3
Implementação do Eixo Medial

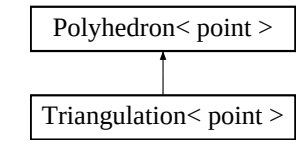
A implementação do eixo medial de uma união de bolas foi feita a partir de [11].

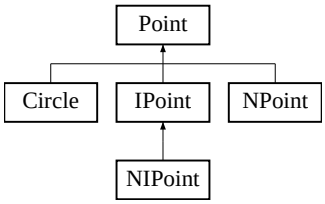
Algoritmo 1 Eixo Medial (Nina Amenta)

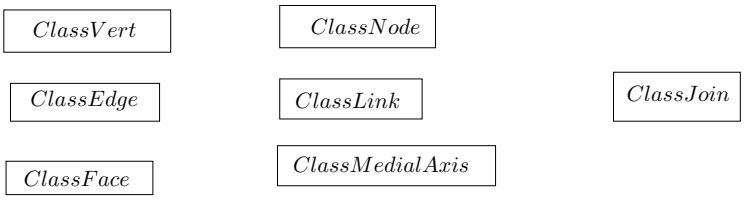
1: Entre com o conjunto de bolas $\mathcal{B}$.
2: Calcule o $\alpha - shape$ $\mathcal{S}_\alpha$ de $\mathcal{B}$.
3: Calcule o conjunto de vértices $\mathcal{V}$ de $\mathcal{U}$ como duais das arestas de $\partial\mathcal{S}_\alpha$.
4: Calcule $Vor(\mathcal{V})$.
5: Adicione a $\mathcal{X}$ todos os vértices de $\partial\mathcal{S}_\alpha$.
6: Adicione a $\mathcal{X}$ todas as arestas singulares de $\mathcal{S}_\alpha$.
7: Adicione a $\mathcal{X}$ todas as arestas de $Vor(\mathcal{V})$ que estão no interior de $\mathcal{S}_\alpha$.
8: Exiba $\mathcal{X}$.

3.1
Classes

A organização das classes para a implementação deste algoritmo foi feita da seguinte maneira







3.2 Class Vert

Nesta classe foi implementada a estrutura de vértice cujas variáveis são

point *p ; Geometria do vértice / Identifica o vértice.
Eid edge ; Inteiro que identifica uma semi-aresta incidente.

3.3 Class Edge

Nesta classe foi implementada a estrutura de semi-aresta. Esta estrutura é assim chamada porque em vez de armazenarmos as arestas de uma malha, armazenamos semi-arestas. Como o próprio nome já diz, uma semi-aresta é a metade de uma aresta. Chamamos de *mate* a semi-aresta que compõe um par, formando uma aresta. Semi-arestas são orientadas e as duas que compõe um par possuem direções opostas. As variáveis desta classe são

Vid vert; Inteiro que identifica o vértice da semi-aresta.
Eid mate; Inteiro que identifica a semi-aresta que compõe um par.
Eid next; Inteiro que identifica a próxima semi-aresta na face.
Fid face; Inteiro que identifica uma face incidente.
Eid dual; Inteiro que identifica a semi-aresta dual.
double alpha; Raio do círculo circunscrito .

3.4 Class Face

Nesta classe foi implementada a estrutura de face cujas variáveis são

Eid edge; Inteiro que identifica uma semi-aresta da face.
point *center; Geometria do Centro de Voronoi correspondente.
double alpha; Raio do círculo circunscrito.

3.5 Class Polyhedron

Nesta classe foi implementada a estrutura base para a construção das triangulações Regular e de Delaunay bem como para a construção dos diagramas de Voronoi e de Potências. Esta classe possui as seguintes variáveis

VertContainer verts; Vértices do poliedro (vetor de vértices).
EdgeContainer edges; Arestas do poliedro (vetor de semi-arestas).
FaceContainer faces; Faces do poliedro (vetor de faces).

E as seguintes funções públicas

– void star(const Vid v, list< Fid > st):

Coloca as faces incidentes ao vértice v em uma lista.

– void radial star(const Vid v, stack< Eid > st):

Coloca as arestas incidentes ao vértice v em uma pilha.

– Fid add triangle(const Vid v1, const Vid v2, const Vid v3):

Cria uma face de dimensão 3, isto é, um triângulo.

– void match(const Eid e1, const Eid e2):

Identifica semi-arestas que compõe um par.

– const point set centers(const Fid f):

Calcula o centro de Voronoi associado à face f e atribui o valor da variável alpha.

– bool check():

Verifica se o poliedro foi construído corretamente.

– Polyhedron<point> *dual():

Constrói o dual da triangulação atual , isto é , os diagramas de Voronoi ou de Potências.

bool is in face3 (Fid f, const point p):

Verifica se um ponto p pertence à face f de dimensão 3.

3.6 Class Triangulation

Esta classe herdou a estrutura da classe *Polyhedron*. Aqui foram implementadas todas as funções relacionadas à criação das triangulações Regular/Delaunay de um determinado conjunto de pontos. São elas

Públicas:

– void triangulate(vector<point> points, vector<point> centers):

Cria a triangulação Regular/Delaunay de um conjunto de pontos (points) e guarda o centro de voronoi associado a cada face em um vetor (centers);

Protegidas:

– void legalize(const Eid e, stack<Eid> st):

Verifica se e é uma aresta válida.

– bool insert(const Vid v, vector<point> centers):

Insere um vértice v.

– void einsert(const Vid v, const Eid e, vector<point> centers):

Insere um vértice v que pertence à aresta e.

– void finsert(const Vid v,const Fid f, vector<point> centers):

Insere um vértice v que pertence ao interior do triângulo f.

O algoritmo foi baseado em [8].

Algoritmo 2 Triangulação Regular (triangulate)

1: Entre com o conjunto de bolas $\mathcal{B}$.
2: Encontre um triângulo T que contenha os centros das bolas de $\mathcal{B}$ e adicione T à triangulação Regular $\mathcal{R}$.
3: Insira uma bola $b \in \mathcal{B}$ em $\mathcal{R}$ e ache um triângulo $t \in \mathcal{R}$ que contenha o centro de b (insert).
4: Caso b pertença ao interior de t , crie três novas arestas de b até cada um dos vértices de t (finsert).
5: Caso b pertença a uma aresta e de t , crie duas novas arestas (se e não é de bordo) ou uma nova aresta (se e é de bordo)(einsert).
6: Verifique se as arestas criadas são válidas (legalize).
7: Volte para o ítem 3.
8: Exiba $\mathcal{R}$.

O algoritmo para a triangulação de Delaunay, bem como as funções envolvidas são os mesmos.

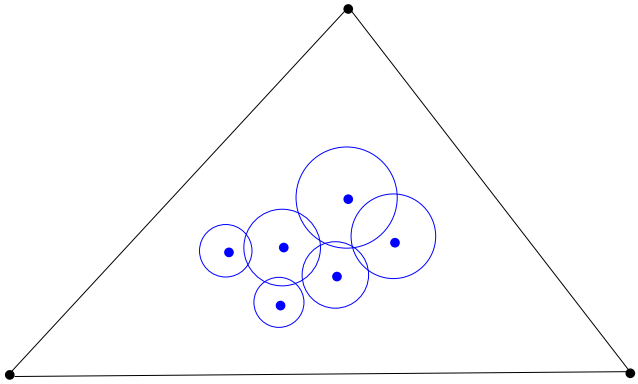


Figure 3.1: Triângulo T que contém todos os pontos de $\mathcal{B}$

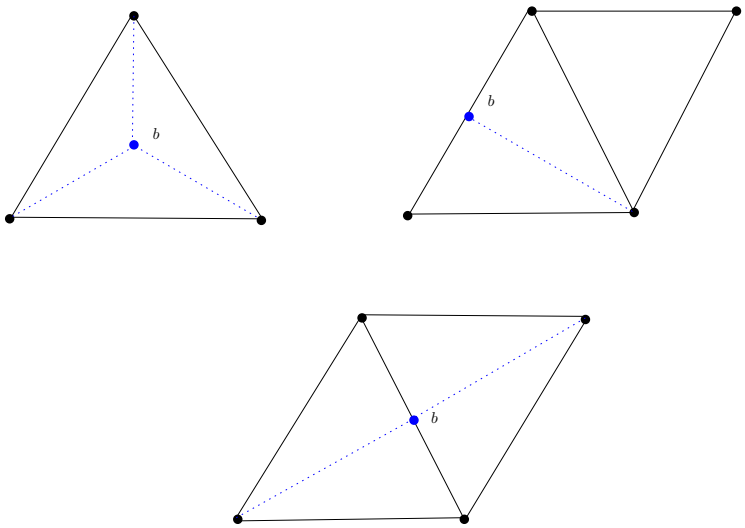


Figure 3.2: b pertence ao interior de t / b pertence a uma aresta e de t

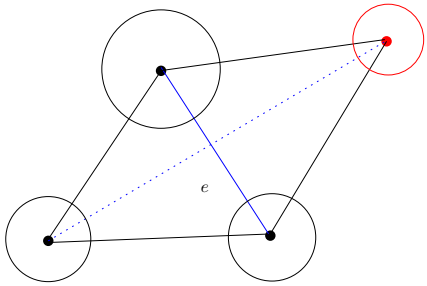


Figure 3.3: Legalize: Esta função troca arestas ilegais por legais através de *edge flips*

3.7 Class Point

Nesta classe foi implementada a estrutura de ponto em $\mathbb{R}^2$ cujas variáveis são

double x, y; Coordenadas do ponto.

Além das funções habituais das operações de soma, subtração, distância euclidiana, etc

- Point operator += (const Point p)
- Point operator -= (const Point p)

- Point operator *= (const double s)
- Point operator /= (const double s)
- double operator * (const Point p)
- bool operator == (const Point p)
- bool operator != (const Point p)
- double norm()
- void normalize()
- double dist(const Point p)

Esta classe possui também as seguintes funções públicas

- Point center(const Point p1, const Point p2, double alpha)

  Centro de Voronoi de 2 pontos. O valor de alpha representa o raio do círculo cujo diâmetro é o comprimento da aresta que liga p1 a p2.

- Point center(const Point p1,const Point p2,const Point p3, double alpha)

  Centro de Voronoi de 3 pontos. O valor de alpha representa o raio do círculo determinado por p1, p2 e p3.

- bool valid(const Point p1,const Point p2,const Point p3, const Point p4)

  Decide se o triângulo de vértices p1, p2, p3 é válido em relação ao ponto p4. Este teste é parte da função legalize, da classe Triangulation. No caso de pontos no plano, um triângulo é válido em relação a um ponto p se o círculo determinado pelos seus vértices não contém p. Condições para a aplicação deste teste serão discutidas no próximo capítulo.

3.8 Class Circle

Nesta classe foi implementada a estrutura de círculo. Esta classe herdou a estrutura da classe *Point* e possui a seguinte variável adicional

double r; Raio do círculo.

E as seguintes funções públicas

- double dist(const Circle p):

  Potência entre círculos.

- Circle center(const Circle p1,const Circle p2,double alpha):

  Centro de Voronoi de 2 círculos. Este cálculo será apresentado abaixo. Ele representa a interseção entre o eixo radical de p1 e p2 com a reta que liga os respectivos centros. O volor de alpha representa a potência entre este ponto de interseção com p1 ou p2.

- Circle center(const Circle p1,const Circle p2,const Circle p3,double alpha) :

  Centro de Voronoi de 3 círculos. Este cálculo também será apresentado abaixo. Ele representa a interseção dos 3 eixos radicais e o valor de alpha representa a potência do centro radical com qualquer um dos círculos p1, p2, p3.

- bool valid(const Circle p1,const Circle p2,const Circle p3,const Circle p4):

  Decide se o triângulo de vértices p1, p2, p3 é válido em relação ao ponto p4. Este teste é parte da função legalize, da classe Triangulation. O algoritmo deste teste para o caso de pontos com peso e condições para a sua aplicação serão discutidos no próximo capítulo.

- *int inter (const Circle p1, const Circle p2, IPoint i1, IPoint i2):*

  Calcula a interseção entre os círculos p1, p2 e guarda em i1 ,i2 . A estrutura de IPoint e sua importância serão discutidos mais adiante.

– *bool is in(const Point p):*

p pertence ao círculo ?

Algoritmo 3 Circle center(const Circle p1, const Circle p2, double alpha)

$r1 \leftarrow p1x * p1x + p1y * p1y - p1r * p1r;$
$r2 \leftarrow p2x * p2x + p2y * p2y - p2r * p2r;$
$dx \leftarrow p2x - p1x;$
$dy \leftarrow p2y - p1y;$
$d \leftarrow p1y * p2x - p1x * p2y;$
$det \leftarrow -2 * (dx * dx + dy * dy);$
$det1 \leftarrow (r1 - r2) * dx + 2 * dy * d;$
$det2 \leftarrow (r1 - r2) * dy - 2 * dx * d;$
$det3 \leftarrow 2*dx*(r1*p2.x()-r2*p1.x())+2*dy*(r1*p2.y()-r2*p1.y())-4*d*d;$
$alpha \leftarrow ((det1 * det1 + det2 * det2) - (det * det3))/(det * det);$
retorna $Circle(det1/det, det2/det);$

Algoritmo 4 Circle center (const Circle p1, const Circle p2, const Circle p3, double alpha)

$r1 \leftarrow p1x * p1x + p1y * p1y - p1r * p1r;$
$r2 \leftarrow p2x * p2x + p2y * p2y - p2r * p2r;$
$r3 \leftarrow p3x * p3x + p3y * p3y - p3r * p3r;$
$det \leftarrow -4*(p1x*p2y+p1y*p3x+p2x*p3y-p2y*p3x-p1x*p3y-p1y*p2x);$
$det1 \leftarrow -2 * (r1 * p2y + p1y * r3 + r2 * p3y - p2y * r3 - r1 * p3y - p1y * r2);$
$det2 \leftarrow -2 * (r2 * p1x + r1 * p3x + r3 * p2x - p3x * r2 - r3 * p1x - p2x * r1);$
$det3 \leftarrow 4 * (r3 * p2y * p1x + p1y * r2 * p3x + r1 * p3y * p2x - r1 * p2y * p3x - p1x * r2 * p3y - p1y * p2x * r3);$
$alpha \leftarrow (det1 * det1 + det2 * det2 - det * det3)/(det * det);$
retorna $Circle(det1/det, det2/det);$

3.9 Class IPoint

Estrutura do conjunto de vértices $\mathcal{V}$ da união de bolas. Esta classe herdou a estrutura da classe *Point* e possui a seguinte variável adicional

Eid edge; Inteiro que identifica a semi-aresta de $\partial\mathcal{S}_\alpha$ dual ao vértice da união de bolas.

Esta estrutura nos permite relacionar o conjunto de vértices $\mathcal{V}$ com o $\mathcal{S}_\alpha$ e, consequentemente, com o conjunto de bolas. Esta estrutura foi importante, por

exemplo, quando precisamos descartar os pontos de interseção entre círculos que não faziam parte da fronteira da união de bolas.

3.10 Class NPoint

Estrutura de ponto para os nós do eixo medial. Esta classe também herdou a estrutura de *Point* e possui a seguinte variável adicional

Nid node; Inteiro que identifica o centro de $Vor(\mathcal{V})$ correspondente ao nó do eixo medial.

3.11 Class NIPoint

Estrutura de ponto para os centros de voronoi de $Vor(\mathcal{V})$. Esta classe herdou a estrutura de *IPoint* e possui a seguinte variável adicional

Nid node; Inteiro que identifica o nó do eixo medial correspondente ao centro de $Vor(\mathcal{V})$.

3.12 Class Node

Nesta classe foi implementada a estrutura de nó cujas variáveis são

point *p; Geometria do nó /Identifica o nó.
LinkContainer links; Identifica as ligações incidentes (vetor de ligações).

3.13 Class Link

Nesta classe foi implementada a estrutura de ligação cujas variáveis são

Nid node1; Inteiro que identifica o primeiro nó.
Nid node2; Inteiro que identifica o segundo nó.

3.14 Class Medial Axis

Nesta classe foi implementada a estrutura de grafo para a criação do eixo medial. Um grafo é um conjunto de pontos, chamados nós, conectados por linhas, chamadas de ligações. As variáveis desta classe são

NodeContainer nodes; Nós do eixo medial (vetor de nós).
LineContainer lines; Ligações do eixo medial (vetor de ligações).

3.15 Class Join

Nesta classe foi guardada a estrutura completa para a implementação do algoritmo do eixo medial de união de bolas que abre este capítulo e também para a implementação da evolução das bolas. As variáveis que compõe esta classe são

vector<Circle> original; Bolas Originais ($\mathcal{B}$).
vector<NIPoint> ibound; Vértices da união de bolas ($\mathcal{V}$).
vector<NPoint> maverts; Nós do eixo medial.
vector<NIPoint>vverts ; Centros de $Vor(\mathcal{V})$.
Triangulation<Circle> regular; Triangulação Regular de $\mathcal{B}$.
Polyhedron<NIPoint> voronoi; Diagrama de Voronoi de $\mathcal{V}$.
MedialAxis<NPoint> medial; Eixo Medial .

Esta classe possui as seguintes funções públicas

– void compute medial():

Constrói o Eixo Medial.

– void compute():

Constrói todas as estruturas da classe.

– void move():

Evolução das bolas originais.

– Point direction(const Nid n):

Calcula a direção do movimento de cada bola.

– void holes(vector<Circle> orig):

Elimina as bolas supérfluas da polibola em cada etapa da evolução, isto é, mantém a escrita mínima.

E as seguintes funções protegidas

– void compute instersection points():

Calcula todos os pontos de interseção entre as bolas.

– void select intersection points():

Seleciona os pontos calculados em **instersection points** *que estão na fronteira da união de bolas, isto é, os vértices* $\mathcal{V}$ *da união de bolas.*

– void select voronoi points():

Seleciona os centros de $Vor(\mathcal{V})$ *que são nós do eixo medial.*