Representação BSP de Curvas Implícitas 2D

Francisco Morgado
Dept. Informática, ESTV-IPV
3500 Viseu
fmorgado@di.estv.ipv.pt

Abel Gomes
Dept. Informática, UBI
6200-001 Covilhã
agomes@di.ubi.pt

Sumário

Os sistemas gráficos actuais incluem primitivas para desenhar segmentos de recta, circunferências, curvas e superfícies de Bézier, NURBS (Non-Uniform Rational B-Splines). No entanto, estes sistemas não fornecem primitivas gerais para representar graficamente curvas implícitas. A razão fundamental para este estado-de-coisas prende-se com o facto destas curvas poderem apresentar singularidades (e.g. auto-intersecções) . Este artigo introduz um algoritmo genérico, eficiente e robusto que permite representar qualquer curva analítica definida implicitamente. Para isso, é utilizado um algoritmo BSP (Binary Space Partition) que particiona recursivamente o espaço ambiente $\Omega \subseteq \mathbf{R}^2$ duma forma não-uniforme de modo a determinar um conjunto de pontos que constituem uma aproximação discreta da curva. Cada ponto não é mais do que a intersecção da recta de bissecção dum subespaço com a curva, sendo determinado através dum algoritmo de aproximação numérica. Não são utilizadas quaisquer técnicas de diferenciação. Ao contrário doutros algoritmos de decomposição, este algoritmo permite também determinar pontos isolados, utilizando o conceito de extremo local de uma função num dado intervalo.

Palavras-chave

Curvas implícitas, BSP, aproximação numérica .

1. INTRODUÇÃO

Neste artigo, considera-se o problema da representação de curvas analíticas definidas implicitamente em $\mathbf{R}^2$. Por outras palavras, pretende-se calcular uma aproximação poligonal da curva $\boldsymbol{C}=\{(x,y)\in\Omega\subseteq\mathbf{R}^2: f(x,y)=0\}$ definida pelo conjunto de zeros duma função analítica f de $\mathbf{R}^2$ em $\mathbf{R}$. Refira-se que o algoritmo proposto neste artigo não se aplica somente a curvas algébricas como em [Blinn82], mas também às curvas analíticas.

Basicamente, existem três categorias de algoritmos para representar curvas implícitas:

- *Conversão da representação*. Estes algoritmos convertem uma curva implícita numa curva paramétrica [Allgower91], [Bloomenthal88], [Lorensen87]. Contudo, raramente existe uma parametrização *global* para uma curva implícita. O que é garantido é que existe uma parametrização *local* na vizinhança de qualquer ponto regular da curva, i.e. $f(x,y)=0$ e $\nabla f(x,y)\neq 0$. Mas, existindo uma parametrização global da curva, utiliza-se a sua representação paramétrica para desenhar a curva.
- *Perseguição ou continuação da curva*. A ideia geral deste tipo de algoritmos é calcular um ponto da curva à custa do ponto anteriormente calculado [Chandler88], [Moller95]. Para isso, efectua-se o cálculo do vector Hamiltoniano $\left(-\frac{\partial f}{\partial y},\frac{\partial f}{\partial x}\right)$ combinado com o método denominado por Newton Corrector [Allgower90]. Estes métodos são atractivos, pois concentram o esforço de processamento onde é necessário, adaptando-se também à variação da forma local da curva. No entanto, há sempre o risco de nem todas as componentes da curva serem desenhadas. Para que todas as componentes da curva sejam desenhadas temos de calcular um ponto inicial em cada uma delas, o que nem sempre é possível.
- *Subdivisão do espaço*. Estes algoritmos fazem uma subdivisão do espaço em subespaços de menor área, eliminando imediatamente os subespaços onde não existe qualquer segmento da curva [Bloomenthal88], [Bloomenthal94], [Triquet01]. Contudo, as técnicas de subdivisão não têm sido muito atractivas, pois o processamento intensivo dos algoritmos de subdivisão tornam lentas muitas aplicações [Snyder92]. Recentemente [Lopes01] apresentou uma técnica usando aritmética intervalar que permite aumentar o desempenho de representação da curva.

Este artigo apresenta uma técnica de subdivisão para curvas implícitas no plano. O algoritmo adapta-se à curvatura da curva, efectuando mais subdivisões onde a curvatura muda notoriamente. O algoritmo permite também a

determinação de pontos isolados da curva. Um ponto isolado é determinado através de um processo de aproximação (apoiado no cálculo dos extremos relativos de *f*) que converge para o tal ponto isolado.

O artigo está organizado em secções. Na secção 2 descreve-se o método de partição BSP não-uniforme, a estrutura BSP usada, assim como o modo como são determinados os pontos isolados. Na secção 3 é descrita a continuação dos pontos na estrutura BSP, de modo a desenhar correctamente a curva. Na secção 4 apresentam-se alguns resultados experimentais relevantes. O artigo termina com as conclusões na secção 5.

2. PARTIÇÃO BINÁRIA NÃO-UNIFORME

2.1 Estrutura de dados BSP

A Figura 1 ilustra o método de partição binária não-uniforme do espaço ambiente $\Omega \subseteq \mathbf{R}^2$ onde se encontra parte ou a totalidade da curva implícita $\boldsymbol{C}$. Este método divide recursivamente o espaço Ω em dois novos subespaços Ω_{left} e Ω_{right} através de uma recta l que intersecta a curva. A recta l é denominada por recta de bissecção (ou recta BSP). Se, porventura, um subespaço contém um segmento da curva, então será novamente subdividido em dois subespaços, a não ser que a distância entre os pontos finais do segmento da curva seja menor ou igual a uma pequena tolerância ξ, ou que a curvatura ao longo do segmento da curva não varie duma forma significativa. Os subespaços onde não existe qualquer segmento da curva são desde logo abandonados.

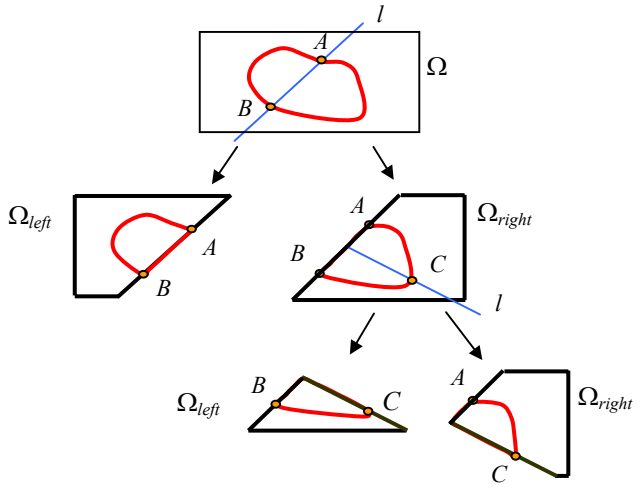


Figura 1: Árvore BSP

A correspondente estrutura de dados é uma árvore binária, designada por árvore BSP. Em termos de código C++, cada nó da árvore BSP é codificada do seguinte modo:

```
class BSP_Node {
    List *fr;
    List *lip;
    BSPline *l;
    BSP_Node *left, *right;
    BSP_Node *next;
}
```

A variável `fr` identifica a fronteira Fr(Ω) de um subespaço convexo Ω. A fronteira dum subespaço consiste numa lista de segmentos de recta que o delimitam; `lip` é a lista de pontos resultante da intersecção Fr(Ω)∩***C***; `l` é a recta que subdivide Ω em dois novos subespaços, Ω_{left} e Ω_{right}; `left` identifica Ω_{left} enquanto que `right` identifica Ω_{right}; `next` é um ponteiro que permite a travessia directa das folhas da árvore, o que permite visualizar ***C*** mais rapidamente.

2.2 Recta BSP

A determinação da recta de bissecção (ou recta BSP) l envolve os seguintes cálculos no subespaço a subdividir:

- Determinam-se os pontos de intersecção Fr(Ω)∩***C***. Caso a intersecção seja nula, são geradas várias rectas até que uma delas intersecte a curva;
- Escolhem-se os pontos mais afastados P e Q do passo anterior;
- Determina-se a mediatriz l do segmento $\overline{PQ}$.

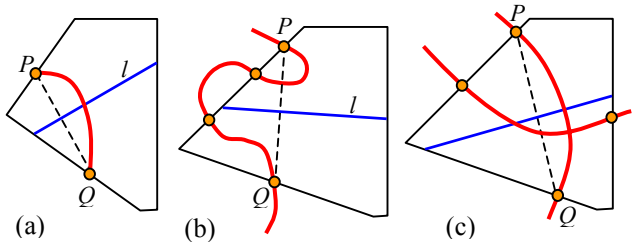


Figura 2: Determinação da recta BSP

A recta BSP é precisamente a mediatriz do segmento de recta $\overline{PQ}$ (Figura 2). A ideia subjacente à escolha dos pontos mais afastados, P e Q, é permitir subdividir o espaço o mais equitativamente possível. Deste modo, a árvore BSP resultante terá tendencialmente menos nós. De facto, através deste critério de subdivisão há a tendência para obter mais rapidamente segmentos da curva quase lineares com um menor número de subdivisões.

O correspondente algoritmo para determinar a recta BSP, poderá ser escrito do seguinte modo:

ALGORITMO 1 (BSPLINE)
INPUT:
(a) Ω: subespaço de R^2
OUTPUT:
(a) *l*: recta BSP

Begin

1. $lip \leftarrow Fr(\Omega) \cap C$
2. ***if*** $(!lip)$
   - $l \leftarrow$ recta arbitrária que passa pelo centro de Ω e intersecta ***C***

   else
   - determinar os pontos *P* e *Q* mais afastados de *lip*
   - $l \leftarrow$ mediatriz de $\overline{PQ}$
   - ***if*** $(d(P,Q) < \xi)$ **return** NULL
   - ***if*** $(d(P,Q) < \tau)$ e $(\#(Fr(\Omega) \cap \boldsymbol{C}) == 2)$
     - $R \leftarrow l \cap \boldsymbol{C}$
     - if $(\angle(\overline{RP}, \overline{RQ}) \approx 180^o)$ **return** NULL
3. ***return*** *l*

End

O algoritmo de partição binária não depende da existência de singularidades na curva (e.g. auto-intersecções). De facto, se um subespço contiver uma auto-intersecção, a sua partição recursiva tende a convergir para tal singularidade. Terminada a partição recursiva do espaço inicial, as singularidades estarão certamente nos nós ou subespaços terminais da árvore BSP.

Um subespaço Ω contém uma auto-intersecção se o número de pontos da curva na sua fronteira for maior ou igual a 3, i.e. $\#(Fr(\Omega) \cap \boldsymbol{C}) \geq 3$. Contudo, para os nós intermédios esta condição não é válida. Por exemplo, na Figura 2(b), a curva intersecta a fronteira de Ω em quatro pontos, mas não existe qualquer auto-intersecção da curva em Ω.

Note-se que os subespaços terminais são relativamente pequenos uma vez que satisfazem a condição $d(P,Q) < \xi$, i.e. a distância entre os dois pontos mais afastados $P,Q \in Fr(\Omega) \cap \boldsymbol{C}$ *é* menor que ξ. Esta condição é o critério de paragem do algoritmo de partição binária recursiva. Nestas condições, assume-se que uma auto-intersecção dum subespaço terminal é o ponto médio de $\overline{PQ}$. Este critério funciona bastante bem mesmo em casos degenerados em que a distância entre *P* e *Q* é muito pequena, mas o comprimento do arco pode ser significativo como acontece com, por exemplo, a curva $y\text{-}\sin(1/x)=0$.

O segundo critério de paragem tem a ver com o controlo da curvatura. Este critério é a conjunção das condições $d(P,Q) < \tau$ e $\#(Fr(\Omega) \cap \boldsymbol{C}) == 2$ implementadas no algoritmo BSPLINE, onde $\tau=3\xi$. Neste caso, se o ângulo $\angle(\overline{RP}, \overline{RQ}) \approx 180^o$, a partição do subespaço termina, i.e. nenhuma recta BSP é calculada, pois os pontos *R*, *P* e *Q* são colineares a menos de uma pequena tolerância. (O ponto *R* é o ponto de intersecção entre a mediatriz *l* e a curva ***C***). Caso contrário, a recta BSP, previamente calculada, é retornada e o processo de subdivisão continua.

2.3 Posicionamento dos pontos da curva em relação à recta BSP

Antes de subdividir o subespaço Ω em dois novos subespaços Ω_{left} e Ω_{right} através da recta *l*, terá de ser feita a classificação dos pontos da curva que se encontram na fronteira de Ω de modo a decidir quais destes pontos pertencem a Ω_{left} e a Ω_{right}.

Seja *P* um ponto da recta *l* (Figura 3). O algoritmo de classificação poderá ser escrito do seguinte modo:

- Determina-se o vector unitário $\vec{u}$ em *P* com a direcção de *l*;
- Determina-se o vector unitário $\vec{w}$ perpendicular a Ω (em $\mathbf{R}^3$);
- Determina-se o vector unitário $\overrightarrow{PX}$ para todo o ponto *X* que intersecta a fronteira de Ω;
- Se o produto misto $(\overrightarrow{PX} \times \vec{u}).\vec{w} > 0$, o ponto *X* pertence ao subespaço Ω_{left};
- Se o produto misto $(\overrightarrow{PX} \times \vec{u}).\vec{w} < 0$, o ponto *X* pertence ao subespaço Ω_{right}.

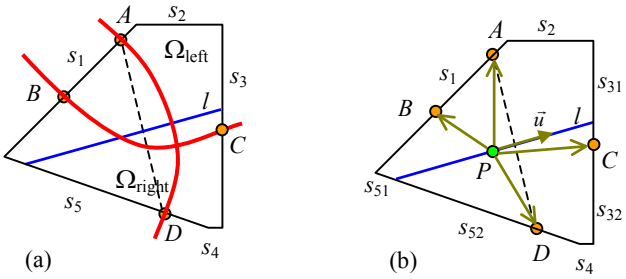


Figura 3: Classificação dos pontos da curva em relação à recta BSP

Na Figura 3, os pontos *A* e *B* pertencem ao subespaço Ω_{left}, enquanto que *C* e *D* pertencem a Ω_{right}. Se o produto misto $(\overrightarrow{PX} \times \vec{u}).\vec{w} = 0$, o ponto *X* pertence ao subespaço Ω_{left} e também ao subespaço Ω_{right}. Isto acontece quando *X* (e.g. *P* na Figura 3(b)) é um ponto de intersecção entre a curva e a recta BSP.

2.4 Criação e actualização dos subespaço Ω_{left} e Ω_{right}

Após a classificação dos pontos de $Fr(\Omega) \cap \boldsymbol{C}$ nos novos subespaços Ω_{left} ou Ω_{right}, temos de redefinir as fronteiras destes novos subespaços. Isto é ilustrado na Figura 3, onde $Fr(\Omega)=\{s_1,s_2,s_3,s_4,s_5\}$ é a fronteira do subespaço convexo Ω em $\mathbf{R}^2$ e *l* é a recta que intersecta Fr(Ω) em

dois pontos. A primeira intersecção subdivide s_3 em dois novos segmentos, s_{31} e s_{32}, enquanto que o segundo subdivide s_5 em s_{51} e s_{52}. Estas duas intersecções originam o segmento $s_l \subset l$ que subdivide Ω em $\Omega_{right}=\{s_l,s_{32},s_4,s_{52}\}$ e $\Omega_{left}=\{s_l,s_{51},s_1,s_2,s_{31}\}$.

A criação destes dois novos subespaços requer também a classificação dos seus segmentos fronteiros em relação à recta *l*. A classificação dos segmentos é baseada no processo de classificação dos pontos descritos na subsecção anterior. Se o produto misto $(\overrightarrow{PX}\times\vec{u}).\vec{w}$ é maior ou igual a 0 para os dois pontos que definem um dado segmento, então este segmento fará parte da fronteira do subespaço Ω_{left}, mas se for menor que 0, então o segmento pertence à fronteira de Ω_{right}.

O correspondente algoritmo de criação e actualização dos subespaços é então:

ALGORITMO 2 (SUBSPACES)

INPUT:

(a) l : recta de partição de Ω

(b) Ω : subespaço de $\mathbf{R}^2$

OUTPUT:

(a) Ω_{left}: $\Omega_{left} \subset \Omega$

(b) Ω_{right}: $\Omega_{right} \subset \Omega$

Begin

- $\Omega_{left} \leftarrow \varnothing$
- $\Omega_{right} \leftarrow \varnothing$
- Determinar os dois pontos de intersecção $Fr(\Omega)\cap l$
- Classificar os pontos $Fr(\Omega)\cap C$ e $Fr(\Omega)\cap l$
- Actualizar fronteira de Ω_{left}
- Actualizar fronteira de Ω_{right}

End

2.5 Intersecção entre a curva e recta BSP

O método da secante é um método de procura de zeros de funções [Akai94]. Basicamente, este método parte de duas estimativas distintas *A* e *B* para a solução de $f(x)=0$ (veja-se Figura 4).

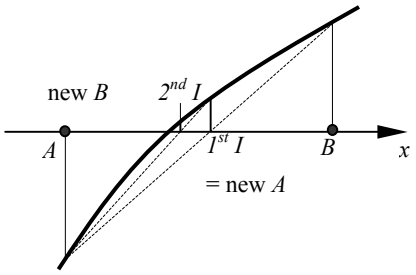


Figura 4: Representação do método da secante.

O processo é iterativo e envolve interpolação linear, sendo actualizados *A* ou *B* em cada iteração. A interpolação é dada pela fórmula

$$I = B - f(B)\frac{B-A}{f(B)-f(A)}$$

onde *I* é a próxima estimativa que aproxima o ponto de intersecção.

O método da secante faz parte do algoritmo para a determinação da intersecção entre a curva $\boldsymbol{C}$ e a mediatriz *l* do subespaço. Inicialmente, a mediatriz é subdividida em vários segmentos mais pequenos. Depois, a cada segmento de recta é aplicado o método da secante se for satisfeita a condição $f(A).f(B)<0$. Esta condição garante que neste segmento existe ponto de intersecção $\boldsymbol{C} \cap l$. A divisão da mediatriz em pequenos segmentos é necessária, pois pode existir mais do que um ponto de intersecção entre a mediatriz e a curva *C*.

Seja MAXLENGTH o comprimento máximo admissível para a mediatriz, i.e. o comprimento da diagonal do espaço inicial Ω, e MAX=10 o número máximo de subdivisões da mediatriz inicial. O número de subdivisões da mediatriz é adaptativa e depende do comprimento da mesma. Seja LENGTH o comprimento da mediatriz. Se LENGTH ≤ (MAXLENGTH / MAX), então aplica-se o método da secante à própria mediatriz (pois esta já é relativamente pequena); caso contrário, divide-se a mediatriz em NSEG segmentos mais pequenos, onde NSEG é dado por

$$NSEG = \frac{MAX \times LENGHT}{MAXLENGTH}$$

sendo depois aplicado o método da secante a cada segmento.

2.6 Determinação de pontos isolados

Por definição, um *ponto isolado* duma curva é todo o ponto *P* para o qual não existe variação de sinal de $f(x)$ numa pequena vizinhança centrada em *P*.

A determinação de pontos isolados é feita nos subespacos terminais da árvore BSP. A ideia base consiste em determinar um ponto da fronteira de Ω onde existe variação da monotonia de *f*. Note-se que existe variação de monotonia em extremos locais de *f*. Normalmente, esta variação de monotonia é calculada através do cálculo da derivada. No entanto, dada a restrição auto-imposta de não usar técnicas de derivação, a variação de monotonia é aproximada pela variação de sinal da taxa de variação média (TVM) entre dois pontos, *A* e *B*:

$$TVM_{[A,B]} = \frac{f(B)-f(A)}{B-A}.$$

Note-se que o processo de determinar um ponto isolado não poderá ser nunca um método baseado no método da

secante, pois este pressupõe que existe sempre variação de sinal de *f* no intervalo que estamos a considerar. Ora, por definição de ponto isolado, essa variação de sinal de *f* não existe numa vizinhança do ponto isolado.

Portanto, o algoritmo de determinação dum ponto isolado começa por determinar o ponto extremo (em valor absoluto) de *f* num dos segmentos fronteiros dum subespaço terminal. Depois, calcula-se o segmento de recta perpendicular ao segmento anterior no ponto extremo. Repete-se este processo de calcular novo ponto extremo e novo segmento perpendicular ao segmento anterior até atingir o ponto mínimo em valor absoluto (ou seja, o ponto isolado) que é um zero da função.

Este processo de determinação do ponto isolado consiste basicamente em determinar a sequência de pontos P_0, P_1,...,P_n, sendo que os vários P_i calculados são os pontos mínimos relativos de *f* nos vários segmentos gerados. Este processo poderá ser interpretado como o percurso dos pontos mínimos relativos até chegar ao ponto isolado, se ele existir (Figura 5).

O algoritmo para determinar um ponto isolado será então:

ALGORITMO 3 (ISOLATEDPOINT)

INPUT:

(a) Ω: subespaço terminal de $\mathbf{R}^2$

(b) *S*: segmento fronteiro de Ω

OUTPUT:

(a) P_i: ponto isolado

Begin

- $P_e \leftarrow$ PONTOEXTREMO *(S)*
- $P_i \leftarrow$ NOVOPONTOEXTREMO *(P_e, S, Ω, 5)*
- ***if*** *(P_i)* **return** P_i
- **return** *NULL*

End

O algoritmo para determinar o ponto candidato é o seguinte:

ALGORITMO 4 (PONTOEXTREMO)

INPUT:

(a) *S*: segmento de recta pertencente a Fr(Ω)

OUTPUT:

(b) P_e: ponto extremo

Begin

Percorrer o segmento de recta S, e determinar o ponto onde existe variação da $TVM_{[A,B]}$, onde $A, B \in S$ e não existe variação de sinal de *f*.

End

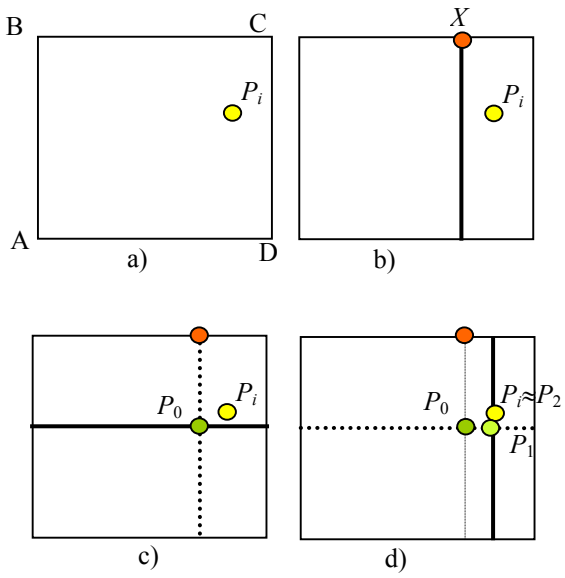


Figura 5: Determinar ponto isolado.

ALGORITMO 5 (NOVOPONTOEXTREMO)

INPUT:

(a) P_e: ponto extremo

(b) *S*: segmento fronteiro de Ω ao qual pertence P_e

(c) Ω: subespaço terminal de $\mathbf{R}^2$

(d) N: nível de recursividade

OUTPUT:

(a) P_i: ponto isolado

Begin

- ***if*** ((N=0)∨(P_e = NULL)) **return** *NULL*
- ***if*** ($|f(P_e)| \leq TOL$) **return** P_c
- $S \leftarrow$ segmento contendo P_e e $\perp S$ em Ω
- $P_e \leftarrow$ PONTOEXTERMO(*S)*
- **return** NOVOPONTOEXTREMO *(P_e, S, Ω, N-1)*

End

3. CONTINUAÇÃO E VISUALIZAÇÃO DA CURVA

Para representar a curva, é necessário ter um processo que permita atravessar a árvore BSP e estabelecer uma sequência correcta dos seus pontos. Pela Figura 1, torna-se claro que só os nós terminais têm os pontos que interessam representar. Os nós intermédios servem somente para controlar o processo de subdivisão do espaço. Deste modo, para efeitos de visualização, interessa somente percorrer os nós terminais duma forma sequencial, de modo que o processo de visualização da curva seja equivalente a percorrer uma lista ligada.

Desta forma, há um ganho que é tendencialmente 50% na representação da curva ***C*** em relação à travessia da árvore BSP na sua totalidade, visto que uma árvore BSP de altu-

ra N tem 2^{N+1}-1 nós, ao passo que a lista dos nós terminais tem somente 2^N nós (cerca de 50% dos nós da BSP).

Deste modo, o algoritmo geral para representar graficamente a curva implícita será o seguinte:

ALGORITMO 6 (CURVE)
INPUT:
(a) ***C***: a curva implícita
Begin
1. $\Omega \leftarrow$ região rectangular de $\mathbf{R}^2$
2. BSP(Ω, 7)
3. Visualizar ***C***
End

O sub-algoritmo BSP (passo 2) é a parte principal de todo o algoritmo, sendo descrito do seguinte modo:

ALGORITMO 7 (BSP)
INPUT:
(a) Ω: subespaço de $\mathbf{R}^2$
(b) d : nível de recursividade
Begin
1. ***if*** (d=0) **return**
2. *l* $\leftarrow$BSPLINE(Ω)
3. ***if*** (*l*)
   - SUBSPACES(*l,* Ω, Ω_{left}, Ω_{right})
   - BSP(Ω_{left}, d - 1)
   - BSP(Ω_{right}, d - 1)

   else *// estamos perante um subespaço terminal*
   - ISOLATEDPOINT(Ω)

End

Note-se que o terceiro critério de paragem é dado pelo nível de recursividade (ou grau da árvore BSP). Para os testes realizados, assumiu-se que o nível máximo de recursividade era 7.

4. RESULTADOS EXPERIMENTAIS

O algoritmo apresentado mostrou-se relativamente rápido, mesmo tendo em conta que é um algoritmo de BSP. Curiosamente, o algoritmo é mais rápido em segmentos da curva com maior variação de curvatura. Isto resulta do facto de os subespaços terminais se tornarem rapidamente mais pequenos em curvas com grande oscilação da curvatura, tornando a procura das soluções mais rápida.

As curvas apresentadas na Figura 6 foram desenhadas com a precisão de $\xi = 10^{-3}$ e com o nível máximo de recursividade na árvore BSP igual a 7. A figura de mérito *fcpp* (acrónimo de *function evaluations per point*) refere-se ao número de vezes que uma função é calculada para que seja determinado com precisão um ponto da curva.

Na Figura 6 mostram-se várias curvas é respectivas subdivisões espaciais. Na Figura 6(a) está representada uma aproximação à curva (x-1)(y-1)=0 através dum conjunto finito de pontos. Esta curva pode ser vista como a união das rectas perpendiculares x-1=0 e y-1=0 em $\mathbf{R}^2$.

Na Figura 6(b) e (c) estão representadas duas curvas, cada uma das quais tem um ponto isolado. A curva da Figura 6(d) tem três componentes, ao passo que as restantes curvas da Figura 6 têm duas componentes, mas nenhuma delas tem pontos isolados.

Note-se que o tempo para desenhar cada curva (e respectiva subdivisão do espaço ambiente) é uma fracção pequena de um segundo, o que é bastante razoável para algoritmos de subdivisão espacial. Isto deve-se em parte, por comparação com outros algoritmos de subdivisão, ao menor número de vezes que a função é calculada para cada ponto que é determinado, i.e. o valor de *fcpp* é menor. Este valor permite, de algum modo, avaliar a eficiência do algoritmo; por exemplo, em [Lopes01], esse valor é cerca de 23. Os resultados dos testes apresentados na Figura 6 foram realizados num PC com processador Intel Pentium 500MHz, 128MB RAM e com o sistema operativo Windows NT.

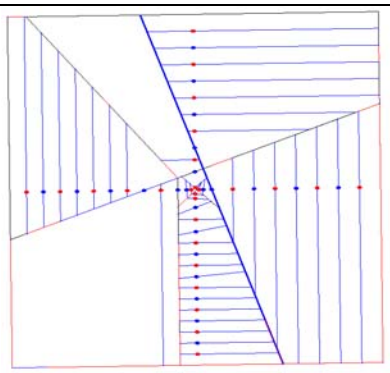
Os sistemas de software comerciais como, por exemplo, o Maple e o Mathematica não incorporam algoritmos precisos e correctos para representar curvas implícitas. Em geral, são capazes de representar parcialmente curvas implícitas, mas falham claramente na representação das suas singularidades (e.g. auto-intersecções). Por exemplo, o ponto P=(1,1) de auto-intersecção da curva (x-1)(y-1)=0, representada na Figura 6(a), não é representável naqueles sistemas comerciais. Os pontos isolados da curvas representadas em (b) e (c) da Figura 6 também não são detectáveis pelos referidos sistemas.

Note-se que, em geral, os algoritmos de subdivisão do espaço permitem determinar uma boa aproximação duma curva implícita [Lopes01]. Infelizmente, estes algoritmos não conseguem distinguir um ponto regular de um ponto singular (ou singularidade). Além disso, só por mera sorte estes algoritmos conseguem determinar a existência dum ponto isolado.

5. CONCLUSÕES

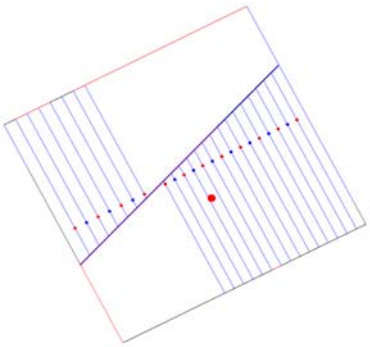
O algoritmo proposto resultou do desenvolvimento dum outro algoritmo proposto pelos autores (veja-se [Morgado02]). No entanto, ao contrário do anterior algoritmo, o algoritmo aqui e agora proposto baseia-se unicamente no conceito de subdivisão binária do espaço. O algoritmo descrito em [Morgado02] é a bem dizer um algoritmo híbrido, pois faz uso não só do conceito de subdivisão do espaço, mas também do conceito de vizinhança usado nos algoritmos de perseguição ou continuação da curva.

Além disso, o algoritmo permite também determinar as várias componentes da curva, pontos isolados inclusivé.



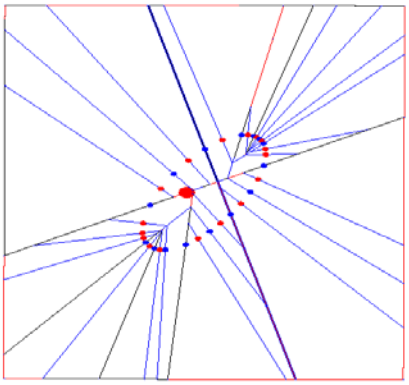
$(x-1)(y-1)=0$

fcpp = 13. 5; drawing time = 0.11s (a)



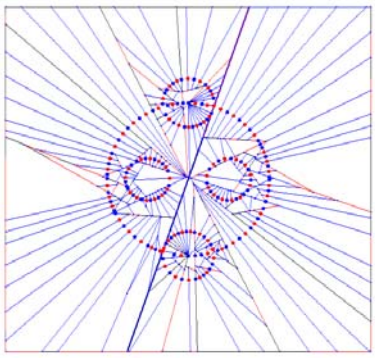
$(x+y)((x-2)^2+(y-1)^2)=0$

fcpp = 13.8; drawing time = 0.11s (b)



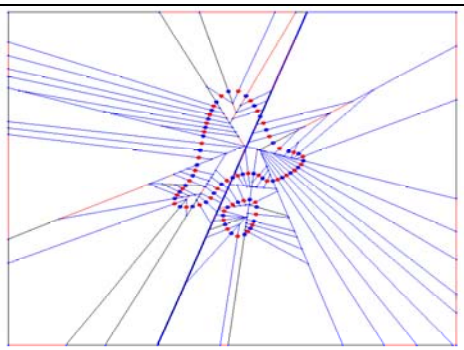
$(x^2/10+y^2/2-1)((x-0.5)^2+(y-0.5)^2)=0$

fcpp = 15.25; drawing time = 0.09s (c)



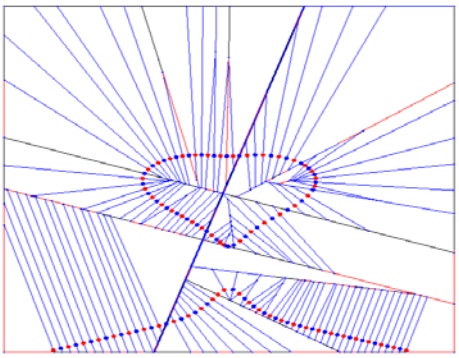
$(x^2+y^2-4)((x-1)^2+y^2-0.3)((x+1)^2+y^2-0.3)(x^2+(y-2)^2-0.4)(x^2+(y+2)^2-0.4) = 0$

fcpp = 15.01;drawing time = 0.31s (d)



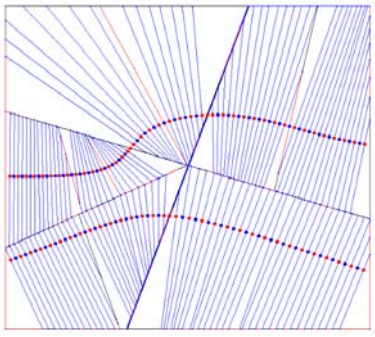
$0.004+0.11x-0.177y-0.174x^2+0.224xy-0.303y^2-0.168x^3+0.327x^2y-0.087xy^2-0.013y^3+0.235x^4-0.667x^3y+0.745x^2y^2-0.029xy^3+0.072y^4= 0$

fcpp = 15.01; drawing time = 0.25s (e)



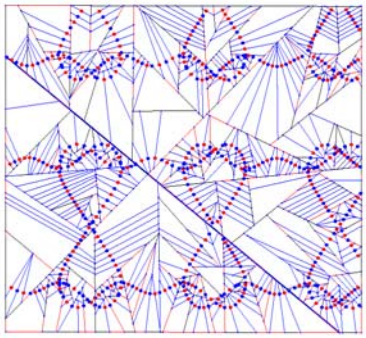
$x^2+4y^2+y^3-2+y/(x^2+0.45)-1=0$

fcpp = 16.01; drawing time = 0.22s (f)



$yx^2+y^4-4-2x = 0$

fcpp = 18; drawing time = 0.23s (g)



$\sin^3(2x)+4\sin^3(y)-3\sin(2x)\sin(y)=0$

fcpp = 14.53; drawing time = 0.92s (h)

Figura 6: Resultados experimentais.

Note-se que, talvez, a maior contribuição deste artigo é precisamente resolver o problema da determinação dos pontos isolados.

O algoritmo mostrou-se relativamente rápido e pensamos que poderia ser incluído nos sistemas gráficos actuais. No entanto, a rapidez do método depende em grande parte da eficiência do método de cálculo da intersecção entre um segmento de recta e a curva. Por isso torna-se importante para o futuro encontrar algoritmos que permitam acelerar esse processo.

6. REFERENCES

[Allgower90] Allgower, E. e Georg, K. Numerical Continuation Methods: An Introduction. Springer-Verlag, (1990)

[Allgower91] Allgower, E., Gnutzmann, S.: Simplicial Pivoting for Mesh Generation of Implicitly Defined Surfaces. Computer Aided Geometric Design, 8 (1991), 305--325

[Akai94] Akai, Terrence J.: Applied Numerical Methods for Engineers. John Wiley & Sons Inc, (1994)

[Blinn82] Blinn J. F., A Generalization of Algebraic Surface Drawing, ACM Transactions on Graphics, 1(3), 1982, 235-256.

[Bloomenthal88] Bloomenthal, J. Poligonisation of implicit surfaces Computer Aided Geometric Design, 5, (1988), 341-355.

[Bloomenthal94] Bloomenthal, J. An Implicit Surface polygonizer. Graphics Gems, IV, (1994).

[Chandler88] Chandler, R. A tracking algorithm for implicitly defined curves, IEEE Computer Graphics & Applications, 8(2), (1988), 83-89.

[Lopes01] Lopes, H., Oliveira, J. e Figueiredo, L., Robust Adaptive Polygonal Approximation of Implicit Curves, Actas de SIBGRAPI'2001, IEEE Computer Society.

[Lorensen87] Lorensen, W. e Cline, W. Marching Cubes: A High Resolution 3D Surface Construction Algorithm, Computer Graphics, 21(4), 1987, 163-169.

[Moller95] Moller, T. and Yagel, R., Efficient Rasterization of Implicit Functions, (1995), artigo publicado em (http://citeseer.nj.nec.com/357413.html).

[Morgado02] Morgado, F. e Gomes, A.., Fast Representation of Implicit Curves Through Space Subdivision, Actas do 1º Simpósio Ibero-Americano de Computação Gráfica (SIACG'2002), Guimarães, Portugal, 2002.

[Snyder92] Snyder, J.: Interval arithmetic for computer graphics. Proceedings of ACM Siggraph, (1992), 121—130.

[Triquet01] Triquet, F. *et al*. "Fast Polygonization of Implicit Surfaces", WSCG' 2001,2,(2001), 283-290.