Relatório do Trabalho 1 de Computação Gráfica

Allan Groisman, Dhruv Babani
Faculdade de Ciência Da Computação — PUCRS

30 de setembro de 2023

1 Introdução

Este trabalho, consiste em desenvolver programa que avalie algoritmos de inclusão de pontos em polígonos de um diagrama de Voronoi, usando OpenGL. A base deste programa que consiste no carregamento dos poligonos e em exibi-los na tela, já foi previamente disponibilizada.

O problema apresentado é a criação de um ponto que move-se pelo diagrama e que retorna o poligono no qual se encontra. Inicialmente há duas formas de testar se o ponto está no poligono, o teste contra poligono convexo e o teste contra poligono côncavo. Porém é necessário descobrir contra quais poligonos deve-se fazer o teste para achar o posicionamento do ponto. Para isto, neste trabalho, foram desenvolvidos quatro algoritmos que filtram os poligonos a serem testados em busca de um melhor desempenho:

- Todos os poligonos.
- Poligonos cujo o envelope tem intersecção com a reta horizontal criada a partir do ponto.
- Poligonos cujo o ponto está dentro do envelope.
- Polígonos a partir da informação de vizinhança.

2 Teste de Polígono Convexo

Este é o teste padrão do trabalho, presente no código como o método testeConvexo(), só não é utilizado em um dos algoritmos. Nele é testado se para todas as arestas de um polígono, o ponto está a direita ou a esquerda desta aresta. Neste caso, se o ponto está a esquerda de alguma das arestas, ele não está incluso no polígono.

O teste busca o ponto a ser testado (P), o inicio da aresta (A) e o final da aresta (B). A partir disto cria dois vetores: vetor da Aresta (B - A) e o vetor inicio aresta até o ponto (P - A). Então é feito o produto vetorial (função previamente disponibilizada no programa) destes dois vetores resultando em um terceiro vetor cuja a coordenada Z ordena o lado do ponto em que estava o poligono: Z positivo ponto na esquerda, Z negativo ponto na direita.

3 Teste de Polígono Côncavo

O teste de polígono côncavo, presente no código como o método testeConcavo(), é utilizado apenas contra polígonos cujo envelope tem intersecção com a reta horizontal do ponto, pois ele reaproveita esta mesma reta. O teste consiste em traçar essa reta para qualquer um dos lados e contar o número

de intersecçoes com arestas que o ponto tem, se for par o ponto está fora do polígono, se for ímpar está dentro.

Para cada aresta, o teste busca os pontos iniciais e finais da aresta, formando sua reta, além do ponto a ser testado e um ponto horizontal bem a esquerda, distante do polígono. Estes quatro pontos são informados em uma função inerente da base do programa chamada HaIntersecao que cálcula se as retas formadas por esses pontos tem intersecção, contando o número de vezes que acontece e concluindo se o ponto está dentro ou não do polígono.

4 Método testaTodos()

Este método é o que testa se o ponto está incluso contra todos os polígonos do diagrama. Para cada um dos polígonos chama o testeConvexo() e descobre e qual o polígono atual, porém em seu pior caso passa por todos os polígonos, testando contra todos, estando incluso apenas no último.

Este teste começa como padrão no programa, e é chamado para desobrir o primeiro polígono a qual o ponto pertence quando criado. Apesar de pouco eficiente, por ser chamado apenas uma vez no carregamento, não faz diferença no desempenho do programa.

5 Envelopes

Nos próximos dois algoritmos são necessários envelopes. Envelopes são retângulos que envolvem o polígono e ajudam na localização dos mesmos. Os testes de inclusão do ponto ou de intersecção de retas contra os envelopes são mais simples do que os polígonos, pois eles possuem apenas duas arestas verticais e duas horizontais.

Se um ponto é testado contra um envelope e retorna falso, logo é garantido que o ponto não pertenece ao respectivo polígono, evitando assim um teste desnecessário e caro. Porém se o teste retorna positivo, tem de se tomar cuidado, pois não há a certeza de que o ponto está no polígono.

No programa, ao carregar cada polígono, foi adicionado também o carregamento de seu respectivo envelope.

5.1 Metodo testaRetaEnvelope()

Neste metodo, é criado uma reta horizontal para a esquerda a partir do ponto a ser testado. Esta reta testa contra os envelopes de todos os poligonos.

Primeiramente testa-se se o ponto está entre os limites de altura possíveis do envelope. Só então, caso esteja, testa se tem intersecção com a aresta mais a esquerda do mesmo. Se há intersecção, há a possibilidade do ponto pertencer ao respectivo polígono, então o teste de inclusão é chamado.

Apenas neste método é utilizado do testeConcavo, pois nele a reta horizontal para a esquerda é reaproveitada internamente.

5.2 Metodo testaDentroEnvelope()

Diferente do anterior, a ponto é testado se dentro do envelope a partir de um método interno do próprio envelope pontoEstaDentro().

Desta forma, caso o ponto esteja fora do envelope, tem-se a certeza que também está fora do respectivo polígono, evitando um teste desnessário. Caso esteja dentro, é utilizado o testeConvexo para concluir se o ponto está ou não no polígono.

6 Arestas e Seus Vizinhos

Após carregar os polígonos, são carregadas também as informações de vizinhança a partir do método obtemVizinhosDaAresta(). Elas serão importantes no último algoritmo do programa.

O objetivo principal deste método é identificar quais arestas nos polígonos do diagrama de Voronoi são vizinhas entre si. Isso é importante porque, em um diagrama de Voronoi, cada aresta forma fronteira com apenas um vizinho.

O método começa iterando por todos os polígonos do diagrama de Voronoi. Isso é feito usando um loop que percorre todos os polígonos do diagrama, identificados por um índice poligonoAtual.Logo,para cada vértice do polígono atual, o método obtém as informações da aresta correspondente. Isso é feito por meio da função getAresta, que retorna o início e o fim da aresta.

Após obter as informações da aresta do vértice atual, o método entra em um loop que percorre os polígonos a partir do polígono atual (ou seja, os polígonos vizinhos). O objetivo aqui é encontrar uma aresta no polígono vizinho que seja igual à aresta do polígono atual.Dentro do loop de polígonos vizinhos, o método obtém as informações da aresta do vértice do polígono vizinho e compara essas informações com a aresta do polígono atual.

Se as informações das duas arestas forem idênticas, isso significa que as arestas são vizinhas entre os polígonos. Portanto, o método registra a vizinhança entre essas arestas, associando os polígonos e vértices correspondentes nos polígonos em questão.

6.1 Metodo testeAresta()

Essa função utiliza das informaçoes da posição anterior e atual do ponto, do polígono no qual ele estava inserido, além das informações de vizinhança previamente carregadas para cada aresta do mesmo.

- A função percorre todas as arestas do último polígono conhecido usando um loop for.
- Para cada aresta, a função obtém os dois pontos que formam a aresta, chamados de inicioAresta e finalAresta, usando Diagrama[pontoClicado.getUltimoPoligono()].getAresta(i, inicioAresta, finalAresta).
- Em seguida, a função cria três vetores: vetorAresta, vetorArestaAnterior e vetorArestaPosterior. O vetor vetorAresta representa a própria aresta, e os outros dois vetores representam vetores do ponto clicado até os pontos de início e fim da aresta.
- A função calcula o produto vetorial entre o vetor da aresta e o vetor da aresta anterior (resultadoUm) e entre o vetor da aresta e o vetor da aresta posterior (resultadoDois).
- Em seguida, a função verifica se os produtos vetoriais têm sinais opostos. Se isso acontecer, significa que o ponto clicado cruzou a aresta e entrou em outro polígono Voronoi.
- O novo polígono é definido como o "vizinho"da aresta cruzada usando inicioAresta.getVizinho().
- O loop é interrompido usando break, pois a função já determinou qual aresta foi cruzada e em qual novo polígono o ponto clicado está agora.

Resumidamente, a função determina qual aresta do polígono o ponto cruzou para entrar no respectivo polígono vizinho.

7 Funções Extras

Algumas funções extras foram utilizadas para verificar o funcionamento e a eficiência dos algoritmos.

- Na função display o ponto também é desenhado para acompanhar seu movimento na tela.
- A função testaPonto() é quem decide qual algoritmo será utilizado. Ela recebe o ponto e um inteiro que informa qual é o teste atual a ser executado.
- Antes de qualquer algoritmo, na testaPonto(), é testado se o ponto continua no mesmo polígono. Caso aconteça, nada mais é feito.
- Em todos os testes, prints são feitos para informar qual algoritmo foi executado e suas peculiaridades. Após os mesmos, são informados o número de execuções de seus principais cálculos, como a chamada da função ProdVetorial ou HaInterseccao. Desta forma fica claro o funcionamento e a eficiência de cada algoritmo.
- Na função display são desenhados os números dos polígonos, além de seus respectivos envelopes para facilitar visualização e verificação da execução.
- Ainda na função display, quando o teste de número 4, que utiliza das vizinhanças e arestas, está ativo, são desenhadas os números das arestas.
- O ponto começa nas coordenadas (0,0,0) e sua movimentação é feita através dos botões "W,A,S,D"ou do botão esquerdo do mouse. Quando movimentado o ponto é chamado a função testaPonto().
- Ao clicar nos botões "1,2,3 ou 4"é possível escolher qual algoritmo está ativo para a próxima execução.

8 Funcionamento e Resultados

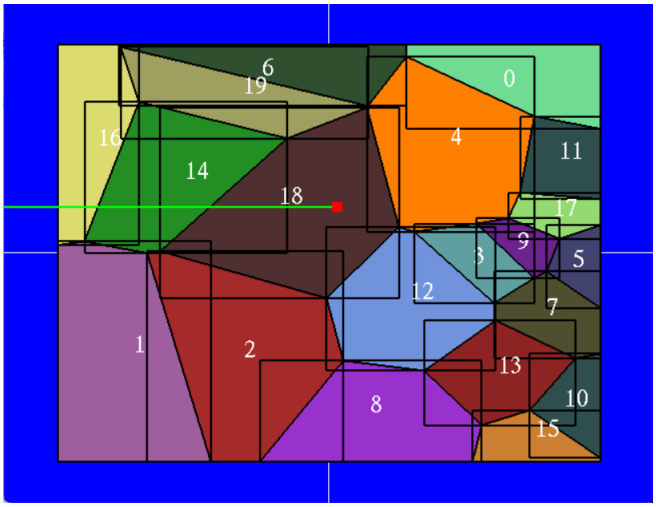
Foram feitos testes em três malhas de polígonos: com 20, 100 e 500 polígonos. Ao rodar cada algoritmo foram contabilizados os números de vezes que suas principais funções foram chamadas, afim de medir a performance do mesmo com o aumento da quantidade de poligonos. Os dados coletados resultaram de quatro testes em polígonos aleatórios para cada algoritmo. Os dados brutos obtidos podem ser observados na tabela abaixo, seguido pela mesma tabela porém contendo as médias.

Número de Polígonos - Teste	Mesmo Polígono	Reta X Envelope	Ponto Dentro Envelope	Vizinho Intersecção Aresta
Teste 20 polígonos	5,6,5,7	29,20,71,51	7,11,9,8	3,9,6,6
Teste 100 polígonos	6,9,4,6	185,15,185,198	9,8,9,6	3,9,21,12
Teste 500 polígonos	6,6,7,7	572,1083,754,123	14,13,10,13	3,18,9,9

Número de Polígonos - Teste	Mesmo Polígono-Media	Reta X Envelope-Media	Ponto Dentro Envelope-Media	Vizinho Intersecção Aresta-Media
Teste 20 polígonos	5.75	42.75	8.75	6.0
Teste 100 polígonos	6.25	145.75	8.0	11.25
Teste 500 polígonos	6.5	633	12.5	9.75

8.1 Mesmo Poligono

Este algoritmo acontece quando o ponto continua no mesmo polígono. Ele informa onde está, além do número do próprio polígono e a quantidade de chamadas da função ProdVetorial.



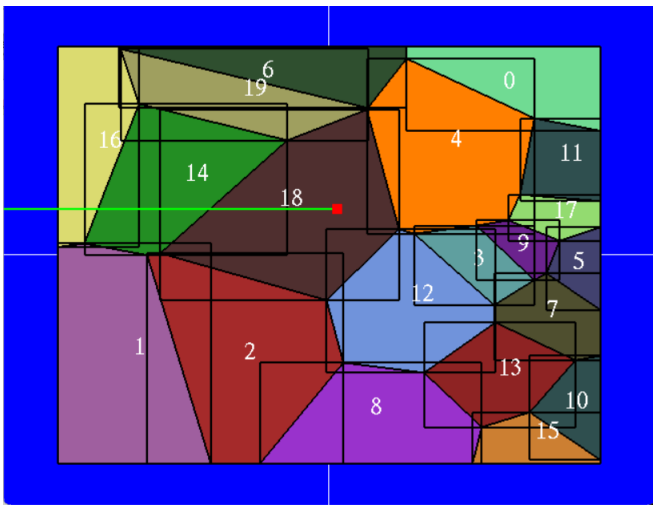

```
MESMO POLIGONO
Esta no poligono de numero: 18
Numero de ProdVet: 5
```

A partir dos testes é possível concluir que sua variação independe da quantidade de poligonos. Ele testa apenas o número de arestas que o poligono atual possui. Segue uma tabela com o resultado médio com o aumento do número de polígonos.

Número de Polígonos - Teste	Mesmo Polígono-Media
Teste 20 polígonos	5.75
Teste 100 polígonos	6.25
Teste 500 polígonos	6.5

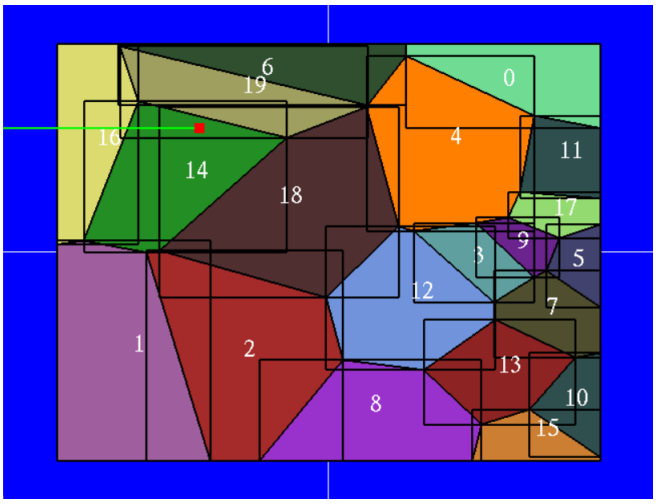
8.2 Reta x Envelope

Este polígono é o que testa apenas contra os polígonos cujos envelopes tem intersecção com a reta horizontal para a esquerda a partir do ponto.




```
TESTE RETA X ENVELOPE:
Testando com poligono de numero: 14
Testando com poligono de numero: 16
Testando com poligono de numero: 18
Esta no poligono de numero: 18
Numero de HaInterseccao: 71
```

O teste é realizado em todos os envelopes de forma crescente, o que pode ser visto na saída do terminal. Em seguida ele diz o polígono atual e quantas vezes a função HaInterseccao foi chamada. Segue um exemplo no qual o teste ignora o teste com polígonos maiores, pois já achou no menor, no caso ele na sequencia deveria testar também com o 16, 18 e 19.



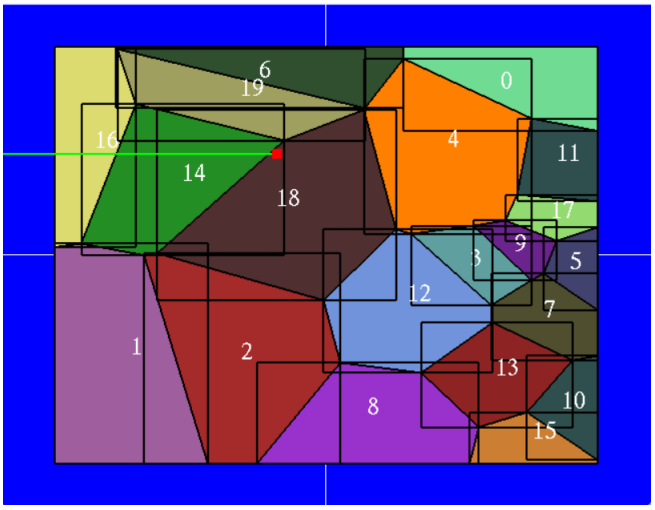

```
TESTE RETA X ENVELOPE:
Testando com poligono de numero: 14
Esta no poligono de numero: 14
Numero de HaInterseccao: 42
```

Após a bateria de testes realizadas com esse algoritmo, o número de HaInterseccao aumentou consideralavelmente de acordo com o quantidade de polígonos. Como ele usa de uma linha horizontal para esquerda, o melhor caso seria se o ponto estivesse em um polígono na extrema esquerda, testando apenas contra este próprio polígono. O pior caso seria a extrema direita, onde são testados todos os polígonos naquela linha, porém de forma crescente o que pode variar o resultado de acordo com o número do próprio polígono.

Número de Polígonos - Teste	Reta X Envelope-Media
Teste 20 polígonos	42.75
Teste 100 polígonos	145.75
Teste 500 polígonos	633

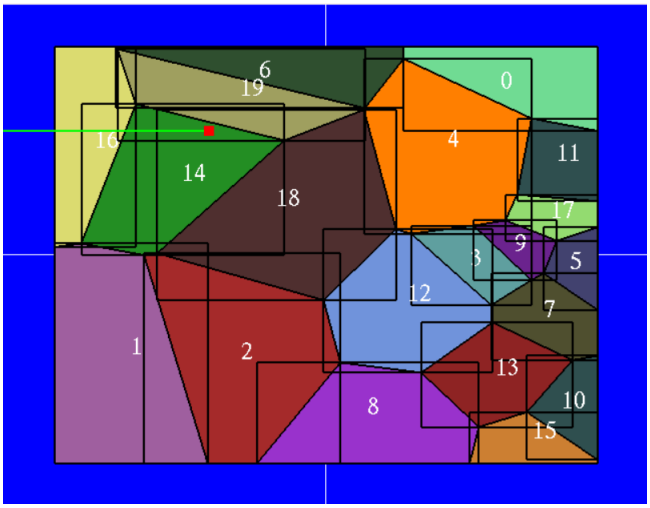
8.3 Dentro Envelope

Este algoritmo testa contra os polígonos cujo envelope contém o ponto. Aqui, assim como no teste com a reta, ele é executado de forma crescente, variando a eficiência de acordo com o número do polígono.




```
TESTE DENTRO ENVELOPE:
Ta dentro do envelope: 14
Ta dentro do envelope: 18
O ponto esta no poligono: 18
Numero de ProdVetorial: 11
```

Ele imprime no terminal os envelopes em que foram testados, além do número de ProdVetorial. Segue o exemplo caso ele já encontre no primeiro e deixa de testar nos maiores, neste caso 18 e 19.




```
TESTE DENTRO ENVELOPE:
Ta dentro do envelope: 14
O ponto esta no poligono: 14
Numero de ProdVetorial: 7
```

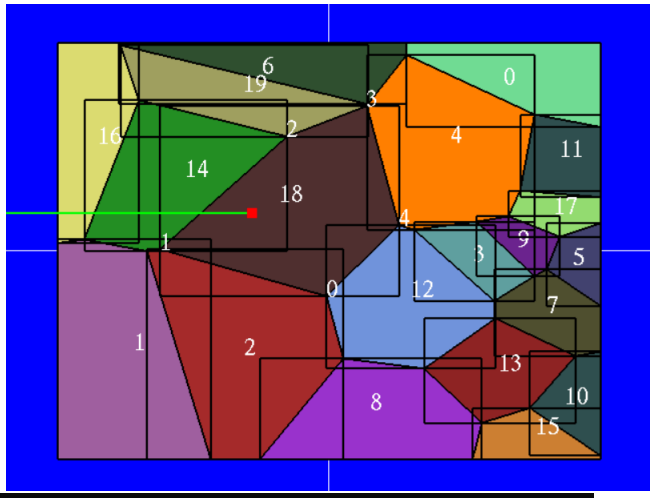
Há uma variação muito pequena e relativa no número de ProdVet ao aumentar o número de polígonos, porém existe uma quantidade maior de envelopes para serem testados. Seu melhor caso é quando está contido apenas no envelope do próprio polígono e vai piorando quanto mais envelopes o contém.

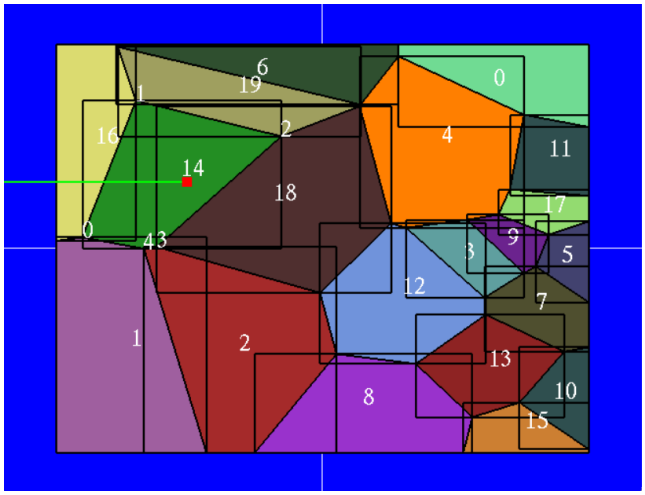
Número de Polígonos - Teste	Ponto Dentro Envelope-Media
Teste 20 polígonos	8.75
Teste 100 polígonos	8.0
Teste 500 polígonos	12.5

8.4 Aresta e Vizinho

A partir deste algoritmo é testado qual aresta do polígono atual é ultrapassada descobrindo qual o novo polígono a partir de vizinho que faz fronteira com aquela aresta (os vizinhos são carregados previamente). É importante observar que com este teste ativado, os números das arestas do polígono

atual aparecem em cima de seus respectivos vértices, como é possível de observar nas imagens abaixo.





```
TESTE ARESTA:
Ultimo Poligono: 18
Houve interseccao da aresta 1
Z: -11679.9 e 30406.7
Foi para o poligono vizinho: 14
Numero de ProdVetorial:8
```

No terminal, ao ser executado, o algoritimo imprime o polígono de onde saiu, a aresta cruzada, os valores de "Z"resultantes do cálculo (um positivo e outro negativo), o vizinho no qual o ponto está e o número de ProdVetorial.

Este algoritmo é o que tem melhor performance em grandes quantidades de polígonos, pois utiliza da informação de quais são os polígonos vizinhos, o que o limita para fazer o teste de cruzamento da aresta para o número máximo de arestas. Desta forma o algoritmo se torna independente da quantidade de poligonos totais, trabalhando de forma local, porém há a necessidade do carregamento prévio das arestas e seus vizinhos.

Número de Polígonos - Teste	Vizinho Intersecção Aresta-Media
Teste 20 polígonos	6.0
Teste 100 polígonos	11.25
Teste 500 polígonos	9.75

Como conclusão geral dos testes pode-se avaliar que a velocidade de execução do programa está diretamente ligada ao carregamento de informações prévias na memória. Utilizar envelopes junto aos dois algoritmos previamente vistos neste trabalho são uma forma válida de filtrar os polígonos a serem testados e podem ser de grande valia em certas situações.

Porém utilizar de mais memória para carregar previamente todas as informações de arestas e vizinhos é a forma mais eficiente em geral para obter resultados rápidos. O algoritmo age de forma local entre o polígono atual e seus vizinhos, independente da quantidade total polígonos. Entretanto quanto maior a quantidade, mais memória e tempo de carregamente é necessário.

9 Link do Video

Link do video: https://youtu.be/d6geYIe7scU