Disciplina: Computação Gráfica

Professor: Marcio Sarroglia Pinho

**Trabalho 01 – Diagrama de Voronoi**

Felipe Freitas e Lucas Wolschick

**Sumário**

[**1. O propósito** 3](#_Toc147010248)

[**2. Explicações Gerais** 3](#_Toc147010249)

[**2.1 Início do programa** 3](#_Toc147010250)

[**2.2 Movimento Interno** 3](#_Toc147010251)

[**2.3 Movimento Externo** 3](#_Toc147010252)

[2.3.1 Inclusão de pontos em polígonos côncavos 4](#_Toc147010253)

[2.3.2 Inclusão de pontos em polígonos convexos 4](#_Toc147010254)

[2.3.3 Inclusão de pontos em polígonos convexos utilizando a informação de vizinhança disponível no diagrama de Voronoi 4](#_Toc147010255)

[**3. Cenários de teste** 5](#_Toc147010256)

[**3.1 Cenário 1 - 20 Polígonos** 5](#_Toc147010257)

[3.1.1 Método inicial 6](#_Toc147010258)

[3.1.2 Movimento interno 7](#_Toc147010259)

[3.1.3 Inclusão de pontos em polígonos côncavos 7](#_Toc147010260)

[3.1.4 Inclusão de pontos em polígonos convexos 9](#_Toc147010261)

[3.1.5 Inclusão de pontos em polígonos convexos com informação de vizinhança 10](#_Toc147010262)

[**3.2 Cenário 2 - 100 Polígonos** 11](#_Toc147010263)

[3.2.1 Método inicial 12](#_Toc147010264)

[3.2.2 Movimento interno 13](#_Toc147010265)

[3.2.3 Inclusão de pontos em polígonos côncavos 13](#_Toc147010266)

[3.2.4 Inclusão de pontos em polígonos convexos 14](#_Toc147010267)

[3.2.5 Inclusão de pontos em polígonos convexos com informação de vizinhança 16](#_Toc147010268)

[**3.3 Cenário 3 - 500 Polígonos** 17](#_Toc147010269)

[3.3.1 Método inicial 18](#_Toc147010270)

[3.3.2 Movimento interno 19](#_Toc147010271)

[3.3.3 Inclusão de pontos em polígonos côncavos 20](#_Toc147010272)

[3.3.4 Inclusão de pontos em polígonos convexos 21](#_Toc147010273)

[3.3.5 Inclusão de pontos em polígonos convexos com informação de vizinhança 22](#_Toc147010274)

[**4. Conclusões** 23](#_Toc147010275)

# **1. O propósito**

O propósito deste relatório é implementar e analisar três métodos relacionados à Computação Gráfica: Inclusão de pontos em polígonos côncavos, inclusão de pontos em polígonos convexos e inclusão de pontos em polígonos convexos utilizando a informação de vizinhança disponível no Diagrama de Voronoi. Os métodos serão avaliados em cenários com 20, 100 e 500 polígonos.

# **2. Explicações Gerais**

## **2.1 Início do programa**

Ao iniciar o programa, após carregar todos os polígonos e desenhar o ponto, deve-se descobrir quais são os vizinhos do polígono e, posteriormente em qual posição e polígono o ponto está localizado inicialmente.

Para o primeiro, temos de passar por todos os polígonos e armazenar as informações de quais outros polígonos compartilham uma aresta com este. Para isso, utilizamos dos envelopes e, caso haja colisão entre o envelope do polígono em questão e o do próximo, considera-se que eles são vizinhos, visto que, como todos os polígonos são convexos, sempre que houver colisão há também um compartilhamento de aresta.

Depois disso, passamos novamente por todos os polígonos, em ordem numérica, e realizamos o produto vetorial para saber se o polígono em questão contém o ponto. Quando alcançarmos o polígono, paramos o algoritmo e salvamos em uma variável em qual polígono estamos. Este processo funciona bem se, por acaso, o primeiro polígono for o central, mas caso o polígono inicial seja o último da lista, teríamos que realizar quantidadeDePoligonos x quantidadeDeVerticesDosPoligonos operações, o que é muito mais custoso.

## **2.2 Movimento Interno**

Ao movimentarmos o ponto, executamos o algoritmo de produto vetorial apenas no polígono em que estávamos anteriormente, que foi salvo no passo anterior. Como só temos de testar N vezes, onde N é o número de vértices do polígono anterior, este é um meio eficiente de definir se ainda estamos dentro do mesmo polígono ou se saímos dele, mas não é o melhor para definir para qual polígono fomos, como vimos anteriormente.

## **2.3 Movimento Externo**

Ao movimentarmos o ponto, caso o algoritmo anterior aponte que não estamos mais no mesmo polígono, vamos realizar 3 testes: inclusão de pontos em polígonos côncavos, inclusão em polígonos convexos e inclusão em polígonos convexos com a informação de quem são os vizinhos do polígono em questão.

### 2.3.1 Inclusão de pontos em polígonos côncavos

Para melhorar a eficiência deste algoritmo, vamos utilizar um algoritmo de faixas para limitar a validação apenas aos polígonos que estiverem dentro da mesma faixa que a linha horizontal usada para teste. Todas as vezes que um polígono estiver dentro dos limites desta faixa, vamos passar por todas as suas arestas e contar quantas vezes existe uma intersecção. Se este número for par, entendemos que o ponto não está neste polígono. Repetimos o algoritmo até encontrar algum polígono no qual a quantidade de intersecções seja ímpar e que, portanto, contém o ponto.

### 2.3.2 Inclusão de pontos em polígonos convexos

Desta vez, optou-se por utilizar um algoritmo de inclusão em envelopes para reduzir a quantidades de polígonos que devem ser testados, ou seja, vamos passar por todos os polígonos do diagrama e, caso o ponto esteja dentro do seu envelope, vamos executar novamente a função de produto vetorial para determinar se o ponto está dentro do polígono em questão. Repetimos o processo até que algum dos polígonos contenha o ponto, isto é, o resultado do produto vetorial seja consistente para todas as arestas.

### 2.3.3 Inclusão de pontos em polígonos convexos utilizando a informação de vizinhança disponível no diagrama de Voronoi

Aqui, ao invés de reduzir as possibilidades por uma linha que cruza todo o diagrama ou então ter de passar por todo o diagrama para determinar quais envelopes envolveriam o ponto, vamos percorrer um limite de N polígonos - onde N é a quantidade de vizinhos do polígono em que o ponto estava antes - que foi obtida no passo 2.1. Para cada um dos vizinhos, vamos executar o algoritmo de produto vetorial, buscando descobrir para qual vizinho e, principalmente, por qual aresta, o ponto passou.

# **3. Cenários de teste**

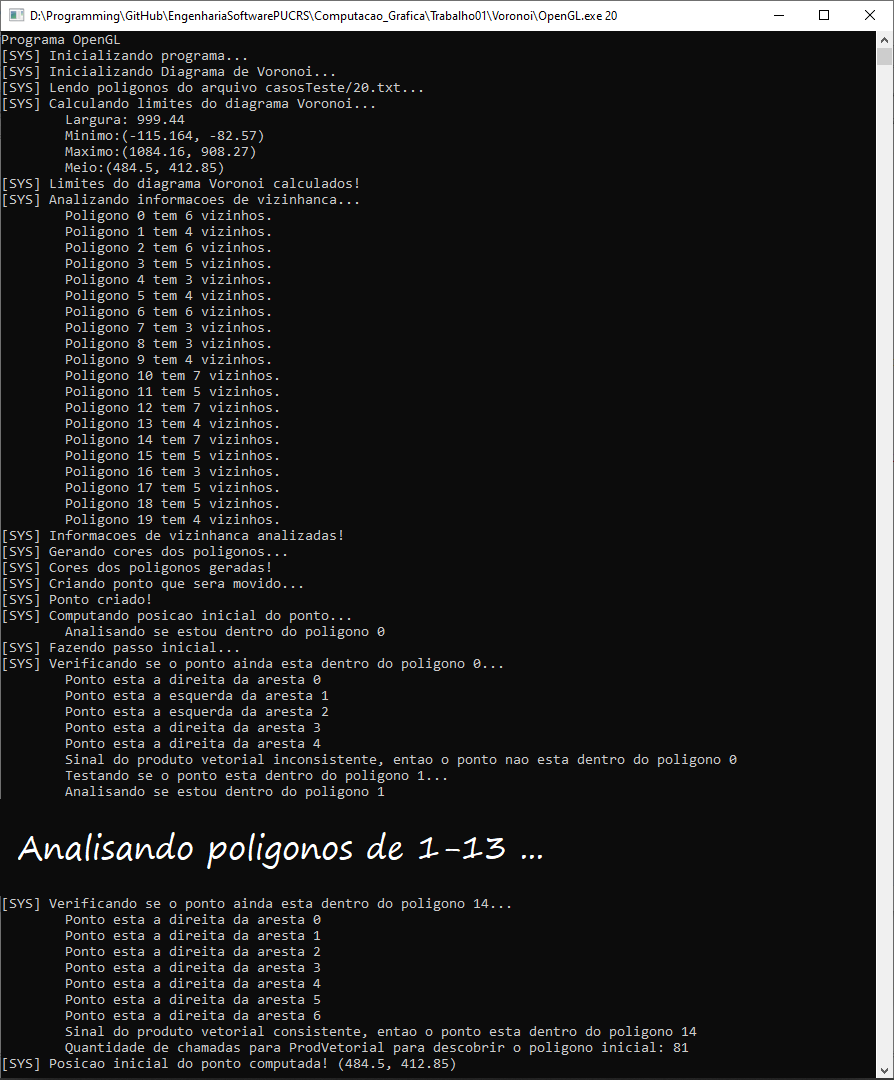
## **3.1 Cenário 1 - 20 Polígonos**

A screenshot of a computer

Description automatically generated

### 3.1.1 Método inicial

Após contarmos e salvarmos os vizinhos de todos os polígonos, foram necessárias 81 chamadas ao método ProdVetorial para determinar a posição inicial do ponto, conforme imagens abaixo.



### 3.1.2 Movimento interno

São necessárias apenas 7 chamadas, 1 para cada aresta do polígono, para determinar se o ponto ainda está no polígono que estava antes do último movimento, conforme imagem abaixo.

A screenshot of a computer program

Description automatically generated

### 3.1.3 Inclusão de pontos em polígonos côncavos

Após mover o ponto para fora do polígono anterior (14) e desenhar uma linha entre o início da tela e o ponto, vamos passar por todos os polígonos até N, onde N é o polígono para o qual me movi, conforme podemos confirmar na imagem a seguir. Como os polígonos 0, 1, 2, 3, 10 e 12 estão dentro da mesma faixa (horizontal) que a linha, vamos testar todas suas arestas procurando por intersecções, já para os polígonos de 4-9 e 11, como eles estão fora da faixa, não precisamos realizar nenhuma computação. Ao chegar no polígono 12, finalmente obtemos um número ímpar de intersecções com a linha traçada, o que nos permite concluir que estamos dentro deste polígono e nos permite passar para o próximo algoritmo. No caso deste diagrama, o total de vezes que chamamos o método HaInterseccao foi 34.

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

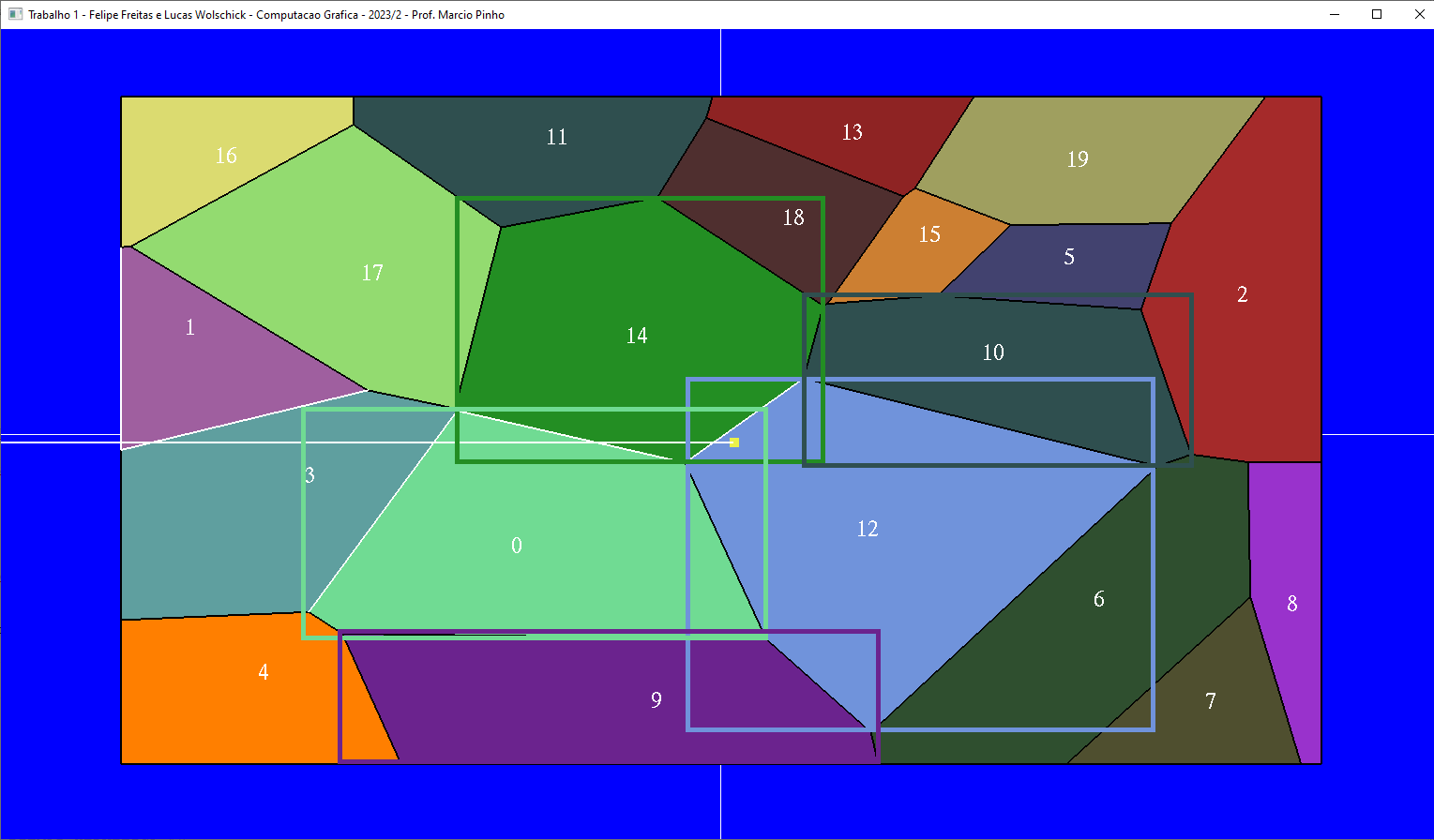
### 3.1.4 Inclusão de pontos em polígonos convexos

Após mover o ponto para fora do polígono anterior (14), vamos passar por todos os polígonos até N, onde N é o polígono para o qual me movi, conforme podemos confirmar na imagem a seguir. Como os envelopes dos polígonos de 1-11 não contém o ponto, não precisamos realizar nenhuma computação com estes, apenas com os polígonos 0 e 12. Aqui novamente vamos calcular o produto vetorial, mas apenas para as 5 arestas do polígono 0 e para as 5 arestas do polígono 12. Ao chegar no polígono 12 conseguimos concluir, através do resultado consistente do produto vetorial, em qual polígono estamos, tendo sido necessárias apenas 10 chamadas para a função ProdVetorial.

A screen shot of a computer screen

Description automatically generated

Imagem simulada dos envelopes de alguns dos polígonos, visto que o método de desenhar envelopes não estava funcionando:



### 3.1.5 Inclusão de pontos em polígonos convexos com informação de vizinhança

Neste método, após sair do polígono anterior (14), vamos analisar apenas os N vizinhos do polígono – no caso, 7 - conforme a imagem anterior. Como os envelopes dos vizinhos 2-5 não contém o ponto, não precisamos realizar nenhuma computação com estes, apenas com o primeiro e último vizinho. Mais uma vez, precisamos fazer o produto vetorial nos polígonos cujos envelopes contém o ponto e, como temos poucos polígonos, a única diferença entre esta solução para o problema e a solução anterior é que esta analisa menos polígonos inicialmente, mas ambas realizam operações apenas sobre os 2 polígonos que contém o ponto. Em ambos os casos chamamos a função ProdVetorial um total de 10 vezes, porém, com a última, sabemos que saímos pela aresta de número 4, conforme imagem abaixo:

A screenshot of a computer screen

Description automatically generated

## **3.2 Cenário 2 - 100 Polígonos**

Neste cenário existem polígonos “sem” cor (pretos), mas o resultado independe desta falta.

A screenshot of a computer

Description automatically generated

### 3.2.1 Método inicial

Após contarmos e salvarmos os vizinhos de todos os polígonos, foram necessárias 180 chamadas ao método ProdVetorial para determinar a posição inicial do ponto, conforme imagem abaixo.



### 3.2.2 Movimento interno

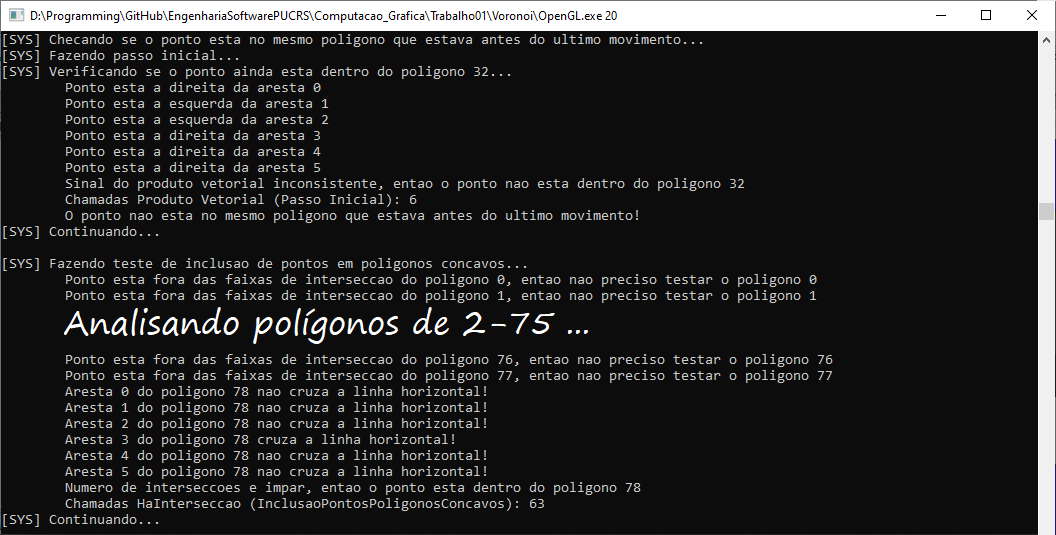
Ao contrário do cenário anterior, apesar de aqui haver mais polígonos, como o inicial possui apenas 6 arestas é necessário fazer somente 6 chamadas para determinar se o ponto ainda está no polígono inicial (32), conforme imagem abaixo.

A computer screen with white text

Description automatically generated

### 3.2.3 Inclusão de pontos em polígonos côncavos

Após mover o ponto para fora do polígono anterior (32) e desenhar uma linha entre o início da tela e o ponto, vamos passar por todos os polígonos até N, onde N é o polígono para o qual me movi (78), conforme podemos confirmar na imagem a seguir. Os polígonos de 0-31, 34-42, 45-51, 54, 56, 58-63, 65-73, 76 e 77, que não estão dentro da mesma faixa (horizontal) que a linha, e o ponto 32 (que é minha origem), não precisam ser testados quanto a intersecção com a reta, nos deixando com os outros 11 polígonos (33, 43-44, 52-53, 55, 57, 64, 74-75 e 78) por testar. Ao comparar com o polígono onde estou (78), finalmente obtemos um número ímpar de intersecções com a linha traçada, o que nos leva ao próximo algoritmo. No caso deste diagrama, o total de vezes que chamamos o método HaInterseccao foi 63, conforme imagem abaixo.



A screenshot of a computer

Description automatically generated

### 3.2.4 Inclusão de pontos em polígonos convexos

Após mover o ponto para fora do polígono anterior (32), vamos passar por todos os polígonos até N, onde N é o polígono para o qual me movi, conforme podemos confirmar na imagem a seguir. Como apenas o envelope do polígono 78 envolve o ponto, só realizamos a computação do produto vetorial com 1 polígono (embora fosse possível criar uma condição para, caso houvesse apenas 1 envelope, assumir o ‘dono’ deste como o polígono que contém o ponto). Certamente, tal polígono contém o ponto e, como este tem 6 lados, realizamos 6 vezes o produto vetorial e passamos para o próximo algoritmo, que deve chegar mais rápido nesta conclusão, visto que este apesar de fazer 1 computação mais ‘pesada’ apenas ainda precisou recusar todos os envelopes do diagrama até encontrar um que contivesse o ponto.

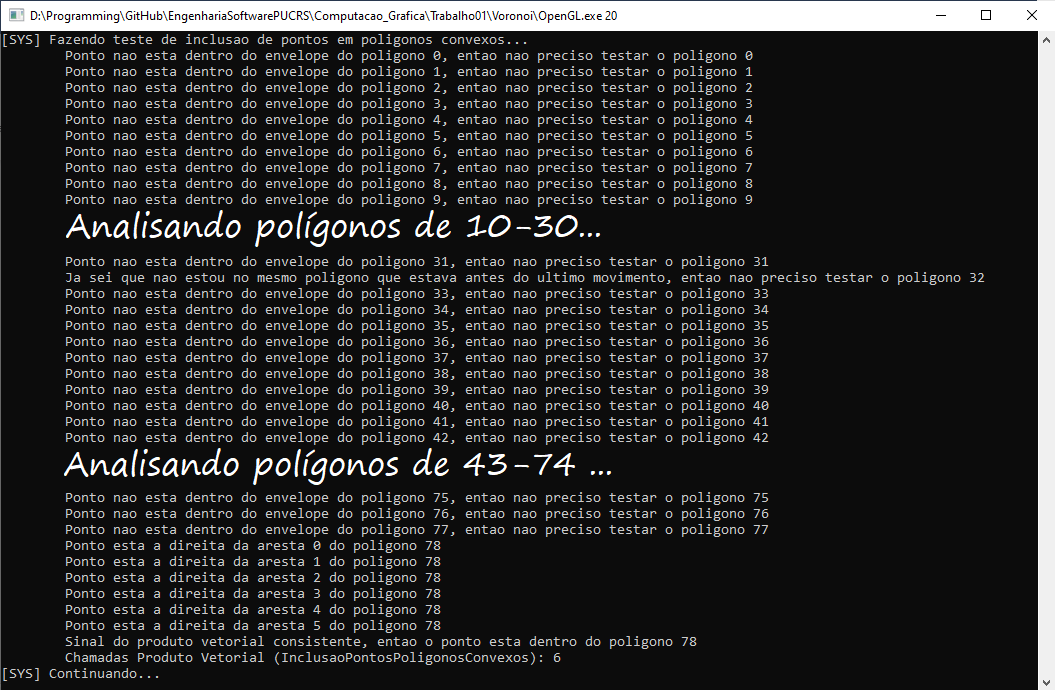
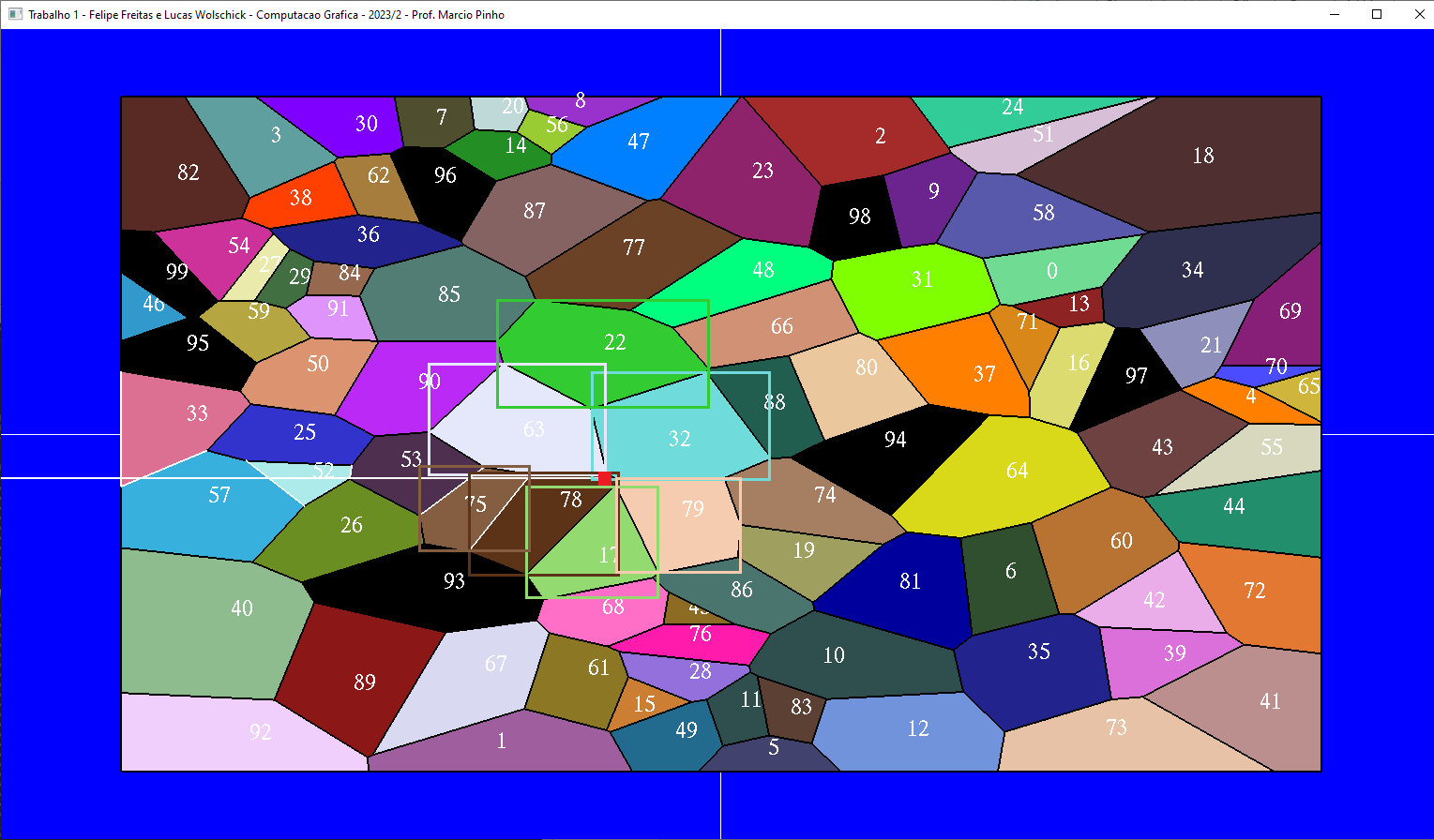


Imagem simulada com o ponto ampliado e vermelho para melhor visualização, com os envelopes de alguns dos polígonos:



### 3.2.5 Inclusão de pontos em polígonos convexos com informação de vizinhança

Conforme citado anteriormente, após sair do polígono anterior (32) vamos analisar os N vizinhos do polígono – no caso, 6 - conforme a imagem anterior. Novamente, os envelopes dos 3 primeiros vizinhos não contêm o ponto, então realizamos o teste no quarto vizinho. Após realizar o produto vetorial com os 6 vértices deste, concluímos que é esta aresta (a 4ª, denominada ‘aresta 3’ na imagem abaixo) que foi cruzada no último movimento. É nítida a diferença, agora que temos 5x mais polígonos do que no cenário 1, de realizar o teste apenas nos vizinhos e realizá-lo contra todos os polígonos do diagrama, e esta diferença apenas cresce conforme aumenta o tamanho dos diagramas.

A screenshot of a computer program

Description automatically generated

## **3.3 Cenário 3 - 500 Polígonos**

Neste cenário existem diversos polígonos sem cor (pretos), mas novamente isto não impacta no resultado ou na análise.

A screenshot of a computer screen

Description automatically generated

### 3.3.1 Método inicial

Após contarmos e salvarmos os vizinhos de todos os polígonos, foram necessárias 2354 chamadas ao método ProdVetorial para determinar a posição inicial do ponto, conforme imagem abaixo.

A screen shot of a computer screen

Description automatically generated

### 3.3.2 Movimento interno

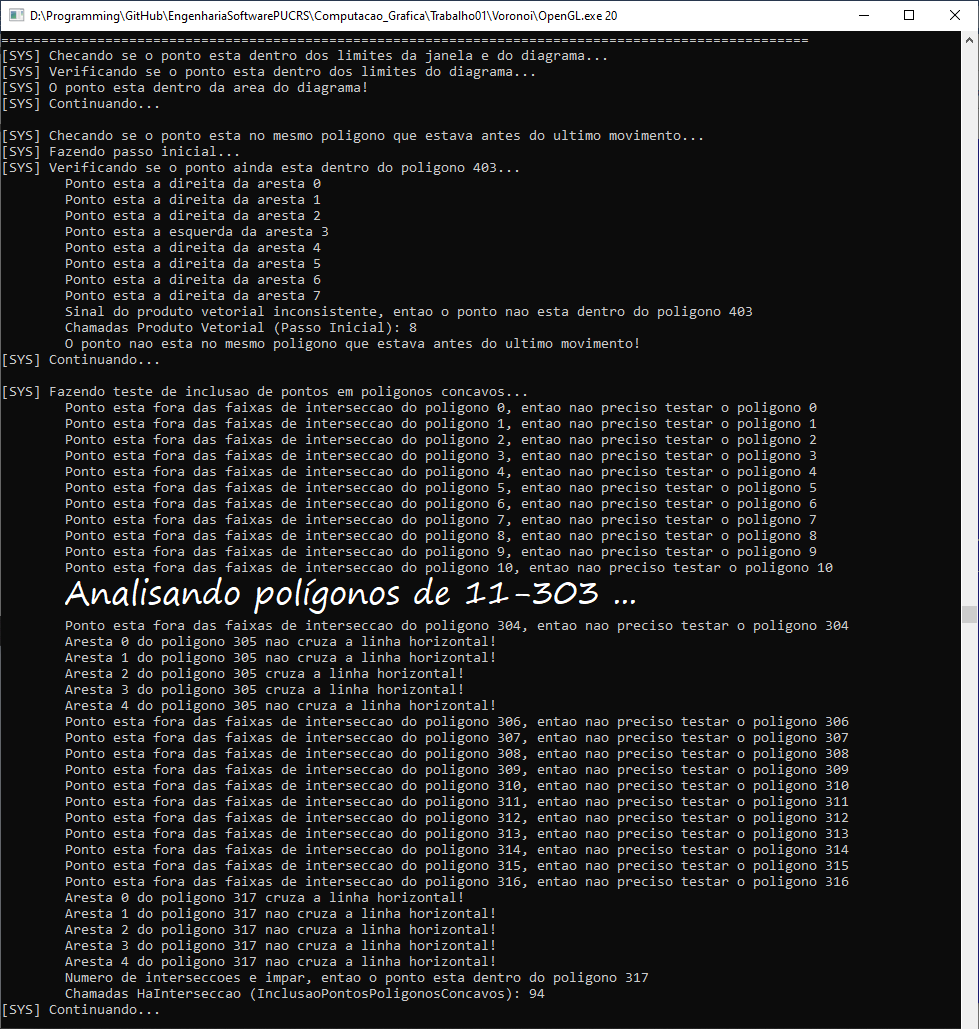
Independente do cenário, é certo que o único fator importante para o método inicial é a quantidade de lados do polígono inicial e, como este possui 8 arestas, são necessárias 8 comparações para garantir que o ponto se mantém no polígono 403.

A screenshot of a computer program

Description automatically generated

### 3.3.3 Inclusão de pontos em polígonos côncavos

Após mover o ponto para fora do polígono anterior (403) e desenhar uma linha entre o início da tela e o ponto, passamos por até N pontos, sendo N o número do ponto em que estou, conforme podemos confirmar na imagem a seguir. Desta vez, a linha cruzou pelos envelopes dos polígonos 17, 20, 38, 61, 79, 101, 104, 143, 159, 170, 185-186, 211, 222, 246, 305 e 317 (um total de 17), enquanto todos os outros 300 foram poupados de qualquer computação mais extensa. Neste diagrama, o total de vezes que chamamos o método HaInterseccao foi apenas 94, o que é um tanto quanto surpreendente, visto que são 500 polígonos ao todo dos quais 318 foram comparados, conforme imagem abaixo.



A screenshot of a computer screen

Description automatically generated

### 3.3.4 Inclusão de pontos em polígonos convexos

Após mover o ponto para fora do polígono anterior (403), vamos passar por todos os polígonos até N, onde N é o polígono para o qual me movi (no caso, 317), conforme podemos confirmar na imagem a seguir. Como no cenário anterior, apenas 1 envelope contém o ponto, porém, foi necessário verificar a colisão com todos os 316 envelopes anteriores para chegar a esta conclusão. Como o polígono contém 5 lados, realizamos apenas 5 vezes a chamadas da função ProdVetorial e seguimos adiante para o último algoritmo testado, conforme imagem abaixo.

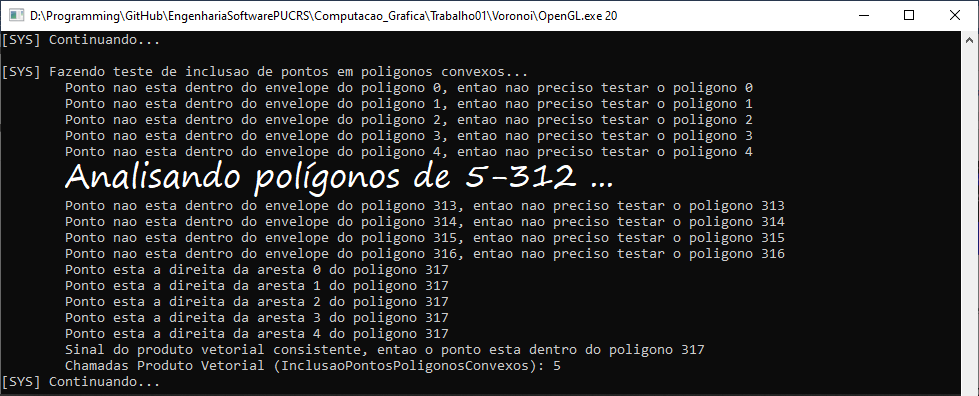
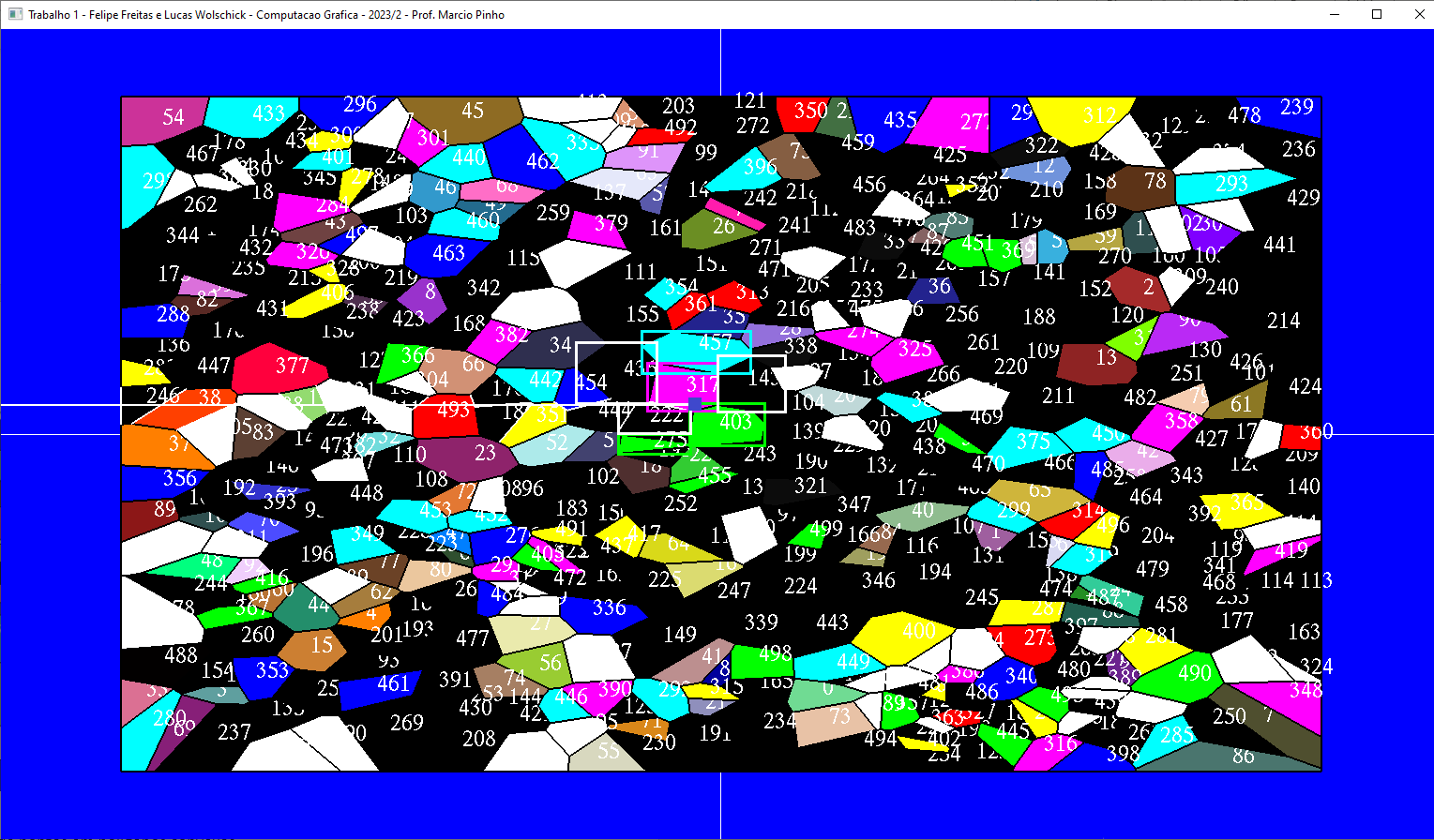


Imagem simulada com o ponto ampliado e azul escuro para melhor visualização, com os envelopes de alguns dos polígonos:



### 3.3.5 Inclusão de pontos em polígonos convexos com informação de vizinhança

Finalmente, após sair do polígono anterior (403) vamos percorrer seus N (8) vizinhos, como mostrado na imagem a seguir. Desta vez, tivemos o azar de ter cruzado a aresta do último vizinho, mas como os envelopes dos outros não continham o ponto, tivemos de realizar o teste do produto vetorial apenas 5 vezes, uma para cada aresta deste último. Conforme imagem abaixo, cruzamos a 7ª (última) aresta do polígono e terminamos os testes.

A screenshot of a computer screen

Description automatically generated

# **4. Conclusões**

Após realizar todas estas análises, podemos concluir que a complexidade dos algoritmos e a quantidade de cálculos mais custosos que devem ser realizados não crescem linearmente de acordo com a quantidade de polígonos, mas claramente existe uma diferença grande de complexidade entre eles e é muito vantajoso utilizar o ‘algoritmo de Voronoi’, isto é, levar em consideração apenas a vizinhança. Para que o algoritmo não fizesse sentido seria necessário ou que o ponto pudesse ‘pular’ por cima de um vizinho, isto é, atravessá-lo em um único movimento, ou então o polígono o qual se compara os vizinhos ter mais lados do que o diagrama tem de polígonos, o que não deve acontecer com frequência.