidVertice++;
            } else if(PI.y == P2.y){
               ColidVertice++;
            }
         }
      }
   }
}
// *********************************************************************
//  Descobre em quais faixa o ponto esta usando regra de 3
// *********************************************************************
Faixa DescobreFaixa(Ponto &PI){
   int faixa = (PI.y - Min.y) * NumFaixas / (Max.y - Min.y);
   return EspacoDividido.getFaixa(faixa);
}
// *********************************************************************
// PontoIncluso retorna verdadeiro para o ponto incluso e falso para quando o ponto fora do poligono de acordo com o algorimo selecionado
//  Este algorimo tambem facilita na geracao do relatorio e teste de cada algoritmo por pode ao ser chamado verifica o algorimo selecionado
// *********************************************************************
bool PontoIncluso(Ponto P){
   int Colisoes=0;
   int ColidVertice=0;
   NoSegmento = false;
   Ponto Esq;
```

```
Ponto Dir (-1,0);
Esq = P + Dir * 100;
// forca bruta:
if(Algoritmo == 0)
    ForcaBruta(Colisoes,ColidVertice, P, Esq, Mapa);
// convex hull:
if(Algoritmo == 1)
    ForcaBruta(Colisoes,ColidVertice, P, Esq, ConvexHull);
// forca bruta considerando faixas
if(Algoritmo == 2){
    Faixa F = DescobreFaixa(P);
    PontoNaFaixa(Colisoes,ColidVertice, P, Esq, F);
}
P.imprime();
cout << "colisoes: " << Colisoes << endl;
if(NoSegmento){
    cout << "Dentro do poligono" << endl;
    return true;
}
ColidVertice = ColidVertice/2;
if(ColidVertice == 0){
    if(Colisoes % 2 == 0){
        cout << "Fora do poligono" << endl;
        return false;
    } else if(Colisoes % 2 != 0){
        cout << "Dentro do poligono" << endl;
        return true;
    }
}
if(Colisoes % 2 == 0 && ColidVertice % 2 == 0 ){
    cout << "Dentro do poligono" << endl;
    return true;
} else if(Colisoes % 2 != 0 && ColidVertice % 2 != 0 ){
    cout << "Dentro do poligono" << endl;
    return true;
} else {
    return false; // paridade distinta entre Colidesões e colisões em vertice
}
```

```
}
// ************************************************************************
//  void display( void )
// ************************************************************************
void display( void )
{
[...]
    //pintar
    glColor3f(1,0,0); // RGB
    Fora.desenhaVertices();
    glColor3f(1,1,0); // RGB
    Hull.desenhaVertices();
    glColor3f(0,0,1); // RGB
    Dentro.desenhaVertices();
    if (FoiClicado == true)
    {
        PontoIncluso(PontoClicado);
        FoiClicado = false;
    }
[...]
}
#include <bits/stdc++.h>
float random(float min, float max){
    float result = ( min + (float)rand() / (float)(RAND_MAX / (max - min + 1) + 1));
    return result;
}


// ************************************************************************
//  Gera pontos aleatorios para testes com 200, 2000 e 20000 pontos
// ************************************************************************
void GeradorDePontos(Poligono &listaDePontos, int numPontos){
    Ponto p;
    //cout << "Limites de X: " << Min.x << " | "<< Max.x<< endl;
    //cout << "Limites de Y: " << Min.y << " | "<< Max.y<< endl;

    for(int i = 0; i < numPontos; i++){
        p = Ponto( random(Min.x, Max.x), random(Min.y, Max.y));
        listaDePontos.insereVertice(p);
        //p.imprime();
    }
```

```
}


// ***********************************************************************
//  Algoritmo que integra os algorimos estudados para visualizacao na apresentacao
// ***********************************************************************

int numPontos = 5;

void Integracao(){

   Poligono Pontos;

   GeradorDePontos(Pontos, numPontos);

   T.getDeltaT();

   for(int i = 0; i < Pontos.getNVertices(); i++){

      Algoritmo = 1; // usando convex

      if(PontoIncluso(Pontos.getVertice(i))){

         Algoritmo = 2;

         if(PontoIncluso(Pontos.getVertice(i))){

            Dentro.insereVertice(Pontos.getVertice(i));

         } else {

            Hull.insereVertice(Pontos.getVertice(i));

         }

      } else {

         Fora.insereVertice(Pontos.getVertice(i));

      }

   }

   T.getDeltaT();

   cout << "Numero de vertices fora: " << Fora.getNVertices() << endl;
   cout << "Numero de vertices dentro da Hull: " << Hull.getNVertices() << endl;
   cout << "Numero de vertices dentro: " << Dentro.getNVertices() << endl;
   cout << "O tempo para os algoritmos integrados foi de: " << T.getDeltaT() << endl;

}

// ***********************************************************************
//  Algoritmo que retorna as informações para o relatório
// ***********************************************************************

void Relatorio(){

   resetContadorInt();

   Poligono listaDePontos;

   GeradorDePontos(listaDePontos, numPontos);


   T.getDeltaT();

   // relatorio para o algoritmo de forca bruta

   for(int i = 0; i < listaDePontos.getNVertices(); i++){

      PontoIncluso(listaDePontos.getVertice(i));
```

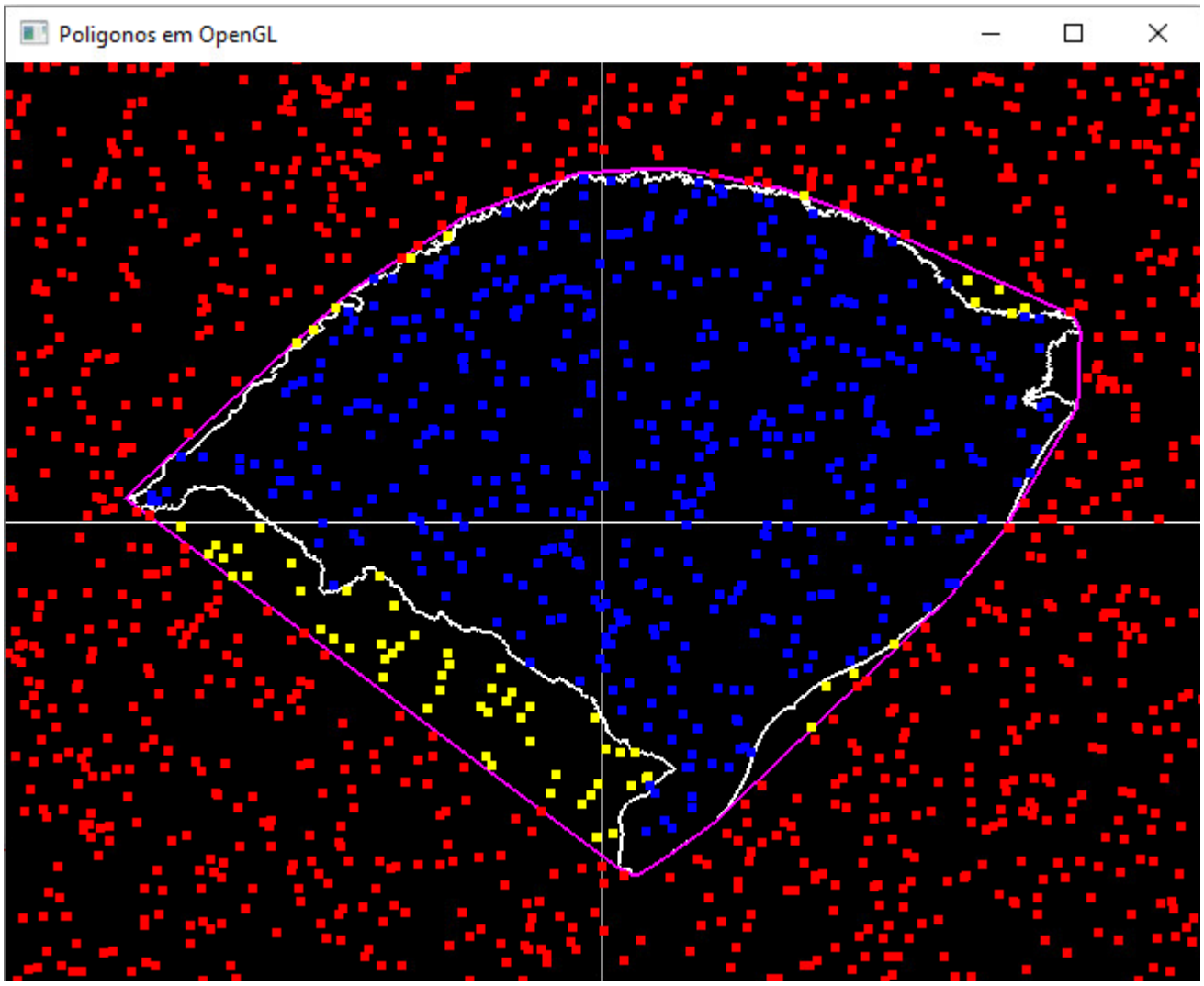
```
        //cout << "passei ponto incluso" << endl;
    }
    cout << "Vezes na funcao HaIntersecao: " << getContadorIntHaInters() << endl;
    //cout << "Vezes na funcao ProdVetorial: " << getContadorIntProdVet() << endl;
    cout << "Vezes na funcao ProdEscalar: " << getContadorIntProdEsc() << endl;
    cout << "O tempo para o algoritmo " << Algoritmo << " com " << numPontos << " pontos foi de " << T.getDeltaT() <<
endl;
}
void keyboard ( unsigned char key, int x, int y )
{
    Temporizador T;
            switch ( key )
            [...]
        case '0':
            Algoritmo = 0;
            cout << "Sera usado algoritmo de força bruta | " << Algoritmo << endl;
        break;
        case '1':
            Algoritmo = 1;
            cout << "Sera usado algoritmo com a convex hull | " << Algoritmo << endl;
        break;
        case '2':
            Algoritmo = 2;
            cout << "Sera usado algoritmo de faixas | " << Algoritmo << endl;
        break;
        case 'a':
            numPontos = 200;
            cout << "O numero pontos para o proximo relatorio sera de " << numPontos << endl;
        break;
        case 'b':
            numPontos = 2000;
            cout << "O numero pontos para o proximo relatorio sera de " << numPontos << endl;
        break;
        case 'c':
            numPontos = 20000;
            cout << "O numero pontos para o proximo relatorio sera de " << numPontos << endl;
        break;
        case 'f':
            cout << "Processo com todos os algoritmos..." << endl;
            Integracao();
```

```
        break;
        case 'r':
            cout << "Gerando relatorio..." << endl;
            Relatorio();
        break;
        default:
        break;
            }
}
```

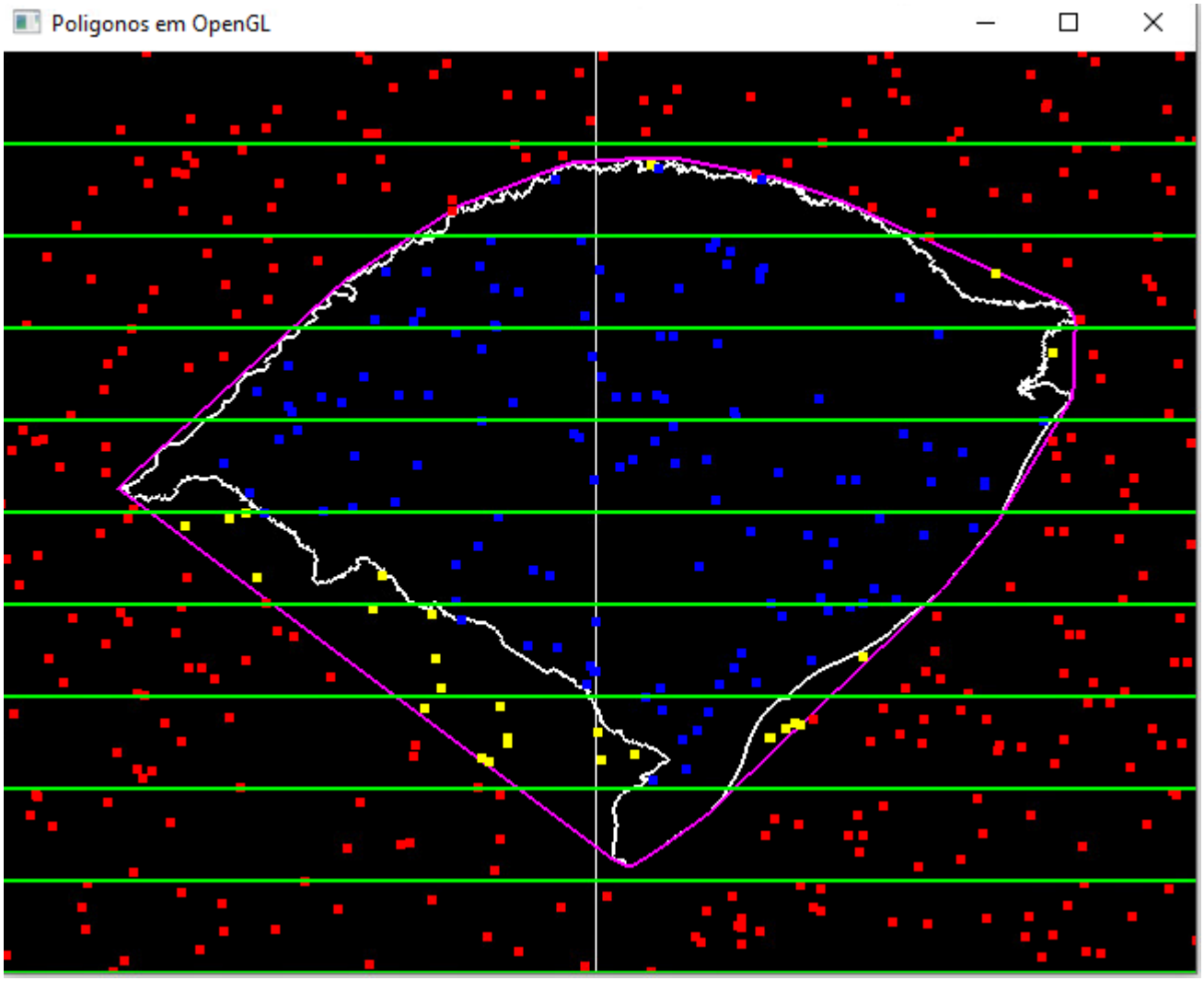
Exemplos de Processamento:

O Convex Hull do polígono. O polígono gerado deverá ser exibido na tela com uma cor diferente do polígono lido;

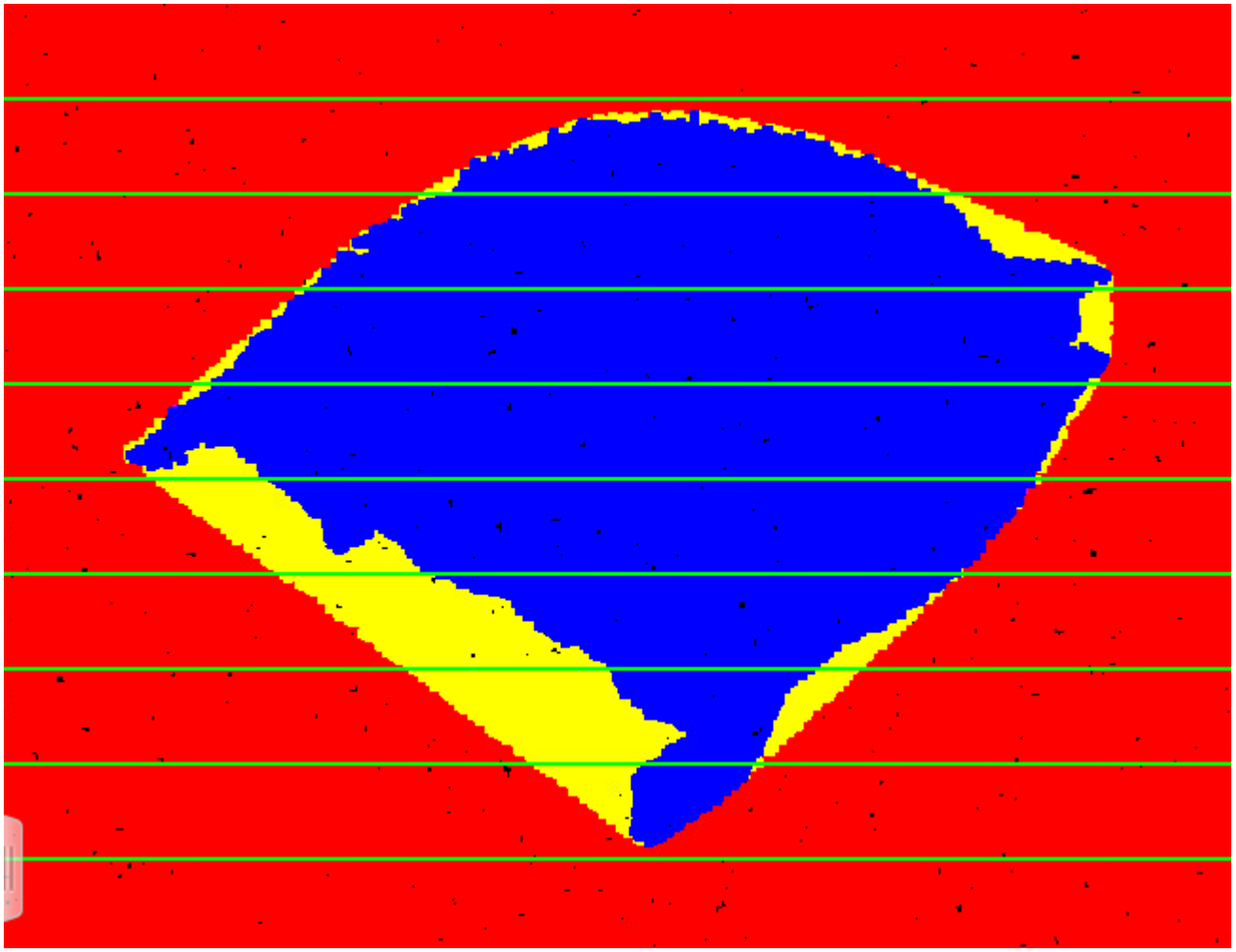
Os pontos que estiverem dentro do polígono deverão ser exibidos em azul e os que estiverem fora do polígono, mas dentro do Convex Hull em amarelo e os que estiverem fora do Convex Hull, em vermelho.



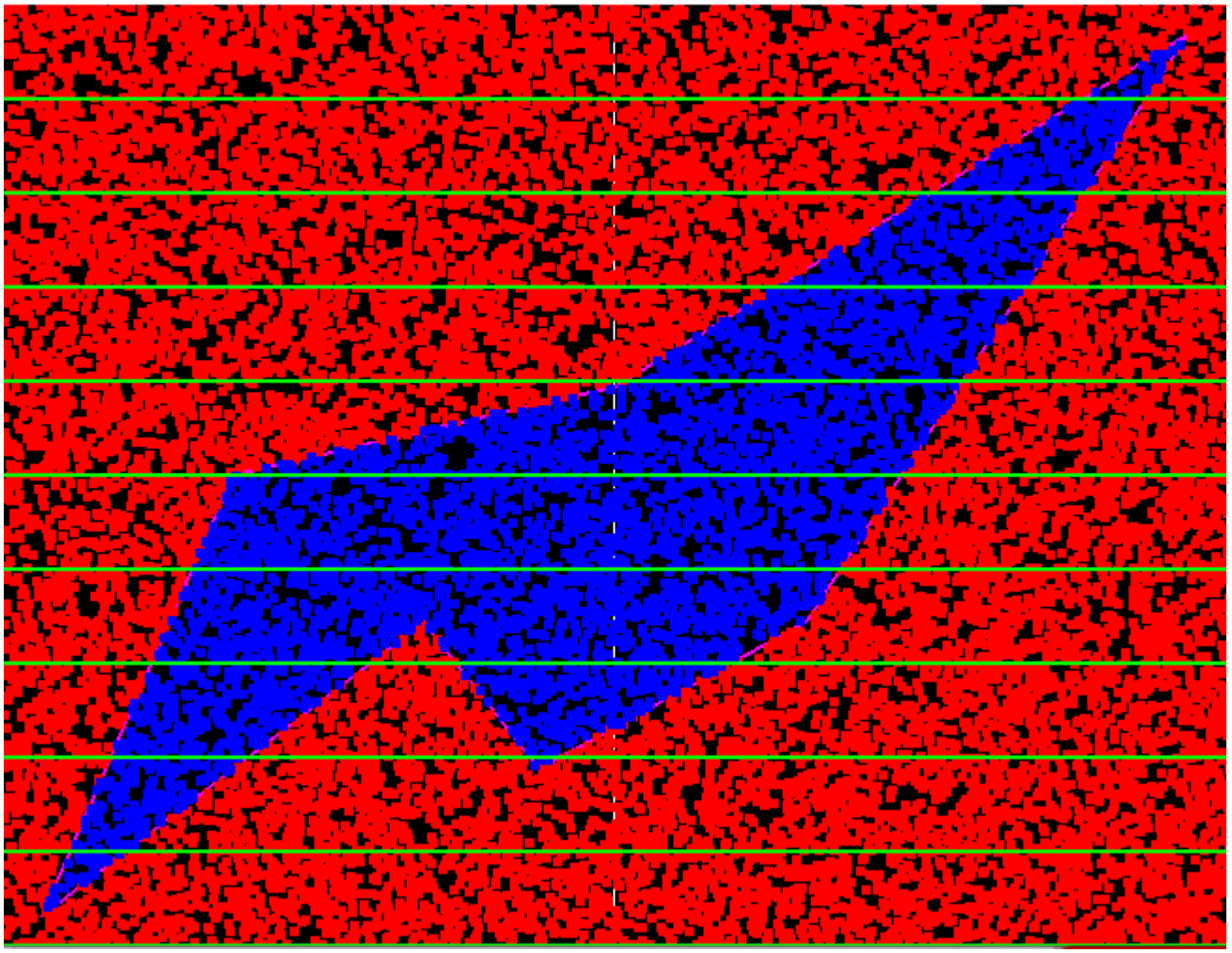
Um conjunto de 10 faixas dividindo o espaço ao longo do eixo Y e cadastrando, em cada faixa, o número das arestas que passam naquela faixa. As faixas deverão ser exibidas na tela.



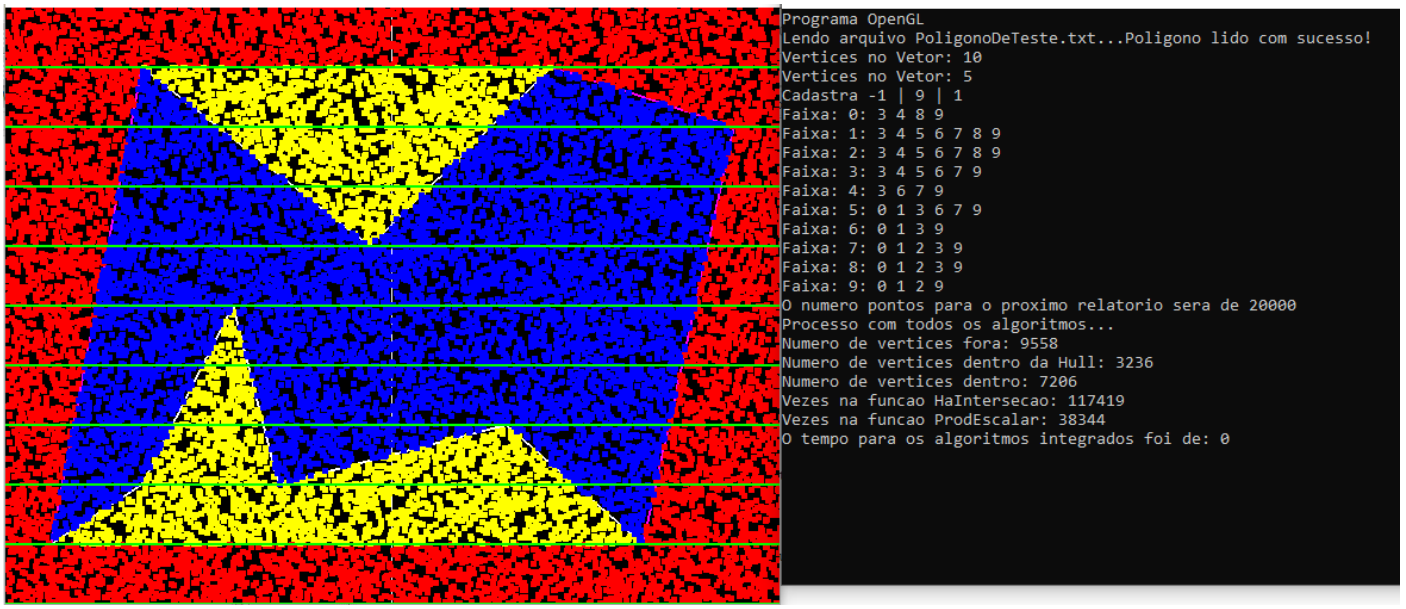
Preenchendo o mapa:



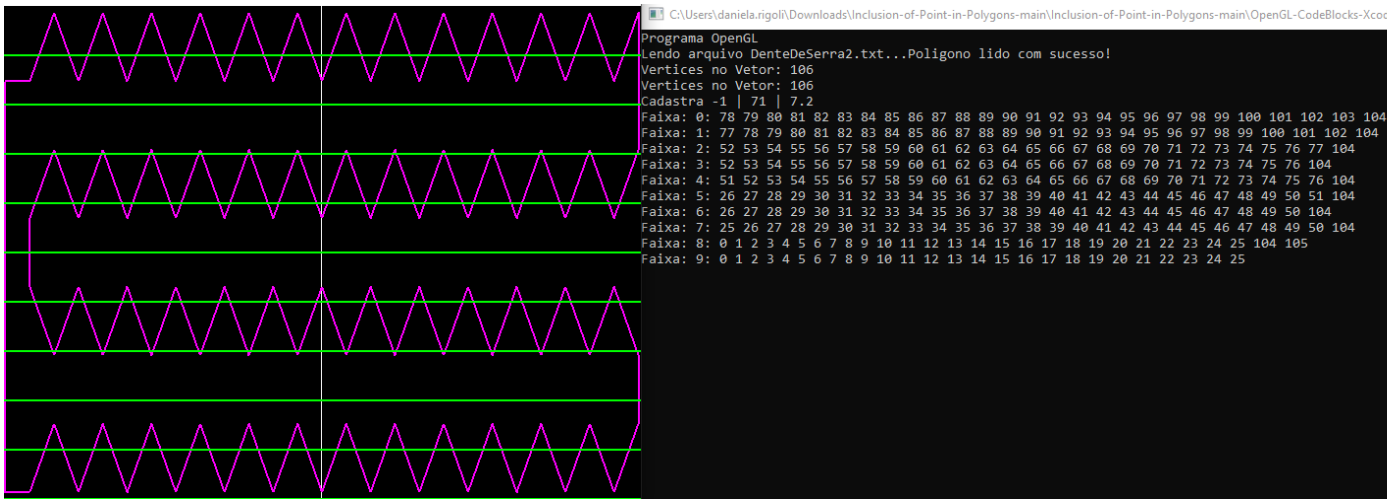
Para o “PoligonoDeTest2.txt”



Para o “PoligonoDeTeste.txt”:



Para o “DenteDeSerra.txt”:



Desempenho para os casos:

com 200, 2000 e 20.000 pontos. Este comportamento deverá ser descrito em termos de :

- o tempo de execução de cada algoritmo
- número de vezes que a função HaIntersecao foi chamada;
- número de vezes que a função ProdVetorial foi chamada.

Algoritmo	Pontos	Tempo de Execução (segundos)	HaIntersecao	ProdVetorial	ProdEscalar
Força bruta	200	0.015	436000	0	204
Força bruta	2000	0.188	4360000	0	2328
Força bruta	20000	1.985	43600000	0	22183
Convex Hull	200	0.000	7000	0	141
Convex Hull	2000	0.000	70000	0	1425
Convex Hull	20000	0.032	700000	0	14433
Integrado	200	0.000	30890	0	284
Integrado	2000	0.000	234379	0	2246
Integrado	20000	0.000	2485052	0	23454

Vezes no produto escalar para gerar o convex: 76370