4
Implementação

4.1
Biblioteca CGAL

CGAL(Computational Geometry Algorithms Library) é uma biblioteca de estruturas de dados e algoritmos geométricos confiáveis e robustos, escrita em C++ e que vem sendo desenvolvida por nove instituições: Instituto Max Plank (Alemanha), INRIA Sophia-Antipolis (França), Universidade de Tel-Aviv (Israel), GeometryFactory (França), ETH Zurich (Suíça), Universidade de Utrecht (Países Baixos), Universidade Livre de Berlim (Alemanha), Forth (Grécia), SciSoft (Argentina). Seu principal objetivo é proporcionar implementações de objetos e algoritmos geométricos e, com isso, facilitar o desenvolvimento de aplicações geométricas mais complexas, através da reutilização de seus componentes.

CGAL foi desenvolvida por diferentes grupos de usuários e é usada para diversos propósitos. Alguns pesquisadores trabalham diretamente com geometria computacional e utilizam a biblioteca para desenvolver e testar seus próprios algoritmos, enquanto pesquisadores de outras áreas usam recursos de geometria computacional em suas aplicações. Há ainda o uso comercial do CGAL, no qual empresas utilizam a biblioteca para o desenvolvimento de aplicações industriais. Os grupos de usuários de CGAL são, portanto, um pouco heterogêneos e, devido a isso, demandam diferentes recursos. Para suprir essa demanda, CGAL procura combinar robustez, generalidade, eficiência e facilidade de uso, apesar de seu uso não ser tão trivial. Como referência, indicamos (1).

Nossa escolha pelo uso do CGAL foi, principalmente, pelo fato de:

- oferecer tipos de números especiais, com maior precisão, evitando, assim, erros de arredondamento;
- apresentar ao usuário mais de uma opção de algoritmo para resolver determinado problema. Cabe ao usuário decidir qual é o mais apropriado para sua aplicação, aumentando a eficiência do algoritmo.

– o uso de *typedefs* da linguagem C++ para evitar contato direto do usuário com os *templates*, que muitas vezes são complicados para usuários menos experientes.

– acessar diversos algoritmos comprovados (como 0-shape e diagrama de Voronoi) numa mesma estrutura.

4.1.1 Descrição da Biblioteca CGAL

Para obter flexibilidade e eficiência, CGAL está divida em três principais camadas, vide Figura 4.1 e como referência indicamos (1).

1. Os núcleos (*kernels*), onde estão implementadas as representações dos números nas coordenadas cartesianas e homogêneas e alguns tipos de números com maior precisão em relação aos tipos básicos de C++. O núcleo contém também objetos geométricos primitivos (como, por exemplo, pontos, retas, semi-retas e círculos) e predicados (operam objetos e retornam valores booleanos ou tipos enumerados, como a comparação de resultados - igual, menor ou maior - e testes de orientação - colinear, lado esquerdo ou lado direito). CGAL fornece a implementação do núcleo em $2D$ e $3D$, para objetos em dimensão arbitrária e para objetos curvos em $2D$.

2. A biblioteca básica, onde estão implementados algoritmos geométricos básicos e algumas estruturas de dados, como

   - algoritmos para determinar o fecho convexo de um conjunto de pontos.

   - operações com polígonos, como o cálculo da área, sua orientação, verificação de sua simplicidade ou convexidade e etc.

   - triangulações, 0-shape, etc.

3. A última camada da biblioteca consiste de facilidades de suporte não geométricos, assim como fornecimento de tipos de números, *handles* ("ponteiros inteligentes"), *circulators*, etc.

Essas três partes do CGAL interagem para qualquer algoritmo da biblioteca, resultando numa biblioteca extremamente flexível. Como dito anteriormente, cabe ao usuário determinar os recursos necessários para sua aplicação. Além disso, se um predicado e/ou algoritmo geométrico do núcleo não satisfazem as necessidades de uma aplicação, o usuário pode implementar seus próprios predicados e objetos geométricos. Porém, essa flexibilidade resulta também numa dificuldade para adaptar o CGAL a um problema específico.

Na CGAL existem as *traits classes*, onde estão implementados os predicados e objetos geométricos básicos. Todo algoritmo da biblioteca básica é parametrizada por uma *traits class* e, por *default*, essa *traits class* é o núcleo. O usuário pode definir sua própria *traits class* e utilizá-la com os algoritmos da biblioteca básica, desde que ela obedeça aos requisitos definidos por tais algoritmos. A utilização de programação genérica proporciona aos usuários de CGAL grande flexibilidade, tornando seu uso bastante atraente.

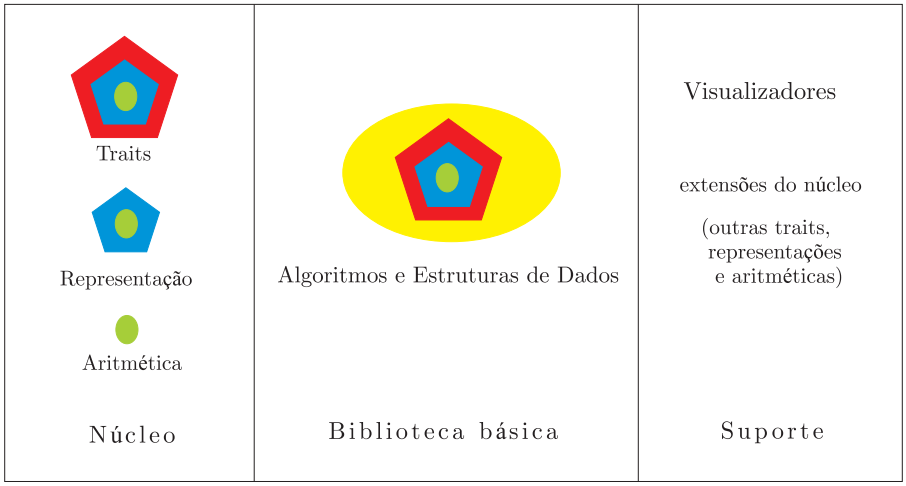

Figura 4.1: Estrutura do CGAL.

Algumas rotinas do nosso programa utilizaram a biblioteca CGAL.

4.2 Implementação do Eixo Medial

O algoritmo para extração do eixo medial de uma união de bolas em $\mathbb{R}^3$ foi realizado a partir de (13) usando a biblioteca CGAL. A idéia do algoritmo é marcar os vértices do $Vor(\mathcal{V})$ como pertencentes ou exteriores ao 0-shape $\mathcal{S}$. Vértices no bordo de $\mathcal{S}$ são marcados como pertencentes a $\mathcal{S}$. A face do $Vor(\mathcal{V})$ onde todos os vértices pertencem a $\mathcal{C}$ e as faces singulares do 0-shape $\mathcal{S}$ consistirão do eixo medial.

Seguindo os resultados enunciados no capítulo 3, as principais etapas da construção do eixo medial estão programadas no algoritmo 1. Como visto no capítulo 2, o cálculo das interseções externas tem a complexidade do cálculo do 0-shape, ou seja $O(n \cdot \log(n))$ em média. O diagrama de Voronoi tem a mesma complexidade, mas com relação ao número de interseções externas. Este número depende a priori linearmente do número de bolas. Os testes de interseção entre o diagrama de Voronoi e a componente regular do 0-shape pode ser implementado com complexidade constante para cada aresta do Voronoi.

Portanto, o algoritmo inteiro de cálculo do eixo medial tem complexidade $O(n \cdot \log(n))$ em média.

Algoritmo 1 Eixo Medial

1: Entre com o conjunto de bolas $\mathcal{B}$.
2: Calcule o 0-shape $\mathcal{S}$ do conjunto de bolas $\mathcal{B}$.
3: Adicione todas as arestas (singulares e regulares) de $\mathcal{S}$ em $\mathcal{X}$.
4: Adicione todos os triângulos singulares de $\mathcal{S}$ em $\mathcal{X}$.
5: Calcule os vértices $\mathcal{V}$ da união $\mathcal{U}$ como os duais dos triângulos do bordo (faces dos tetraedros) de $\mathcal{S}$.
6: Calcule o diagrama de Voronoi $Vor(\mathcal{V})$.
7: Determine todas as interseções do diagrama de Voronoi $Vor(\mathcal{V})$ com os triângulos regulares (tetraedros) do 0-shape $\mathcal{S}$.
8: Adicione a $\mathcal{X}$ todas as faces do diagrama de Voronoi de $\mathcal{V}$ no qual todos os vértices pertencem a $\mathcal{S}$.
9: Retorne $\mathcal{X}$.

4.2.1 Classe

As classes utilizadas foram a **class Point**, definida por três coordenadas e o peso (raio da bola), a **class Triangle**, definida por três vértices e a **class Polygon**.

4.2.2 Implementação do Alpha-Shape

Rotina para a criação do objeto 0-shape usando a biblioteca CGAL está descrita no algoritmo 2.

Algoritmo 2 0-shape

1: Entre com o conjunto de bolas $\mathcal{B}$.
2: Construa uma lista de pontos com peso.
3: Itere sobre as bolas e coloque na lista de pontos.
4: Aloque, utilizando CGAL, o α-shape no modo 'GENERAL' usando os pontos com peso da lista e faça α=0.
5: Retorne $*\&as$ (ponteiro para referenciação dos *handles*), onde será alocada a estrutura do 0-shape.

4.2.3 Rotina para encontrar os vértices $\mathcal{V}$ da união $\mathcal{U}$

A idéia do rotina é encontrar o circumcentro das faces da componente regular do 0-shape $\mathcal{S}$, traçar a normal passando pelo circumcentro e calcular a

interseção com a bola do bordo mais próxima. Os vértices $\mathcal{V}$ que encontraremos são duais aos triângulos do $\partial\mathcal{C}$.

4.2.4 Rotina do diagrama de Voronoi

Rotina para desenhar o diagrama de Voronoi como dual da triangulação de Delaunay usando a biblioteca CGAL está descrita no algoritmo 3. Os itens 2 e 3 constroem as arestas do diagrama de Voronoi, enquanto os itens 4, 5 constroem os polígonos.

Algoritmo 3 Diagrama de Voronoi

1: Itere sobre as faces de Delaunay.
2: Crie o objeto dual e adicione o dual da face.
3: Retorne o diagrama de Voronoi.

Observe que o dual da face pode ser raio, segmento ou ponto (se for infinito).

4.2.5 Interseção do diagrama de Voronoi

A rotina que determina a interseção do diagrama de Voronoi com as partes regulares do 0-shape está descrita no algoritmo 4.

Algoritmo 4 Parte regular do Alpha-Shape

1: Percorra todas as arestas de Delaunay, cujos vértices são as interseções.
2: Crie o polígono dual da aresta.
3: Classifique o polígono como interior ou exterior.
4: Retorne o polígono que corresponde ao eixo medial.