GERAÇÃO DE CONTORNOS A PARTIR DE CONJUNTOS DE PONTOS NO PLANO

LUIS PAULO BUENO[1,2]
ANTONIO A. F. DE OLIVEIRA[1]
MARIA CÉLIA P. DE FREITAS[1]

[1]COPPE-Sistemas, UFRJ
Caixa Postal 68511
Rio de Janeiro - RJ - 21945

[2]INSTITUTO DE FÍSICA-UFRJ
Caixa Postal 68528
Rio de Janeiro - RJ - 21945

Abstract. Given a set of *N* points in the plane obtained by some process and assumed to be on the boundary of a closed, bounded region, an algorithm is presented to construct in time *O (N logN)* the "best" polygonal contour, in a sense to be precised, which encloses all the given points. Using that algorithm a polygonal approximation of the boundary including all the original points can be obtained.

1. INTRODUÇÃO

Um conjunto finito de pontos no plano pode ser tomado como um primeiro nível de representação da solução de vários problema originados em diferentes campos. No caso das Geociências, seleciona-se pontos, da superfície terrestre, onde através de algum processo de medida, atribuiu-se o valor de um certo parâmetro. Para cada parâmetro constrói-se as suas isolinhas: curvas de nível, se o parâmetro é a altura. Na Medicina, para fornecer um diagnóstico, se utiliza, cada vez mais, de imagens de seções planas, geradas a partir de Tomografia Computadorizada e Ressonância Magnética Nuclear. O tratamento dessas imagens pode levar, como no caso de observações microscópicas, à geração de um conjunto finito de pontos de um plano.

Em uma certa medida, estes exemplos retomam o paradigma do conhecido jogo infantil de ligar pontos numa dada configuração de pontos do plano. Neste caso, um desenho é discretizado, escolhendo-se um conjunto finito de pontos sobre as linhas que o compõem e a cada ponto é atribuído um número. O desenho de linhas ligando os pontos na ordem (crescente) de sua numeração é a estratégia que permite definir arcos de curvas e curvas fechadas, estas últimas caracterizando fronteiras de regiões limitadas do plano, cujo conjunto recria, aproximadamente o desenho de partida, um "objeto" *visualmente* familiar à criança. Quanto maior o número de pontos selecionados melhor será a aproximação final.

Podemos identificar três etapas (ordenadas temporalmente) envolvidas em numerosos processos que incluem o jogo acima descrito:

Preparação: **amostragem**;
Manipulação: **reconstrução**;
Identificação: **reconhecimento**.

Na primeira, a partir de uma dada representação dos objetos de uma cena, um conjunto discreto de pontos nos contornos dos objetos é selecionado .

Na segunda é aplicada uma *estratégia de conversão* de uma representação, em termos de um conjunto finito de pontos, para uma outra em termos de linhas, definindo, no caso de curvas fechadas, fronteiras de regiões limitadas do plano. Esta etapa envolve a construção de uma topologia e de uma geometria, caracterizando um objeto geométrico do plano *compatível* com o universo selecionado.

Na terceira, a comparação do desenho obtido com um "catálogo" já armazenado, pode envolver componentes subjetivas e deve permitir a identificação do "objeto" ou o aprendizado de uma nova forma.

A seleção do conjunto de pontos do "objeto", representando um conteúdo visual potencial, a ser melhor explicitado na etapa de manipulação, deve, na caracterização das adjacências, levar em conta a noção de *proximidade*, ingrediente fundamental de qualquer significado visual [8,9,10].

O paradigma acima caracterizado pode então ser colocado na sua forma mais abstrata, e considerado como *o objetivo mais amplo deste trabalho* [5,8,9]:

"Dado um conjunto de pontos do plano, identificado como o conjunto de nós de um grafo planar, obtido por algum processo de amostragem, estabelecer um estratégia de ligação dos pontos por linhas. Isto é, construir um grafo representando um objeto geométrico que melhor aproxime, segundo algum critério, o "objeto", pertencente a um universo previamente delimitado, de onde os pontos foram selecionados ".

Na sequência vamos caracterizar, para o paradigma estabelecido, os detalhes quantitativos de cada uma das duas primeiras etapas, já que não discutiremos o problema do reconhecimento.

Na seção 2 vamos caracterizar um universo de objetos através de propriedades topológicas e geométricas e o processo de amostragem.

Na seção 3 vamos definir, em termos dos conceitos da Teoria dos Grafos, os grafos aceitáveis como representação dos elementos geométricos do nosso universo.

Na seção 4 vamos caracterizar estratégias de construção do grafo solução, apresentar um algoritmo para a reconstrução de contornos, descrevendo também a estrutura de dados por ele utilizada.

Na seção 5 vamos comentar possíveis extensões desse algoritmo e apresentar exemplos de sua aplicação.

2. O UNIVERSO DE OBJETOS

Vamos supor que os objetos geométricos (*OG*) de nosso universo são subconjuntos do $\mathbf{R}^2$, limitados e com interior não vazio. A nossa representação, neste caso, consiste apenas de pontos arbitrários de sua fronteira, composta de curvas fechadas, sem auto-interseções. Assim, o nosso material primário, um conjunto finito de pontos, será definido por:

$$S = \{ (x_i,y_i) \in \mathbf{R}^2; i = 1, 2, ..., N\}$$

Pelo teorema da curva de Jordan, cada curva fechada divide o plano em duas regiões: uma limitada e a outra ilimitada. Cada componente simplesmente conexa do nosso objeto geométrico é representada pela parte limitada enquanto que as componentes multiplamente conexas serão representadas pela interseção de partes ilimitadas e partes limitadas.

HIPÓTESE

O nosso universo de trabalho (a nossa cena) é constituído de um conjunto de objetos geométricos limitados, de dimensão 2, com interior não vazio: regiões. O processo de amostragem é tal que o conjunto *S* só contem pontos das fronteiras dos OG.

A questão agora transforma-se na busca, a partir de sua representação por um conjunto finito *S*, de um objeto geométrico do $\mathbf{R}^2$ que possui um certo conjunto de atributos topológicos e geométricos, cuja definição vai delimitar o universo de objetos admissíveis. Os atributos topológicos permitem uma primeira classificação e os geométricos um refinamento dentro de cada classe.

A simples informação que um conjunto finito de pontos representa uma curva é "pobre" para a obtenção de uma "boa" reconstrução. Mesmo que consigamos obter a topologia correta, a geometria pode ser muito diferente daquela efetivamente apresentada pela curva. A definição de requisitos para o processo de amostragem, envolvendo a sua resolução e a densidade dos pontos amostrados, permite que o conjunto *S* contenha, implicitamente, informações mais refinadas sobre a geometria do *OG* representado.

Entretanto, neste trabalho, estaremos interessados em construir uma aproximação poligonal da fronteira da "menor" região *R(S)* do plano contendo todos os pontos de *S*, mesmo que a amostragem disponível seja esparsa e, em seguida, tratar os pontos interiores a *R(S)* para obter as fronteiras das subregiões nela incluídas. O caso particular em que exige-se a construção de uma poligonal ortogonal ligando pontos do plano é discutido em [7].

3. O CONJUNTO DE ENTRADA COMO CONJUNTO DE NÓS DE UM GRAFO.

3.1. OS SUBGRAFOS ADMISSÍVEIS <=> AS TOPOLOGIAS ADMISSÍVEIS

O nosso conjunto de entrada ou lista de pares de números reais pode ser visto como o conjunto de vértices $V(G) = S$ de um grafo planar *G*, cujo conjunto de arestas *E(G)* queremos construir, segundo algum critério, e que, neste sentido, dará uma melhor aproximação do objeto geométrico do universo pré-selecionado e portanto um significado ao conjunto *S*

Podemos adotar duas estratégias básicas na construção de *E(G)*. Uma primeira, que chamaremos **construtiva**, onde os pares de vértices vão sendo ligados, obedecendo a um determinado critério, e ao final do processo o grafo construído é a aproximação desejada [5]. Lembremos que foi esta a estratégia usada no jogo de ligar pontos, onde o critério era o da sua numeração. Uma abordagem ingênua levaria à ligação de cada ponto ao seu vizinho mais próximo, o

que entretanto pode dar origem a inconsistências, como na figura abaixo.

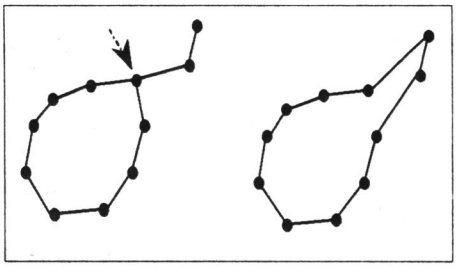


Figura 1: Inconsistência e possível contorno

A outra, que chamaremos **destrutiva**, constrói um *grafo de referência* G^S e em seguida, arestas são identificadas como pertencentes à fronteira, enquanto que outras são substituídas por caminhos que, segundo algum critério, forneçam uma melhor aproximação poligonal para a fronteira do objeto: $G \subseteq G^S$.

Colocado no contexto da Teoria de Grafos e tendo presente a nossa HIPÓTESE, o problema da reconstrução de objetos, do universo selecionado acima, equivale ao da identificação de famílias de ciclos disjuntos em G^S, representações das linhas fechadas que constituem a fronteira do objeto de nosso universo. Em outros termos é preciso determinar uma partição do conjunto $S = V(G)$ em um certo número de subconjuntos disjuntos, onde cada um constitui um ciclo.

3.2. A ADJACÊNCIA CONSTRUÍDA A PARTIR DA PROXIMIDADE.

Na tentativa de caracterizar a maneira pela qual os nossos processos de percepção organizam os dados sensoriais brutos, que atingem os nossos olhos, a escola de psicologia da Gestalt enunciou vários princípios, dentre os quais o mais básico é o da **proximidade**. Em consequência a organização de nossa percepção visual favorece agrupamentos de pontos que apresentam menores distâncias entre eles [10].

Alguns ingredientes importantes, que precisam ser considerados, surgem no exemplo de um conjunto de pontos S do plano selecionados de dois círculos de mesmo raio, cujos centros estão "consideravelmente" separados, como mostra a figura abaixo.

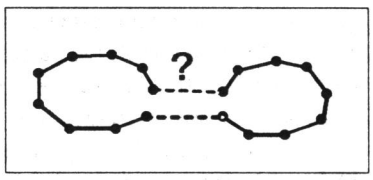


Figura 2: Dois círculos ou um halteres ?

A construção do grafo G com $S = V(G)$ consistindo de um único ciclo, dá origem a uma linha fechada em forma de halteres, contrariando a expectativa do aparecimento de dois ciclos, a melhor aproximação visual para o objeto de partida. Um aspecto relevante do problema, não contemplado nesta formulação, é o da definição dos pontos adjacentes a cada ponto de S. Um critério visual favoreceria a construção dos dois ciclos se os pontos extraídos de um mesmo círculo estivessem mais próximos do que pontos selecionados em círculos diferentes. Em outros termos, pontos de um círculo só poderiam ser adjacentes a pontos do mesmo círculo.

Cabe lembrar que é possível ter-se situações visualmente ambígüas e que, nestes casos, nenhuma estratégia será capaz de resolver tais ambigüidades. Além disso, o uso de coordenadas dos pontos, expressas em termos de aritmética de ponto flutuante pode gerar soluções diferentes das esperadas por inspeção apenas visual.

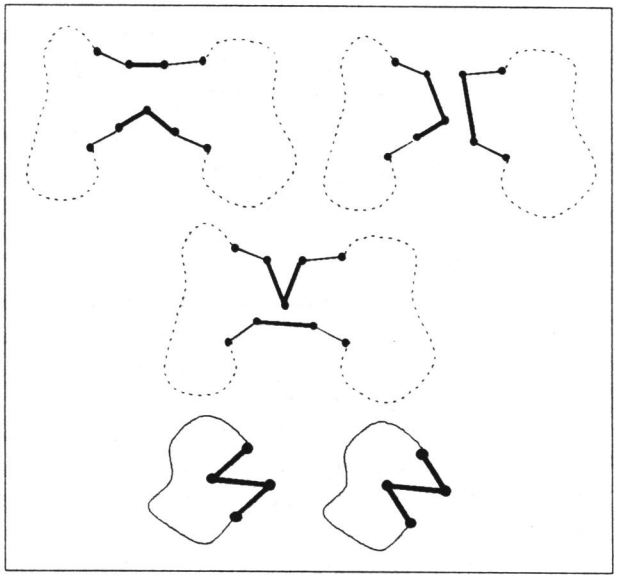


Figura 3 : Ambigüidades

4. UMA ESTRATÉGIA DE RECONSTRUÇÃO

4.1. ESCOLHA DO GRAFO DE REFERÊNCIA

O problema caracterizado nas seções anteriores pode ser formulado como:

PROBLEMA

Dado um conjunto finito de pontos do plano

$S = \{ (x_i,y_i) \in \mathbf{R^2}; i = 1, 2, ..., N\}$

determinar subconjuntos disjuntos S_j, $S_j \subseteq S$, j=1,2,...,ns tais que cada S_j define uma poligonal fechada passando por todos os seus pontos.

Colocado em termos de Teoria de Grafos [2], queremos construir um grafo $G = (V_G, E_G)$ com $V_G = S$ tal que G seja a união disjunta de subgrafos G_j, j = 1,2, ..., ns, $G = \cup_{j=1,...,ns} G_j$ cada um dos quais é um ciclo. A cada subgrafo G_j estará associada à linha de fronteira de uma componente conexa.

Como estratégia adotaremos um enfoque destrutivo, ou seja, definiremos um grafo de referência G^S e a partir deste identificaremos as arestas que comporão cada subgrafo, "descartando" as outras.

Levando em conta a proximidade, como ingrediente básico para a escolha de G^S, vamos escolher o grafo planar definido pela Triangulação de Delaunay de S, notado G_T, como o nosso grafo de referência: $G^S = G_T$.

Esta escolha é motivada ainda pelos seguintes aspectos:

- G_T é o grafo dual do diagrama de Voronoi, a representação consagrada da relação de proximidade entre pontos de um conjunto;
- a existência de algoritmos ótimos para a sua construção;
- o seu caráter maximal, no sentido que podemos considerar a seguinte cadeia de inclusões [5]:

$$G_0 \subset G_1 \subset G_2 \subset G_3 \subset G_4 \subset G_T$$

onde G_0 é o grafo totalmente desconexo definido pelo próprio S, G_1 é o grafo dos vizinhos mais próximos, G_2 é a árvore geradora euclidiana mínima, G_3 é o grafo dos vizinhos relativos e G_4 é o grafo de Gabriel.

Além disso, é particularmente importante, para a estratégia que iremos desenvolver, o fato da fronteira do fecho convexo do conjunto S, notada por G_X, ser também um subconjunto de G_T.

Assim, construído G_T, podemos estabelecer uma heurística de "poda" daquelas arestas que possam ser identificadas como exteriores ou como interiores; as restantes deverão fazer parte da fronteira, a representação desejada para o objeto. Usualmente tem sido adotada esta estratégia quando os objetos são simplesmente conexos e a cena contem apenas um objeto [1]. A necessidade de representar vários objetos de uma cena ou objetos multiplamente conexos exige a capacidade de detetar componentes conexas e para cada uma destas os ciclos do grafo que representam sua fronteira e os seus buracos, se existirem.

Além disso, G_X pode ser visto como a fronteira de um conjunto candidato a solução, que a estratégia de "poda" se encarregará de "erodir", como um processo de "fora para dentro", avançando o contorno inicial até que um "melhor" contorno seja caracterizado.

Observemos que desta forma estaremos exigindo que o processo tenha, a cada etapa, um objeto geométrico invariante: *o contorno corrente*. Além disso, a cada etapa estaremos classificando arestas como interiores, pertencentes ou exteriores ao contorno corrente.

4.2. A ESTRATÉGIA DE RECONSTRUÇÃO

A idéia básica do algoritmo (**A**) é: partindo de G_T como grafo de referência e de G_X como contorno inicial, deformar este contorno, substituindo algumas de suas arestas, por arestas de G_T, até que o contorno resultado seja obtido.

Para isso vamos associar a cada aresta e a cada contorno um custo. O custo de uma aresta poderá ser definido em termos da noção de distância entre pontos de S, segundo vários critérios que explicitaremos mais adiante. **O custo de um contorno será dado pelo custo da maior de suas arestas.**

O processo de alteração do contorno corrente deverá ter como objetivo a *minimização do seu custo* e, para isso, deve ser retirada, a cada etapa, a aresta de maior custo que seja retirável, no sentido que, eliminando-se a aresta, deve ainda ser possível encontrar um novo contorno satisfazendo a:

I) Propriedade $\mathbf{p_1}$: O seu interior deve ter uma única com ponente conexa;
II) O seu custo não deve ser maior que o do contorno corrente;

Evidentemente o processo pára em um número finito de passos, quando não mais houver arestas de G_T retiráveis.

Ao invés de simplesmente eliminar de G_T as arestas, caracterizadas como interiores e exteriores, vamos empregar um esquema de coloração de vértices e arestas que, a cada etapa reflita o estado de evolução do contorno.

Essa evolução é determinada pelo seguinte critério: escolhe-se a aresta de maior custo do contorno e tenta-se retirá-la, desenvolvendo um processo de verificação da possibilidade da aresta ser retirável, que daqui em diante será identificado por *PVR*.

O *PVR* inicia com a retirada de uma aresta do contorno a_{max}, caso isso não acarrete a perda da propriedade $\mathbf{p_1}$, e sua substituição pelas duas outras arestas de Delaunay a ela adjacentes ainda não retiradas. Se alguma dessas arestas tem custo maior que o de a_{max}, repete-se para ela o processo de retirada acima, caso esta retirada não implique na perda de $\mathbf{p_1}$.

O processo continua até que seja encontrado um caminho C ligando os nós extremos de a_{max} cujas arestas têm custo menor que o de a_{max}. Neste caso diz-se que o *PVR* tem resposta *afirmativa*, a_{max} é dita

retirável e substituída pelo caminho. Alternativamente ele pode constatar que a existência desse caminho é impossível, sem violar $\mathbf{p_1}$. Neste caso o *PVR* tem resposta *negativa* e a aresta é dita *não-retirável*. O *PVR* é implementado, no algoritmo descrito mais adiante, pela função recursiva **Sonda.**

Com relação à coloração de vértices e arestas, mencionada acima, adotaremos a seguinte tabela de cores, parte dela dando origem a uma semântica topológica:

Arestas podem assumir as seguintes cores:
- BRANCA - arestas interiores ao contorno corrente, mas que podem vir a fazer parte do contorno em uma etapa posterior. Em particular são BRANCAS as arestas ainda não atingidas pelo algoritmo;
- CINZA - arestas do contorno corrente que ainda não se tentou retirar;
- PRETA - arestas definitivas do contorno. Aquelas que se tentou retirar e não se conseguiu sem aumentar o custo da solução;
- AZUL - arestas exteriores ao contorno, ou seja, as que já foram retiradas;
- VERMELHA - arestas interiores ao contorno, já atingidas pelo algoritmo, que têm custo maior que *todas* as arestas CINZA do contorno corrente e que, portanto, são arestas interiores definitivas;

Além destas, duas outras cores são empregadas pelo algoritmo.
- VERDE - arestas atingidas durante o *PVR* de uma aresta CINZA **e**, que têm custo maior que o de **e**, mas podem ser retiradas e substituídas por outras de custo menor que o de **e** ; se o *PVR* tem resposta afirmativa, estas arestas são tornadas AZUIS (isto é, são retiradas) e caso contrário são tornadas VERMELHAS;
- CREME - arestas atingidas durante o *PVR* de uma aresta CINZA e que têm custo menor que o desta; nelas o *PVR* se detém, isto é, pára a "escavação". Se a resposta do *PVR* é afirmativa as arestas CREME se tornam CINZA e, em caso contrário, voltam a ser BRANCAS (Na realidade uma aresta não é pintada de CREME, mas apenas colocada em uma lista que chamamos de lista de arestas CREME).

Vértices podem assumir três cores:
- BRANCA - vértices que ainda podem ser atingidos na sequência do *PVR* em curso. Em particular os vértices ainda não atingidos pelo algoritmo são BRANCOS;
- CREME - vértices que são origem de arestas CREME (a exemplo do que acontece com as arestas, um vértice não é pintado de CREME, mas apenas colocado na lista de vértices CREME);
- CINZA - vértices do contorno corrente, exceto o vértice de destino da aresta que se está tentando substituir, apenas durante a execução do *PVR* desta aresta;

O algoritmo determina um único contorno externo envolvendo o conjunto de pontos dado. Caso este contorno não passe por todos os pontos, em uma etapa de pós-processamento, deve-se considerar as duas seguintes possibi lidades:
- a existência de contornos internos (buracos);
- que uma solução melhor possa ser obtida, através de contornos disjuntos, formados pela combinação de trechos do contorno determinado pelo algoritmo, com pontos internos;

A etapa de pós-processamento será melhor comentada mais adiante.

Na definição do custo de uma aresta surgem várias alternativas baseadas na noção de distância entre pontos de S, isto é, no comprimento das arestas de G_T. A primeira e mais imediata é dada pelo próprio comprimento da aresta:

$$custo(e) = c(e) = d(P_i, P_j);$$
$$e = (P_i, P_j) \in G_T$$

Uma outra possibilidade, que podemos também utilizar no algoritmo e que melhor expressa a idéia de uma "erosão" a partir do exterior, de um contorno, é dada pelo **maior** dos dois seguintes valores: o raio do círculo circunscrito ao triângulo de Delaunay interior ao contorno e adjacente à aresta e o raio do círculo centrado no ponto médio da aresta e passando pelos seus extremos. Deve-se observar que, neste caso, o custo de uma aresta está definido apenas para as arestas desse contorno e depende do lado onde está o interior do contorno.

Podemos ainda cogitar em utilizar, na definição de custo de uma aresta conceitos como "α-shape" [3], " β-skeleton" [5] e fatores de forma definidos pelo quociente de dois quaisquer custos .

4.3. A ESTRUTURA DE DADOS EMPREGADA

A estrutura "*half-edge*" é uma implementação natural e eficiente de um grafo conexo que modela a topologia de subdivisões planares. Esta estrutura é capaz de produzir todas as possíveis relações de adjacência entre os vértices, arestas e faces do grafo em um tempo ótimo [6].

Associamos as seguintes informações topológicas a cada um dos nós do grafo e que determinam as relações de adjacência entre eles.

Aresta (e)
- semi-arestas esquerda e direita (*e.he1* e *e.he2*)

SemiAresta (he)
- a sua aresta correspondente (*he.e*)
- próxima aresta no ciclo (*he.next*)
- aresta antecessora no ciclo (*he.prev*)
- vértice inicial da semi-aresta (*he.v*)
- face incidente à semi-aresta *he* (*he.f*)

Face (f)
- uma das semi-arestas de sua fronteira (*f.he*)

Vértice (v)
- uma das semi-arestas incidentes ao vértice (*v.he*)

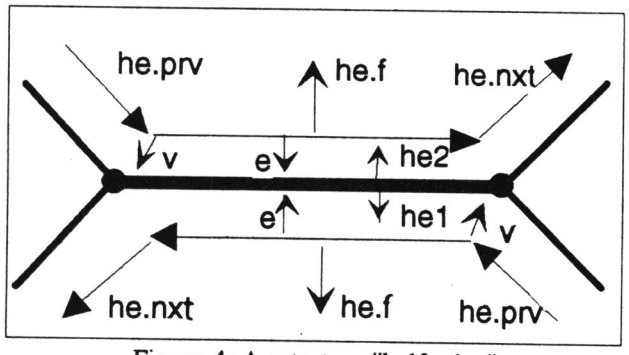


Figura 4: A estrutura "half-edge"

A cada aresta estão associadas 2 semi-arestas, cada uma delas definindo uma das possíveis orientações da aresta. Chamamos de *sim(he)* a semi-aresta de orientação oposta a *he* , isto é, a sua simétrica. Além disso associamos ao nó aresta os campos *cor* e *custo*, necessários ao algoritmo de reconstrução.

A triangulação de Delaunay utiliza esta estrutura e de modo que todas as faces estejam orientadas consistentemente. O algoritmo utilizado para a sua construção está descrito em [4].

4.4. O ALGORITMO A

```
função Esculpe ( S )
{
  Delaunay ( S );
  Pintar todas as arestas e vértices do contorno do
  Fecho Convexo de CINZA;
  Pintar todas as arestas e vértices interiores da
  Tiangulação de Delaunay de BRANCO;
  enquanto (houver arestas CINZA) faça
    Escolher a aresta CINZA de maior custo (a_max);
    Escolher he = semi-aresta de a_max interior ao
    contorno;
    Pintar he.v de BRANCO;
    PVR = Sonda ( he.nxt ,c( a_max ) ) AND*
          Sonda ( he.prv , c(a_max ) )
    Se ( PVR ) então
      Pintar a_max de AZUL;
      Pintar arestas da lista VERDE de AZUL;
      Pintar arestas e vértices da lista CREME de
      CINZA;
    senão
      Pintar a_max de PRETO;
      Pintar cada ramo da lista VERDE de
      VERMELHO;
    Pintar he.v de CINZA;
    Liberar listas de arestas;
}

função Delaunay ( S )
{
  Constrói a Triangulação de Delaunay do conjunto S e
  identifica a fronteira (contorno) do fecho convexo de
  S;
}

função Sonda ( he , c )
{
  a = he.e ;
  Se ( a é colorida ) então
    Retorne FALSO;
  senão
    Se ( sim(he).v é CINZA ou pertence à lista
    CREME) então
      Pintar a de VERMELHO;
      Retorne FALSO;
    senão
      Se ( c( a ) ≤ c ) então
        Coloque a na lista CREME;
        Coloque he.v na lista CREME;
        Retorne VERDADEIRO;
      senão
        Pintar a de VERDE;
        Coloque a na lista VERDE;
        retira = Sonda ( sim(he).nxt ,c ) AND*
                 Sonda ( sim(he).prv , c )
        Se ( NOT retira ) então
          Pintar a de VERMELHO;

        Retorne retira;
}
```

Observações:

O operador booleano *AND** é definido de tal forma que ele só atua sobre o segundo operando se o resultado da aplicação sobre o primeiro for **VERDADEIRO**; caso contrário ele retorna **FALSO**.

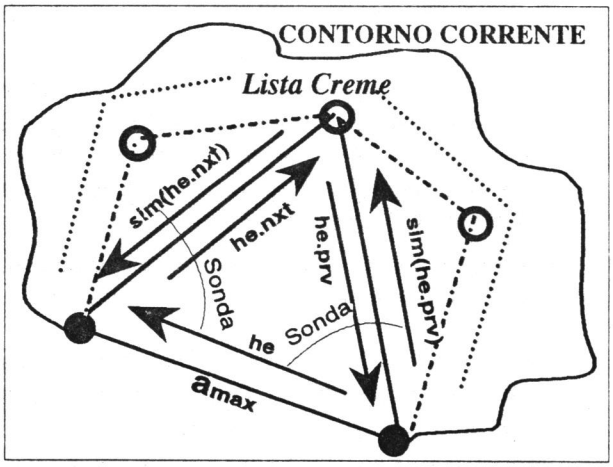


Figura 5: O avanço do contorno

4.5. A DESCRIÇÃO DO ALGORITMO

Como já mencionado, o algoritmo acima procura melhorar o custo do contorno obtido, retirando dele uma única aresta (**e**) de cada vez e substituindo-a, caso seja possível, por um caminho *t* entre seus extremos. Esse caminho é constituído pela sequência de arestas para as quais **Sonda** responde **VERDADEIRO** e vai sendo armazenado na lista CREME. Caso a ligação entre os extremos de **e** se complete, todos os seus vértices e arestas são tornados CINZA, isto é, são colocados no novo contorno, quando se volta à **Esculpe**. Quando é chamada por essa última rotina, **Sonda** tenta inicialmente substituir **e** pelas duas arestas, ainda não retiradas, que com ela formam um triângulo de Delaunay (TD). Caso alguma dessas arestas (**e'**) tenha custo maior que *c*(**e**) então ela tenta substituir **e'** pelas duas outras arestas do outro TD adjacente a ela. E assim, sucessivamente **Sonda** vai se aprofundando no interior do conjunto *S'*, delimitado pelo contorno vigente, sempre mantendo como invariante um caminho entre os extremos de **e**. Eventualmente, esse processo se detem quando ele atinge um vértice do contorno não adjacente a **e** (*sim(he).v* é CINZA) ou *t* deixa de ser um caminho simples (*sim(he).v* já pertence à lista CREME).

Nesses dois casos o algoritmo conclui que não se pode prosseguir o aprofundamento sem desrespeitar a propriedade $\mathbf{p_1}$. O mesmo se pode afirmar se ele atinge uma aresta já do contorno (CINZA ou PRETA).

Se não pode prosseguir sem violar $\mathbf{p_1}$ e se até esse momento **e** não pode ser substituída sem aumentar-se o custo do contorno, então pode-se concluir que essas duas propriedades não podem ser satisfeitas simultâneamente. A partir desse momento, o nível de recursão decai até se voltar à **Esculpe**, por conta do operador *AND**. O algoritmo "desempilha", então uma sequência de arestas consecutivamente adjacentes e de custo maior que *c*(**e**) até voltar a **e**. Estas arestas desempilhadas vão sendo pintadas de VERMELHO, o que servirá para que, futuramente, se possa detetar mais cedo a condição de impossibilidade de se substituir uma aresta do contorno, sem piorar a solução.

As arestas VERMELHAS funcionam como barreiras para o processo de aprofundamento, pela simples razão de que atravessar, durante o *PVR* de **e**, uma aresta pintada de VERMELHA no *PVR* de **e'** faria com que o primeiro *PVR* prosseguisse até chegar a **e'**, quando então se concluiría pela impossibilidade de substituir **e** sem piorar a solução.

O atributo "colorida" no primeiro teste realizado por **Sonda** significa CINZA, PRETO, VERMELHO ou VERDE. Já nos referimos à três primeiras. Esse teste, no que se refere à cor VERDE, evita que o procedimento possa entrar em "loop", com o nível de recursão aumentando inde finidamente, porque ele impede que uma aresta seja substituída duas vezes no mesmo *PVR*.

Quando o controle volta à **Esculpe**, com a resposta que **Sonda** encontrou um caminho para substituir **e**, todos os ramos VERDES pertencem ao interior da região delimitada por **e** e pelo caminho que vai substituí-lo. Com a troca de um pelo outro, essas arestas se tornam externas e portanto devem ser retiradas, isto é pintadas de AZUL. Quando a resposta de **Sonda** é **FALSO** enttão as arestas VERDES são feitas VERMELHAS, pois também elas podem servir como barreiras para os *PVR* posteriores.

A finalidade das listas VERDE e CREME é permitir que **Esculpe** possa mudar a cor de todas as arestas que tenham uma mesma cor, sem precisar per correr todo o conjunto de arestas, perguntando para cada uma se ela tem essa cor.

4.6. JUSTIFICATIVA TEÓRICA

Vamos mostrar que a estratégia empregada pelo algoritmo **A** conduz a um determinado tipo de otimalidade que é explicitado a seguir.

Teorema 1: O algoritmo **A** resolve o problema **P** explicitado abaixo.

Problema **P**: *"Dentre os contornos constituídos unicamente de arestas de Delaunay, encontrar aquele de menor custo e que, satisfaça à propriedade* $\mathbf{p_1}$ *e inclua todos os pontos dados região do* $\mathbf{R^2}$ *por ele delimitada* ".

Esboço da Demonstração

Sejam C e C^*, respectivamente, o contorno indicado pelo algoritmo e uma solução ótima para **P**. Pode-se mostrar que C atende à propriedade $\mathbf{p_1}$. Represente por $S(C)$ o subconjunto de $\mathbf{R^2}$ delimitado por C . Suponha agora que C^* tenha uma aresta $\mathbf{e}^*$ exterior a $S(C)$. Se esta aresta era CINZA no momento em que foi retirada, então como as arestas CINZA são retiradas ou confirmadas (tornadas PRETAS) em ordem decrescente (do custo), temos que $c(C^*) \geq c(\mathbf{e}^*) \geq c(C)$. Se ela era VERDE quando foi retirada, então ela tem custo maior que uma CINZA também retirada no comando imediatamente anterior e podemos obter o mesmo resultado. Assim, em ambos os casos C também é ótimo.

Resta a possibilidade de todas as arestas de C^* estarem em $S(C)$. Tome então um trecho t^* de C^* entre dois pontos a , $b \in C$. Se a e b não são consecutivos em C, isso fatalmente determina que C^* deva ter uma aresta externa de $S(C)$, como mostra a figura abaixo:

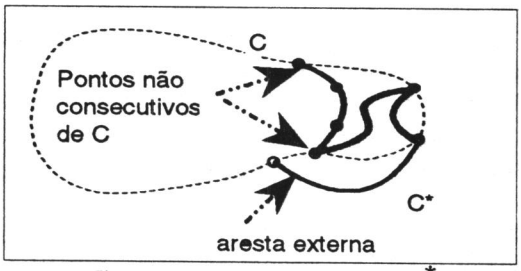


Figura 6: Os caminhos C e C^*

Portanto, em função de nossa hipótese, a e b são consecutivos. Seja **e** a aresta de C que liga a a b e imagine $t^* \neq \mathbf{e}$. Quando **e** é confirmada, existe uma sequência $(\mathbf{e}_k)_k$ formada da seguinte maneira: $\mathbf{e}_0 = \mathbf{e}$, $\mathbf{e}_i$, $i = 1, \ldots, m$ são arestas feitas VERMELHAS no final do *PRV* de **e** e consecutivamente adjacentes; $\mathbf{e}_{m+1}$, adjacente a $\mathbf{e}_m$ é a aresta que determinou o fim da recursão no *PRV* de **e**.

Com relação a esta última aresta temos 3 casos a considerar:

a) $\mathbf{e}_{m+1}$ tem dois vértices no contorno corrente, podendo portanto ter as cores CINZA, PRETO ou VERMELHO.

Nesse caso, como $\mathbf{e}_{m+1} \neq \mathbf{e}$, ela não pode estar na região delimitada pelo caminho $\mathbf{e} \cup t^*$, cujos únicos vértices de C são a e b. Desse modo, alguma das $\mathbf{e}_k$ certamente está em t^*. Como essa $\mathbf{e}_k$ é VERMELHA, isso significa que:
t^* possui uma aresta $\mathbf{e}^*$ tal que $c(\mathbf{e}^*) \geq c(\mathbf{e})$ ***(I)***

b) $\mathbf{e}_{m+1}$ é uma aresta feita VERMELHA no *PRV* de outra aresta $\mathbf{e}'$. Nesse caso, juntando $\mathbf{e}_k$ e a sequência que levou a $\mathbf{e}_{m+1}$ no *PRV* de $\mathbf{e}'$, colocada na ordem inversa, obtemos uma sequência $(\mathbf{e}''_j)_j$ onde $c(\mathbf{e}''_j) \geq c(\mathbf{e})$, $j \geq 1$ e cujo último termo é $\mathbf{e}'$. Aplicando o raciocínio do caso anterior a essa sequência obtemos novamente **I**.

c) $\mathbf{e}_{m+1}$ é VERDE. Nesse caso ela já foi atingida anteriormente no próprio *PRV* de **e** e portanto existe um ciclo na parte final de $(\mathbf{e}_k)_k$. Pode-se mostrar que pelo menos uma aresta desse ciclo não pertence à região delimitada por $\mathbf{e} \cup t^*$, o que nos leva mais uma vez a **I**.

Assim em qualquer dos três casos temos **I**. Incluindo agora os casos em que **e** é comum a C e C^*, temos sempre que, para todo $\mathbf{e} \in C$ existe $\mathbf{e}^* \in C^*$ tal que $c(\mathbf{e}) \leq c(\mathbf{e}^*)$, o que determina a otimalidade de C^*.

Teorema 2: O tempo de execução do algoritmo **A** é $O(N \log N)$.

Demonstração

Acompanhando passo a passo o algoritmo **A** temos que:

Encontrar a triangulação de Delaunay e armazená-la sob a forma de uma estrutura de *half-edge* pode ser feito em $O(N \log N)$ e a partir daí, identificar e pintar de CINZA os nós e arestas do contorno do fecho convexo pode ser feito em $O(N)$. Ordenar essas arestas segundo o seu custo gasta $O(N \log N)$, como também o trabalho global de manter a lista de arestas CINZA ordenada, quando se substituem essas arestas por outras.

Para mostrar que o número de vezes que o algoritmo pinta um aresta ou vértice é linear, observemos inicialmente o diagrama abaixo que representa todas as transições de cor que podem ser efetuadas por ele.

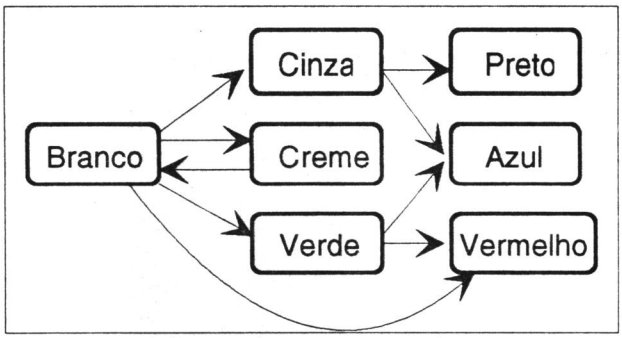


Figura 7: Transições de cores de arestas

Exceto no caso da cor CREME, uma aresta ou vértice só pode ser pintado de uma cor uma única vez. Isto determina que para as outras cores o número de "pinturas" é linear. Com relação à cor CREME, o número de vezes que uma aresta pode ser colocada na lista CREME é cer tamente limitado por 4, dado que uma aresta CREME e é sempre atingida a partir de uma VERDE ou CINZA que não pode ser usada mais de uma vez para isso.

Da mesma forma, para que um vértice seja colocado na lista CREME, é preciso que uma aresta oposta a ele num triângulo de Delaunay se torne VERDE. Assim, o número de entradas de vértices nessa lista está limitado por duas vezes o número de arestas de Delaunay, sendo portanto também linear. Além disso, determinar se um ponto já pertence a essa lista pode ser feito em temp *O (N logN)* se ela for estruturada como árvore-B e portanto o tempo total despendido na gestão da lista de vértices CREME pode ser feito em *O (N logN)* .

Além de pintar vértices e arestas de alguma cor, o algoritmo efetua:

a) Chamadas à rotina **Sonda**, que são efetuadas no máximo duas vezes para cada aresta CINZA ou VERDE, e portanto o seu tempo é também linear;

b) Os testes realizados por **Sonda**, cujo número é da ordem do número de chamadas à rotina;

Isso nos permite finalmente concluir que **A** pode ser executado em tempo *O (N logN)*.

5. CONCLUSÕES E EXTENSÕES

O algoritmo **A** obtem um único contorno conexo, fronteira de uma "menor" região englobando todos os pontos do conjunto S . Dois problemas podem surgir:

(i) Se a **cena** é composta de um único objeto de interior conexo, então, supõe-se que os pontos internos, não atingidos pelo algoritmo, representam buracos e aplica-se **A** novamente a eles. Deve ser observado que as arestas VERMELHAS, que ligam pontos do contorno determinam um particionamento para esses pontos. Assim, ao invés de termos um único novo problema, eventualmente podemos ter um conjunto de problemas menores. A solução global assim obtida pode não ser a melhor, como ilustra a próxima figura.

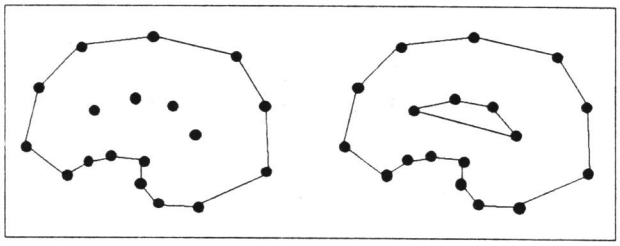


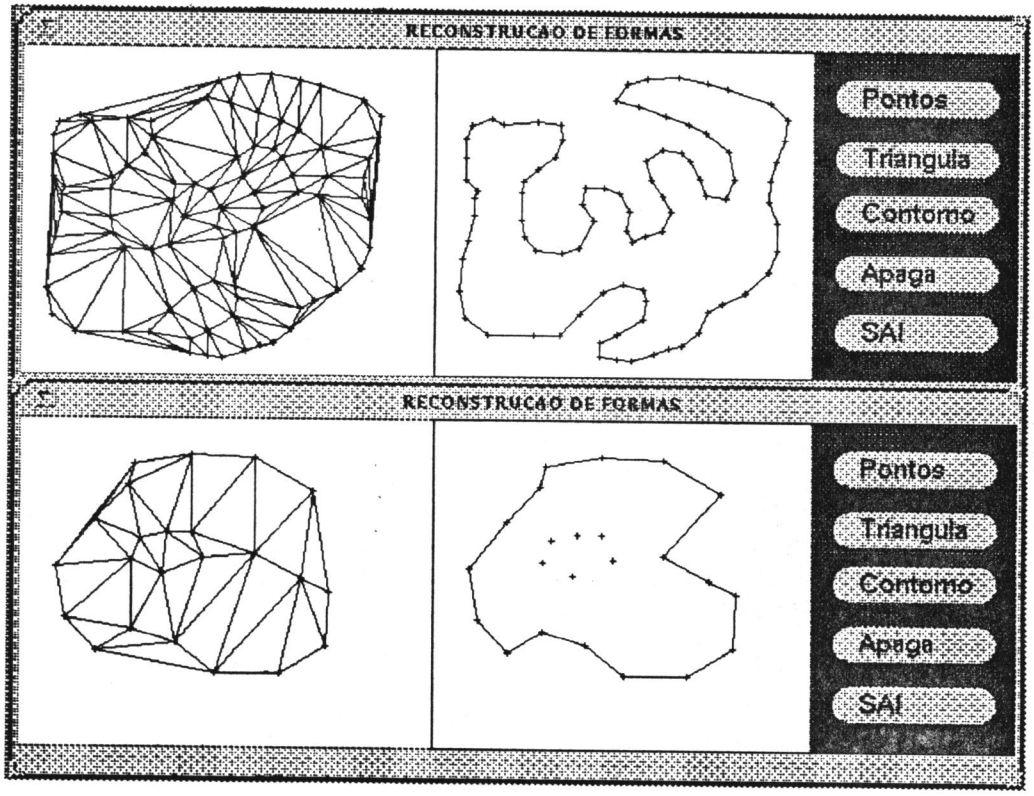
Figura 8: Aplicação repetida de **A**

(ii) A **cena** pode ser composta de vários objetos do tipo acima. Neste caso podemos identificar sequências de pontos próximos ao contorno obtido por **A**, cada uma delas pertencentes a um mesmo objeto. Tomando cada uma dessas sequências como um núcleo, um procedimento de identificação de "clusters" pode ser utilizado para particionar o conjunto de pontos dado. Um exemplo de tal procedimento, onde a árvore geradora euclidiana mínima é empregada, é descrito em [10]. Esta estratégia é particularmente adequada pois o nosso grafo de referência $G^S = G_T$ contem tal árvore como subgrafo: $G_2 \subset G^S$.

Estes problemas evidenciam uma limitação do algoritmo: **A** não é capaz de alterar a topologia do contorno corrente, isto é, criar buracos ou componentes conexas, o que é deixado para uma etapa de pós-processamento.

O procedimento apresentado neste trabalho pode ser estendido para tratar o caso em que, ao invés de um conjunto de pontos, temos um conjunto de ares tas obtidas, por exemplo, por um processo de vetorização aplicado a uma imagem. Nesse caso, ao invés de usar como grafo de referência uma triangulação de Delaunay simples, usaríamos uma triangulação de Delaunay restrita, onde as arestas dadas fossem obrigatórias.

O algoritmo **A** foi implementado em ambiente Unix (SPARCstation 2), usando-se a linguagem C e alguns exemplos podem ser vistos a seguir.



6. BIBLIOGRAFIA

1 - BOISSONAT, J. D. - " Geometric Structures for Three-Dimensional Shape Representation " - ACM Trans. on Graphics, vol. 3(4),(1984), 266-286

2 - BONDY, J. A., MURTY, U. S. - " Graph Theory with Applications " - North Holland 1980

3 - EDELSBRUNNER, H., KIRKPATRICK, D. G., SEIDEL, R. - " On the Shape of a Set of Points in the Plane " - IEEE Trans. on Inf. Theory, vol.IT-29(4), (1983), 551-559

4 - GUIBAS, L., STOLFI, J. - " Primitive for Manipulation of General Subdivisions and the Computation of Voronoi Diagrams " - ACM Trans. Graphics, vol. 42 (2),(1985), 74-123.

5 - KIRKPATRICK, D. G., RADKE, J. D. - " A Framework for Computational Morphology " in " Computational Geometry ", TOUSSAINT, G. T. (ed.), North- Holland 1985.

6 - MANTYLA, M. - " An Introduction to Solid Modeling " - Computer Science Press Inc., Helsinki 1988

7 - O'ROURKE, J. - " Uniqueness of Orthogonal Connect-the-Dots " in " Computational Geometry ", TOUSSAINT, G. T. (ed.), North- Holland 1985.

8 - RADKE, J. D. - " On the Shape of a Set of Points " in " Computational Morphology ", TOUSSAINT, G. T.(ed.) - North Holland 1988

9 - TOUSSAINT, G. T. (ed.) - " Computational Morphology. A Computational Approach to the Analysis of Form " - North-Holland 1988

10 - ZAHN, C. T. - " Graph Theoretical Methods for Detecting and Describing Gestalt Clusters " - IEEE Trans. on Computers, vol.C-20(1),(1971),68-86