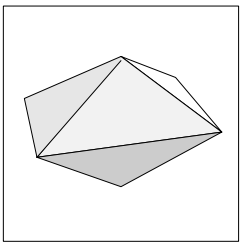
GEOMETRIA COMPUTACIONAL

FECHO CONVEXO TRIDIMENSIONAL

1. Introdução



Estudaremos agora a construção do fecho convexo de um dado conjunto de ponto S em $\mathbb{R}^3$. Veremos algumas possíveis maneiras de representarmos conv(S) (em termos de estrutura de dados). Discutiremos o método do embrulho para presente de Chand & Kapur [2] para construir o fecho convexo (este método é uma generalização do Algoritmo de Jarvis para encontrar o fecho convexo de um conjunto de pontos no plano). Veremos também um algoritmo incremental para o problema.

Começaremos tentando adquirir alguma intuição sobre poliedros (seguindo os passos feitos em O'Rourke [7]) e depois trataremos de politopos regulares e Fórmula de Euler. Da Fórmula de Euler deduziremos limites superiores para o número de arestas e faces do fecho convexo de um conjunto de n pontos em $\mathbb{R}^3$ (limites estes que são importantes para a análise de algoritmos).

Observação. Os tópicos que cobriremos foram extraídos do: capítulo 3 de [4]; capítulo 4 de [6], capítulo 3 de [7]; e capítulos 3 de [8].

2. Poliedro

Um *poliedro* é uma generalização natural de um polígono para o espaço tridimensional: é a região do espaço cuja fronteira é formada por um número finito de faces poligonais, qualquer par das quais são ou disjuntas, ou tem um vértice em comum ou tem uma aresta em comum. Esta descrição é vaga e é uma tarefa delicada descrever a classe exata desses objetos.

A definição que encontramos na página 19 do livro de Preparata & Shamos [8] é a seguinte:
Poliedro. Um poliedro em $\mathbb{R}^3$ é a região do espaço limitada por um conjunto finito de polígonos (planos) tal que cada aresta de um destes polígonos é compartilhada por exatamente um outro polígono (por um *polígono adjacente*) e nenhum subconjunto dos polígonos tem essa mesma propriedade. Os vértices e arestas do polígono são os *vértices* e *arestas* do poliedro; os polígonos são as *faces* do poliedro.

Vamos desenvolver um pouco mais a intuição sobre o que é um poliedro como é feito em O'Rourke[7]. A fronteira do poliedro é formada por três tipos de objetos: pontos (vértices) de dimensão zero; segmentos (arestas) de dimensão um; e polígonos (faces) de dimensão dois. Às vezes é conveniente exigirmos que os polígonos sejam convexos (nesse caso permitimos que duas faces sejam coplanares). Portanto, da mesma maneira que aconteceu quando

Date: 1 de dezembro de 2009.

definimos um polígono, um poliedro é dado através da relação entre os seus componentes. Em um poliedro, temos que:

(1) *Componentes se intersectam 'propriamente'.* Para cada par de faces, exigimos que:
   (a) elas sejam disjuntas, ou
   (b) elas tenham um único vértice em comum, ou
   (c) elas tenham dois vértices e a aresta os ligando em comum.

   Interseção imprópria elimina objectos em que uma face 'atravessa' a outra e também o objeto mostrado na figura 1, onde a interseçao de duas faces não é uma aresta.

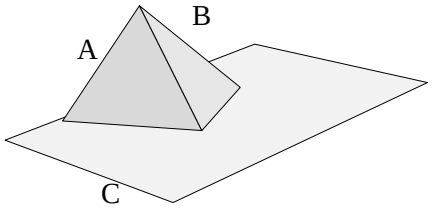


FIGURA 1. A face A e B não intersectam C propriamente.

(2) *A topologia local é 'própria'.* A topologia local é como a superfície (fronteira) do poliedro se parece na vizinhança de um ponto. Na vizinhança de cada ponto, a superfície de um poliedro deve ser indistinguível de um disco a menos de dobrar e esticar a superfície como se esta fosse feita de um material moldável (não podemos cortar a superfície). Com isto estamos excluindo os objetos da figura 2 de serem um poliedro.

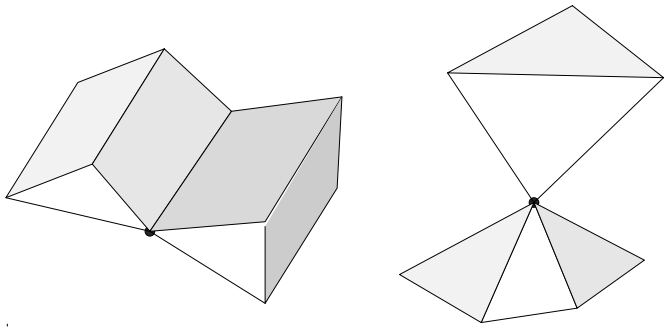


FIGURA 2. Na vizinhança dos pontos indicados a superfície não se parece com um disco.

(3) *A topologia global é 'própria'.* A superfície deve ser conexa, fechada e limitada. A superfície é conexa no sentido em que, de qualquer ponto da superfície, uma formiguinha pode andar, sem sair da superfície, até qualquer outro ponto. Combinatoriamente isto é equivalente a dizer que o *1-esqueleto* (*1-skeleton*, o grafo formado pelos vértices e arestas dos polígonos) é conexo.

Estamos admitindo aqui que um poliedro tenha buracos (como um pneu ou *donut*; veja figura 3). O número de buracos é chamado de *genus* da superfície. Poliedros de genus zero são topologicamente equivalente a uma esfera e muitas vezes são chamados de *poliedros simples*.

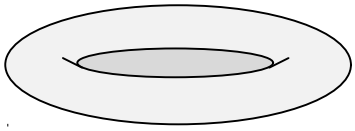


FIGURA 3. Uma superfície de genus 1; um torus.

Resumindo, a fronteira de um poliedro é uma coleção finita de polígonos planos convexos limitados (as faces do poliedro) tal que:

(1) As faces se intersectam propriamente;
(2) A vizinhança de cada ponto é topologicamente um disco aberto;
(3) A superfície é conexa, ou equivalentemente o 1-esqueleto é conexo.

A fronteira é fechada e contém no seu interior uma porção limitada do espaço. Cada aresta é compartilhada por exatamente duas faces; essas faces são ditas *adjacente*.

Um poliedro é convexo se todos os seus ângulos diedrais são convexos, ou seja, no máximo π. Um ângulo *diedral* com relação a uma aresta e de um poliedro P é o ângulo interno entre os planos formados pelas faces de P que compartilham e. Veja a figura 4. Poliedros convexos são chamados de *politopos*.

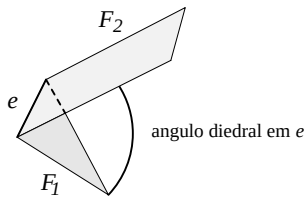


FIGURA 4. Ângulo diedral entre duas faces.

3. Politopos regulares

Para continuar a desenvolver mais a nossa intuição, vamos discutir sobre o assunto que é tratado na seção 1.2 de O'Rourke [7].

Um *polígono regular* é um que tem ângulos iguais em cada vértice e cujo comprimento de todas as aresta é o mesmo. É claro que existem infinitos polígonos regulares: triângulo equilátero, quadrado, pentágono, hexágono, heptágono, etc.

Uma extensão natural deste conceito de regularidade para o espaço tridimensional é considerarmos politopos regulares. Um politopo é *regular* se suas faces são polígonos regulares congruentes e o número de faces incidentes a cada vértice é o mesmo.

Ao contrário do que ocorre na versão bidimensional, existem somente cinco politopos regulares: os chamados *sólidos Platônicos*. Isto é conhecido desde os tempos de Platão e aparece em seus escritos *Timaeus*.

A prova é elementar (uma outra prova pode ser obtida da Fórmula de Euler que será discutida na próxima seção). A intuição é que os ângulos internos de um polígono regular aumentam com o número de vértices, mas não existe muito espaço (no máximo 2π) para colocarmos esses ângulos ao redor de um vértice do poliedro.

Seja P um poliedro regular com p vértices em cada uma de suas faces (polígonos regulares). A soma dos ângulos internos de uma face é igual a $(p-2)\pi$. Logo o ângulo interno em cada vértice da face é $(1-2/p)\pi$.

Seja r o número de arestas incidentes a cada vértice de P. Sabemos que a soma dos ângulos faciais em cada vértice de P é no máximo 2π. Logo,

$$\begin{aligned} r(1-2/p)\pi < 2\pi &\Rightarrow 1-2/p < 2/r \\ &\Rightarrow pr < 2r+2p \\ &\Rightarrow pr-2r-2p+4 < 4 \\ &\Rightarrow (p-2)(r-2) < 4. \end{aligned}$$

Como um polígono tem pelo menos três vértices, temos que $p \geq 3$. Ademais, pelo menos três faces devem ser incidentes a cada vértice do poliedro P, logo $r \geq 3$. A tabela abaixo mostra todas as soluções das desigualdades obtidas.

p	r	$(p-2)(r-2)$	Nome	Descrição
3	3	1	tetraedro	3 triângulos incidentes a cada vértice
4	3	2	cubo	3 quadrados incidentes a cada vértice
3	4	2	octaedro	4 triângulos incidentes a cada vértice
5	3	3	dodecaedro	3 pentâgonos incidentes a cada vértice
3	5	3	icosaedro	5 triângulos incidentes a cada vértice

Os polígonos resultantes são mostrados na figura 5.

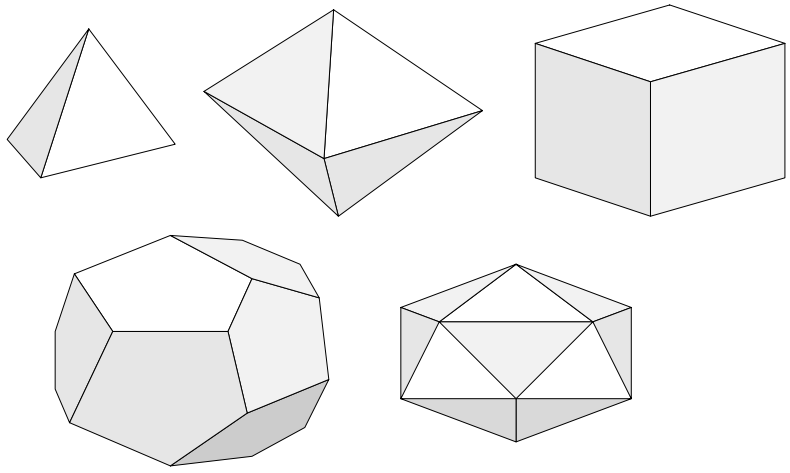


FIGURA 5. Os cinco sólidos Platônicos: tetraedro, cubo, octaedro, dodecaedro e icosaedro.

4. Fórmula de Euler

Em 1758, Leonard Euler notou uma regularidade no número de vértices, arestas, e faces de um poliedro de genus zero: o número de vértices mais o número de faces é sempre igual ao número de arestas mais dois. A tabela abaixo mostra esta relação para os cinco sólidos Platônicos.

Nome	(p, r)	V	E	F
Tetraedro	$(3, 3)$	4	6	4
Cubo	$(4, 3)$	8	12	6
Octaedro	$(3, 4)$	6	12	8
Dodecaedro	$(3, 5)$	20	30	12
Icosaedro	$(5, 3)$	12	30	20

Se denotarmos, como é usual, o número de vértices, arestas e faces de um poliedro por V, E e F, respectivamente, então a fórmula abaixo é conhecida como *Fórmula de Euler* (para poliedros de genus zero):

$$V - E + F = 2.$$

4.1. **Prova da Fómula de Euler.** A prova consiste em três partes:

(1) Transformar o poliedro em um grafo planar.
(2) Um teorema para árvores.
(3) Prova por indução.

Primeiro 'achataremos' a superfície do poliedro P através de consideráveis distorções. Remova de P uma face arbitrária f, deixando um buraco na superfície. Imagine que a superfície do poliedro é feita de borracha muito fina (como uma bexiga) e aumente este buraco até que o poliedro P, sem a face f, seja planar (esteja contido em um plano). Desta maneira transformamos o 1-esqueleto do poliedro P em um grafo planar: um grafo desenhado no plano sem que duas arestas cruzem. Seja G este grafo planar. Temos que os vértices de P correspondem a vértices de G, as arestas de P correspondem a arestas de G. A face f de P corresponde à fronteira do grafo G. As faces de P, exceto por f, correspondem as regiões do plano delimitadas por arestas de G. Essas regiões, juntamente com a região exterior de G que corresponde à face f de P, são chamadas de faces de G. A face correpondente a f é dita a *face exterior* de G. A figura 6 mostra um possível grafo correspondente ao processo de achatamento de um cubo.

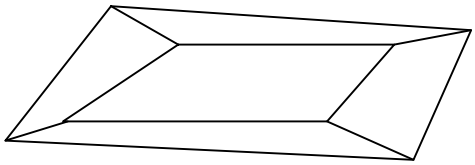


Figura 6. O 1-esqueleto de um cubo obtido pelo processo de achatamento.

Assim há uma correspondência bijetora entre vértice, arestas e faces de P, e vértice, arestas e faces de G, respectivamente. Podemos então nos concentrar em provar a Fórmula de Euler para grafos planares.

O segundo passo da prova é verificar que a Fórmula de Euler vale quando G é uma árvore. (Note que, neste caso, G não corresponde a nenhum poliedro, mas o importante é que isto será útil no próximo passo.) Se G é uma árvore, temos que $V = E + 1$, um fato que assumimos, e $F = 1$, já que a única face de G é a exterior. Logo, temos que a Fórmula de Euler é válida para árvores.

O terceiro passo é provar, por indução no número E de arestas, que a Fórmula de Euler vale para qualquer grafo conexo planar G. Suponha que a Fórmula de Euler vale para todos os grafos conexos planares com no máximo $E-1$ arestas, e seja G um grafo conexo planar com V vértices, E arestas e F faces. Se G é uma árvore, não há o que provar. Suponha que G tem um circuito C, e seja e uma aresta deste circuito. Temos que o grafo $G' = G-\{e\}$ é conexo e tem V vértices, $E-1$ arestas e $F-1$ faces (as duas faces de G incidentes a e correspondem a uma face de G'). Logo, pela hipótese de indução, temos que

$$V - (E-1) + (F-1) \;=\; 2 \;=\; V - E + F.$$

o que conclui a nossa prova.

4.2. **Consequências: linearidade do número de arestas e faces.** A Fórmula de Euler implica que o número de vértices, arestas e faces de um poliedro são linearmente relacionados: se $n = V$, então $E = \mathrm{O}(n)$ e $F = \mathrm{O}(n)$. Isso nos será útil ao analisarmos os algoritmos que serão estudados.

Primeiro, como queremos estipular limites superiores para E e F em função de V, podemos assumir, sem perda de generalidade, que todas as faces do poliedro são triângulos (se as faces não são triângulos, podemos triangularizá-las). Logo, o número E de arestas do poliedro é igual a $3F/2$. Assim, temos que $2E = 3F$. Aplicando a Fórmula de Euler, temos que

$$\begin{aligned} V - E + F = 2 \;&\Rightarrow\; V - E + 2E/3 \;=\; 2 \\ &\Rightarrow\; V - 2 \;=\; E/3 \\ &\Rightarrow\; E \;=\; 3V - 6 \;<\; 3V = \mathrm{O}(n) \end{aligned}$$

e

$$F \;=\; 2E/3 \;\Rightarrow\; 2V - 4 \;<\; 2V \;=\; 2n \;=\; \mathrm{O}(n).$$

Resumindo, temos o seguinte teorema:

Teorema 1. *Em um poliedro com $n = V$ vértices, E arestas e F faces, temos que, $V - E + F = 2$, $E = \mathrm{O}(n)$ e $F = \mathrm{O}(n)$.* ∎

Observação. Um resultado relacionado ao teorema acima é o seguinde (veja o corolário 8.2 de [3]):

Teorema 2. *Seja P o fecho convexo de um conjunto de n pontos em $\mathbb{R}^d$. O número de k-faces e o número de incidências entre k faces e $(k+1)$-faces de P é $\mathrm{O}(n^{\min\{\lfloor d/2 \rfloor, k+1\}})$, para $k = 0, \ldots, d-1$. Esses limites são assintoticamente justos.* ∎

5. Uma primitiva geométrica

Um triângulo é determinado por três pontos não-colineares. Um vetor perpendicular a um dado plano π é chamado de um vetor *normal* ou *ortogonal* a π. Dados três pontos não-colineares p_0, p_1 e p_2 em um plano π, um vetor normal a π é dado pelo produto vetorial $a \times b$ (ou $a \wedge b$), onde $a = p_1 - p_0$ e $b = p_2 - p_0$. Seja $a := (x_a, y_a, b_a)$ e $b := (x_b, y_b, z_b)$. O produto vetorial é definido por:

$$a \times b = \begin{vmatrix} y_a & z_a \\ y_b & z_b \end{vmatrix} \vec{i} + \begin{vmatrix} z_a & x_a \\ z_b & x_b \end{vmatrix} \vec{j} + \begin{vmatrix} x_a & y_a \\ x_b & y_b \end{vmatrix} \vec{k}.$$

Para mostrarmos que o produto vetorial $a \times b$ é perpendicular ao plano gerado pelos vetores a e b, basta verificarmos que $a \cdot (a \times b) = 0$ e $b \cdot (a \times b) = 0$.

A direção do produto vetorial é mostrada na figura 7. Quando visto a partir do ponto determinado por $a \times b$, o triângulo $\triangle(\mathbf{0}, a, b)$ é orientado positivamente (sentido anti-horário). O vetor normal tendo o mesmo comprimento mas direção oposta é dado por $-a \times b = b \times a$.

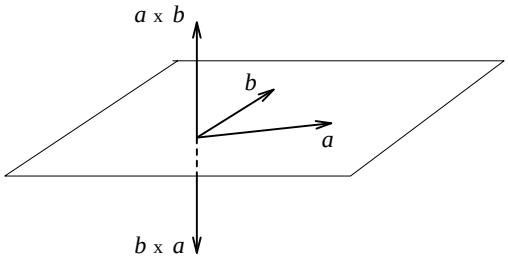


Figura 7. O produto vetorial $a \times b$.

Como já vimos, se a e b pertencem ao plano xy, então o comprimento do produto vetorial é $||a \times b|| = |x_a y_b - y_a x_b|$, que é a área do parelelogramo com vértices $\mathbf{0}, a, b$ e $a + b$.

O plano determinado por um triângulo (orientado) divide o espaço em dois semi-espaços. O semi-espaço para onde o vetor normal ao triângulo está apontando é chamado de *semi-espaço positivo* do triângulo (pois os pontos deste semi-espaço vêem o triângulo orientado positivamente. O outro semi-espaço é chamado de *semi-espaço negativo*.

Dados pontos p_0, p_1, p_2 e p_3 em $\mathbb{R}^3$, se estamos interessados em saber de qual lado do plano π, gerado pelos vetores $a := p_1 - p_0$ e $b := p_2 - p_0$, o ponto p_3 está, consideramos uma orientação de π dada pelo triângulo orientado (digamos) $\triangle(p_0, p_1, p_2)$ (o que estamos fazendo aqui é orientar o plano π através da sua base orientada (a, b)) e verificamos se o produto escalar entre $c := p_3 - p_0$ e o vetor $a \times b$ é: maior que zero, zero, ou menor que zero. Se $c \cdot (a \times b) > 0$, então o ponto p_3 está no semi-espaço positivo do triângulo $\triangle(p_0, p_1, p_2)$. Se $c \cdot (a \times b) = 0$, então o ponto p_3 está no plano π. Finalmente, se $c \cdot (a \times b) < 0$, então o ponto p_3 está no semi-espaço negativo do triângulo $\triangle(p_0, p_1, p_2)$. Logo, dados pontos p_0, p_1, p_2, p_3 em $\mathbb{R}^3$, a operação primitiva Teste-de-Orientação(p_0, p_1, p_2, p_3) (ou Esquerda(p_0, p_1, p_2, p_3)) decide em qual semi-espaço do plano orientado π (orientado pelo triângulo $\triangle(p_0, p_1, p_2)$), o ponto p_3 está. De maneira similar

ao que vimos no caso bidimensional, se $p_i = (x_i, y_i, z_i)$, para $i = 0, 1, 2, 3$, então o resultado do teste de orientação é dado pelo sinal do determinante

$$\begin{vmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix}.$$

O valor absoluto deste determinte é 6 vezes o volume sinalizado do tetraedro cujos vértices são p_0, p_1, p_2, p_3. Chamaremos de `Volume6` a função que devolve o valor deste determinante (em analogia à função ÁREA2).

[**Observação.** A orientação descrita acima pode ser generalizada para uma dimensão d arbitrária. O que estamos fazendo aqui é orientar bases do espaço $\mathbb{R}^d$ da mesma maneira que isto é feito em um curso de Álgebra Linear.]

6. O problema: estrutura de dados

Consideraremos o seguinte problema: dado um conjunto finito S de pontos no $\mathbb{R}^3$, encontrar o fecho convexo $\text{conv}(S)$ dos pontos em S. Da mesma maneira que fizemos no caso bidimensional, antes de continuarmos devemos ter claro o quê desejamos como resposta dos algoritmos.

Quando tratamos do problema de encontrar o fecho convexo de um conjunto de pontos no plano, a resposta dos algoritmos era o polígono que representava o fecho convexo (vértices extremos junto com uma ordem). No caso tridimensional, existem várias possibilidades de representarmos o fecho convexo e, como era de se esperar, diferentes representações levarão a diferenças no consumo de tempo de um mesmo algoritmo.

Trataremos nesta seção do problema de como representar um fecho convexo tridimensional. O material que veremos foi extraído do capítulo 3 de [4] e capítulo 4 de [7].

6.1. **Listas de faces.** A maneira mais simples de representarmos um poliedro tridimensional é representando cada uma de suas faces através de uma lista (ordenada) de vértices. Desta forma um poliedro é dado através de uma lista de suas faces (polígonos) que formam a sua superfície.

A grande desvantagem desta estrutura de listas de faces é que ela contém só um tipo de informação de adjacência: adjacência entre vértices de uma mesma face. Gastaríamos muito tempo se estivéssemos interessados em determinar outro tipo de adjacências, e.g., todas as faces adjacentes a uma face dada ou se desejássemos listar todas as faces que são adjacentes a um dado vértice; em ambos os casos deveríamos percorrer toda a estrutura.

6.2. **Estrutura de dados para poliedros simpliciais.** Poliedros *simpliciais* são aqueles cujas faces são triângulos. Para representarmos um polígono P, temos que representar três tipos primitivos de tipos de dados: vértices, arestas e faces. Uma maneira de fazermos isto é mantermos todos os vértices em uma lista duplamente ligada, a ordem dos elementos não tem nenhum significado. Faremos o mesmo com arestas e faces. Cada elemento destas listas tem um tamanho fixo. A estrutura correspondente a um vértice contém as suas coordenadas. A estrutura da aresta contém apontadores para os vértices que são seus extremos e apontadores para as duas faces que são incidentes a aresta. A estrutura correspondente a uma face contém apontadores para os três vértices que formam a face, além de apontadores para as três arestas.

```
struct tVertexStructure {
        int v[3];
        tVertex next, prev;
};
struct tEdgeStructure {
        tVertex endpts[2];
        tFace adjface[2];
        tEdge next, prev;
};
```

```
struct tFaceStructure {
        tVertex vertex[3];
        Edge edge[3];
        tFace next, prev;
};
```

6.3. **Estrutura de dados winged-edge.** Uma outra maneira de representarmos um poliedro é representando a sua superfície, mais especificamente, o seu *1-esqueleto*, que nada mais é do que um grafo planar. (Estruturas de dados que representam propriedades topológicas do poliedro são chamadas de *estruturas dados topológicas*; veja a página 57 de [4].)

A representação que veremos chama-se *winged-edge* ('aresta-alada') e foi proposta por Baumgart [1] em 1975. Ela é ainda a estrutura de dados mais popular para representar a superfície de um poliedro.

Esta representação é capaz de produzir rapidamente todas as possíveis relações de adjacência entre vértices, arestas e faces de um poliedro. Por exemplo, dadas duas faces do poliedro, podemos determinar se elas são adjacentes em tempo linear no número total de vértices destas faces. Dado um vértice do poliedro, seremos capazes de listar todas as faces que são incidentes a este vértice em tempo proporcional ao número destas faces.

O foco da estrutura de dados winged-edge são as arestas. Esta estrutura mantém uma lista de vértices, arestas e faces onde:

Vértice: a estrutura que representa cada vértice v mantém as coordenadas (x, y, z) do vértice juntamente com um apontador para uma aresta arbitrária incidente ao vértice; denotaremos por $av(v)$ a aresta incidente ao vértice v.

Face: para cada face f, mantemos um apontador para uma aresta arbitrária $af(f)$ da fronteira de f.

Aresta: a estrutura correspondente a cada aresta e tem oito apontadores:

(1) Dois apontadores para os vértices $v_1(e)$ e $v_2(e)$ que são pontas da aresta e. A ordem destes vértices fornece uma orientação para a aresta.

(2) Um apontador para cada uma das duas faces incidentes a e, $fccw(e)$ e $fcw(e)$. A face $fccw(e)$ é a esquerda de $e = v_1(e)v_2(e)$ (*counterclockwise*, já que a orientação induzida por e nesta face é anti-horária) e a face $fcw(e)$ é a direita (*clockwise*, já que a orientação induzida por e nesta face é horária).

(3) Quatro apontadores para arestas adjacentes a e, que são as asas (*wings*) de e: arestas que precedem e sucedem e nas faces $fccw(e)$ e $fcw(e)$. Especificamente, $pccw(e)$ (*previous counterclockwise*) e $nccw(e)$ (*next counterclockwise*) representam as arestas que precedem e sucedem e na face $fccw(e)$, respectivamente (orientada de acordo com a orientação da aresta e, i.e., sentido anti-horário). Analogamente, $pcw(e)$ e $ncw(e)$ representam as arestas que precedem e sucedem e na face $fcw(e)$.

A figura 8 ilustra esta estrutura. Note que as estruturas que representam um vértice, aresta ou face têm tamanho constante, uma propriedade que é bem útil.

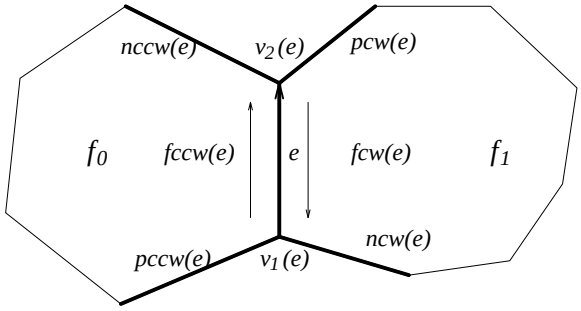


FIGURA 8. A estrutura de dados winged-edge.

A estrutura *winged-edge* correspondente ao cubo da figura 9 está representada pelas listas de vértices, arestas e faces abaixo.

vértice	(x, y, z)	av
p_1	$(1, 0, 0)$	e_1
p_2	$(1, 1, 0)$	e_2
p_3	$(1, 1, 1)$	e_3
p_4	$(1, 0, 1)$	e_4
p_5	$(0, 0, 0)$	e_9
p_6	$(0, 1, 0)$	e_{10}
p_7	$(0, 1, 1)$	e_{11}
p_8	$(0, 0, 1)$	e_{12}

face	af
f_1	e_1
f_2	e_2
f_3	e_{10}
f_4	e_{12}
f_5	e_1
f_6	e_8

aresta	v_1	v_2	$fccw$	fcw	$pccw$	$nccw$	pcw	ncw
e_1	p_1	p_2	f_1	f_5	e_4	e_2	e_6	e_5
e_2	p_2	p_3	f_1	f_2	e_1	e_3	e_7	e_6
e_3	p_3	p_4	f_1	f_6	e_2	e_4	e_8	e_7
e_4	p_4	p_1	f_1	f_4	e_3	e_1	e_5	e_8
e_5	p_1	p_5	f_5	f_4	e_1	e_9	e_{12}	e_4
e_6	p_2	p_6	f_2	f_5	e_2	e_{10}	e_9	e_1
e_7	p_3	p_7	f_6	f_2	e_3	e_{11}	e_{10}	e_2
e_8	p_4	p_8	f_4	f_6	e_4	e_{12}	e_{11}	e_3
e_9	p_5	p_6	f_5	f_3	e_5	e_6	e_{10}	e_{12}
e_{10}	p_6	p_7	f_2	f_3	e_6	e_7	e_{11}	e_9
e_{11}	p_7	p_8	f_6	f_3	e_7	e_8	e_{12}	e_{10}
e_{12}	p_8	p_5	f_4	f_3	e_8	e_5	e_9	e_{11}

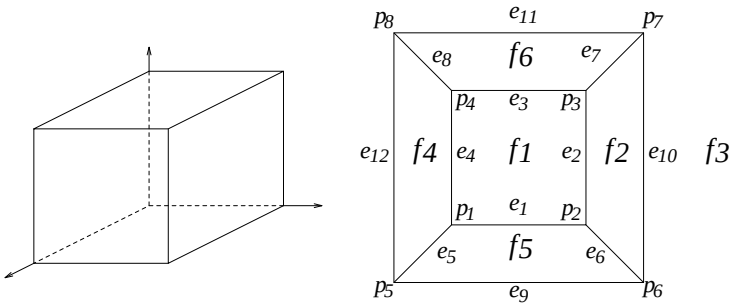


FIGURA 9. Um cubo e seu 1-esqueleto (grafo planar).

7. O método do embrulho para presente (gift wrapping)

Como é mecionado em Preparata & Shamos [8], não se conhece nenhum algoritmo razoavelmente eficiente para construir o fecho convexo de um conjunto de pontos que não produza uma descrição da fronteira (1-esqueleto) do politopo (fecho convexo). (Lembre-se dos algoritmo que vimos para construir o fecho convexo de um conjunto de pontos no plano.)

O algoritmo que veremos é uma generalização do algoritmo de Jarvis [5] que encontrar o fecho convexo de um conjunto de pontos no plano. O método do embrulho para presente (*gift wrapping)* foi desenvolvido po Chand & Kapur [2] para construir o fecho convexo de um dado conjunto S de pontos em dimensões arbitrárias. O consumo de tempo do algoritmo é $O(nh)$, onde h é o número de arestas de $\text{conv}(S)$, i.e., o algoritmo é *output-sensitive*.

[**Observação.** O material desta seção foi extraído do Capítulo 3 de [4], Capítulo 3 de [8], e Capítulo 3 de [7].]

7.1. **O algoritmo.** Para facilitar a exposição vamos assumir que os pontos de S estão em posição geral: não existem quatro pontos em S que sejam coplanares. Intuitivamente o algoritmo simula o enrolar do conjunto S por um papel (de presente). A ideia, analogamente ao algoritmo de Jarvis, é usar as arestas das faces do fecho convexo (politopo) que já foram encontradas para encontrar outras faces (já que cada aresta é compartilhada por exatamente duas faces). Em cada passo do algoritmo, temos que uma parte do fecho convexo já foi construída. Uma face f na superfície parcial do fecho convexo é selecionada. Uma aresta e de f para a qual a outra face adjacente não foi encontrada ainda também é selecionada. (Uma tal aresta será dita uma aresta *livre*.) O plano π contendo f é rotacionado ao redor de e na direção do conjunto de pontos dado até que um ponto p é encontrado (veja a Figura 10). Então $\text{conv}(e \cup \{p\})$ é uma nova face triangular do fecho convexo e o processo continua.

Seja P o fecho convexo de S, e $e = p_0p_1$ uma aresta de uma face f de P tal que a outra face de P incidente a e ainda não foi encontrada pelo algoritmo. Como f é uma face de P, todos os pontos do conjunto S estão em um mesmo semi-espaço definido pelo plano (orientado) π que contém a face f. A face f' adjacente a f em e é tal que todos os semiplanos determinados por e e

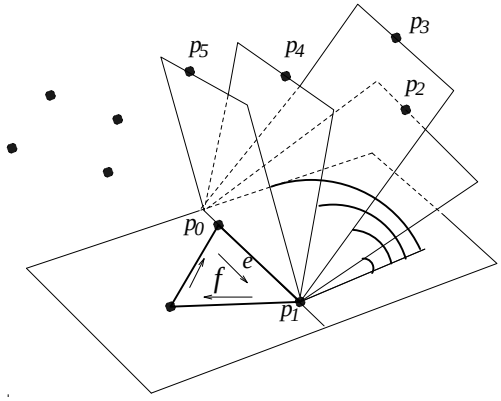


FIGURA 10. Ilustração de um passo do algoritmo EMBRULHO3D.

por pontos em S estão situados em um ângulo diedral maior ou igual ao ângulo diedral formado por e, pelo plano π e pelo plano π' contendo f' (veja a Figura 10). Desta forma, dentre todos os semiplanos determinados por e e por pontos em S, a face f' está contida naquele que forma o menor ângulo em relação a e e o plano π. (Esta descrição esta vaga. Por favor, olhe a Figura 10 e note que os pontos de S estão todos no semi-espaço positivo determinado por $\triangle(p_0, p_1, p_2)$.) Este passo do algoritmo é chamado em [4] de *passo do embrulho para presente* e é o passo central do algoritmo.

Como começar o algoritmo já que no início não temos nenhuma face de $\text{conv}(S)$? (Veja [4], pg. 63.) Procederemos de maneira análoga ao algoritmo de Jarvis, que chamamos de EMBRULHO, para fecho convexo bidimensional. Começamos determinando um ponto extremo p_0 de S: seja p_0 o ponto de S com z-coordenada mínima. (Em caso de empate, escolha o ponto com menor y-coordenada e, se persistir o empate, escolha o de menor x-coordenada—na realidade, se tivermos empates o nosso trabalho à frente fica, de certa forma, mais fácil.) Consideremos agora a reta r que é paralela ao eixo das abscissas e contém o ponto p_0 e consideremos ainda o semi-plano horizontal π contendo p_0 e orientado positivamente na direção do eixo y. Giramos este semi-plano no sentido de y para z até encontrarmos outro ponto extremo p_1 de S. Agora, giramos o novo plano π em torno da aresta $e = p_0p_1$ até encontrar um outro ponto extremo p_2 de S. O triângulo $\triangle(p_0, p_1, p_2)$ será a nossa face inicial. (O que foi descrito aqui nada mais é do que o passo do embrulho para presente aplicado duas vezes.)

7.2. **O Algoritmo.** Abaixo encontra-se uma descrição do algoritmo EMBRULHO3D. O algoritmo começa com uma face inicial, gerada como foi descrito acima. Para cada aresta adjacente a esta face, o algoritmo encontra a face adjacente. No próximo passo, o algoritmo passa a examinar uma outra face e termina quando todas as faces foram geradas. Quando uma nova face f do fecho convexo é gerada pelo passo do embrulho para presente, suas arestas são examinadas para verificar se elas pertencem a uma face já criada. Neste caso, as duas faces incidentes a esta aresta já estão determinadas e a aresta não precisará mais ser examinada pelo passo do embrulho para presente. Se só uma das faces da aresta foi determinada, chamamos a aresta

de *livre*. Cada face recém-criada é colocada em uma fila Q. A cada passo do algoritmo, uma face é removida da fila e o passo embrulho-para-presente é aplicado a cada uma de suas arestas livres. O algoritmo termina quando a fila estiver vazia (em outra palavras, quando não existirem mais arestas livres a serem examinadas).

```
EMBRULHO3D(S, n)
 1  CRIEFILA(Q)   ▷ fila com as faces encontradas
 2  CRIEWE(T)     ▷ ED winged edges para conv(S)
 3  f ← FACEINICIAL(S, n)
 4  INSIRAFILA(Q, f)
 5  INSIRAWE(T, f)
 6  enquanto não FILAVAZIA(Q) faça
 7      f ← REMOVAFILA(Q)
 8      para cada aresta livre e de f faça
 9          f' ← FACEADJACENTE(f, e)
10          INSIRAFILA(Q, f')
11          INSIRAWE(T, f)
12  devolva T
```

7.3. **Análise do algoritmo.** Os passos dominantes do algoritmo são os Passos 7 e 9. O Passo 7 (embrulho-para-presente) pode ser realizado em tempo $O(n)$. Logo, se o número de arestas do fecho convexo de S é h, temos que este passo gasta tempo $O(hn)$ durante toda a execução do algoritmo. No Passo 9, cada vez que uma face é encontrada, é necessário percorrermos a estrutura de dados para verificar se as arestas desta face já foram criadas. Para cada aresta o tempo gasto com esta verificação é $O(n)$. Como cada aresta é gerada exatamente duas vezes temos que o tempo total gasto por este passo durante a execução do algoritmo é $O(hn)$. A primitiva usada pelo algoritmo é `Teste-de -Orientação` (ou `Volume6`).

Teorema 3. *O Algoritmo* `Embrulho-para-Presente` *constrói o fecho convexo de um conjunto de n pontos em* E^3 *em tempo* $O(hn)$*, onde h é o número de arestas do fecho convexo.*

Note que se a descrição gerada pelo algoritmo para o fecho convexo contém faces coplanares isto não é inconveniente algum. A partir desta representação podemos obter uma nova representação sem faces coplanares em tempo linear.

8. Um algoritmo incremental

Como foi mencionado, o algoritmo INCREMENTAL para construir o fecho convexo de um conjunto de pontos no plano tem uma extensão natural para a versão tridimensional do problema. O algoritmo INCREMENTAL3D é idêntico ao algoritmo INCREMENTAL que já estudamos.

[Lembremos que um *algoritmo incremental* examina os dados do problema, um item por vez, enquanto uma solução corrente dos itens que já foram vistos até o momento é mantida. Em cada iteração, o próximo item é examinado e processado

e a solução corrente para o problema é atualizada para incorporar este novo item.]

Observação. O material desta seção foi extraído do capítulo 4 de [7].

8.1. **O algoritmo.** A ideia básica do algoritmo é examinar um ponto por vez, atualizando o fecho convexo dos pontos já examinados. O trabalho torna-se mais simples já que, a cada passo, precisamos incluir um único ponto no fecho convexo do pontos já examinados.

Assumiremos, para facilitar a exposição, que o conjunto S de pontos dados não contém três pontos colineares e nem quatro pontos coplanares, i.e., os pontos em S estão em posição geral. (Assim, de novo, as faces de $\text{conv}(S)$ serão triângulos.)

A descrição em alto nível do algoritmo INCREMENTAL3D é a mesma da versão bidimensional:

INCREMENTAL(S, n)
1 $P_3 \leftarrow \text{conv}(\{p_0, p_1, p_2, p_3\})$
2 para $k \leftarrow 4$ até $n-1$ faça
3 $\quad P_k \leftarrow \text{conv}(P_{k-1} \cup \{p_k\})$
4 devolva P_{n-1}

Como mostra o algoritmo acima, o primeiro fecho convexo é o tetraedro $\text{conv}(\{p_0, p_1, p_2, p_3\})$. Seja $Q = P_{k-1}$ e $p = p_k$. O problema de construir $\text{conv}(Q \cup \{p\})$ (linha 3 do algoritmo) cai em um dos dois casos a seguir:

(1) $p \in Q$. Uma vez que foi determinado que $p \in Q$, o ponto p pode ser descartado e sabemos que $\text{conv}(Q \cup \{p\}) = Q$. Para decidir se $p \in Q$, podemos usar a rotina VOLUME6. Esta decisão pode ser feita em tempo $O(n)$: $p \in Q$ se e somente se, para cada face f de Q, temos que p pertence a f ou p pertence ao semi-espaço negativo determinado por f. (A escolha aqui de qual é o semi-espaço negativo foi arbitrária, como todas as orientações que temos escolhido. Desta maneira, um ponto p está no interior de Q se e somente se uma pessoa sentada neste ponto vê cada face de Q orientada negativamente.)

(2) $p \notin Q$. Se, em algum momento, p vê alguma face de Q orientada positivamente, então p está fora de Q e temos que construir $\text{conv}(Q \cup \{p\})$. No caso bidimensional, tínhamos que encontrar somente as duas retas que passam pelo ponto p e que são tangentes ao polígono Q. Na versão tridimensional, temos que encontrar planos tangentes em vez de restas tangentes. Estes planos tangentes determinam um cone que tem como faces triângulos e tem como bico o ponto p (veja a figura 11).

Discutiremos agora como estas faces triangulares do cone com ponta em p podem ser encontradas. Imagine-se sentado no ponto p, olhando para o poliedro Q. Algumas faces de Q são visíveis a partir de p, outras faces não são. As faces de Q que serão descartadas são precisamente aquelas que são visíveis da posição que está o ponto p (do ponto p um observador vê estas faces orientadas positivamente). Além disso, as arestas na fronteira das faces visíveis são aquelas que formarão as faces triangulares do cone juntamente

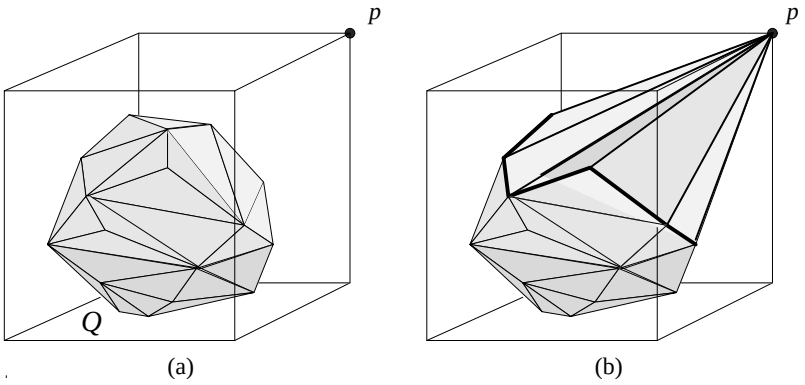


FIGURA 11. (a) Poliedro Q antes do algoritmo examinar o ponto p. (b) Poliedro $\text{conv}(Q \cup \{p\})$.

com o ponto p (i.e., cada uma das arestas da fronteira juntamente com o ponto p formarão uma face triangular do cone, e portanto de $\text{conv}(S \cup \{p\})$). De fato, suponha que e é uma aresta de Q tal que o plano contendo e e o ponto p é tangente a Q. Como cada aresta é compartilhada por exatamente duas faces, temos que uma das faces incidentes a e é visível a partir de p e a outra não é. Logo, e está na fronteira da região visível de p.

Da discussão acima, para determinar as faces de Q que serão descartadas e as faces de $\text{conv}(Q \cup \{p\})$ que não são faces de Q (i.e., as faces triangulares do cone formado pelo ponto p e pelos planos tangentes a Q), basta determinarmos quais faces de Q podem ser vistas por um observador que está no ponto p.

Uma definição mais precisa de visibilidade é a seguinte. Uma face f de um poliedro Q é *visível* de p se existe um ponto q interior a face f tal que o segmento pq intersecta o poliedro Q somente no ponto q (i.e., $pq \cap Q = \{q\}$). Para decidirmos se uma face f determinada pelo triângulo $\triangle(a, b, c)$ é visível de p, basta verificarmos o sinal do volume do tetraedro formado por a, b, c e p (i.e., uma chamada da função VOLUME6): f é visível de p se e somente se este sinal é positivo.

Uma descrição mais detalhada do algoritmo encontra-se abaixo.

```
INCREMENTAL3D(S, n)
 1  P_3 ← TETRAEDRO(p_0, p_1, p_2, p_3)
 2  para k ← 4 até n − 1 faça
 3      para cada face f de P_{k−1} faça
 4          v ← VOLUME6(f, p_k)
 5          se v > 0 então marque f como visível de p_k
 6      se nenhuma face é visível de p_k
 7          então P_k ← P_{k−1}
 8          senão para cada aresta e na fronteira das faces visíveis faça
 9                    construa a face determinada por e e p_k
10                para cada face visível f faça
11                    remova f de P_{k−1}
12                faça os acertos finais obtendo P_k
13  devolva P_{n−1}
```

8.2. **Análise do algoritmo.** Pela Fórmula de Euler, o número de faces e de arestas do fecho convexo P_k em toda iteração é $O(n)$, onde n é o número de pontos no conjunto dado S. Logo, cada um dos laços envolvendo arestas e faces, nas linhas 3, 8 e 10, consome tempo $O(n)$. Os acertos na linha 12 consomem tempo proporcional ao número de faces construídas nesta iteração. As linhas 4, 9 e 11 consomem tempo constante. Logo, como cada um destes laços é executado $O(n)$ vezes, o consumo de tempo do algoritmo INCREMENTAL3D é $O(n^2)$.

REFERÊNCIAS

1. B.G. Baumgart, *A polyhedron representation for computer vision*, Proc. AFIPS Natl. Comput. Conf., vol. 44, 1975, pp. 589–596.
2. D.R. Chand and S.S. Kapur, *An algorithm for convex polytopes*, JACM **17** (1970), no. 1, 78–86.
3. H. Edelsbrunner, *Algorithms in combinatorial geometry*, EATCS Monographs on Theoretical Computer Science, no. 10, Springer-Verlag, Berlin, 1987, QA758 E21a.
4. L.H. Figueiredo and P.C.P. Carvalho, *Introdução à geometria computacional*, 18º Colóquio Brasileiro de Matemática, IMPA, 1991, QA758 F475i.
5. R.A. Jarvis, *On the identification of the convex hull of a finite set in the plane*, Information Processing Letters **2** (1973), 18–21.
6. M.J. Laszlo, *Computational geometry and computer graphics in C++*, Prentice Hall, Upper Saddle River, NJ, 1996.
7. J. O'Rourke, *Computational geometry in C*, Cambridge University Press, Cambridge, 1993.
8. F.P. Preparata and M.I. Shamos, *Computational geometry: An introduction*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1985, QA758 P927c.