# MC409/MO603 – Computação Gráfica

© Jorge Stolfi

**Segundo Semestre de 1994**

## Notas de Aula – Fascículo 7

# Podando a cena

## 7.1 O volume da imagem

Antes de desenhar a projeção de qualquer parte da cena, devemos verificar se a mesma é de fato visível.

### 7.1.1 A janela da imagem

Como sabemos, para desenhar a imagem $p'$ de um ponto $p$ da cena, precisamos em princípio tomar a semi-reta que sai da posição do observador $o$ e passa por $p$; a imagem desejada $p'$ é então o ponto onde esta semi-reta encontra o plano $\pi$ da imagem.

Portanto, um ponto $p$ da cena só poderá ser visível na imagem se a semi-reta $op$ encontrar o plano $\pi$ dentro da área $I$ que corresponde à imagem. Em outras palavras, só se $p$ estiver na região do espaço que é a união de todas as semi-retas com origem em $o$ que passam por $I$. Veja a figura ??(a). Chamaremos esta região de *volume da imagem*.

Qualquer ponto da cena fora deste volume é com certeza invisível. Um ponto $p$ dentro deste volume é visível, a menos que seja oculto por alguma outra parte opaca da cena, mais próxima ao observador do que $p$ (um problema de que trataremos no próximo fascículo).

Como vimos, o cálculo da imagem pode ser efetuado algebricamente aplicando-se, a todos os pontos da cena, três transformações projetivas em seqüência: (1) uma translação $T$ que traz a origem do SC para a origem $f$ do SI; (2) uma rotação $R$ que roda os vetores $rst$ de modo a coincidir com os eixos do SI; e, finalmente, (3) uma transformação não-afim $P$ que mantém todos os pontos do plano $Z = 0$ fixos, e leva a posição do observador $[1, 0, 0, d]$ para o ponto no infinito $[0, 0, 0, 1]$. Feito isso, a imagem pode ser calculada projetando-se os pontos da cena transformados, paralelamente ao eixo $Z$, sobre o plano $Z = 0$ — isto é, descartando-se a coordenada $Z$.

Considere a situação depois que $T$ e $R$ (mas não $P$) foram aplicadas. Suponha que a imagem corresponde a um retângulo finito $I = [X_{\min} \_ X_{\max}] \times [Y_{\min} \_ Y_{\max}]$ do plano $Z = 0$. Neste momento, o volume da imagem é a pirâmide infinita que tem o observador $o = (0, 0, d)$ por ápice, e $I$ por secção. Veja a figura **??**(a). Se aplicarmos agora a transformação de perspectiva $P$, esta pirâmide transforma-se num prisma semi-infinito, paralelo ao eixo $Z$, cuja secção é o retângulo $I$. Este prisma se estende desde o *plano de fundo* $Z = -d$ (que é a imagem do plano $\Omega_2$ por $P$), até o infinito na direção de $Z > 0$. Vide a figura **??**(b).

### 7.1.2 O plano de frente

À medida que uma parte da cena se aproxima do olho do observador, sua imagem fica infinitamente ampliada pela projeção de perspectiva. Esse fenômeno pode causar problemas numéricos de precisão e *overflow* no cálculo da imagem.

Para evitar esses problemas, costuma-se na prática eliminar da cena, além dos pontos invisíveis, também todos os pontos que, depois das transformações $T$ e $R$, teriam $Z \geq d - \varepsilon$, onde $\varepsilon$ é uma constante positiva apropriada.

Isto equivale a truncar artificialmente a pirâmide da imagem por um *plano de frente*, paralelo ao plano da imagem, que passa a uma distância $\varepsilon$ na frente do observador. Depois da transformação $P$, este plano de frente torna-se o plano $Z = d(d - \varepsilon)/\varepsilon$; e a pirâmide truncada transforma-se no prisma retangular finito (paralelepípedo)

$$[X_{\min} \_ X_{\max}] \times [Y_{\min} \_ Y_{\max}] \times [Z_{\min} \_ Z_{\max}] \tag{7.1}$$

onde $Z_{\min} = -d$ e $Z_{\max} = d(d - \varepsilon)/\varepsilon$.

### 7.1.3 Antes ou depois da perspectiva?

Em algum momento ao longo do cálculo da imagem precisamos descartar os pontos da cena que estão fora do volume da imagem. Chamaremos este processo de *poda da cena.*

Em princípio, é possível podar a cena tanto antes de aplicar a transformação $TRP$ (isto é, no SC), quanto depois dela (isto é, no SI).

Intuitivamente, talvez pareça mais eficiente podar a cena antes da transformação, pois assim evitaríamos aplicar esta às partes da cena que estão fora do volume da imagem. Entretanto, no SC o volume da imagem é uma pirâmide com orientação arbitrária, enquanto que no SI ele é um prisma retangular (paralelepípedo) paralelo aos eixos. Como veremos, a poda é bem mais fácil para o prisma do que para a pirâmide; e na verdade a diferença entre os dois é exatamente igual ao custo de aplicar a transformação $TRP$. Portanto, as duas opções são na verdade igualmente eficientes. Como é mais fácil de codificar a poda após a transformação, a maioria das implementações, tanto em *software* como em *hardware*, adotam esta seqüência.

(Por outro lado, é importante podar a cena *antes* de aplicar os algoritmos de visibilidade do próximo fascículo, pois estes são bem mais demorados que o algoritmo de poda.)

### 7.1.4 Espaço orientado ou não-orientado?

Os algoritmos de poda e visibilidade que veremos a seguir praticamente exigem o uso da geometria projetiva orientada, e do espaço de dois lados. Por exemplo, esses algoritmos são baseados em rotinas que calculam a intersecção entre objetos geométricos e semi-espaços; e, como sabemos, a noção de semi-espaço é problemática em espaços projetivos não orientados. Além disso, eles tipicamente representam um segmento pelos seus extremos, um triângulo pelos seus vértices, e assim por diante; e já vimos que essas representações são ambíguas na geometria não-orientada.

Essa ambigüidade não é apenas teórica. Considere um segmento ordinário $ab$ da cena onde $a$ está atrás do plano do observador, e $b$ na frente. Como esse segmento atravessa o plano do observador, após a transformação de perspectiva ele estará atravessando o plano no infinito, e portanto será formado por duas semi-retas opostas. Portanto, a imagem de um segmento ordinário pode ser tanto um segmento ordinário quanto um par de semi-retas opostas. A geometria projetiva clássica não nos permite decidir facilmente entre estas duas possibilidades.

Em contraste, no espaço orientado $\mathbb{T}_3$, essa ambigüidade não existe: a imagem do segmento $ab$ é sempre o segmento que une as imagens de $a$ e de $b$. No exemplo em questão, a imagem de $b$ será um ponto do aquém, e a de $a$ um ponto do além; e portanto o segmento que liga estas duas imagens será automaticamente um par de semi-retas opostas, uma em cada lado de $\mathbb{T}_3$.

### 7.1.5 A *pipeline* de poda

Como vimos, o volume visível (após a transformação de perspectiva) é um paralelepípedo, que é a intersecção de seis semi-espaços. Portanto, para podar as partes da cena que estão fora do volume visível basta cortar a cena sucessivamente por cada um dos seis planos que definem esses semi-espaços, e a cada etapa descartar as partes que estão do lado errado do plano em questão.

Uma dificuldade na codificação deste processo é que o corte de um objeto simples (segmento, triângulo, ou polígono) por um plano pode produzir zero objetos, ou mais de um objeto, de cada lado do plano. Por exemplo, a intersecção de um polígono simples com um semi-espaço pode consistir de vários polígonos isolados.

Uma maneira de contornar esta dificuldade é codificar cada etapa como um procedimento que recebe uma *lista* ligada de objetos, e devolve uma *lista* ligada das partes desses objetos que estão do lado certo do plano.

Outra solução, que evita o trabalho e custo de de percorrer e construir listas ligadas, é considerar cada etapa como um "filtro" (no sentido do sistema TUNIX) que "lê" um objeto de cada vez de um *stream* de entrada, e "escreve" um objeto de cada vez num *stream* de saída, assincronamente. O processo todo pode então ser descrito como uma *pipeline* de seis destes filtros, onde a saída de cada estágio é a entrada do estágio seguinte. Veja a figura **??**. Note que para cada objeto que o filtro lê do *stream* de entrada, ele pode escrever zero objetos, ou mais do que um objeto, no *stream* de saída.

Este modelo *pipeline* é particularmente conveniente para linguagens de programação que suportam o conceito de co-rotinas ou processos, como Modula e os *shells* do TUNIX. Ele também é implementado em *hardware* em certas estações gráficas de alto desempenho.

A noção de *pipeline* também será útil mais adiante, quando estudarmos algoritmos de visibilidade (para a eliminação de partes da cena ocultas por outros objetos da mesma).

Uma desvantagem da organização *pipeline* é que ela pode produzir mais fragmentos do que necessário, tanto na saída final quanto nas etapas intermediárias. Em particular, os estágios iniciais podem quebrar um objeto em dezenas de fragmentos, que são todos eliminados pelo estágio final. (Veremos exemplos concretos deste problema mais adiante.) Existem na literatura vários algoritmos que permitem detectar estes casos e curto-circuitar a *pipeline*, evitando esse trabalho inútil.

### 7.1.6 Corte de segmentos

O algoritmo abaixo descreve um filtro que corta segmentos (abertos) contra um plano $\pi = \langle \mathcal{W}, \mathcal{X}, \mathcal{Y}, \mathcal{Z} \rangle$, e devolve as partes desses segmentos que estão no semi-espaço positivo (aberto) de $\pi$:

1. Repita:

   1.1. Leia o próximo segmento. Sejam $a = [w_a, x_a, y_a, z_a]$ e $b = [w_b, x_b, y_b, z_b]$ seus extremos.

   1.2. Calcule $s_a = \mathcal{W}w_a + \mathcal{X}x_a + \mathcal{Y}y_a + \mathcal{Z}z_a$, $s_b = \mathcal{W}w_b + \mathcal{X}x_b + \mathcal{Y}y_b + \mathcal{Z}z_b$.

   1.3. Se $s_a \geq 0$ e $s_b \geq 0$, e $(s_a, s_b) \neq (0, 0)$, então

      1.3.1. Escreva o segmento $ab$ na saída.

   1.4. Senão, se $s_a > 0$ ou $s_b > 0$, então

      1.4.1. Calcule o ponto $u$ onde o segmento $ab$ cruza o plano $\pi$; isto é,

      $$u = [\,|s_b|w_a{+}|s_a|w_b,\; |s_b|x_a{+}|s_a|x_b,\; |s_b|y_a{+}|s_a|y_b,\; |s_b|z_a{+}|s_a|z_b\,]$$

      1.4.2. Se $s_a > 0$, escreva o segmento $au$ na saída, senão escreva $ub$.

A poda de segmentos contra ao volume da imagem pode então ser feita por uma *pipeline* com seis instâncias deste filtro.

### 7.1.7 A otimização de Cohen-Sutherland

Como observamos antes, a *pipeline* de poda às vezes desperdiça ciclos, recortando segmentos que são inteiramente descartados por um estágio posterior.

Uma otimização que parece valer a pena (especialmente quando a maioria dos segmentos está fora do volume da imagem) é calcular os coeficientes $s_a$, $s_b$ para todos os seis estágios, usando os extremos $a, b$ originais do segmento, antes de qualquer outro cálculo. (Note que estes cálculos podem ser feitos em paralelo, com *hardware* apropriado.)

Se para algum estágio estes coeficientes forem ambos negativos ou nulos, o segmento inteiro está no lado errado do plano correspondente a esse estágio, e pode ser descartado sem mais. Senão, se em todos os estágios os coeficientes $a_a, s_b$ forem positivos, o segmento inteiro está dentro do volume da imagem, e pode ser imediatamente colocado na saída da *pipeline*. Nos demais casos, o segmento é processado normalmente pela *pipeline*, pulando-se porém todos os estágios para os quais os dois coeficientes $s_a, s_b$ eram positivos.

Esta otimizaçào é geralmente descrita na literatura sob o nome de *algoritmo de Cohen-Sutherland.*

### 7.1.8 Corte de triângulos

O corte de triângulos por um plano é apenas um pouco mais complicado que o corte de segmentos. O filtro abaixo é uma possível solução. Ele corta uma seqüência de triângulos (abertos) contra um plano $\pi = \langle \mathcal{W}, \mathcal{X}, \mathcal{Y}, \mathcal{Z} \rangle$, devolvendo as partes desses triângulos que estão no semi-espaço (aberto) positivo.

Note que a intersecção de um triângulo com um semi-espaço pode ser um quadrilátero. Quando isso acontece, o filtro abaixo automaticamente decompõe a mesma em dois triângulos, para que a saída possa servir de entrada a outra instância do mesmo algoritmo.

1. Repita:

 1.1. Leia o próximo triângulo da entrada. Sejam $v_i = [w_i, x_i, y_i, z_i]$ seus vértices, para $i \in \{1, 2, 3\}$.

 1.2 Calcule $s_i = \mathcal{W}w_i + \mathcal{X}x_i + \mathcal{Y}y_i + \mathcal{Z}z_i$, para $i \in \{1, 2, 3\}$.

 1.3. Permute os vértices $v_i$ junto com os valores $s_i$ de modo que estes estejam em ordem decrescente.

 1.4. Se $s_1 > 0$ e $s_3 \geq 0$, então

  1.4.1. Escreva o triângulo $v_1 v_2 v_3$ na saída.

 1.5. Senão, se $s_1 > 0$ e $s_2 \leq 0$, então

  1.5.1 Calcule os pontos $p$ e $q$ onde os segmentos $v_1 v_2$ e $v_1 v_3$ cruzam o plano $\pi$, respectivamente. (Note que $p$ pode ser o próprio $v_2$.)

  1.5.2. Escreva o triângulo $v_1 p q$ na saída.

 1.6. Senão, se $s_2 > 0$ e $s_3 < 0$, então

  1.6.1. Calcule os pontos $p$ e $q$ onde os segmentos $v_2 v_3$ e $v_1 v_3$ cruzam o plano $\pi$, respectivamente.

1.6.2. Escreva os triângulos $v_1v_2q$ e $v_2pq$, e o segmento $v_2q$, na saída.

Note que este algoritmo pode inverter a ordem cíclica dos vértices de um triângulo. Esta propriedade pode ser inconveniente em certos contextos. Por exemplo, se o triângulo tem cores diferentes nos dois lados (frente e verso), esta inversão pode fazer com que ele seja pintado com a cor errada. Para evitar este problema, basta tomar nota do número de pares de vértices que foram trocados no passo **??**; se este número for ímpar, devemos permutar dois vértices quaisquer de todo triângulo que for escrito na saída.

Note também que uma *pipeline* de seis estágios sucessivos usando o algoritmo acima pode quebrar um triângulo de entrada em mais triângulos do que seria necessário. A intersecção de um triângulo com um paralelepípedo é um polígono convexo de no máximo 9 arestas, que pode ser coberto com 7 triângulos. Entretanto, seis passos do algoritmo acima podem produzir 26 triângulos na saída, talvez mais.

Não sei se este problema é significativo na prática. Se for, um possível remédio é acrescentar após cada "filtro cortador" um "filtro aglutinador", que coleta todos os triângulos consecutivos com a mesma cor e pertencentes ao mesmo plano, tenta juntá-los num polígono simples, e decompõe este num número mínimo de triângulos. Mas uma solução mais eficiente é trabalhar diretamente com polígonos em vez de triângulos, como veremos na próxima seção.

A otimização de Cohen-Sutherland, descrita anteriormente para o caso de segmentos, pode também ser aplicada a triângulos. Neste caso, ela consiste em calcular primeiro todos os coeficientes $s_1, s_2, s_3$ para todos os seis estágios, usando os vértices originais, antes de qualquer outro cálculo. Se em algum estágio os três coeficientes forem negativos ou nulos, o triângulo inteiro é descartado. Se em todos os estágios os três coeficientes forem positivos, o triângulo inteiro é colocado diretamente na saída da *pipeline*. Nos demais casos, o triângulo é processado normalmente pela *pipeline*, pulando-se os estágios onde os três coeficientes eram positivos.

### 7.1.9 Corte de polígonos

Em teoria, algoritmos para corte de polígonos são supérfluos, pois todo polígono de $n$ lados pode ser decomposto em $n-2$ triângulos. (Algoritmos eficientes para este fim são estudados em geometria computacional.) Entretanto, como vimos, a estratégia de reduzir tudo a triângulos pode levar a uma certa ineficiência. Além disso, a grande maioria das cenas que ocorrem na prática (prédios, peças mecânicas, móveis, etc.) são naturalmente descritos em termos de polígonos com quatro ou mais lados.

Um polígono com mais do que três vértices é um objeto muito mais complicado do que um segmento ou um triângulo. Por exemplo, quatro pontos do espaço não pertencem necessariamente ao mesmo plano; assim, representar um polígono pela lista de seus vértices — como fizemos com triângulos — é um desperdício de espaço, e exige cuidados especiais para evitar inconsistências. Outra razão é que os polígonos que ocorrem em objetos reais freqüentemente têm buracos e/ou vários pedaços isolados. Em alguns casos é necessário descrever polígonos que consistem de um plano inteiro, menos alguns buracos. Assim, dados os vértices de um polígono, não é óbvio como eles estão ligados, e quais pontos do plano então dentro ou fora do polígono.

Por essas razões e outras, é aconselhável representar um polígono $P$ no espaço por (1) um polígono $P^*$ no *plano* $\mathbb{T}_2$, e (2) uma transformação projetiva $\mu_P$ que leva $\mathbb{T}_2$ para um determinado plano $\pi_P$ de $\mathbb{T}_3$. Por definição, $P$ é a imagem de $P^*$ sob $\mu_P$.

A função $\mu_P$ pode ser especificada por uma matriz real $\mathsf{M}_P$ de dimensão $3 \times 4$, que, multiplicada pelas coordenadas homogêneas de um ponto de $\mathbb{T}_2$, dá as coordenadas de sua imagem em $\mathbb{T}_3$:

$$
\begin{aligned}
\mu_P \quad &\equiv \quad [w,x,y] \;\mapsto\; [w,x,y]\mathsf{M} \\
&\equiv \quad [w,x,y] \;\mapsto\; [w,x,y]\begin{bmatrix} m_{ww} & m_{wx} & m_{wy} & m_{wz} \\ m_{xw} & m_{xx} & m_{xy} & m_{xz} \\ m_{yw} & m_{yx} & m_{yy} & m_{yz} \end{bmatrix}
\end{aligned}
$$

O polígono $P^*$ em $\mathbb{T}_2$ pode ser representado por um conjunto não-vazio de listas de pontos de $\mathbb{T}_2$. Cada lista representa uma seqüência circular de vértices e arestas — um *ciclo* — da fronteira do polígono;

os vértices são os pontos da lista, e as arestas são os segmentos abertos que ligam cada vértice ao vértice seguinte. (Supõe-se que vértices consecutivos são distintos e não antípodas.) A ordem dos vértices no ciclo é tal que o interior do polígono fica à esquerda de quem percorre o mesmo na ordem indicada.

Assim, por exemplo, o polígono da figura ?? abaixo pode ser descrito pelas listas de vértices $\{(a,b,c,d),(e,f,g),(h,i,j,k)\}$.

Note que a regra que define o interior e exterior do polígono pressupõe que os ciclos não se cruzam; caso contrário, certas regiões do plano poderiam ficar com estado contraditório — interior segundo uma aresta, exterior segundo outra. Mais ainda, as arestas de um polígono (consideradas como segmentos abertos) devem ser disjuntas duas a duas. Além disso, os ciclos devem estar aninhados e orientados de maneira consistente: intuitivamente, dentro de cada ciclo anti-horário, os ciclos mais "externos" devem ser todos horários, e vice-versa.

Dado um polígono $P$ nesta representação, e um ponto qualquer $q$ do plano, podemos determinar se $q$ está dentro ou fora de $P$, percorrendo alguma reta $m$ que sai de $q$ e encontra a fronteira do polígono. Seja $e$ a *primeira* aresta do polígono encontrada pela reta $m$, depois do ponto $q$; comparando a direção de $e$ com a direção de $m$, podemos decidir se $q$ está dentro ou fora do polígono.

Este algoritmo, simples em princípio, torna-se complicado quando consideramos todos os casos excepcionais que podem ocorrer. Em primeiro lugar, a reta $m$ pode encontrar a fronteira de $P$ num vértice $v$, ou pode conter inteiramente uma ou mais arestas $e$ de $P$. Nesses casos, deve-se examinar as arestas vizinhas a $v$ ou $e$ para decidir se $q$ está dentro ou fora do polígono. Além disso, o ponto $p$ pode estar sobre a fronteira de $P$, caso em que a resposta é inerentemente ambígua — nem dentro, nem fora.

Finalmente, a reta $m$ pode encontrar a fronteira de $P$ num ponto em que dois ou mais vértices coincidem. (Infelizmente, na maioria das aplicações práticas não é possível exigir que todos os vértices sejam distintos). Nesse caso, é necessário examinar todas as arestas do polígono que incidem nesses vértices, para decidir se $q$ está dentro ou fora.

Para cortar o polígono $P$ por um plano $\pi = \langle \mathcal{W}, \mathcal{X}, \mathcal{Y}, \mathcal{Z} \rangle$, é conveniente primeiro transferir esse problema para o plano $\mathbb{T}_2$. Ou seja, devemos determinar a reta orientada $\pi^* = \langle \mathcal{W}^*, \mathcal{X}^*, \mathcal{Y}^* \rangle$ de $\mathbb{T}_2$ que corresponde à intersecção do plano de corte $\pi$ com o plano do polígono $\pi_P$. Um ponto genérico $u = [w, x, y]$ de $\mathbb{T}_2$ está no lado positivo dessa reta se e somente se sua imagem $u\mu_P$ está no lado positivo de $\pi$. Isto é, devemos ter,

$$[wxy] \cdot \langle \mathcal{W}^*, \mathcal{X}^*, \mathcal{Y}^* \rangle^{\mathrm{tr}} = [wxy] \mathsf{M} \langle \mathcal{W}, \mathcal{X}, \mathcal{Y}, \mathcal{Z} \rangle^{\mathrm{tr}} \tag{7.2}$$

para todo $[w, x, y]$; ou seja,

$$\langle \mathcal{W}^*, \mathcal{X}^*, \mathcal{Y}^* \rangle = \langle \mathcal{W}, \mathcal{X}, \mathcal{Y}, \mathcal{Z} \rangle \mathsf{M}^{\mathrm{tr}} \tag{7.3}$$

Uma vez determinada a reta $\pi^*$, precisamos recortar o polígono $P^*$ contra a mesma. Alguns ciclos de $P^*$ serão preservados, por estarem inteiramente no lado positivo de $\pi^*$; outros serão eliminados, por estarem inteiramente sobre $\pi^*$ ou no seu lado negativo. Os demais ciclos, que cruzam a reta, devem ser podados pela mesma; o resultado é uma coleção de cadeias que começam e terminam em $\pi^*$. Estas cadeias precisam ser ligadas entre si, com segmentos de $\pi^*$, de modo a formar novos ciclos. A maior dificuldade é garantir que estes novos ciclos não se cruzam, não tem arestas sobrepostas, e estão corretamente encaixados e orientados; para isso, é necessário atentar para a ordem em que eles entram e saem da reta $\pi^*$. O algoritmo abaixo tenta formalizar um pouco mais estas idéias:

1. Inicialize os conjuntos $R \leftarrow \{\}$, $K \leftarrow \{\}$.

2. Para cada ciclo $C = (v_0, .. v_{m-1})$ de $P^*$, repita:

   2.1. Se todos os vértices de $C$ estão no lado positivo de $\pi^*$, acrescente $C$ ao conjunto $R$. Se nenhum vértice está no lado positivo, ignore $C$. Nos demais casos,

      2.1.1. Elimine todos os vértices, arestas e partes de arestas de $C$ que não estão no lado positivo de $P$. O resultado é uma ou mais cadeias poligonais $T_1, .. T_k$, sendo que cada $T_i$ começa e termina com uma aresta.

      2.1.2. Complete cada $T_i$ com dois vértices, inicial e final. (Esses vértices estão necessariamente sobre a reta $\pi^*$.)

      2.1.3. Acrescente as cadeias $T_i$ ao conjunto $K$.

3. Se o conjunto $K$ está vazio, então

3.1. Determine se a reta $\pi^*$ está dentro ou fora do polígono $P$. Se ela estiver dentro, acrescente ao conjunto $R$ mais um ciclo $(v_0, v_1, v_2)$, onde $v_0, v_1, v_2$ são três pontos de $\pi^*$ tais que os segmentos $v_0v_1$, $v_1v_2$, e $v_2, v_0$ estão bem definidos, são disjuntos dois a dois, cobrem a reta $\pi^*$, e estão orientados no mesmo sentido de $\pi^*$.

4. Caso contrário (o conjunto $K$ não é vazio), faça

4.1. Ordene circularmente o conjunto $K$ pelos vértices iniciais das cadeias, na ordem de ocorrência desses vértices ao longo da reta $\pi^*$. Em casos de empate, ordene as cadeias pelas direções de suas arestas iniciais, no sentido horário.

4.2. Enquanto o conjunto $K$ não estiver vazio,

4.2.1. Retire uma cadeia qualquer $T$ de $K$. Seja $u$ o vértice inicial de $T$.

4.2.2. Enquanto o vértice final de $T$ for diferente de $u$, repita

4.2.2.1. Sejam $e$ e $v$ a última aresta e o último vértice da cadeia $T$.

4.2.2.2. Se houver uma ou mais cadeias em $K$ que começam no vértice $v$, seja $T'$ aquela cuja primeira aresta $e'$ tem direção mais próxima da direção de $e$, quer no sentido horário, quer no anti-horário.

4.2.2.3. Caso contrário, encontre na lista $K$ a primeira cadeia $T'$ cujo vértice inicial $u'$ ocorre depois do ponto $v$ ao longo da reta $\pi^*$. Se houverem várias cadeias em $K$ começando nesse mesmo ponto $u$, escolha aquela cuja aresta inicial $e'$ tem a direção mais anti-horária possível, em relação à direção de $\pi^*$.

4.2.2.4. Retire a cadeia $T'$ de $K$. Se $u'$ não é antípoda de $v$, e o segmento orientado $vu'$ tem a mesma direção da reta $\pi^*$, então emende as cadeias $(v, u')$ e $T'$ no fim de $T$ (ou só $T'$, se $v = u'$). Caso contrário, determine um ponto $w$ de $\pi^*$ que, num percurso da mesma, ocorre depois do antípoda de $u'$ e antes do antípoda de $v$; e concatene as cadeias $(v, w, u')$ e $T'$ no fim de $T$.

4.2.3. Acrescente a cadeia $T$ (que agora é um ciclo) ao conjunto $R$.

5. Devolva o conjunto $R$ como resposta.

Note que este algoritmo roda em tempo $\Theta(n + m \log m)$ no pior caso, onde $n$ é o número total de arestas do polígono $P$, e $m$ é o número de novas arestas Existem algoritmos assintoticamente mais eficientes, mas eles são de utilidade prática duvidosa, pois em cenas típicas a grande maioria dos polígonos tem poucas arestas.

Figura 7.1: O volume da imagem.

Figura 7.2:

Figura 7.3: A *pipeline* de poda.

Figura 7.4: