Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) – Escola Politécnica
Curso de Ciência da Computação – Disciplina de Computação Gráfica (Trabalho I)
Professor: Márcio Sarroglia Pinho
Alunos: Ricardo Süffert e Vinícius Turani

Inclusão de Pontos em Polígonos Convexos com Diagrama de Voronoi

ACESSE O VÍDEO EXPLICATIVO AQUI

1. CONSIDERAÇÕES INICIAIS E DESCRIÇÃO DO PROBLEMA:

Para a validação do *software* desenvolvido, será apresentado um breve resumo do funcionamento do programa para diagramas de Voronoi com 20, 100 e 500 polígonos convexos. O programa calcula a inclusão do ponto utilizando três algoritmos diferentes:

I. Inclusão de ponto em polígono côncavo: este algoritmo também funciona para polígonos convexos e utiliza o número de intersecções que uma linha reta horizontal traçada de uma das laterais do diagrama até o ponto de interesse tem com as arestas de cada polígono. Considera-se que o ponto está "dentro" do polígono cujas arestas forem interceptadas um número ímpar de vezes pela linha horizontal.

II. Inclusão de ponto em polígono convexo: funciona apenas para polígonos convexos e itera sobre todos os polígonos do diagrama, considerando que o ponto está dentro do polígono cujas coordenadas *z* do vetor resultante do produto vetorial entre o vetor que representa cada aresta e o vetor que parte do ponto inicial da aresta e chega no ponto de interesse (para o qual queremos calcular a inclusão no polígono) apresentarem todas o mesmo sinal.

III. Inclusão de ponto em polígono convexo com diagrama de Voronoi: durante a inicialização do programa, para cada polígono, são calculados os vizinhos de suas arestas. Com isso, quando detecta-se que o ponto trocou de polígono (esta detecção é feita com o algoritmo anterior), sabe-se qual aresta ele cruzou e, com isso, é possível concluir qual o novo polígono sem a necessidade de nenhum cálculo adicional.

Para a exibição, são numerados os polígonos em diagramas com menos de 30 polígonos. Com mais polígonos, o resultado de numerá-los seria um diagrama muito poluído. Ademais, para as demonstrações a seguir, foi implementado um mecanismo de *debug* no código, que permite, por meio de diretivas de pré-compilação, incluir ou não os trechos de código de depuração ao código compilado que será executado. Pode-se usar a constante simbólica `ENABLE_DEBUG_FEATURES` definida no código para ativar ou não o mecanismo de depuração. Valores *falsy* farão o programa não exibir nenhum mecanismo de auxílio à depuração, enquanto valores *non-falsy* ativam tal mecanismo. **Quando ativado, os polígonos cujos envelopes contiverem o ponto que se move na tela serão pintados de branco e uma linha vermelha é traçada da lateral esquerda do diagrama até o ponto**.

2. **<u>DEMONSTRAÇÃO DE FUNCIONAMENTO DO PROGRAMA PARA 20 POLÍGONOS</u>:** para demonstrar o funcionamento do programa, tomemos o diagrama abaixo, de 20 polígonos, como exemplo.

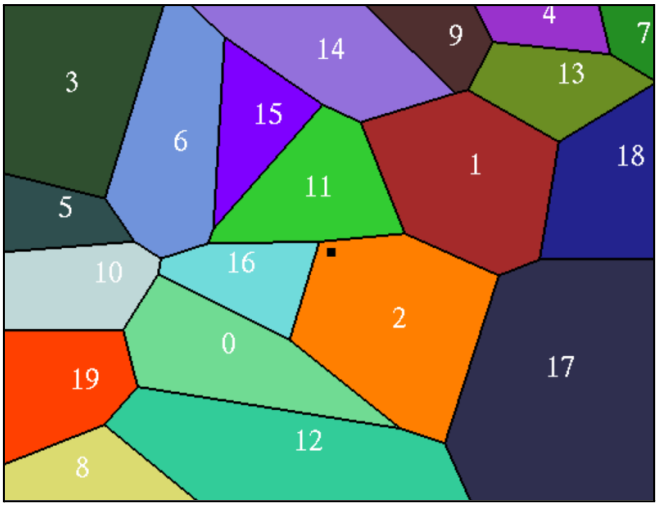

Imagem 1 – Diagrama com 20 polígonos

Primeiramente, ao observarmos as saídas do programa, antes de chamar qualquer um dos algoritmos abaixo, é utilizado o algoritmo 2 (explicado abaixo) <u>apenas para o último polígono conhecido</u> para determinar se o ponto ainda está nele. Se estiver, nenhum cálculo adicional é feito. Se não estiver, aí sim são chamados os algoritmos abaixo. Então, sempre que o ponto se move, verificamos se ele ainda está no último polígono conhecido utilizando a mesma lógica do algoritmo 2. Entretanto, como precisamos investigar apenas um polígono, que é o último conhecido, o número de chamadas à função `ProdVetorial` é sempre menor ou igual ao número de arestas do último polígono conhecido. É isso que vemos na *Imagem 2* e na *Imagem 3*.

2.1. Algoritmo 1 – Inclusão de ponto em polígono côncavo: para o cálculo de inclusão em polígono côncavo, é utilizada a função `HaInterseccao`. Como saída para este algoritmo, são fornecidas três informações: a cor e o índice do polígono atual e o número de vezes que foi chamada a função `HaInterseccao`. Tal função é chamada para aqueles polígonos cujo envelope contém o ponto que se move no diagrama, a começar pelo polígono de menor índice. Desse modo, a *Imagem 2* demonstra que o algoritmo determina corretamente o índice e a cor do polígono atual. Além disso, a saída do número de chamadas para a função `ProdVetorial` também corresponde ao valor esperado: estão com os envelopes ativos[1] os polígonos 1 e 2. Como o polígono 1 tem o menor índice, o algoritmo começa a investigação por ele. Para determinar se o ponto está dentro ou não, no caso do algoritmo para polígonos côncavos, é necessário iterar por <u>todas</u> as arestas do polígono para ter, ao final, a quantidade de intersecções. É exatamente isso que acontece na saída ilustrada na *Imagem 3*: a função `HaInterseccao` é chamada sete vezes para encontrar o número de intersecções da linha vermelha horizontal com o polígono 1. Como o número de intersecções encontrado é um (1), que é ímpar, o algoritmo pode terminar sua execução. Por outro lado, na imagem 3, o ponto está no polígono 2. Como ainda estão com os envelopes ativos apenas os polígonos 1 e 2, o algoritmo deve iterar por todas as arestas de cada polígono, do menor ao maior, até encontrar um polígono em que o ponto esteja dentro (número ímpar de intersecções). Se observarmos a saída, veremos 13 chamadas à função `HaInterseccao`. Isso acontece porque temos que chamá-la para todas as arestas do polígono 1 (sete arestas) para descobrir que o ponto não está nele. Então, partimos para o próximo polígono com o envelope ativo, que é o polígono 2. Então, chamamos a função para todas as arestas do polígono 2 (6 arestas) para descobrir que

[1] Considera-se "ativo" um envelope que contém o ponto de interesse e, portanto, está pintado de branco na representação do diagrama.

o ponto está nele. Portanto, somando as chamadas, teremos 13 chamadas à função `HaInterseccao`. Por fim, é válido explicar como funciona o cálculo no geral: na *Imagem 2*, o algoritmo concluiu que o ponto estava no polígono 1 porque, ao contar as intersecções com cada polígono com envelope ativo do diagrama, em ordem crescente, foi constatado que a linha interceptava um número ímpar de arestas do polígono 1 e, portanto, o ponto estava dentro dele.

2.2. Algoritmo 2 – Inclusão de ponto em polígono convexo: da mesma forma que para o anterior, este algoritmo tem como saída a cor e o índice do polígono atual, que, pelas imagens, podemos ver que está correta, e o número de chamadas à função `ProdVetorial`. Neste algoritmo, para a contagem de chamadas à função `ProdVetorial`, a lógica é similar à do algoritmo anterior, com a diferença de que, quando o ponto não está dentro do polígono sendo investigado, geralmente não precisaremos chamar a função para todas as arestas do polígono. Logo, na *Imagem 2*, vemos que a função `ProdVetorial` é chamada sete vezes, pois foi necessário investigar a lateralidade do ponto com todas as arestas do polígono 1 para determinar que o ponto está nele. Entretanto, na *Imagem 3*, percebe-se que a função foi chamada apenas oito vezes, e não treze como no algoritmo anterior. Isso porque, ao investigar a segunda aresta do polígono 1, constatou-se que o ponto estava à "direita" dela e, portanto, já foi possível concluir que ele não estava dentro do polígono 1 (o fato de termos chamado a função `ProdVetorial` duas vezes para determinar que o ponto não está mais no polígono 1, o que está escrito na saída da *Imagem 3*, comprova que o algoritmo 2 chama essa função também duas vezes para chegar à mesma conclusão). Então, avançou-se para o próximo polígono com envelope ativo, que é o polígono 2. Para ele, investigamos todas as arestas (seis arestas) para constatar que o ponto estava dentro do polígono – ou seja, à "esquerda" de todas as arestas. Somando as chamadas, obtemos oito chamadas à função `ProdVetorial`. Sumarizando e explicando de forma mais geral, na *Imagem 3*, o algoritmo concluiu que o ponto estava dentro do polígono 2 porque, já ao investigar a segunda aresta do polígono 1 com o produto vetorial, concluiu que o ponto estava à sua "direita" e, portanto, fora do polígono. Então, avançou para o polígono 2 e, ao investigar a lateralidade do ponto com todas as suas arestas utilizando o produto vetorial, concluiu que ele estava à "esquerda" de todas e, portanto, dentro do polígono.

2.3. Algoritmo 3 – Inclusão de ponto em polígono convexo com diagrama de Voronoi: pela imagem, pode-se ver que o algoritmo detecta corretamente o polígono em que o ponto atualmente está, tanto pela cor do polígono, quanto por seu número. Ademais, considera-se que, para este algoritmo, a função `ProdVetorial` é chamada zero vezes, pois, conforme explicado anteriormente, ao fazermos a verificação inicial — para verificar se o ponto ainda está no mesmo polígono — quando verificamos que o ponto trocou de polígono, essa função já retorna a aresta do polígono anterior que o ponto cruzou (ou seja, a aresta que tem o ponto à sua direita), e esse valor é passado como parâmetro para a função que implementa este terceiro algoritmo. Com isso, basta consultar na estrutura de dados (carregada na inicialização do programa) do último polígono que sabe-se que o ponto esteve qual é o vizinho da aresta em questão, concluindo, então, qual é o novo polígono sem a necessidade de quaisquer cálculos adicionais.

3. **FUNCIONAMENTO DO PROGRAMA PARA 100 E 500 POLÍGONOS:** na verdade, por se tratar de algoritmos genéricos, o programa deve se comportar da mesma forma para qualquer número de polígonos. É claro que, se chegarmos a uma quantidade absurdamente grande de polígonos, podemos ter problemas de performance, mas o funcionamento do algoritmo será o mesmo e o programa seguirá funcionando, mesmo que não seja performático. Nesse caso, o papel dos envelopes é fundamental para evitar que seja necessário testar contra todos os polígonos do diagrama a cada *frame*. Se não tivéssemos os envelopes, o programa não escalaria para uma quantidade de polígonos muito maior. DIto isso, considere os diagramas abaixo, de 100 e 500 polígonos, respectivamente.

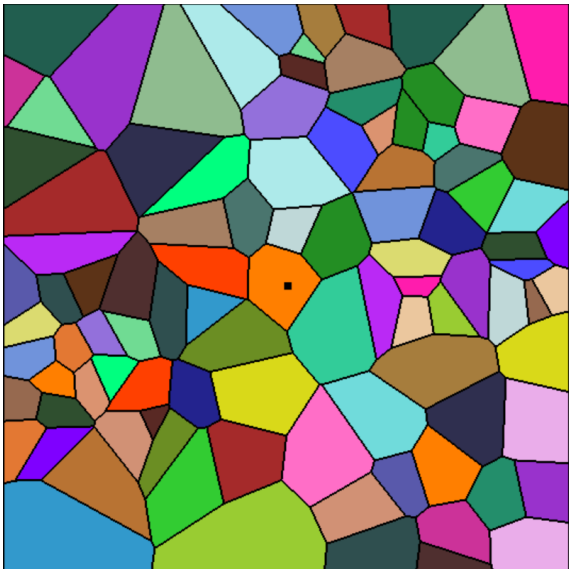

Imagem 4 – Diagrama com 100 polígonos

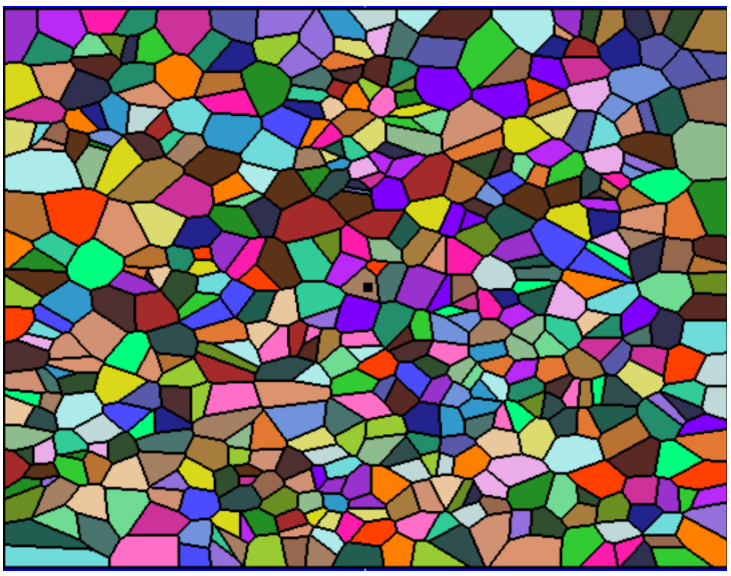

Imagem 5 – Diagrama com 500 polígonos

3.1. **EXPLICAÇÃO PARA 100 POLÍGONOS:** na *Imagem 6*, vemos um exemplo de funcionamento do algoritmo para o diagrama com 100 polígonos, retratado na *Imagem 4*. Há dois polígonos com o envelope ativo, e o de baixo é o que contém o ponto. Sabe-se que o polígono de cima tem o menor índice e, portanto, é testado primeiro. Para o primeiro algoritmo, são feitas doze chamadas à função `HaInterseccao` porque é a soma do número de arestas dos polígonos com os envelopes ativos. Para o segundo algoritmo, são feitas sete chamadas à função `ProdVetorial` porque, ao investigar já a primeira aresta do polígono de cima (de menor

índice), conclui-se que o ponto está à direita dele e fora do polígono; em seguida, testamos contra todas as arestas do próximo polígono com envelope ativo e concluímos que o ponto está à "esquerda" de todas, assim estando dentro do polígono. Por fim, para o último algoritmo, a lógica se mantém e não é necessária nenhuma chamada adicional à função `ProdVetorial`.

3.2. **<u>EXPLICAÇÃO PARA 500 POLÍGONOS</u>:** na *Imagem* 7, vemos um exemplo de funcionamento para o diagrama com 500 polígonos, retratado na *Imagem 5*. Há dois polígonos com o envelope ativo, e o da esquerda é o de menor índice. Portanto, para o primeiro algoritmo, ao analisarmos o número de intersecções com todas as arestas do polígono (8 arestas), concluímos que o ponto está dentro dele e, portanto, encerramos com oito chamadas à função `HaInterseccao`. O cálculo do segundo algoritmo é feito de forma similar: calculamos a lateralidade do ponto em relação a cada aresta do primeiro polígono (8 arestas) e concluímos que o ponto está dentro dele com oito chamadas à função `ProdVetorial`. Novamente, para o terceiro algoritmo, o número de chamadas à função `ProdVetorial` mantém-se nulo.

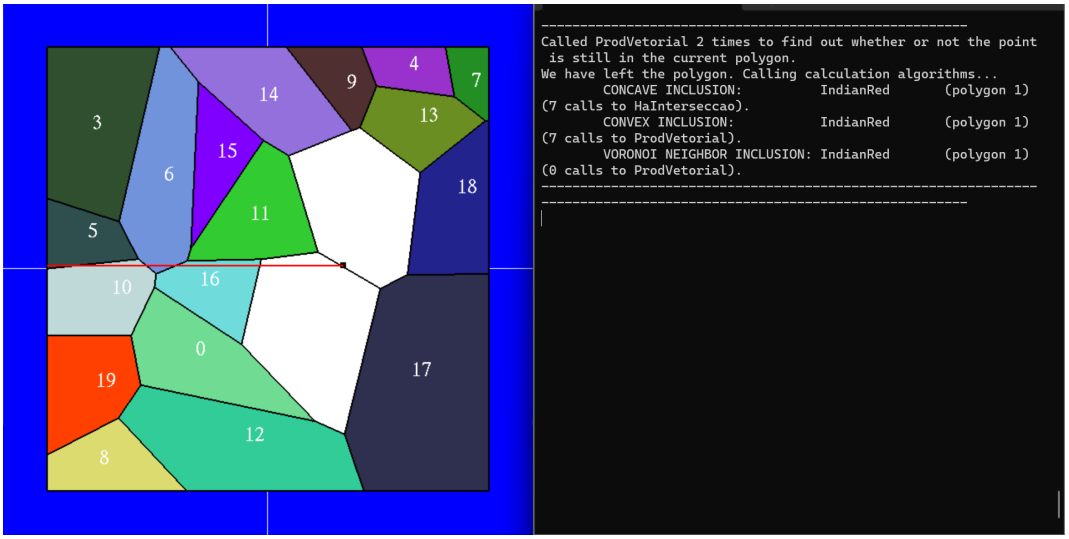

Imagem 2 – Algoritmos em funcionamento para o polígono número 1 no diagrama com 20 polígonos

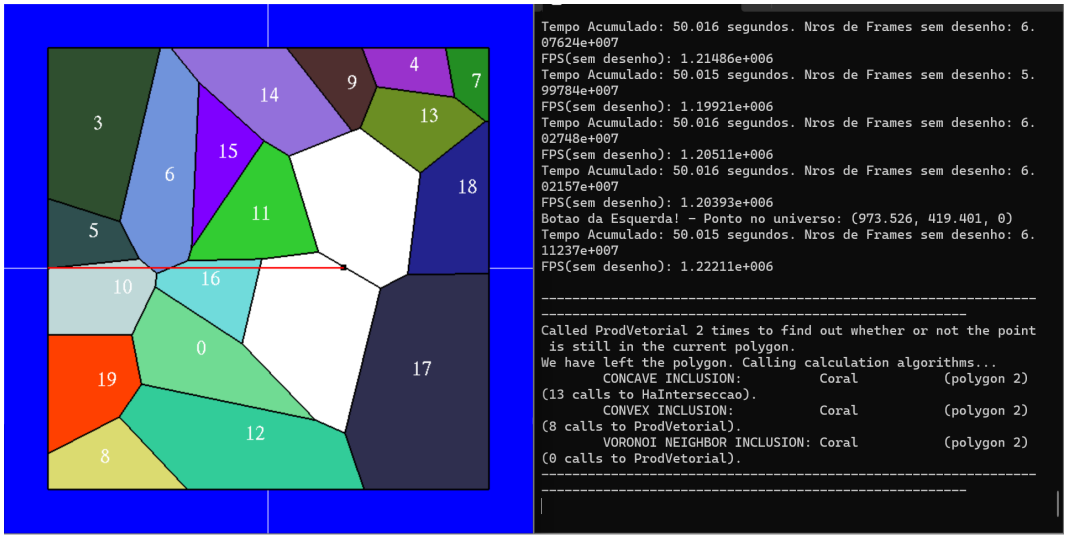

Imagem 3 – Algoritmo em funcionamento para o polígono número 2 no diagrama com 20 polígonos

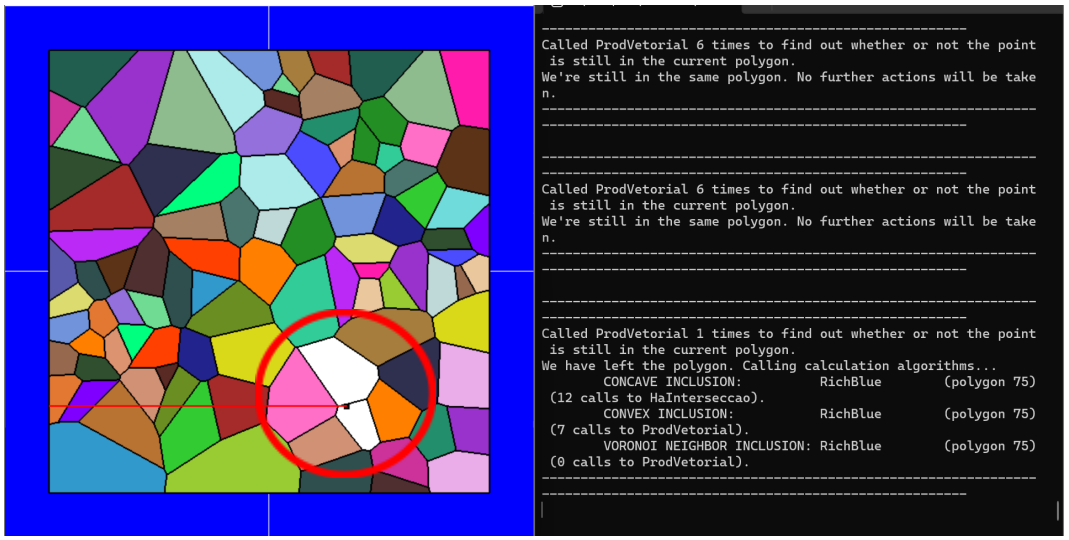

Imagem 6 – Algoritmos em funcionamento para o polígono número 75 no diagrama com 100 polígonos

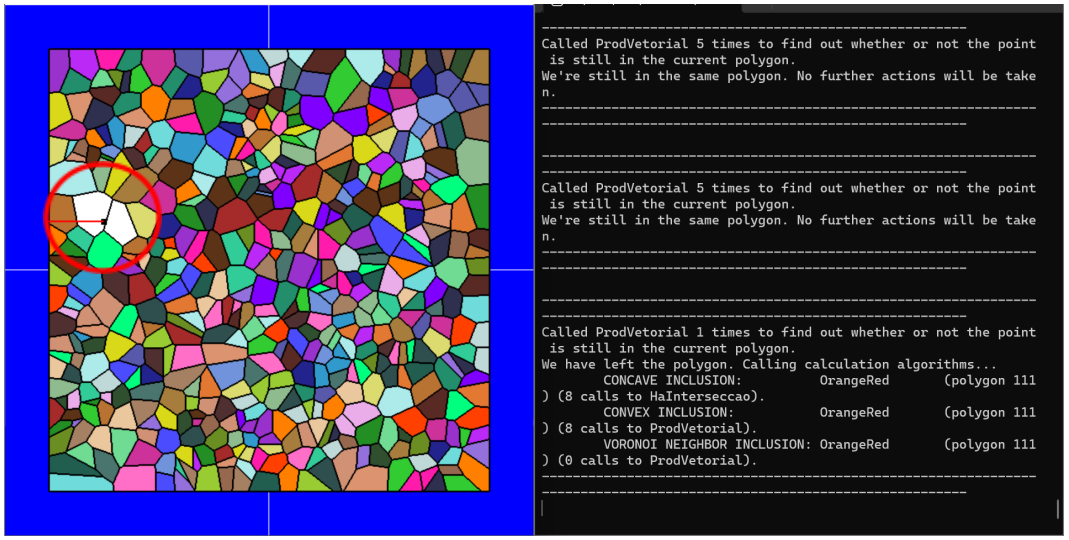

Imagem 7 – Algoritmos em funcionamento para o polígono número 16 no diagrama com 500 polígonos