Universidade da Beira Interior
Departamento de Informática

Poligonização de Superfícies Implícitas por Amostragem baseada num Corrector de Newton

Adriano Nunes Raposo

Mestrado em Engenharia Informática

Fevereiro de 2006

Departamento de Informática
Universidade da Beira Interior

Poligonização de Superfícies Implícitas por Amostragem baseada num Corrector de Newton

Adriano Nunes Raposo

Tese realizada sob a orientação do
Prof. Doutor Abel João Padrão Gomes

Fevereiro de 2006

Abstract

Most algorithms for sampling implicit surfaces are based on the Intermediate Value Theorem. That is, it is assumed that the function that describes a surface changes sign in the neighborhood of each point of it. It happens that not all functions describing implicit surfaces change sign in the neighborhood of each surface point. We introduce a new Newton-Corrector based algorithm capable of sampling implicit surfaces, regardless whether there exists sign changes near the surface or not.

Resumo

A maioria dos algoritmos para amostragem de superfícies implícitas baseia-se no Teorema do Valor Intermédio. Isto é, assume-se que existe variação de sinal da função que representa a superfície. Acontece que nem sempre é assim, ou seja, nem sempre existe variação de sinal da função na vizinhança da superfície. Introduz-se assim um novo algoritmo baseado num Corrector de Newton capaz de efectuar a amostragem de superfícies implícitas quer exista ou não variação de sinal da função na vizinhança da superfície.

Agradecimentos

Ao Prof. Doutor Abel Gomes, pelo apoio incondicional desde o primeiro momento, pela persistência, pela infinita paciência e, sobretudo, pela amizade.

Ao Prof. Doutor José Francisco Morgado e ao Prof. Doutor Frutuoso Silva, pelas preciosas ajudas técnicas, pela total disponibilidade e pelos inesquecíveis debates de ideias.

Ao Prof. Doutor João Queiroz, Vice-Reitor da Universidade da Beira Interior (UBI) e Presidente da Faculdade de Ciências da Saúde (FCS), por todo o apoio dado; e a todos os meus colegas da FCS, em especial ao Gabinete de Informática e Comunicação Digital.

A todos os meus professores e colegas de mestrado, pelo ambiente de camaradagem, entreajuda e amizade que uniu todo o grupo desde o início desta aventura.

A todos os meus amigos verdadeiros, aqueles que estiveram sempre em presença ou em espírito nos momentos bons e, principalmente, nos momentos difíceis. Em especial à Elsa Alves pela preciosa ajuda na revisão final deste texto.

Aos meus pais Alda e António José, e ao meu irmão, futuro engenheiro electrotécnico António Raposo, sem eles nada teria sido possível.

À Xana, por tudo.

A Deus.

Conteúdo

Abstract **iii**

Resumo **iv**

Agradecimentos **v**

Índice de Figuras **x**

Índice de Tabelas **xii**

1 Introdução **1**

1.1 Motivação e Objectivos 1

1.2 Estrutura da Tese 2

2 O Estado da Arte **4**

2.1 Introdução 4

2.2 Superfícies Implícitas 5

2.2.1 Superfícies Implícitas Degeneradas 5

2.3 Superfícies Paramétricas 7

2.4 Representação Implícita vs Representação Paramétrica 8

2.5 Aplicação em Modelação Geométrica 9

2.5.1 Operações Booleanas e Modelação CSG 9

2.5.2 Operações de *blending* 11
2.6 Algoritmos de Visualização de Superfícies Implícitas 11
2.6.1 Algoritmos de Subdivisão Espacial 11
2.6.2 Algoritmos de Continuação 12
2.6.3 Algoritmos de Aproximação 13
2.7 Considerações Finais 13

3 Algoritmo **14**
3.1 Introdução 14
3.2 Conceitos Fundamentais 16
3.2.1 Ponto Inicial da Expansão 16
3.2.2 Frentes de Expansão 17
3.2.3 Ângulos de Expansão 18
3.2.4 Caixa de Delimitação Espacial 19
3.3 Corrector de Newton 20
3.4 Estrutura Geral do Algoritmo 22
3.5 Passo 1: Processamento Simbólico 24
3.5.1 Factorização 24
3.5.2 Inspecção Simbólica da Factorização 26
3.5.3 Considerações Adicionais 27
3.6 Passo 2: Hexágono Inicial 27
3.7 Passo 3: Actualização dos Ângulos de Expansão 28
3.8 Passo 4: Controlo das Colisões entre Frentes de Expansão . . . 31
3.8.1 Subdivisão de uma Frente de Expansão 32
3.8.2 Fusão de Duas Frentes de Expansão 35
3.9 Passo 5: Expansão 37

3.9.1 Cálculo do número de triângulos de expansão 38
3.9.2 Construção dos Triângulos 40
3.9.3 Projecção Ortogonal 42
3.9.4 Rotação com Eixo Arbitrário 43
3.9.5 Limitação da Expansão 44
3.10 Considerações Finais 46

4 Implementação e Testes **47**
4.1 Introdução 47
4.2 Ferramentas Utilizadas 47
4.2.1 Java 3D 48
4.2.2 Java Symbolic Computing Library (jscl) 48
4.2.3 Java Expression Parser (**JEP** e **DJEP**) 49
4.3 A Implementação do Algoritmo: **iSurf** 51
4.3.1 Arquitectura Geral 52
4.3.2 Diagrama de Classes 53
4.3.3 GUI (*Graphical User Interface*) 56
4.3.4 Testes de Desempenho 56
4.3.4.1 Quádricas 57
4.3.4.2 Superfícies Não Degeneradas 60
4.3.4.3 Superfícies Degeneradas 60
4.4 Considerações Finais 63

5 Conclusões **64**
5.1 Resultados Obtidos 64
5.2 Trabalho Futuro 65
5.2.1 Determinação da Intersecção de Componentes Simbólicas 65

5.2.2 Identificação e Representação de Singularidades 66

Bibliografia **67**

Lista de Figuras

2.1 $x^2 + y^2 + z^2 - 1 = 0$ 5

2.2 $x + y + z - 1 = 0$ 6

2.3 $x^2 + y^2 - 1 = 0$ 6

2.4 $x^2 - y^2 = 0$ 7

2.5 $\frac{x^2}{2} + y^2 + z^2 - 2 = 0$ 8

2.6 Árvore CSG 10

3.1 Superfície construída por expansão. 15

3.2 $x^3 + y^2 + z = 0$ 15

3.3 Conceitos fundamentais. 18

3.4 Ângulo de expansão. 19

3.5 Parabolóide recortado pela caixa de delimitação espacial. . . . 20

3.6 Corrector de Newton. 22

3.7 $x \ln x + \ln x \cos z - xy - y \cos z = 0$ 25

3.8 $x \ln x + \ln x \cos z - xy - y \cos z - 1 = 0$ 26

3.9 Hexágono inicial. 29

3.10 Casos possíveis no cálculo de um ângulo de expansão. 30

3.11 Sobreposição da superfície. 31

3.12 Subdivisão da frente de expansão actual. 32

3.13 Esfera maior que a caixa de delimitação espacial. 34

3.14 Exemplo da esfera $x^2 + y^2 + z^2 - 13 = 0$. 34
3.15 Fusão de duas frentes de expansão. 36
3.16 Maus vértices próximos. 36
3.17 Triângulo irregular. 38
3.18 Divisão do ângulo de expansão. 39
3.19 Expansão no caso em que $n_t = 1$. 41
3.20 Expansão no caso em que $n_t > 1$. 41
3.21 Projecção ortogonal. 43
3.22 Rotação com eixo arbitrário. 44
3.23 Limitação da expansão pela CDE. 44

4.1 Arquitectura geral da aplicação 53
4.2 Diagrama de Classes 55
4.3 GUI 57
4.4 Esfera com variação de sinal (a) e sem variação de sinal (b) . . 58
4.5 Parabolóide com variação de sinal (a) e sem variação de sinal (b) 58
4.6 Elipsóide (a) e parabolóide hiperbólico (b) 59
4.7 $x^3 + y = 0$ (a) e $ln(x) + cos(y) - z = 0$ (b) 61
4.8 $x^4 + 2x^3 + xyz + 2xy + x^2z^2 + 2x^2z + z^2y + 2zy = 0$ 61
4.9 Superfície degenerada com componentes simbólicas explícitas (a) Superfície degenerada com componentes simbólicas não explícitas (b) 63

Lista de Tabelas

4.1 Esferas e parabolóides. 59
4.2 Outras quádricas. 60
4.3 Superfícies implícitas não degeneradas. 60
4.4 Superfície 4.1. 62
4.5 Superfície 4.2. 62

Capítulo 1

Introdução

Este capítulo apresenta a motivação e os objectivos que estiveram na origem deste trabalho. A estrutura da tese aparece descrita no final do capítulo.

1.1 Motivação e Objectivos

O que levou à realização deste trabalho foi o facto das **superfícies implícitas** serem um tema de investigação de relevo em computação gráfica, mais precisamente na área da modelação geométrica. Este tipo de superfícies, quando combinadas entre si, através de técnicas como CSG (*Constructive Solid Geometry*) [20] e operações de *blending* [5], permitem construir e representar um largo espectro de objectos geométricos. Por exemplo, é possível representar objectos definidos por funções algébricas (polinomiais), racionais, exponenciais, logarítmicas e transcendentes.

Um dos aspectos fundamentais da investigação em superfícies implícitas é a amostragem e poligonização de superfícies implícitas, que é o tema desta tese. A maioria dos algoritmos de amostragem e poligonização de superfícies implícitas baseia-se no Teorema do Valor Intermédio, isto é, assumem que existe variação de sinal da função que representa a superfície para localizarem pontos da mesma (zeros da função). Ou seja, parte-se sempre de duas estimativas, entre as quais se encontra o zero da função que corresponde a um ponto da superfície. Acontece que existem superfícies implícitas definidas por funções onde não existe variação de sinal e, nestes casos, os métodos tradicionais não conseguem representar os pontos da superfície.

O principal objectivo deste trabalho foi o de propor um algoritmo que não dependesse da variação do sinal da função de uma superfície para efectuar a amostragem e poligonização dessa superfície. Para isso, propõe-se um método numérico baseado no Método de Newton em $\mathbb{R}^3$, a partir daqui designado por Corrector de Newton. Este método, ao contrário de métodos como o Método da Secante ou o Método da Falsa Posição, efectua a amostragem apenas a partir de uma estimativa, ou seja, um só ponto em vez de dois, como acontece com estes métodos.

Utilizam-se muitas vezes algoritmos que geram malhas de triângulos bastante distintos, o que compromete a qualidade da representação obtida. Uma técnica aplicável a este tipo de problema é a construção de malhas "hexagonais"semelhantes a "favos de abelha"[10]. Esta técnica tira partido do facto de cada hexágono regular ser composto por seis triângulos equiláteros, ou seja, hexágonos semelhantes vão dar origem a triângulos semelhantes que, quando combinados entre si, vão gerar uma malha tendencialmente regular. Assim, o segundo dos objectivos do trabalho aqui apresentado é aplicar este tipo de técnica à representação de superfícies implícitas de maneira a obter malhas regulares que melhorem a qualidade da representação das superfícies.

Um caso particular das superfícies implícitas são as **superfícies implícitas degeneradas**, ou seja, superfícies que podem ser decompostas em várias superfícies mais simples, representadas também elas na forma implícita [11]. Cada uma destas superfícies elementares corresponde a um factor ou componente simbólica da expressão da função. Este facto é útil uma vez que permite fazer a amostragem e poligonização de cada componente simbólica individualmente. Como a identificação das componentes simbólicas duma superfície implícita degenerada pode ser feita através de técnicas de factorização simbólica da sua expressão [11], o terceiro dos objectivos deste trabalho é incluir este método no algoritmo proposto.

1.2 Estrutura da Tese

Esta tese é composta por mais quatro capítulos para além do actual, nomeadamente:

- No Capítulo 2 faz-se uma introdução às superfícies implícitas em geral, seguida de um pequeno levantamento do estado da arte em relação aos vários tipos de algoritmos existentes para representação deste tipo de superfícies.

- No Capítulo 3 descreve-se detalhadamente o algoritmo proposto para amostragem e poligonização de superfícies implícitas baseada num Corrector de Newton. Começa por introduzir-se os conceitos básicos do algoritmo, para fazer de seguida uma apresentação da dua estrutura geral. Finalmente, descreve-se cada um dos passos do algoritmo em detalhe.

- No Capítulo 4 apresenta-se uma implementação real do algoritmo proposto. Apresentam-se também os resultados de alguns testes efectuados com a aplicação desenvolvida em Java. Neste capítulo, apresentam-se ainda as ferramentas utilizadas na implementação do algoritmo, assim como a sua arquitectura e estrutura de classes.

- A tese termina com o Capítulo 5, no qual se dá conta das conclusões mais significativas sobre os resultados obtidos. Apontam-se também novas direcções para o trabalho futuro.

Capítulo 2

Superfícies Implícitas: O Estado da Arte

2.1 Introdução

A computação gráfica é um ramo do saber que se dedica ao estudo de imagens digitais geradas por computador a partir de dados numéricos e modelos matemáticos que representam os objectos que se pretende simular ou visualizar graficamente [7].

A modelação geométrica, em particular, a modelação de superfícies implícitas, é tão somente uma das sub-áreas da computação gráfica que se dedica à representação, modelação e processamento de objectos geométricos. Embora haja esquemas de representação mais adequados do que outros para situações particulares, a utilização de funções matemáticas é a solução ideal para representar determinados objectos geométricos tais como curvas, superfícies e sólidos. Esta forma de representação tem várias vantagens, mas as principais são: (i) grande economia de memória em consequência da concisão da representação e (ii) capacidade de descrever objectos de forma exacta, ou seja, não aproximada, sem os erros numéricos que as aproximações implicam [7]. No entanto, sempre que se pretende visualizar uma superfície, há que transformar uma expressão simbólica duma função numa representação gráfica, o que conduz normalmente a erros de aproximação. Basta para isso relembrar que os sistemas gráficos estão optimizados para representar triângulos, pelo que a representação gráfica de superfícies é usualmente feita por amostragem de pontos e triangulação.

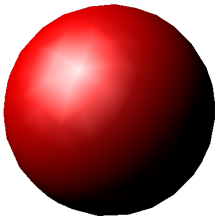

Figura 2.1: $x^2 + y^2 + z^2 - 1 = 0$

Como se descreve nas secções que se seguem, existem duas formas básicas de descrever superfícies matematicamente: **implícita** e **paramétrica**.

2.2 Superfícies Implícitas

Qualquer superfície implícita num espaço afim tridimensional pode ser representada por uma função real da seguinte forma:

$$f(x, y, z) = 0 \tag{2.1}$$

onde $f(x, y, z)$ é um polinómio multivariado em x, y e z. A superfície é constituída pelo conjunto de todos os pontos (x, y, z) que satisfazem a equação (2.1). Formalmente:

$$S = \{(x, y, z) \in \mathbb{R}^3 | f(x, y, z) = 0\} \tag{2.2}$$

É comum dizer que a função $f(x, y, z)$ é a própria superfície. Exemplos bem conhecidos de superfícies implícitas são (i) a esfera unitária centrada na origem (ver Figura 2.1) e (ii) plano arbitrário (ver Figura 2.2).

Um plano genérico tem a sua representação implícita na forma $ax + by + cz - d = 0$ como se exemplifica na Figura 2.2.

2.2.1 Superfícies Implícitas Degeneradas

Uma superfície implícita f diz-se *irredutível* se $f(x, y, z)$ não for factorizável [11], isto é, se não existirem pelo menos duas funções reais $h(x, y, z)$ e $k(x, y, z)$ tais que:

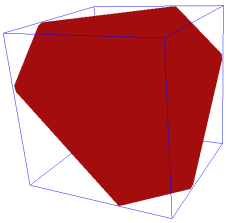

Figura 2.2: $x + y + z - 1 = 0$

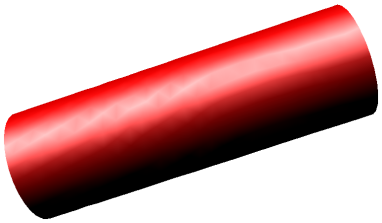

Figura 2.3: $x^2 + y^2 - 1 = 0$

$$f(x, y, z) = h(x, y, z) \cdot k(x, y, z) \tag{2.3}$$

Uma superfície que não seja irredutível diz-se *redutível* ou **degenerada** [11]. Por exemplo, o cilindro da Figura 2.3, definido por $x^2 + y^2 - 1 = 0$, não é uma superfície degenerada.

No entanto, considerando agora outro exemplo, a superfície $x^2 - y^2 = 0$ é degenerada, pois

$$f(x, y, z) = x^2 - y^2 = (x + y) \cdot (x - y)$$

Como se pode observar na Figura 2.4, esta superfície é formada pela reunião de dois planos $h(x, y, z) = 0$ e $k(x, y, z) = 0$, cujas expressões algébricas são $x + y = 0$ e $x - y = 0$, respectivamente.

Assim, quando um superfície implícita é **degenerada**, significa que pode ser decomposta em, pelo menos, duas superfícies distintas, podendo cada uma delas também ser representada por uma equação implícita [11].

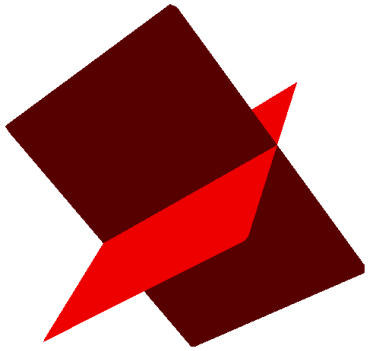

Figura 2.4: $x^2 - y^2 = 0$

2.3 Superfícies Paramétricas

Embora não sejam objecto de estudo no presente texto, considerou-se útil apresentar neste capítulo uma pequena introdução às superfícies paramétricas.

Uma representação paramétrica duma superfície pode ser vista como uma transformação do espaço $\mathbb{R}^2$ numa superfície do espaço $\mathbb{R}^3$.

Caso exista, a representação de uma superfície na forma paramétrica é composta por três funções:

$$x = h_1(s,t); y = h_2(s,t); z = h_3(s,t) \tag{2.4}$$

Atribuindo valores concretos aos parâmetros s e t, as funções (2.4) devolvem as coordenadas de pontos da superfície. Portanto, a representação paramétrica é essencialmente uma representação geradora da própria superfície, o que facilita a sua visualização gráfica. Por exemplo, uma representação paramétrica da esfera unitária da Figura 2.1 poderia ser o conjunto de equações (2.5), donde, para valores dos parâmetros $s = t = 1$, obteriamos o ponto $(-\frac{1}{3}, \frac{2}{3}, \frac{2}{3})$ que está na superfície da esfera [11].

$$x = \frac{1 - s^2 - t^2}{1 + s^2 + t^2}; y = \frac{2s}{1 + s^2 + t^2}; z = \frac{2t}{1 + s^2 + t^2} \tag{2.5}$$

No exemplo da esfera, a parametrização não permite encontrar o ponto $(1, 0, 0)$, uma vez que não é permitida a atribuição de valores infinitos aos parâmetros s e t. A estes pontos dá-se o nome de *singularidades* da parametrização.

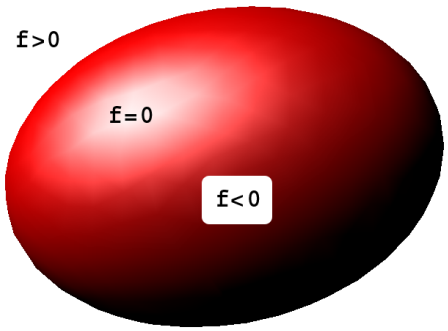

Figura 2.5: $\frac{x^2}{2} + y^2 + z^2 - 2 = 0$

2.4 Representação Implícita vs Representação Paramétrica

Não se pode afirmar que um tipo de representação seja melhor do que outro. Cada representação oferece vantagens e desvantagens para cada situação específica.

A forma implícita tem a vantagem de permitir localizar pontos arbitrários em relação a uma superfície [7]. Por exemplo, considerando um ponto $P = (x_p, y_p, z_p)$ e uma superfície fechada dada por $f(x, y, z) = 0$, basta substituirmos as coordenadas de P em f para que uma (e só uma) das três condições se verifique, como se ilustra na Figura 2.5:

- $f(x_p, y_p, z_p) < 0 \Rightarrow$ o ponto P está no interior da superfície;
- $f(x_p, y_p, z_p) = 0 \Rightarrow$ o ponto P está sobre a superfície;
- $f(x_p, y_p, z_p) > 0 \Rightarrow$ o ponto P está no exterior da superfície.

O cálculo do *vector normal* a um ponto de uma superfície também é mais simples na representação implícita [7]. As coordenadas da normal num ponto P são obtidas como os valores das derivadas parciais de f no ponto P:

$$\overrightarrow{n}_P = (\partial f/\partial x_P, \partial f/\partial y_P, \partial f/\partial z_P)$$

Este facto é importante uma vez que o algoritmo proposto no capítulo seguinte faz uso frequente do vector normal ao efectuar a amostragem e triangulação de superfícies implícitas.

A representação implícita é também muito mais geral do que a paramétrica. Embora todas as superfícies paramétricas possam ser convertidas em superfícies implícitas, existem superfícies que não possuem uma representação paramétrica [11].

Independentemente das vantagens e desvantagens de cada forma de representação, nos últimos anos tem aumentado bastante o interesse pelas superfícies implícitas, impulsionado por uma série de trabalhos que introduziram técnicas sofisticadas e eficientes para a manipulação e visualização deste tipo de superfícies. Vejam-se, por exemplo, os trabalhos publicados anualmente pela Internacional Conference on Shape Modeling and Applications (http://www.shapemodeling.org).

2.5 Aplicação em Modelação Geométrica

Como já se referiu anteriormente, duas técnicas de combinação de objectos implícitos são as operações booleanas, essenciais na modelação CSG, e as operações de *blending* [28].

2.5.1 Operações Booleanas e Modelação CSG

A forma mais comum de utilização de objectos implícitos em computação gráfica é em sistemas de modelação CSG (Constructive Solid Geometry), bastante populares desde que a técnica foi introduzida por Requicha e Voelker (veja-se, por exemplo, [20]) nos finais da década de 70, e que permitem modelar sólidos matematicamente complexos através de um processo construtivo.

No processo de construção, o utilizador dispõe de um conjunto de primitivas — objectos ou superfícies geometricamente simples — quase sempre representadas de forma implícita. Sólidos mais complexos poderão ser construídos por composição destas primitivas. As mais comuns, que estão presentes na maioria das implementações, são esferas, paralelipípedos, pirâmides, planos, cones, cilindros e quádricas.

A combinação dos objectos é feita utilizando **operações booleanas** (uniões, intersecções e diferenças) sobre conjuntos de pontos definidos por funções implícitas. Estas operações, sendo operações binárias, são sempre aplicadas a pares de objectos. No entanto, devem ser aplicadas respeitando certas restrições, de maneira a garantir que o resultado de cada operação seja um

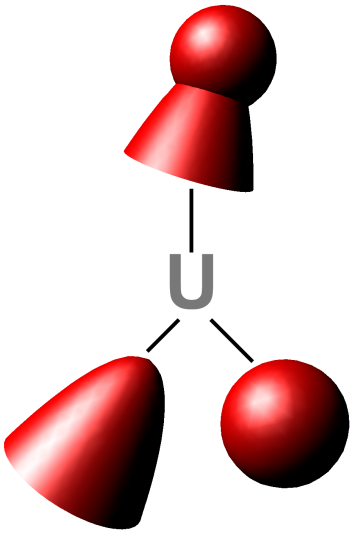

Figura 2.6: Árvore CSG

sólido cuja fronteira seja considerada uma variedade (*manifold*) bidimensional [28]. Esta propriedade foi designada por regularidade de sólidos [20].

Antes de aplicar uma operação booleana, os objectos poderão ser posicionados e dimensionados, de acordo com o objectivo pretendido, através de transformações geométricas simples, tais como translações, rotações, reflexões, contracções e dilatações. A informação relativa a estas transformações vai sendo armazenada numa estrutura de dados que é uma árvore binária designada por árvore CSG. Esta estrutura armazena dois tipos de informação: as primitivas geométricas ou folhas e as operações booleanas e/ou transformações geométricas.

Na Figura 2.6 pode observar-se um exemplo simples de uma árvore CSG. É perceptível que o objecto final na raíz resulta da reunião de duas superfícies mais simples, concretamente, uma superfície parabolólica definida implicitamente por $x^2 + y + z^2 = 0$ e uma superfície esférica definida pela expressão $x^2+y^2+z^2-2 = 0$, embora as primitivas tenham os seus tamanhos alterados no objecto final.

Outra característica que torna esta técnica ainda mais poderosa é a possibilidade de armazenar os objectos compostos, podendo estes ser reutilizados como novas primitivas.

2.5.2 Operações de *blending*

Outro dos factores que contribuiu bastante para o crescimento da popularidade das superfícies implícitas é a combinação destas em modelos mais complexos através de operações de **blending**. Um *blend* de duas superfícies, que poderão ou não intersectar-se, será uma nova forma geométrica que equivale a uma transição suave entre as superfícies originais [28].

Os objectos implícitos são particularmente adequados para a aplicação de operações de *blending*. Dado um conjunto de objectos ou superfícies $f_1(x, y, z)$, $f_2(x, y, z)$, ..., $f_n(x, y, z)$, definidos implicitamente, obtém-se um *blend* através da composição das funções $B(f_1(x, y, z), .., f_n(x, y, z))$, segundo alguma regra de composição, de forma a obter um objecto também ele implicitamente definido.

Uma operação de *blending* pode ser aplicada de duas formas: globalmente ou localmente. Enquanto num *blending* global todos os objectos são considerados e influenciam a forma final, num *blending* local pode restringir-se a certos subconjuntos de objectos ou a certas regiões do objecto final.

Existem vários tipos de *blending*. O tipo mais simples é o *blending* linear mas existem ainda outros tipos como, por exemplo, o *blending* hiperbólico e o super-elíptico [7]. Estes tipos de *blending* não são aqui apresentados detalhadamente por não serem relevantes no contexto do presente trabalho.

2.6 Algoritmos de Visualização de Superfícies Implícitas

Além das operações booleanas e das operações de *blending* outro aspecto fundamental das superfícies implícitas em modelação geométrica é a sua visualização gráfica. Em termos gerais, existem três grandes categorias de algoritmos de visualização de superfícies implícitas: algoritmos de subdivisão espacial, algoritmos de continuação e algoritmos de aproximação.

2.6.1 Algoritmos de Subdivisão Espacial

Os algoritmos de subdivisão espacial efectuam uma subdivisão (regular ou adaptativa) dum sub-espaço rectangular de $\mathbb{R}^3$ num conjunto de células,

localizando depois as que intersectam a superfície implícita [4] [9] [14] [19] [26] [27]. O sinal da função da superfície implícita nos vértices das células utilizadas, determina a maneira como a poligonização será efectuada.

Usualmente, as células são cubos ou tetraedros. Ao contrário dos cubos, os tetraedros vão gerar malhas triangulares topologicamente consistentes (isto é, sem ambiguidades). No entanto estes triângulos podem apresentar-se distorcidos. Este facto, ou seja, a existência de triângulos distorcidos, implica um processamento posterior que faça a reparação da malha resultante.

Um problema grave é o facto de que uma poligonização baseada em células cúbicas poder levar a configurações ambíguas, uma vez que é possível obter várias malhas diferentes para o mesmo tipo de configuração. Existem na literatura várias propostas de estratégias para a resolução deste tipo de problema: decomposição *simplex*, tabela de desambiguação, técnicas de interpolação tri-linear e subdivisão recursiva do espaço, entre outras.

2.6.2 Algoritmos de Continuação

Os algoritmos de continuação geram, de maneira iteractiva, uma aproximação à superfície utilizando uma malha poligonal por expansão, ou seja, uma malha que vai crescendo a partir de um ponto inicial (semente) da superfície [8] [16].

Neste tipo de algoritmo, parte-se de um ponto inicial da superfície, resultante da intersecção das arestas de uma célula inicial com a própria superfície, e os pontos vizinhos vão sendo calculados também a partir da intersecção das células vizinhas com a superfície, isto é, a expansão da malha resulta da expansão das células ao longo da vizinhança da superfície.

Infelizmente, este tipo de algoritmo pode não detectar detalhes importantes da superfície como, por exemplo, pequenas componentes da superfície ou mesmo ponto isolados, uma vez que o tamanho da células é fixo. Outro dos pontos fracos destas técnicas advém da necessidade de definir uma célula inicial para cada componente da superfície, o que, em alguns casos, não é uma tarefa simples.

Assim, entre as técnicas de continuação contam-se: *predictor-corrector* (PC) e *piecewise-linear* (PL) [3] [22] [21] [6]. Nesta tese usa-se a técnica PC em que a previsão se faz através da determinação de pontos num plano tangente a um ponto da superfície, seguida da sua correcção através dum Corrector de Newton.

2.6.3 Algoritmos de Aproximação

Ao contrário dos algoritmos de subdivisão espacial, os algoritmos de aproximação não efectuam uma partição celular do espaço. Neste tipo de algoritmo o processamento inicia-se a partir de uma malha poligonal inicial, que raras vezes estará próxima da superfície que se pretende visualizar, e, a partir desta aproximação, efectua-se uma deformação da malha inicial até se obter uma aproximação satisfatória à superfície implícita [23] [31] [13].

2.7 Considerações Finais

Após o estudo dos trabalhos mais relevantes sobre poligonização de superfícies implícitas, optou-se por escolher a técnica de continuação para levar a cabo este trabalho de investigação.

As razões que estiveram na origem desta escolha foram as seguintes:

- Com esta técnica é possível, como se verá mais adiante, representar superfícies implícitas sem depender da variação do sinal da sua função, ou seja, sem utilizar o Teorema do Valor Intermédio;
- Esta técnica possibilita ainda efectuar a amostragem de pontos da superfície a partir de uma só estimativa em vez de duas, como acontece com a maioria dos métodos existentes;
- Além disso, esta técnica permite criar malhas triangulares tendencialmente regulares, ou seja, com triângulos aproximadamente equiláteros, o que melhora substancialmente a qualidade das superfícies e evita *cracks* nas malhas.

Capítulo 3

Algoritmo de Poligonização de Superfícies Implícitas Degeneradas por Amostragem baseada num Corrector de Newton

3.1 Introdução

No capítulo anterior apresentaram-se várias abordagens para a representação de superfícies implicítas. O algoritmo proposto neste trabalho, e que é apresentado em pormenor neste capítulo, pertence à família dos algoritmos de continuação, isto é, utiliza uma expansão triangular progressiva para efectuar a poligonização de superfícies implícitas num espaço tri-dimensional.

Como já foi dito, o facto de ser um algoritmo de continuação (ou expansão) significa que os pontos de uma zona da superfície são calculados a partir daqueles que lhes são próximos e que foram calculados anteriormente. Noutras palavras, a superfície vai sendo amostrada e triangulada a partir de frentes de expansão consecutivas. Esta abordagem é exemplificada na Figura 3.1 onde, em cada frente de expansão da superfície (a vermelho), vai sendo construída a partir da superfície já representada (a cinzento). O mecanismo de expansão será apresentado detalhadamente na secção 3.9 do presente capítulo.

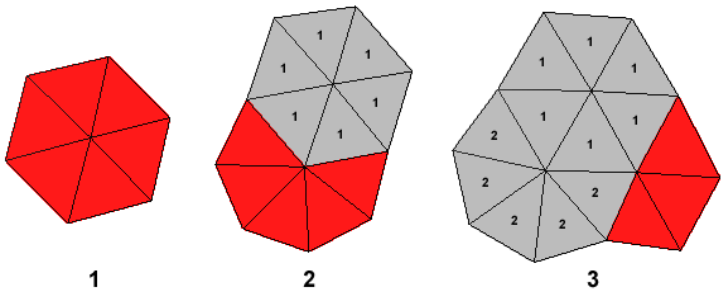

Figura 3.1: Superfície construída por expansão.

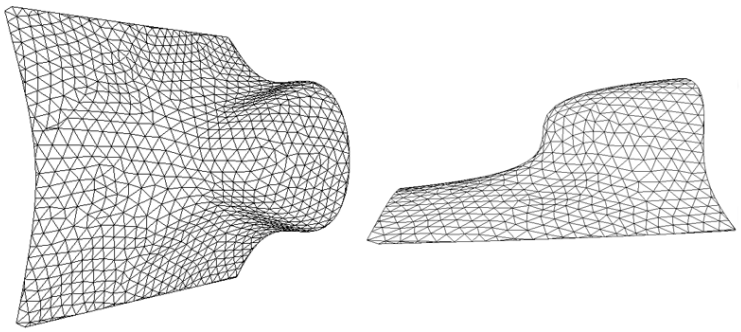

Figura 3.2: $x^3 + y^2 + z = 0$

A partir de um ponto inicial, o algoritmo constrói uma malha triangular com uma particularidade: os triângulos tendem a ser aproximadamente equiláteros com áreas muito semelhantes. Esta característica faz com que as malhas geradas pelo algoritmo sejam bastante regulares, tornando imperceptível para o observador o uso de triângulos na construção das mesmas, após a aplicação de luzes e sombras. Como se pode observar pela malha da Figura 3.2 (gerada pelo algoritmo aqui apresentado) os triângulos são, salvo pequenas excepções que serão posteriormente justificadas, bastante semelhantes, quer nas regiões mais planas, quer nas regiões em que a curvatura é mais acentuada.

No entanto, independentemente do tipo de abordagem adoptada, os algoritmos de representação de superfícies implícitas poderão também basear-se em dois tipos de processamento distintos: **simbólico** ou **numérico**. Também neste caso o algoritmo apresentado é misto, uma vez que recorre quer a processamento simbólico, quer a processamento numérico.

Numa primeira fase do algoritmo é feito um **processamento simbólico** da expressão da superfície com dois objectivos: (i) determinar se se trata ou não de uma superfície implícita degenerada e, em caso afirmativo, (ii) decompor a expressão principal de modo a identificar as expressões das suas componentes. A decomposição é feita com base na factorização da função, sendo este passo do algoritmo apresentado em detalhe na secção 3.5.

A partir do momento em que a função seja decomposta e sejam determinadas as expressões das suas respectivas componentes, o algoritmo passa a usar exclusivamente **processamento numérico**, isto é, todos os valores serão calculados por aproximação numérica, respeitando uma tolerância máxima para os erros, tolerância essa que deverá ser definida *à priori*.

3.2 Conceitos Fundamentais

Nesta secção são introduzidos os conceitos fundamentais que estão na génese do algoritmo descrito na secção seguinte.

Estes conceitos são os seguintes: (i) ponto inicial de expansão, (ii) frentes de expansão, (iii) ângulos de expansão e (iv) caixa de delimitação espacial.

3.2.1 Ponto Inicial da Expansão

Como em qualquer algoritmo de continuação, é sempre necessário definir o ponto a partir do qual se iniciará a expansão. No entanto, e devido à natureza das superfícies implícitas, a localização de um ponto qualquer da mesma no interior da caixa de delimitação espacial, pode não ser uma operação trivial, uma vez que implica o cálculo de um zero da função.

Adoptou-se assim o seguinte mecanismo de definição do ponto inicial da expansão:

1. Toma-se um ponto q, qualquer, no interior da caixa de delimitação espacial;
2. Através do método de Newton-Raphson efectua-se a correcção numérica de q de modo a obter um ponto p da superfície.

Embora a selecção de q possa ser feita aleatoriamente, em algumas situações

será vantajoso fazê-lo manualmente, uma vez que esta estratégia poderá melhorar a qualidade das superfícies.

Quanto à correcção de q, será descrita detalhadamente na secção 3.3. Importa no entanto referir que p será o ponto inicial da expansão.

3.2.2 Frentes de Expansão

Um dos conceitos mais importantes do algoritmo aqui proposto, é o de frente de expansão. Designa-se por frente de expansão o polígono formado pela fronteira de uma secção poligonizada conexa da superfície previamente expandida.

É claro que no início da expansão existirá apenas uma frente de expansão. No entanto, dependendo da superfície a representar, o número de frentes de expansão poderá ser alterado ao longo da execução do algoritmo, ou seja, existirá um lista dinâmica de frentes de expansão.

Serão criadas novas frentes de expansão como resultado da divisão das frentes existentes. Por outro lado, serão removidas da lista as frentes de expansão que se vão fundindo com outras. O mecanismo de controlo de divisão e fusão das frentes de expansão, e a sua importância, serão apresentados em detalhe na secção 3.8.

Na prática, a lista de frentes de expansão será de facto uma fila de espera, isto é, a expansão começa com a primeira frente de expansão e só passa à seguinte quando se verificar algum dos critérios de paragem. Uma vez processada uma frente de expansão, esta é retirada da fila de espera e a frente seguinte passará então a ser a primeira. A expansão termina quando a fila de espera não tiver frentes por processar. Assim, à primeira frente de expansão da fila dar-se-á sempre o nome de frente de expansão actual.

A título de exemplo, veja-se a Figura 3.3. Nela, Π_0 é a frente de expansão actual de uma qualquer superfície e Π_1 e Π_2 são outras frentes de expansão. O processamento do algoritmo concentra-se em Π_0 que, poderá vir ou não a fundir-se com outra frente de expansão quando estiver suficientemente próxima.

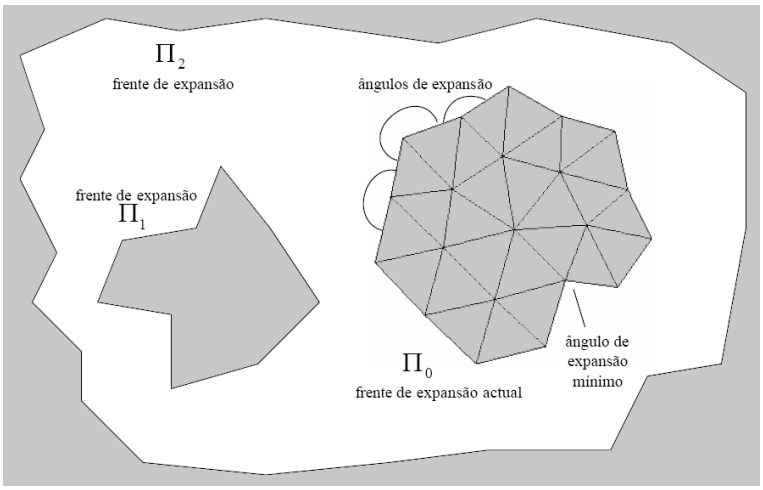

Figura 3.3: Conceitos fundamentais.

3.2.3 Ângulos de Expansão

Outro dos conceitos importantes para a compreensão do algoritmo apresentado, é o conceito de ângulo de expansão.

Regra geral, considera-se que o ângulo entre dois vectores é o menor ângulo definido entre eles. No entanto, esta abordagem não serve o propósito presente, uma vez que o que se pretende é determinar o valor do ângulo entre os dois vectores mas definido pela região não expandida, que poderá não ser o menor ângulo.

Designa-se por ângulo de expansão o ângulo formado por cada par de arestas consecutivas pertencente à fronteira de uma determinada frente de expansão, medido pela área não poligonizada. Portanto, esta definição nem sempre coincide com a definição clássica de ângulo entre duas rectas que estipula sempre o menor ângulo.

Ilustra-se este conceito na Figura 3.4. Os pontos p_{n-1}, p_n e p_{n+1} são vértices consecutivos de uma frente de expansão qualquer. As arestas a e b são os segmentos de recta definidos, respectivamente, pelos pares de vértices (p_{n-1}, p_n) e (p_n, p_{n+1}). Os ângulos α e β são definidos pelas arestas a e b.

Assim, o ângulo de expansão formado pela arestas a e b é o ângulo α e não o ângulo β. Toma-se como ângulo de expansão aquele que se encontra no exterior da frente de expansão, mesmo nos casos em que o ângulo interior seja menor, como acontece no exemplo apresentado ($\beta < \alpha$).

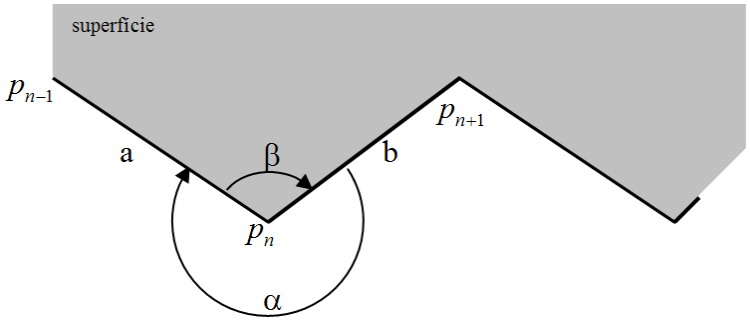

Figura 3.4: Ângulo de expansão.

3.2.4 Caixa de Delimitação Espacial

Embora exista um vasto número de superfícies implícitas confináveis a sub-domínios paralelipipédicos de $\mathbb{R}^3$, a verdade é que existe um grande número de superfícies implícitas que não se podem incluir totalmente em qualquer sub-domínio paralelipipédico de $\mathbb{R}^3$.

Enquanto no primeiro caso as superfícies não obrigam a qualquer limitação espacial, isto é, estão limitadas nos espaço pela sua própria topologia, no segundo caso será necessário estabelecer limites, uma vez que se correria o risco das superfícies se expandirem interminavelmente, com grande probabilidade de tomarem direcções pouco relevantes para a representação pretendida.

A título de exemplo considerem-se duas quádricas elementares: o elipsóide definido pela expressão $(x^2/2) + y^2 + z^2 - 2 = 0$ e o parabolóide definido pela expressão $x^2 + y + z^2 - 4 = 0$. Embora no caso do elipsóide seja trivial encontrar uma caixa que o contenha na sua totalidade, no caso do parabolóide isso não é possível.

De modo a evitar situações de expansão ilimitada, no algoritmo aqui apresentado adoptou-se um mecanismo simples de delimitação do espaço de representação das superfícies. Concretamente, utiliza-se um cubo ou um paralelepípedo em $\mathbb{R}^3$ como limite para a representação das superfícies.

Esta caixa, a partir daqui designada por Caixa de Delimitação Espacial (CDE) ou *bounding box*, poderá ter um tamanho qualquer e estar centrada em qualquer ponto do espaço, desde que, por uma questão de simplificação do processamento, esteja perfeitamente alinhada com os eixos coordenados, isto é, que cada uma das suas faces seja paralela um dos planos $x = 0$, $y = 0$ ou $z = 0$.

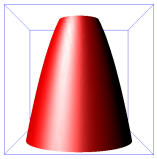

Figura 3.5: Parabolóide recortado pela caixa de delimitação espacial.

Na prática, serão representadas total ou parcialmente, as superfícies, ou partes destas, que estejam dentro da CDE escolhida. No caso de inclusão parcial, as superfícies serão representadas até às suas intersecções — caso existam — com as faces da caixa. Por exemplo, se no caso do parabolóide $x^2+y+z^2-4=0$ for usado como caixa de delimitação espacial um cubo centrado na origem e com vértices $(3,3,3)$, $(3,-3,3)$, $(3,3,-3)$, $(3,-3,-3)$, $(-3,3,3)$, $(-3,-3,3)$, $(-3,3,-3)$ e $(-3,-3,-3)$, o parabolóide será recortado pelos planos $z=3$ e $z=-3$ como se ilustra na Figura 3.5.

3.3 Corrector de Newton

Uma das ferramentas fundamentais do algoritmo proposto é a aproximação numérica à superfície de pontos da sua vizinhança. Esta correcção é feita utilizando o método de Newton-Raphson para funções trivariadas em $\mathbb{R}^3$. Por coincidência, a aproximação do ponto à superfície ocorre de forma semelhante às leis físicas de atracção estabelecidas por Newton.

De facto, a superfície funciona como um atractor dos pontos vizinhos. Noutras palavras, dado um ponto q qualquer na vizinhança da superfície, pretende-se determinar um ponto p suficientemente próximo da superfície de modo que se possa considerar p como pertencente à própria superfície.

Considere-se, então, $f(x,y,z)=0$ a expressão de uma superfície implícita qualquer. Seja ∇f o gradiente de f, que deverá existir e não ser nulo em qualquer ponto envolvido no processamento. Tome-se q um ponto qualquer na vizinhança da superfície e ε o erro máximo admissível para a distância entre um ponto p da superfície e um ponto q_i calculado pelo método de Newton. É possível determinar um ponto p da superfície aplicando o seguinte algoritmo ou corrector de Newton:

Algoritmo 3.1 (Newton)

(a) $q_0 = q$

(b) REPETIR

$$q_{k+1} = q_k - \frac{f(q_k)}{\nabla f(q_k)^2} \nabla f(q_k)$$

ATÉ $\|q_{k+1} - q_k\| \leq \varepsilon$

(c) $p = q_{k+1}$

De forma a exemplificar o algoritmo anterior considere-se novamente a esfera unitária definida implicitamente por $x^2 + y^2 + z^2 - 1 = 0$, já ilustrada na capítulo anterior pela Figura 2.1. Ao tomar como ponto inicial para o algoritmo o ponto $q = (1, 1, 1) = q_0$, e considerando $\varepsilon = 0.001$, obtêm-se os seguintes resultados:

k	q_k	$\|q_k - q_{k-1}\|$
0	(1.00000,1.00000,1.00000)	-
1	(0.66667,0.66667,0.66667)	0.57735
2	(0.58333,0.58333,0.58333)	0.14434
3	(0.57738,0.57738,0.57738)	0.01031
4	(0.57735,0.57735,0.57735)	0.00005

Por uma questão de simplicação os resultados foram arredondados à quinta casa decimal.

Ao atingir o critério de paragem dado por ε, chega-se a p que neste caso seria $p = q_4 = (0.57735, 0.57735, 0.57735)$. Facilmente se demonstra que p é efectivamente um ponto pertencente à superfície:

$$f(p) = f(0.57735, 0.57735, 0.57735) = 0.57735^2 + 0.57735^2 + 0.57735^2 - 1 \approx 0$$

O funcionamento deste algoritmo, concretamente para o exemplo anterior, é ilustrado na Figura 3.6.

Além da localização de pontos da superfície, será necessário determinar três vectores específicos associados a cada ponto p:

- $\overrightarrow{n}$, o **vector normal** à superfície no ponto p calculado da seguinte forma:

$$\overrightarrow{n} = \nabla f(p) / \|\nabla f(p)\|$$

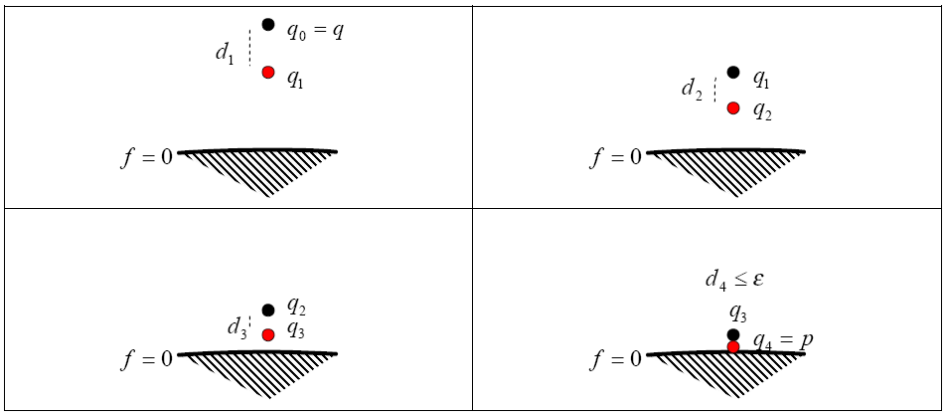

Figura 3.6: Corrector de Newton.

- $\overrightarrow{t_1}$ e $\overrightarrow{t_2}$, dois vectores tangentes à superfície no ponto p, perpendiculares a $\overrightarrow{n} = (n_x, n_y, n_z)$, e perpendiculares entre si, calculados do seguinte modo:

  SE $n_x > 0.5$ ou $n_y > 0.5$

  $t_1 = (n_y, -n_x, 0)/\|(n_y, -n_x, 0)\|$

  SENÃO

  $t_1 = (-n_z, 0, n_x)/\|(-n_z, 0, n_x)\|$

  FIMSE

  $t_2 = n \times t_1$

Note-se que os vectores deverão ser sempre normalizados. Estes serão necessários em vários passos do algoritmo.

3.4 Estrutura Geral do Algoritmo

Nesta secção será feita uma apresentação geral do algoritmo. Note-se que o funcionamento detalhado de cada um dos passos mencionados será omitido e remetido para secções posteriores, apresentando aqui apenas uma descrição resumida. Esta decisão deve-se ao facto de se ter chegado à conclusão que a complexidade e os pormenores inerentes a cada sub-algoritmo poderia dispersar a atenção da essência do algoritmo.

Passo 1 (Processamento Simbólico da Função): Toma-se a expressão principal da superfície a representar e efectua-se a sua factorização. Processa-se o resultado da factorização de maneira a determinar se se trata de uma superfície implícita degenerada e, em caso afirmativo, identificam-se as suas componentes simbólicas e respectivas expressões. Para cada componente simbólica encontrada aplicam-se os restantes passos do algoritmo.

Passo 2 (Hexágono Inicial): Escolhe-se um ponto qualquer q_0 na vizinhança da superfície. Utilizando o corrector de Newton determina-se o correspondente ponto da superfície p_0 com a restrição que p_0 pertença ao interior da CDE. Constrói-se um hexágono regular (centrado em p_0) sobre o plano tangente à superfície em p_0. Utilizando um corrector de Newton determinam-se os pontos $p_1, \ldots, p_6$ a partir dos vértices do hexágono $q_1, \ldots, q_6$. Os triângulos (p_0, p_1, p_2), (p_0, p_2, p_3), (p_0, p_3, p_4), (p_0, p_4, p_5), (p_0, p_5, p_6) e (p_0, p_6, p_1) serão os primeiros seis triângulos da superfície (ver Figura 3.9). O hexágono formado pelos pontos $p_1, \ldots, p_6$ será a primeira frente de expansão Π_0 (ver Secção 3.2.2).

Passo 3 (Actualização dos Ângulos de Expansão): Para cada vértice da fronteira da frente de expansão actual, ou seja, Π_0, determina-se o seu ângulo de expansão (ver Secção 3.2.3).

Passo 4 (Controlo das Frentes de Expansão): Verifica-se se alguns dos pontos p_i da frente de expansão actual está próximo de:

1. Um ponto de Π_0 que não adjacente a p_i;
2. Um ponto qualquer de Π_k com $k > 0$.

No primeiro caso, divide-se a frente de expansão actual Π_0 em duas: Π_0 e Π_{k+1} (em que k é o tamanho actual da fila de espera das frentes de expansão). No segundo caso, se p_i está próximo de um ponto de uma frente de expansão Π_m, está deverá fundir-se com Π_0 e será removida da fila de frentes de expansão.

Passo 5 (Expansão da Superfície): Determinar o ponto p_m da frente de expansão Π_0 com ângulo de expansão mínimo. Preencher o espaço definido pelo ângulo de expansão de p_m com triângulos de forma que os seus ângulos interiores sejam de aproximadamente $\pi/3$ radianos (60^o). Remover p_m da frente de expansão actual Π_0. Adicionar os novos pontos na mesma frente de expansão Π_0.

Passo 6 (Critérios de Paragem): Repetem-se os passos 3, 4 e 5 até que a frente de expansão actual Π_0 seja composta apenas por três pontos que formem entre si um novo triângulo. A seguinte frente de expansão não vazia, caso exista, passará então a ser a nova frente de expansão actual Π_0 e repetem-se os passos 3, 4 e 5 do algoritmo. Caso não existam mais frentes de expansão nestas condições, a superfície estará concluída e o algoritmo cessa a sua execução. Note-se que os pontos resultantes de cortes pela CDE não devem ser condiderados pontos efectivos da frentes de expansão, razão pela qual uma determinada frente de expansão poderá terminar o seu processamento mesmo contendo mais de três pontos.

3.5 Passo 1: Processamento Simbólico

Como já foi referido no capítulo anterior, uma superfície implícita degenerada é uma superfície formada por composição de outras superfícies também elas definidas implicitamente. Assim, dada a expressão da superfície a representar será necessário determinar se se trata de uma superfície implícita degenerada e, em caso afirmativo, identificar as suas componentes.

3.5.1 Factorização

De forma a determinar se uma dada expressão representa uma superfície implícita degenerada, será necessário proceder à sua **factorização**.

Formalmente, a **factorização** de uma expressão é a sua decomposição num produto de outras expressões de menor grau, ou **factores**, de cuja multiplicação resulte a expressão original [30]. Por exemplo, o polinómio $x^2 - 4$ é factorizável como $(x-2)(x+2)$, uma vez que se trata de um dos casos notáveis da multiplicação, ou seja, uma diferença de quadrados. A factorização de polinómios é fundamentada pelo **Teorema Fundamental da Álgebra** [29] que não será abordado no presente contexto.

Se a expressão de uma superfície implícita for factorizável, isto é, se for possível transformá-la num produto de expressões mais simples, cada uma dessas expressões será a representação implícita de uma das componentes simbólicas da superfície principal. Caso contrário, a superfície principal não é degenerada.

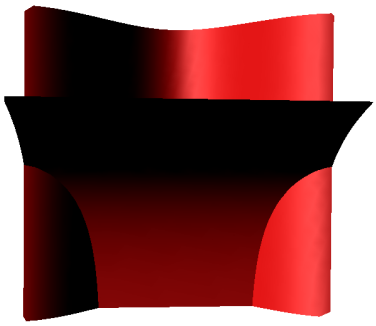

Figura 3.7: $x \ln x + \ln x \cos z - xy - y \cos z = 0$

Exemplo 1: Superfície Implícita Degenerada

Considere-se a superfície implícita dada por:

$$x \ln x + \ln x \cos z - xy - y \cos z \tag{3.1}$$

Depois de factorizada, a expressão transforma-se no produto de duas expressões mais simples:

$$(\ln x - y)(x + \cos z) \tag{3.2}$$

Conclui-se assim que se trata de uma superfície implícita degenerada e que as suas componentes são, respectivamente, as superfícies $\ln x - y$ e $x + \cos z$, o que é verdade e se pode comprovar observando a figura 3.7.

Exemplo 2: Superfície Implícita Não Degenerada

Considere-se agora a seguinte superfície implícita, muito semelhante à anterior:

$$x \ln x + \ln x \cos z - xy - y \cos z - 1 \tag{3.3}$$

Ao tentar factorizar chega-se à conclusão que a expressão não é factorizável. Donde, se conclui que não se trata de uma superfície implícita degenerada, ou seja, que é composta por uma única componente que é a própria superfície como se pode observar na figura 3.8.

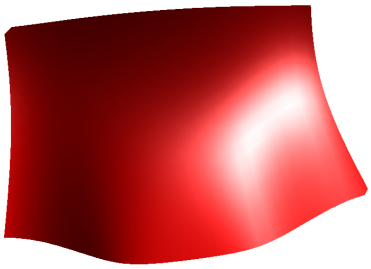

Figura 3.8: $x \ln x + \ln x \cos z - xy - y \cos z - 1 = 0$

3.5.2 Inspecção Simbólica da Factorização

Uma vez efectuada a factorização da expressão da superfície, é necessário avaliar e interpretar o resultado obtido de forma a verificar se se trata ou não de um produto de expressões e, se for o caso, identificar os seus factores. Com esse objectivo, foi criado um pequeno algoritmo simbólico que se apresenta de seguida.

Seja s a expressão da superfície após a factorização.

Algoritmo 3.2 (InspectFactors)

1: SE s contém "+" ou "−" fora de parêntesis ENTÃO
    SE existe algum "−" com operando à esquerda ENTÃO
        s não é uma superfície degenerada
        PARAR
    SENÃO
        IR PARA 3
    FIMSE
SENÃO
    IR PARA 3
FIMSE

2: partir s pelos "$*$" fora de parêntesis

Se se tratar de uma superfície implícita degenerada, do passo 2 do algoritmo anterior resultam as expressões das várias componentes da superfície principal. No entanto, é importante referir que a expressão não deverá conter multiplicações implícitas, ou seja, todas as operações deste tipo deverão ser representadas pelo símbolo "$*$".

3.5.3 Considerações Adicionais

Na secção actual não é proposto nenhum método específico de factorização de expressões polinomais pois não é considerado relevante no presente contexto. Na implementação do protótipo do algoritmo foi utilizada uma ferramenta já existente para efectuar as factorizações necessárias [12].

A **análise simbólica da função** é um passo fundamental no algoritmo de visualização de superfícies aqui proposto, uma vez que cada componente será processada individualmente. À excepção do presente passo, o algoritmo irá sendo aplicado repetidamente para construir cada uma das superfícies que compõem a superfície principal.

Assim, a partir deste ponto, e uma vez que, pela sua inspecção simbólica, tenham sido identificadas as componentes da superfície implícita degenerada, os restantes passos do algoritmo serão efectuados para cada destas, pelo que, daqui em diante, ao referir a superfície estará a referir-se a uma qualquer das suas componentes.

3.6 Passo 2: Hexágono Inicial

Concluída a factorização da expressão de f, segue-se a amostragem e triangulação da superfície. O primeiro passo da triangulação da superfície será a localização de um ponto inicial qualquer pertencente à própria superfície, desde que esteja dentro da CDE definida, a partir do qual se possa iniciar a expansão.

Para isso, escolhe-se um ponto qualquer na vizinhança da superfície (por exemplo por selecção aleatória) e aplica-se o método de Newton-Raphson para obter o ponto inicial p_0 e os vectores $\overrightarrow{n}$ (vector normal à superfície no ponto p_0), $\overrightarrow{t_1}$ e $\overrightarrow{t_2}$ (perpendiculares a $\overrightarrow{n}$). Note-se que $\overrightarrow{n}$, $\overrightarrow{t_1}$ e $\overrightarrow{t_2}$ formam um referencial ortonormado com origem em p_0.

Depois, e uma vez encontrados p_0, $\overrightarrow{n}$, $\overrightarrow{t_1}$ e $\overrightarrow{t_2}$, considera-se o plano α, tagente à superfície no ponto p_0 definido pelos vectores $\overrightarrow{t_1}$ e $\overrightarrow{t_2}$.

Se se considerar δ o raio de uma circunferência imaginária centrada em p_0 no plano α, determinam-se sucessivamente os pontos q_i, com $i \in \{1, 2, 3, 4, 5, 6\}$, da seguinte forma (ver Figura 3.9):

$$q_{i+1} = p_1 + \delta \cos(i\pi/3)t_{11} + \delta \sin(i\pi/3)t_{12} \tag{3.4}$$

Ou seja, os ponto q_i sobre a circunferência em α são determinados por rotação sucessiva de $\pi/3$ radianos em torno do eixo formado por p_0 e $\overrightarrow{n}$, até que a soma dos ângulos de rotação perfaça 2π radianos.

Note-se que depois de determinar cada q_i, e antes de passar ao cálculo do seguinte, aplica-se-lhe o Corrector de Newton obtendo-se assim um novo ponto p_i da superfície.

Algoritmicamente, e considerando que a função ***newton*** representa o Corrector de Newton, o procedimento apresentado resume-se a:

Algoritmo 3.3 (FirstHexagonVertices)

PARA $i = 0$ ATÉ 5

$q_{i+1} = p_1 + \delta \cos(i\pi/3)t_{11} + \delta \sin(i\pi/3)t_{12}$
$p_{i+1} = newton(q_{i+1})$

FIMPARA

Desta forma, obtêm-se os seis pontos p_i da superfície que são os vértices de um hexágono aproximadamente regular centrado em p_0. Este hexágono é precisamente o **polígono de expansão** incial, ou seja, o conjunto dos seis primeiros triângulos da malha da superfície, nomeadamente:

(p_0, p_1, p_2), (p_0, p_2, p_3), (p_0, p_3, p_4), (p_0, p_4, p_5), (p_0, p_5, p_6) e (p_0, p_6, p_1)

3.7 Passo 3: Actualização dos Ângulos de Expansão

Como já foi possível observar anteriormente, os ângulos de expansão são fundamentais no processamento do algoritmo. Uma vez que as frentes de expansão irão sendo modificadas ao longo da execução, também os ângulos de expansão serão diferentes de uma iteração para a seguinte. Assim, de modo a utilizar sempre os valores correctos e actualizados dos ângulos, será

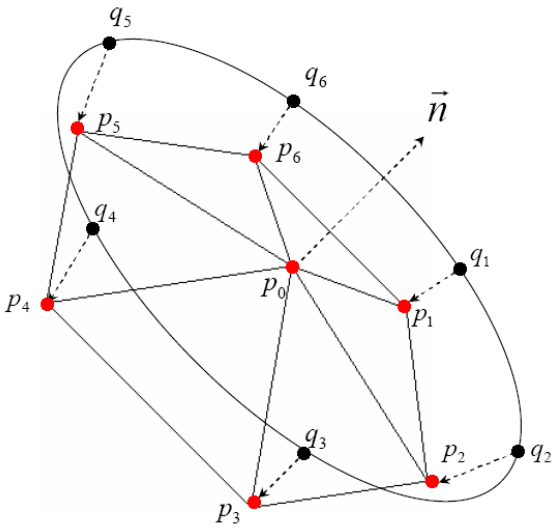

Figura 3.9: Hexágono inicial.

necessário proceder à sua actualização antes de qualquer nova expansão da superfície.

A alteração dos ângulos da frente de expansão actual poderá ocorrer em duas situações:

- A inserção de um novo ponto na frente de expansão vai originar a existência de um novo ângulo de expansão relativo a esse mesmo ponto;
- A inserção de pontos adjacentes a pontos já existentes na frente de expansão vai alterar o valor dos seus ângulos de expansão.

Considerando então que $\Pi_0 = (p_1, p_2, \ldots, p_{N_0})$ é a frente de expansão actual, se p_k for um novo ponto ou se for inserido um ponto adjacente a p_k, será necessário re-calcular o ângulo de expansão centrado em p_k.

Sendo cada frente de expansão uma lista circular de pontos devidamente ordenados, será possível determinar quais os pontos anterior e seguinte a p_k, ou seja, p_{k-1} e p_{k+1}, respectivamente, como se vê na Figura 3.10.

Considere-se então os vectores $\overrightarrow{u} = \overrightarrow{p_k p_{k-1}}$ e $\overrightarrow{v} = \overrightarrow{p_k p_{k+1}}$, e designe-se por ω o ângulo de expansão formado por estes, medido sempre no sentido da ordenação da frente de expansão, ou seja, do anterior para o seguinte. Pretende-se então calcular o valor de ω.

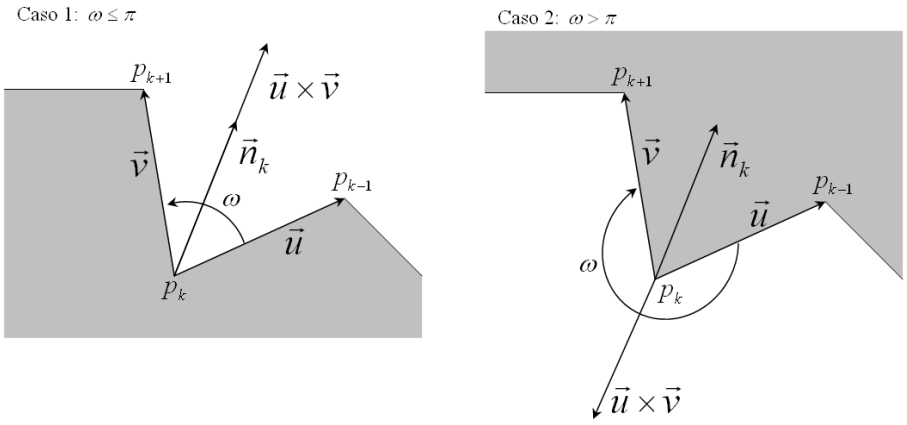

Figura 3.10: Casos possíveis no cálculo de um ângulo de expansão.

Como se pode observar na Figura 3.10, existem dois casos possíveis:

Caso 1: O ângulo pretendido é o menor, ou seja, $\omega \leq \pi$;

Caso 2: O ângulo pretendido é o maior, ou seja, $\omega > \pi$.

O mecanismo adoptado para determinar se o ângulo de expansão ω é o maior ou o menor dos ângulos entre os vectores $\overrightarrow{u}$ e $\overrightarrow{v}$, baseia-se na comparação do seu produto vectorial com o vector normal $\overrightarrow{n}_k$ no ponto p_k. Se $\overrightarrow{u} \times \overrightarrow{v}$ tiver o mesmo sentido que $\overrightarrow{n}_k$ então ω será o menor ângulo entre $\overrightarrow{u}$ e $\overrightarrow{v}$. Caso contrário ω será igual à diferença entre 2π e o menor ângulo. Refira-se a propósito que o ângulo entre dois vectores é, por definição, o menor dos ângulos entre eles, mas o ângulo de expansão nem sempre é o menor dos ângulos.

Algoritmicamente, sendo $\overrightarrow{u}_k$ o vector normal à superfície no ponto p_k e `angle` uma função que devolve o valor do menor ângulo entre dois vectores, o algoritmo usado para o cálculo do ângulo de expansão será o seguinte:

Algoritmo 3.4 (FrontAngle)

(1) $\overrightarrow{u} \leftarrow \overrightarrow{p_k p_{k-1}}$ e $\overrightarrow{v} \leftarrow \overrightarrow{p_k p_{k+1}}$;

(2) $\overrightarrow{t} \leftarrow \overrightarrow{u} \times \overrightarrow{v}$;

(3) SE $\overrightarrow{t} \cdot \overrightarrow{n} < 0$ ENTÃO

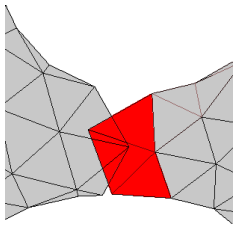

Figura 3.11: Sobreposição da superfície.

$\omega \leftarrow 2\pi - angle(\overrightarrow{u}, \overrightarrow{v})$;

SENÃO

$\omega \leftarrow angle(\overrightarrow{u}, \overrightarrow{v})$;

(4) marcar ω para que não seja re-calculado.

Note-se mais uma vez que este cálculo deverá ser feito para cada novo vértice da frente de expansão ou para vértices que tenham novos vértices adjacentes. Além disso os ângulos actualizados deverão ser marcados para que não sejam re-calculados duas ou mais vezes na mesma iteração.

3.8 Passo 4: Controlo das Colisões entre Frentes de Expansão

Ao efectuar a triangulação de uma superfície com recurso a métodos de expansão ou continuação, poderá haver o risco da expansão avançar por regiões previamente poligonizadas, o que irá originar sobreposição de secções poligonizadas da superfície. Nesta secção apresenta-se o mecanismo utilizado para controlar o avanço das frentes de expansão e evitar o fenómeno da sobreposição ou colisão ilustrado na Figura 3.11.

Este passo do algoritmo, deverá ser executado após a actualização dos ângulos de expansão (ver Secção 3.2.3) e antes de efectuar a expansão propriamente dita (ver Secção 3.9).

No entanto, antes de efectuar qualquer verificação de distâncias, deverá proceder-se à expansão (ver Secção 3.9) dos pontos com ângulos de expansão inferiores a $\pi/3$ radianos, ou seja, 60º;

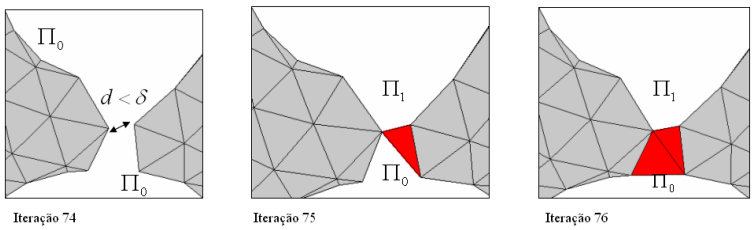

Figura 3.12: Subdivisão da frente de expansão actual.

Assim, na primeira fase deste procedimento será necessário verificar se se está perante uma das seguintes situações:

(a) *Subdivisão de uma frente de expansão.* Considerando as distâncias entre cada vértice da frente de expansão actual Π_0 e cada um dos restantes vértices de Π_0 — excepção feita aos seus vizinhos directos ou aos vizinhos directos dos seus vizinhos — será necessário verificar se alguma dessas distâncias é menor que o valor de δ definido, ou seja, menor do que o raio aproximado dos hexágonos da malha da superfície. Em caso afirmativo deverá proceder-se à divisão da frente de expansão actual. Este mecanismo é apresentado formal e detalhadamente em 3.8.1.

(b) *Fusão de duas frentes de expansão.* Considerando as distâncias entre cada vértice da frente de expansão actual Π_0 e cada um dos vértices de todas as outras frentes de expansão existentes, será necessário verificar se alguma dessas distâncias é menor que o valor de δ definido, ou seja, menor do que o raio aproximado dos hexágonos da malha da superfície. Se assim for deverá proceder-se à fusão da frente de expansão actual com a outra frente de expansão em causa. Esta situação e o seu processamento são descritos em pormenor em 3.8.2.

3.8.1 Subdivisão de uma Frente de Expansão

Nesta secção será tratada a primeira das duas situações de controlo das frentes de expansão. Mais precisamente, a situação em que será necessário proceder a uma ou mais divisões da frente de expansão actual devido à proximidade dos seus pontos.

Assim, considerando p_{0i} e p_{0j} (com $i < j$) dois vértices de Π_0, tais que, p_{0i} não seja adjacente a p_{0j}, nem seja adjacente a outro vértice qualquer de Π_0 adjacente a p_{0j}. Isto é, os vértices deverão ter entre si, pelo menos, dois

outros vértices de Π_0. Considere-se também δ, o valor definido para o raio aproximado dos hexágonos de expansão.

Se a distância entre p_{0i} e p_{0j} for menor que δ, ou seja, se $||p_{0i} - p_{0j}|| < \delta$, a frente de expansão actual deverá ser dividida em duas, dando origem a uma nova frente de expansão Π_m (Figura 3.12). Como resultado da divisão, a frente de expansão actual Π_0 passará a ser composta apenas pelos pontos $p_{01}, \ldots, p_{0i}, p_{0j}, \ldots, p_{0N_0}$, ficando assim com $N_0 - (j - i - 1)$ pontos. Quanto à nova frente de expansão Π_m, será composta pelos pontos $p_{0i}, \ldots, p_{0j}$, ou seja, terá um total de $j - i + 1$ pontos.

Finalmente, os pontos p_{0i} e p_{0j} deverão ser marcados, de maneira a que não sejam submetidos de novo a este controlo de frentes de expansão.

Utilizando uma notação algorítmica, o mecanismo de divisão de uma frente de expansão resume-se a:

Algoritmo 3.5 (SplitFrontPolygon)

(1) Criar Π_m;

(2) PARA $p = p_{0i}$ ATÉ p_{0j}

Inserir p em Π_m;

FIMPARA

(3) PARA $q = p_{0i+1}$ ATÉ p_{0j-1}

Remover q em Π_0;

FIMPARA

(4) Marcar p_{0i} e p_{0j} em Π_0;
Marcar p_{m1} e p_{mj-i+1} em Π_m.

Considere-se, por exemplo, a esfera $x^2 + y^2 + z^2 - 13 = 0$ confinada à CDE definida pelos planos $x = \pm 3$, $y = \pm 3$ e $z = \pm 3$. Como se pode observar na Figura 3.13, no final do algoritmo a superfície será uma esfera com seis furos, ou seja, com seis frentes de expansão.

Como já foi referido, no início da triangulação duma supefície implícita existe apenas uma frente de expansão. Descrevem-se de seguida as circunstâncias

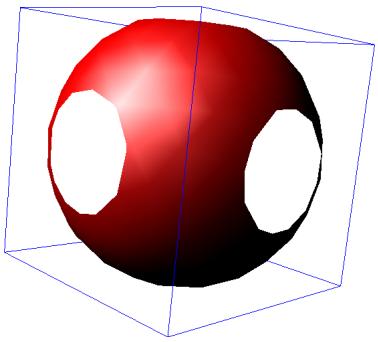

Figura 3.13: Esfera maior que a caixa de delimitação espacial.

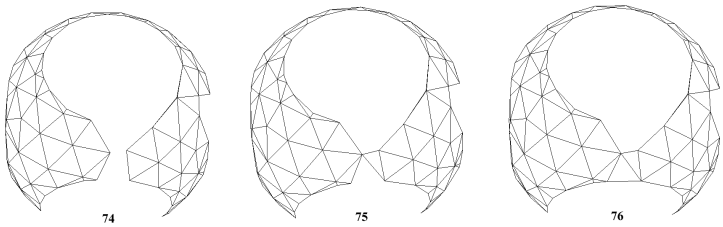

Figura 3.14: Exemplo da esfera $x^2 + y^2 + z^2 - 13 = 0$.

em que, no exemplo apresentado, ocorrem as divisões das frentes de expansão, e o modo como estas evitam a sobreposição de regiões da superfície.

Na Figura 3.12 apresenta-se um pormenor da superfície da Figura 3.13, mais precisamente, as modificações numa pequena região da superfície em três iterações de expansão consecutivas.

Na iteração 74 existe apenas uma frente de expansão Π_0 e, como se pode observar, a distância entre dois dos seus vértices é menor que δ, ou seja, reunem-se as condições que implicam a subdivisão da frente de expansão. Portanto, a divisão ocorrerá na iteração 75 antes do passo de expansão, passando assim a existir uma segunda frente de expansão Π_1 que corresponderia ao primeiro dos seis furos da esfera.

A Figura 3.14 permite ter uma visão mais clara deste mecanismo e do seu funcionamento, uma vez que, tratando-se exactamente da mesma superfície e das mesmas iterações apresentadas na Figura 3.12, dá uma perspectiva mais abrangente da esfera.

Neste exemplo, caso não se procedesse ao controlo das frentes de expansão, e consequentemente à sua divisão, ocorreriam várias sobreposições da su-

perfície. Na Figura 3.11 pode observar-se exactamente a mesma região da mesma superfície representada na Figura 3.12 numa situação em que não se procedeu ao controlo das frentes de expansão.

3.8.2 Fusão de Duas Frentes de Expansão

Nesta secção será descrita a segunda das duas situações de controlo das frentes de expansão. Ou seja, a situação em que será necessário proceder à fusão da frente de expansão actual com outra frente de expansão devido à proximidade dos seus vértices.

Pretende-se analisar a distância dos vértices da frente de expansão actual Π_0 aos pontos de todas as outras frentes de expansão. Caso existam pontos $p_{0i} \in \Pi_0$ e $p_{mj} \in \Pi_m$ tais que $||p_{0i} - p_{mj}|| < \delta$, então as frentes de expansão $\Pi_0 = (p_{01}, \ldots, p_{0N_0})$ e $\Pi_m = (p_{m1}, \ldots, p_{mN_m})$ deverão fundir-se, passando a frente de expansão actual a ser:

$$\Pi_0 = (p_{01}, \ldots, p_{0i}, p_{mj}, \ldots, p_{mN_m}, p_{m1}, \ldots, p_{mj}, p_{0i}, \ldots, p_{0N_0})$$

A nova frente Π_0 será então composta por N_0+N_m+2 vértices, uma vez que os vértices p_{0i} e p_{mj} estarão repetidos. Este processo encontra-se esquematizado na Figura 3.15.

Antes de prosseguir com a execução do algoritmo é necessário re-calcular os ângulos de expansão correspondentes aos vértices p_{0i} e p_{mj} e efectuar a expansão (ver Secção 3.9) no ponto com o ângulo de expansão mínimo. Após esta operação, as primeiras ocorrências de p_{0i} e p_{mj} deverão ser removidas de Π_0. Os vértices p_{0i} e p_{mj} deverão ser marcados de maneira a não serem envolvidos posteriormente em cálculos de distâncias.

Ainda que raramente, poderá ocorrer um caso de "maus vértices próximos". Neste caso, embora a distância entre os vértices p_{0i} e p_{mj} seja inferior a δ, não deverá proceder-se à fusão das frentes de expansão Π_0 e Π_m. Considere-se a situação representada na Figura 3.16. A detecção deste caso baseia-se na comparação do ângulo de expansão de p_{0i} com o ângulo ω definido entre p_{0i-1} e p_{mj}. Os vértices são considerados "maus vértices próximos" se ω for maior que o ângulo de expansão no ponto p_{0i}.

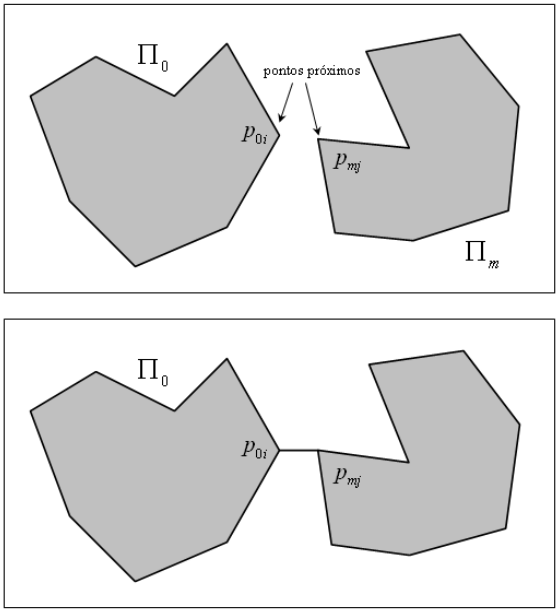

Figura 3.15: Fusão de duas frentes de expansão.

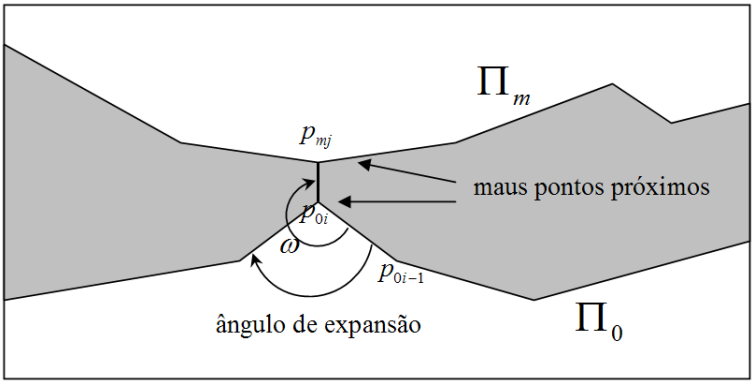

Figura 3.16: Maus vértices próximos.

3.9 Passo 5: Expansão

Chegado a este último passo do algoritmo, faltará apenas efectuar a expansão propriamente dita, ou seja, construir os novos triângulos que em cada iteração serão adicionados à malha da superfície.

Resumidamente, o processo inicia-se com a localização do vértice da frente de expansão actual com ângulo de expansão mínimo. Uma vez localizado este vértice, será necessário localizar os seus vizinhos directos, isto é, o vértice anterior e o vértice seguinte. Assim, conhecendo estes três vértices e o ângulo de expansão formado entre eles, preencher-se-á esse espaço com novos triângulos aproximadamente equiláteros cujos vértices serão pontos da superfície.

Considere-se o ponto p_{0m}, pertencente a Π_0, cujo ângulo de expansão ω seja o menor de todos os ângulos da frente de expansão. Então, a expansão da superfície processar-se-á do seguinte modo:

Algoritmo 3.6 (Expand)

1. Determinar em Π_0 o ponto anterior e o ponto posterior a p_{0m}. Sejam esse pontos, respectivamente, v_1 e v_2;

2. Determinar o número de triângulos n_t que serão construídos dentro do ângulo de expansão de p_{0m}. (Este procedimento é descrito detalhadamente na Secção 3.9.1 e requer a divisão do ângulo de expansão mínimo.);

3. Construir os novos triângulos. (A construção dos novos triângulos é apresentada na Secção 3.9.2.);

4. Actualizar a frente de expansão Π_0, ou seja, remover p_{0m} e, caso $n_t > 1$, inserir os novos pontos $p_{N+1}, \ldots, p_{N+n_t-1}$ amostrados da superfície como vértices de Π_0.

5. Os vértices $v_1, p_{N+1}, \ldots, p_{N+n_t-1}, v_2$ devem ser marcados de forma a que os seus ângulos de expansão sejam actualizados na iteração seguinte.

Segue-se uma descrição mais detalhada dos passos fundamentais do algoritmo de expansão acima transcrito.

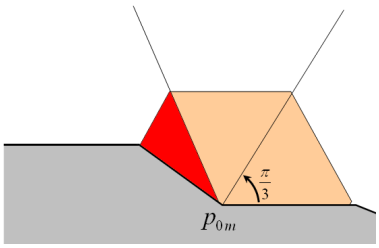

Figura 3.17: Triângulo irregular.

3.9.1 Cálculo do número de triângulos de expansão

Como se viu acima, antes da expansão é necessário determinar qual o número de triângulos a construir, isto é, quantos ângulos com aproximadamente $\pi/3$ radianos (60^o) cabem no ângulo de expansão mínimo. Pretende-se determinar, por exemplo, se determinado ângulo de expansão vai ser dividido em n ou em $n+1$ fatias, ou seja, se vão ser contruídos n ou $n+1$ novos triângulos.

Esta não é uma tarefa tão trivial como pode parecer à primeira vista. Poder-se-ia dividir o ângulo de expansão (em radianos) por $\pi/3$ e teriamos o número de triângulos. Acontece que só com uma grande dose de sorte se obtém um número inteiro, ou seja, o resultado seria, na grande maioria dos casos, um número real que teria de ser arredondado. No entanto, se se dividisse o ângulo de expansão em ângulos com $\pi/3$ radianos, o último triângulo iria ficar com o espaço restante, espaço esse que, devido ao arredondamente, seria muitas vezes bastante menor que $\pi/3$ radianos, originando triângulos pouco regulares como se pode observar na Figura 3.17. Nessa figura, o triângulo vermelho é notoriamente mais estreito que os outros dois triângulos.

Assim, desenvolveu-se um algoritmo para determinação do número de novos triângulos n_t. Este algoritmo é baseado numa optimização da divisão do ângulo de expansão em ângulos com aproximadamente $\pi/3$ radianos. Para isso, devem definir-se *a priori* dois parâmetros Δ_{max} e Δ_{min}, que serão, respectivamente, o ângulo máximo e mínimo admissíveis na divisão do ângulo de expansão ω. Assim, divide-se ω por Δ_{max} e por Δ_{min} e obtêm-se dois números de triângulos: $n_{t_{max}}$ e $n_{t_{min}}$. Depois, dividindo ω por $n_{t_{min}}$ e $n_{t_{max}}$ analisa-se qual dos dois valores origina triângulos mais próximos do valor óptimo pretendido, ou seja, $\pi/3$ radianos, e atribui-se esse valor a n_t.

Considere-se um exemplo em que ω poderá ser dividido em dois ou em três triângulos, ou seja, será necessário determinar se $n_t = 2$ ou $n_t = 3$. Como

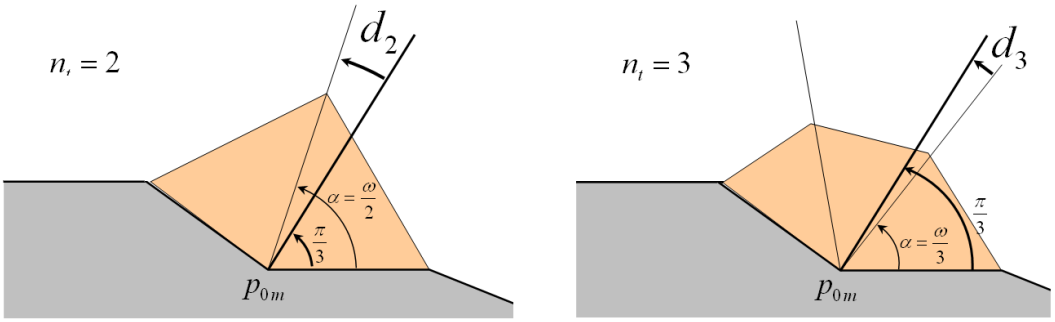

Figura 3.18: Divisão do ângulo de expansão.

se pode observar na Figura 3.18, existem duas situações possíveis. Pode observar-se ainda que o ângulo $\omega/3$, obtido quando $n_t = 3$, está mais próximo do ângulo ideal $\pi/3$ do que quando $n_t = 2$. Assim, $n_t = 3$.

Então, o algoritmo para determinação do valor óptimo de n_t será o seguinte:

Algoritmo 3.7 (NumberTriangles)

1. $n_{t_{min}} = arredondar(\omega/\Delta_{min})$
   $n_{t_{max}} = arredondar(\omega/\Delta_{max})$
2. $\alpha_{min} = \omega/n_{t_{min}}$
   $\alpha_{max} = \omega/n_{t_{max}}$
3. SE $(\frac{\pi}{3} - \alpha_{max}) \leq (\frac{\pi}{3} - \alpha_{min})$ ENTÃO

   $n_t = n_{t_{max}}$

   SENÃO

   $n_t = n_{t_{min}}$

   FIMSE
4. SE $|(\frac{\pi}{3} - \alpha_{max}) - (\frac{\pi}{3} - \alpha_{min})| \leq 0.01$ ENTÃO

   $n_t = n_{t_{max}}$

   FIMSE
5. SE $n_t = 0$ ENTÃO

   $n_t = 1$

FIMSE

6. SE $(\omega/n_t) \leq \Delta_{max}$ E $||p_{0m-1} - p_{0m+1}|| \leq \delta$ ENTÃO

   $n_t = 1$

   FIMSE

Note-se que os passos (4), (5) e (6) foram incluídos no algoritmo sem a justificação que se segue.

O passo (4) justifica-se com a prioridade que se pretende dar a triângulos mais "largos", isto é, mesmo que se divida em mais triângulos se obtenha ângulos mais próximos de $\pi/3$, se a diferença for desprezável dá-se prioridade a um menor número de triângulos que irão criar uma malha mais regular e ocupar menos recursos de memória.

O passo (5) é imprescindível uma vez que se poderá obter um valor nulo de triângulos. Isto acontece quando $\omega/\Delta_{min} < 0.5$ ou $\omega/\Delta_{max} < 0.5$. Como o número mínimo de triângulos em cada iteração é 1 faz-se a correcção de 0 para 1.

Por fim, o passo (6) obriga à criação de apenas um triângulo se a distância entre os pontos anterior e seguinte do ponto p_{0m} for inferior a δ, isto é, inferior ao tamanho definido para as arestas dos triângulos.

3.9.2 Construção dos Triângulos

Uma vez determinado um número de triângulos n_t, poder-se-á então proceder efectivamente à sua construção, ou seja, à expansão da triangulação da superfície no ponto p_{0m}. Nesta secção, considerem-se v_1 e v_2, respectivamente, os pontos anterior e seguinte de p_{0m}.

Algoritmo 3.8 (Triangulation)

Se $n_t = 1$ o processamento é trivial. Neste caso, basta adicionar à malha da superfície o triângulo formado pelos pontos (v_1, p_{0m}, v_2), como se ilustra na Figura 3.19.

Nos casos em que $n_t > 1$ o procedimento será um pouco mais complexo:

- Seja q_0 o ponto resultante da projecção ortogonal (ver Secção 3.9.3) do ponto v_1 sobre o plano P tangente à superfície no ponto p_{0m};

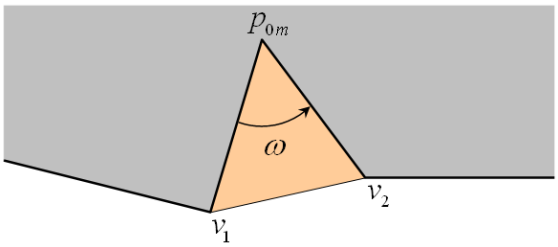

Figura 3.19: Expansão no caso em que $n_t = 1$.

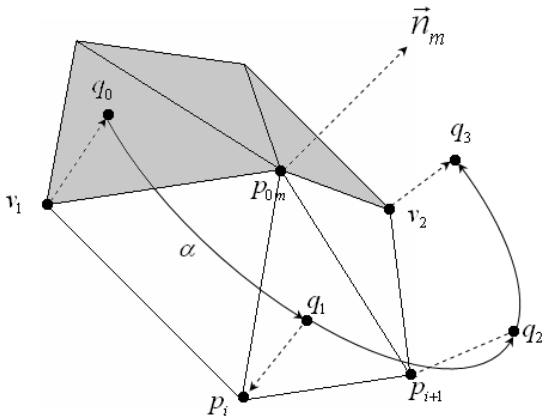

Figura 3.20: Expansão no caso em que $n_t > 1$.

- Calcula-se o valor de α (valor do ângulo de cada triângulo). Na prática, $\alpha = \omega/n_t$;
- Tomando o ponto q_0, aplica-se uma rotação (ver Secção 3.9.4) de α radianos sobre o plano P em torno do eixo $\overrightarrow{n_m}$ (vector normal à superfície no ponto p_{0m}) obtendo-se o ponto q_1;
- Finalmente, aplica-se o Corrector de Newton (ver Secção 3.3) ao ponto q_1, determinando assim o novo ponto p_{0i}. Importa referir que, no caso de p_{0i} estar fora da CDE definida, é necessário proceder ao corte da sua aresta, trazendo p_{0i} para a fronteira (ver Secção 3.9.5);
- Repete-se o processo até que tenham sido construídos $n_t - 1$ triângulos. Por fim, o triângulo final obter-se-á ligando o último ponto calculado a v_2.

Como se pode observar na Figura 3.20, na qual se ilustra o procedimento apresentado nesta secção, como resultado da expansão da superfície no ponto p_{0m} obter-se-ão os seguintes n_t novos triângulos:

$$(v_1, p_i, p_{0m}), (p_i, p_{i+1}, p_{0m}), \ldots, (p_{i+n_t-1}, v_2, p_{0m})$$

3.9.3 Projecção Ortogonal

Como foi referido na Secção 3.9.2, no processo de construção dos triângulos, será necessário efectuar projecções ortogonais de pontos da superfície sobre um determinado plano tangente. Para isso, usa-se o método de projecção ortogonal descrito em [24] que se resume de seguida.

A projecção ortogonal é um dos tipos de projecção mais simples e consiste simplesmente em projectar pontos sobre um plano de forma perpendicular (Figura 3.21).

Considere-se o plano α definido pelo ponto p_{0m} e pelo seu vector normal $\overrightarrow{n_p}$. Este plano será tangente à superfície no ponto p_{0m} e será perpendicular a $\overrightarrow{n_p}$. Seja p_i o ponto que se pretende projectar sobre α, e seja q_i o resultado dessa projecção. Calcula-se q_i utilizando a expressão (3.5).

$$q_i = p_i - ((p_i - p_{0m}) \cdot \overrightarrow{n_p})\overrightarrow{n_p} \tag{3.5}$$

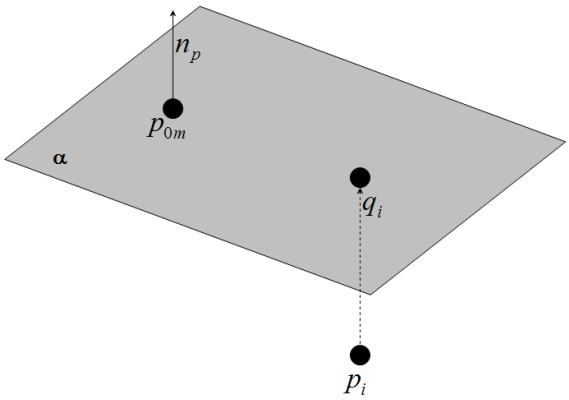

Figura 3.21: Projecção ortogonal.

3.9.4 Rotação com Eixo Arbitrário

Na Secção 3.9.2 também foi referido que será necessário efectuar rotações de alguns pontos em torno de um determinado eixo, mais concretamente, em torno do eixo definido pelo vector normal do ponto onde se procede à expansão da superfície em cada iteração. Trata-se, portanto, de uma rotação em torno de um eixo arbitrário, ou seja, um eixo que não é paralelo a nenhum dos eixos coordenados.

Em grande parte dos textos de computação gráfica, decompõe-se este tipo de rotação numa sequência de transformações individuais: translação do ponto para a origem, determinação de três ângulos de rotação relativos aos três eixos coordenados, e uma translacção simétrica final que desfaça a primeira. Este método, embora seja eficaz, implica várias multiplicações de matrizes, o que o torna bastante pesado e complexo.

Nesta secção, apresenta-se um método baseado em álgebra vectorial que efectua a rotação em torno de um eixo arbitrário como uma transformação única e não como uma sequência de várias transformações [24].

Considere-se que o eixo de rotação é definido pelo ponto p_{0m} e pelo seu vector normal $\overrightarrow{n_p}$. Pretende-se efectuar uma rotação de α radianos de um ponto q_i em torno do eixo referido, de maneira a obter o ponto q_{i+1}, como se pode observar na Figura 3.22. Tome-se o vector $\overrightarrow{v}$ como o vector definido entre o ponto p_{0m} e o ponto q_i. O vértice q_{i+1} é, assim, dado pela seguinte fórmula:

$$q_{i+1} = p_{0m} + (\cos\alpha)\overrightarrow{v} + (1 - \cos\alpha)(\overrightarrow{v} \cdot \overrightarrow{n_p})\overrightarrow{n_p} + (\sin\alpha)(\overrightarrow{v} \times \overrightarrow{n_p}) \quad (3.6)$$

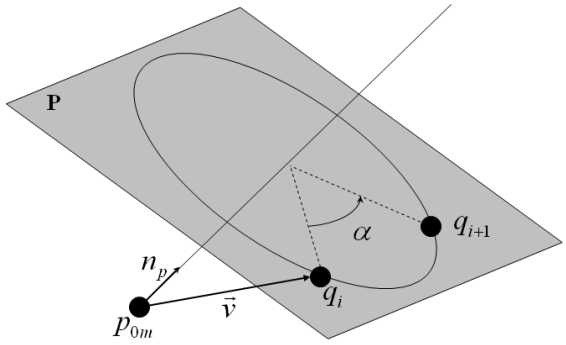

Figura 3.22: Rotação com eixo arbitrário.

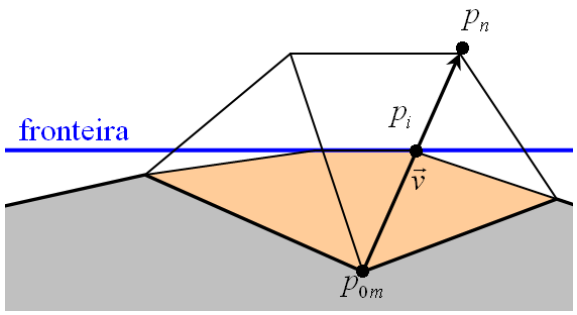

Figura 3.23: Limitação da expansão pela CDE.

3.9.5 Limitação da Expansão

Como já foi referido na Secção 3.2.4, existem superfícies para as quais será necessário definir uma CDE que limite a sua expansão. Nesta secção apresenta-se o procedimento usado para efectuar a limitação na construção dos triângulos (ver Secção 3.9.2).

Resumidamente, far-se-á o recorte das arestas dos triângulos (segmentos de recta) em que um dos extremos esteja fora da caixa de fronteira definida, passando o extremo da aresta a ser o ponto de intersecção desta com a face da CDE.

Considere-se p_{0m} o ponto de expansão e o triângulo (p_i, p_n, p_{0m}). Se p_n for exterior à CDE, será necessário recortar o segmento $\overline{p_{0m}p_n}$ obtendo um novo segmento $\overline{p_{0m}p_i}$ em que $p_i = (p_{i_x}, p_{i_y}, p_{i_z})$ é o ponto de intersecção do segmento com a fronteira obtido usando o algoritmo que se apresenta de seguida, como se ilustra na Figura 3.23.

Algoritmo 3.9 (CDE)

Sendo a CDE definida pelas coordenadas (x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}) e o vector $\overrightarrow{v} = (v_x, v_y, v_z)$ o vector entre p_{0m} e p_n tem-se:

1. SE $v_x > 0$ ENTÃO

   $$t = min(t, x_{max} - p_{0m_x}/v_x)$$

   SENÃO

   $$t = min(t, x_{min} - p_{0m_x}/v_x)$$

   FIMSE

2. SE $v_y > 0$ ENTÃO

   $$t = min(t, y_{max} - p_{0m_y}/v_y)$$

   SENÃO

   $$t = min(t, y_{min} - p_{0m_y}/v_y)$$

   FIMSE

3. SE $v_z > 0$ ENTÃO

   $$t = min(t, z_{max} - p_{0m_z}/v_z)$$

   SENÃO

   $$t = min(t, z_{min} - p_{0m_z}/v_z)$$

   FIMSE

4. $p_{i_x} = p_{0m_x} + t \cdot \overrightarrow{v_x}$
   $p_{i_y} = p_{0m_y} + t \cdot \overrightarrow{v_y}$
   $p_{i_z} = p_{0m_z} + t \cdot \overrightarrow{v_z}$

Os pontos p_i que resultem do recorte pela fronteira devem ser marcados de maneira a não serem tidos em conta como pontos efectivos da frente de expansão nas próximas iterações do algoritmo.

3.10 Considerações Finais

Embora inspirado no trabalho de Hartmann [10], o algoritmo aqui descrito faz uso de sub-algoritmos distintos e mais simplificados para o caso da triângulação.

Aliás, diga-se de passagem, que a implementação de Hartmann não funcionou numa implementação prévia feita no início destes trabalhos, o que se deve talvez a uma interpretação errónea da nossa parte. Por exemplo, o cálculo do ângulo de expansão é, nesta tese, feito duma forma bastante mais elegante e subtil.

Além disso, houve também influência do trabalho de Morgado [18] no que respeita à invariância do sinal de algumas funções que descrevem superfície implícitas. Esta percepção da invariância do sinal não transparece no trabalho de Hartmann, nem sequer na maioria dos trabalhos em superfícies implícitas.

Foi interessante constatar que o algoritmo aqui proposto funciona independentemente da variação do sinal da função, o que se justifica pelo facto deste método numérico usar uma só estimativa, em vez de duas.

Capítulo 4

Implementação e Testes

4.1 Introdução

A proposta de um algoritmo para representação de superfícies implícitas leva, obrigatoriamente, ao desenvolvimento do respectivo protótipo de testes, ou seja, à criação de uma aplicação real capaz de demonstrar, na prática, o funcionamento do algoritmo. Neste capítulo, será apresentado um protótipo desenvolvido com o objectivo de implementar e testar o algoritmo proposto no capítulo anterior.

4.2 Ferramentas Utilizadas

Na implementação do algoritmo descrito no capítulo anterior, foi utilizada uma panóplia de ferramentas de distribuição livre e gratuita.

A integração destas ferramentas numa única plataforma de trabalho permitiu concentrar as atenções no desenvolvimento do algoritmo propriamente dito, e não desperdiçar tempo com problemas ou questões para os quais já existissem soluções implementadas.

O princípio adoptado é simples: não investir tempo e esforço a programar passos do algoritmo para os quais já existisse uma implementação disponível, ou seja, a integração e re-utilização de aplicações. Este princípio permitiu acelerar consideravelmente o processo de desenvolvimento do protótipo.

4.2.1 Java 3D

A API Java 3D, como a própria designação sugere, é um pacote de software de distribuição livre e gratuita, usado no desenvolvimento de aplicações e *applets* 3D em Java. Resumidamente, fornece mecanismos de alto nível para criação, manipulação e renderização de cenas tridimensionais.

Esta API faz parte de um conjunto de APIs designado por JavaMedia, disponibilizado para a maioria das plataformas existentes. Assim, a Java 3D trouxe para a computação gráfica as vantagens inerentes ao universo Java [25]:

- Independência de plataforma, isto é, "*write once, run anywhere*";
- Integração com a Internet, uma vez que, as aplicações e *applets* que utilizam a API Java 3D têm acesso a todas as classes base da *framework* Java.

Outro dos motivos que justificam a adopção da API Java 3D, em detrimento de outros pacotes de software para computação gráfica, foi o de dispor de uma ferramenta de alto nível com orientação a objectos, que permitisse o desenvolvimento rápido de aplicações 3D com um elevado grau de sofisticação.

Ao nível mais baixo, os elementos gráficos da API Java 3D sintetizam as melhores ideias encontradas em pacotes como Direct3D, OpenGL, QuickDraw3D e XGL. Analogamente, os seus elementos de mais alto nível sintetizam as melhores ideias encontradas em várias sistemas 3D baseados na criação de cenas. Estamos, assim, perante o melhor de dois mundos.

Para uma introdução mais detalhada a esta API recomenda-se a consulta do guia de referência [25].

4.2.2 Java Symbolic Computing Library (jscl)

O **jscl-meditor** é um pacote de sofware *open source*, de utilização livre e gratuita, para computação simbólica em aplicações Java. Assim, as necessidades de computação simbólica, concretamente a factorização de expressões de superfícies implícitas, e a adopção da API Java 3D na implementação do algoritmo proposto, justificam a sua integração no protótipo implementado.

No excerto de código que se segue exemplifica-se a utilização do *jscl* na factorização da expressão da superfície implícita dada por `f`:

```
jscl.math.Expression exp = null;
String fact = null;

try {

    exp = Expression.valueOf(f);
    fact = exp.factorize().toString();

} catch(Exception e) {

    System.err.println(e.getMessage());

}
```

O resultado da factorização será, já na forma de uma `String`, o valor de `fact` após a chamada ao método `factorize`.

Na sua versão original, o **jscl-meditor** integra um pequeno editor matemático (**meditor**) que faz a interface com o utilizador. No entanto, e como este editor não era necessário para a implementação do algoritmo, procedeu-se à recompilação do pacote sem o referido editor, facto pelo qual se usa simplesmente a designação **jscl** e não a sua designação oficial **jscl-meditor**.

Embora na implementação do algoritmo se utilize o *jscl* apenas para efectuar factorizações, esta biblioteca disponibiliza muitas outras funcionalidades. Para conhecer mais pormenores sobre esta biblioteca aconselha-se a consulta de [12], onde também se poderá fazer o seu download gratuito.

4.2.3 Java Expression Parser (JEP e DJEP)

JEP é uma API Java para manipulação e cálculo de expressões matemáticas. Com esta biblioteca é possível, a partir de uma expressão em formato `String`, e dados os valores dos argumentos, obter de imediato o seu valor. A JEP suporta expressões com variáveis, constantes, operadores lógicos e artiméticos, funções trigonométricas e logarítmicas, entre outras.

Como exemplo, considere-se uma função `f` de $\mathbb{R}^3$ em $\mathbb{R}$ e um ponto `p` pertencente a $\mathbb{R}^3$. O método `value` apresentado de seguida devolve o valor da função no ponto `p`:

```
public double value(Point3d p) {

    JEP jep = new JEP();
    jep.addStandardFunctions();
    jep.addStandardConstants();

    jep.addVariable("x", p.x);
    jep.addVariable("y", p.y);
    jep.addVariable("z", p.z);

    jep.parseExpression(f);

    return jep.getValue();
}
```

No método anterior, começa-se por criar um *parser* JEP com as funções e constantes estandardizadas. De seguida definem-se as variáveis da função, isto é, `x`, `y` e `z`, e atribui-se-lhes os valores das coordenadas respectivas do ponto `p`. Por fim, calcula-se o valor da função com a chamada ao método `parseExpression`.

Por sua vez, a DJEP é uma biblioteca que contém um conjunto de extensões à JEP, isto é, acrescenta-lhe uma colecção de funcionalidades, onde talvez a mais importante, e sem dúvida a mais relevante para o presente propósito, seja o cálculo diferencial.

Um dos passos do algoritmo implica o cálculo do gradiente de uma função em $\mathbb{R}^3$, o que obriga ao cálculo das suas derivadas parciais. Sendo `f` a expressão da função, o excerto de código seguinte exemplifica a utilização da DJEP na obtenção das expressões das suas derivadas parciais em `x`, `y` e `z`:

```
djep = new DJep();
djep.addStandardConstants();
djep.addStandardFunctions();
djep.setImplicitMul(true);
djep.addStandardDiffRules();

try {

    Node node  = djep.parse(f);
```

```
    Node diffx = djep.differentiate(node, "x");
    Node dx    = djep.simplify(diffx);
    strDx      = djep.toString(dx);

    Node diffy = djep.differentiate(node, "y");
    Node dy    = djep.simplify(diffy);
    strDy      = djep.toString(dy);

    Node diffz = djep.differentiate(node, "z");
    Node dz    = djep.simplify(diffz);
    strDz      = djep.toString(dz);

} catch(ParseException e) {
    System.err.println("ERROR: " + e.getMessage());
}
```

Assim, `strDx`, `strDy` e `strDz` serão as expressões das derivadas parciais de `f`. Para obter as três componentes do vector gradiente num ponto específico, basta aplicar o método `value`, apresentado anteriormente, a cada uma das expressões das derivadas.

As API's JEP e DJep têm vindo a ser vastamente utilizadas, e consequentemente testadas, por inúmeros utilizadores e entidades em todo o mundo. Entre essas entidades contam-se: NASA Jet Propulsion Laboratory, MySQL AB, Cisco Systems Inc., Bajaj Allianz General Insurance Co., SED Systems Inc., Object Reservoir, ERIN Engineering and Research Inc., NovoDynamics Inc., Ebase Technology Ltd., ChemAxon Ltd., Technology High School, Saiph, entre outras.

Para uma apresentação completa das API's JEP e DJEP aconselha-se a consulta de [2] e [1].

4.3 A Implementação do Algoritmo: iSurf

Com a finalidade de implementar e testar o algoritmo proposto para representação de superfícies implícitas, considerou-se indispensável o desenvolvimento de uma aplicação experimental. Assim, desenvolveu-se uma *package* (ou biblioteca) Java à qual se deu o nome de **iSurf** (**i**mplicit **surf**aces).

A *package* **iSurf** foi implementada tirando partido das pontecialidades da orientação a objectos do Java. Disponibiliza uma *framework* completa e independente da plataforma para visualização de superfícies implícitas. Embora o protótipo desenvolvido inclua uma interface gráfica acente sobre *Swing*, a biblioteca permite uma utilização quer em outras aplicações independentes Java, quer em *applets* sobre Internet.

Assim, a *package* desenvolvida pode ser utilizada em qualquer aplicação ou *applet* Java 3D de forma simples e rápida (encapsulando a implementação do algoritmo) através da importação de um ficheiro JAR. Este mecanismo torna o seu funcionamento completamente transparente para os utilizadores que terão apenas de definir os parâmetros iniciais e a expressão da superfície a visualizar. A biblioteca encarrega-se do processamento, devolvendo um objecto Java 3D. O objecto devolvido será a própria superfície.

Segue-se uma apresentação da arquitectura do protótipo de testes **iSurf**, bem como da sua estrutura de classes.

4.3.1 Arquitectura Geral

Seguindo os princípios de integração e re-utilização de software, a aplicação desenvolvida obedece a uma arquitectura modular, isto é, integra diferentes módulos de software na plataforma Java e Java 3D (Figura 4.3.1. No entanto, não se pode afirmar que se esteja perante uma aquitectura vertical rigidamente estratificada.

Na base de toda a aplicação, e uma vez que se trata de uma aplicação totalmente implementada em Java, está a JVM (*Java Virtual Machine*). Assim, garante-se o acesso a todas as funcionalidades da linguagem, principalmente à sua vocação para aplicações para a Internet, e a sua independência de plataforma.

Sendo a API Java 3D uma extensão à linguagem Java, grande parte da aplicação integra recursos provenientes deste pacote de software. Concretamente, os elementos geométricos utilizados pela **iSurf**, ou seja, pontos e triângulos, assim como as operações geométricas e vectoriais utilizadas, são disponibilizados pelo pacote Java 3D, concretamente, pela sua biblioteca **javax.vecmath**.

Sendo a *package* **iSurf** o núcleo da aplicação, apenas este módulo interage directamente com o pacote **JEP**. A aplicação recorre a este pacote de software para efectuar o *parsing* de funções, isto é, calcular o valor de uma função

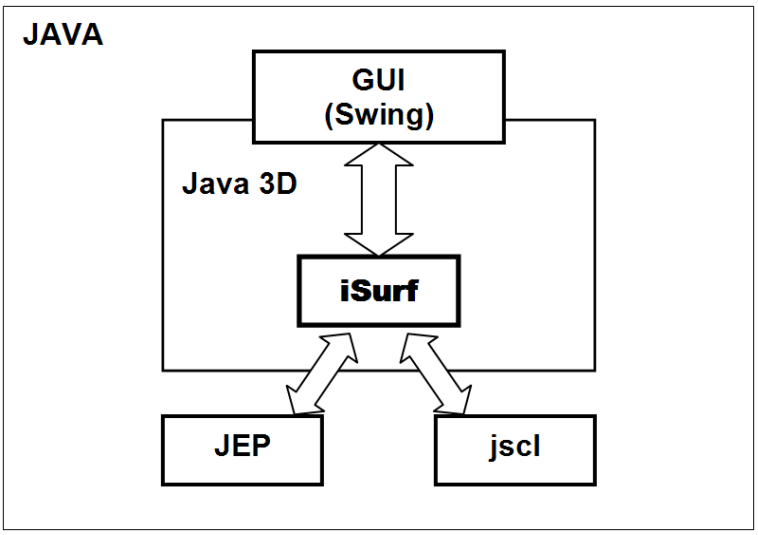

Figura 4.1: Arquitectura geral da aplicação

num dado ponto, e para efectuar todos os cálculos diferenciais necessários na implementação do algoritmo.

Também apenas a *package* **iSurf** interage directamente com o pacote **jscl**. A aplicação utiliza este pacote de software para, a partir de uma expressão introduzida pelo utilizador no GUI da aplicação, efectuar a decomposição simbólica da superfície que se pretende visualizar.

A face visível de toda a aplicação é a sua GUI (*Graphical User Interface*). No caso concreto da implementação aqui apresentada, desenvolveu-se uma GUI sobre uma aplicação Java independente. Mais precisamente, recorreu-se ao pacote **Swing** para a construção da janela da interface da aplicação. No entanto, o pacote **iSurf** poderá ser usado da mesma forma por *applets* Java.

Basicamente, no protótipo implementado, é através da GUI que o utilizador introduz a expressão da superfície. É também aí que, após o processamento efectuado pelo **iSurf**, a superfície será visualizada numa *canvas* (tela) Java 3D integrada num contentor Swing.

4.3.2 Diagrama de Classes

Na implementação do algoritmo, e tendo adoptado a plataforma de desenvolvimento Java e Java 3D, optou-se por uma abordagem totalmente orientada a objectos. Assim, foi desenvolvida a *package* **iSurf** que faz o

mapeamento dos elementos basilares do algoritmo numa colecção de classes Java.

Na Figura 4.2 pode observar-se o diagrama UML da *package* **iSurf**. A classe principal é a classe `Surface` que, como a designação indica, representa a própria superfície. Assim, para construir uma superfície basta instanciar esta classe através duma chamada ao seu construtor.

A classe `Surface` contem uma lista (`_components`) de objectos que são instâncias da classe `SurfaceComponent`. Estes objectos representam as componentes simbólicas da superfície principal. Eventualmente, caso não se esteja na presença de uma superfície degenerada, esta lista contém apenas um objecto.

As classes `SurfaceComponent` e `BoundingBox` são especializações da classe `Shape3D` da *package* `javax.media.j3d`, isto é, as instâncias destas classes são adicionadas à cena Java 3D como se de objectos desta classe se tratassem. A classe `BoundingBox` corresponde à CDE (Caixa de Delimitação Espacial) da superfície.

Cada superfície simbólica é composta por uma malha de triângulos, concretamente, na forma de uma lista de objectos da classe `Triangle`. Esta classe é uma especialização da classe `IndexedTriangleArray` da *package* `javax.media.j3d`, cujas instâncias podem ser adicionadas a objectos da classe `Shape3D`, como é o caso.

Outra das propriedades de cada componente simbólica da superfície é a função que a representa, ou seja, uma instância da classe `Function` que é a abstração de uma função algébrica (polinomial). Esta classe contém ainda instâncias de `JEP` e `DJep` (ver 4.2.3) que são utilizadas nos cálculos necessários ao processamento do algoritmo.

De acordo com as necessidades do algoritmo, cada componente simbólica da superfície contém ainda uma lista de polígonos de expansão (`_polygons`). Esta lista contém objectos da classe `FrontPolygon`, classe esta que implementa todas as operações necessárias a este tipo de objectos.

Por sua vez, cada objecto `FrontPolygon` é composto por uma colecção de objectos `FrontPoint` que, de acordo com os conceitos apresentados previamente no contexto do algorimo, representam os seus pontos de fronteira. Cada `FrontPoint` é uma especialização da classe `Point3d` (pontos com coordenadas do tipo `double`) da *package* `javax.vecmath`.

Finalmente, a classe `Gui` implementa uma interface gráfica com o utilizador.

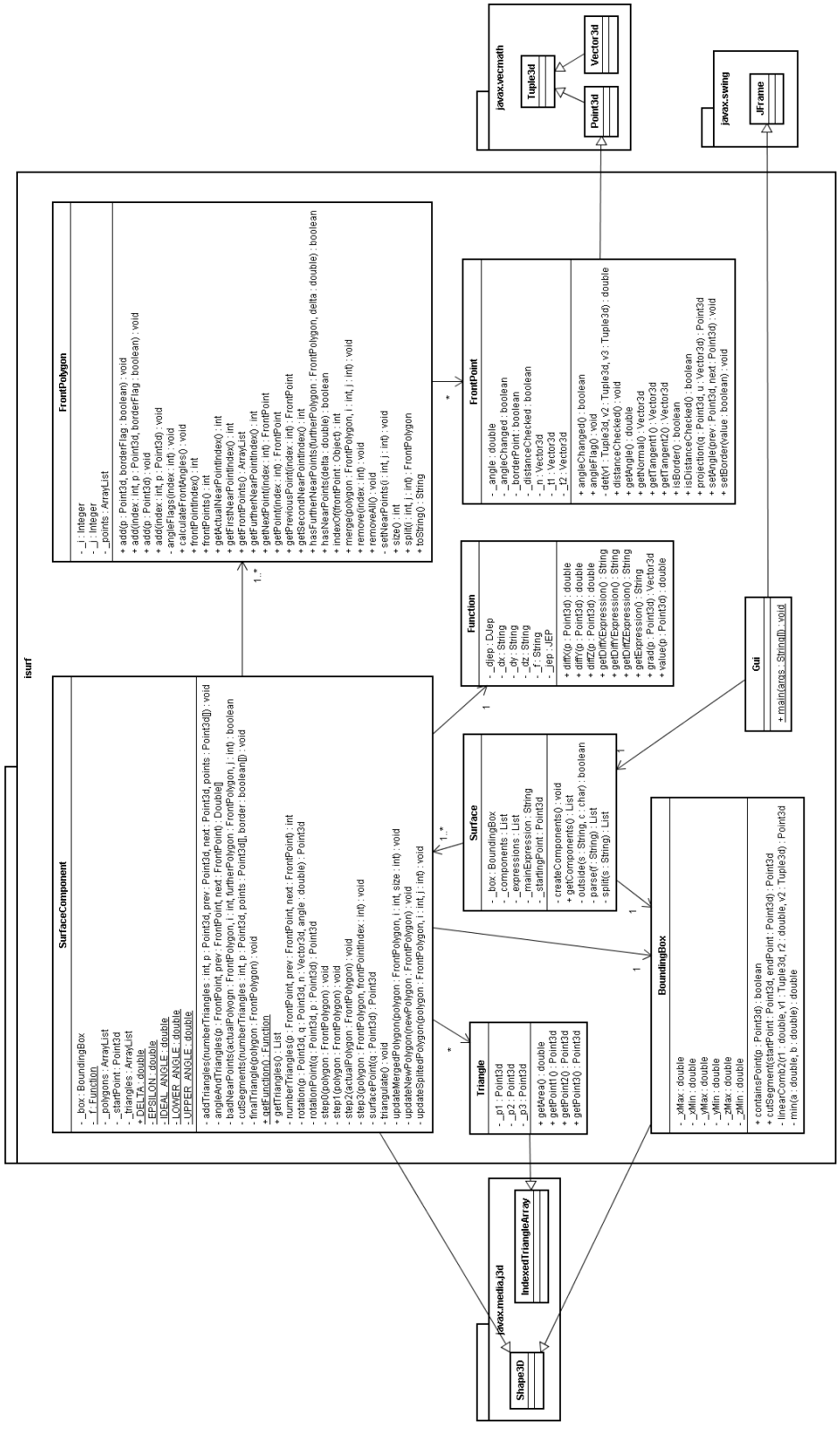

Figura 4.2: Diagrama de Classes

Neste caso, optou-se por utilizar uma janela `JFrame` do pacote `Swing`. No entanto, poderia optar-se por qualquer outro tipo de interface gráfica Java como, por exemplo, uma *applet*. É nesta classe que é instanciada a classe `Surface` adicionando o objecto a uma tela Java 3D.

4.3.3 GUI (*Graphical User Interface*)

Umas das formas de visualizar cenas Java 3D é através da sua integração numa interface gráfica Swing. Isto permite que a aplicação possa fazer uso de objectos visuais como botões, caixas de texto ou menus. As cenas em Java 3D serão visualizadas numa sub-janela dentro da janela principal da aplicação [15].

Assim, optou-se por incluir na *package* **iSurf** uma interface gráfica desenvolvida em Swing. Como se ilustra na Figura 4.3, esta interface é composta por uma janela principal, por uma caixa de texto destinada à introdução da expressão da superfície, por um botão que inicia o processamento da superfície e, finalmente, por uma tela 3D onde é representada a superfície e a CDE (Caixa de Delimitação Espacial).

No GUI apresentado existe ainda a possibilidade de interacção entre o utilizador e a tela 3D através de eventos do rato, ou seja, utilizando os botões do rato é possível efectuar rotações ou translacções da superfície e aproximar ou afastar o ponto de vista do utilizador, ou seja, *zooming*.

4.3.4 Testes de Desempenho

A maneira como um novo algoritmo se comporta numa aplicação real é sempre uma questão de grande importância. Aspectos como rapidez de processamento, gestão de recursos de memória e até mesmo a qualidade da imagem produzida, permitem medir o comportamento de um algoritmo em termos do seu desempenho.

Nesta secção apresentam-se alguns testes de desempenho realizados para a aplicação desenvolvida. Estes, por motivos de limitação temporal, não foram testes de carga intensivos, servem apenas para ter uma ideia global do desempenho do algoritmo na visualização de superfícies degeneradas e não degeneradas, com e sem variação de sinal.

Importa referir que o facto de um programa escrito em Java ser interpretado

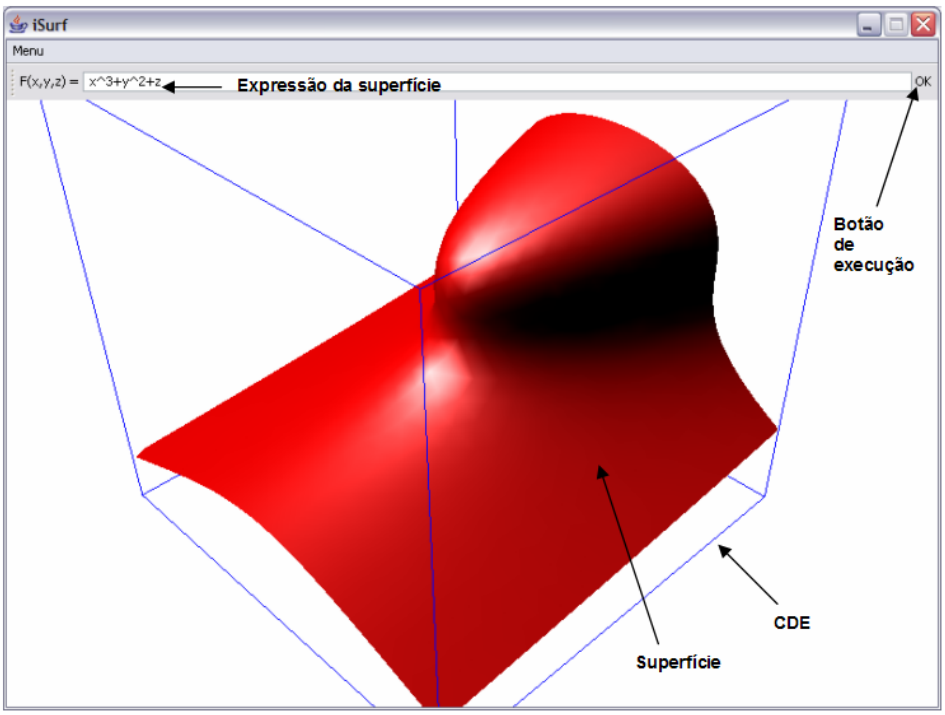

Figura 4.3: GUI

por uma máquina virtual, e não usar código nativo, leva a uma perda de desempenho em comparação com linguagens como C ou C++. Assim, os tempos apresentados poderiam, eventualmente, ser melhorados numa implementação do mesmo algoritmo numa linguagem mais eficiente. Perder-se-ia, no entanto, as vantagens da flexibilidade, portabilidade e independência da plataforma que levaram à escolha da linguagem Java.

Em termos de *hardware*, na realização de todos os testes aqui apresentados, utilizou-se um computador pessoal portátil com processador Intel Pentium 4 a 2.4 GHz, 736 MB de memória RAM e placa gráfica ATI RADEON IGP 340M com 32 MB de memória. Em termos de *software* de sistema, utilizou-se o sistema operativo Microsoft Windows XP Professional com máquina virtual Java Sun J2SE 5.0.

4.3.4.1 Quádricas

Entre as superfícies que mais vulgarmente surgem na forma implícita encontram-se as quádricas. Esferas, elipsóides e parabolóides são algumas da

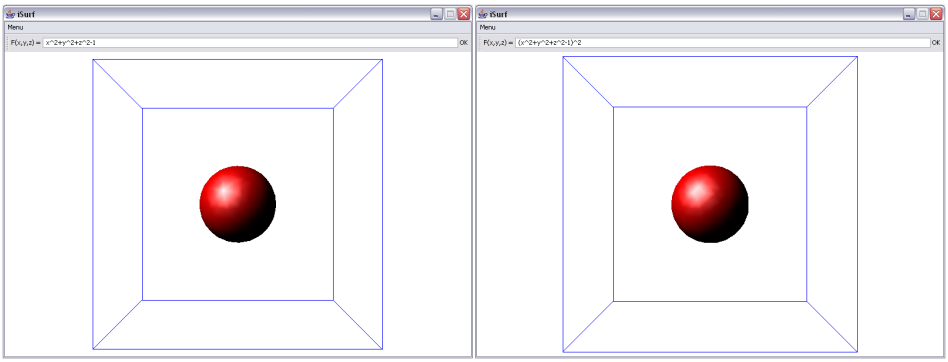

Figura 4.4: Esfera com variação de sinal (a) e sem variação de sinal (b)

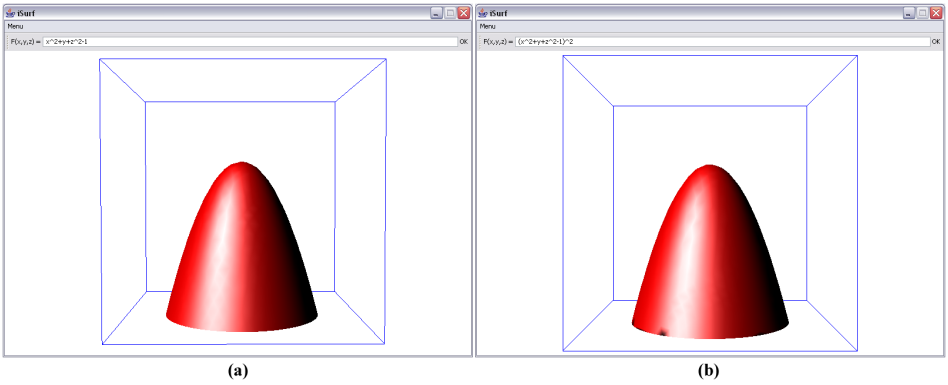

Figura 4.5: Parabolóide com variação de sinal (a) e sem variação de sinal (b)

primitivas mais comuns em sistemas 3D, pelo que se pensou ser importante testar o comportamento do algoritmo perante este tipo particular de superfícies.

Embora estas superfícies sejam relativamente simples de tratar, e até mesmo triviais para a maioria dos métodos de visualização, o mesmo poderá não acontecer caso não exista variação do sinal da sua função. Tal acontece, por exemplo, ao elevar a expressão da superfície a uma potência par, o que torna o seu valor sempre positivo.

Com o objectivo de analisar o comportamento da aplicação perante superfícies morfologicamente semelhantes com e sem variação de sinal, realizaram-se alguns testes com esferas e parabolóides simples, como se ilustra nas Figuras 4.4 e 4.5.

Chegou-se à conclusão que as diferenças são visualmente quase imperceptívies,

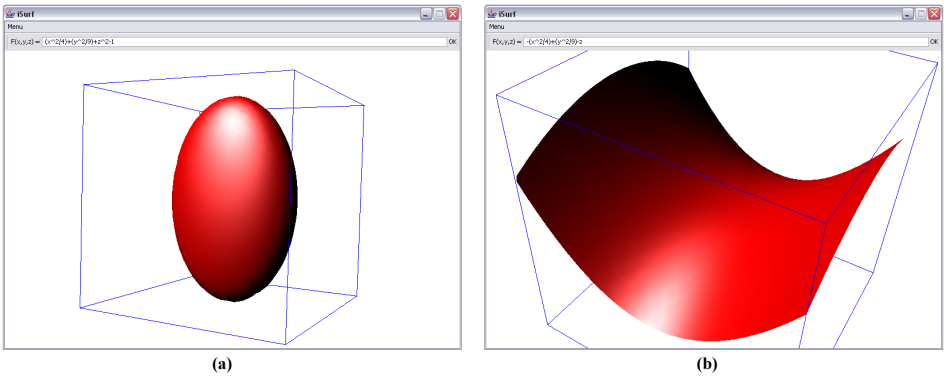

Figura 4.6: Elipsóide (a) e parabolóide hiperbólico (b)

sendo apenas de registar o facto de que nos casos em que não existe variação de sinal, a superfície demora ligeiramente mais a ser processada, como se pode observar na Tabela 4.1.

Tabela 4.1: Esferas e parabolóides.

Figura	Superfície	Nº Triângulos	Tempo (segundos)
4.4 (a)	$x^2 + y^2 + z^2 - 1 = 0$	430	0,711
4.4 (b)	$(x^2 + y^2 + z^2 - 1)^2 = 0$	430	1,953
4.5 (a)	$x^2 + y + z^2 - 1 = 0$	1350	2,443
4.5 (b)	$(x^2 + y + z^2 - 1)^2 = 0$	1241	4,036

Outras das quádricas de uso mais frequente em sistemas de 3D são os elipsóides. Isto deve-se ao facto de que, ao mesmo tempo que são relativamente simples de processar, permitem inúmeras possibilidades quando usados em CSG (*Constructive Solid Geometry*) e em *blending*.

Ainda que menos usados que as quádricas anteriormente referidas, existem ainda os parabolóides hiperbólicos. De forma a analisar o desempenho do algoritmo proposto, considerou-se também relevante a inclusão deste tipo de superfície nos testes realizados.

Assim, na Figura 4.6 podem observar-se os resultados visuais de dois dos testes realizados. Concretamente, pode ver-se um elipsóide (Figura 4.6(a)) e um parabolóide hiperbólico (Figura 4.6(b)). As expressões das superfícies, assim como os tempos de processamento, encontram-se na Tabela 4.2.

4.3.4.2 Superfícies Não Degeneradas

Achou-se também importante testar o comportamento da aplicação na visualização de superfícies não degeneradas que envolvessem funções polinomiais de grau superior a 2, funções logarítmicas e funções trignométricas.

Embora do ponto de vista geométrico ou morfológico estas funções não tornem, necessariamente, as superfícies mais problemáticas de tratar, era importante averiguar se poderiam criar problemas de processamento simbólico ou no cálculo de derivadas.

Efectuaram-se testes para várias superfícies destes tipo, entre as quais uma superfície trivial de grau 3, uma superfície com funções logarítmicas e trigonométricas e uma superfície não trivial de grau 4 (ver legenda da Figura 4.8). As expressões destas superfícies e os tempos de processamento obtidos são apresentados na Tabela 4.3.

Assim, como se pode observar nas Figuras 4.7 e 4.8 (e na tabela anterior), quer em termos de visualização das superfícies, quer em termos de tempo de processamento, a aplicação apresentou uma robustez bastante satisfatória para superfícies de grau superior a 2 e com funções trigonométricas e/ou logarítmicas.

4.3.4.3 Superfícies Degeneradas

Relativamente a superfícies implícitas degeneradas efectuaram-se testes para dois casos distintos: superfícies degeneradas com componentes explícitas e superfícies degeneradas com componentes não explícitas.

Tabela 4.2: Outras quádricas.

Figura	Superfície	Nº Triângulos	Tempo (segundos)
4.6 (a)	$\frac{x^2}{4} + \frac{y^2}{9} + z^2 - 1 = 0$	1470	2,925
4.6 (b)	$-\frac{x^2}{4} + \frac{y^2}{9} - z = 0$	1515	2,944

Tabela 4.3: Superfícies implícitas não degeneradas.

Figura	Expressão	Nº de Triângulos	Tempo (segundos)
4.7 (a)	$x^3 + y = 0$	2749	4,807
4.7 (b)	$ln(x) + cos(y) - z = 0$	1314	3,335
4.8	(ver legenda)	1926	10,245

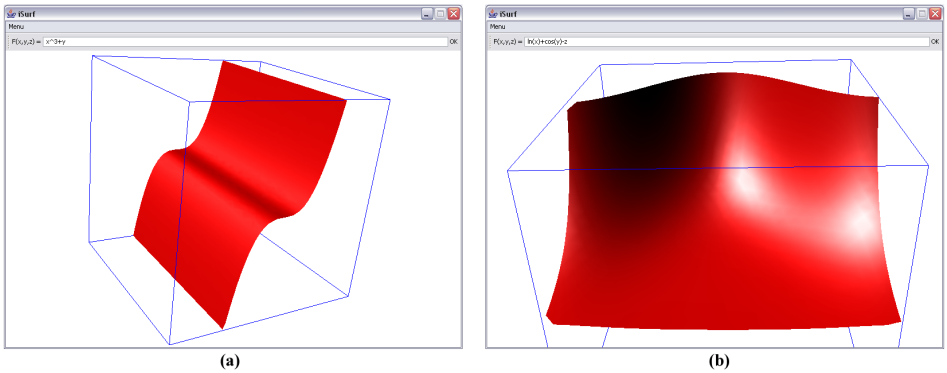

Figura 4.7: $x^3 + y = 0$ (a) e $ln(x) + cos(y) - z = 0$ (b)

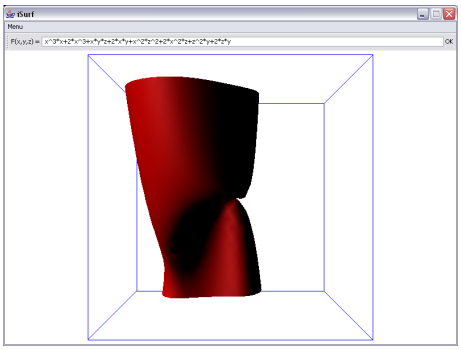

Figura 4.8: $x^4 + 2x^3 + xyz + 2xy + x^2z^2 + 2x^2z + z^2y + 2zy = 0$

Adoptou-se a designação "**superfície degenerada com componentes explícitas**" para superfícies em que, por observação directa da sua expressão, seja trivial identificar as suas componentes simbólicas, sem necessidade de efectuar a factorização da função. Tome-se como exemplo a superfície dada pela expressão 4.1:

$$(\sin x + y + 2) \cdot (\frac{x^2}{2} + y^2 + z^2 - 2) \cdot ((x - 1.5)^2 + (y - 1.5)^2 + z^2 - 1) = 0 \quad (4.1)$$

Por observação directa da expressão 4.1, imediatamente se identificam as três componentes simbólicas da superfície como se observa na Tabela 4.4.

Tabela 4.4: Superfície 4.1.

Componente Simbólica	Nº de Triângulos
$\sin x + y + 2$	1490
$\frac{x^2}{2} + y^2 + z^2 - 2$	1040
$(x - 1.5)^2 + (y - 1.5)^2 + z^2 - 1$	430

O processamento completo da superfície dada pela expressão 4.1 levou 7,290 segundos, e o resultado é ilustrado pela Figura 4.9(a).

Por sua vez, para as superfícies em que as componentes simbólicas só sejam identificáveis após a factorização, adoptou-se a designação "**superfície degenerada com componentes não explícitas**". Como exemplo, considere-se a superfície dada pela expressão 4.2:

$$x^3 + x^2 z + xy + yz = 0 \quad (4.2)$$

Não é possível, por observação directa da expressão 4.2, identificar imediatamente as suas componentes simbólicas. No entanto, após a factorização da mesma, identificam-se as suas duas componentes simbólicas como se pode observar na Tabela 4.5.

Tabela 4.5: Superfície 4.2.

Componente Simbólica	Nº de Triângulos
$x^2 + y$	1486
$x + z$	1728

O processamento completo da superfície dada pela expressão 4.2 levou 7,681 segundos, e o resultado é ilustrado pela Figura 4.9(b).

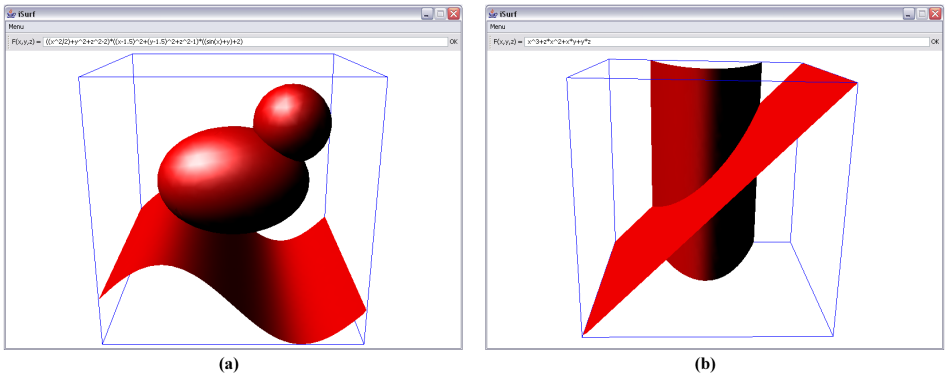

Figura 4.9: Superfície degenerada com componentes simbólicas explícitas (a) Superfície degenerada com componentes simbólicas não explícitas (b)

4.4 Considerações Finais

A concepção e implementação deste algoritmo não tinha como objectivo principal a comparação quantitativa do desempenho com outros algoritmos. Daí que a codificação do algoritmo tenha sido efectuada em Java.

Estávamos mais interessados na sua avaliação qualitativa, ou seja, o que ele era capaz ou não de fazer. Por exemplo, se o algoritmo era capaz de ultrapassar o problema das componentes invariantes ao sinal.

Constatámos que assim era, o que se deve ao algoritmo numérico usado na amostragem das superfícies, como já foi explicado no capítulo anterior.

Capítulo 5

Conclusões

5.1 Resultados Obtidos

O principal objectivo deste trabalho era desenvolver, implementar e testar um algoritmo de poligonização de superfície implícitas cujo mecanismo de processamento, ao contrário de grande parte dos método existentes, não dependesse da variação do sinal da função matemática representativa da superfície.

Ora, como se viu ao longo do presente texto, o principal objectivo foi atingido com sucesso. Foi proposto, implementado e testado um algoritmo numérico cujas amostragens se baseiam numa só estimativa, o que faz com que a amostragem e poligonização das superfícies implícitas não dependa da variação do sinal da sua função.

Considera-se também relevante o facto do algoritmo ser capaz de identificar e efectuar o processamento simbólico de superfícies implícitas degeneradas. Esta capacidade possibilita o tratamento de forma independente das componentes simbólicas deste tipo particular de superfícies implícitas. Doutra forma, algumas dessas componentes simbólicas poderiam não ser representadas.

Outras das particularidades importantes do algoritmo apresentado é a regularidade das malhas resultantes da poligonização da superfície. O facto de se utilizar um método que efectua a expansão da superfície a partir de malhas de hexágonos tendencialmente regulares, dá origem a malhas de triângulos tendencialmente equiláteros, o que melhora em muito a qualidade

das superfícies implícitas visualizadas.

Importa ainda referir que a utilização da linguagem Java, mais concretamente do pacote Java 3D, para a implementação do protótipo do algoritmo apresentado, se veio a revelar uma opção extremamente acertada. A rapidez de desenvolvimento que as suas funcionalidades de base possibilitam, aliadas à integração com pacotes externos como o JEP e o jscl, permitiram acelerar bastante o processo de implementação e testes do algoritmo.

Ainda relativamente à adopção da linguagem Java 3D, constatou-se algo bastante interessante. O desempenho da aplicação, apesar da necessária utilização de uma máquina virtual Java, revelou-se bastante positivo em termos de tempos de processamento. Além disso, torna possível tirar partido de toda a independência e portabilidade inerente à plataforma Java, assim como da sua vocação natural para aplicações para a Internet.

De uma maneira geral, e tendo em conta a natureza, contexto, enquadramento e tempo disponível para a realização do presente trabalho, isto é, uma dissertação de mestrado em Engenharia Informática, os resultados obtidos foram considerados bastante satisfatórios. Pode dizer-se que foram atingidos, na generalidade, os objectivos estabelecidos à partida.

5.2 Trabalho Futuro

A poligonização de superfícies implícitas continua, ainda, a ser um problema em aberto. Não existe ainda um algoritmo óptimo aplicável a toda e qualquer superfície implícita. Assim, também o algoritmo apresentado nesta dissertação poderá vir a ser objecto de melhoramentos futuros como, por exemplo, os que se sugerem de seguida.

5.2.1 Determinação da Intersecção de Componentes Simbólicas

Como já foi anteriormente dito, o algoritmo proposto têm a capacidade de identificar casos de superfícies implícitas degeneradas. Além disso, é ainda capaz de efectuar a decomposição simbólica duma função e processar de forma independente as várias componentes simbólicas da superfície.

Assim, e uma vez que a malha de cada componente da superfície é inde-

pendente das malhas das restantes componentes, será possível acrescentar ao algoritmo a capacidade de identificar e representar as intersecções de eventuais componentes simbólicas de uma qualquer superfície implícita degenerada.

Uma vez que as malhas das componentes das superfícies implícitas são exclusivamente compostas por triângulos, aponta-se como possível caminho para determinação das suas intersecções a aplicação do método proposto em [17] para o cálculo das intersecções entre triângulos.

5.2.2 Identificação e Representação de Singularidades

Como é sabido, existem casos de superfícies implícitas em que se verificam, por exemplo, auto-intersecções da superfície consigo própria, existência de ápices, arestas ou pontos isolados. Estes fenómenos, que dificultam a correcta representação destas superfícies, são vulgarmente designados por *singularidades*.

No entanto, pensa-se que este fenómeno poderá ser relativamente simples de detectar. Considerando que uma dada superfície é representada por uma função f, as suas singularidades, caso existam, serão as soluções do sistema de equações 5.1.

$$\begin{cases} \frac{\partial f}{\partial x} = 0 \\ \frac{\partial f}{\partial y} = 0 \\ \frac{\partial f}{\partial z} = 0 \end{cases} \tag{5.1}$$

O desafio lançado para o futuro será, portanto, o de desenvolver um método capaz de, a partir da resolução do sistema de equações (5.1), representar graficamente as soluções obtidas, ou seja, as singularidades.

Este desafio será maior nos casos em que o sistema seja possível e indeterminado, ou seja, nos casos em que as singularidades sejam curvas. O caso trivial é quando o sistema é impossível o que significa que não existem singularidades.

Bibliografia

[1] Djep - differentiation, vectors and matrices in jep. (veja-se http://www.singularsys.com/jep/doc/html/djep/).

[2] Jep - java math expression parser. (veja-se http://www.singularsys.com/jep/).

[3] E. Allgover and S. Gnutzmann. An algorithm for piecewise linear aproximation of implicitly defined two-dimensional surfaces. *SIAM Journal on Numerical Analysis*, 24(2):452–649, 1987.

[4] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[5] J. Bloomenthal. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

[6] M. Brodzik. He computation of simplicial approximations of implicit defined p-manifolds. *Computers and Mathematics with Applications*, 36(6):389–423, 1998.

[7] A. Cusatis Junior. Raycasting intervalar de superfícies implícitas com aritmética afim. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 1999.

[8] D. Dobkin, S. Levy, and A. Wilks. Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics*, 9(4):389–423, 1990.

[9] M. Hall and J. Warren. Adaptive polygonization of implicitly defined surfaces. *IEEE Computer Graphics and Applications*, 10(6):33–42, 1990.

[10] E. Hartmann. A marching method for the triangulation os surfaces. *The Visual Computer*, 14(3):95–108, 1998.

[11] C. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, 1992. (veja-se http://www.cs.purdue.edu/homes/cmh/distribution/books/geo.html).

[12] R. Jolly and E. Stamatogiannakis. jscl-meditor - java symbolic computing library and mathematical editor. (veja-se http://jscl-meditor.sourceforge.net/).

[13] Q. Li, D. Wills, R. Phillips, W. Viant, J. Griffiths, and J. Ward. Implicit fitting using radial basis functions with ellipsoid constraint. *Computer Graphics Forum*, 23(1):55–70, 2004.

[14] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[15] K. Meissner. Integrating java 3d and swing. (veja-se http://java3d.j3d.org/tutorials/quick_fix/swing.html).

[16] R. Melville and D. Mackey. New algorithm for two-dimensional numerical continuation. *Computers and Mathematics with Applications*, 30(1):31–46, 1995.

[17] T. Möller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2):25–30, 1997.

[18] José Francisco Morgado. *Poligonização de Curvas e Superfícies Implícitas Não-Homogéneas com Preservação Topológica*. PhD thesis, Universidade da Beira Interior, 2005.

[19] M. Muller, H.and Stark. Adaptive generation of surfaces in volume data. *The Visual Computer*, 9(4):182–199, 1993.

[20] A. Requicha. Representation of rigid solids: Theory, methods and systems. *ACM Computing Surveys*, 12(4):437–464, 1980.

[21] W. Rheinboldt. On a moving frame algorithm and the triangulation of equilibrium manifolds. *ISNM79: Bifurcation: Analysis, Algorithms, Applications*, pages 256–267, 1987.

[22] W. Rheinboldt and J. Burkardt. A locally parameterized continuation process. *ACM Transactions and Mathematical Software*, 9(2):215 – 235, 1983.

[23] F. Schmitt, B. Barsky, and W. Du. An adaptive subdivision method for surface-fitting from sampled data. *Computer Graphics*, 20(4):179–188, Julho 1986.

[24] P. Schneider and D. Eberly. *Geometric Tools for Computer Graphics*. Morgan Kaufmann, 2003.

[25] H. Sowizral, K. Rushforth, and M. Deering. *The Java 3D (TM) API Specification (2nd Edition)*. Sun Microsystems, 2001.

[26] B. Stander and J. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. *ACM SIGGRAPH Computer Graphics*, 31(3):279–286, 1997.

[27] L. Velho, L. Figueiredo, and J. Gomes. A unified approach for hierarchical adaptative tesselation of surfaces. *ACM Transactions on Graphics*, 18(4):329–360, 1999.

[28] L. Velho, J. Gomes, and L.H. Figueiredo. *Implicit Objects in Computer Graphics*. Springer Verlag, 2002.

[29] E. Weisstein. Fundamental theorem of algebra. *MathWorld*, 2005. (veja-se http://mathworld.wolfram.com/FundamentalTheoremofAlgebra.html).

[30] E. Weisstein. Polynomial factorization. *MathWorld*, 2005. (veja-se http://mathworld.wolfram.com/PolynomialFactorization.html).

[31] J. Zhou, N. Patrikalakis, S. Tuohy, and X. Ye. Scattered data fitting with simplex splines in two and three-dimensional spaces. *The Visual Computer*, 13(7):295–315, 1997.