Operações Booleana em polígonos arbitrários

Aluno: Rafael V. Carvalho

Orientadora: Cristina G. Fernandes
Computer Science Department
IME USP

Overview

Overview

Operações Booleanas

- **Geralmente vemos operações Booleanas em outros contextos como álgebra, lógica, teoria dos conjuntos etc;**
- **Quando se faz uma operação Booleana entre dois polígonos estamos interessados na região resultante dessa operação;**
- **Ex:**
  - A área comum entre dois polígonos (interseção)
  - A área coberta pelos dois polígonos (união)
  - A área de um polígono que não é comum ao outro (diferença)

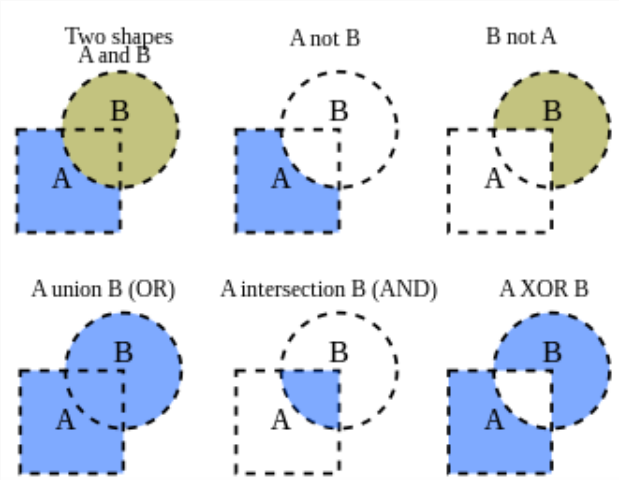


Figure 1: *Operações Booleanas. Imagem tirada de https://en.wikipedia.org/wiki/Boolean_operations_on_polygons*

Por quê queremos descobrir tais regiões?

- **Áreas como computação gráfica precisam dessas operações para determinar superfícies que serão visíveis no monitor;**

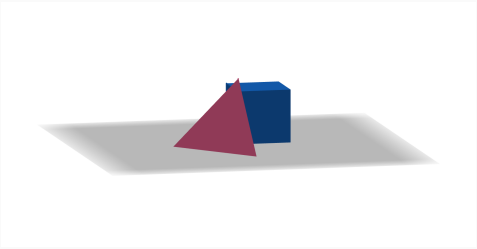


Figure 2: *Dois sólidos geométricos. Note que não é necessário renderizar uma parte da superfície do cubo*

Por quê queremos descobrir tais regiões?

- **Softwares como CAD (Computer Aided Design) precisam dessas operações para criar peças ou instrumentos;**

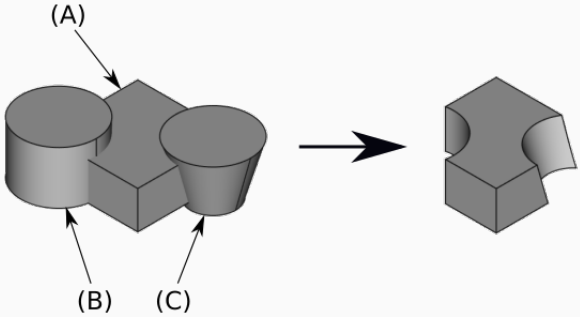


Figure 3: *Formação de uma peça através da operação de diferença. Imagem de https://wiki.freecadweb.org/images/1/16/PartDesign_Boolean_example.png*

Por quê queremos descobrir tais regiões?

- **Softwares como GIS (Geographic Information System) precisam dessa operações para criar mapas;**

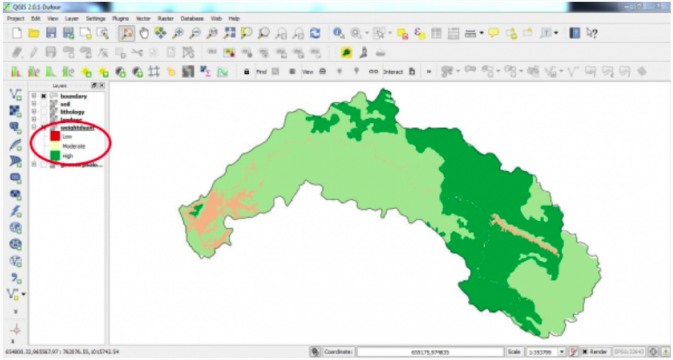


Figure 4: *Operação chamada de weighted overlay pelo software. Fonte https://grindgis.com/software/qgis/raster-overlay-analysis-qgis*

Overview

Definição

- Antes de irmos para o algoritmo precisamos de um "ingrediente" para entender seu funcionamento e também para determinar os tipos de polígonos que lidaremos

Even odd rule

Regra

Dados um ponto q e um polígono P. Trace uma semirreta partindo de q, o número de vezes que essa semirreta se intersecta com a fronteira de P nos diz se $q \in P$ ou $q \notin P$

Even odd rule

Regra

Dados um ponto q e um polígono P. Trace uma semirreta partindo de q, o número de vezes que essa semirreta se intersecta com a fronteira de P nos diz se $q \in P$ ou $q \notin P$

- **Se o número de interseções for ímpar então q está dentro de P**
- **Se o número de interseções for par então q está fora de P**

Even odd rule

Regra

Dados um ponto q e um polígono P. Trace uma semirreta partindo de q, o número de vezes que essa semirreta se intersecta com a fronteira de P nos diz se $q \in P$ ou $q \notin P$

- **Se o número de interseções for ímpar então q está dentro de P**
- **Se o número de interseções for par então q está fora de P**

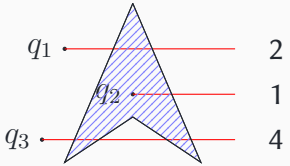


Figure 5: *Pontos que estão dentro ou fora do polígono.*

Porque estamos falando disso?

- Estamos falando da even odd rule por dois motivos

Porque estamos falando disso?

- **Estamos falando da even odd rule por dois motivos**
  - O primeiro é porquê o algoritmo utiliza dessa técnica para fazer uma rotulação;

Porque estamos falando disso?

- **Estamos falando da even odd rule por dois motivos**
  - O primeiro é porquê o algoritmo utiliza dessa técnica para fazer uma rotulação;
  - O segundo é porque os polígono que lidaremos permitem auto-interseção e portanto precisamos definir o que é dentro e fora para esses polígonos;
  - Portanto, fica definido que um polígono é a união de todas regiões cuja even odd rule é ímpar;

Porque estamos falando disso?

- **Estamos falando da even odd rule por dois motivos**
  - O primeiro é porquê o algoritmo utiliza dessa técnica para fazer uma rotulação;
  - O segundo é porque os polígono que lidaremos permitem auto-interseção e portanto precisamos definir o que é dentro e fora para esses polígonos;
  - Portanto, fica definido que um polígono é a união de todas regiões cuja even odd rule é ímpar;

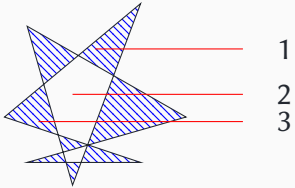


Figure 6: *Região em azul é definida como região do polígono*

Overview

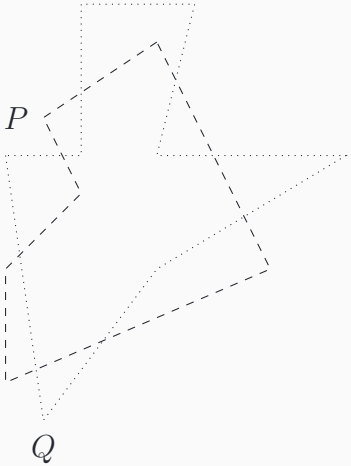
Algoritmo de Greiner-Hormann

- Algoritmo foi proposto em 1989;
- Muitas pessoas enxergam tal algoritmo como elegante por conta da sua simplicidade em resolver o problema;
- Primeiro veremos a operação de interseção, depois expandiremos para as outras.

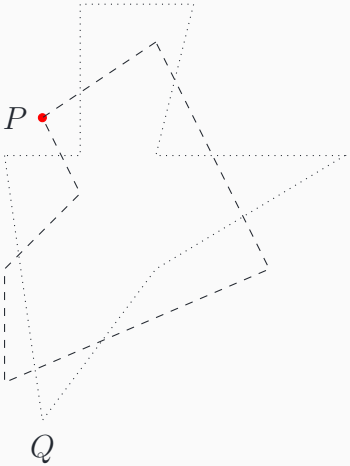
Ideia Geral

- **Dado dois polígonos P e Q;**
- **Suponha que você tem um carrinho de areia com uma escotilha que quando aberta despeja areia;**
- **Suponha que você anda sobre o a fronteira do polígono P no sentido anti-horário;**
- **Se a aresta em que você começou a caminhar estiver dentro de Q, a escotilha começa aberta, caso contrário ela está fechada;**
- **Caminhe sobre a fronteira de P e toda vez que cruzar com uma aresta de Q feche a escotilha se estiver aberta ou abra se a escotilha estiver fechada;**
- **Repita o mesmo processo para Q e agora junte as respostas.**

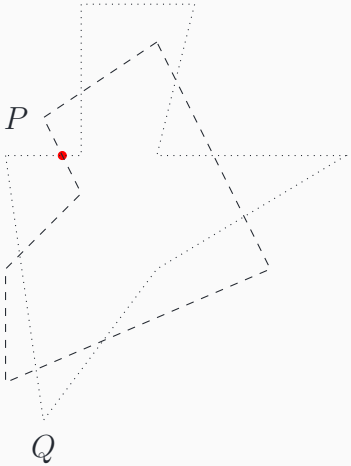
Funcionamento



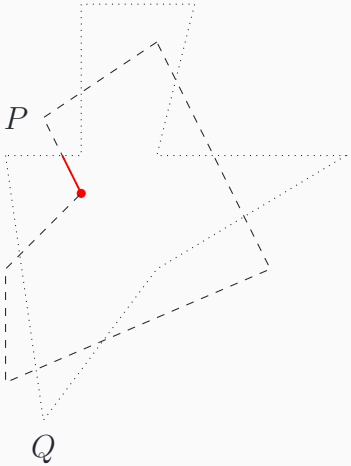
Pintando P



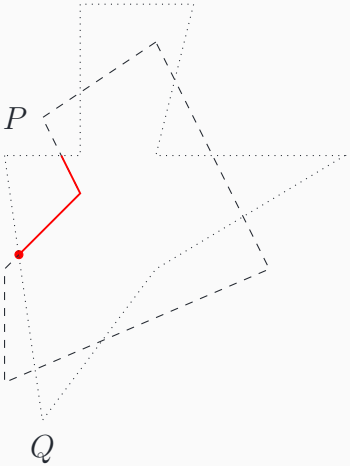
Pintando P



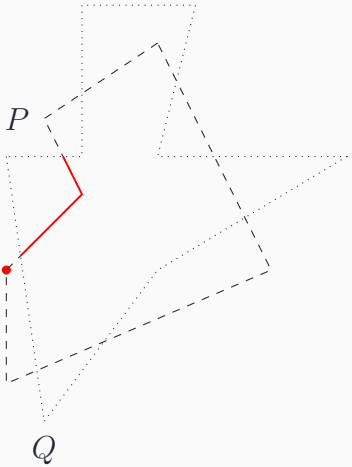
Pintando P



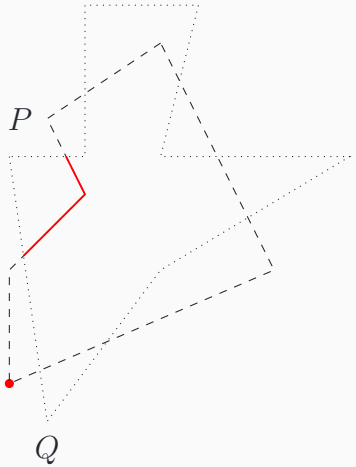
Pintando P



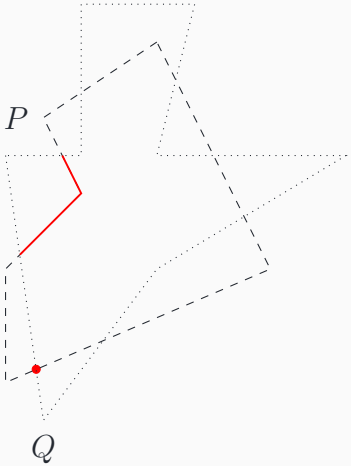
Pintando P



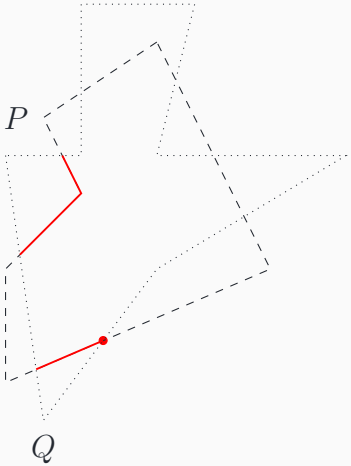
Pintando P



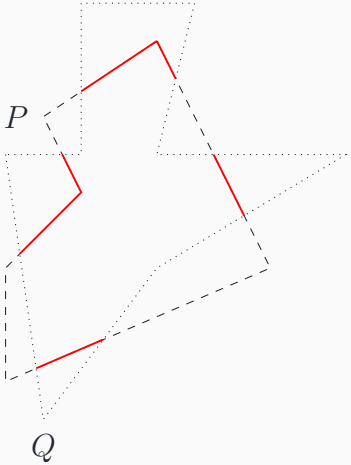
Pintando P



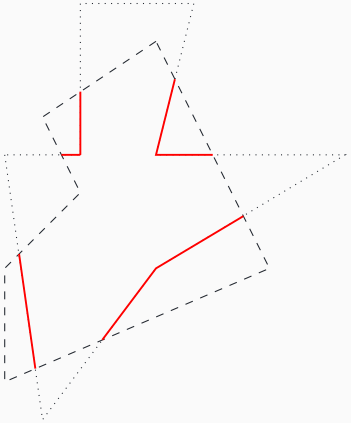
Pintando P



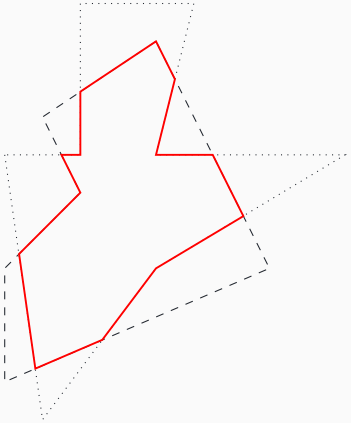
P pronto



Q pronto



Juntando as fronteiras



Overview

Estrutura de dados

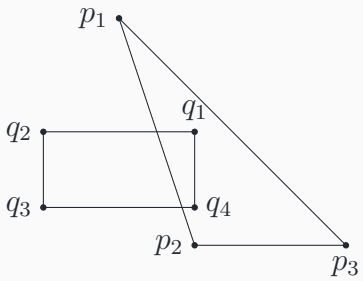
- **Para representar os polígonos usamos uma lista circular duplamente ligada com os seguinte atributos:**
  - vertex - par ordenado (x, y) que representa as coordenadas do vértice;
  - next - apontador para próxima célula;
  - prev - apontador para célula anterior;
  - nextPoly - apontador para o polígono;
  - intersect - flag que indica se um vértice é de interseção;
  - entry_exit - flag para indicar se o vértice de interseção é de entrada ou saída;
  - neighbor - apontador para a célula corresponde do outro polígono;
  - alpha - localização do vértice em relação à aresta;

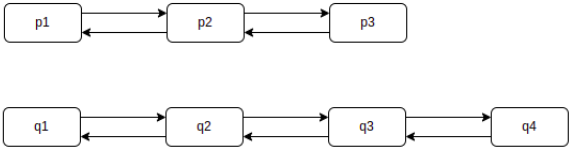
Overview

Encontrar todas as interseções

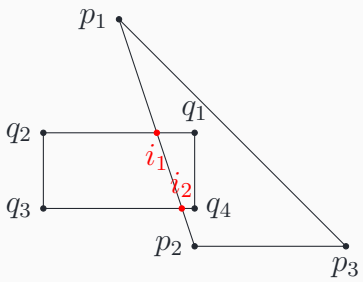
- Aplicaremos um algoritmo de força bruta para encontrar todos os vértices de interseção (testa tudo);
- Apos encontrar um vértice de interseção inserimos uma cópia dele na lista de P e de Q e ligamos eles pelo atributo neighbor;
- Quando criarmos o nó devemos atualizar o campo alpha;

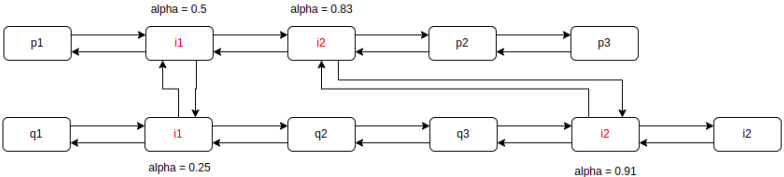
ED Inicial





ED final


p_1
q_1
q_2
i_1
i_2
q_3
q_4
p_2
p_3

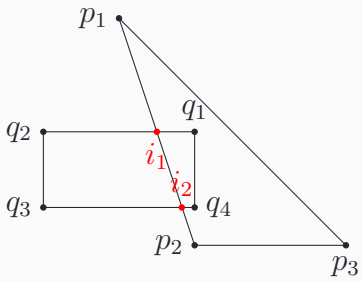

alpha = 0.5
alpha = 0.83
p1
i1
i2
p2
p3
q1
i1
q2
q3
i2
i2
alpha = 0.25
alpha = 0.91

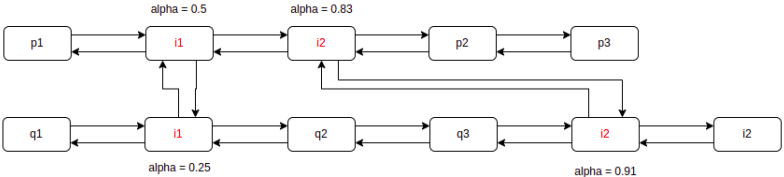
Overview

Rotulando interseções

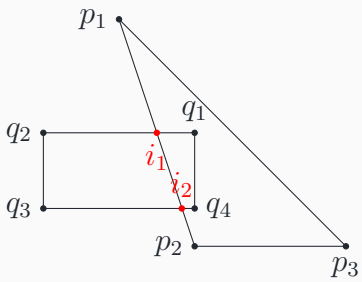
- Essa fase corresponde à quando andamos com o carrinho de areia em cima da fronteira dos polígonos;
- Para realizar tal tarefa executamos a even odd rule no primeiro vértice de P para saber se estamos dentro ou fora de Q;
- Após isso vamos andando sobre P. A medida que encontramos um vértice de interseção sabemos que se estávamos dentro após esse vértice estaremos fora de Q e vice-versa. Chamamos esses vértice de interseção de entrada, se após esse vértice a aresta esta dentro do outro polígono e de saída, se após esse vértice a aresta esta fora do outro polígono;
- Assim como na analogia do carrinho de areia, devemos realizar a rotulação para Q também.

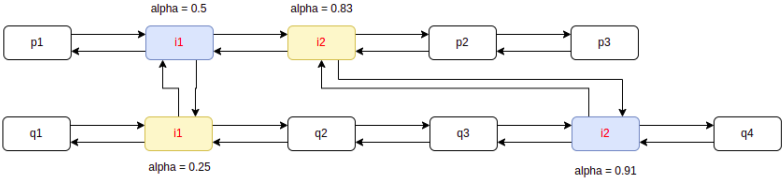
Rotulação inicial





Rotulação final



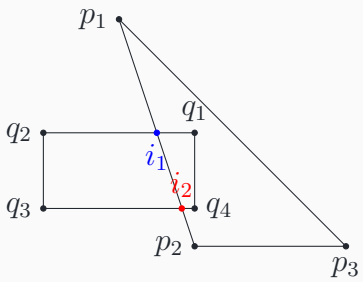


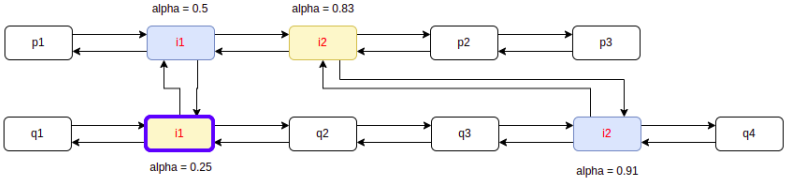
Overview

Construindo o polígono

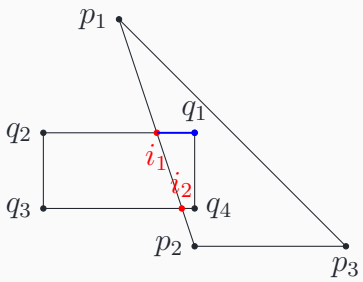
- **Nessa etapa faremos uma filtragem na estrutura de dados para encontrar os vértices que fazem parte do polígono resultante;**
- **Para realizar tal tarefa percorremos o polígono da seguinte maneira:**
  - Encontre o primeiro vértice de interseção não processado;
  - Se o vértice de interseção for de entrada:
    - » *Percorremos a lista, usando o atributo next*
  - Se for de saida:
    - » *Percorremos a lista usando o atributo prev*
  - Percorremos até encontrar um outro vértice de interseção;
  - Após chegar nesse outro vértice pulamos para lista do outro polígono;
  - repetimos esse processo nessa lista até encontrar o vértice de interseção que começamos;
  - Repete-se esse processo até todos os vértices de interseção

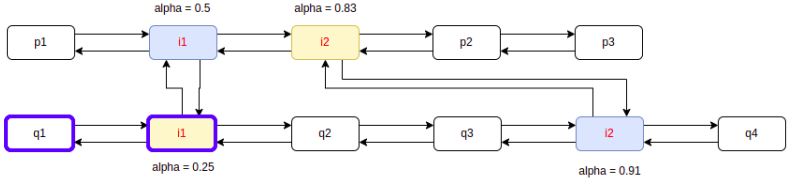
Achando o polígono



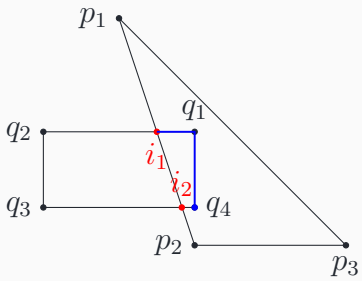


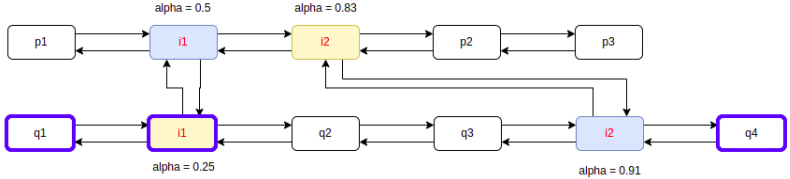
Achando o polígono



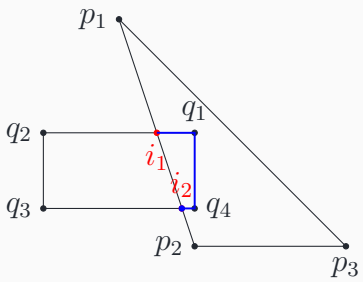


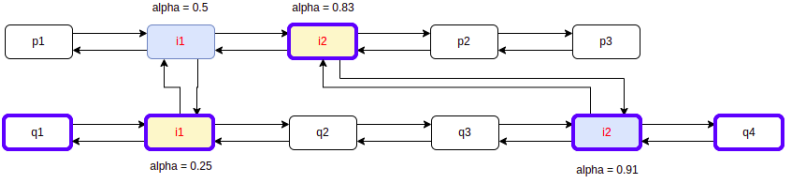
Achando o polígono



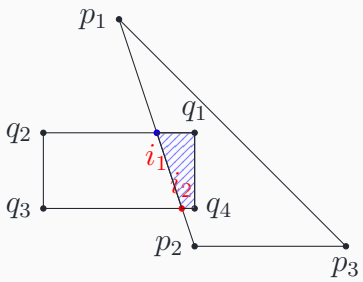


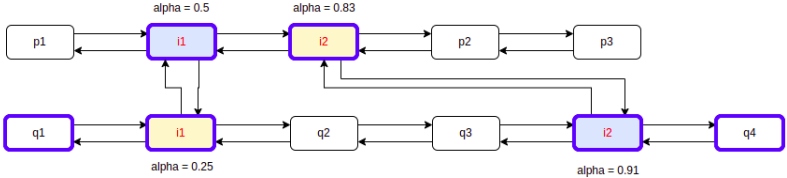
Achando o polígono





Achando o polígono

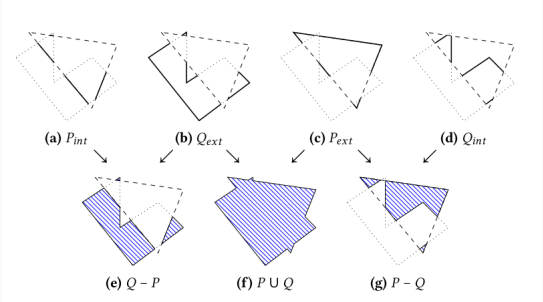




Overview

Outras operações

- Até agora vimos apenas a interseção
- A interseção pode ser vista como a combinação das aresta de P que entram em Q com as arestas de Q que entram em P;
- Para as outras operações basta apenas combinarmos arestas externas com externas, externas com internas etc.



Outras operações

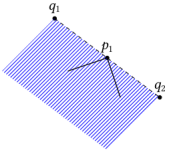
- **Com isso, o algoritmo permanece o mesmo para realizar as outras operações.**

Overview

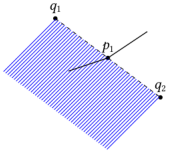
Casos degenerados

- Os casos degenerados ocorrem quando o vértice de interseção tem alpha = 0 ou alpha = 1;
- Isso ocorre quando a interseção ocorre no extremo de uma aresta;
- Quando isso ocorre acrescentamos pelo menos uma cópia desnecessária em uma da lista dos polígonos;
- Para consertar esse erro os autores sugerem para fazer uma perturbação aleatória nos vértices;
- Tal perturbação deve magnitude menor que 1 pixel para não ser visível

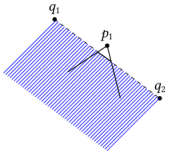
Possíveis configurações para perturbação



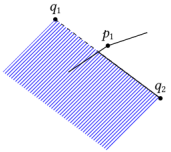
(a) *Configuração inicial com ambas as arestas de P dentro de Q*



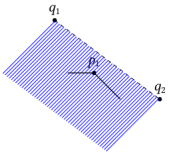
(b) *Configuração inicial com uma das arestas de P dentro de Q*



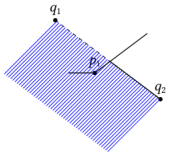
(c) *Configuração* ***(a)*** *com perturbação deixando p_1 fora de Q*



(d) *Configuração* ***(a)*** *com perturbação deixando p_1 fora de Q*



(e) *Configuração* ***(a)*** *com perturbação deixando p_1 dentro de Q*



(f) *Configuração* ***(b)*** *com perturbação deixando p_1 dentro de Q*

Overview

Conclusão

- **Embora o algoritmo de Greiner-Hormann seja simples e relativamente fácil de implementar ele tem alguns aspectos negativos:**
  - O primeiro é que essa perturbação para evitar casos degenerados pode não ser desejável se a aplicação necessita ser precisa;
  - O segundo é que a primeira etapa (encontrar todas as interseções) consome a maior parte do tempo, por conta da estratégia de força bruta.
- **Existem outros algoritmos que resolvem esses dois problemas, mas não trataremos deles aqui**