Implementação do Plano Projetivo Orientado na Biblioteca CGAL

Alessandra Guaracy de Oliveira

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Implementação do Plano Projetivo Orientado na Biblioteca CGAL

Alessandra Guaracy de Oliveira

Dezembro de 2004

Banca Examinadora:

- Pedro J. de Rezende (Orientador)
- Wu, Shin-Ting
  F.E.E.C. - Unicamp
- Alexandre X. Falcão
  I.C. - Unicamp
- Siome K. Goldenstein
  I.C. - Unicamp

Substitua pela ficha catalográfica

Implementação do Plano Projetivo Orientado na Biblioteca CGAL

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Alessandra Guaracy de Oliveira e aprovada pela Banca Examinadora.

Campinas, 20 de dezembro de 2004.

Pedro J. de Rezende (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Substitua pela folha com a assinatura da banca

Prefácio

CGAL (Computational Geometry Algorithms Library) é uma biblioteca de estruturas de dados e algoritmos geométricos confiáveis que vem sendo desenvolvida de forma cooperativa por um consórcio formado por instituições na Europa e em Israel.

Os algoritmos de CGAL estão implementados sobre a geometria Euclidiana, onde, geralmente, é necessário tratar muitos casos especiais.

A geometria projetiva orientada engloba a geometria Euclidiana e em ambas, existe a noção de convexidade e de orientação [Sto91]. Como mencionado em [Sto91], algoritmos desenvolvidos sobre a geometria projetiva orientada são mais simples e sucintos e, além disso, o uso de coordenadas homogêneas simplifica as fórmulas e evita operações de divisão, as quais, muitas vezes, podem gerar imprecisão nos resultados dos algoritmos. Sendo assim, o objetivo deste trabalho foi estender para o plano projetivo orientado (PPO), vários algoritmos da biblioteca CGAL implementados em $\mathbb{R}^2$ e comprovar a redução do número de casos tratados.

Dentre os algoritmos desenvolvidos, verificou-se que vários deles apresentaram soluções mais homogêneas no PPO, enquanto outros, em razão de características deste espaço, requerem o tratamento de alguns casos especiais.

Observou-se que uma das grandes vantagens do PPO é poder representar pontos no infinito e distâncias infinitas, assim como compará-las relativamente.

Verificou-se ainda que, no PPO, é mais difícil projetar algoritmos por varredura do que em $\mathbb{R}^2$, pois, como mostrado no capítulo 5, é necessário ter um certo cuidado com a identificação do ponto de parada.

Desta forma, podemos concluir que alguns algoritmos são mais propícios ao PPO, enquanto outros podem apresentar a necessidade de tratamento de casos especiais. Sendo assim, recomenda-se um estudo minucioso do algoritmo antes de optar por implementá-lo em $\mathbb{R}^2$ ou estendê-lo para o PPO.

Agradecimentos

Em primeiro lugar, agradeço a Deus por estar concluindo mais esta jornada.

Gostaria de agradecer ao professor Pedro J. de Rezende por toda a orientação na produção deste trabalho e por sua prontitude em sanar minhas dúvidas.

Agradeço também ao meu companheiro de trabalho e grande amigo Fábio Pakk Selmi-Dei que acompanhou todo o meu trabalho de perto e contribuiu muito para o desenvolvimento do mesmo.

Agradeço o professor Lucas Monteiro Chaves por ter contribuído muito para minha formação acadêmica e por ter me incentivado muito a fazer o mestrado.

Agradeço aos amigos Fábio Dadam, Rodrigo, Tiago, Gustavo, Émille e Mylene, com quem convivi todo esse tempo de mestrado e passei momentos muito especiais.

Agradeço o querido Rogério, pois sem seu apoio teria sido impossível concluir este trabalho.

Meus mais sinceros agradecimentos vão para meus pais por sempre estarem presentes em minha vida e por serem pessoas tão amigas e compreensivas. Agradeço também a toda minha família por seu imensurável apoio e incentivo.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

CGAL[1] (*Computational Geometry Algorithms Library*) é uma biblioteca de estruturas de dados e algoritmos geométricos, escrita em C++ e vem sendo desenvolvida por um consórcio formado por sete instituições: Universidade de Utrech (Holanda), ETH Zurich (Suíça), RISC Linz (Áustria), Universidade de Tel Aviv (Israel), INRIA Sophia-Antipolis (França) e Instituto Max Planck Saarbrucken (Alemanha). Seu principal objetivo é proporcionar implementações de objetos e algoritmos geométricos básicos e, com isso, facilitar o desenvolvimento de aplicações geométricas mais complexas.

CGAL implementa todas as suas primitivas e algoritmos geométricos sobre a geometria Euclidiana, onde existe necessidade de se tratar muitos casos especiais, tornando os algoritmos muito longos e complexos. Na busca por soluções mais simples e sucintas, tais algoritmos poderiam ser desenvolvidos sobre a geometria projetiva *clássica* (denotada por $\mathbb{P}^2$ em duas dimensões), a qual apresenta vantagens em relação à geometria Euclidiana, como por exemplo:

- o uso de coordenadas homogêneas, que, geralmente, resulta em fórmulas mais simples envolvendo apenas operações básicas da álgebra linear (multiplicações de matrizes, produto escalar, produto vetorial etc). Além disso, permite a representação de pontos no infinito, os quais podem ser utilizados como sentinelas ou fazer parte da saída de algoritmos [Sto91];

- a existência de uma dualidade perfeita entre pontos e retas através de um mapeamento bijetor $*$, onde cada ponto p corresponde a uma reta p^*, cujos coeficientes são as coordenadas de p. Este mapeamento preserva a relação de incidência, em outras palavras, se um ponto p está numa reta l, então a reta p^* contém o ponto l^*. Esta é uma propriedade bastante útil, pois cada definição, teorema ou algoritmo

[1] CGAL é livre para uso acadêmico e tem licença específica para uso comercial.

definido para um conjunto de pontos também o está para o conjunto de retas duais e vice-versa [Sto91];

- a homogeneidade decorrente da unificação de conceitos, como é o caso da definição de curvas cônicas no plano projetivo, as quais são todas elipses [Pin98].

Entretanto, apesar destas vantagens, a geometria projetiva clássica traz algumas dificuldades: o plano projetivo $\mathbb{P}^2$ não é orientável, sendo, portanto, impossível definir a orientação de trios de pontos. Essa deficiência é muito grave, pois esta operação é necessária em muitos algoritmos geométricos. Além disso, muitos conceitos geométricos são ambíguos e, ademais, não se tem a noção de convexidade [Sto91].

A geometria projetiva orientada, GPO, veja [Sto91], por sua vez, possui todas as vantagens da geometria projetiva clássica e elimina as deficiências mencionadas anteriormente. Nesta geometria, tem-se tanto a noção de orientabilidade quanto de convexidade. É importante ressaltar que a geometria projetiva orientada engloba a geometria Euclidiana, ou seja, todos os conceitos e operações presentes nesta também estão definidos na primeira. Além disso, a geometria projetiva orientada introduz uma série de benefícios e facilidades para a computação geométrica.

Devido às vantagens proporcionadas pelas propriedades geométricas da GPO, os algoritmos desenvolvidos nessa geometria são mais simples e sucintos, ou seja, geralmente, não existe a necessidade de tratamento de exceções.

Diante desse quadro, esta dissertação descreve uma implementação da GPO em duas dimensões, ou seja, o plano projetivo orientado, PPO, e sua incorporação à biblioteca CGAL. A introdução do PPO nessa biblioteca traz muitas facilidades para o desenvolvimento de aplicações, pois problemas geométricos considerados no mesmo apresentam situações mais homogêneas e soluções muito menos complexas.

A meta deste trabalho foi estender alguns algoritmos presentes em CGAL para o PPO, reutilizando, sempre que possível, as implementações da biblioteca e comprovar a redução do número de casos especiais tratados. Como veremos mais adiante, alguns algoritmos são naturalmente propícios para o PPO, enquanto que algumas características intrínsecas do mesmo induz à existência de situações especiais em outros. Em suma, este trabalho apresenta a extensão de primitivas, predicados e algoritmos para o PPO e aborda a existência da necessidade de tratamento de casos especiais. Para exibir a saída dos algoritmos foi utilizado o $\mathbb{T}^2$Viewer, que é um visualizador dos modelos plano e esférico de $\mathbb{T}^2$ [SDto].

CGAL vem sendo desenvolvida utilizando-se programação genérica em C++ (mais detalhes podem ser encontrados no capítulo 2) e, para produzir uma implementação dentro dos padrões de CGAL, os algoritmos estendidos também foram implementados de forma genérica [GHJV95, BM00] .

1.1 Organização deste Trabalho

Além desta introdução, a dissertação é constituída por mais cinco capítulos:

- o capítulo 2 menciona algumas bibliotecas de algoritmos e estruturas de dados geométricos e ambientes para o desenvolvimento de algoritmos produzidos anteriormente e dá ênfase às funcionalidades e características de CGAL;
- o capítulo 3, por sua vez, introduz o PPO, alguns conceitos presentes no mesmo e seus dois modelos;
- a extensão das primitivas e predicados geométricos de CGAL ao PPO são discutidos no capítulo 4;
- as características dos algoritmos estendidos constituem o capítulo 5;
- e o sexto capítulo sintetiza as conclusões baseadas nos resultados apresentados nos dois capítulos anteriores.

Capítulo 2

CGAL e Trabalhos Anteriores

2.1 Descrição da Biblioteca CGAL

CGAL, como mencionado anteriormente, é uma biblioteca de algoritmos e estruturas de dados geométricos, escrita em C++ utilizando-se programação genérica [GHJV95, BM00], cujo principal objetivo é auxiliar pesquisadores e desenvolvedores na produção de aplicações geométricas mais complexas, através da reutilização de seus componentes.

CGAL foi desenvolvida por diferentes grupos de usuários e é usada para propósitos diversos. Existem pesquisadores trabalhando diretamente com geometria computacional e que utilizam a biblioteca para desenvolver e testar seus próprios algoritmos, assim como pesquisadores de outras áreas que usam recursos de geometria computacional nas suas aplicações. Há ainda o uso comercial de CGAL, no qual empresas utilizam a biblioteca para o desenvolvimento de aplicações.

Os grupos de usuários de CGAL são, portanto, bastante heterogêneos e, devido a isso, demandam diferentes recursos. Para suprir essa demanda, CGAL segue alguns princípios de *design*, dos quais os principais são: robustez, generalidade, eficiência e facilidade de uso. Embora isso não seja trivial, a biblioteca tenta combinar esses princípios.

Robustez

Em geometria computacional, robustez dos *softwares* é fundamental, pois se ocorre algum erro de arredondamento durante a execução do programa, o mesmo pode produzir respostas (conceitualmente) incorretas. Como os números são representados no computador por tipos de precisão limitada, erros de arredondamento freqüentemente acontecem. Para amenizar esse problema, CGAL oferece tipos de números especiais, com maior precisão. Se o processamento rápido é mais importante do que gerar sempre respostas corretas, é possível utilizar tipos básicos de C++, *floats* ou *doubles* (tipos com precisão limitada e cuja computação é mais rápida). Por outro

lado, se a produção de resultados exatos é essencial, pode-se utilizar os tipos especiais definidos na biblioteca. Esses tipos têm representação mais longa resultando em maior precisão, e até mesmo números inteiros com precisão ilimitada, obtidos através da alocação de listas de inteiros concatenados. Além disso, alguns tipos utilizam mais de um campo para representar um valor, como é o caso do tipo *quotient* que é constituído por um campo numerador e outro denominador. O uso do tipo *quotient* evita operações de divisão e, por conseqüência, muitos dos erros de arredondamento.

Generalidade

Como foi dito anteriormente, CGAL é utilizada para diversos fins. Para prover generalidade, CGAL vem sendo desenvolvida utilizando-se programação genérica em C++ [GHJV95, BM00]. Assim, os tipos de representação de números utilizados pelos algoritmos são escolhidos pelo usuário, de acordo com sua necessidade, e os algoritmos são implementados como funções *templates* em C++ para prover essa funcionalidade.

Além disso, o usuário opta também pela representação dos pontos, isto é, coordenadas cartesianas ou homogêneas (sendo $w \geq 0$ — na versão atual de CGAL).

Eficiência

Nem sempre um dado algoritmo é o mais eficiente para todas as aplicações e, além disso, sua eficiência pode ainda depender do tipo de representação de número que é usada. Nesses casos, CGAL oferece mais de um algoritmo para resolver os problemas e cabe ao usuário decidir qual é o mais apropriado para sua aplicação.

Facilidade de Uso

Os desenvolvedores de CGAL estão muito empenhados em prover uma biblioteca fácil de ser usada. Embora nem sempre isto seja simples, CGAL tenta combinar facilidade de uso e generalidade.

Em programação genérica, *templates* são utilizados para obter generalidade, e isso pode tornar o uso da biblioteca complicado para usuários novatos. Para amenizar essa dificuldade, é utilizado o mecanismo *typedefs* da linguagem C++ para evitar o contato direto do usuário com os *templates*. Os *typedefs* que são definidos em arquivos de cabeçalho, são incluídos na aplicação do usuário. Desta forma, muitas vezes o uso de *templates* se torna transparente para o usuário.

Além de tudo isso, para obter flexibilidade e eficiência, CGAL está divida em três camadas (figura 2.1).

1. o núcleo (à esquerda), onde estão implementados dois tipos de representação de números (coordenadas cartesianas e coordenadas homogêneas sem sinal) e alguns tipos especiais de números com maior precisão em relação aos tipos básicos de C++. O núcleo contém também objetos geométricos básicos (ex.: pontos, retas, semi-retas, círculos, etc.) e alguns predicados (ex.: teste de orientação de trios de pontos, teste de intersecção entre primitivas, etc.) que são usados pelos algoritmos da biblioteca básica. CGAL provê a implementação do núcleo 2-dimensional, do 3-dimensional e do d-dimensional.

2. a biblioteca básica (ao centro), onde estão implementados algoritmos geométricos básicos e algumas estruturas de dados, como:

   - algoritmos para determinar a envoltória convexa de conjuntos de pontos;
   - operações com polígonos: cálculo da área sinalada de um polígono, determinação de sua orientação ou verificação de sua simplicidade ou convexidade, etc;
   - algoritmos para partição de polígonos;
   - triangulação 2D, etc.

3. e, por fim (à direita), a parte de suporte da biblioteca responsável por I/O, visualização, fornecimento de outros tipos de números, etc.

Essas três partes da biblioteca interagem, ou seja, para qualquer algoritmo da biblioteca básica, o núcleo provê os predicados necessários à sua computação e, se há necessidade de visualização dos resultados na tela, a biblioteca de suporte disponibiliza um visualizador.

Note que, com essa separação entre algoritmos, predicados e objetos geométricos básicos e recursos de suporte, a biblioteca se torna mais flexível, pois um mesmo algoritmo pode usar objetos geométricos, predicados e recursos de suporte distintos. Como dito anteriormente, fica a critério do usuário decidir o que é mais apropriado (eficiente) para sua aplicação.

Ainda assim, se alguns predicados e/ou algoritmos geométricos do núcleo não satisfazem as necessidades de uma aplicação, o usuário pode implementar seus próprios predicados e objetos geométricos. Em CGAL, existem as chamadas *traits classes*, onde estão implementados predicados e objetos geométricos básicos. Todo algoritmo da biblioteca básica é parametrizado 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 requerimentos definidos por tais algoritmos. A utilização de programação de programação genérica proporciona aos usuários de CGAL

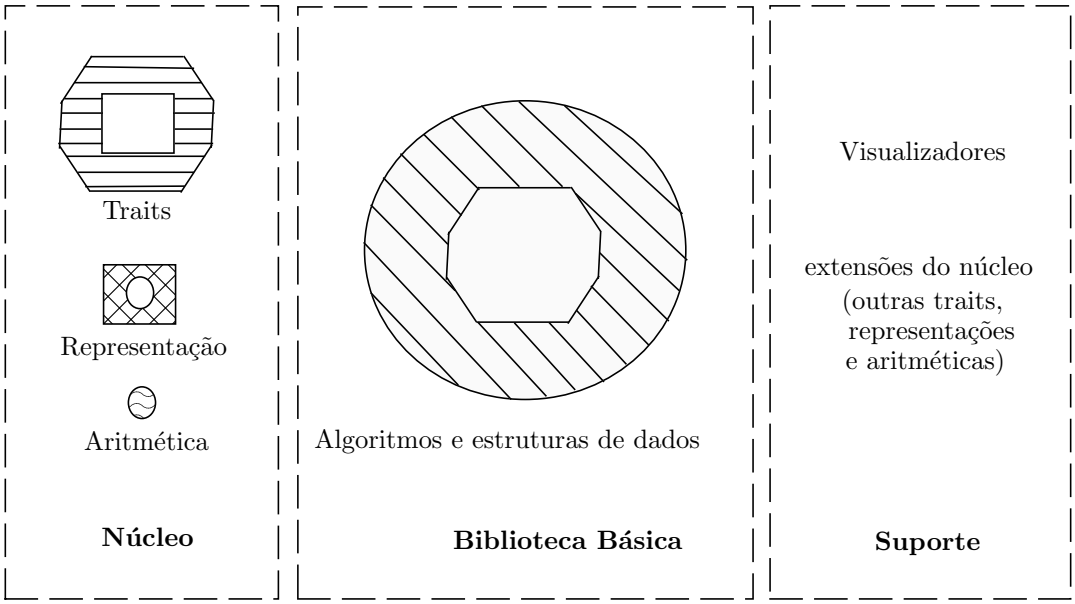


Figura 2.1: Camadas do *design* de CGAL

uma flexibilidade notável e torna o uso de CGAL bastante atraente. Na versão atual de CGAL (versão 3.0.1), assim como alguns tipos de aritmética (tipos de números e suas operações), todos os tipos de representação de números estão implementados no núcleo, enquanto outros são pacotes oferecidos pela biblioteca de suporte. Sendo assim, é possível estender os tipos de números proporcionados por CGAL, implementando pacotes que podem ser inseridos na camada de suporte da biblioteca.

2.2 Trabalhos Anteriores

Nesta seção, serão apresentados alguns ambientes e bibliotecas existentes para auxílio no desenvolvimento de aplicações geométricas:

GeoLab [dRJ93]: é um ambiente para implementação, testes e animação de algoritmos geométricos no plano Euclidiano que permite a inclusão de novos algoritmos e objetos geométricos externos. Foi implementado em C++ e roda sobre a plataforma SUN/OS usando a biblioteca gráfica XView, e é de uso acadêmico gratuito.

LEDA [LED]: é uma biblioteca de algoritmos e estruturas de dados, escrita em C++, sendo que uma pequena parte é constituída por algoritmos geométricos no plano

Euclidiano. Até algum tempo atrás, LEDA era livre para uso acadêmico, mas hoje, tanto para uso acadêmico como comercial, é necessária a aquisição de licença.

XYZ GeoBench [XYZ]: é um ambiente para programação geométrica, desenvolvido em *Object Pascal* para *Macintosh*. O mesmo é composto por ferramentas de criação, edição e manipulação de objetos geométricos. Uma característica do XYZ GeoBench é a ênfase na robustez da implementação dos algoritmos.

Computation Geometry WorkBench [KMM+90]: é um ambiente para criação, edição e manipulação de objetos geométricos e provê animação e ferramentas para depuração de algoritmos. Contém essencialmente algoritmos em 2D e foi implementado em *SmallTalk*.

GeoPrO [dRG96]: é um ambiente distribuído para implementação, testes e animação de algoritmos geométricos na geometria projetiva orientada que permite a inclusão de novos algoritmos e visualizadores os quais executam de modo distribuído e se comunicam com o núcleo do ambiente por passagem de mensagens via rede. Foi implementado em C++, mas roda exclusivamente na plataforma IRIX de estações gráficas da Silicon Graphics.

GeomView [Geoa]: é um visualizador 3D que pode ser usado como um visualizador *standalone* para objetos estáticos ou pode ser usado para exibir objetos geométricos produzidos dinamicamente por outros programas. GeomView exibe objetos descritos em vários formatos de arquivos diferentes. Já existem vários objetos implementados no GeomView, mas o usuário pode incluir novos objetos ao visualizador. Este *software* é livre e disponibilizado segundo os termos da licença GPL, e roda em plataformas Unix.

GeoSheet [Geob]: é um visualizador de objetos geométricos 2D, escrito em C++, baseado no *software* `Xfig`. Embora seja um visualizador 2D, permite facilmente que sua saída seja exportada para outros visualizadores 2D ou 3D. O GeoSheet proporciona uma interface interativa para entrada e saída, e além disso, permite que a saída seja exibida na máquina local ou em máquinas remotas, conforme especificado pelo usuário.

Qt_widget [sup02]: é um visualizador escrito em C++, portável e com código aberto, que permite interação com objetos 2D de CGAL. Ele é baseado em camadas, as quais podem ser ativadas ou desativadas, sendo que é exibido na tela o resultado da intersecção das camadas ativas.

Capítulo 3

O Plano Projetivo Orientado

Neste capítulo, mencionaremos dois espaços projetivos clássicos e faremos uma breve descrição de dois espaços projetivos orientados, em uma e duas dimensões, visando rever conceitos fundamentais. O leitor não familiarizado com estes conceitos pode referenciar [dRS94] e [Sto91] para um estudo mais aprofundado.

3.1 A Reta Projetiva Clássica $\mathbb{P}^1$ e a Reta Projetiva Orientada $\mathbb{T}^1$

A reta projetiva clássica, denotada por $\mathbb{P}^1$, pode ser modelada por uma cópia de $\mathbb{R}$ e um ponto no infinito [Sto91]. Em $\mathbb{P}^1$, um único ponto no infinito representa duas direções distintas e, portanto, não podemos diferenciá-las.

Já a reta projetiva orientada, denotada por $\mathbb{T}^1$, pode ser modelada por duas cópias da reta real $\mathbb{R}$ e mais dois pontos indicando as direções $+d_\infty$ e $-d_\infty$. Podemos visualizar $\mathbb{T}^1$ como uma circunferência unitária (figura 3.1) ou como duas cópias independentes da reta $\mathbb{R}$ estendida (figura 3.2).

Os pontos em $\mathbb{T}^1$ são representados por coordenadas homogêneas, ou seja, por pares $[w, x]$ (exceto o par $[0, 0]$, considerado inválido). Existe uma correspondência parcial entre $\mathbb{T}^1$ e a reta real $\mathbb{R}$, onde todo ponto com $w \neq 0$ é mapeado no ponto $\frac{x}{w}$ em $\mathbb{R}$. Já os pontos em $\mathbb{T}^1$ com coordenada $w = 0$ são pontos no infinito. Os pontos em $\mathbb{T}^1$ com coordenadas $[w, x]$ e $[\alpha w, \alpha x]$, onde $\alpha \in \mathbb{R}$ e $\alpha > 0$, são pontos iguais. Denomina-se aquém ao conjunto dos pontos de $\mathbb{T}^1$ com $w > 0$ e além ao conjunto dos pontos com $w < 0$.

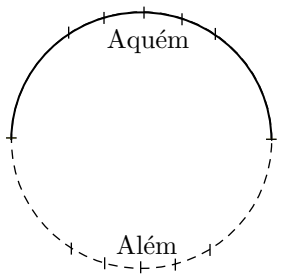


Figura 3.1: Circunferência unitária representando $\mathbb{T}^1$

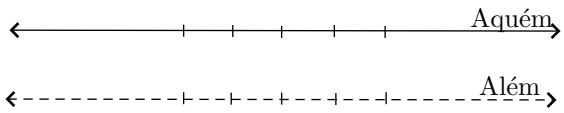


Figura 3.2: Duas cópias da reta real $\mathbb{R}$ representando $\mathbb{T}^1$ (exceto $+d_\infty$ e $-d_\infty$).

3.2 O Plano Projetivo Clássico $\mathbb{P}^2$ e o Plano Projetivo Orientado $\mathbb{T}^2$

O plano projetivo clássico, denotado por $\mathbb{P}^2$, é uma extensão do plano Euclidiano, onde para cada feixe de retas paralelas, temos um ponto no infinito [Sto91].

Em $\mathbb{P}^2$, não é possível determinar orientação de trios de pontos e o conceito de convexidade não está bem definido [Sto91]. Uma vez que, nos algoritmos estendidos, necessitamos extensivamente dessa operação e do conceito de convexidade, optamos por estender os algoritmos para o plano projetivo orientado, no qual todas as operações e conceitos Euclidianos estão definidos.

A geometria projetiva orientada em duas dimensões que é denominada *plano projetivo orientado*, PPO, denotado por $\mathbb{T}^2$, é um espaço geométrico definido como uma extensão estrita do Euclidiano [dRS94], seus pontos são representados por coordenadas homogêneas sinaladas e consistem de todas as triplas $[w, x, y]$, onde w, x, $y \in \mathbb{R}$ (exceto a tripla $[0, 0, 0]$ que é considerada uma tripla inválida).

Existe uma correspondência parcial entre coordenadas homogêneas e coordenadas cartesianas: um ponto de $\mathbb{T}^2$ representado pelas coordenadas $[w, x, y]$, com $w \neq 0$, possui coordenadas cartesianas $(x/w, y/w)$.

O PPO pode ser visualizado através dos modelos plano (figura 3.3) e esférico (figura 3.4).

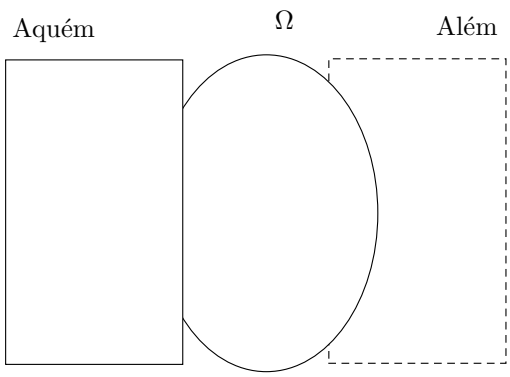


Figura 3.3: Modelo plano de $\mathbb{T}^2$

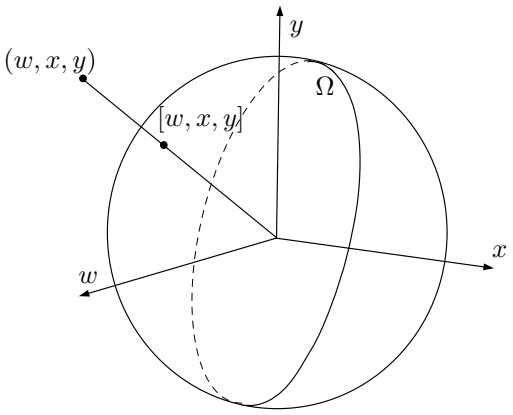


Figura 3.4: Modelo esférico de $\mathbb{T}^2$

O modelo esférico consiste da superfície da esfera unitária S^2 de $\mathbb{R}^3$ com centro na origem. Neste modelo, os pontos de $\mathbb{T}^2$ são obtidos a partir dos pontos (w, x, y) de $\mathbb{R}^3$ por projeção central (figura 3.4):

$$[w, x, y] = (w, x, y)/\sqrt{w^2 + x^2 + y^2}.$$

A circunferência de raio máximo, onde a coordenada $w = 0$, representa a reta do infinito Ω e divide a esfera em dois hemisférios: um onde os pontos possuem coordenada $w > 0$ (aquém) e o outro onde a coordenada $w < 0$ (além). Os hemisférios constituídos pelo aquém e metade de Ω e pelo além e a outra metade de Ω são denominados hemisférios canônicos de $\mathbb{T}^2$.

O PPO pode ainda ser realizado pela composição de duas cópias do plano Euclidiano $\mathbb{R}^2$ e por uma cópia do círculo unitário S^1 (figura 3.3). A cópia de $\mathbb{R}^2$ onde a coordenada w é positiva é chamada de aquém, os pontos com coordenada $w = 0$ constituem a reta do infinito, que é representada por Ω e a outra cópia de $\mathbb{R}^2$ com coordenada w negativa é denominada além. Esta realização corresponde ao que é chamado de modelo plano.

Em $\mathbb{T}^2$, o ponto $p = [w, x, y]$ é distinto do ponto $\neg p = [-w, -x, -y]$ chamado de antípoda de p. No modelo plano, estes pontos são coincidentes (embora não iguais), sendo que um deles está no aquém e o outro no além. Já no modelo esférico, estes pontos são diametralmente opostos.

Geralmente, pontos pertencentes a Ω são chamados de pontos impróprios enquanto que os demais pontos são chamados de pontos próprios.

Note também que os pontos de $\mathbb{T}^2$ com coordenadas $[w, x, y]$ e $[\alpha w, \alpha x, \alpha y]$, onde $\alpha \in \mathbb{R}$ e $\alpha > 0$, são pontos iguais [dRS94, Sto91].

3.3 Orientação de três pontos

Sejam p, q e r três pontos não colineares. Define-se, informalmente, a orientação da tripla (p, q, r) como sendo o sentido (horário ou anti-horário) que o segmento $\overline{pu}$ rotaciona em torno de p, quando u segue de q em direção a r, ao longo do segmento $\overline{qr}$ (figura 3.5) [dRS94, Sto91]. Algebricamente, a orientação está definida na seção 4.3.4.

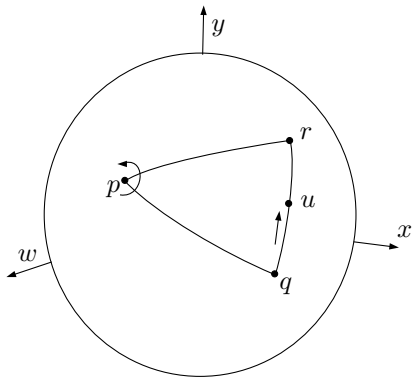


Figura 3.5: Definição de orientação

3.4 Retas e Segmentos

Uma reta em $\mathbb{T}^2$, definida por três coeficientes e denotada por $<W, X, Y>$, é o lugar geométrico dos pontos $p = [w, x, y]$ de $\mathbb{T}^2$, tais que $Ww + Xx + Yy = 0$. Note que, se negarmos as três coordenadas de p, este continua pertencendo à reta. Logo, um ponto p está em uma reta se, e somente se, $\neg p$ também está. A equação da reta define um plano em $\mathbb{R}^3$ e este corta a esfera S^2 em um círculo de raio máximo. Portanto, toda reta em $\mathbb{T}^2$, no modelo esférico (figura 3.6), corresponde a um círculo de raio máximo em S^2. Já no modelo plano, uma reta é representada por duas retas Euclidianas superpostas, uma no aquém e outra no além e mais dois pontos no infinito correspondentes às duas direções paralelas às retas.

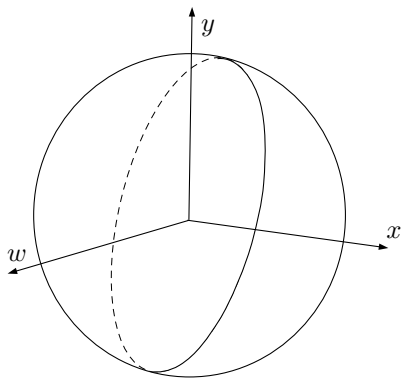


Figura 3.6: Reta no modelo esférico de $\mathbb{T}^2$

Dados dois pontos distintos p_0 e p_1 não antipodais, define-se, no modelo esférico, o segmento $\overline{p_0p_1}$ (figura 3.7) como sendo o menor arco do círculo máximo que passa por p_0 e p_1. Já no modelo plano é preciso considerar algumas situações:

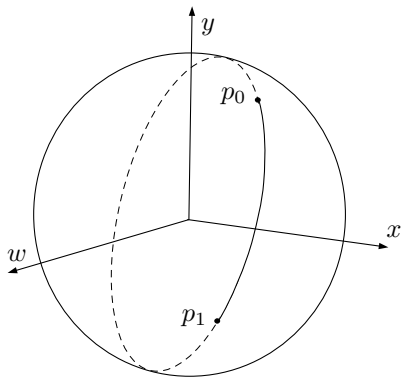


Figura 3.7: Segmento no modelo esférico de $\mathbb{T}^2$

1. se p_0 e p_1 estão no aquém, tem-se o segmento Euclidiano no aquém ligando p_0 a p_1 (figura 3.8);

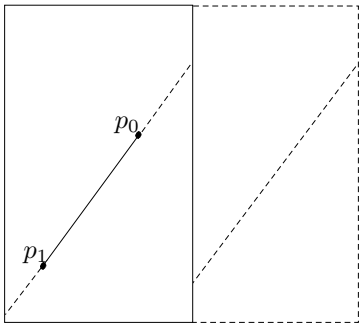


Figura 3.8: Segmento no aquém de $\mathbb{T}^2$

2. da mesma forma, se p_0 e p_1 estão ambos no além, tem-se o segmento Euclidiano unindo p_0 a p_1(figura 3.9);

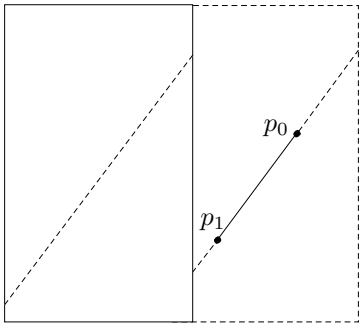


Figura 3.9: Segmento no além de $\mathbb{T}^2$

3. se um dos extremos está no aquém (suponha p_0) e o outro está no além (p_1) tem-se a situação em que o segmento consiste de duas semi-retas: uma no aquém, saindo de p_0 na direção oposta a $\neg p_1$ e outra no além saindo de p_1 na direção oposta a $\neg p_0$ (figura 3.10).

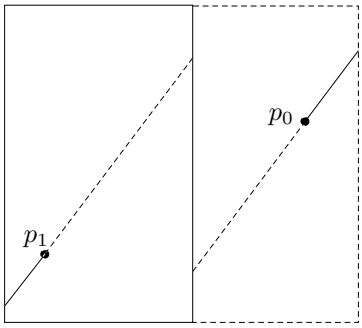


Figura 3.10: Segmento do aquém ao além de $\mathbb{T}^2$

Note que existem infinitas retas (e conseqüentemente infinitos possíveis segmentos de reta) que passam por pontos antipodais. Por esta razão, não se define *o* segmento entre tais pontos.

3.4.1 Os dois lados de uma reta

Os pontos não pertencentes a uma reta $r = <W, X, Y>$, podem ser divididos em dois conjuntos, denominados lados da reta, conforme o sinal da expressão $Ww + Xx + Zz$ (vide seções 4.3.4 e 4.3.6). Este teste define o lado positivo e o lado negativo da reta [dRS94].

No modelo esférico, r divide a esfera S^2 em dois hemisférios, os quais constituem seu lado positivo e seu lado negativo (figura 3.12).

No modelo plano, por sua vez, o lado positivo de uma reta r consiste de um semi-plano do aquém limitado por r e do outro semi-plano do além, que não é antípoda do primeiro (figura 3.11). O lado positivo de r inclui também todos os pontos no infinito pertencentes à meia-reta limitada pelas duas direções paralelas a r. O dois outros semi-planos adicionados à meia-reta complementar do infinito constituem o lado negativo de r.

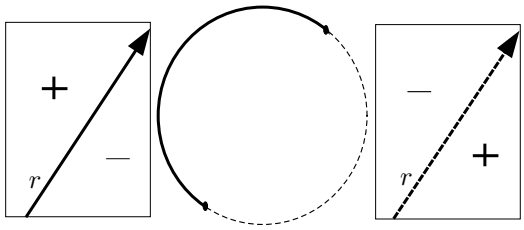


Figura 3.11: Os lados de uma reta no modelo plano de $\mathbb{T}^2$

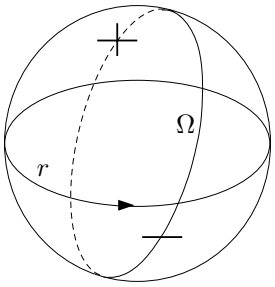


Figura 3.12: Os lados de uma reta no modelo esférico de $\mathbb{T}^2$

Notações

Nesta seção, definimos algumas notações relativas a subconjuntos de $\mathbb{T}^1$ e a subconjuntos de $\mathbb{T}^2$ que serão usadas nos capítulos posteriores. Na seção 4.3.1, utilizamos o lado não-negativo de $\mathbb{T}^1$ na definição da função distância e o denotamos por: $\mathbb{T}^1_{x\geq 0} = \{[w,x] \in \mathbb{T}^1 \mid x \geq 0\}$. Utilizamos também nessa seção o subconjunto de $\mathbb{T}^2$ constituído por pontos próprios, denotado por: $\mathbb{T}^2_* = \mathbb{T}^2 - \Omega$. Na seção 4.1.7, na definição da representação de retângulos isotéticos, utilizamos o subconjunto de $\mathbb{T}^1$, formado pelos pelos pontos no aquém e pelos dois pontos no infinito, denotado por $\mathbb{T}^1_{w\geq 0} = \{[w,x] \in \mathbb{T}^1 \mid w \geq 0\}$.

Capítulo 4

Extensão das Primitivas Geométricas e Predicados

O conjunto de primitivas geométricas de CGAL estendido para o PPO é constituído basicamente por pontos, retas, segmentos, semi-retas, vetores, direções, triângulos, retângulos isotéticos, circunferências e caixas limitantes (*bounding boxes*). Foram estendidas também transformações afins, como rotação, translação e escala, que operam sobre as primitivas geométricas mencionadas. Já a coleção de predicados estendidos abrange testes de intersecção entre as primitivas básicas, localização de pontos em relação a polígonos simples, testes de orientação de trios de pontos, testes envolvendo comparação lexicográfica e ordenação de pontos colineares, comparação de distâncias relativas entre pontos próprios e impróprios, etc.

Neste capítulo, discutiremos a extensão de cada primitiva mencionada, abordando o tipo de representação usada, as características especiais apresentadas no PPO e sua visualização nos modelos plano e esférico. Da mesma forma, será analisada a extensão dos principais predicados, destacando-se a existência ou não da necessidade de tratamento de casos especiais.

4.1 Primitivas Geométricas

Como mencionado anteriormente, CGAL possui implementações de primitivas geométricas em $\mathbb{R}^2$ e $\mathbb{R}^3$. Na extensão da maioria das primitivas foi reutilizado o código que implementa primitivas em $\mathbb{R}^3$ existente em CGAL, através de agregação, ou seja, foi definido na implementação de cada primitiva em $\mathbb{T}^2$ um apontador para o objeto que define a primitiva em $3D$ correspondente, sendo este apontador interno à classe das primitivas em $\mathbb{T}^2$ e, portanto, transparente para o usuário.

4.1.1 Ponto

Um ponto (próprio ou imprório, vide seção 3.2) em $\mathbb{T}^2$ assim como um ponto em $\mathbb{R}^3$ é definido por três coordenadas, sendo assim, através de um mapeamento de coordenadas foi possível reutilizar o código que implementa ponto em $\mathbb{R}^3$ para implementar ponto em $\mathbb{T}^2$. Na classe que define ponto em $\mathbb{T}^2$, foi declarado e alocado um apontador para um objeto do tipo ponto $3D$ (ponto em $\mathbb{R}^3$), de tal forma que sua coordenada x foi mapeada para a coordenada w do ponto $\mathbb{T}^2$, suas coordenadas y e z para as coordenadas x e y do ponto em $\mathbb{T}^2$, respectivamente. Note que o ponto $3D$ é visível apenas dentro da classe do ponto em $\mathbb{T}^2$ e a reutilização de código fica transparente para o usuário.

Optamos por mapear a primeira coordenada do ponto $3D$, em vez da terceira, para a coordenada homogênea do ponto em $\mathbb{T}^2$ com o objetivo de facilitar uma possível extensão futura de ponto *n-dimensional* para ponto em $\mathbb{T}^{n-1}$, uma vez que não seria necessário deslocar para a direita a coordenada homogênea.

4.1.2 Reta

Em $\mathbb{R}^3$, uma reta pode ser caracterizada por dois pontos ou por um ponto e uma direção, e esta última constitui a representação presente em CGAL. Em $\mathbb{T}^2$, quase todas as retas podem ser representadas por um ponto e uma direção, exceto a reta do infinito, a qual não possui uma direção definida. Desta forma, optamos por não reutilizar a implementação de reta $3D$ presente no núcleo de CGAL e implementar as retas em $\mathbb{T}^2$ com uma representação distinta, a qual consiste de dois pontos em $\mathbb{T}^2$, já que esta cobre todos os casos.

4.1.3 Segmento

Um segmento em $\mathbb{T}^2$ é definido por dois pontos não-antipodais (vide seção 3.2). Sua representação em $\mathbb{R}^3$ é compatível com sua representação em $\mathbb{T}^2$, a qual consiste em dois pontos denotando origem e destino. Sendo assim, foi possível a reutilização da implementação do segmento $3D$, de forma similar à implementação do ponto $3D$, através de agregação, ou seja, foi declarado e alocado um ponteiro para um segmento $3D$, sendo que o seu atributo *origem* (ponto $3D$) foi mapeado para o atributo origem (ponto em $\mathbb{T}^2$) do segmento em $\mathbb{T}^2$, o mesmo mapeamento ocorreu para os atributos *destinos* de ambos os segmentos.

4.1.4 Semi-Retas

Semi-retas em $\mathbb{T}^2$ são casos particulares de segmentos com um dos extremos no infinito. Sendo assim, as semi-retas em $\mathbb{T}^2$ foram implementadas através de herança da classe que

implementa o segmento em $\mathbb{T}^2$.

4.1.5 Vetores

Vetores são segmentos orientados e, portanto, para representá-los são necessários dois pontos. Na implementação presente em CGAL, assume-se que um destes pontos é a origem do espaço Euclidiano. Para manter a compatibilidade e a coerência, a representação de vetores em $\mathbb{T}^2$ consiste nas coordenadas do ponto destino enquanto que sua origem é a origem do aquém de $\mathbb{T}^2$. Vale ressaltar que *o* vetor, assim como *o* segmento com extremos antipodais, não está definido.

4.1.6 Triângulos

Para implementação de triângulo em $\mathbb{T}^2$ (figura 4.1), reutilizou-se o código que implementa triângulo em $3D$, substituindo-se as coordenadas cartesianas dos três pontos que representam este último por coordenadas homogêneas sinaladas. Sejam a, b e c três pontos em $\mathbb{T}^2$. Note que, para que eles sejam vértices de um triângulo é necessário inicialmente que cada um de seus lados esteja definido, logo, a, b e c não podem ser antipodais dois a dois. Ademais, como o triângulo é uma figura convexa, outra pré-condição para sua existência é que o antípoda de um dos vértices não esteja no segmento que une os outros dois ($\neg a \notin \overline{bc}$, $\neg b \notin \overline{ac}$ e $\neg c \notin \overline{ab}$).

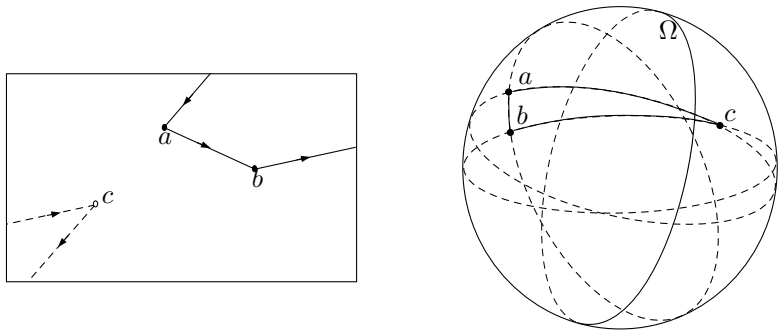


Figura 4.1: Triângulo no modelo plano (à esquerda) e no modelo esférico (à direita)

4.1.7 Retângulos Isotéticos

Retângulos isotéticos são retângulos com lados paralelos aos eixos X e Y. Seja r um retângulo isotético no aquém de $\mathbb{T}^2$ (figura 4.2) e sejam a, b, c e d seus extremos ($ab \parallel dc$ e $ad \parallel bc$). Note que, se deslocarmos b ao logo da reta $\overrightarrow{ab}$, seguindo sua orientação, e simultaneamente deslocarmos c ao longo da reta $\overrightarrow{dc}$ de acordo com sua orientação, quando "chegarem ao infinito", b e c serão pontos iguais (figura 4.3), logo, um dos lados de r se degenera para um ponto. Se não pararmos o deslocamento no infinito, os segmentos $\overline{ab}$ e $\overline{dc}$ se cruzam no infinito (figura 4.4) e, portanto, r deixa de ser um polígono simples e conseqüentemente um retângulo. Com isso, conclui-se que retângulos com extremos em lados distintos de $\mathbb{T}^2$ não estão definidos.

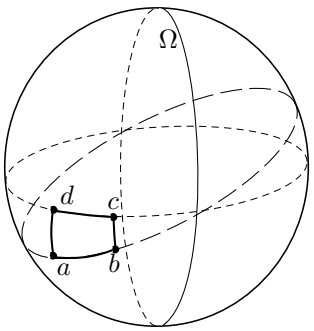


Figura 4.2: Retângulo definido

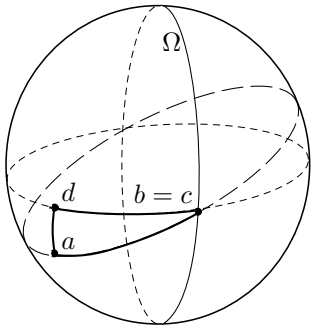


Figura 4.3: Retângulo degenerado

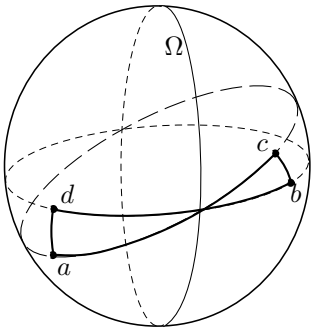


Figura 4.4: Retângulo indefinido

No espaço Euclidiano, retângulos isotéticos são definidos por dois pontos a e b, a partir dos quais determinam-se as coordenadas dos outros dois. Em $\mathbb{T}^2$, esta representação não é adequada pois, quando b é impróprio, não é possível definir o terceiro vértice, ou seja, quando b representa a direção horizontal (vertical), o terceiro ponto possui a mesma abcissa (ordenada) que a, mas não é possível determinar sua ordenada (abcissa). Uma representação consistente para um retângulo isotético é um ponto a e dois valores Δ_x e Δ_y, os quais combinados com a resultam nos demais pontos. O domínio dos valores de Δ_x e Δ_y é o conjunto $\mathbb{R} \cup \{-\infty, +\infty\}$ e portanto para representá-los utilizamos dois pontos em $\mathbb{T}^1_{w \geq 0}$. Para finalizar, incluímos na representação mais um parâmetro para indicar a orientação. Com essa representação, a partir de $a = [w_a, x_a, y_a]$, $\Delta_x = [w_{\Delta_x}, x_{\Delta_x}]$ e $\Delta_y = [w_{\Delta_y}, y_{\Delta_y}]$, obtemos, os outros extremos b, c e d (figuras 4.2, 4.3 e 4.4) através de operações algébricas simples. Se $w_a \neq 0$, $w_{\Delta_x} \neq 0$ e $w_{\Delta_y} \neq 0$, as coordenadas cartesianas de $a = (\frac{x_a}{w_a}, \frac{y_a}{w_a})$, $\Delta_x = (\frac{x_{\Delta_x}}{w_{\Delta_x}})$, $\Delta_y = (\frac{x_{\Delta_y}}{w_{\Delta_y}})$, $b = (X_b, Y_b)$, $c = (X_c, Y_c)$ e $d = (X_d, Y_d)$. Desta forma, podemos dizer que:

$$X_b = \left(\frac{x_a}{w_a} + \frac{x_{\Delta_x}}{w_{\Delta_x}}\right), \quad Y_b = \left(\frac{y_a}{w_a}\right)$$

$$X_c = \left(\frac{x_a}{w_a} + \frac{x_{\Delta_x}}{w_{\Delta_x}}\right), \quad Y_c = \left(\frac{y_a}{w_a} + \frac{x_{\Delta_y}}{w_{\Delta_y}}\right)$$

$$X_d = \left(\frac{x_a}{w_a}\right), \quad Y_d = \left(\frac{x_a}{w_a} + \frac{x_{\Delta_y}}{w_{\Delta_y}}\right)$$

Logo, em termos de coordenadas homogêneas, temos: $b = [w_a w_{\Delta_x}, x_a w_{\Delta_x} + x_{\Delta_x} w_a, y_a w_{\Delta_x}]$, $c = [w_a w_{\Delta_x} w_{\Delta_y}, x_a w_{\Delta_x} w_{\Delta_y} + x_{\Delta_x} w_a w_{\Delta_y}, y_a w_{\Delta_x} w_{\Delta_y} + x_{\Delta_y} w_a w_{\Delta_x}]$ e $d = [w_a w_{\Delta_y}, x_a w_{\Delta_y}, y_a w_{\Delta_y} + x_{\Delta_y} w_a]$.

Se Δ_x e Δ_x são infinitos, c, calculado como descrito no parágrafo anterior, será a tripla inválida $[0, 0, 0]$. Portanto, para eliminar esse problema e garantir consistência, somente neste caso, as coordenadas de c se tornam $[0, x_a, y_a]$

4.1.8 Caixas Limitantes

Caixas limitantes (*bounding boxes*) são retângulos isotéticos não orientados e referem-se a retângulos que confinam objetos geométricos em seu interior. Em CGAL, está implementada a caixa limitante mínima, ou seja, o menor retângulo que limita uma figura. É importante ressaltar que a caixa limitante de objetos ilimitados com partes nos dois lados de $\mathbb{T}^2$ não está definida, uma vez que dois de seus lados se interceptariam no infinito. A representação de caixas limitantes é bastante semelhante à de retângulos isotéticos, exceto pelo atributo que define a orientação.

A operação de união de duas caixas limitantes presentes em lados distintos de $\mathbb{T}^2$ não está definida, uma vez que a caixa limitante resultante não seria um polígono simples. Já a união entre duas caixas limitantes infinitas apresentam algumas situações especiais, onde algumas das caixas limitantes resultantes não são figuras convexas como ilustra a figura 4.5 (Nesta figura, a caixa resultante da união das caixas b_1 e b_2 não está definida). Sejam duas caixas limitantes b_1 e b_2, se $b_1.\Delta_x$ e/ou $b_2.\Delta_x$ são infinitos, a caixa limitante resultante da união existe somente se $b_1.\Delta_x \neq \neg b_2.\Delta_x$. De forma semelhante, se $b_1.\Delta_y$ e/ou $b_2.\Delta_y$ são infinitos, a caixa limitante resultante da união existe somente se $b_1.\Delta_y \neq \neg b_2.\Delta_y$.

Em contrapartida, a operação de intersecção está sempre definida para qualquer instância, inclusive para caixas limitantes em lados distintos, as quais se encontram no infinito se ambas forem infinitas na mesma direção (figura 4.6), caso contrário, a intersecção é vazia.

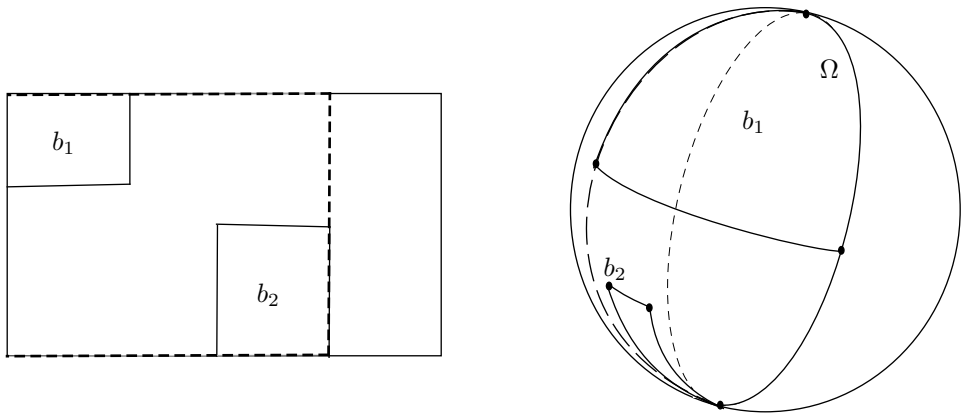


Figura 4.5: Caso onde a caixa limitante resultante da união é inválida

4.1.9 Circunferências

Em $\mathbb{R}^2$, uma circunferência pode ser definida por um ponto (centro), por um número real (raio) e um *bit* indicando a orientação. Em CGAL, circunferências podem ser instanciadas a partir de um ponto e do raio, de dois pontos (que determinam o diâmetro) e de três pontos, sendo que a partir desses três formatos de dados de entrada é possível obter-se o centro e raio que constituem a representação utilizada para implementação da circunferência em $\mathbb{R}^2$.

Circunferências em $\mathbb{T}^2$, por sua vez, podem ter raio finito, infinito ou "além do infinito" o que torna impossível armazená-lo em um número real. Quando o raio da circunferência é finito, temos uma circunferência semelhante à ilustrada nas figuras 4.7 e 4.8. Quando o raio é um valor "além do infinito", temos uma circunferência similar às das figuras 4.9 e 4.10. Uma circunferência possui raio infinito em duas situações: quando seu centro está no infinito ou quando sua fronteira coincide com Ω, sendo que, neste caso, seu centro está em um dois hemisférios canônicos (figura 4.11).

No caso de raio infinito, a representação da circunferência por um ponto (centro) e pelo raio é inadequada, pois seria impossível diferenciar os círculos da figura 4.11. Além disso, seria impossível determinar exatamente a circunferência quando o centro estiver no infinito.

Uma solução homogênea é armazenar o centro, um ponto da fronteira da circunferência e um bit para indicar a orientação, pois, dessa forma, a circunferência pode ser sempre determinada.

A seguir, vamos examinar alguns casos, onde a circunferência determinada pelo diâmetro (dois pontos) ou por três pontos nem sempre estão bem definidas.

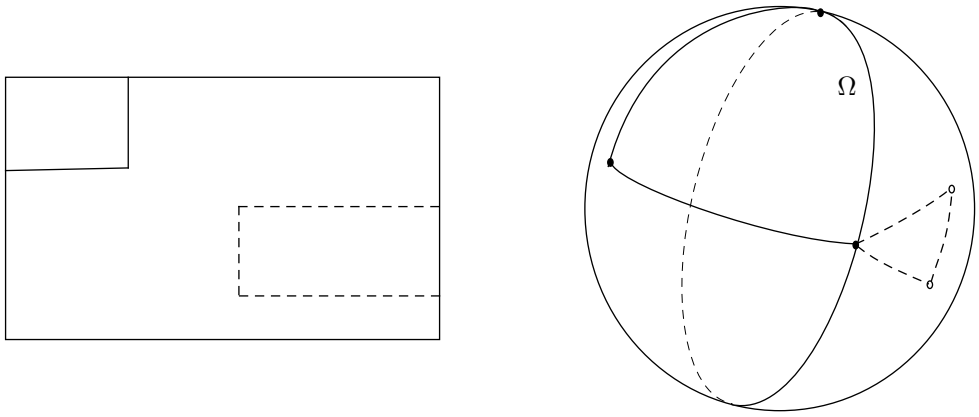


Figura 4.6: Intersecção de caixas limitantes, neste exemplo ambas são infinitas na direção $[0, -1, 0]$: à esquerda, no modelo plano e à direita, no modelo esférico.

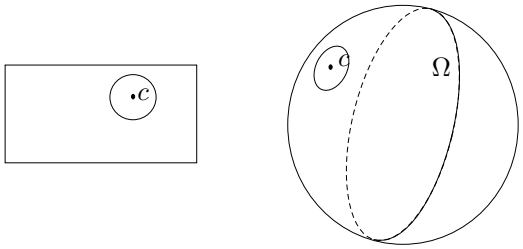


Figura 4.7: Circunferências de raio finito no aquém

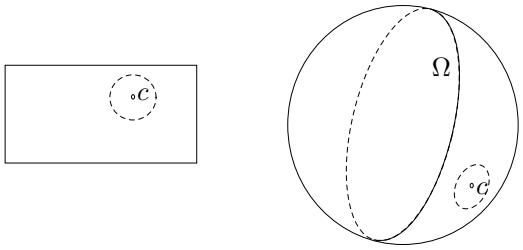


Figura 4.8: Circunferências de raio finito no além

Circunferências em $\mathbb{T}^2$, determinadas pelo diâmetro dado por dois pontos próprios p e q em hemisférios canônicos distintos de $\mathbb{T}^2$, estão definidas somente quando p e q possuem mesma distância relativa (vide seção 4.3.1) ao ponto $c = \overline{pq} \cap \Omega$. De forma similar, uma circunferência determinada por três pontos próprios p, q e r, em hemisférios canônicos distintos de $\mathbb{T}^2$, não está definida quando, relativamente, as distâncias de p, q e r ao ponto c são distintas, onde c é a interseção dos bisetores de p e q e de q e r, supondo que p e q estejam no mesmo hemisfério canônico.

Circunferências determinadas pelo diâmetro dado por dois pontos p e q quando pelo menos um deles está sobre Ω, não estão definidas, uma vez que o ponto equidistante de p e q não está bem definido. De forma similar, circunferências determinadas por três pontos, quando um ou dois deles são impróprios, não estão bem definidas.

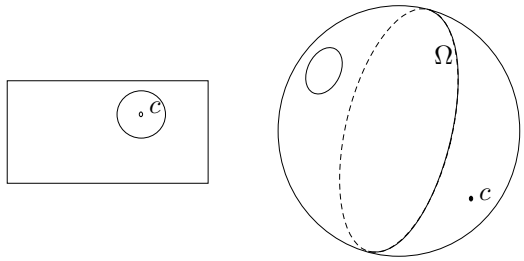


Figura 4.9: Circunferências de raio "além do infinito" no aquém

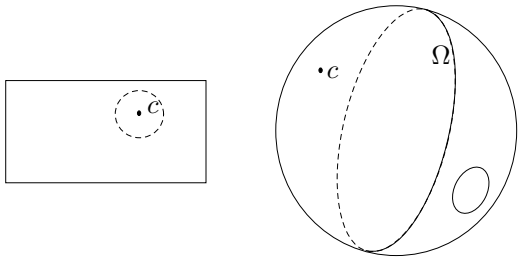


Figura 4.10: Circunferências de raio "além do infinito" no além

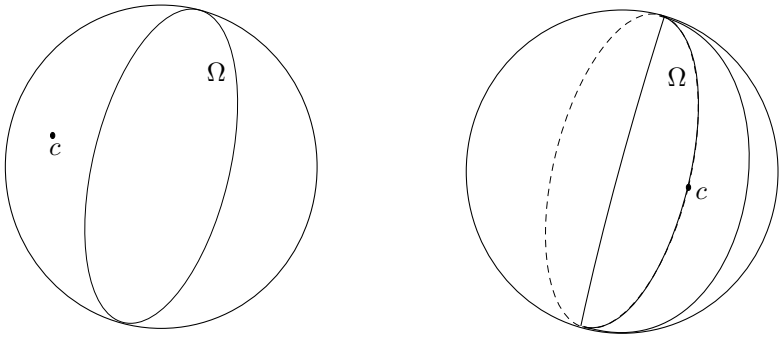


Figura 4.11: Circunferências de raio infinito

4.1.10 Direções

Direções em $\mathbb{T}^2$ foram implementadas a partir de direções em $\mathbb{R}^3$, através de agregação de forma semelhante à implementação do ponto em $\mathbb{T}^2$. Dois pontos representam direções em $\mathbb{R}^3$ e em CGAL, definiu-se que um deles é sempre a origem, ou seja, na implementação presente na biblioteca, a representação é composta por um ponto apenas. Direções em $\mathbb{T}^2$ podem ser instanciadas por dois pontos desde que eles não sejam antipodais ou que ambos não sejam impróprios, pois, neste caso, temos duas direções já que cada ponto impróprio representa uma direção.

4.2 Transformações Afins

Existem quatros tipos de transformações afins implementadas em CGAL: translação, escala, rotação e transformação afim genérica. A representação utilizada consiste em uma matriz de ordem 3. Em $\mathbb{T}^2$, essas transformações são análogas. Uma vez que em CGAL, as transformações de $\mathbb{R}^2$ foram implementadas utilizando-se coordenadas homogêneas não-sinaladas presentes na implementação original da biblioteca, para implementar essas transformações em $\mathbb{T}^2$, foi necessário apenas informar que a coordenada homogênea era a primeira em vez da terceira.

4.3 Predicados e Construções

Em CGAL, denominam-se predicados todas as funções cujo resultado é um tipo booleano ou um tipo enumerado definido na biblioteca. Esses tipos enumerados, geralmente, referem-se à orientação (colinear, lado esquerdo ou lado direito) ou resultados de comparações (igual, menor ou maior). De forma similar, em CGAL, define-se como construção toda função cujo resultado seja um objeto do núcleo da biblioteca. Nas subseções a seguir, descrevemos as principais funções do núcleo $2D$ de CGAL estendidas para $\mathbb{T}^2$.

4.3.1 Cálculo e Comparação de Distâncias em $\mathbb{T}^2$

Nesta seção, iremos discutir o conceito de distância em $\mathbb{T}^2$ e apresentaremos uma função para o cálculo do quadrado da distância entre pontos próprios e um predicado para comparação da distância relativa entre pontos próprios e impróprios e entre pontos impróprios apenas.

A distância absoluta entre pontos próprios no mesmo hemisfério canônico de $\mathbb{T}^2$ é finita, enquanto que a distância entre pontos próprios em hemisférios canônicos distintos de $\mathbb{T}^2$ é um valor "além do infinito". Já a distância entre um ponto próprio e um ponto impróprio, assim como a distância entre dois pontos impróprios distintos, é infinita. Sendo assim, não é possível calcular a distância absoluta entre pontos próprios e impróprios e nem entre pontos impróprios. Porém, como veremos nesta seção, é possível determinar dentre dois pontos próprios dados, o mais próximo a um ponto impróprio. Além disso, dados dois pontos impróprios e um próprio, também é possível determinar, dentre os dois pontos impróprios, o mais próximo ao ponto próprio.

Quadrado da Distância entre dois pontos

A extração de raiz quadrada, além de uma operação lenta, geralmente produz erros de arredondamento comprometendo a robustez do *software*. Muitas vezes, calcula-se o valor

da distância entre dois pontos para usá-lo simplesmente em comparações. Nesses casos, é menos dispendioso e mais confiável usar o quadrado da distância e por essa razão, CGAL oferece apenas funções que calculam o quadrado da distância entre dois pontos. A distância absoluta entre dois pontos próprios é uma função $d_{\mathbb{T}^2_*}$: $\mathbb{T}^2_* \times \mathbb{T}^2_* \longrightarrow \mathbb{T}^1_{x\geq 0}$, definida pela fórmula $d_{\mathbb{T}^2_*}(a,b) = \left[a_w b_w, \sqrt{(a_x b_w - b_x a_w)^2 + (a_y b_w - b_y a_w)^2}\right]$. O quadrado da distância, por sua vez, é definido pela fórmula:

$$d_{\mathbb{T}^2_*}(a,b)^2 = \left[\mathrm{sign}(a_w b_w)(a_w b_w)^2, (a_x b_w - b_x a_w)^2 + (a_y b_w - b_y a_w)^2\right]$$

Distância relativa entre pontos próprios e impróprios

Inicialmente, iremos mencionar uma forma de comparar a distância relativa de dois pontos próprios a um impróprio.

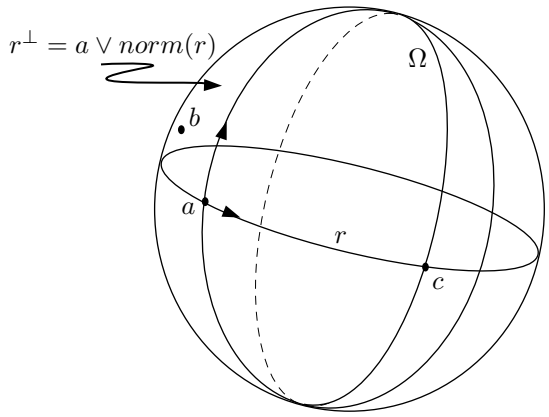


Figura 4.12: Distância relativa entre pontos próprios e impróprios

A princípio, introduzimos algumas notações que serão utilizadas na definição do predicado. Sejam a e b dois pontos; $a \vee b$ denota a reta orientada de a para b. Seja a reta $r = a \vee b$, denota-se $norm(r)$ a direção perpendicular à direção da reta r, a qual, a partir de a, entra no lado positivo de r. $norm(r)$ é representado por um ponto de Ω, conforme mencionado na seção 4.3.7. Utilizaremos também a operação $\diamond$ definida na seção 4.3.6.

Considere a e b pontos próprios no mesmo hemisfério canônico de $\mathbb{T}^2$ e seja c um ponto impróprio (figura 4.12). Queremos definir um predicado que determina se a está mais próximo ou mais distante de c do que b está. Considere a reta $r = a \vee c$ e $r^\perp = a \vee norm(r)$ (figura 4.12). Se $r^\perp \diamond b > 0$, então a está mais próximo de c do que b; por outro lado, se

$r^{\perp} \diamond b < 0$, então b está mais próximo de c do que a; caso contrário, a e b estão à mesma distância de c [Pin98, Sto91].

De forma bastante intuitiva, se varrermos o plano, a partir de a, na direção de c, e encontrarmos b, antes de c, então b está mais próximo de c do que a. Caso contrário, se $b \notin r^{\perp}$, a está mais próximo de c do que b.

A seguir, vamos discutir uma forma de comparar a distância relativa de um ponto próprio a dois impróprios.

Seja um ponto impróprio b e um ponto próprio a conforme ilustra a figura 4.13, podemos observar que quanto mais próximo a está de b (distante da origem de $\mathbb{T}^2$), menor é o ângulo $a\hat{o}b$ (onde o denota a origem de $\mathbb{R}^3$), logo, a medida do ângulo a partir de o é um bom parâmetro para comparar relativamente distâncias infinitas.

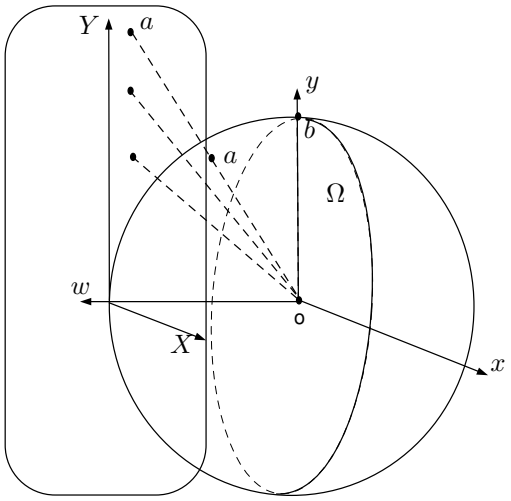


Figura 4.13: Relação entre distâncias infinitas e ângulos

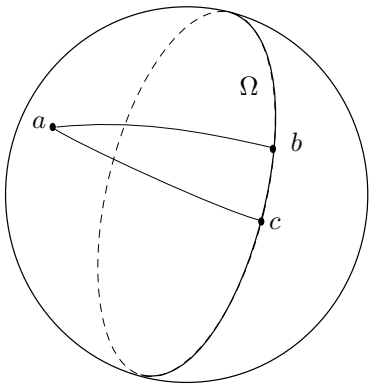


Figura 4.14: Distância relativa entre pontos próprios e impróprios

Sendo assim, para comparar a distância de um ponto próprio a a dois impróprios b e c (figura 4.14), basta comparar a medida dos ângulos $a\hat{o}b$ e $a\hat{o}c$, de forma que:

- se $a\hat{o}b < a\hat{o}c$, a está mais próximo de b do que de c;
- se $a\hat{o}b > a\hat{o}c$, a está mais próximo de c do que de b;
- caso contrário, a está igualmente distante de b e c.

Como veremos, também é possível comparar relativamente a distância entre pontos impróprios. Sejam a e b dois pontos impróprios, como o comprimento do segmento $\overline{ab}$ é proporcional à medida do ângulo $a\hat{o}b$, para comparar a distância de a a b relativamente à

distância de a a outro ponto impróprio c, basta comparar a medida dos ângulos $a\hat{o}b$ e $a\hat{o}c$, de forma semelhante à descrita anteriormente (figura 4.15). De forma análoga, podemos comparar a distância entre dois pontos impróprios e um ponto próprio (figura 4.16).

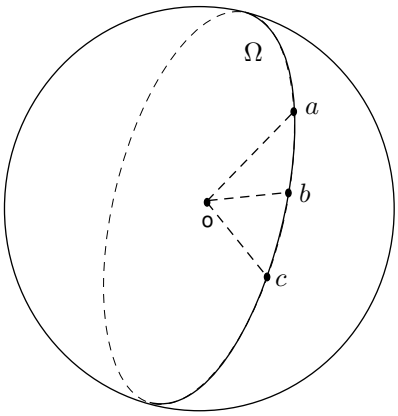


Figura 4.15: Distância relativa - três pontos impróprios

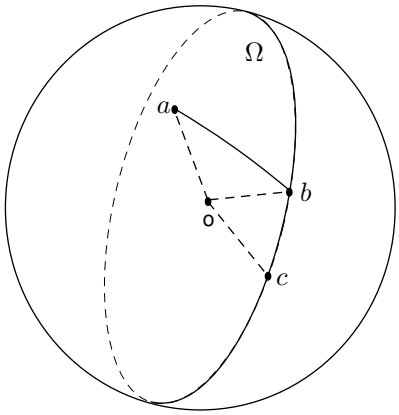


Figura 4.16: Distância relativa - dois pontos impróprios e um próprio

De forma sintética, podemos dizer que a medida do ângulo a partir da origem de $\mathbb{R}^3$ pode ser utilizada para comparar distâncias infinitas envolvendo um ou mais pontos impróprios. Quando temos somente um ponto impróprio, podemos utilizar também a noção de varredura do plano descrita anteriormente.

4.3.2 Ponto Médio

Iremos discutir a extensão da fórmula para se determinar o ponto médio em $\mathbb{T}^2$, ilustrando fórmulas intermediárias inadequadas até chegar a uma fórmula mais consistente.

O ponto médio entre dois pontos $p_0 = (X_0, Y_0)$ e $p_1 = (X_1, Y_1)$ em $\mathbb{R}^2$ é dado pela fórmula:

$$p_m(p_0, p_1) = \left(\frac{X_0 + X_1}{2}, \frac{Y_0 + Y_1}{2}\right)$$

Substituindo-se as coordenadas cartesianas dos pontos por coordenadas homogêneas, ou seja, tomando-se $p_0 = [w_0, x_0, y_0]$ e $p_1 = [w_1, x_1, y_1]$ e fazendo-se $X_0 = \frac{x_0}{w_0}$, $X_1 = \frac{x_1}{w_1}$, $Y_0 = \frac{y_0}{w_0}$ e $Y_1 = \frac{y_1}{w_1}$, a fórmula se torna:

$$p_m(p_0, p_1) = [2w_0w_1, x_0w_1 + x_1w_0, y_0w_1 + y_1w_0]$$

Entretanto, os pontos p_0 e p_1 podem estar no mesmo lado de $\mathbb{T}^2$, ou em lados distintos ou sobre Ω. A fórmula resultante da substituição das coordenadas cartesianas

por coordenadas homogêneas é claramente válida para o caso de dois pontos no aquém (figura 4.17).

Consideremos, o caso em que p_0 e p_1 estejam no além, figura 4.17: é natural que o ponto médio entre ambos esteja no segmento que os une, ou seja, o mesmo deve estar no além. Mas ao aplicarmos a fórmula encontrada, obtemos um ponto no aquém. Por outro lado, se p_0 e p_1 estiverem em lados distintos de $\mathbb{T}^2$ (figura 4.17), a distância entre eles é infinita. Suponha, sem perda de generalidade, que p_0 está no aquém e p_1 no além. Se o ponto médio estiver no aquém, ele estará mais próximo de p_0 do que de p_1. De forma análoga, se o mesmo estiver no além, estará mais próximo de p_1 do que de p_0. Sendo assim, p_m deve estar sobre Ω. Portanto, a fórmula mais consistente, veja [dRS94], é:

$$p_m(p_0, p_1) = [|w_1|w_0 + |w_0|w_1, |w_1|x_0 + |w_0|x_1, |w_1|y_0 + |w_0|y_1]$$

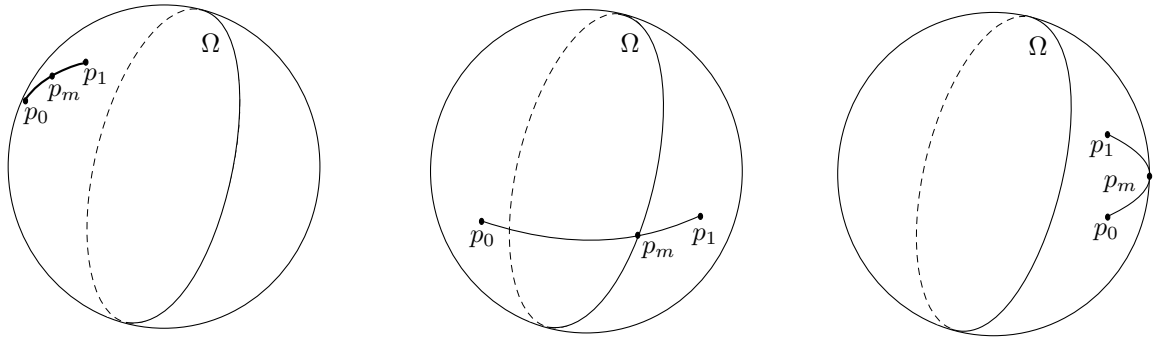


Figura 4.17: Ponto médio de dois pontos no aquém (à esquerda), de um ponto no aquém e outro no além (no meio), de dois pontos no além (à direita)

4.3.3 Comparação lexicográfica

Sejam os pontos p_0 e $p_1 \in \mathbb{T}^2$, denote suas coordenadas normalizadas por $p_0 = [w_0, x_0, y_0]$ e $p_1 = [w_1, x_1, y_1]$, dizemos que p_0 é lexicograficamente menor que p_1 na ordem wxy:

1. se $w_0 < w_1$ ou
2. se $w_0 = w_1$ e $x_0 < x_1$ ou
3. se $w_0 = w_1$ e $x_0 = x_1$ e $y_0 < y_1$

De forma semelhante, podemos definir o predicado lexicograficamente maior, simplesmente trocando o operador $<$ por $>$. Assim, os pontos do além são lexicograficamente, na ordem wxy, menores que os pontos sobre Ω, os quais são lexicograficamente menores que os pontos do aquém.

4.3.4 Orientação de trios de pontos

Outra operação muito utilizada em algoritmos geométricos é a determinação da orientação de um trio de pontos, a qual consiste em um simples cálculo de determinante. Sejam $p = [w_p, x_p, y_p]$, $q = [w_q, x_q, y_q]$ e $r = [w_r, x_r, y_r]$ três pontos em $\mathbb{T}^2$, a orientação deles é denotada por $\triangle(p, q, r)$ e definida como:

$$\triangle(p, q, r) = \begin{vmatrix} w_p & x_p & y_p \\ w_q & x_q & y_q \\ w_r & x_r & y_r \end{vmatrix}$$

- Se $\triangle(p, q, r) = 0$, então os pontos p, q e r são colineares;
- se $\triangle(p, q, r) = +1$, então o ponto p está do lado positivo da reta $\overrightarrow{qr}$;
- se $\triangle(p, q, r) = -1$, então o ponto p está do lado negativo da reta $\overrightarrow{qr}$;

Note que essa operação é equivalente ao teste de sinal de reta contra ponto, sendo o uso desta mais apropriado quando dispomos das coordenadas de três pontos e não dos coeficientes de uma reta e um ponto e vice-versa.

4.3.5 Ponto pertencente a segmento

Este predicado determina se um ponto dado pertence a um segmento. Sejam p, q e r pontos em $\mathbb{T}^2$. Queremos determinar se r pertence ao segmento $\overline{pq}$. A primeira condição é que p, q e r sejam colineares. Dada tal condição, o problema pode ser mapeado para $\mathbb{R}^3$ da seguinte forma: sejam $a = (w_p, x_p, y_p)$, $b = (w_q, x_q, y_q)$, $c = (w_r, x_r, y_r)$ e $d = (-w_p, -x_p, -y_p)$ pontos de $\mathbb{R}^3$. Como p, q e r são colineares em $\mathbb{T}^2$, a, b e c estão num mesmo plano em $\mathbb{R}^3$ o qual inclui a origem o de $\mathbb{R}^3$ (figura 4.18). Se b e c estiverem do mesmo lado da reta $\overrightarrow{ad}$, então $r \in \overline{pq}$ se $\Delta(c, o, a) \neq \Delta(c, o, b)$ em $\mathbb{R}^3$ ou $r = p$.

4.3.6 Posição de ponto em relação a reta

Um predicado bastante freqüente em algoritmos é o de se determinar de que lado de uma reta um ponto está. Sejam $p = [w, x, y]$ e $r = <W, X, Y>$, tal operação é denominada

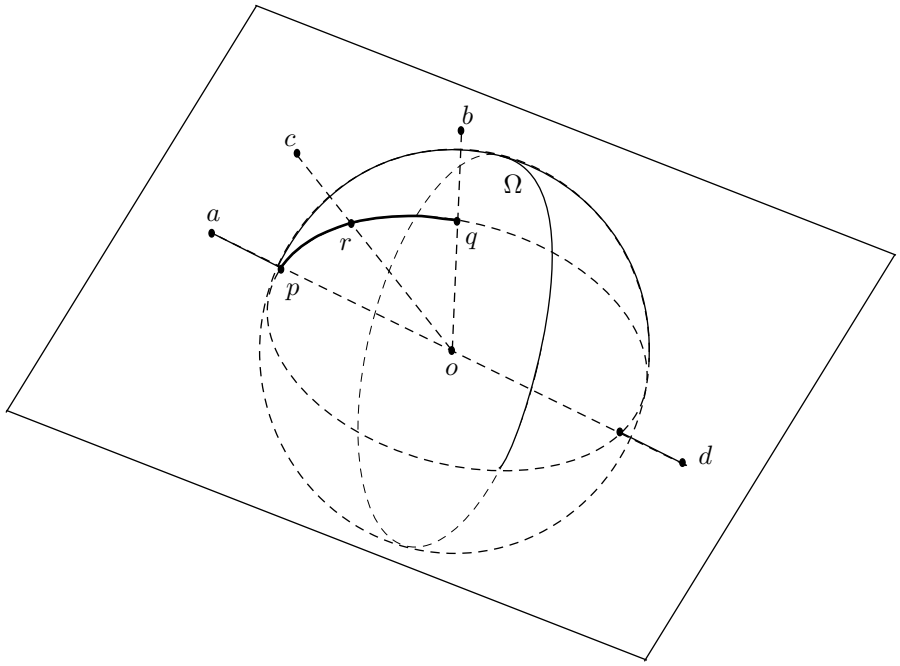


Figura 4.18: Verificação de pertinência de um ponto a um segmento

teste de sinal de p contra r, denotada $r \diamond p$ e definida como:

$$r \diamond p = \text{sign}(Ww + Xx + Yy)$$

- Se $r \diamond p = 0$, p pertence a r;
- se $r \diamond p = +1$, p está do lado positivo de r;
- se $r \diamond p = -1$, p está do lado negativo de r.

Note que, se p está do lado positivo de r, então $\neg p$ estará do lado negativo e, conseqüentemente, $r \diamond \neg p = -(r \diamond p)$. Por outro lado, se $p \in r$ então, $\neg p \in r$ e $r \diamond p = r \diamond \neg p = 0$.

4.3.7 Ângulos

Na geometria Euclidiana, ângulos são, geralmente, representados por números reais e sua determinação pode produzir valores transcendentais mesmo a partir de dados algébricos, comprometendo a robustez do *software*. Ângulos também podem ser representados por dois pontos de uma circunferência, onde o comprimento do arco determinado por eles

indica o valor do ângulo. Em $\mathbb{T}^2$, podemos representar ângulos por pontos de Ω, sendo assim, é possível calcular o ângulo formado por duas retas através de operações algébricas simples, como veremos a seguir.

Observe que o ângulo entre duas retas próprias é uma função de suas direções e não de suas posições absolutas, portanto, para encontrá-lo, basta achar o valor da separação entre dois pontos de Ω.

Em $\mathbb{T}^2$, o ângulo $p\hat{q}r$, formado pelos pontos próprios p, q e r, pode ser calculado através da diferença angular das direções dos segmentos $\overline{qp}$ e $\overline{qr}$, denotada por $\hat{-}$, [Sto91].

$$[0, d_{\overline{qp}}.x, d_{\overline{qp}}.y]\hat{-}[0, d_{\overline{qr}}.x, d_{\overline{qr}}.y] =$$
$$[0, d_{\overline{qp}}.x \times d_{\overline{qr}}.x + d_{\overline{qp}}.y \times d_{\overline{qr}}.y, d_{\overline{qp}}.y \times d_{\overline{qr}}.x - d_{\overline{qp}}.x \times d_{\overline{qr}}.y]$$

onde $d_{\overline{qp}}$ é a direção do segmento $\overline{qp}$ e $d_{\overline{qr}}$ é a direção do segmento $\overline{qr}$.

O ângulo formado por uma reta própria e Ω é sempre um ângulo reto. O ângulo $p\hat{q}r$, supondo que q é impróprio e p e r próprios, será zero graus (figura 4.19).

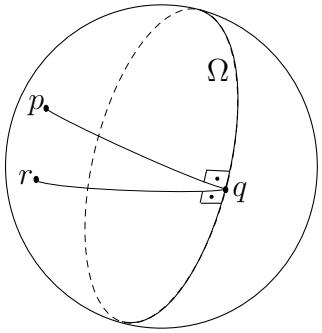


Figura 4.19: Ângulo entre retas próprias e Ω

A comparação entre dois ângulos é bastante simples, sendo preciso refletí-los para os dois primeiros quadrantes antes de compará-los efetivamente (figura 4.20). Utiliza-se dois pontos auxiliares $d = [0, 1, 0]$ (o qual denota o ângulo zero graus) e a origem de $\mathbb{T}^2$, a qual denotamos por o. Sejam p e q pontos no infinito representando dois ângulos. O algoritmo de comparação de ângulos (algoritmo 1) consiste em determinar a orientação de d e de q em relação à reta $\overrightarrow{po}$ (figura 4.20). Desta forma, se d e q estiverem do mesmo lado de $\overrightarrow{po}$, o ângulo p é maior que o ângulo q; se q estiver na reta $\overrightarrow{po}$, os ângulos p e q são iguais e caso contrário, o ângulo q é maior que p.

De forma resumida, este capítulo apresentou a extensão das principais primitivas e funções geométricas que foram utilizadas neste trabalho. Observamos o comportamento

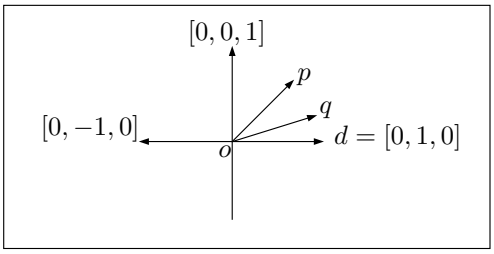


Figura 4.20: Comparação entre ângulos

Algorithm 1 Algoritmo Compara_angulos(p,q)

if $\triangle(p, o, q) \cdot \triangle(p, o, d) = +1$ **then**
  Retornar $(\sphericalangle p > \sphericalangle\ q)$
else if $\triangle(p, o, q) = 0$ **then**
  Retornar $(\sphericalangle p = \sphericalangle q)$
else
  Retornar $(\sphericalangle p < \sphericalangle q)$

das primitivas ao serem estendidas "até o infinito" e "além do infinito". Desta forma, pudemos identificar situações onde algumas dessas primitivas, devido a topologia de $\mathbb{T}^2$, não estão bem definidas. Dentre as funções estendidas, a principal contribuição deste trabalho, foi a estratégia encontrada para comparar distâncias infinitas utilizando-se medidas de ângulos.

No próximo capítulo, discutiremos a extensão de alguns algoritmos para $\mathbb{T}^2$, onde utilizaremos os predicados e as primitivas definidas neste capítulo.

Capítulo 5

Algoritmos Estendidos para $\mathbb{T}^2$

Neste capítulo, apresentamos e discutimos a extensão para $\mathbb{T}^2$ de alguns algoritmos em $\mathbb{R}^2$ da biblioteca CGAL, que é uma biblioteca robusta e confiável [23k02, bas02].

Cada seção deste capítulo apresenta a extensão para $\mathbb{T}^2$ de um algoritmo de CGAL para resolver problemas específicos, mencionando, quando existem, casos especiais a serem tratados em cada um dos algoritmos.

Foram estendidos para $\mathbb{T}^2$ algoritmos de CGAL para localização de pontos em polígonos simples (seção 5.1), para construção de casco convexo (seção 5.2), para construção de uma triangulação arbitrária (seção 5.3), para construção da triangulação de Delaunay (seção 5.3.1), para construção do diagrama de Voronoi (seção 5.3.2), para construção da árvore geradora de distância mínima (seção 5.4), para determinação do maior círculo vazio de pontos com centro próprio (seção 5.5), para construção do grafo de todos os vizinhos mais próximos (seção 5.6), para localização de pontos em uma subdivisão planar (seção 5.7), para determinação da interseção de segmentos (seção 5.8) e para busca em amplitude com iso-retângulo - problema de contagem (seção 5.9).

5.1 Localização de ponto em relação a polígono simples

Este problema consiste em determinar se um ponto está do lado positivo, negativo ou na fronteira de um polígono simples [dRS94]. Seja P um polígono simples, p_0 um vértice de P e q um ponto a ser localizado. Um algoritmo para localização de q em relação a P consiste em determinar o lado do polígono em que o segmento $\overline{p_0q}$ entra ao sair de p_0 e o número de vezes que $\overline{p_0q}$ intercepta os lados do polígono, de forma que o lado de P onde se encontra q seja determinado pela paridade do número de intersecções (algoritmo 2 e figura 5.1).

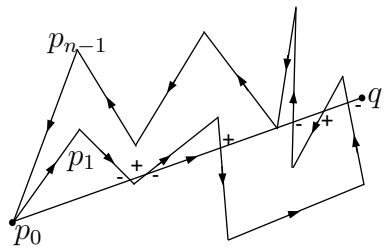


Figura 5.1: Localização de ponto em relação a polígono

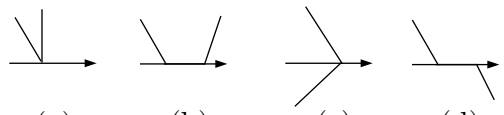
Algorithm 2 Localização de um ponto q em relação a um polígono simples P

1: $s \leftarrow \Delta(q, p_0, p_1)$
2: $t \leftarrow \Delta(p_{n-1}, p_0, p_1)$
3: $b \leftarrow \Delta(q, p_{n-1}, p_0)$
4: **if** $(s \times b) = -1$ **then**
5: $\quad s \leftarrow -t$
6: **else if** $(s \times b) = 0$ **then**
7: $\quad$ **if** $s = 0$ **then**
8: $\quad\quad s \leftarrow (\text{esta\_sobre}(p_0, p_1, q) - 1) \times t$ {retorna 1 se q pertence a $\overline{p_0 p_1}$}
9: $\quad$ **else**
10: $\quad\quad s \leftarrow (\text{esta\_sobre}(p_{n-1}, p_0, q) - 1) \times t$
11: **else**
12: $\quad s$ mantém o valor atribuído inicialmente
13: **for** $i \leftarrow 1$ to $n - 2$ **do**
14: $\quad$ **if** $\overline{p_i p_{i+1}}$ intercepta $\overline{p_0 q}$ **then**
15: $\quad\quad s = -s$
16: Retornar s

Este problema apresenta algumas situações especiais, onde a interseção de $\overline{p_0 q}$ com os lados do polígono devem ser computadas 0 ou 2 vezes/1 ou 3 vezes (figura 5.2). Nos casos (a) e (b), a posição do segmento em relação ao polígono permanece inalterada. Nos casos (c) e (d), o segmento pode estar deixando o lado positivo (negativo) do polígono e entrando em seu lado negativo (positivo).

Para o tratamento destes casos e uma prova de corretude do algoritmo acima, veja [PS85].

Seja n o número de vértices de P. A complexidade do algoritmo descrito para localização do ponto em relação a polígono simples é $O(n)$, uma vez que cada intersecção é determinada em tempo constante e o número de intersecções é $O(n)$.



(a) (b) (c) (d)

Figura 5.2: Casos especiais que ocorrem na localização de ponto em relação a polígono

5.2 Casco Convexo

Para definir casco convexo em $\mathbb{R}^2$, é necessário introduzirmos o conceito de conjunto convexo: um conjunto C é convexo se e somente se *o* segmento que une quaisquer dois pontos de C está no lado limitado de C.

O casco convexo de um conjunto de pontos A é o menor conjunto convexo que contém A, ou seja, é a intersecção de todos os conjuntos convexos que contêm A.

Em $\mathbb{T}^2$, *o* segmento entre dois pontos não está definido quando os mesmos são antipodais. Assim, para que o casco convexo esteja definido, é necessário que o conjunto de pontos não possua pontos antipodais e que as combinações convexas destes não gerem pontos antipodais.

5.2.1 Existência do Casco Convexo

Nesta seção, apresentamos um algoritmo para verificar a existência do casco convexo de um conjunto de pontos A, o qual é denotado por $CH(A)$. O algoritmo consiste em construir o casco convexo dos pontos em cada um dos dois hemisférios canônicos, projetar um deles no hemisfério que se encontra o outro e em seguida, verificar se eles se interceptam. Se ocorre a intersecção, o casco convexo não está definido (algoritmo 3).

Algorithm 3 Verificar a existência de $CH(A)$

1: Particionar A em dois subconjuntos A_1 e A_2: A_1 contém os pontos no aquém e em uma metade de Ω e A_2 consiste em $A - A_1$.
2: **for** $i = 0$ to n **do**
3: Retire p_i de A_2 {n é o número de pontos em A_2 e p_i é um ponto de A_2}
4: Insira $\neg p_i$ em $\neg A_2$
5: Construir os cascos de A_1 e $\neg A_2$ (denotados por $CH(A_1)$ e $CH(\neg A_2)$, respectivamente).
6: Retornar $CH(A_1) \cap CH(\neg A_2) = \emptyset$.

5.2.2 Construção do Casco Convexo

Um algoritmo para construção do casco convexo de A ($CH(A)$) pode se subdividir em dois blocos:

1. Verificar a existência de $CH(A)$;
2. Construir efetivamente $CH(A)$.

No primeiro bloco, pode-se utilizar o algoritmo 3 descrito anteriormente. O passo 2, por sua vez, pode seguir duas abordagens distintas:

1. Descartar os cascos construídos para A_1 e $\neg A_2$ e construir, a partir do zero, o casco convexo de A. Como existem algoritmos ótimos para construí-lo [dRS94], a complexidade total do algoritmo é $O(n_1 \log n_1) + O(n_2 \log n_2) + O(n \log n)$, onde $n_1 = |A_1|$ e $n_2 = |A_2|$.
2. Uma outra abordagem consiste em aproveitar os cascos construídos para A_1 e $\neg A_2$. Uma vez que o casco de A_2 é igual ao conjunto antípoda do casco de $\neg A_2$, o mesmo pode ser encontrado em tempo linear, simplesmente através de uma varredura no conjunto $\neg A_2$, substituindo cada vértice $v \in CH(\neg A_2)$ por $\neg v$. O próximo passo é a união dos cascos de A_1 e A_2, a qual resultará no casco de A. Para isso, basta encontrar as arestas de suporte inferior e superior de $CH(A_1)$ e $CH(A_2)$ e eliminar algumas arestas e vértices desses conjuntos, os quais são internos ao casco convexo de A. Nessa abordagem, a complexidade para encontrar as retas de suporte é $O(n)$, logo, a complexidade total, nesse caso, é $O(n_1 \log n_1) + O(n_2 \log n_2) + O(n)$, onde $n_1 = |A_1|$ e $n_2 = |A_2|$.

Optamos pela segunda abordagem pelo fato de a mesma ser mais eficiente, embora assintoticamente as complexidades sejam equivalentes.

A seguir, é apresentado o algoritmo 4 para construção do casco convexo do conjunto A.

Os algoritmos de CGAL para construção de casco convexo estendidos para $\mathbb{T}^2$ foram a Marcha de Jarvis, o *Quick Hull* e o algoritmo *Graham-Scan*. Em suma, o processo de extensão desses algoritmos consiste em isoladamente aplicá-los em cada lado de $\mathbb{T}^2$ e em seguida combiná-los para gerar o casco desejado. Portanto, qualquer algoritmo aplicável em $\mathbb{R}^2$ pode ser aplicado em $\mathbb{T}^2$, sempre que o casco convexo em $\mathbb{T}^2$ estiver bem definido.

Algorithm 4 Construir $CH(A)$

1: **if** existir $CH(A)$ {Algoritmo 3} **then**
2: **for** $i = 0$ to m **do**
3: Substituir p_i em $CH(\neg A_2)$ por $\neg p_i$. {m é o número de pontos de $CH(\neg A_2)$ e p_i é um ponto de $CH(\neg A_2)$. Ao final do laço, obteremos $CH(A_2)$.}
4: Determinar as retas de suporte inferior e superior de $CH(A_1) \cup CH(A_2)$.
5: Remover as arestas e vértices de $CH(A_1)$ e $CH(A_2)$ não pertencentes a $CH(A)$.
6: Retornar $CH(A)$.
7: **else**
8: $CH(A)$ não existe.

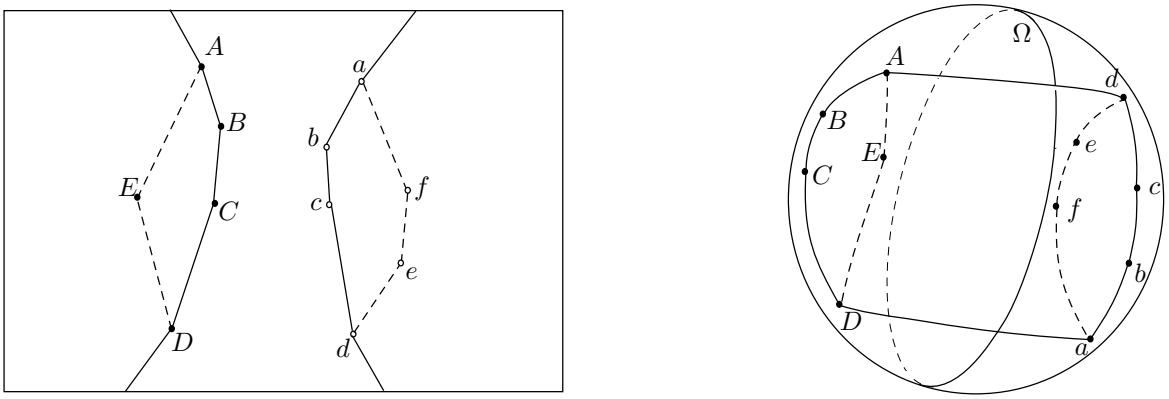


Figura 5.3: Construção das arestas de suporte dos cascos intermediários e eliminação das arestas interiores ao casco final

5.2.3 Marcha de Jarvis

A marcha de Jarvis [PS85] é um algoritmo bastante simples e parte do princípio de que um segmento s faz parte da fronteira do casco convexo de um conjunto de pontos A se, e somente se, todos os pontos de A estão de um mesmo lado ou sobre a reta de suporte de A contendo s (algoritmo 5). Como cada vértice do casco é determinado em tempo linear, se h denota o número de vértices do casco, a complexidade do algoritmo Marcha de Jarvis é $O(n\,h)$. Note, entretanto, que h pode ser $O(n)$.

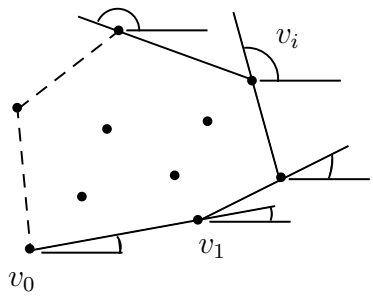


Figura 5.4: Funcionamento da Marcha de Jarvis

Algorithm 5 Marcha de Jarvis (A é o conjunto dos pontos da entrada)

1: Determinar, dentre os vértices de A, v_0 (o vértice de menor ordenada) que certamente fará parte da fronteira do casco
2: **if** $|A| > 1$ **then**
3: $w \leftarrow$ vértice $v \neq v_0 \in A$
4: $u \leftarrow v_0$
5: **while** $w \neq v_0$ **do**
6: **for all** $v_i \in A$ **do**
7: **if** Ângulo formado pelo segmento $\overline{uv_i}$ e a horizontal $<$ Ângulo formado pelo segmento $\overline{uw}$ e a horizontal **then**
8: $w = v_i$
9: Retirar w de A
10: $u \leftarrow w$
11: Inserir w em $CH(A)$
12: **else**
13: Inserir v_0 em $CH(A)$
14: Retornar $CH(A)$

5.2.4 *Quick-Hull*

Este é um algoritmo por divisão e conquista, baseado no paradigma utilizado no algoritmo de ordenação *Quick-Sort*. Seja A o conjunto de pontos da entrada, o *Quick-Hull* [PS85] define uma partição em A, a qual é determinada por uma reta gerada por dois pontos l e r, os quais são o ponto mais à esquerda de A e o mais à direita de A, respectivamente (figura 5.5 e algoritmo 6). Assim como o *Quick-Sort*, no caso médio, o *Quick-Hull* é $O(n \log n)$, mas se a partição gera subconjuntos com cardinalidades desbalanceadas, a complexidade do algoritmo pode chegar a $O(n^2)$.

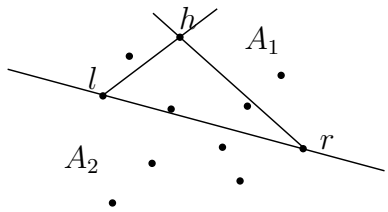


Figura 5.5: Funcionamento do Quick-Hull

Algorithm 6 Quick_Hull (A é o conjunto dos pontos da entrada)

1: Determinar l (ponto de menor abcissa) e r (ponto de maior abcissa) de A
2: Seja A_1 o subconjunto de A formado pelos pontos à esquerda da reta $\overrightarrow{lr}$
3: Seja A_2 o subconjunto de A formado pelos pontos à direita da reta $\overrightarrow{lr}$
4: $S \leftarrow$ Quick_Hull_Aux(A_1, l, r)
5: $I \leftarrow$ Quick_Hull_Aux(A_2, l, r)
6: Concatenar S e I e ordená-los circularmente em sentido anti-horário

Algorithm 7 Quick_Hull_Aux(A, l, r)

1: **if** $|A| = 1$ **then**
2: Retornar o ponto $p \in A$
3: Determinar h, ponto de A que torna máxima a área do triângulo lhr
4: Inserir h na estrutura de dados R
5: Seja A_1 o subconjunto de A formado pelos pontos à esquerda da reta $\overrightarrow{lh}$
6: Seja A_2 o subconjunto de A formado pelos pontos à esquerda da reta $\overrightarrow{hr}$
7: Retornar $R \cup$ Quick_Hull_Aux(A_1, l, h)
8: Retornar $R \cup$ Quick_Hull_Aux(A_2, h, r)

5.2.5 *Graham-Scan*

Seja A, o conjunto dos pontos de entrada, o algoritmo Graham-Scan [PS85] consiste na construção de um polígono estrelado P seguido de uma varredura circular em torno de um ponto no núcleo de P (algoritmo 8):

Algorithm 8 Graham-Scan (A é o conjunto dos pontos da entrada)

1: Construir um polígono estrelado P (figura 5.6). Os pontos de A são ordenados circularmente, em sentido anti-horário, em torno de um dos pontos dados. Essa ordenação induz uma ordem nos vértices de P
2: Varrer circularmente os vértices de P para eliminar ângulos reflexos (figura 5.7)

A complexidade do passo 1 é $O(n \log n)$ enquanto que a varredura do passo 2 pode ser feita em tempo linear, resultando, portanto, num algoritmo ótimo.

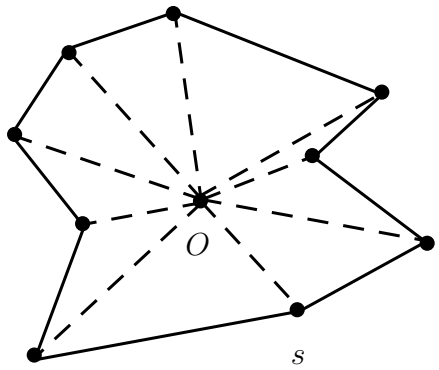


Figura 5.6: Graham-Scan - construção do polígono estrelado

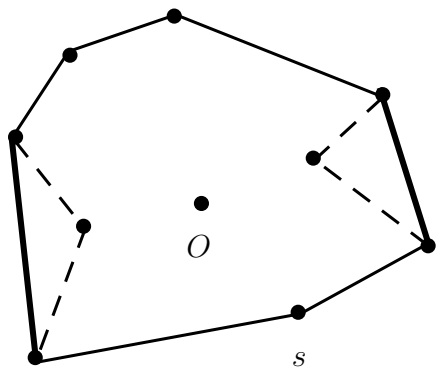


Figura 5.7: Graham-Scan - eliminação de ângulos reflexos

5.3 Triangulação

Uma triangulação de um conjunto de pontos S em $\mathbb{R}^2$ é uma sudivisão do casco convexo de S, onde as faces são triângulos (figura 5.8). A triangulação de Delaunay é um tipo especial de triangulação que possui propriedades interessantes [PS85]:

1. todo círculo que circunscreve um triângulo (uma face da triangulação) não possui nenhum outro vértice da triangulação em seu interior;

2. essa triangulação maximiza o menor ângulo.

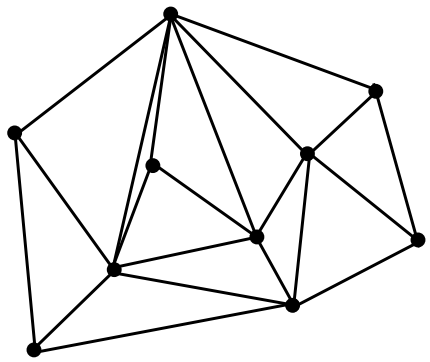


Figura 5.8: Triangulação Arbitrária

Em $\mathbb{R}^2$, qualquer triangulação está totalmente definida no interior do casco convexo (figura 5.8), já em $\mathbb{T}^2$, o casco convexo está definido somente se os pontos estão totalmente contidos em um hemisfério, devido à antipossidade do espaço. Uma triangulação pode ultrapassar a fronteira do casco e particionar os dois hemisférios canônicos de $\mathbb{T}^2$ (figura 5.9). Daqui por diante, denotaremos uma triangulação arbitrária por T e, durante a discussão deste problema, iremos contrastar as características da mesma em $\mathbb{R}^2$ e em $\mathbb{T}^2$.

Em $\mathbb{R}^2$, temos as faces interiores ao casco convexo e uma face de área ilimitada, a qual é equivalente a $\mathbb{R}^2 - T$. Desta forma, temos faces limitadas triangulares e uma face ilimitada poligonal complementar ao casco convexo em relação a $\mathbb{R}^2$.

Em $\mathbb{T}^2$, por sua vez, quando o casco convexo não está definido, todas as faces são triangulares, ou seja, temos uma situação mais homogênea do que em $\mathbb{R}^2$.

Geralmente, em $\mathbb{R}^2$, utiliza-se um artifício na implementação para que as faces sejam sempre triangulares e para que possam ser tratadas de forma geral. Este artifício consiste em criar um ponto no infinito v_∞ e decompor a face ilimitada original em triângulos. Para isso, toma-se cada vértice v do casco e cria-se uma aresta $\overline{vv_\infty}$ (figura 5.10). O vértice v_∞ não possui coordenadas representáveis em $\mathbb{R}^2$ e, por isso, deve ser tratado de forma especial.

Um algoritmo para construção de uma triangulação arbitrária estendido para $\mathbb{T}^2$ é um algoritmo incremental, cuja implementação em $\mathbb{R}^2$ está presente em CGAL.

Este algoritmo é constituído essencialmente de duas etapas:

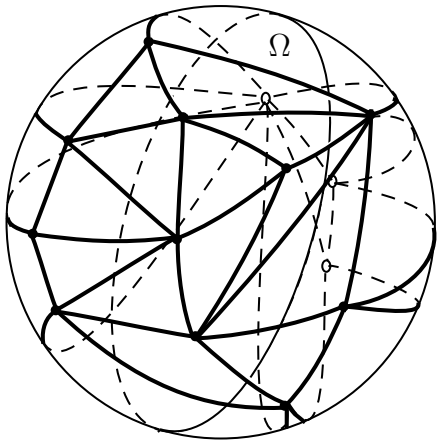


Figura 5.9: Triangulação Arbitrária em $\mathbb{T}^2$ quando o casco convexo não está definido

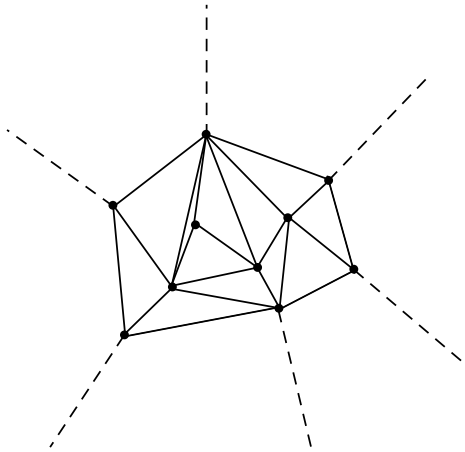


Figura 5.10: Inserção de um vértice infinito v_∞ estendendo-se $\mathbb{R}^2$. As arestas pontilhadas denotam arestas onde um extremo é v_∞

1. Localização do vértice v a ser inserido na triangulação construída até o momento;
2. Inserção de v na triangulação de acordo com sua localização.

Em $\mathbb{R}^2$, um vértice pode estar no interior de uma face, ou sobre uma aresta, ou fora do casco convexo ou ainda ser um vértice já existente na triangulação.

Em $\mathbb{T}^2$, as possíveis localidades de um ponto são as mesmas de $\mathbb{R}^2$, caso a triangulação construída até então esteja no interior do casco convexo (casco convexo definido). Caso contrário, temos três possibilidades: ou o ponto está no interior de uma face, ou sobre uma aresta ou é um vértice já existente na triangulação.

A seguir, iremos descrever as duas etapas deste algoritmo estendido para $\mathbb{T}^2$.

O algoritmo para localização de v na triangulação consiste em traçar um segmento de reta partindo de um vértice qualquer da triangulação (o qual denotamos por s) até v (figura 5.11). Em seguida, caminha-se ao longo deste segmento, passando por cada face interceptada por ele, até chegar a v, então, verifica-se se v é um vértice da face atual, ou se está em seu interior, ou sobre alguma de suas arestas ou se a face atual é uma face que contém v_∞ e, neste caso, v é externo ao casco convexo.

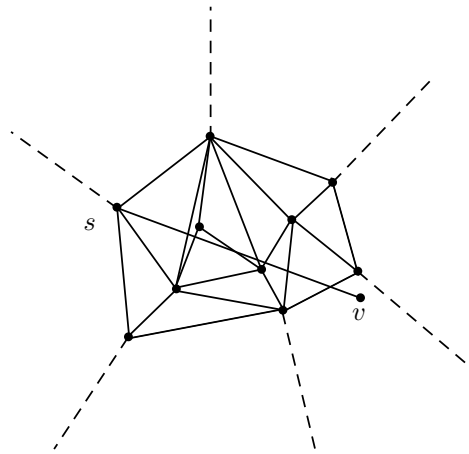


Figura 5.11: Localização do vértice v a ser inserido na triangulação

A inserção de um vértice v no interior de uma face provoca a divisão da mesma em três novas faces: vv_0v_1, vv_1v_2 e vv_0v_2 (figuras 5.12 e 5.13). Logo, a função para inserção de um vértice no interior de uma face consiste em criar três faces e identificar seus vizinhos.

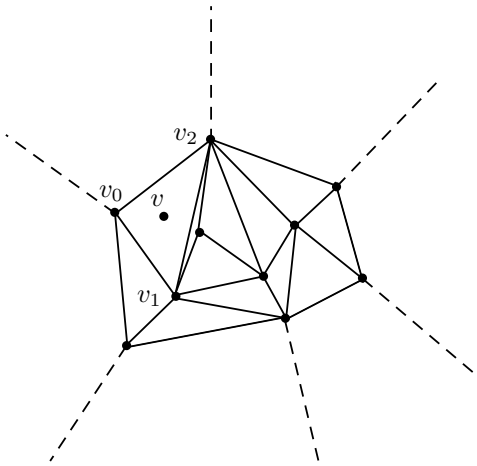


Figura 5.12: Vértice v localizado no interior de uma face

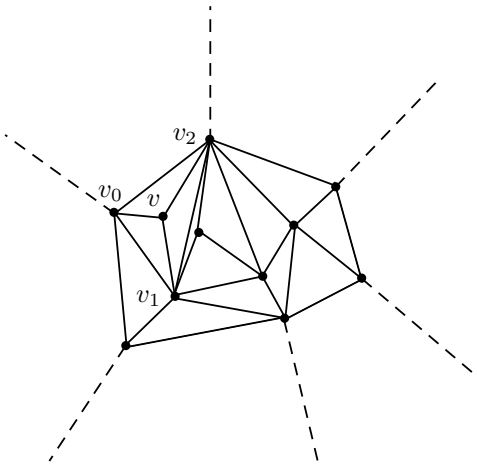


Figura 5.13: Inserção do vértice v no interior de uma face

A inserção de um vértice sobre uma aresta e (figuras 5.14 e 5.15) provoca a criação de quatro novas faces, sendo que o número total de faces é aumentado de dois, uma vez que as duas faces que compartilham e serão subdivididas. Para inserção de um vértice sobre uma aresta, utiliza-se a função para inserção de um vértice no interior de uma face, seguido de uma inversão da diagonal (denotada por D-inversão) de um quadrilátero.

Seja o quadrilátero $v'v_1vv_0$, a D-inversão troca a aresta $\overline{v_0v_1}$ pela aresta $\overline{v'v}$ (figura 5.15).

Se v já estiver na triangulação, o mesmo é descartado e não ocorre nenhuma inserção.

A inserção de um vértice v no exterior do casco convexo também é uma inserção no interior de uma face (uma face ilimitada que contém v_∞). Logo, ao inserir v no interior de uma face, três novas faces são criadas. Observe que duas delas são ilimitadas e correspondem exatamente às duas novas arestas que foram criadas devido à inserção de v (figura 5.16).

A triangulação deve estar completa, ou seja, os vértices incidentes nas faces ilimitadas devem ser vértices do casco do conjunto de pontos (desconsiderando-se v_∞). Para isso, todos os ângulos reflexos formados por vv_iv_{i+1}, onde v_i e v_{i+1} são vértices do casco convexo da triangulação, antes da inserção de v, devem ser eliminados. Observe na figura 5.17 que, para eliminar esses ângulos é suficiente trocar a diagonal $\overline{v_iv_\infty}$ do quadrilátero $v_\infty\, v_{i+1}\, v_i\, v$ pela diagonal $\overline{v_{i+1}v}$. O número de vezes em que ocorre uma D-inversão é igual ao número de faces onde o ângulo $v\hat{v}_iv_{i+1}$ é reflexo.

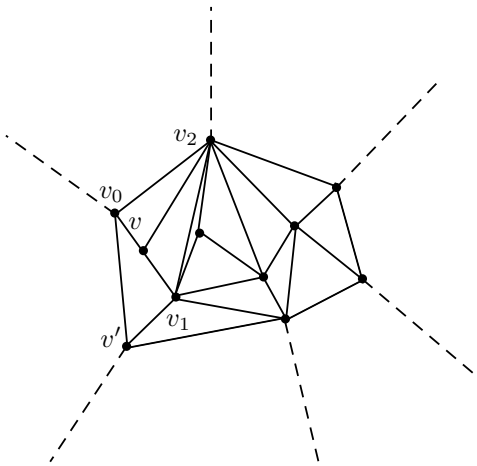


Figura 5.14: Vértice v localizado sobre uma aresta

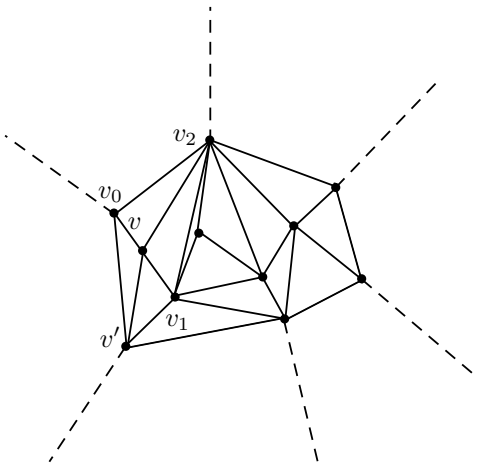


Figura 5.15: Inserção do vértice v sobre uma aresta

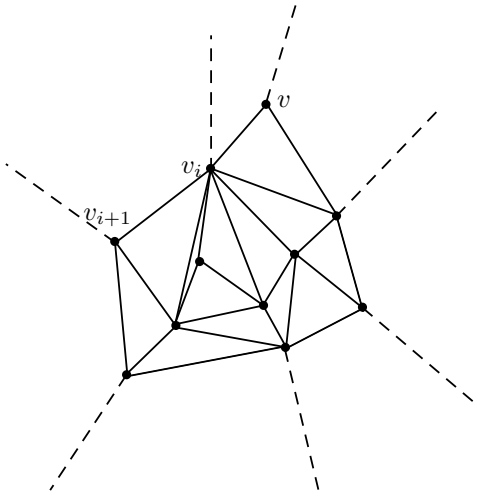


Figura 5.16: Inserção do vértice v no exterior do casco convexo

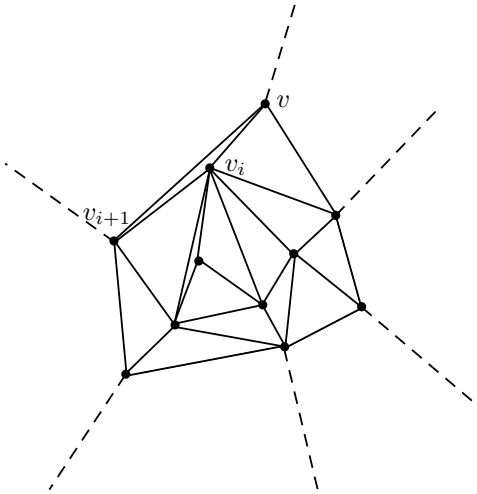


Figura 5.17: Triangulação resultante da inserção de v após as D-inversões necessárias

Após a inserção de um ponto no exterior do casco convexo, duas situações podem ocorrer:

1. o casco convexo do conjunto de vértices continua definido (figura 5.17).
2. o casco convexo do conjunto de vértices não está mais definido. Neste caso, a triangulação particiona os dois hemisférios canônicos de $\mathbb{T}^2$ (figura 5.9).

Se ocorre a situação 2, então v é inserido em todas as faces ilimitadas externas ao casco convexo da triangulação e, como a triangulação passa a estar definida nos dois hemisférios canônicos de $\mathbb{T}^2$, v_∞ é substituído por v.

A partir desse ponto, todas as inserções realizadas na triangulação ou são inserções de vértices sobre arestas ou inserções de vértices no interior de faces, uma vez que vértices já existentes na triangulação são descartados. Sendo assim, o número de possíveis localidades de vértices na triangulação é reduzido de quatro para três.

Se, após a inserção de um vértice v, o casco convexo torna-se indefinido, então v pertence ao hemisfério complementar ao hemisfério que continha a triangulação antes da inserção de v.

Situações especiais interessantes ocorrem quando v é antípoda de algum vértice do casco convexo do conjunto de vértices da triangulação construída até o momento, pois, neste caso, a triangulação possuirá um segmento entre pontos antipodais (figura 5.18).

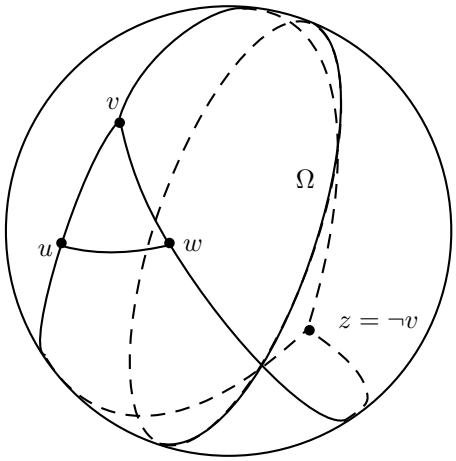


Figura 5.18: Inserção de vértice antípoda ($\neg v$) a um vértice do casco convexo (v)

Como discutido anteriormente, existem infinitos segmentos entre pontos antipodais, logo, é preciso definir consistemente sua direção. É imprescindível garantir que a escolha dessa direção em $\mathbb{T}^2$ não gere intersecções para que a triangulação continue sendo uma subdivisão planar. Seja v um ponto da triangulação, pertencente ao casco convexo e, sejam u e w seus vizinhos na fronteira do casco (figura 5.18). Suponha que $\neg v$ esteja sendo inserido na mesma. Uma direção conveniente para o segmento $\overline{v\neg v}$ é aquela que divide ao meio o ângulo $u\hat{v}w$. Essa é uma escolha arbitrária, mas que garante a consistência da triangulação e, como veremos mais adiante, preserva as propriedades da triangulação de Delaunay sendo, portanto, uma solução homogênea para os dois problemas.

Observe que segundo a situação descrita na figura 5.18, antes da inserção de $\neg v$, o casco convexo estava definido e, após sua inserção, a triangulação passou a estar definida nos dois hemisférios canônicos de $\mathbb{T}^2$. A partir desse ponto, é importante ressaltar que a inserção de um ponto b antípoda a qualquer ponto da triangulação é realizada no interior de uma face que não contém $\neg b$ ou sobre uma aresta que não possui $\neg b$ como extremo. Logo, após a inserção do primeiro ponto antípoda a um ponto do casco convexo, não surgem mais faces que possuam dois pontos antipodais como vértices. Essa característica surge devido à escolha da direção do segmento entre pontos antipodais.

Note que problemas de degenerações como faces com arestas colineares ocorrem apenas quando um ponto inserido é antípoda de algum ponto do casco convexo ou é antípoda de outro que pertença a uma aresta do casco. Em outras palavras, inserções de pontos antípodas a pontos do interior do casco convexo são tratadas como inserções comuns.

Considere a triangulação da figura 5.18, seja t um ponto pertencente ao segmento $\overline{vw}$, a inserção de t na triangulação provoca a divisão da face uvw nas novas faces tuv e tuw e, também, a divisão da face $vw\neg v$ na face $tv\neg v$ e na face degenerada $t\neg vw$ (figura 5.19).

Note que a inserção de um ponto em uma aresta de uma face que possui pontos antipodais, sendo tal aresta com extremos não antipodais, implica no surgimento de novas faces degeneradas, onde três pontos são colineares.

Novamente, considere a figura 5.18, e seja um ponto t pertencente à aresta $\overline{v\neg v}$ (de acordo com sua direção), a inserção de t na triangulação provoca a divisão da face $v\neg vw$ nas novas faces não degeneradas, tvw e $t\neg vw$ e a divisão da face $v\neg vu$ nas novas faces tuv e $tu\neg v$ (figura 5.20).

A seguir, apresentamos o algoritmo estendido (algoritmo 9) de forma mais concisa.

Ou seja, a inserção de um ponto em uma aresta com extremos antipodais ocasiona o desaparecimento de faces degeneradas.

A complexidade do algoritmo de CGAL estendido para $\mathbb{T}^2$ é dada pela complexidade de localização adicionada à complexidade de inserção. Na localização, no pior caso, o segmento $\overline{sv}$ intercepta $O(n)$ faces (sendo n o número de pontos da triangulação). Como, dada uma face f, um triângulo, na qual o vértice v a ser inserido se encontra, discriminar

Algorithm 9 Construir uma triangulação arbitrária T (A é o conjunto dos pontos de entrada)

1: **for all** $v_i \in A$ **do**
2: **if** $|T| = 0$ **then**
3: Inserir v_i em T e retirar v_i de A
4: $d \leftarrow 1$
5: **else if** $|T| = 1$ **then**
6: Inserir v_i em T, criar as faces uv_i, uv_∞ e v_iv_∞ {u é o vétice em T} e retirar v_i de A
7: **else if** $d = 1$ **then**
8: **if** colinear(u, v, v_i) {u e v são os dois vértices incidentes a v_∞} **then**
9: **if** $v_i \in \overline{uv}$ {v_i pertence ao casco de T?} **then**
10: **while** Não Inseriu {p/ cada face finita f_i de T, sejam w e z os vértices} **do**
11: **if** $v_i \in \overline{wz}$ **then**
12: Inserir v_i em T e criar as faces wv_i e v_iz
13: **else if** $v_i \in \overline{\neg u \neg v}$ **then**
14: Substituir v_∞ por v_i, serão criadas as faces uv_i e vv_i, onde u e v são os vértices incidentes a v_∞.
15: **else**
16: **if** seja uv_∞ uma face infinita e seja uz a outra face incidente a u. $u \in \overline{v_iz}$ **then**
17: Inserir v_i e criar as faces uv_i e v_iv_∞
18: **else**
19: Inserir v_i e criar as faces vv_i e v_iv_∞
20: **else**
21: Inserir v_i em uma face wz de T e eliminar todos os ângulos reflexos
22: $d \leftarrow 2$
23: **else**
24: Determinar a localização do vértice v a ser inserido na triangulação
25: **if** v está no interior de uma face $v_0v_1v_2$ **then**
26: A face $v_0v_1v_2$ é decomposta nas faces: vv_0v_1, vv_1v_2 e vv_0v_2 (figuras 5.12 e 5.13)
27: **else if** v está sobre uma aresta v_0v_1 e esta pertence às faces $v_0v_1v_2$ e v_0v_1v' **then**
28: A face $v_0v_1v_2$ é decomposta nas faces: v_0vv_2 e vv_1v_2
29: A face v_0v_1v' é decomposta nas faces: v_0vv' e vv_1v'
30: **else if** v é externo ao casco convexo **then**
31: **if** v é antípoda de um vértice z do casco convexo **then**
32: v_∞ é substituído por v e a direção do segmento $\overline{vz}$ é a direção que divide ao meio o ângulo $u\hat{v}w$ (figura 5.18), onde uvz e vzw são as duas faces que contém o segmento antipodal.
33: **else**
34: Inserir v na face $v_iv_{i+1}v_\infty$ ($\triangle(vv_iv_{i+1}) = -1$) {o passo 2 descreve a inserção}
35: Eliminar ângulos reflexos $v\hat{v}_iv_{i+1}$ (onde v_i e v_{i+1} – vértices do casco)
36: Retornar T

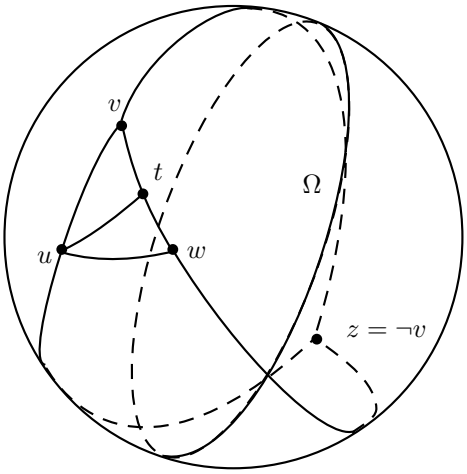


Figura 5.19: A inserção de t na triangulação resulta na criação de face degenerada

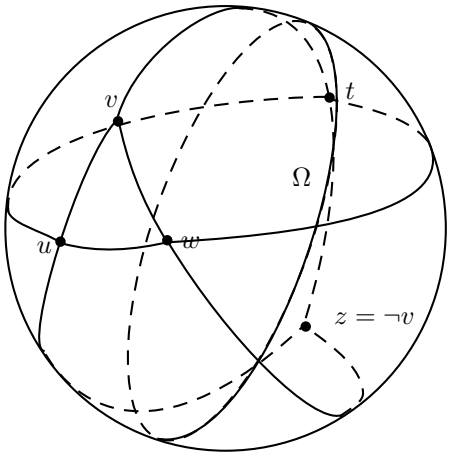


Figura 5.20: A inserção de t na triangulação resulta no desaparecimento de face degenerada

se v é um vértice de f, está sobre uma aresta de f ou está no interior de f toma tempo constante, a complexidade da localização de um ponto na triangulação é feita em $O(n)$. A segunda etapa, ou seja, a inserção de um ponto na triangulação é feita em tempo constante, exceto quando o mesmo está no exterior do casco convexo da triangulação, sendo a complexidade de inserção, neste caso, $O(n)$. Logo, a complexidade para se inserir cada ponto na triangulação é $O(n)$ e a complexidade para se construir toda a triangulação é $O(n^2)$.

5.3.1 Triangulação de Delaunay

Como definido anteriormente, uma triangulação de Delaunay possui as seguintes propriedades: o círculo que circunscreve os vértices de uma face é vazio de outros vértices e a triangulação maximiza o menor ângulo.

O algoritmo utilizado para construção da triangulação de Delaunay também é um algoritmo incremental que constrói uma triangulação arbitrária e em seguida, restaura as propriedades de Delaunay.

Após a inserção de cada ponto, as propriedades de Delaunay devem ser reestabelecidas. Devido ao fato de a circunferência com centro impróprio não estar sempre bem definida, não foi possível utilizar essa propriedade na implementação. Por outro lado, seria possível

utilizar a propriedade de maximização do menor ângulo, mas devido ao fato de o cálculo de ângulos utilizar funções que geram valores transcendentais, por questões de robustez, optou-se por não calcular explicitamente ângulos na implementação.

Uma propriedade equivalente à maximização do menor ângulo é a maximização da menor altura dos triângulos gerados. Essa propriedade foi utilizada na implementação da função que restaura as propriedades de Delaunay.

Na implementação, a classe que implementa a triangulação de Delaunay é uma subclasse da classe que implementa a triangulação arbitrária, sendo que as funções de inserção e localização de pontos da classe pai não foram redefinidas na classe filha. Na classe da triangulação de Delaunay, foi desenvolvida uma função para restaurar as propriedades de Delaunay, a qual é chamada após a inserção de cada ponto.

A complexidade da inserção de cada ponto na triangulação é linear e, no pior caso, o número de D-inversões necessárias para se reestabelecer as propriedades de Delaunay pode ser linear no número de vértices da triangulação. Logo, a complexidade do algoritmo utilizado para construção da triangulação de Delaunay é $O(n^2)$.

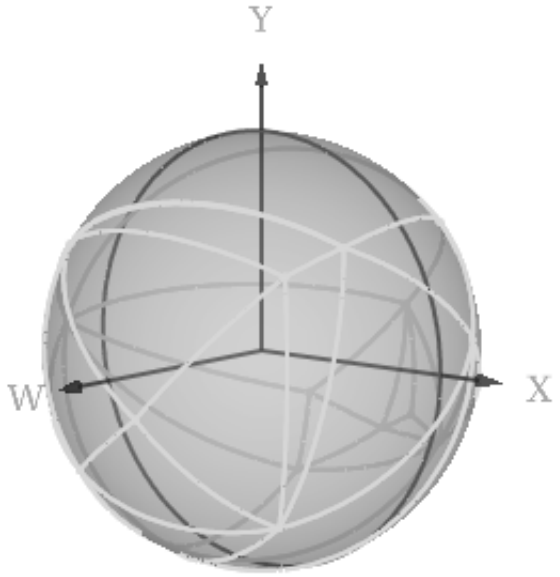


Figura 5.21: Triangulação de Delaunay com vértices no aquém e no além

A seguir, apresentamos, de forma simplificada, o algoritmo 10 utilizado para cons-

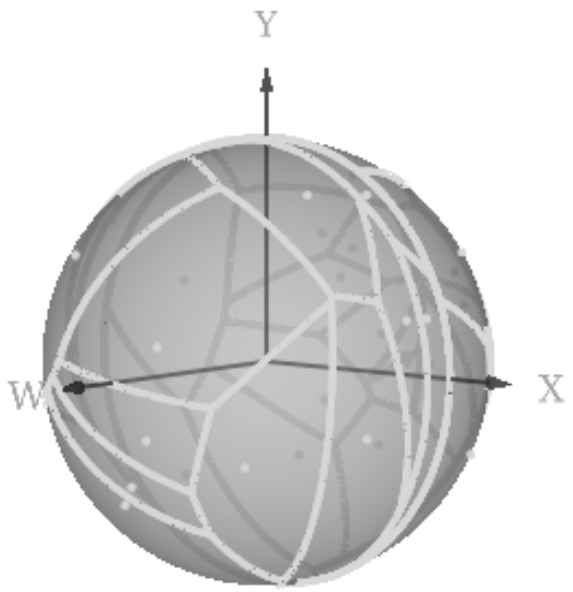


Figura 5.22: Diagrama de Voronoi com sítios nos dois hemisférios de $\mathbb{T}^2$

trução da triangulação de Delaunay.

5.3.2 Diagrama de Voronoi

Dado um conjunto S de n sítios no plano, o diagrama de Voronoi particiona o plano em regiões, sendo que cada região R está associada a um sítio $p \in S$, tal que a distância de qualquer ponto q no interior de R a p é menor do que a distância de q a qualquer outro sítio em S [PS85].

O diagrama de Voronoi é dual da triangulação de Delaunay [dBvKOS97], onde cada vértice de Voronoi é circumcentro de uma face da triangulação e, se duas faces possuem uma aresta em comum, os vértices de Voronoi correspondentes estão conectados por uma aresta do diagrama (algoritmo 11).

Em $\mathbb{T}^2$, quando os sítios de S estão apenas em um hemisfério canônico, o diagrama de Voronoi se estende para o outro hemisfério canônico, construindo o diagrama de vizinho mais distante dos pontos antípodas dos sítios de S [Pin98]. O diagrama de vizinho mais distante particiona o plano em regiões, de forma que os pontos no interior de uma região

Algorithm 10 Construir triangulação de Delaunay T (A é o conjunto dos pontos da entrada)

1: **for all** $v_i \in A$ **do**
2: **if** $|T| = 0$ **then**
3: Inserir v_i em T e retirar v_i de A
4: $d \leftarrow 1$
5: **else if** $|T| = 1$ **then**
6: Inserir v_i em T, criar as faces uv_i, uv_∞ e v_iv_∞ {u é o vétice em T} e retirar v_i de A
7: **else if** $d = 1$ **then**
8: **if** colinear(u, v, v_i) {u e v são os dois vértices incidentes a v_∞} **then**
9: **if** $v_i \in \overline{uv}$ {v_i pertence ao casco de T?} **then**
10: **while** Não Inseriu {p/ cada face finita f_i de T, sejam w e z os vértices} **do**
11: **if** $v_i \in \overline{wz}$ **then**
12: Inserir v_i em T e criar as faces wv_i e v_iz
13: **else if** $v_i \in \overline{\neg u \neg v}$ **then**
14: Substituir v_∞ por v_i, serão criadas as faces uv_i e vv_i, onde u e v são os vértices incidentes a v_∞.
15: **else**
16: **if** seja uv_∞ uma face infinita e seja uz a outra face incidente a u. $u \in \overline{v_iz}$ **then**
17: Inserir v_i e criar as faces uv_i e v_iv_∞
18: **else**
19: Inserir v_i e criar as faces vv_i e v_iv_∞
20: **else**
21: Inserir v_i em uma face wz de T e eliminar todos os ângulos reflexos
22: $d \leftarrow 2$
23: **else**
24: Determinar a localização do vértice v a ser inserido na triangulação
25: **if** v está no interior de uma face $v_0v_1v_2$ **then**
26: A face $v_0v_1v_2$ é decomposta nas faces: vv_0v_1, vv_1v_2 e vv_0v_2 (figuras 5.12 e 5.13)
27: **else if** v está sobre uma aresta v_0v_1 e esta pertence às faces $v_0v_1v_2$ e v_0v_1v' **then**
28: A face $v_0v_1v_2$ é decomposta nas faces: v_0vv_2 e vv_1v_2
29: A face v_0v_1v' é decomposta nas faces: v_0vv' e vv_1v'
30: **else if** v é externo ao casco convexo **then**
31: **if** v é antípoda de um vértice z do casco convexo **then**
32: v_∞ é substituído por v e a direção do segmento $\overline{vz}$ é a direção que divide ao meio o ângulo $u\hat{v}w$ (figura 5.18), onde uvz e vzw são as duas faces que contém o segmento antipodal.
33: **else**
34: Inserir v na face $v_iv_{i+1}v_\infty$ ($\triangle(vv_iv_{i+1}) = -1$) {o passo 2 descreve a inserção}
35: Eliminar ângulos reflexos $v\hat{v}_iv_{i+1}$ (onde v_i e v_{i+1} – vértices do casco)
36: Restaurar as propriedades de Delaunay em T
37: Retornar T

Algorithm 11 Construir diagrama de Voronoi V

1: Construir triangulação de Delaunay T (algoritmo 10)
2: **for all** face $f_i \in T$ **do**
3: $c \leftarrow$ circumcentro(f_i)
4: **for all** face h_j vizinha de f_i **do**
5: $p \leftarrow$ circumcentro(h_j)
6: Construir aresta $\overline{cp}$ e inserí-la em V

estão mais distante(s) do(s) sítio(s) associado(s) a ela do que dos demais sítio(s).

Logo, ao se resolver o problema da construção do diagrama de Voronoi dos sítios de um conjunto S contido num dos hemisférios canônicos, automaticamente, obtemos por antipossidade o diagrama de vizinho mais distante dos sítios de S.

Este problema ilustra a característica interessante de $\mathbb{T}^2$ de, ao se resolver um problema, automaticamente solucionar outro relacionado, sendo essa uma característica notável do espaço, pois, como neste caso, dispensa o esforço de se projetar dois algoritmos para resolver dois problemas complementares. Essa dualidade, geralmente, ocorre em problemas que envolvem o conceito de distância em $\mathbb{T}^2$.

Neste trabalho, o diagrama de Voronoi foi obtido a partir da triangulação de Delaunay, utilizando-se a propriedade da dualidade e, como o número de faces da triangulação é linear no número de sítios de S (denotado por n), o diagrama de Voronoi é obtido em tempo linear, logo, como a complexidade do algoritmo incremental para construir a triangulação de Delaunay é $O(n^2)$, a complexidade do algoritmo para construção do diagrama de Voronoi é $O(n^2)$.

5.4 Árvore Geradora de Distância Mínima

Seja $G = (V, E)$ um grafo conexo, onde V denota o conjunto de vértices e E o conjunto de arestas de G. Seja uma função $w : E \to \mathbb{R}$ que associa a cada aresta $e \in E$ um valor $w(e) \in \mathbb{R}$, denominado peso de e. A árvore geradora mínima, denotada por AGM, é um subgrafo conexo, acíclico, gerador de G, que minimiza a soma dos pesos das arestas. A árvore geradora Euclidiana mínima, denotada por $AGEM$, é uma AGM, onde o peso das arestas é a distância Euclidiana entre seus extremos [PS85].

Considerando este problema em $\mathbb{T}^2$, a função peso w das arestas passa a ser a distância em $\mathbb{T}^2$ entre seus extremos e chamaremos a árvore de $AGDM$ - árvore geradora de distância mínima - e o valor de $w(e)$ pode ser:

1. finito (distância entre dois pontos próprios no mesmo hemisfério canônico de $\mathbb{T}^2$);

2. infinito (distância entre dois pontos, onde pelo menos um está sobre Ω) ou

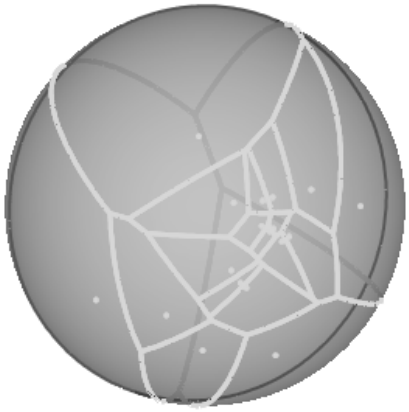


Figura 5.23: Diagrama de Voronoi de sítios contidos somente no aquém

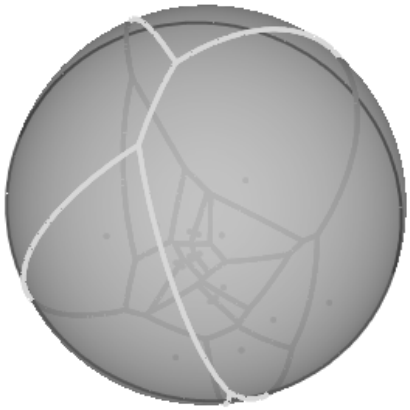


Figura 5.24: Diagrama de Voronoi de vizinho mais distante dos antípodas dos sítios do aquém

3. além do infinito (distância entre dois pontos próprios em hemisférios canônicos distintos de $\mathbb{T}^2$).

Para encontrar a $AGDM$ em $\mathbb{T}^2$, utilizamos o algoritmo de Kruskal [Man89], no qual, uma operação básica é comparar o peso de duas arestas e_1 e e_2 e decidir qual delas será a próxima a ser incluída na $AGDM$.

Quando e_1 e e_2 são infinitas, (caso 2) utilizamos o predicado para comparar relativamente distâncias infinitas que foi descrito na seção 4.3.1 do capítulo 4.

Como provado a seguir, a $AGDM$ é um subgrafo da triangulação de Delaunay e, sendo assim, no algoritmo estendido a $AGDM$ é gerada a partir da mesma, logo, a construção desta constitui uma etapa de pré-processamento do algoritmo.

Teorema: Uma $AGDM$ é um subgrafo da triangulação de Delaunay.

Prova: Seja T a $AGDM$ de V, seja $w(T)$ a soma dos pesos de T. Sejam a e b dois sítios tais que $\overline{ab}$ é uma aresta de T. Suponha que não existe um círculo vazio de pontos passando por a e b e, em particular, o círculo cujo diâmetro é o segmento $\overline{ab}$ contém um sítio, chamado c (figura 5.26).

A remoção de $\overline{ab}$ de T divide a árvore T em duas subárvores. Assuma, sem perda de generalidade, que c está na mesma subárvore que a. Removendo-se a aresta $\overline{ab}$ e incluindo-se $\overline{bc}$ a T, temos uma árvore geradora T' cujo peso total é:

$$w(T') = w(T) + |\overline{bc}| - |\overline{ab}| < w(T)$$

Figura 5.25: Árvore geradora de distância mínima

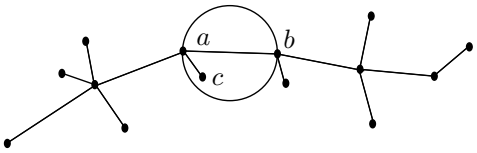


Figura 5.26: Prova de que uma *AGDM* é um subgrafo da triangulação de Delaunay

Pois $|\overline{ab}|$ é o diâmetro do círculo e, logo, $|\overline{bc}| < |\overline{ab}|$. Portanto, o círculo de diâmetro $\overline{ab}$ é vazio de sítios. Por conseguinte, existe um sítio $s \in S$, tal que o círculo abs é vazio de outros sítios. Logo, abs é um triângulo da triangulação de Delaunay de S. CQD

A seguir, apresentamos de forma resumida o algoritmo utilizado para construir a AGDM (algoritmo 12).

A complexidade do algoritmo é dada pela complexidade da construção da triangulação ($O(|V|^2)$) adicionada à complexidade do algoritmo de Kruskal $O(|E| \log |V|)$ [Man89], sendo $|E| = O(|V|)$, a complexidade total é $O(|V|^2)$.

Note que existem algoritmos [dBvKOS97, PS85] com complexidade $O(|V| \log |V|)$ para construção da triangulação de Delaunay, logo, se substituirmos o algoritmo para construção da mesma por um com complexidade $O(|V| \log |V|)$, a complexidade para se de-

Algorithm 12 Construir AGDM

1: Construir triangulação de Delaunay T (algoritmo 10)
2: Aplicar algoritmo de Kruskal a T([Man89])

terminar a $AGDM$ passa a ser $O(|V| \log |V|)$.

5.5 Maior Círculo Vazio de Sítios com Centro Próprio

Dado um conjunto de sítios S em $\mathbb{R}^2$, o objetivo deste problema é determinar o maior círculo vazio de sítios determinado por sítios de S [PS85].

Não é difícil provar que o centro do maior círculo vazio de sítios é um vértice de Voronoi ou um ponto de intersecção do diagrama de Voronoi com o casco convexo dos pontos de S [PS85].

Daqui por diante, discutiremos este problema em $\mathbb{T}^2$ e apresentaremos um algoritmo que foi implementado para resolvê-lo.

Ao se considerar este problema em $\mathbb{T}^2$, é necessário introduzir a restrição de que tanto o centro do círculo quanto os pontos em sua fronteira sejam próprios, já que, como visto no capítulo 4, círculos com centro impróprio ou com pontos impróprios em sua borda nem sempre estão bem definidos.

Observe na figura 5.27 que à medida que o raio do círculo que passa pelos sítios a e b cresce, a borda do mesmo se aproxima de Ω e, consequentemente, temos um círculo vazio de sítios cada vez maior, porém não é possível determinar *o* maior deles com centro próprio.

Com o objetivo de evitar as dificuldades mencionadas, para considerar este problema em $\mathbb{T}^2$, o definimos como sendo o maior círculo vazio de sítios C com centro próprio que satisfaça as seguintes propriedades:

1. se o casco convexo dos sítios de S estiver definido, o centro de C está no hemisfério em que o casco está definido; e

2. C circunscreve três sítios ou

3. C circunscreve dois sítios que constituem seu diâmetro.

Quando o casco convexo está sempre definido, temos, em $\mathbb{T}^2$, uma situação semelhante a que ocorre em $\mathbb{R}^2$, sendo pontos candidatos a centro do maior círculo vazio de sítios, os vértices de Voronoi ou pontos de intersecção do diagrama de Voronoi com o casco convexo [PS85].

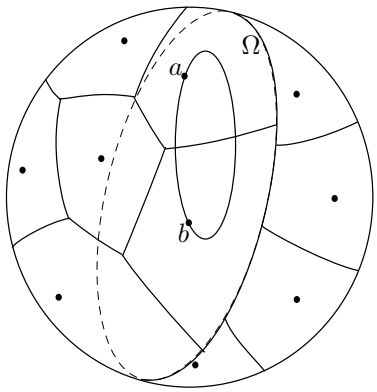


Figura 5.27: Os pontos candidatos a centro do maior círculo vazio de sítios

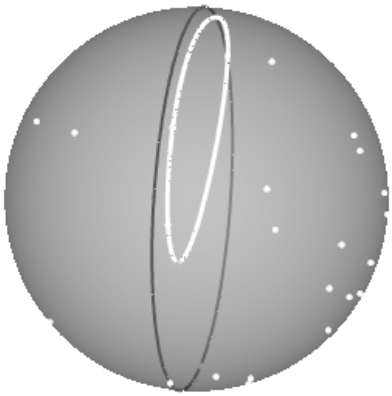


Figura 5.28: Maior Círculo Vazio de Sítios com Centro Próprio

Já quando o casco convexo não está bem definido, os pontos candidatos a centro do maior círculo vazio de sítios são apenas os vértices de Voronoi, uma vez que as propriedades descritas acima devem ser satisfeitas.

A figura 5.28 ilustra uma instância do problema, onde os sítios se encontram nos dois hemisférios canônicos de $\mathbb{T}^2$.

Um algoritmo para determinação do maior círculo vazio de sítios em $\mathbb{T}^2$ possui como primeiro passo a construção do diagrama de Voronoi. Se o casco convexo dos sítios está definido, calculam-se os pontos de interseção do diagrama de Voronoi com o mesmo. Em seguida, estando o casco definido ou não, examina-se cada um dos círculos com centro nos pontos candidatos e determina-se o maior círculo vazio de sítios (algoritmo 13).

Seja n o número de sítios do conjunto S. Este problema foi resolvido a partir da construção do diagrama de Voronoi, e a complexidade do algoritmo de CGAL estendido para construí-lo é $O(n^2)$. Como o número de vértices de Voronoi é linear no número de sítios [PS85] e como os pontos de intersecção do diagrama com o casco podem ser encontrados em tempo linear no número de sítios [PS85], o algoritmo utilizado para determinação do maior círculo vazio de sítios possui complexidade $O(n^2)$.

Se substituirmos o algoritmo para construção do diagrama de Voronoi por um algoritmo com complexidade $O(n \log n)$, a complexidade para se determinar o maior círculo vazio de sítios passa a ser $O(n \log n)$.

Algorithm 13 Maior círculo vazio de pontos com centro próprio

1: Construir diagrama de Voronoi V (algoritmo 11)
2: $m \leftarrow$ circulo$(v, 0)$ {m armazenará o maior círculo vazio de pontos. circulo$(v, 0)$ indica um círculo com centro em um vértice de Voronoi v e raio nulo.}
3: **for all** sítio $s \in V$ **do**
4: seja c o círculo com centro num vértice de Voronoi v e que passa pelos sítios que geram v.
5: **if** raio de $c >$ raio de m **then**
6: $m \leftarrow c$
7: **if** casco convexo estiver definido **then**
8: **for all** aresta e_j do casco **do**
9: seja c o círculo de diâmetro determinado pelos extremos de e_j
10: **if** raio de $c >$ raio de m **then**
11: $m \leftarrow c$
12: Retornar m

5.6 Grafo de Todos os Vizinhos mais Próximos

Dado um conjunto de n sítios S em $\mathbb{R}^2$, o problema da construção do Grafo de Todos os Vizinhos mais Próximos consiste em determinar o vizinho p_j de cada sítio $p_i \in S$, tal que a distância em $\mathbb{T}^2$ de p_i a p_j seja mínima dentre as distâncias de p_i a todos os demais sítios de S [PS85].

Considere o diagrama de Voronoi de S e seja $V(p_i)$ a região de Voronoi associada a p_i e $V(p_j)$ a região de Voronoi associada ao vizinho mais próximo de p_i. Como demonstrado em [PS85], $V(p_i)$ e $V(p_j)$ são vizinhas e, conseqüentemente, para determinar p_j é preciso apenas verificar entre as regiões vizinhas de $V(p_i)$, qual sítio de S associado às mesmas é o mais próximo a p_i (figuras 5.29 e 5.30).

As definições deste problema em $\mathbb{R}^2$, assim como em $\mathbb{T}^2$, são similares, de forma que um mesmo algoritmo pode ser aplicado em ambos espaços.

Um algoritmo para resolver o problema da construção do Grafo de Todos os Vizinhos mais Próximos em $\mathbb{R}^2$, o qual foi estendido para $\mathbb{T}^2$, consiste na construção do diagrama de Voronoi e, em seguida, para cada p_i determinar, entre seus vizinhos, o mais próximo (algoritmo 14).

Quando pelo menos um dos sítios de Voronoi é impróprio, utilizamos a medida do ângulo a partir da origem de $\mathbb{R}^3$ para determinar qual é o vizinho mais próximo, mais detalhes na seção 4.3.1.

Como o número de regiões de Voronoi é linear no número de sítios [PS85], após o diagrama de Voronoi ser construído, todos os vizinhos mais próximos podem ser encontrados em tempo linear no número de sítios. Como a complexidade do algoritmo para construir

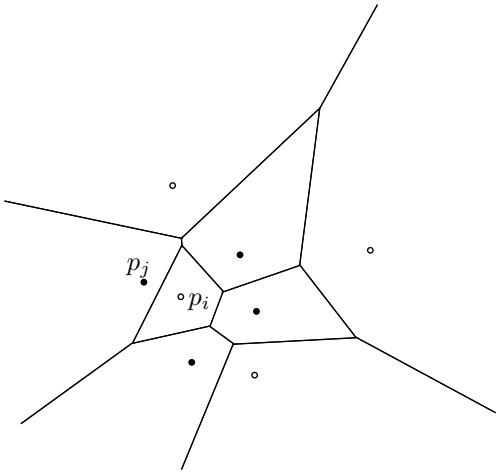


Figura 5.29: Os sítios associados às regiões vizinhas de p_i estão representados por pontos cheios

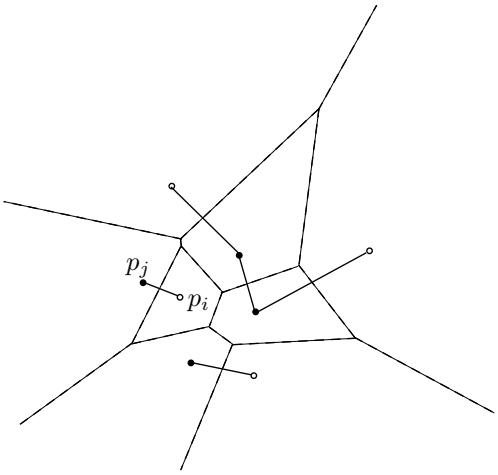


Figura 5.30: O grafo de vizinhos mais próximos

Algorithm 14 Construir o grafo de todos os vizinhos mais próximos G

1: Construir diagrama de Voronoi V (algoritmo 11)
2: **for all** sítio $p_i \in V$ **do**
3: $\quad d \leftarrow \infty$
4: $\quad$ **for all** sítio p_j vizinho de $p_i \in V$ **do**
5: $\quad\quad$ **if** $d_{\mathbb{T}^2}(p_j, p_i) < d$ **then**
6: $\quad\quad\quad v_i = p_j$ $\{v_i$ denota o vizinho mais próximo de $p_i\}$
7: $\quad\quad\quad d \leftarrow d_{\mathbb{T}^2}(p_j, p_i)$
8: $\quad$ Inserir em G a aresta $p_i v_i$
9: Retornar G

Figura 5.31: Grafo de todos vizinhos mais próximos

o diagrama de Voronoi em $\mathbb{T}^2$ é $O(n^2)$ [PS85], a complexidade do algoritmo que implementamos em CGAL usado para construir o Grafo de Todos os Vizinhos mais Próximos é também $O(n^2)$, onde n é o número de sítios de S.

Se substituirmos o algoritmo para construção do diagrama de Voronoi por um algoritmo com complexidade $O(n \log n)$, a complexidade para se construir o Grafo de Todos os Vizinhos mais Próximos passa a ser $O(n \log n)$.

5.7 Localização de Pontos em Subdivisão Planar

Uma subdivisão (ou mapa) planar é uma partição do plano em regiões bidimensionais conexas disjuntas, denominadas faces. A fronteira de cada face é formada por uma seqüência de pontos (vértices) e arcos de curvas (arestas) que se alternam em torno da face. Além disso, a união das fronteiras de todas as faces deve ser conexa [1]. Na verdade, a mesma definição se aplica para uma subdivisão de $\mathbb{T}^2$.

Uma triangulação, assim como um diagrama de Voronoi ou um diagrama de vizinho mais distante são exemplos de subdivisões planares que satisfazem propriedades especiais.

[1] uma definição mais formal aparece na definição 5.2 de [dRS94]

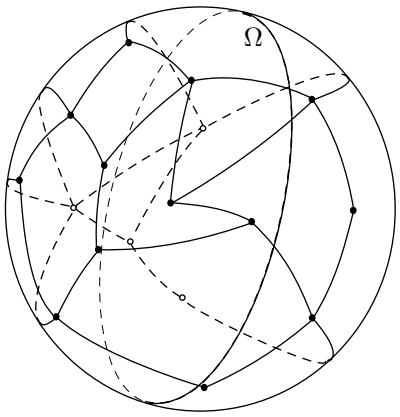


Figura 5.32: Subdivisão Planar particionando os dois hemisférios canônicos de $\mathbb{T}^2$

Neste trabalho, estendemos para $\mathbb{T}^2$ um algoritmo para localizar pontos em mapas poligonais, que são mapas onde as arestas são segmentos de reta [dRS94].

Um algoritmo para localização de um ponto p numa subdivisão em $\mathbb{T}^2$ consiste em escolher um ponto q da subdivisão e caminhar ao longo do segmento $\overline{qp}$, saindo de q na direção de p. As possíveis localidades de p são no interior de uma face, sobre uma aresta ou sobre um vértice (algoritmo 15).

Sejam v, e e f os números de vértices, arestas e faces de uma subdivisão, respectivamente. Em $\mathbb{R}^2$ ou $\mathbb{T}^2$, estes três parâmetros estão relacionados pela fórmula de Euler [Man89]:

$$v - e + f = 2$$

Como observado na fórmula de Euler, o número de faces f da subdivisão é linear em e e em v. No pior caso, o número de faces que o segmento $\overline{qp}$ atravessa é linear em f, e como a complexidade total para determinar se um ponto está no interior de cada face, sobre uma aresta ou sobre um vértice de cada face é linear no pior caso; a complexidade do algoritmo de CGAL que estendemos para $\mathbb{T}^2$ é linear em f, em e e em v.

5.8 Intersecção de Segmentos

Dado um conjunto S de n segmentos no plano, deseja-se determinar os pontos de intersecção entre os mesmos. Nesta seção, apresentaremos algoritmos para resolver o problema

Algorithm 15 Localizar ponto em subdivisão planar S

```
Tomar uma face f de S
Tomar um vértice q de f
s ← qp
for all aresta e_i de f do
  if s ∩ e_i é um segmento r then
    if p ∈ r then
      if p ≠ origem de r e p ≠ destino de r then
        Retornar p está no interior da aresta e_i de f
      else
        Retornar p é um extremo de e_i de f
    else
      q ← extremo de r mais próximo a p
      for all face f_j incidente a q do
        if qp intercepta uma aresta de f_j ou entra no interior de f_j na direção de p
        then
          f ← f_j
  else if s ∩ e_i é um ponto a no interior da aresta e_i then
    if a = p then
      Retornar p está sobre a aresta e_i
    else
      f ← face f_k vizinha de f pela aresta e_i
  else if s ∩ e_i é um extremo a de e_i then
    if a = p then
      Retornar p está sobre um vértice da aresta e_i
    else
      q ← a
      for all face f_j incidente a q do
        if qp intercepta uma aresta de f_j ou entra no interior de f_j na direção de p
        then
          f ← f_j
  else if s ∩ e_i = ∅ then
    Retornar p está no interior da face f
```

em $\mathbb{R}^2$ e em $\mathbb{T}^2$, ilustrando os casos especiais que devem ser tratados nos dois espaços.

Um algoritmo trivial para resolver este problema tanto em $\mathbb{T}^2$ como em $\mathbb{R}^2$ consiste em verificar a existência e calcular a intersecção entre cada dois segmentos de S e, desta forma, sua complexidade é $O(n^2)$.

Um algoritmo para $\mathbb{R}^2$, mais eficiente, proposto por Bentley-Ottman [BO79], consiste de uma varredura planar com uma reta vertical (denominada reta de varredura) que inicia-se em $x = -\infty$ e estende-se até $x = +\infty$. Os pontos de evento da varredura são os extremos dos segmentos de S e os pontos de intersecção entre os mesmos. Logo, o primeiro passo do algoritmo é ordenar os extremos dos segmentos de S lexicograficamente na ordem (x, y). Se o evento é o ponto origem de um segmento, tal segmento é inserido na estrutura que mantém o *status* da varredura. Por outro lado, se o evento é o ponto destino de um segmento, então ele é removido da estrutura de *status* da varredura. Quando o evento é um ponto de intersecção entre os segmentos s_0 e s_1, a ordem dos mesmos na reta de varredura é invertida. Logo, a estrutura de *status* da varredura mantém os segmentos que interceptam a reta de varredura, chamados de segmentos ativos no evento corrente.

Ao se desenvolver este algoritmo, observou-se que a detecção da interseção entre dois segmentos pode ser feita no momento em que eles sejam consecutivos ao longo da reta de varredura. Esta propriedade possibilitou que esse algoritmo fosse mais eficiente que o algoritmo trivial apresentado anteriormente.

Dois segmentos se tornam consecutivos na reta de varredura em 3 situações:

1. quando um segmento é adicionado à estrutura de *status* (o evento é o ponto origem de mesmo - figura 5.33) ou

2. quando um segmento é removido da estrutura de *status* (o evento é o ponto destino do segmento - figura 5.34).

3. quando um ponto de interseção é processado (figura 5.35).

Na situação 1 (figura 5.33), s_0 e s_1 deixam de ser consecutivos e passam a ser consecutivos a s. Logo, é necessário apenas calcular a intersecção entre s e s_0 e entre s e s_1. Na situação 2 (figura 5.34), por sua vez, ocorre o inverso s e s_0, assim como s e s_1, deixam de estar consecutivos e s_0 e s_1 se tornam consecutivos. Conseqüentemente, é necessário determinar apenas a intersecção de s_0 e s_1. Na situação 3, a ordem de s_0 e s_1 na reta de varredura é invertida, logo, o outro segmento consecutivo a s_0 (diferente de s_1) e o outro segmento consecutivo a s_1 (diferente de s_0) passam a ser consecutivos a s_1 e s_0, respectivamente (figura 5.35). Se algum ponto de interseção é encontrado durante as situações 1 ou 2 ou 3, o mesmo é inserido na fila de eventos.

Em $\mathbb{T}^2$, desenvolver algoritmos por varredura é mais complicado do que em $\mathbb{R}^2$, pois requer um certo cuidado com a identificação do ponto de parada da varredura. Imagine

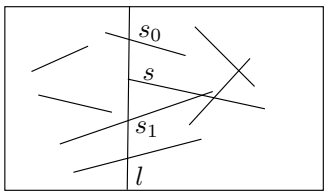


Figura 5.33: *Status* de l quando o evento é o ponto origem de s

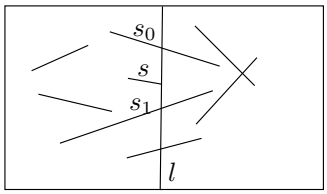


Figura 5.34: *Status* de l quando o evento é o ponto destino de s

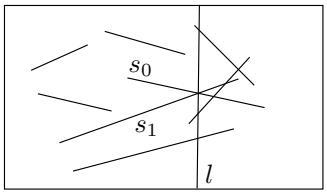


Figura 5.35: *Status* de l quando o evento é o ponto de interseção entre s_0 e s_1

uma meia reta l com um dos seus extremos fixados na origem do aquém, denotada por o, e o outro fixado em $\neg o$ (figura 5.36). Iremos varrer $\mathbb{T}^2$ circularmente com l, mas não se sabe quantas voltas ao redor de S^2 são necessárias para se resolver este problema. Considere a instância do problema ilustrada na figura 5.37. Se iniciarmos a varredura a partir da posição inicial indicada na figura 5.37 e se for dada apenas uma volta em $\mathbb{T}^2$, a interseção dos segmentos s_0 e s_1 não será detectada.

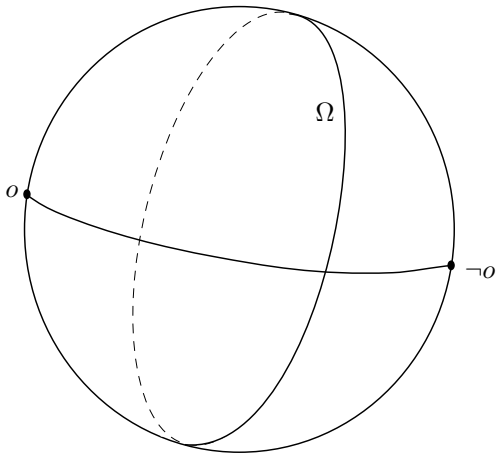


Figura 5.36: Posição inicial da meia reta de varredura

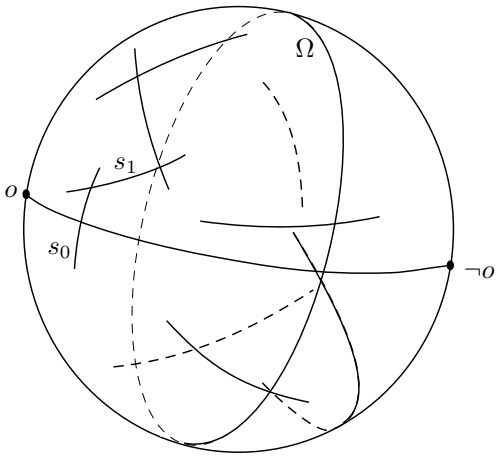


Figura 5.37: A intersecção de s_0 e s_1 não é detectada

A dificuldade surge quando um segmento s_0 intercepta l em sua posição inicial, pois a varredura é iniciada no meio de s_0 e seu ponto origem nem foi processado ainda (fi-

gura 5.37).

A solução encontrada para contornar essa situação foi particionar cada segmento s_0, que intercepta l em sua posição inicial, em dois novos segmentos s_i e s_j, sendo $s_i = \overline{uw}$, onde u é o ponto origem de s_0 e w é o ponto de interseção de s_0 e l e $s_j = \overline{wv}$, onde v é o ponto destino de s_0.

Foi necessário introduzir um tratamento especial para evitar que intersecções entre segmentos particionados fossem reportadas.

Note que, o algoritmo utilizado (algoritmo 16) trata de forma homogênea as intersecções próprias e impróprias (segmentos no aquém, além ou infinito).

Em $\mathbb{T}^2$, os pontos são ordenados circularmente em torno da origem, em sentido anti-horário e a idéia principal do algoritmo em $\mathbb{R}^2$ é mantida. A figura 5.37 ilustra a varredura circular no modelo esférico de $\mathbb{T}^2$.

Existem alguns casos especiais inerentes ao problema - como por exemplo, intersecções que envolvem segmentos contidos na reta de varredura - logo, os mesmos surgem tanto em $\mathbb{R}^2$ quanto em $\mathbb{T}^2$.

Ao estender-se esse algoritmo, pode-se perceber a dificuldade de se projetar algoritmos por varredura em $\mathbb{T}^2$, sendo importante ressaltar esse fato, pois vários algoritmos geométricos eficientes utilizam essa técnica.

Este algoritmo utiliza duas estruturas de dados: uma para armazenar os pontos de eventos e outra para manter o *status* da varredura. Para que a complexidade do algoritmo não se torne $O(n^2)$, é preciso que a complexidade de inserção, remoção e busca na estrutura que mantém o *status* da varredura não seja linear. Na implementação, a estrutura de *status* é uma árvore binária de busca balanceada, onde a complexidade dessas operações é $O(\log n)$.

Em suma, temos três tipos de eventos:

1. ponto origem do segmento s: o tratamento desse tipo de evento requer a inserção de s na estrutura de *status* ($O(\log n)$), a identificação dos segmentos consecutivos a s ($O(1)$), verificação e cálculo das intersecções ($O(1)$) e se houver intersecção, inserção do ponto na fila de eventos ($O(\log n)$). Logo, a complexidade para tratar esse tipo de evento é $O(\log n)$.

2. ponto destino do segmento s: o mesmo é removido da estrutura de *status* ($O(\log n)$), em seguida, identificam-se os novos segmentos consecutivos s_0 e s_1 ($O(1)$), calcula-se a intersecção destes segmentos ($O(1)$) e, se houver intersecção, insere-se-a na fila de eventos ($O(\log n)$). Analogamente, a complexidade para tratar esse tipo de evento é $O(\log n)$.

3. Quando o evento a ser processado é o ponto de intersecção entre dois segmentos s_0 e s_1, identificam-se os dois novos segmentos consecutivos ($O(1)$) e calculam-se

Algorithm 16 Determinar interseção de segmentos por varredura

1: Particionar os segmentos de S que interceptam a meia-reta de varredura em sua posição inicial
2: Inserir os pontos de S na fila F
3: Ordenar os pontos em F circularmente em torno da origem de $\mathbb{T}^2$, em sentido anti-horário
4: **while** $F \neq \emptyset$ **do**
5: Retirar o ponto de evento e de F
6: **if** e é ponto de origem de um segmento s **then**
7: Inserir s na estrutura de status da varredura R
8: Sejam s_0 e s_1 os segmentos consecutivos a s na reta de varredura
9: **if** s intercepta s_0 **then**
10: Inserir em F o ponto de intersecção e reordenar F
11: **if** s intercepta s_1 **then**
12: Inserir em F o ponto de intersecção e reordenar F
13: **else if** e é ponto de destino de um segmento s **then**
14: Retirar s de R
15: Sejam s_0 e s_1 seus vizinhos em R
16: **if** s_0 intercepta s_1 **then**
17: Inserir os pontos de intersecção em F e reordenar F
18: **else**
19: {e é ponto de intersecção de s_0 e s_1}
20: Inverter em R a posição de s_0 e s_1
21: Reportar e
22: Calcular a intersecção de s_0 e s_i e de s_1 e s_j,{onde s_i e s_j são os novos segmentos consecutivos a s_0 e s_1, respectivamente}
23: Inserir os pontos de intersecção, caso houver, em F e reordenar F

a interseção de s_0 e s_1 com seus respectivos novos segmentos consecutivos ($O(1)$) e se houver interseções inserem-se-as na fila de eventos ($O(\log n)$). Portanto, a complexidade para processar esse tipo de evento é $O(\log n)$.

Como o número de eventos do tipo ponto origem e ponto destino é $O(n)$, a complexidade para tratá-los é $O(n \log n)$. Seja k o número de pontos de intersecção. Como a complexidade para processar cada ponto de intersecção é $O(\log n)$, a complexidade para tratar todos os eventos desse tipo é $O(k \log n)$. Logo, a complexidade do algoritmo por varredura para determinar interseção entre segmentos é $O((n+k) \log n)$, sendo que k pode ser até $O(n^2)$.

Foi estendido para $\mathbb{T}^2$ um algoritmo geral de CGAL para determinar a intersecção entre n curvas planares e, para prover essa generalidade, o tipo de curvas planares é passado para o algoritmo como parâmetro. Assim, o usuário pode estender a funcionalidade proporcionada pela biblioteca implementando outros tipos de curva e reutilizando o algoritmo geral desenvolvido. Os tipos de curvas para as quais o algoritmo de Bentley-Ottman [BO79] foi estendido para $\mathbb{T}^2$ foram segmentos e linhas poligonais.

5.9 Busca em Amplitude com Iso-retângulo - Problema de Contagem

Dados n pontos no plano, este problema consiste em determinar o número de pontos internos a um iso-retângulo dado [PS85].

Em aplicações, buscas geométricas sobre os mesmos dados podem surgir esporadicamente ou repetidas vezes. No primeiro caso (poucas consultas), considerando o problema descrito, pode-se utilizar um algoritmo ingênuo com complexidade linear no número de pontos dados, pois basta examinar se cada ponto está no interior do iso-retângulo. Já no segundo, sendo m o número de consultas, a complexidade da busca deste algoritmo se tornaria $O(mn)$. Neste caso, seria interessante organizar os dados em uma estrutura que facilitasse a busca e, com essa abordagem, seria necessária uma etapa de pré-processamento para organizar os dados na estrutura, aumentando a complexidade de espaço e a complexidade de tempo de pré-processamento com o objetivo de reduzir o tempo de consulta.

Parece complexo encontrar uma forma de organizar os pontos de modo que, consultas com retângulos arbitrários sejam efetuadas de forma mais eficiente. Por outro lado, é impossível resolver o problema para todos os retângulos, pois existe um número infinito deles. Sendo assim, foi utilizada a técnica *locus method*, onde o espaço de busca é subdividido (particionado) em regiões (*loci*), dentro das quais a resposta para consultas é invariante.

Um algoritmo para $\mathbb{R}^2$ trabalha com os quatro pontos do iso-retângulo, sendo o problema subdividido em quatro subproblemas que são combinados para produzir a resposta final. Em cada subproblema, seja p um vértice do iso-retângulo de consulta, o objetivo é determinar o número de pontos $Q(p)$ do conjunto de pontos S ao sul e a oeste de p (figura 5.38), também denominados de pontos dominados por p [PS85].

Sejam p_1, p_2, p_3 e p_4 os extremos de um iso-retângulo (figura 5.39), o número $N(p_1p_2p_3p_4)$ de pontos contidos no retângulo $p_1p_2p_3p_4$ é dado por:

$$N(p_1p_2p_3p_4) = Q(p_1) - Q(p_2) - Q(p_4) + Q(p_3) \tag{5.1}$$

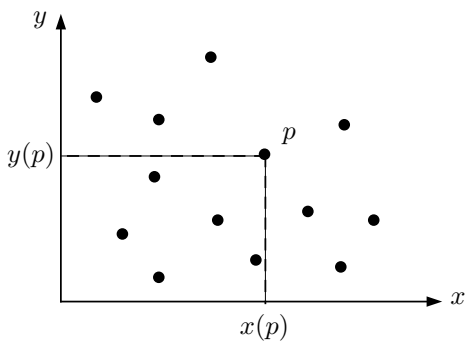


Figura 5.38: Pontos dominados por p

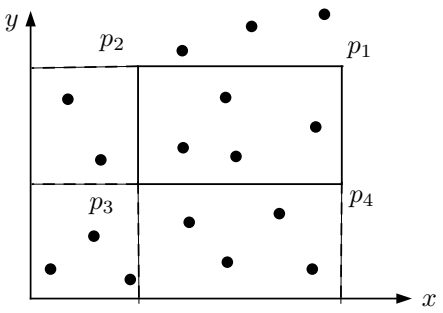


Figura 5.39: Pontos no interior do retângulo $p_1p_2p_3p_4$

O pré-processamento consiste na subdivisão do espaço de busca em regiões que produzem respostas equivalentes e, assim, tais regiões são denominadas classes de equivalência.

O primeiro passo do pré-processamento é a construção de uma malha de ordem $(n+1) \times (n+1)$, onde os pontos nas linhas estão ordenados em x e os pontos nas colunas estão ordenados em y (figura 5.40). São mantidas duas estruturas A e B com os pontos de S: A com os pontos ordenados em x e B com os pontos ordenados em y e, desta forma, para determinar os pontos da malha, basta combinar as coordenadas x e y em ordem crescente. Para cada ponto $p \in S$, traça-se uma vertical e uma horizontal, o que resulta em uma malha com $O(n^2)$ retângulos. Em seguida, determina-se o número de pontos dominados - $Q(p)$ - por cada ponto superior direito p de cada um dos retângulos da malha(figura 5.38). Para armazenar $Q(p)$ é utilizada uma matriz M de ordem $(n+1) \times (n+1)$, a qual é preenchida com zeros inicialmente (figura 5.40). A seguir, para cada um dos $O(n^2)$ retângulos r_{ij} da malha, se seu canto superior direito $p_{ij} \in S$, então $M_{(i+1)(j+1)} = 1$ (figura 5.41).

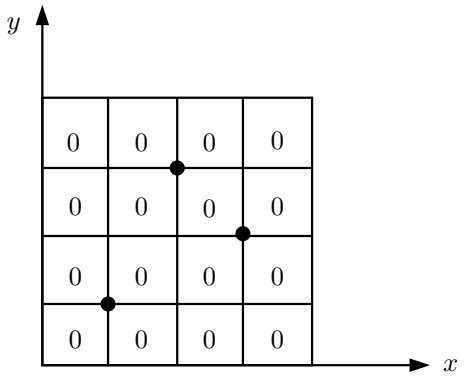


Figura 5.40: Malha inicializada com zeros

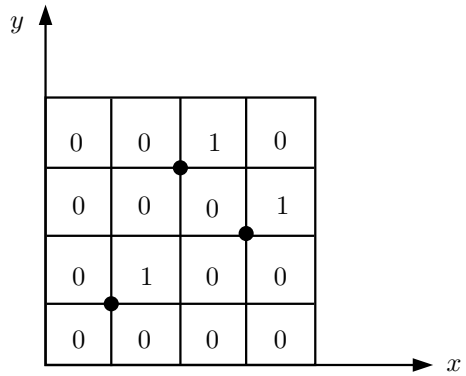


Figura 5.41: Identificação de pontos da malha que fazem parte da entrada

No próximo passo, por meio de programação dinâmica, calcula-se $Q(p_{ij})$ para cada p_{ij} (figura 5.42) (da esquerda para direita, de baixo para cima), utilizando-se a fórmula:

$$Q(p_{ij}) = Q(p_{ij}) + Q(p_{i(j-1)}) + Q(p_{(i-1)j}) - Q(p_{(i-1)(j-1)})$$

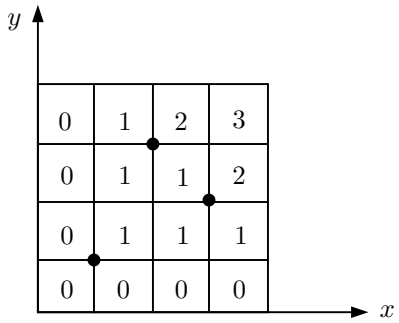


Figura 5.42: Estrutura construída durante o pré-processamento para otimizar a busca

Na consulta, o objetivo é determinar o maior retângulo s, cujos extremos são pontos da malha resultante do pré-processamento, contido no interior ou coincidente ao retângulo de consulta r, de forma que, qualquer ponto em S no interior de r também esteja no

interior de s, portanto, $N(r) = N(s)$. Para isso, a consulta é realizada através de quatro buscas binárias:

- Duas na estrutura A para determinar a menor e a maior abcissa de s.
- Outras duas na estrutura B para determinar a menor e a maior ordenada de s.

Conseqüentemente, os quatro extremos de s são determinados facilmente e $N(s)$ é calculado pela fórmula 5.1.

Como mencionado, um retângulo com extremos nos dois hemisférios canônicos de $\mathbb{T}^2$ não está definido, ou seja, lados paralelos de um retângulo r podem se estender até o infinito (Ω) numa dada direção, mas não pode ultrapassá-lo.

Sendo r um iso-retângulo, as possíveis direções infinitas são $[0, 0, 1]$, $[0, 0, -1]$, $[0, 1, 0]$ ou $[0, -1, 0]$. Pode-se afirmar que nenhum ponto $p \in \Omega$ está no interior de r, pois quando r é infinito, p faz, no máximo, parte da fronteira de r. Logo, pontos em Ω podem ser descartados durante o pré-processamento e, a solução do problema para $\mathbb{T}^2$ e $\mathbb{R}^2$ é a mesma, sendo necessário fazer o pré-processamento para os pontos nos dois hemisférios de $\mathbb{T}^2$ separadamente. Sendo assim, um retângulo utiliza apenas a malha correspondente ao hemisfério de $\mathbb{T}^2$ no qual ele está contido para realizar as consultas.

A seguir, apresentamos o algoritmo de pré-processamento para cada hemisfério canônico de $\mathbb{T}^2$ (algoritmo 17) e o algoritmo para consulta (algoritmo 18).

Algorithm 17 Pré-processamento

1: Criar a matriz M de ordem $(n+1) \times (n+1)$
2: Inicializar todos os elementos de M com zero
3: Ordenar em x os pontos de S e inserí-los na estrutura A
4: Ordenar em y os pontos de S e inserí-los na estrutura B
5: **for all** ponto $p \in A$ **do**
6: **for all** ponto $q \in B$ **do**
7: **if** ponto $(p.x, q.y) \in S$ {$p.x$ e $q.y$ são as coordenadas x e y de p e de q, respectivamente} **then**
8: $M_{(i+1)(j+1)} \leftarrow 1$
9: **for** $i \leftarrow 1$ to n **do**
10: **for** $j \leftarrow 1$ to n **do**
11: $Q(p_{ij}) \leftarrow Q(p_{ij}) + Q(p_{i(j-1)}) + Q(p_{(i-1)j}) - Q(p_{(i-1)(j-1)})$

O algoritmo descrito deve ser analisado separando-se a complexidade de pré-processamento, de consulta e de espaço. Durante o pré-processamento, ordenam-se os pontos de S em x e em y ($O(n \log n)$) e constrói-se a malha, onde o número de pontos dominados por cada ponto é determinado em tempo constante, logo, como existem $O(n^2)$ pontos na

Algorithm 18 Consulta com o retângulo r

1: $x_{min} \leftarrow$ Busca_Binaria$(A, r.x_{min})$
2: $x_{max} \leftarrow$ Busca_Binaria$(A, r.x_{max})$
3: $y_{min} \leftarrow$ Busca_Binaria$(A, r.y_{min})$
4: $y_{max} \leftarrow$ Busca_Binaria$(A, r.y_{max})$
5: $N(r) \leftarrow M(x_{max}, y_{max}) - M(x_{min}, y_{max}) - M(x_{max}, y_{min}) + M(x_{min}, y_{min})$
6: Retornar $N(r)$

malha, a complexidade de pré-processamento é $O(n^2)$. A complexidade da consulta, por sua vez, diante da organização dos dados passa a ser $O(\log n)$, devido às buscas binárias. E finalmente, a complexidade de espaço é $O(n^2)$, em razão da armazenagem da matriz.

5.10 Comparação entre os algoritmos em $\mathbb{R}^2$ e sua extensão para $\mathbb{T}^2$

A tabela 5.10 ilustra, em termos de casos tratados e dificuldades,os pontos positivos e/ou negativos de se estender um algoritmo de $\mathbb{R}^2$ para $\mathbb{T}^2$.

Algoritmo	$\mathbb{R}^2$	$\mathbb{T}^2$
Construção do casco convexo		**Ponto negativo** – É preciso verificar a existência.
Construção de Triangulação (Arbitrária ou de Delaunay)	**Ponto negativo** – Tratamento especial para que todas as faces sejam triangulares.	**Ponto negativo** – Tratamento especial para faces degeneradas que surgem devido à topologia de$\mathbb{T}^2$
Construção do diagrama de Voronoi		**Ponto positivo** – Sítios próprios num hemisfério canônico, temos diagrama de vizinho mais distante no outro.
Árvore geradora de distância mínima	**Ponto negativo** – Distâncias finitas e infinitas não são representáveis de forma homogênea.	**Pontos positivos** – Representação homogênea de distâncias finitas e infinitas. Possibilidade de comparar relativamente distâncias infinitas.
Maior círculo vazio de pontos com centro próprio	Mesma dificuldade.	Mesma dificuldade.

Grafo de todos os vizinhos mais próximos	**Ponto negativo** – Distâncias finitas e infinitas não são representáveis de forma homogênea.	**Ponto positivo** – Representação homogênea de distâncias finitas e infinitas. Possibilidade de comparar relativamente distâncias infinitas.
Localização de ponto em subdivisão planar	Mesma dificuldade.	Mesma dificuldade.
Interseção de segmentos		**Ponto positivo** – Intersecções no aquém, no além e infinito são calculadas de forma homogênea. **Ponto negativo** – Dificuldades na definição do ponto de parada da varredura.
Busca por amplitude	Mesma dificuldade.	Mesma dificuldade.

Tabela 5.1: Algoritmos em $\mathbb{R}^2$ x em $\mathbb{T}^2$

Capítulo 6

Conclusão

Este trabalho envolveu a extensão para $\mathbb{T}^2$ de vários algoritmos geométricos em $\mathbb{R}^2$ da biblioteca CGAL, assim como sua implementação nos moldes da biblioteca. Inicialmente, foi feito o trabalho de base, ou seja, a extensão e implementação de um conjunto de primitivas, predicados e construções geométricas básicas imprescindível ao desenvolvimento dos algoritmos, o qual denominamos núcleo estendido de CGAL.

Neste capítulo, sintetizamos e destacamos as principais contribuições deste trabalho. Iremos discutir inicialmente os resultados obtidos com a extensão dos algoritmos e, posteriormente, destacamos as principais contribuições provenientes da extensão do núcleo de CGAL.

Dentre os problemas estudados, verificou-se que vários deles apresentam solução mais homogênea em $\mathbb{T}^2$ e alguns outros, em razão de características de $\mathbb{T}^2$, requerem o tratamento de casos especiais.

Devido à antipossidade de $\mathbb{T}^2$, para se construir o casco convexo é necessária uma etapa de pré-processamento para determinar se o mesmo está definido. Similarmente, casos degenerados, provenientes da antipossidade do espaço, podem surgir na triangulação. Quando um ponto antípoda a um vértice do casco convexo é inserido na triangulação, surgem faces degeneradas com três pontos colineares e é necessário determinar uma direção para o segmento antípoda criado. Nesses casos, foi necessário inserir tratamentos especiais nos algoritmos de CGAL estendidos para se construir casco convexo e triangulação.

Necessidade de tratamento de casos especiais também aparece quando o algoritmo envolve predicados para comparação de distâncias. Como distâncias entre pontos impróprios ou entre um ponto impróprio e um ponto próprio não está definida, é necessário utilizar uma medida alternativa consistente para compará-las. Esse foi o caso dos algoritmos para construir a árvore geradora de distância mínima, o grafo de todos os vizinhos mais próximos e a triangulação de Delaunay implementados em CGAL.

Algoritmos envolvendo retângulos e circunferências sofrem restrições quando ambas

primitivas devem ser tratadas sobre Ω. Como vimos, ao problema do maior círculo vazio de pontos foi imposta a condição de que seu centro fosse próprio e, no caso de retângulos, na busca por amplitude com retângulos, pontos impróprios podem ser descartados no pré-processamento, uma vez que retângulos não possuem pontos impróprios em seu lado limitado.

Em $\mathbb{T}^2$, ao se projetar algoritmos por varredura é preciso identificar um ponto de parada consistente, onde todo o processamento necessário já tenha finalizado. Neste trabalho, foi estendido o algoritmo de Bentley-Ottman [BO79] implementado em CGAL, para determinar a interseção entre curvas planares e, como não é possível determinar o número de voltas ao redor de S^2 necessário para se encontrar todas as intersecções, foi preciso subdividir todos os segmentos que interceptassem a reta de varredura em sua posição inicial. Sendo assim, foi introduzido um tratamento especial para impedir que intersecções entre segmentos subdividos fossem reportadas. Logo, algoritmos por varredura em $\mathbb{T}^2$, geralmente, terão que tratar essa idiossincrasia.

Embora apareçam casos degenerados na construção da triangulação em $\mathbb{T}^2$, certas propriedades de $\mathbb{T}^2$ também facilitam sua construção, uma vez que a face ilimitada externa ao casco convexo pode ser decomposta em triângulos facilmente com a criação de um ponto infinito v_∞ e arestas de cada vértice do casco até v_∞ (mais detalhes no capítulo 5). Em $\mathbb{R}^2$, essa decomposição envolve o tratamento de casos especiais, uma vez que v_∞ não possui coordenadas representáveis em $\mathbb{R}^2$.

Com a construção do diagrama de Voronoi em $\mathbb{T}^2$, quando os sítios estão todos em um lado de $\mathbb{T}^2$, obtém-se por antipossidade o diagrama de vizinho mais distante [Pin98, Sto91]. Logo, não é necessário se projetar um algoritmo para resolver o problema do diagrama de vizinho mais distante. Resolver um problema e obter, sem esforços, a solução de um problema complementar é uma vantagem notável de $\mathbb{T}^2$.

Com relação à extensão das primitivas geométricas de CGAL, as principais contribuições foram as representações obtidas para retângulos isotéticos e circunferências (vide capítulo 4) e, além disso, o estudo de seus comportamentos em $\mathbb{T}^2$, sendo importante destacar que em algumas situações os mesmos não estão bem definidos.

Dentre os predicados de CGAL estendidos, a maior contribuição foi a estratégia encontrada para se comparar relativamente a extensão de segmentos infinitos, com ambos ou apenas um extremo impróprio. Este predicado foi utilizado por alguns algoritmos estendidos, entre eles: a árvore geradora de distância mínima, a triangulação de Delaunay e o grafo de todos os vizinhos mais próximos.

6.1 Trabalhos Futuros

Nesta seção, sugerimos a extensão de alguns algoritmos geométricos, assim como sua implementação e incorporação a CGAL, sendo que a implementação será grandemente facilitada, uma vez que o núcleo estendido (um conjunto de primitivas e predicados em $\mathbb{T}^2$) já está pronto.

- **Diagrama de Voronoi de Ordem** k: seria interessante buscar problemas que apresentassem comportamento semelhante ao do diagrama de Voronoi em $\mathbb{T}^2$. Considere o diagrama de Voronoi de ordem k, o qual divide o plano em regiões, tal que cada uma está associada a uma combinação dos sítios tomados k a k, tal que para todo ponto interior a região, os k sítios mais próximos são a combinação associada à região. Uma sugestão de trabalho futuro seria implementar e incorporar a CGAL o algoritmo para construir o diagrama de Voronoi de ordem k em $\mathbb{T}^2$ apresentado em [Wes99], uma vez que, como provado em [Wes99], ao se resolver este, obtém-se por antipossidade o diagrama de Voronoi de ordem $n-k$, para $2 \leq k \leq n-2$.

- **Diagrama de Voronoi com peso**: o diagrama de Voronoi com peso é uma generalização do diagrama de Voronoi, onde a cada sítio está associado um peso. Como mencionado em [Pin98], o mesmo é sempre conexo em $\mathbb{T}^2$, ao contrário do caso euclidiano e, portanto, pode ser tratado de forma mais homogênea em $\mathbb{T}^2$.

- **Busca em Amplitude**: neste trabalho, estendemos para $\mathbb{T}^2$, o problema de contagem da busca por amplitude retangular. Uma sugestão seria estender e incorporar a CGAL problemas de enumeração, onde a busca é realizada em estruturas de dados mais sofisticadas, como *kd-trees* ou *range-trees*, pois a busca se torna muito eficiente [dBvKOS97].

- **Arranjos de retas e Dualidade**: o processo de síntese de imagens é composto por duas fases: a determinação de porções de objetos que está visível em cada *pixel* e a determinação da quantidade de luz emitida por eles. Um algoritmo bastante utilizado para síntese de imagens é o *Ray tracing*, onde são criados segmentos com origem no olho do observador e destino nos *pixels*. Um *pixel*, porém, não é um ponto, e sim uma pequena área quadrada. Logo, é preciso escolher um ponto no interior da área do *pixel*, até o qual será traçado um segmento. Para que a imagem construída seja mais realista, em vez de determinarmos um ponto, determinamos um conjunto de pontos e são traçados vários segmentos em vez de um. Conforme algoritmo descrito em [dBvKOS97], esse conjunto de pontos pode ser dualizado para um arranjo de retas, o qual é utilizado para resolver a primeira fase do processo de síntese de imagens. Em $\mathbb{T}^2$, como existe uma dualidade perfeita entre pontos e retas

(em $\mathbb{R}^2$, temos apenas uma dualidade parcial), a dualização e a manipulação do arranjo de retas podem ser feitos de forma mais homogênea. Portanto, a extensão desse algoritmo para $\mathbb{T}^2$ é bastante propícia.

6.2 Disponibilidade do Pacote Desenvolvido

O pacote desenvolvido engloba o núcleo estendido e os algoritmos descritos no capítulo 5. Este pacote é compatível com as plataformas suportadas pela biblioteca CGAL (as quais estão especificadas no endereço `http://www.cgal.org`) e para utilizá-lo, é necessário instalar a versão atual da biblioteca (versão 3.0.1). O mesmo está disponível, sob os termos da licença GPL, no endereço `http://www.ic.unicamp.br/~rezende/T2_CGAL`.

Referências Bibliográficas

[23k02] CGAL 2D and 3D Kernel Manual, May 2002. http://www.cgal.org/manual.

[bas02] CGAL Basic Library, May 2002. http://www.cgal.org/manual.

[BM00] D. Bulka and D. Mayhew. *Efficient C++: Performance Programming Techniques*. Addison Wesley, 2000.

[BO79] J. L. Bentley and T. A. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers 28*, pages 643–647, 1979.

[dBvKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 1997.

[dRG96] P. J. de Rezende and C. N. Gon. Um Ambiente Distribuído de Visualização com Suporte para Geometria Projetiva Orientada. *In Anais do IX Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens*, pages 71–78, 1996.

[dRJ93] P. J. de Rezende and W. R. Jacometti. Animation of Geometric Algorithms using GeoLab. *Proceedings of the Ninth Annual Symposium on Computational Geometry*, 1993. http://portal.acm.org/citation.cfm?id=161176\&coll=portal\&dl=ACM\&ret=1\#Fulltext).

[dRS94] P. J. de Rezende and J. Stolfi. *Fundamentos de Geometria Computacional*. UFPE-DI, IV Escola de Computação, 1994.

[Geoa] Página do Visualizador GeomView. http://www.geomview.org.

[Geob] Página do Visualizador GeoSheet. http://www.ece.northwestern.edu/\~{}dtlee/theory/gs_tech_1_html/gs.html.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of reusable object oriented software*. Addison Wesley, 1995.

[KMM+90] A. Knight, J. May, J. Mcaffer, T. Nguyen, and J. R. Sack. A Computation Geometry WorkBench. *Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkley*, 1990. `http://portal.acm.org/citation.cfm?id=98602\&coll=portal\&dl=ACM\\\&ret=1\#Fulltext`.

[LED] LEDA Overview. `http://www.algorithmic-solutions.com/enleda.htm`.

[Man89] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989.

[Pin98] G. A. Pinto. Generalizações do Diagrama de Voronoi construídas através de Cônicas no Plano Projetivo Orientado. Master's thesis, Universidade Estadual de Campinas, 1998.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York Inc., 1985.

[SDto] F. P. Selmi-Dei. Um Visualizador para uma Extensão de CGAL ao Plano Projetivo Orientado. Master's thesis, Universidade Estadual de Campinas, em andamento.

[Sto91] J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric computations*. Academic Press, Inc., 1991.

[sup02] CGAL Support Library, May 2002. `http://www.cgal.org/manual`.

[Wes99] R. B. Westrupp. Diagramas de Voronoi de Ordem k na Geometria Projetiva Orientada. Master's thesis, Universidade Estadual de Campinas, 1999.

[XYZ] Página do GeoBench. `http://www.schorn.ch/geobench/XYZGeoBench.html`.