Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

Tecnologia Java na representação de curvas e superfícies paramétricas

Por

Vasco Alexandre dos Santos Dionísio Domingos

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Informática

Orientador: Professor Doutor Manuel Próspero dos Santos

Lisboa

12 de Fevereiro de 2009

Ao Francisco e ao Tomás,

Olhando para a História facilmente identificamos as marcas deixadas pelo Homem no uso do poder que lhe é conferido pela Ciência. Que a vossa geração o saiba utilizar bem, melhor do que outras no passado… É o meu desejo e, simultaneamente, a minha convicção.

AGRADECIMENTOS

Um trabalho desta natureza é feito de avanços e recuos, de construção e desconstrução, de alegria e desespero. Este caminho, aparentemente árduo, torna-se muito mais fácil quando se tem o privilégio de se trabalhar com alguém, que além de dotado de uma competência técnica invulgar, possui também qualidades enquanto ser humano que o tornam numa pessoa que recordarei para sempre com reconhecimento e gratidão. Ao meu orientador, Professor Doutor Manuel Próspero dos Santos o meu profundo agradecimento por todo o interesse, paciência, ajuda e frontalidade dispensados ao longo deste processo.

Aos meus pais agradeço todo o apoio e carinho que me deram e os valores que me transmitiram na mais dificíl tarefa que um ser humano pode empreender, a de educar um filho.

À Maria João, por estar sempre presente em cada hora difícil.

E a todos os que, amigos e colegas, de alguma forma contribuíram para a concretização deste trabalho.

SUMÁRIO

Estuda-se neste trabalho a existência de tecnologia Java para a representação de curvas e superfícies do tipo NURBS. Assim, foi efectuada uma pesquisa com o intuito de avaliar as diferentes opções disponíveis actualmente para este efeito.

Implementamos uma biblioteca Java que permite a representação deste tipo de curvas e superfícies, em particular, superfícies do tipo NURBS recortadas (*trimmed NURBS surfaces*) por um conjunto de curvas (*trimming curves*) definidas no espaço paramétrico da superfície.

Nesta biblioteca implementamos alguns algoritmos não existentes até à data em nenhuma biblioteca inteiramente baseada em tecnologia Java, nomeadamente, um algoritmo para efectuar a triangulação dos pontos de uma *trimmed NURBS surface,* bem como um algoritmo para calcular de forma dinâmica o valor de incremento do parâmetro de amostragem dos pontos da superfície. Este último algoritmo permite não só a aproximação da superfície por uma malha de polígonos de forma a que o erro não seja superior a um determinado valor de tolerância, mas também a aproximação de uma curva do tipo NURBS em segmentos de recta, obedecendo ao mesmo critério.

ABSTRACT

In this work, we explore the availability of Java technology for the rendering of NURBS curves and surfaces. We have done so by conducting a search, with the goal of accessing the nowadays existing application programming interfaces (APIs) implemented with Java technology, that are able of rendering these kind of curves and surfaces.

We implement an API, entirely based on Java technology, for the rendering and tessellation of NURBS curves and surfaces and, particularly, for the tessellation of trimmed NURBS surfaces.

Within this API we deliver some algorithms which were not yet implemented in a Java API. Namely, we implement an algorithm for tessellating a trimmed NURBS surface based on a triangle mesh, and we also implement an algorithm for approximating a NURBS surface by a triangle mesh within a specified error tolerance. This last algorithm can also be use to approximate a NURBS curve with piecewise line segments using the same error tolerance criterion.

ÍNDICE

Capítulo 1 – Introdução

1.1 Motivação

As curvas e superfícies paramétricas apresentam-se, actualmente, como um pilar importante em várias áreas da computação gráfica e, em particular, no domínio da modelação geométrica. Aplicações tão diversas como o desenho assistido por computador (*Computer Aided Design*, CAD), seja no ciclo de desenvolvimento de um novo produto, seja na modelação de uma molécula numa aplicação científica, o desenvolvimento de jogos de computador (*videogames*) ou a definição do caminho percorrido por uma câmara na animação de um simulador de voo são apenas alguns exemplos do âmbito da sua utilização.

De entre os vários tipos de curvas e superfícies paramétricas usadas na computação gráfica, as NURBS (*NonUniform Rational B-Splines*) têm um lugar de destaque pelo facto de ultrapassarem um conjunto de dificuldades colocadas por outros tipos de curvas e superfícies estudadas no final da década de 60 como, por exemplo, as curvas e superfícies de Bézier. As NURBS apresentam vantagens, entre as quais o facto de permitirem um controlo local da sua forma e de permitirem também uma representação exacta das Cónicas.

O principal factor de motivação do trabalho desenvolvido para esta dissertação consiste na inexistência, à data de início da mesma em Setembro de 2007, de uma biblioteca (ou *Aplication Programming Interface,* API) implementada

em tecnologia Java que permita a representação de superfícies do tipo NURBS recortadas (*trimmed NURBS surfaces*).

A opção pela tecnologia Java surge de forma natural quando se pensa nas actuais aplicações desta tecnologia a domínios como, a título de exemplo, os jogos de computador para telemóveis ou em aplicações de carácter didáctico para o ensino e a divulgação de ciências tais como a Física ou a Matemática. No sítio (*site)* da Universidade de Brown ([BROWwp] podemos ver um exemplo prático da afirmação anterior através de um conjunto bastante interessante de *applets Java* desenhados para o ensino/aprendizagem de importantes conceitos da área da Computação Gráfica. Em [NGCKwp] e [SUTHwp] é possível interagir com um conjunto de aplicações gráficas que permitem a aprendizagem de conceitos da Física e Astronomia, uma vez mais sob a forma de *applets*, e em [EDINwp]) temos outro exemplo para ciências como a Matemática e a Química.

A tecnologia Java nasceu com o objectivo inicial de ser uma tecnologia intrinsecamente ligada à *World Wide Web*, característica que mantém ainda tendo no entanto alargado o seu âmbito para outro tipo de aplicações, nomeadamente, aplicações de *desktop*.

Esta tecnologia tem ainda como uma das suas principais características o facto de ser totalmente independente de plataforma, o que constitui por si só um factor de enorme relevância, em particular em ambientes universitários onde tipicamente existem diversas plataformas tecnológicas e sistemas operativos.

Acresce que o Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa optou, recentemente, pela adopção desta tecnologia como denominador comum a todos os programas das disciplinas dos cursos do departamento.

1.2 Objectivos

Propomo-nos implementar uma biblioteca em tecnologia Java que permita a representação de curvas e superfícies do tipo NURBS com especial ênfase em superfícies recortadas. Esta implementação terá por base uma outra biblioteca já existente denominada jgeom [JGEOwp] e que, no contexto de superfícies do tipo NURBS, suporta o cálculo dos pontos de superfícies com apenas uma curva de recorte (*trimming curve*). É nosso propósito complementar a funcionalidade disponibilizada pela biblioteca com a possibilidade da representação de superfícies recortadas por um número arbitrário de curvas do tipo NURBS. Para além disso, pretendemos implementar um algoritmo de triangulação que permita aproximar a superfície por uma malha de polígonos.

A biblioteca jgeom disponibiliza, adicionalmente, *bindings* para *Java3D*, isto é, código (um conjunto de classes Java) que permite a utilização directa da biblioteca em aplicações implementadas em Java3D [J3DPwp]. No entanto, se tivermos em conta que a tecnologia OpenGL é das mais utilizadas, senão mesmo a mais utilizada, no mundo da computação gráfica seria bastante interessante implementar também *bindings* para JOGL [JOGLwp]. Efectuaríamos, assim, a ligação entre a biblioteca jgeom e a biblioteca JOGL, disponibilizando aos utilizadores desta última um conjunto de funcionalidades básicas já existentes noutras linguagens de programação como, por exemplo, o C, através de bibliotecas complementares ao OpenGL (neste caso a biblioteca GLU que implementa as funcionalidades relativas a NURBS) [OPGLwp].

1.3 Estrutura da Dissertação

No primeiro capítulo pretendemos esclarecer quais as motivações que nos levaram a empreender o trabalho aqui apresentado, os objectivos do mesmo e, de uma forma sumária, quais os resultados obtidos.

No segundo capítulo iremos abordar a teoria das curvas e superfícies paramétricas no contexto da sua aplicação à computação gráfica. Apresentaremos as definições de curvas e de superfícies de Bézier, B-Splines e NURBS e identificaremos alguns resultados teóricos importantes para a computação gráfica.

No terceiro capítulo apresentaremos, de forma detalhada, o trabalho de ampliação da biblioteca jgeom, bem como a sua ligação à biblioteca JOGL. Os vários algoritmos abordados para a avaliação dos pontos de curvas e de superfícies produzem resultados diferentes traduzindo-se, assim, de uma forma natural, em domínios de aplicação potencialmente distintos.

No quarto capítulo iremos expor os resultados de uma pesquisa sobre o suporte da representação de NURBS em tecnologia Java. Em particular procuraremos comparar as tecnologias JOGL e Java3D no que diz respeito ao suporte para a representação deste tipo de curvas e superfícies.

No quinto e último capítulo iremos estabelecer algumas considerações finais sobre o trabalho desenvolvido.

1.4 Contributos

O principal contributo que nos propomos apresentar consiste na implementação de uma biblioteca, em tecnologia Java, que permita a representação de superfícies do tipo NURBS recortadas, algo inexistente até à data.

Capítulo 2 – Curvas e Superfícies Paramétricas

Existem várias formas de representar matematicamente uma curva ou superfície. A representação paramétrica é, possivelmente, para o caso das curvas e superfícies que nos propomos estudar a aproximação mais frequentemente utilizada. No entanto, em função do tipo de aplicações, as curvas e superfícies podem também ser representadas de forma explícita ou implícita.

A representação explícita $y = f(x)$ é dependente do sistema de eixos coordenados, não permite a representação de aplicações com mais de uma ordenada para a mesma abcissa e apresenta problemas de cálculo numérico na representação de pontos com derivada infinita ou numa sua vizinhança [ROGE01].

A representação implícita $f(x, y) = 0$ também é dependente do sistema de eixos coordenados mas já permite a representação de aplicações com mais de uma ordenada para a mesma abcissa. É, no entanto, complexo nesta forma de representação o cálculo das derivadas nos pontos fronteira, facto que é muito relevante para as aplicações de modelação geométrica, como iremos ver já nas próximas secções. Não obstante esta desvantagem existem outras vantagens, o que potencia a utilização desta representação em diversas aplicações no domínio da computação gráfica.

2.1 Representação paramétrica

Uma função representada parametricamente tem a forma $f(t) = (x(t), y(t), z(t))$, sendo o parâmetro t a variável independente. Se considerarmos o caso particular de uma curva limitada no espaço temos

$C(t) = (x(t), y(t), z(t))$ com o parâmetro t limitado por duas constantes reais a e b, isto é, $a \leq t \leq b$. Frequentemente, os extremos do intervalo assumem os valores $a = 0$ e $b = 1$. Esta representação apresenta algumas vantagens, permitindo nomeadamente representar aplicações com mais de uma ordenada para a mesma abcissa. Adicionalmente é comum, nas aplicações deste tipo de curvas e superfícies ao domínio da computação gráfica, existir uma limitação física à própria curva ou superfície, o que, na representação paramétrica, se concretiza de uma forma natural através dos limites impostos ao parâmetro t.

Note-se, no entanto, que a representação paramétrica também tem os seus inconvenientes, nomeadamente o facto de algumas operações comuns como, por exemplo, o cálculo de distâncias serem mais difíceis de efectuar com a mesma.

Vejamos um exemplo de uma representação paramétrica de uma curva. Consideremos a curva definida, parametricamente, no espaço bidimensional por:

$$\begin{cases} x(t) = r\cos(t) \\ y(t) = r\sin(t) \end{cases} \qquad 0 \leq t \leq 2\pi \qquad (2.1)$$

Esta curva representa uma circunferência de raio r centrada na origem, como mostra a figura 2.1. O parâmetro t representa o ângulo indicado na figura e, se o fizermos variar entre os valores 0 e 2π, conseguimos representar toda a circunferência. No entanto facilmente poderíamos ter obtido uma semicircunferência variando o parâmetro apenas, por exemplo, entre 0 e π.

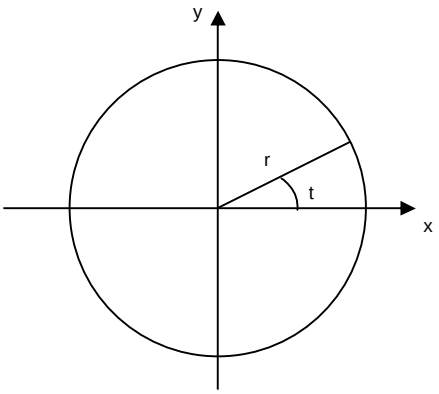

Figura 2.1 – Representação paramétrica de uma circunferência

Por vezes, é esclarecedor, como propõe L. Pigel em [PIEG97], pensar no parâmetro como uma variável de tempo. Neste caso, a curva poderia representar a trajectória de uma partícula em função do tempo.

Note-se, também, que podem existir várias representações paramétricas para uma mesma curva ou superfície. Por exemplo, no caso de uma circunferência, outra representação paramétrica possível pode ser obtida substituindo o parâmetro t pelo parâmetro u definido por $u = tg\left(t/2\right)$ nas expressões (2.1) e utilizando a relação trigonométrica $tg\left(\frac{t}{2}\right) = \pm\sqrt{\frac{1-\cos(t)}{1+\cos(t)}}$. Obtém-se, assim, tal como exposto em [PIEG97], a seguinte representação paramétrica da circunferência:

$$\begin{cases} x(u) = r\dfrac{1-u^2}{1+u^2} \\ y(u) = r\dfrac{2u}{1+u^2} \end{cases} \qquad 0 \le u \le 1. \qquad (2.2)$$

Vejamos mais um caso. Se ao exemplo bidimensional de uma circunferência adicionarmos uma terceira dimensão definida da seguinte forma:

$$\begin{cases} x(t) = r\cos(t) \\ y(t) = r\sin(t) \\ z(t) = t \end{cases} \qquad 0 \le t \le 2\pi\,, \qquad (2.3)$$

iremos obter uma curva em forma de hélice como a representada na figura 2.2.

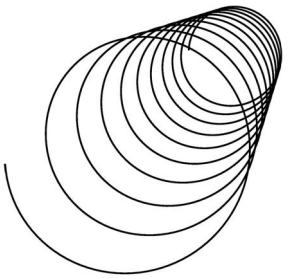

Figura 2.2 – Curva em forma de hélice definida por (2.3) em projecção perspectiva

A generalização de curva para superfície paramétrica ocorre de forma directa através da extensão do espaço paramétrico a duas dimensões. Assim, uma superfície representada na forma paramétrica é definida por $S(u,v) = (x(u,v), y(u,v), z(u,v))$ sendo os parâmetros u e v as variáveis independentes (figura 2.3).

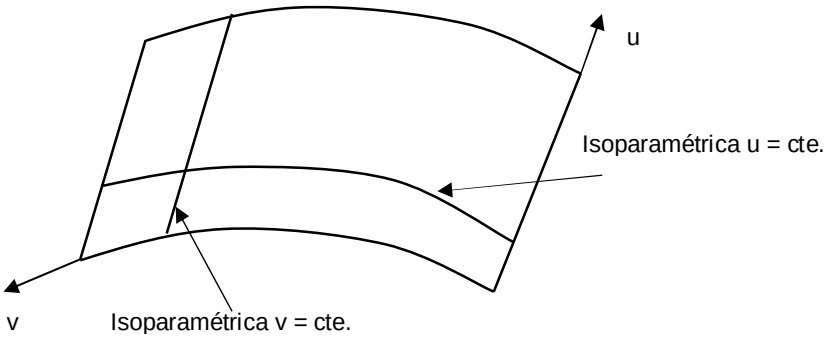

Figura 2.3 – Representação paramétrica de uma superfície

Como exemplo de uma superfície, consideremos uma superfície esférica definida por:

$$\begin{cases} x(u,v) = r\sin(u)\cos(v) \\ y(u,v) = r\sin(u)\sin(v) \\ z(u,v) = r\cos(u) \end{cases} \qquad 0 \le u \le \pi \text{ e } 0 \le v \le 2\pi . \qquad (2.4)$$

Neste caso as curvas isoparamétricas correspondem precisamente aos paralelos[1] (curvas de igual latitude) e aos meridianos (curvas de igual longitude) da superfície esférica.

Com estes exemplos pretendemos introduzir o leitor no tema das representações paramétricas de curvas e superfícies sublinhando, simultaneamente, o facto de que podem existir várias representações paramétricas diferentes para uma mesma curva ou superfície.

2.2 Continuidade e Superfícies por Secções ("Piecewise Surfaces")

É frequente em aplicações de modelação geométrica e desenho assistido por computador a necessidade de representar superfícies que não têm uma definição analítica. Estas superfícies só podem ser representadas por um conjunto de várias superfícies menores (*patches* ou secções) que são unidas umas às outras através das suas fronteiras (curvas limítrofes). Alguns exemplos típicos são as superfícies de fuselagens de automóveis, de aviões ou ainda os cascos de navios. Para algumas destas aplicações é muito importante que se verifiquem determinadas restrições na fronteira entre um determinado *patch* e um *patch* adjacente. Estas restrições consistem, maioritariamente, em restrições aos valores das derivadas paramétricas parciais nos pontos fronteira de cada *patch*.

[1] Utilizando uma analogia com a geografia

2.2.1 Continuidade Geométrica

Consideremos dois segmentos de curva, Q1 e Q2, unidos no ponto P2 como representado na figura 2.4.

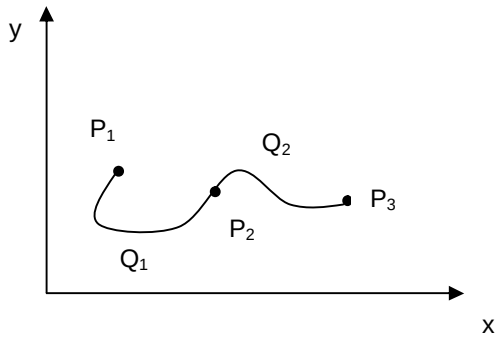

Figura 2.4 – Continuidade geométrica e continuidade paramétrica

Dizemos que existe continuidade geométrica de ordem zero – G^0 – na fronteira entre os dois segmentos de curva se e só se o ponto final do segmento Q1 for o mesmo que o ponto inicial do segmento Q2. Se o vector tangente à curva Q1 no seu ponto final tiver a mesma direcção e sentido que o vector tangente à curva Q2 no seu ponto inicial dizemos que existe continuidade geométrica de primeira ordem – G^1. Se acontecer o mesmo para a direcção e sentido do vector segunda derivada dizemos que existe continuidade geométrica de segunda ordem – G^2, etc. Em suma, existe continuidade geométrica de ordem n quando a direcção e sentido do vector derivada de ordem n, $\frac{d^n C(t)}{dt^n}$, são os mesmos quando avaliados no ponto final da primeira curva e no ponto inicial da segunda curva. Repare-se que nada se afirma sobre o módulo deste vector, i.e., a definição não obriga, por exemplo, a que estes vectores tenham o mesmo módulo. O mesmo já não é válido para o segundo tipo de continuidade que iremos abordar na próxima secção.

2.2.2 Continuidade Paramétrica

Da mesma forma que de duas curvas unidas por um mesmo ponto fronteira se diz exibirem continuidade geométrica de ordem 0 – G^0, também se diz que exibem continuidade paramétrica de ordem 0 – C^0. Resulta de imediato que C^0 implica G^0 e vice-versa. No entanto, diz-se que uma curva exibe continuidade paramétrica de ordem 1 – C^1 – se os vectores tangentes no ponto fronteira para ambas as curvas tiverem igual direcção, sentido, **e módulo**, i.e., a derivada paramétrica $\frac{dC(t)}{dt}$ avaliada para o ponto final da curva Q1 tem de ser igual em direcção, sentido e módulo à mesma derivada avaliada para o ponto inicial da curva Q2. A continuidade paramétrica de ordens superiores define-se da mesma forma, ou seja, através dos constrangimentos de igual direcção, sentido e módulo para as derivadas paramétricas de ordens superiores no ponto fronteira. Note-se que nestas definições estamos, apenas, a restringir as derivadas nos pontos fronteira, uma vez que os vários segmentos de curva são contínuos no seu interior.

Considerando mais uma vez o exemplo em que a curva $C(t)$ representaria a trajectória de uma partícula material, ter-se-ia então que $\frac{dC(t)}{dt}$ representaria a velocidade da partícula e $\frac{d^2C(t)}{dt^2}$ a sua aceleração. Se pensarmos que esta partícula poderá ser, por exemplo, a posição da câmara num simulador de voo, facilmente se compreende a importância de manter a continuidade paramétrica dos pontos fronteira ao longo dos vários segmentos de curva, tanto de primeira como de segunda ordem. Se existisse apenas, por exemplo, G^1 e não existisse C^1 iríamos observar um salto brusco no módulo da velocidade no ponto fronteira apesar de a

direcção e sentido da mesma se manter. Adicionalmente, é frequente encontrarmos aplicações que têm como constrangimento a existência de uma mudança suave de um *patch* da superfície para um *patch* adjacente, por exemplo aplicações de CAD que necessitem de uma iluminação sem saltos abruptos ao longo de toda a superfície. Tal condição implica a existência de continuidade paramétrica não inferior à segunda ordem (C^2).

2.3 Curvas e Superfícies de Bézier

Na final da década de 1960 Pierre Bézier, Eng. Mecânico de formação e funcionário do fabricante de automóveis Renault, desenvolveu uma nova forma de desenhar a carroçaria de automóveis – um sistema de desenho assistido por computador baptizado com o nome de UNISURF. Esta necessidade surgiu porque o processo, tal como existia à data, envolvia a intervenção de um considerável número de funcionários com diferentes funções e, em cada etapa do mesmo, eram introduzidas ligeiras modificações ao desenho inicialmente apresentado pelos desenhadores. Para além disso todo o processo era baseado em suporte físico de papel e em moldes físicos da carroçaria. Bézier e os seus colaboradores acreditavam que poderiam tornar o processo mais eficiente se a transmissão da informação, ao longo do processo, se baseasse em dados numéricos e não em desenhos em suporte de papel [BEZI98]. Recorrendo a considerações geométricas Bézier derivou as funções base (polinómios de Bernstein) para a aproximação polimonial de um tipo de curvas e superfícies que viriam, mais tarde, a ser designadas por curvas e superfícies de Bézier, as quais utilizou no seu processo de desenho industrial.

2.3.1 Pré-requisitos de um sistema de modelação

Na implementação de um sistema de modelação geométrica é recomendável que as funções de base respeitem um conjunto de propriedades que se enumeram seguidamente [PIEG97]:

- permitam a representação de todas as curvas e superfícies necessárias aos utilizadores do sistema;
- sejam utilizáveis por um computador de forma fácil, eficiente e precisa, de tal modo que o cálculo dos pontos e das derivadas da curva seja eficiente e numericamente estável, no que respeita a erros resultantes de cálculos com números de vírgula-flutuante;
- utilizem a memória da máquina de forma eficiente;
- sejam simples e bem estudadas do ponto de vista matemático.

As funções polimoniais verificam os últimos três pontos, no entanto existe uma grande quantidade de curvas e superfícies que não podem ser modeladas através da interpolação ou aproximação por funções polimoniais. Estas curvas e superfícies podem, apenas, ser representadas de forma aproximada pelas mesmas. As curvas e superfícies de Bézier, apesar de polimoniais, foram aplicadas com sucesso a nível industrial, e historicamente são percursoras de outros tipos de curvas e superfícies que iremos estudar seguidamente, como as superfícies do tipo B-Spline e as NURBS. Bézier utilizou com sucesso este tipo de curvas, não só para a modelação de carroçarias de automóveis mas também para o desenho de asas de aviões, de cascos de navios e até para objectos comuns como, por exemplo, um assento de uma carruagem de comboio para os caminhos de ferro franceses [ROGE01].

2.3.2 Curvas de Bézier

Uma curva de Bézier de grau *n* define-se da seguinte forma:

$$C(u)=\sum_{i=0}^{n} B_{i,n}(u)P_i \qquad 0\le u\le 1 \qquad (2.5)$$

em que as funções de base $B_{i,n}(u)$ são os polinómios de Bernstein definidos por

$$B_{i,n}(u)=\frac{n!}{i!(n-i)!}u^i(1-u)^{n-i} \quad (2.6)$$

e os P_i representam os pontos de controlo que definem a forma do polígono de controlo. A título de exemplo considere-se uma curva de Bézier de grau 3 (*n=3*) definida por

$$C(u)=(1-u)^3P_0+3u(1-u)^2P_1+3u^2(1-u)P_2+u^3P_3. \qquad (2.7)$$

Note-se que os pontos de controlo são pontos tridimensionais dados por $P_i=(x_i,y_i,z_i)$ ou por $P_i=(x_i,y_i,z_i,w_i)$ no caso da representação por coordenadas homogéneas. Seguem-se alguns exemplos de curvas de Bézier de grau 3, logo com 4 pontos de controlo (figuras 2.5 a 2.7).

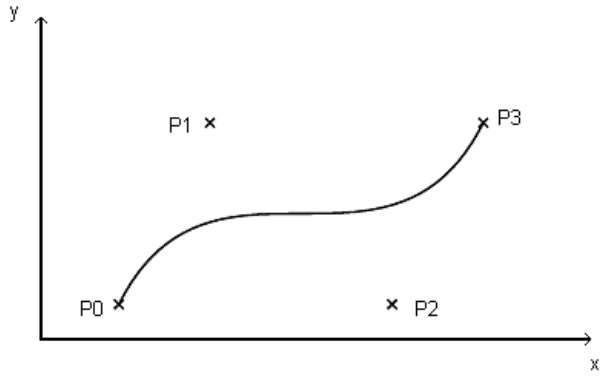

Figura 2.5 – Curva de Bézier de grau 3 (exemplo 1)

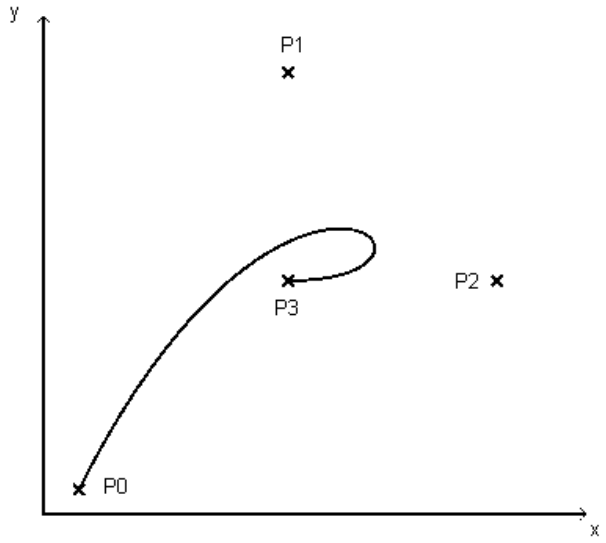

Figura 2.6 – Curva de Bézier de grau 3 (exemplo 2)

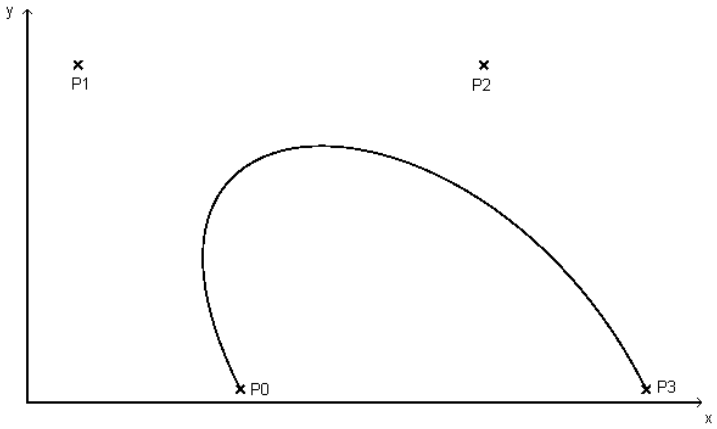

Figura 2.7 – Curva de Bézier de grau 3 (exemplo 3)

Podemos identificar nestes exemplos (sem perda de generalidade) as seguintes propriedades das curvas de Bézier:

- o polígono de controlo (polígono definido pelos vários pontos de controlo) aproxima a forma da curva;
- o primeiro ponto de controlo é interpolado pela curva paramétrica para o valor do parâmetro $u = 0$, i.e., $P_0 = C(0)$; similarmente, o último ponto de controlo é interpolado pela curva para o valor do parâmetro $u = 1$, i.e, $P_n = C(1)$;

- os vectores tangentes aos pontos P_0 e P_n têm a direcção dos vectores definidos por $P_1 - P_0$ e $P_n - P_{n-1}$, respectivamente;
- a curva é limitada pelo seu polígono de controlo (*convex hull property*);
- não existe nenhuma linha recta que intersecte a curva mais vezes do que intersecta o seu polígono de controlo (*variation diminishing property*);
- as curvas são invariantes perante as operações de transformação: rotação, translação, mudança de escala e *shear;*

Em particular esta última propriedade significa que aplicar uma transformação de rotação, translação, mudança de escala ou *shear* a uma curva ou superfície de Bézier implica, apenas, aplicar a mesma aos seus pontos de controlo, sendo possível, posteriormente, recalcular os pontos da superfície através da sua própria definição sem que seja necessário aplicar a transformação a todos os pontos da curva.

Estas propriedades permitiram a este tipo de curvas e superfícies todas as aplicações anteriormente mencionadas. No entanto existe uma operação fundamental que não é invariante para as curvas e superfícies de Bézier, a saber: a transformação de projecção geométrica. Adicionalmente, estas curvas não permitem um controlo local da curva através dos seus pontos de controlo, i.e., ao ser deslocado apenas um ponto de controlo toda a curva irá ser afectada e não apenas os pontos da curva na vizinhança desse mesmo ponto de controlo.

2.3.3 Superfícies de Bézier

Uma superfície de Bézier define-se através de uma rede bidimensional de pontos de controlo e do produto dos polinómios de Berstein (2.6) pela expressão seguinte:

$$S(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)P_{i,j},\ 0\le u,v\le 1. \qquad (2.8)$$

Todas as propriedades indicadas acima para as curvas de Bézier são extensíveis ao caso das superfícies, com excepção da propriedade denominada *variation diminishing property*, que até ao momento ainda não foi provada para superfícies.

Na figura 2.8 apresenta-se uma representação esquemática da rede bidimensional de pontos de controlo. Na figura 2.9 apresenta-se um exemplo de uma superfície de Bézier com os pontos de controlo representados por uma cruz. Note-se que os quatro pontos nos cantos do polígono de controlo são interpolados pela superfície e note-se, também, que o polígono de controlo aproxima a mesma.

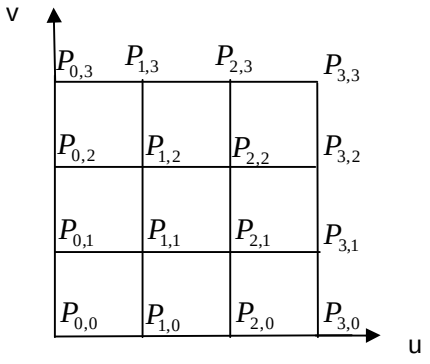

Figura 2.8 – Rede bidimensional de pontos de controlo de uma superfície

A superfície representada na figura 2.9 é bi-cúbica (grau 3 em ambas as direcções paramétricas) e tem os seguintes pontos de controlo:

$$P_{0,3} = (0,0,0), P_{1,3} = (2,0,0), P_{2,3} = (4,0,0), P_{3,3} = (6,0,0)$$

$$P_{0,2} = (0,0,2), P_{1,2} = (2,6,2), P_{2,2} = (4,6,2), P_{3,2} = (6,0,2)$$

$$P_{0,1} = (0,0,4), P_{1,1} = (2,6,4), P_{2,1} = (4,6,4), P_{3,1} = (6,0,4)$$

$$P_{0,0} = (0,0,6), P_{1,0} = (2,0,6), P_{2,0} = (4,0,6), P_{3,0} = (6,0,6)$$

Devido às restrições indicadas anteriormente, a saber o facto de as curvas e superfícies de Bézier não serem invariantes perante a transformação de projecção e o facto de não permitirem um controlo local da superfície, surgiu a necessidade de procurar outro tipo de curvas e superfícies que não apresentam estas restrições. Surgiram assim as curvas e superfícies do tipo B-Spline, as quais têm como caso particular as curvas e superfícies de Bézier.

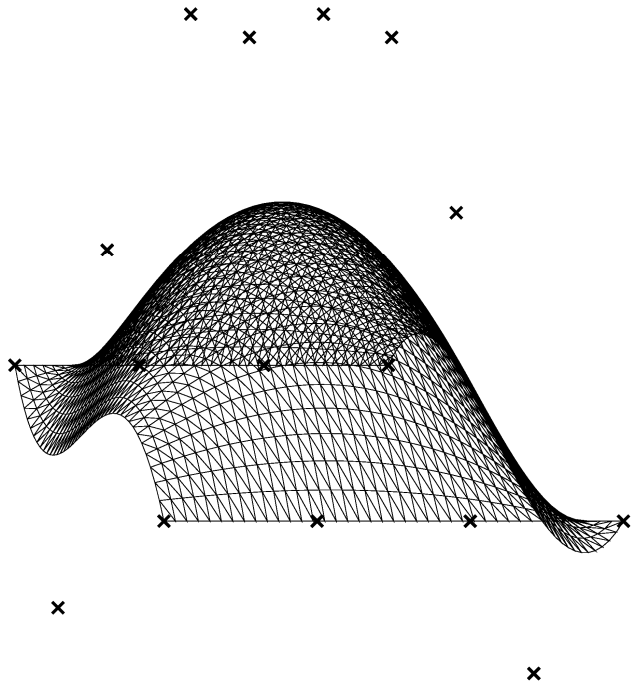

Figura 2.9 – Superfície de Bézier gerada com a API jgeom

2.4 Curvas e Superfícies B-Spline

Uma curva do tipo B-Spline de grau *p* é definida por

$$C(u)=\sum_{i=0}^{n} N_{i,p}(u)P_i \qquad a \le u \le b \tag{2.9}$$

em que os P_i são os pontos de controlo que definem o polígono de controlo e as funções B-Spline $N_{i,p}(u)$ de grau *p* são definidas tendo por base o vector de nós (*knot vector*)

$$U = \{u_0, u_1, u_2, \ldots, u_{(n+1)+(p+1)}\} \tag{2.10}$$

sendo dadas pela expressão recorrente (2.11)

$$N_{i,0}(u)=\begin{cases}1 & se \quad u_i \le u < u_{i+1}\\ 0 & caso \quad contrário\end{cases}$$

$$N_{i,p}(u)=\frac{u-u_i}{u_{i+p}-u_i}N_{i,p-1}(u)+\frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u). \tag{2.11}$$

É comum na representação de curvas e superfícies do tipo B-Spline utilizarem-se dois tipos de vectores de nós: *open* e *periodic*. Para abordarmos a sua definição temos primeiro que definir o conceito de multiplicidade de um nó. A multiplicidade de um nó define-se como o número de repetições de um mesmo valor em nós consecutivos. Por exemplo, para o vector de nós {0, 1, 1, 1, 2, 3, 3, 3, 3, 4}, 0 tem multiplicidade 1, 1 tem multiplicidade 3, 2 tem multiplicidade 1, 3 tem multiplicidade 4 e 4 tem multiplicidade 1. Voltando aos dois tipos de vectores de nós, diz-se que um

vector é do tipo *open* se os primeiros e os últimos $p+1$ nós têm uma multiplicidade igual ao grau da curva + 1, precisamente $p+1$. Para uma curva de grau 3 um vector de nós do tipo *open* teria que ter obrigatoriamente os primeiros e os últimos nós com multiplicidade 4 (grau+1 = 3+1), por exemplo {0, 0, 0, 0, 1/2, 1, 1, 1, 1}. A multiplicidade dos primeiros nós é 4 (0, 0, 0, 0), bem como a dos últimos nós que também é 4 (1, 1, 1, 1). Para a mesma superfície, um vector do tipo *periodic* poderia ser por exemplo {0, 1/2, 1, 3/2, 2, 5/2, 3, 7/2, 4}.

Os vectores de nós são também classificados, segundo outra taxionomia, em uniformes (intervalo constante entre cada nó), por exemplo {0, 1, 2, 3, 4, 5, 6, 7}, ou não-uniformes (em que o intervalo entre cada nó deixa de estar restrito a ser uma constante podendo ser qualquer valor desde que o valor do nó $i+1$ seja maior ou igual ao valor do nó i), por exemplo {0, 1, 3, 6, 7, 7/2}.

Passando para o domínio das superfícies, uma superfície não racional do tipo B-Spline $S(u,v)$ é definida com base numa rede bidimensional de pontos de controlo $P_{i,j}$ (*bidirectional net of control points*), dois vectores de nós (*knot vectors*) $U=\{u_0,u_1,u_2,\ldots,u_{(n+1)+(p+1)}\}$ e $V=\{v_0,v_1,v_2,\ldots,v_{(m+1)+(q+1)}\}$ e os produtos das funções B-Spline $N_{i,p}(u)$ definidas pela expressão (2.11), ou seja

$$S(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)P_{i,j}\,. \qquad (2.12)$$

Nesta definição temos $n+1$ pontos de controlo na direcção paramétrica *u*, $m+1$ pontos de controlo na direcção paramétrica *v*, funções B-Spline de grau *p* na direcção paramétrica *u* e de grau *q* na direcção paramétrica *v*.

Como exemplo de uma superfície do tipo B-Spline considere-se a superfície apresentada na figura 2.10, cujos pontos de controlo são dados por

$$P_{0,4} = (0,0,0), P_{1,4} = (2,0,0), P_{2,4} = (4,0,0), P_{3,4} = (6,0,0), P_{4,4} = (8,0,0)$$

$$P_{0,3} = (0,0,2), P_{1,3} = (2,1,2), P_{2,3} = (4,2,2), P_{3,3} = (6,1,2), P_{4,3} = (8,0,2)$$

$$P_{0,2} = (0,0,4), P_{1,2} = (2,2,4), P_{2,2} = (4,4,2), P_{3,2} = (6,2,4), P_{4,2} = (8,0,4)$$

$$P_{0,1} = (0,0,6), P_{1,1} = (2,1,6), P_{2,1} = (4,2,6), P_{3,1} = (6,1,6), P_{4,1} = (8,0,6)$$

$$P_{0,0} = (0,0,8), P_{1,0} = (2,0,8), P_{2,0} = (4,0,8), P_{3,0} = (6,0,8), P_{4,0} = (8,0,8)$$

e os vectores de nós dados por

$$U = \{0,0,0,0, 1/2, 1,1,1,1\}$$

$$V = \{0,0,0,0, 1/2, 1,1,1,1\}.$$

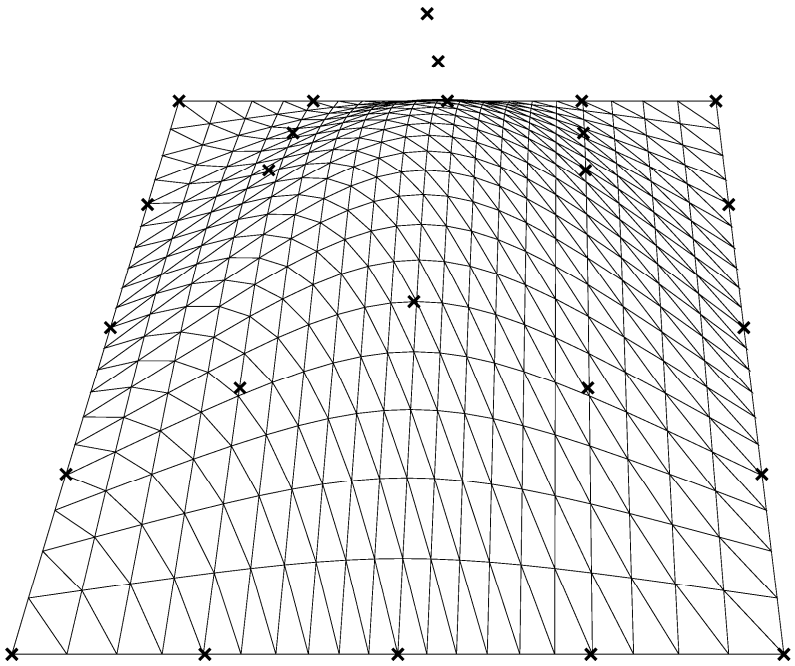

Figura 2.10 – Superfície do tipo B-Spline gerada com a API jgeom

Como demonstrado em [ROGE01], as curvas e superfícies do tipo B-Spline gozam das propriedades indicadas anteriormente para as curvas e superfícies de

Bézier, mas também apresentam a seguinte propriedade adicional de extrema importância:

- Controlo local – ao movermos o ponto de controlo P_i de uma curva do tipo B-Spline só estamos a modificar o troço da curva correspondente ao intervalo $\lfloor u_i, u_{i+p+1} \lfloor$, em que *p* é o grau da curva.

Esta propriedade é de extrema importância em aplicações de desenho assistido por computador porque proporciona ao utilizador um controlo local da curva que lhe permite um manuseamento mais detalhado da forma da mesma.

A mesma propriedade é válida para superfícies do tipo B-Spline, sendo agora enunciadas da seguinte forma:

- Controlo local – ao movermos o ponto de controlo $P_{i,j}$ de uma superfície do tipo B-Spline só estamos a modificar a secção da superfície correspondente ao rectângulo $\lfloor u_i, u_{i+p+1} \lfloor \mathrm{x} \lfloor v_i, v_{i+q+1} \lfloor$, em que *p* é o grau da superfície na direcção paramétrica *u* e *q* é o grau na direcção paramétrica *v*.

Este tipo de curvas e superfícies, não obstante verificarem as importantes propriedades mencionadas de controlo local e de serem invariantes perante as transformações afins (translações, rotações, mudanças de escala e *shears*), não apresentam ainda uma outra propriedade muito importante: não são invariantes perante a transformação de projecção. Esta é uma propriedade presente nas curvas e superfícies do tipo NURBS (*Non-Uniform Rational B-Splines*), que iremos abordar na próxima secção.

2.5 NURBS

Uma curva do tipo NURBS de grau *p* é definida por

$$C(u)=\frac{\sum_{i=0}^{n} N_{i,p}(u)w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u)w_i} \qquad a \le u \le b \tag{2.13}$$

em que os P_i são os pontos de controlo que definem o polígono de controlo, os w_i são os designados *weights* ou pesos de cada parcela (usualmente utilizam-se apenas $w_i > 0$) e as funções B-Spline $N_{i,p}(u)$, definidas pela expressão (2.11), de grau *p* são definidas tendo por base o vector de nós (*knot vector*)

$$U = \{u_0, u_1, u_2, \ldots, u_{(n+1)+(p+1)}\} \tag{2.14}.$$

Podemos definir as funções (*rational basis functions*)

$$R_{i,p}(u)=\frac{N_{i,p}(u)w_i}{\sum_{j=0}^{n} N_{j,p}(u)w_j} \tag{2.15}$$

obtendo assim a expressão

$$C(u)=\sum_{i=0}^{n} R_{i,p}(u)P_i \ . \tag{2.16}$$

Antes de prosseguirmos o estudo destas curvas e superfícies gostaríamos de salientar duas notas importantes.

- Em primeiro lugar notemos que uma curva/superfície do tipo B-Spline é um caso particular de uma curva/superfície do tipo NURBS (para chegar a esta conclusão basta fazer todos os $w_i = 1$ e utilizar a propriedade das funções $N_{i,p}$ que conclui que $\sum_{i=0}^{n} N_{i,p}(u)=1$ [ROGE01]);
- Em segundo lugar notemos, também, que a utilização de coordenadas homogéneas [PIEG97] nos permite trabalhar com curvas e superfícies do tipo NURBS como se estivéssemos a trabalhar com curvas e superfícies do tipo

B-Spline, em que a quarta coordenada de cada ponto de controlo corresponde precisamente ao seu peso (w_i). Temos, assim, uma ferramenta matemática que nos permite simplificar os processos de cálculo, à partida mais complexos, pela introdução de um quociente na definição das funções. É suficiente para tal trabalhar no espaço homogéneo, convertendo no final do processo de cálculo as coordenadas homogéneas para o espaço tridimensional euclidiano.

Se analisarmos o efeito da introdução dos pesos associados a cada ponto de controlo (w_i) iremos verificar que quanto maior o seu valor absoluto mais a curva ou superfície se irá aproximar desse ponto de controlo. Assim temos, para além do controlo local já associado a curvas e superfícies do tipo B-Spline, um mecanismo para diferenciar (positivamente ou negativamente) a influência de cada ponto de controlo no cálculo dos pontos da curva ou superfície.

Uma superfície do tipo NURBS $S(u,v)$ é definida com base numa rede bidimensional de pontos de controlo $P_{i,j}$ (*bidirectional net of control points*) cada um afectado pelo respectivo peso $w_{i,j}$, dois vectores de nós (*knot vectors*) $U=\{u_0,u_1,u_2,\ldots,u_{(n+1)+(p+1)}\}$ e $V=\{v_0,v_1,v_2,\ldots,v_{(m+1)+(q+1)}\}$ e os produtos das funções *rational basis functions* $R_{i,j}(u,v)$ definidas por,

$$R_{i,j}(u,v)=\frac{N_{i,p}(u)N_{j,q}(v)w_{i,j}}{\sum_{k=0}^{n}\sum_{l=0}^{m}N_{k,p}(u)N_{l,q}(v)w_{k,l}} \qquad (2.17)$$

ou seja

$$S(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}R_{i,j}(u,v)P_{i,j}\,. \qquad (2.18)$$

Como exemplo, considere-se a mesma superfície da secção anterior (figura 2.10). No entanto considerem-se, agora, as duas situações que se seguem em que os valores dos pesos são dados por:

- $w_{i,j} = 1$ com excepção do peso correspondente ao ponto central da rede de pontos de controlo cujo valor será 20, i.e., $w_{2,2} = 20$ (figura 2.11);
- $w_{i,j} = 1$ com excepção de $w_{1,2}$ que terá o valor 20 (figura 2.12).

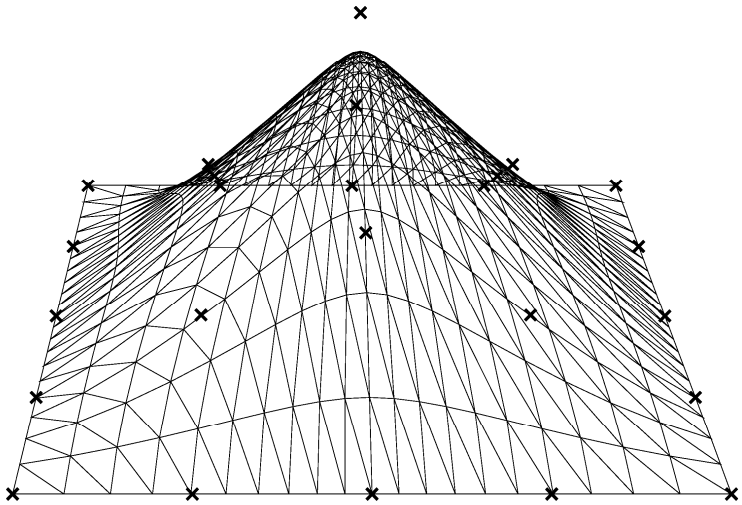

Figura 2.11 – Superfície do tipo NURBS gerada com a API jgeom, $w_{2,2} = 20$

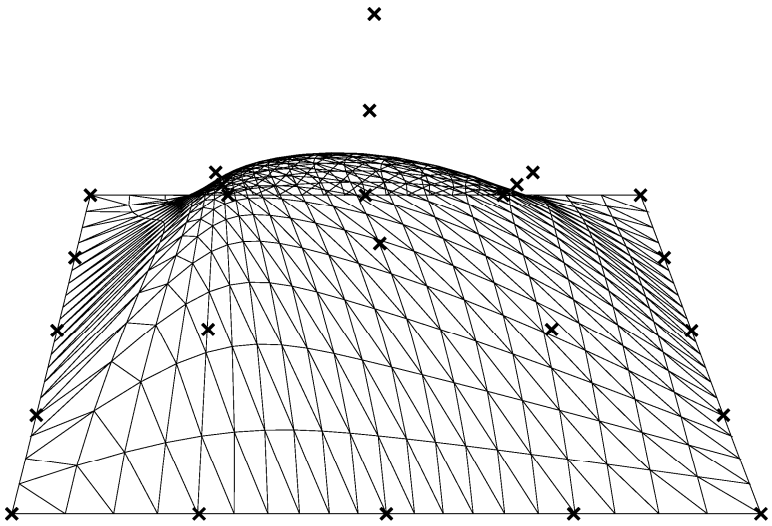

Figura 2.12 – Superfície do tipo NURBS gerada com a API jgeom, $w_{1,2} = 20$

Ao compararmos estas figuras com a figura 2.10 é claramente visível a influência dos pesos associados a cada ponto de controlo como sendo um factor que permite uma maior influência do ponto de controlo em causa, seja para que a superfície se aproxime mais do mesmo ($w_{i,j} > 1$) ou para que se afaste ($w_{i,j} < 1$). É possível inclusivamente a utilização de pesos com valores negativos, o que proporciona um género de "depressão" ou "vale" na secção da superfície influenciada por esse ponto de controlo.

Do ponto de vista matemático a transformação de um ponto em coordenadas homogéneas para o espaço euclidiano tridimensional consiste na divisão pelo peso w, i.e., se tivermos o ponto $P(x, y, z)$ no espaço euclidiano tridimensional, este ponto será representado como $P_w(wx, wy, wz, w)$ no espaço tetradimensional das coordenadas homogéneas. Para fazer a conversão de P_w para P bastará, portanto, dividir as coordenadas de P_w por w e ignorar a quarta coordenada (de valor 1).

$$P_w(wx, wy, wz, w) \mapsto P\left(\frac{wx}{w}, \frac{wy}{w}, \frac{wz}{w}, 1\right) \mapsto P(x, y, z) \tag{2.19}$$

Utilizando esta transformação de coordenadas, as expressões matemáticas relativas à transformação perspectiva [FOLE96] e a própria definição de NURBS é possível demonstrar que as curvas e superfícies deste tipo são invariantes relativamente à transformação perspectiva. Esta propriedade é de extrema importância, dado que permite perspectivar uma curva ou superfície do tipo NURBS através da aplicação da transformação perspectiva apenas aos pontos de controlo, e não a todos os pontos da curva ou superfície.

Outra propriedade muito importante das curvas e superfícies do tipo NURBS é a capacidade da representação exacta das cónicas enquanto que as curvas e superfícies do tipo B-Spline permitiam apenas uma representação aproximada das mesmas.

2.5.1 Superfícies recortadas do tipo NURBS (*Trimmed NURBS Surfaces*)

Nas aplicações do mundo real revela-se necessário, frequentemente, representar superfícies recortadas por um conjunto de curvas (*trimming curves*). Estas curvas podem ser de diversos tipos. No entanto, de uma forma geral, quando estamos a trabalhar com superfícies do tipo NURBS é desejável que as curvas sejam elas próprias curvas do tipo NURBS.

Assim, para termos uma *trimmed NURBS surface* necessitamos de ter uma superfície do tipo NURBS tal como definida na secção anterior e, também, de um conjunto de N curvas orientadas que estejam totalmente contidas no espaço paramétrico que define a superfície [PIEG95], tal que:

$$C_k(t) = (u_k(t), v_k(t)) = \sum_{i=0}^{n} N_{i,p}(t) P_i^k \quad k = 1,2,...,N\,. \tag{2.20}$$

As curvas deverão estar correctamente orientadas no sentido em que deverão formar $M \leq N$ curvas fechadas (*loops*), que são as fronteiras que definem o recorte das superfícies.

Apresentamos na figura 2.13, a título de exemplo, um modelo de uma moto que contém duas superfícies do tipo NURBS recortadas num total de 49 superfícies que definem o modelo.

Nas figuras 2.14 e 2.15 visualizam-se, claramente, as duas superfícies que foram recortadas com o objectivo de permitir a rotação da roda traseira ([CAS1wp]).

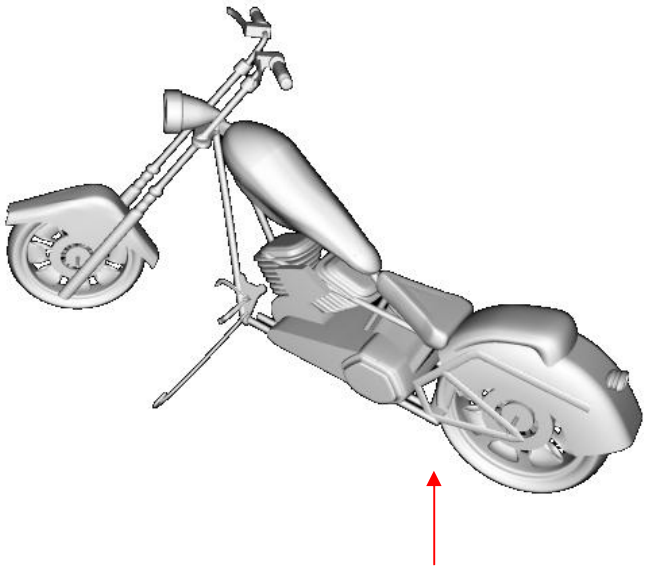

Figura 2.13 – Modelo de uma moto Harley definido através de superfícies do tipo NURBS recortadas [CASCwp].
A seta indica as superfícies recortadas.

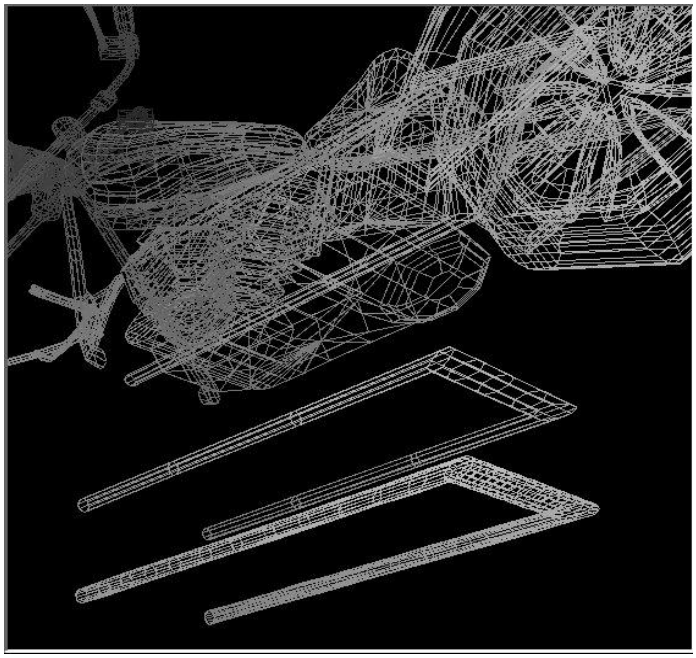

Figura 2.14 – Superfícies a recortar (wireframe)[2]

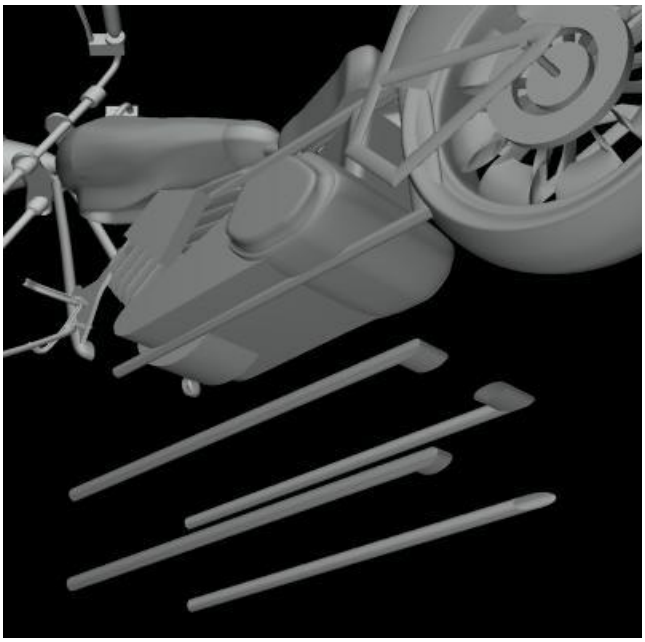

Figura 2.15 – Superfícies recortada

[2] As figuras 2.14 e 2.15 foram cortesia do Professor Giulio Casciola, Università di Bologna

Na figura 2.16, apresentamos a mesma superfície da figura 2.11 mas com duas *trimming curves* associadas.

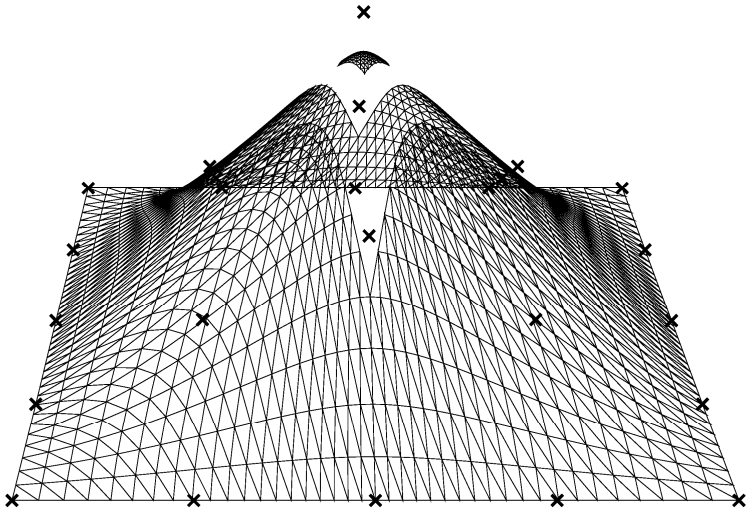

Figure 2.16 - *Trimmed NURBS surface* gerada com a API jgeom (incluindo as novas funcionalidades desenvolvidas para esta dissertação), $w_{2,2} = 20$

Capítulo 3 – Trabalho de análise e de implementação

À data de início deste trabalho, em Setembro de 2007, um programador que desejasse desenvolver uma aplicação em tecnologia Java que utilizasse superfícies do tipo NURBS recortadas, i.e., com um conjunto de *trimming curves* associadas à superfície que delimitassem zonas de recorte na mesma, não poderia fazê-lo utilizando uma API pública implementada em Java. Existiam, então, apenas duas bibliotecas implementadas em Java que poderiam ser utilizadas para a representação de superfícies do tipo NURBS. No entanto nenhuma destas bibliotecas suportava um número arbitrário de *trimming curves*.

O projecto jgeom [JGEOwp] permitia o cálculo dos pontos de uma superfície do tipo NURBS, mas com as seguintes restrições:

- suportava apenas uma única *trimming curve*;
- permitia apenas a utilização de *open knot vectors*, i.e., vectores de nós em que o primeiro e o último nó têm uma multiplicidade igual à ordem da superfície;
- não implementava nenhum algoritmo de triangulação dos pontos calculados para a superfície de forma a aproximar a mesma por uma malha de polígonos.

Adicionalmente, a biblioteca implementada no âmbito do projecto mencionado tinha apenas implementados os *bindings* para Java3D, não existindo ainda *bindings* para JOGL.

Por outro lado, no decorrer do ano de 2007, Tomas Hrasky, um estudante da Universidade de Hradec Králové (República Checa), migrou parte do código C++ da biblioteca GLU NURBS do OpenGL para Java, sendo este código integrado na biblioteca JOGL em 9 de Outubro do mesmo ano, na release JSR-231 1.1.1 RC4 [JGFRwp]. No entanto, esta implementação tinha a restrição muito importante de incluir apenas superfícies sem *trimming curves*. Não obstante tinha também a vantagem de a assinatura dos seu métodos em Java serem já semelhantes às funções C++ da API GLU NURBS.

Ambas as implementações anteriores não permitiam a representação de superfícies do tipo NURBS com um número arbitrário de *trimming curves*. Este é um facto crucial, dado que o suporte do mesmo permitiria alargar consideravelmente o leque de aplicação das bibliotecas. Para ilustração apenas, considerem-se os modelos NURBS em 3D disponibilizados por G. Casciola em [CASCwp], onde podemos verificar que um número significativo dos mesmos incluem mais do que uma *trimming curve*.

Tendo por base o objectivo de implementar uma biblioteca que permitisse a representação de curvas e superfícies do tipo NURBS com suporte para um número arbitrário de *trimming curves* tínhamos duas hipóteses imediatas a considerar. Poderíamos continuar o trabalho de migração do código C++ para java ou, em alternativa, estender a API jgeom. Se migrássemos o código C++ (OpenGL) para java (JOGL) iríamos, provavelmente, obter melhor desempenho do produto final. No entanto, iríamos perder a flexibilidade de implementarmos uma biblioteca independente do sistema gráfico que nos permitisse trabalhar, por exemplo, com ambas as tecnologias JOGL e Java3D ou, inclusivamente, com outras bibliotecas que venham a existir no futuro. Por outro lado, a biblioteca jgeom foi implementada

com recurso a algoritmos bem conhecidos, simples e de referência na matéria em causa. Após uma análise do código fonte da mesma, verificou-se que os algoritmos implementados são baseados num livro de referência nesta área, a saber o livro *The NURBS book* [PIEG97], de L. A. Piegel. Assim, decidimos, após troca de correspondência com o autor da biblioteca jgeom, Samuel Gerber [GERBwp], e após a obtenção da respectiva autorização[3], estender a biblioteca com as seguintes funcionalidades que não incluía inicialmente:

- suporte de um número arbitrário de *trimming curves*;
- triangulação dos pontos de uma superfície do tipo NURBS recortada por *trimming curves*;
- suporte para ambos os tipos de *knot vectors* mais comuns, a saber, *open knot vectors* e *periodic knot vectors*;
- implementação dos *bindings* para JOGL.

Foram, também, implementados alguns algoritmos com o intuito de melhorar o desempenho da biblioteca na representação de superfícies do tipo NURBS recortadas. Em particular o algoritmo responsável pelo recorte da superfície era uma algoritmo de força bruta e foi substituído por um algoritmo de preenchimento de polígonos ([FOLE96] e [ECKEwp]) adaptado por L.A. Piegel em [PIEG95] para o recorte de superfícies com *trimming curves*.

[3] Não obstante a biblioteca jgeom estar sob a licença GNU GPL License [GNULwp], estabelecemos contacto com o autor da biblioteca no sentido, por um lado, de solicitar a sua autorização para a utilização e ampliação da biblioteca, e, também, com o intuito de ter acesso à última versão do código fonte da mesma. O autor disponibilizou-se ainda para uma eventual integração do trabalho de ampliação no próprio projecto jgeom.

3.1 Análise detalhada da biblioteca jgeom no seu estado inicial

Inicialmente a biblioteca jgeom, na sua vertente de representação de superfícies do tipo NURBS, era composta pelas seguintes classes e interfaces[4] (*package* net.jgeom.core.nurbs):

- *ControlPoint4f* – representa um ponto de controlo da superfície;
- *ControlNet* – representa uma malha bidimensional de pontos de controlo;
- *KnotVector* – representa um vector de nós numa determinada direcção paramétrica;
- *NurbsCurve* (interface) – representa uma curva do tipo NURBS;
- *NurbsSurface* (interface) – representa uma superfície do tipo NURBS;
- *BasicNurbsCurve* – representa uma curva do tipo NURBS;
- *BasicNurbsSurface* – representa uma superfície do tipo NURBS;
- *TrimCurve* – representa uma *trimming curve*.

Para a utilização da biblioteca dever-se-ia criar uma instância da classe *ControlNet* que iria, por sua vez, conter um *array* bidimensional de instâncias da classe *ControlPoint4f* (figura 3.1).

[4] Não se apresentam de forma exaustiva todas as classes da biblioteca mas apenas as relevantes no contexto discutido.

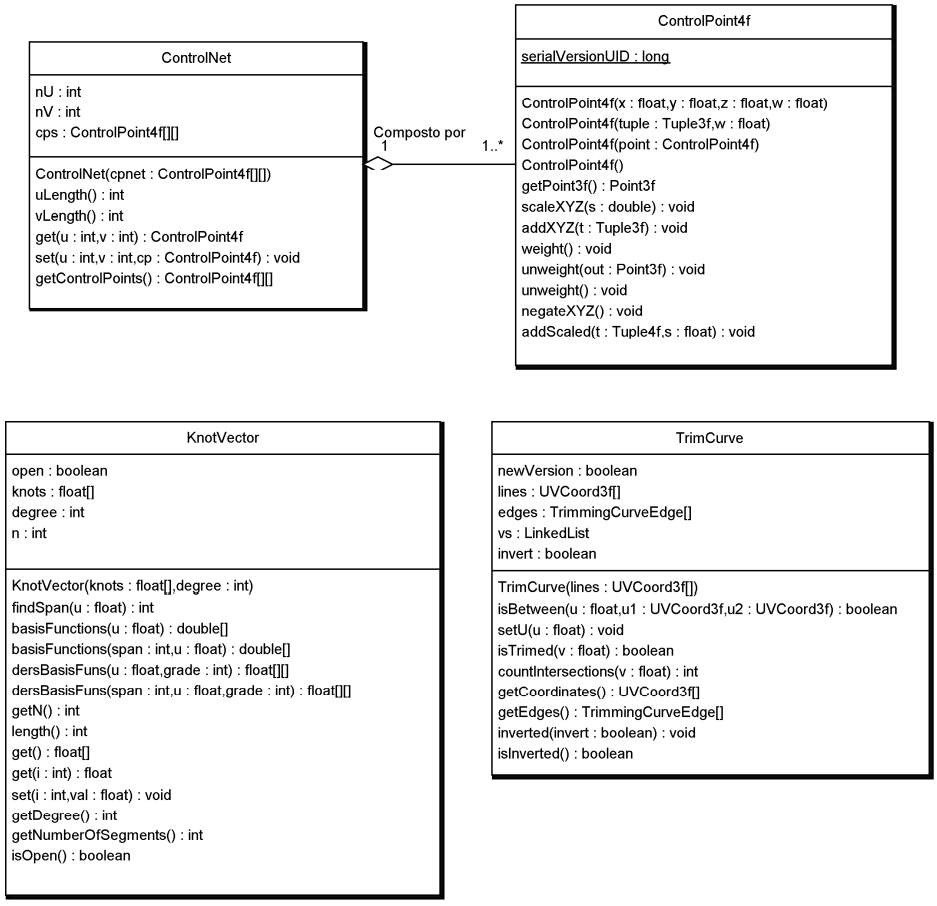

Figura 3.1 – Classes iniciais presentes na biblioteca jgeom

Seguidamente, iríamos criar uma instância da classe que representa uma superfície – *BasicNurbsSurface* – passando como argumentos ao seu construtor uma instância da classe *ControlNet* e de duas instâncias da classe *KnotVector*, uma para cada direcção paramétrica (figura 3.2).

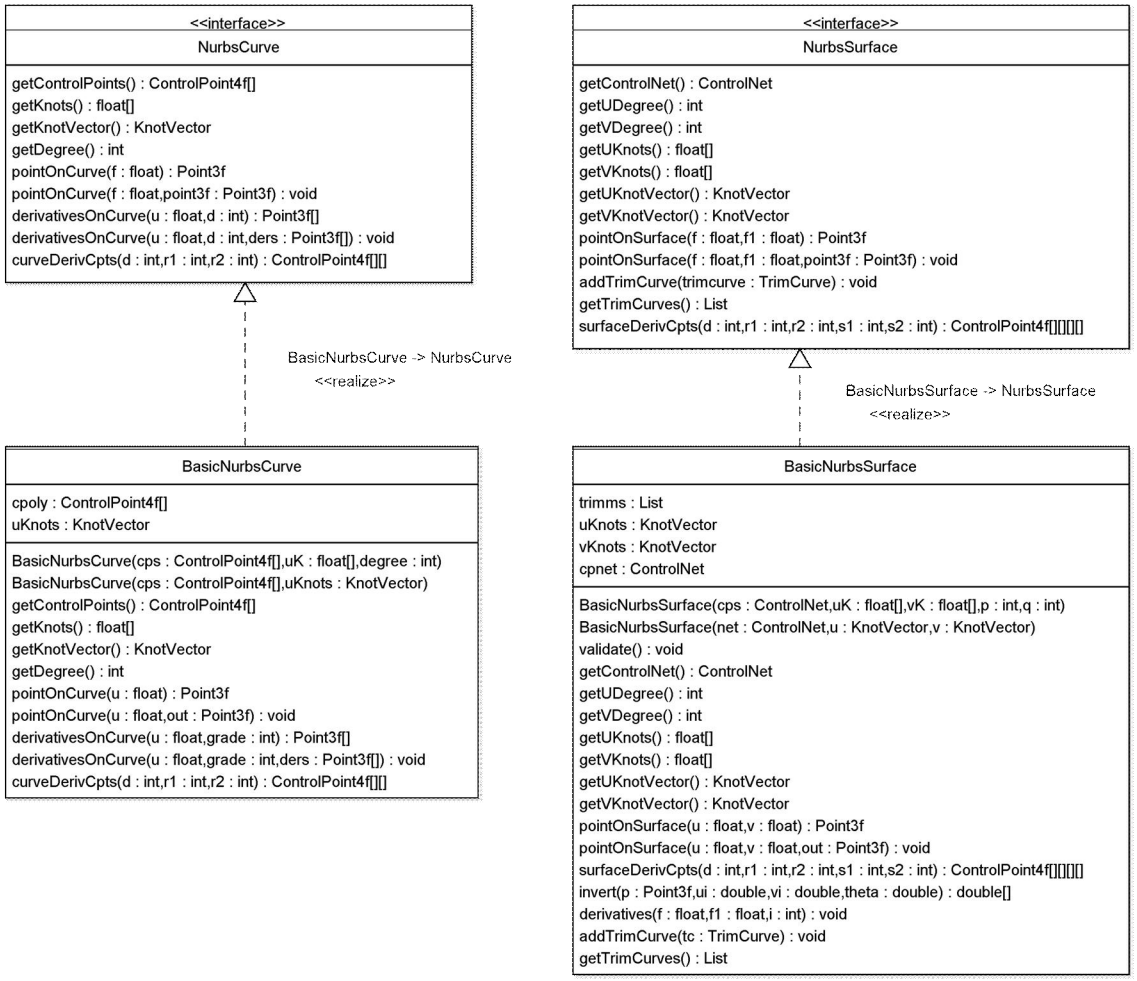

Figure 3.2 – Classes iniciais presentes na biblioteca jgeom

Adicionalmente, para a avaliação dos pontos da superfície a biblioteca inclui algumas classes, denominadas *evaluators*, no *package* net.jgeom.j3d.evaluators que implementam os *bindings* para Java3D:

- *BasicNurbsSurfaceEvaluator* – permite calcular os pontos da superfície em intervalos de amostragem constantes para ambas as direcções paramétricas, fazendo uma subdivisão da superfície em polígonos (quadriláteros) que a aproximam;
- *BasicNurbsCurveEvaluator* – permite calcular os pontos da curva em intervalos de amostragem constantes para o valor do parâmetro;

- *TrimSurfaceEvaluator* – permite calcular os pontos de uma superfície recortada por uma e uma só *trimming curve* não fazendo, no entanto, a subdivisão da superfície em polígonos que a aproximem.

Para o cálculo dos pontos da superfície basta invocar o método *evaluateSurface()* (figura 3.3.) com uma instância da classe que representa a superfície e indicar o número de "linhas de varrimento" em cada direcção paramétrica.

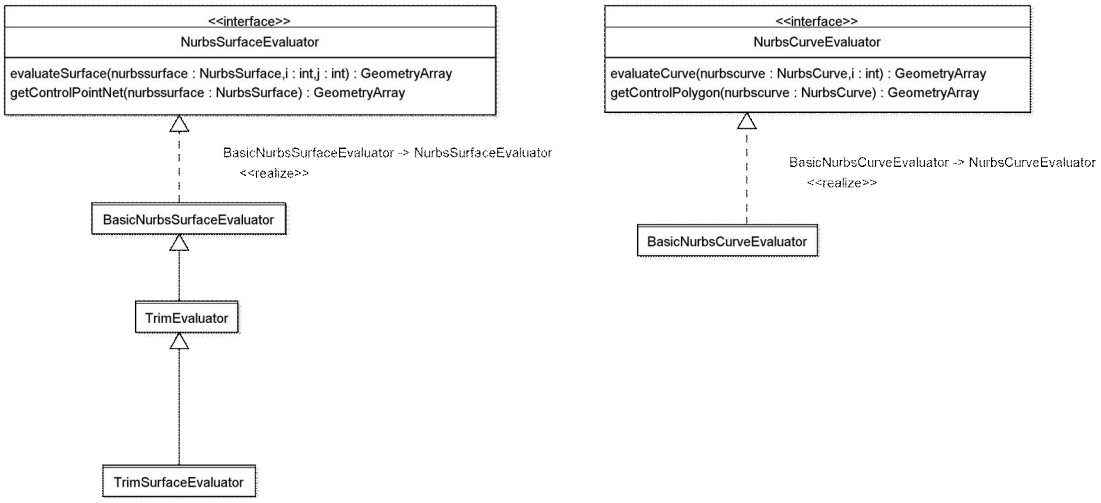

Figura 3.3 – Classes iniciais presentes na biblioteca jgeom

3.2 Ampliação da biblioteca jgeom

Para atingirmos os nossos objectivos foi necessário, por um lado, implementar um algoritmo que permitisse a representação de superfícies do tipo NURBS recortadas por um número arbitrário de *trimming curves* e, por outro lado, encontrar um algoritmo que efectuasse uma triangulação dos pontos calculados.

Para além disso, procurámos incluir na biblioteca um algoritmo que permitisse o cálculo do valor de incremento do parâmetro de forma dinâmica, com o objectivo de não impor ao utilizador a definição obrigatória do número de linhas de varrimento em cada direcção paramétrica. O termo "parâmetro" na frase anterior significa a variável independente nas expressões que definem uma curva ou superfície paramétrica. Por exemplo, na expressão (2.17) que define uma superfície do tipo NURBS, u e v são os parâmetros. Ao procedermos ao cálculo dos pontos de uma curva ou superfície paramétrica fazemos variar o(s) parâmetro(s) no seu domínio (tipicamente o intervalo $[0,1]$) e calculamos o ponto da curva ou superfície correspondente. Ao variarmos o valor do parâmetro, incrementamos o mesmo por uma determinada quantidade que poderá ser constante ou variável, ou seja, o intervalo $\Delta u = u_{i+1} - u_i$, o qual passaremos a designar doravante por valor de incremento do parâmetro, poderá ser constante na avaliação de todos os pontos da curva ou superfície ou poderá variar adaptando-se dinamicamente à forma da curva ou superfície. Este último algoritmo permite calcular o valor de incremento do parâmetro, Δu, de forma dinâmica.

3.2.1 Algoritmo para a representação de superfícies do tipo NURBS com um número arbitrário de *trimming curves*

Em [FOLE96] e em [ECKEwp] é descrito um algoritmo para o preenchimento de polígonos. O mesmo foi adaptado por L. A. Piegel, em [PIEG95], para o preenchimento de superfícies do tipo NURBS com um número arbitrário de *trimming curves*. Este algoritmo optimiza o cálculo dos pontos de intersecção das linhas de varrimento (*scanlines*) com os segmentos de recta das *trimming curves*. Optámos por implementar o mesmo na biblioteca jgeom pelo facto de este suportar um número arbitrário de *trimming curves* e por ser mais eficiente que o algoritmo inicial, do tipo força bruta, em que para cada linha de varrimento eram calculados os pontos de intersecção refazendo, em cada iteração, os respectivos cálculos.

De uma forma resumida, eis os diversos passos do algoritmo:

1. Determinação do valor de incremento do parâmetro;
2. Cálculo do número de linhas de varrimento (que define a resolução da grelha bidimensional a utilizar para dividir o espaço paramétrico);
3. Criação de uma tabela de arestas (TA) com todos os segmentos de recta que compõem as *trimming curves* (descritas ao nível do espaço paramétrico);
4. Cálculo dos pontos da superfície e respectiva triangulação.

O primeiro passo do algoritmo é calcular o valor de incremento para o parâmetro. Este cálculo foi implementado de duas formas distintas e o utilizador da biblioteca poderá optar por uma, ou por outra em alternativa, através de configuração da própria biblioteca.

A primeira utiliza um algoritmo baseado no cálculo das derivadas de ordem 2 da superfície. A segunda limita-se a calcular o valor de incremento tendo por base a especificação do número de linhas de varrimento definidas pelo utilizador da biblioteca. Ambos os métodos serão abordados, em detalhe, na próxima secção.

Uma vez calculado o valor de incremento (o qual é utilizado para ambas as direcções paramétricas), o segundo passo do algoritmo consiste em calcular o número de linhas de varrimento (*number of scanlines*) NSL dado por:

$$NSL = \left\lfloor \left(\frac{v_{\max} - v_{\min}}{step} \right) \right\rfloor + 1 \qquad (3.1)$$

em que v representa uma dada direcção paramétrica, $step$ representa o valor de incremento calculado anteriormente e o símbolo $\lfloor num \rfloor$ representa o maior número inteiro não superior a *num*.

Uma vez calculado o valor de NSL dividimos o espaço paramétrico numa grelha bidimensional de acordo com a representação da figura 3.4.

O terceiro passo do algoritmo consiste na criação de uma estrutura de dados denominada tabela de arestas. Esta estrutura não é mais de que um vector (*array*) unidimensional, com cardinal igual a NSL. O índice deste *array* é um número inteiro que irá corresponder a cada uma das linhas de varrimento da direcção paramétrica v. Cada uma destas entradas irá conter uma lista dos segmentos de recta de *trimming curves* em que um dos seus pontos extremos (aquele que tem o menor valor para a coordenada v) seja intersectado pela linha de varrimento em processamento. Consideremos o exemplo da figura 3.4, em que podemos visualizar uma representação do espaço paramétrico associado a uma determinada superfície do tipo NURBS.

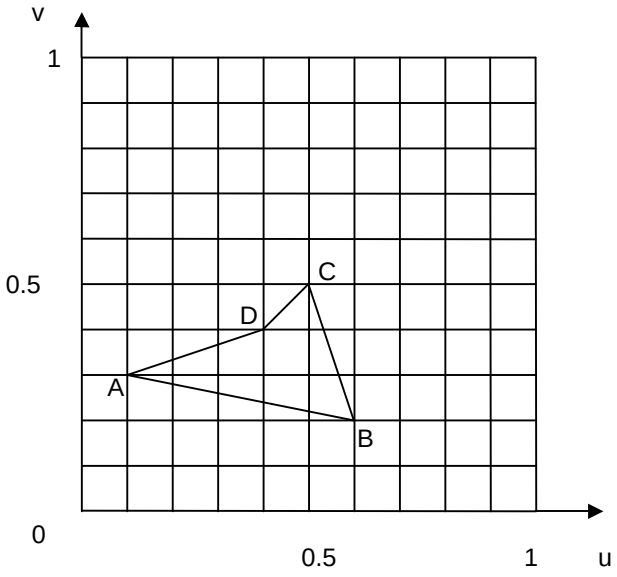

Figura 3.4 – Grelha bidimensional que divide o espaço paramétrico
(estão também representadas as *trimming curves*)

A primeira etapa da construção da tabela de arestas seria converter a representação no espaço paramétrico dos vários segmentos de recta para uma representação por

números inteiros. Por exemplo o segmento de recta *[AB]* no espaço paramétrico é definido pelos pontos de coordenadas (0.1, 0.3) e (0.6, 0.2). Procederíamos, inicialmente, à sua conversão para uma escala inteira tendo por base o número de linhas de varrimento NSL (no exemplo, 11 linhas de varrimento em cada direcção paramétrica). Sendo assim, o mesmo segmento de recta passaria a ser definido pelos pontos de coordenadas (1, 3) e (6, 2). Para o segmento de recta *[BC]* teríamos os pontos de coordenadas (6, 2) e (5, 5), para *[CD]* os pontos (5, 5) e (4, 4) e para *[DA]* os pontos (4, 4) e (1, 3). A tabela de arestas (figura 3.5) iria, assim, consistir num *array* com 11 entradas em que a entrada de índice 2 iria conter uma lista com os segmentos de recta *[AB]* e *[BC]* (porque ambos têm o ponto extremo com menor valor de v igual a 2, i.e, os dois segmentos têm como extremo o ponto B, cujo valor da coordenada v é igual ao índice da tabela de arestas).

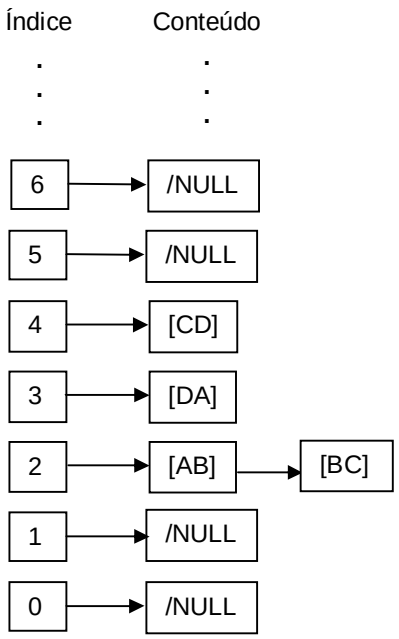

Figura 3.5 – Representação esquemática da tabela de arestas associada ao espaço paramétrico da figura 3.4

Seguidamente iríamos ter uma entrada na tabela de arestas com índice 3 que iria conter apenas o segmento de recta *[DA]*, dado que um dos seus pontos extremos, *A*, tem $v_{\min} = 3$. Por fim, a entrada de índice 4 iria ter o segmento de recta *[CD]*, porque o ponto extremo *D* tem $v_{\min} = 4$. Todas as outras entradas da tabela de arestas seriam vazias (representado na figura por /*NULL*).

Cada segmento de recta de uma *trimming curve* é representado por uma instância da classe *ETEdge* (figura 3.6).

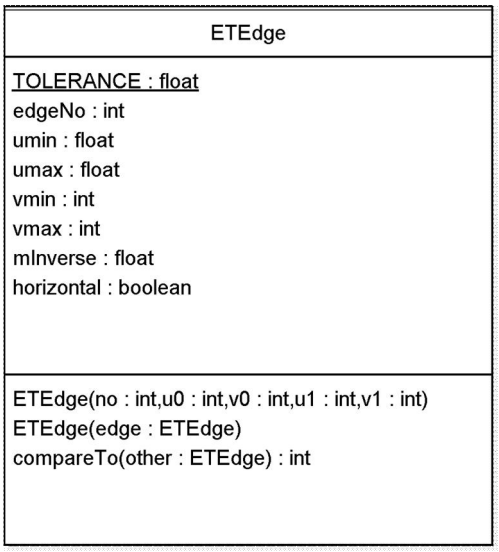

Figura 3.6 – Classe *ETEdge*

O quarto passo do algoritmo consiste no cálculo dos pontos da superfície que não estão recortados, bem como na respectiva triangulação. O algoritmo de triangulação irá ser abordado na secção 3.2.3. Quanto ao cálculo dos pontos da superfície não recortados inicia-se com a construção de uma tabela de arestas activas (TAA) vazia. Uma TAA não é mais do que uma lista de segmentos de recta (instâncias da classe ETEdge) que contém apenas os segmentos de recta que são intersectados pela linha de varrimento em processamento. Esta estrutura vai, assim, evoluindo de forma dinâmica durante a execução do algoritmo. Processa-se,

inicialmente, um ciclo exterior em que cada iteração corresponde a uma linha de varrimento na direcção paramétrica v. Para cada iteração, ou seja, para cada linha de varrimento v_{curr}:

- são adicionados à TAA novos segmentos de recta que a intersectem ($v_{curr} = v_{\min}$);
- são removidos os segmentos de recta que já não a intersectem ($v_{curr} \geq v_{\max}$);
- é ordenada a lista de segmentos de recta por ordem crescente do valor mínimo da coordenada u, $u_{\min}$.

Seguidamente processa-se um ciclo interior, desta vez para as linhas de varrimento em u, em que se calcula o número de intersecções entre a linha de varrimento em processamento v_{curr} e os segmentos de recta da TAA para os quais $u_{\min} \geq u_{curr}$. Se este número for par, o ponto (v_{curr}, u_{curr}) no espaço paramétrico deverá ser considerado válido, caso contrário não o será.

Finalmente, são actualizados os pontos de intersecção dos segmentos de recta com a próxima linha de varrimento em v_{curr+1}.

Em pseudo-código teríamos:

```
TA    → tabela de arestas;
TAA   → tabela de arestas activas;
NSLU  → número de linhas de varrimento em u;
NSLV  → número de linhas de varrimento em v;

Construir TA;
TAA = vazia;
For (v=0 until v<NSLV) {
      Adicionar novos segmentos à TAA ( v_curr = v_min );
      Remover os segmentos não relevantes da TAA ( v_curr ≥ v_max );
      Ordenar TAA por u_min ;
      For (u=0 until u<NSLU) {
            N = número de intersecções tais que u_min ≥ u_curr ;
            Se N é par calcular o ponto da superfície (v_curr, u_curr)
            Se N é ímpar recortar o ponto
      }
      Actualizar na TAA as coordenadas dos pontos de intersecção  para
      u_{min+1} = u_min + 1/m
}
```

Consideremos o exemplo da figura 3.4, a qual se reproduz novamente na figura 3.7 mas com a escala em termos de linhas de varrimento.

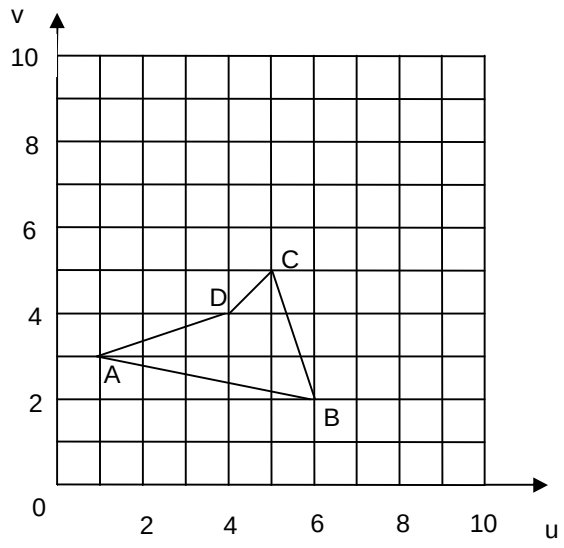

Figura 3.7 – Espaço paramétrico com a representação das *trimming curves*

Para a linha de varrimento $v_{curr} = 2$, por exemplo, teríamos na TAA os segmentos de recta *[AB]* e *[BC]*. Se estivéssemos a processar o valor $u_{curr} = 4$ iríamos ter dois pontos de intersecção tais que $u_{\min} = 6 \geq u_{curr} = 4$, ambos correspondentes ao ponto *B*, extremo dos dois segmentos de recta. Assim, o ponto da superfície dado por (4, 2) será considerado interior à superfície pelo facto do número de pontos de intersecção tais que $u_{\min} \geq u_{curr}$ ser par. Por outro lado se considerássemos a linha de varrimento $v_{curr} = 3$ iríamos incluir na TAA o segmento de recta *[DA]* e remover o segmento *[AB]*. Assim, para o valor $u_{curr} = 1$, por exemplo, iríamos ter dois pontos de intersecção (com ambos os segmentos *[DA]* e *[BC]*) tais que $u_{\min} \geq u_{curr}$, enquanto que para $u_{curr} = 5$, por exemplo, iríamos ter apenas um (com o segmento *[BC]*). O primeiro seria classificado como ponto interior (número par) enquanto que o segundo seria classificado como ponto exterior (número ímpar).

O produto final deste algoritmo é um vector bidimensional de cardinal $NSLU$ x $NSLV$, em que cada entrada corresponde a uma célula da grelha que divide o espaço paramétrico. Cada entrada deste vector irá conter informação relativa à célula, através de uma relação bi-unívoca entre o ponto inferior esquerdo da mesma e uma instância da classe *SurfacePoint*, representada na figura 3.8, e que contém os seguintes atributos:

- *int x_grid* – inteiro que representa o número da coluna;
- *int y_grid* – inteiro que representa o número da linha;
- *double u* – valor de *u* do ponto na grelha bidimensional (ponto inferior esquerdo da célula);
- *double v* – valor de *v* do ponto da grelha bidimensional (ponto inferior esquerdo da célula);

- *int trimmed* – indicação se o ponto está ou não recortado; se não estiver recortado contém o valor -1, caso contrário contém o valor 1;

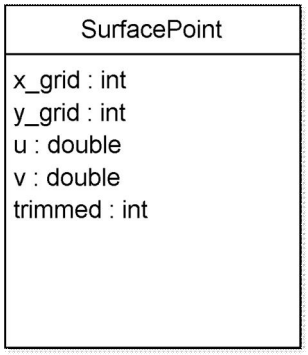

Figura 3.8 – Classe SurfacePoint

A eficiência deste algoritmo, relativamente ao algoritmo já existente na biblioteca (do tipo força bruta), aumenta. Tal deve-se ao facto de o cálculo da coordenada u, do ponto de intersecção de cada segmento de recta com a linha de varrimento, se basear no valor de u calculado na iteração anterior, no valor do inverso do declive de cada segmento de recta (calculado no início do algoritmo no momento da criação da tabela de arestas) e, também, no designado princípio da coerência de linhas de varrimento, i.e., na premissa de que de uma linha de varrimento para a seguinte linha de varrimento a probabilidade de modificação da TAA é baixa ou, por outras palavras, os segmentos de recta de *trimming curves* não variam consideravelmente de uma linha de varrimento para a seguinte linha de varrimento. Assim, o cálculo de $u_{\min}$ da próxima linha de varrimento é dado, simplesmente, pela adição entre $u_{\min}$ da linha de varrimento anterior e o inverso do declive do segmento de recta, ou seja, $u_{\min+1} = u_{\min} + \frac{1}{m}$.

3.2.2 Algoritmo para o cálculo do valor de incremento do parâmetro

Em [PIEG95] é apresentado um método para o cálculo do valor de incremento do parâmetro na representação de superfícies do tipo NURBS. Inicialmente, é definido um valor de tolerância ε de tal forma que a aproximação triangular da superfície não se desvie da superfície ideal matemática por um valor superior a ε. A partir desta premissa é desenvolvido um método que permite o cálculo do valor de incremento. Este valor é dado por:

$$step = \frac{2^{1/2}}{2}\lambda \qquad (3.2)$$

com

$$\lambda = 3\left(\frac{\varepsilon}{2\left(D_1 + 2D_2 + D_3\right)}\right)^{1/2} \qquad (3.3)$$

em que

$$\begin{aligned} D_1 &= \sup\left\|\frac{\partial^2 S(u,v)}{\partial u^2}\right\| \\ D_2 &= \sup\left\|\frac{\partial^2 S(u,v)}{\partial u \partial v}\right\| \\ D_3 &= \sup\left\|\frac{\partial^2 S(u,v)}{\partial v^2}\right\| \end{aligned} \qquad (3.4)$$

É um facto conhecido da teoria das NURBS que as derivadas de segunda ordem de uma superfície deste tipo são elas próprias superfícies do tipo NURBS, sendo os seus pontos de controlo dados pelas relações apresentadas seguidamente, a saber (3.8), (3.11) e (3.14). Por facilidade de exposição, apresentamos o caso de superfícies do tipo B-Splines não racionais, sendo os resultados extensíveis às superfícies do tipo NURBS através da utilização de coordenadas homogéneas.

Como já referido no capítulo 2, uma superfície não racional do tipo B-Spline $S(u,v)$ é definida com base numa rede bidimensional de pontos de controlo $P_{i,j}$, dois vectores de nós $U=\{u_0,u_1,u_2,\ldots,u_{(n+1)+(p+1)}\}$ e $V=\{v_0,v_1,v_2,\ldots,v_{(m+1)+(q+1)}\}$ e os produtos das funções B-Spline $N_{i,p}(u)$ definidos pelas relações (2.11), as quais se repetem por conveniência de exposição:

$$N_{i,0}(u)=\begin{cases}1 & se \quad u_i \le u < u_{i+1}\\ 0 & caso \quad contrário\end{cases} \qquad (2.11)$$

$$N_{i,p}(u)=\frac{u-u_i}{u_{i+p}-u_i}N_{i,p-1}(u)+\frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u)$$

ou seja

$$S(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)P_{i,j}\,. \qquad (2.12)$$

Recordemos que nesta definição temos $n+1$ pontos de controlo na direcção paramétrica *u*, $m+1$ pontos de controlo na direcção paramétrica *v* e funções B-Spline de grau *p* na direcção paramétrica *u* e de grau *q* na direcção paramétrica *v*. Como já afirmado anteriormente, as derivadas de segunda ordem de uma superfície deste tipo são elas próprias superfícies B-Splines, como apresentado, por exemplo, em [ROGE01] e em [PIEG95]. A segunda derivada na direcção paramétrica *u* é dada por:

$$\frac{\partial^2 S(u,v)}{\partial u^2}=\sum_{i=0}^{n-2}\sum_{j=0}^{m}N_{i,p-2}(u)N_{j,q}(v)P_{i,j}^{(2,0)} \qquad (3.5)$$

com

$$P_{i,j}^{(2,0)}=\frac{p(p-1)}{u_{i+p+1}-u_{i+2}}\left(\frac{P_{i+2,j}-P_{i+1,j}}{u_{i+p+2}-u_{i+2}}-\frac{P_{i+1,j}-P_{i,j}}{u_{i+p+1}-u_{i+1}}\right) \qquad (3.6)$$

em que os vectores de nós são dados por

$$U^{(2)} = \{u_0, u_1, u_2, \ldots, u_{(n+1)-2+(p+1)-2}\} \quad (3.7)$$

$$V^{(0)} = V .$$

A segunda derivada na direcção paramétrica *v* é dada por:

$$\frac{\partial^2 S(u,v)}{\partial v^2} = \sum_{i=0}^{n} \sum_{j=0}^{m-2} N_{i,p}(u) N_{j,q-2}(v) P_{i,j}^{(0,2)} \quad (3.8)$$

com

$$P_{i,j}^{(0,2)} = \frac{q(q-1)}{u_{j+q+1} - u_{j+2}} \left(\frac{P_{i,j+2} - P_{i,j+1}}{u_{j+q+2} - u_{j+2}} - \frac{P_{i,j+1} - P_{i,j}}{u_{j+q+1} - u_{j+1}} \right) \quad (3.9)$$

em que os vectores de nós são dados por

$$U^{(0)} = U \quad (3.10)$$

$$V^{(2)} = \{v_0, v_1, v_2, \ldots, v_{(m+1)-2+(q+1)-2}\}.$$

As derivadas parciais mistas são dadas por:

$$\frac{\partial^2 S(u,v)}{\partial u \partial v} = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} N_{i,p-1}(u) N_{j,q-1}(v) P_{i,j}^{(1,1)} \quad (3.11)$$

com

$$P_{i,j}^{(1,1)} = \frac{pq}{u_{j+q+1} - u_{j+1}} \left(\frac{P_{i+1,j+1} - P_{i,j+1} - P_{i+1,j} + P_{i,j}}{u_{i+p+1} - u_{i+1}} \right) \quad (3.12)$$

em que os vectores de nós são dados por

$$U^{(1)} = \{u_0, u_1, u_2, \ldots, u_{(n+1)-1+(p+1)-1}\} \quad (3.13)$$

$$V^{(1)} = \{v_0, v_1, v_2, \ldots, v_{(m+1)-1+(q+1)-1}\}.$$

Os pontos de controlo das superfícies derivadas são calculados a partir dos pontos de controlo da superfície original com as relações anteriormente apresentadas. Utilizámos o algoritmo definido em [PIEG97] (algoritmo A3.3) para este efeito. Em [PIEG95], o autor demonstra que as derivadas de segunda ordem

indicadas em (3.4) são limitadas superiormente pelo máximo das normas dos pontos de controlo das superfícies derivadas de segunda ordem, ou seja,

$$\begin{aligned} D_1 &\leq \max\left\|P_{i,j}^{(2,0)}\right\| \\ D_2 &\leq \max\left\|P_{i,j}^{(1,1)}\right\|. \\ D_3 &\leq \max\left\|P_{i,j}^{(0,2)}\right\| \end{aligned} \tag{3.14}$$

Para o caso das superfícies racionais (NURBS) efectuam-se exactamente os mesmos cálculos trabalhando-se, no entanto, em coordenadas homogéneas. A única diferença consiste em calcular a tolerância no espaço homogéneo, ε^w, de forma a obtermos uma tolerância de ε no espaço tridimensional. Esta relação é dada por:

$$\varepsilon^w = \left(\min w_{i,j}\right)\left(\frac{\varepsilon}{1+\max\left\|P_{i,j}\right\|}\right). \tag{3.15}$$

Utilizando este método é, assim, possível calcular de forma dinâmica um valor para o incremento do parâmetro.

Em alternativa ao método anteriormente apresentado, também é possível que o utilizador da API, o programador, indique um número fixo de linhas de varrimento em cada uma das direcções paramétricas. Neste caso, o cálculo do valor do incremento é dado simplesmente pelas relações:

$$step_u = \frac{u_{\max} - u_{\min}}{NSL_u} \tag{3.16}$$

$$step_v = \frac{v_{\max} - v_{\min}}{NSL_v} \tag{3.17}$$

Estes dois métodos podem ser escolhidos pelo utilizador da API através da simples indicação de um parâmetro do método gluNurbsProperty() da classe GLU. Caso seja indicado o número de linhas de varrimento será utilizado o cálculo do

valor de incremento estático, caso contrário será utilizado o método dinâmico baseado nas segundas derivadas.

3.2.3 Algoritmo de triangulação dos pontos da superfície

Com o intuito de obtermos uma triangulação para os pontos calculados da superfície complementámos o algoritmo descrito na secção 3.2.1 com a lógica que descrevemos de seguida.

A ideia base do algoritmo de triangulação é muito simples e está descrita pelo seu autor, Vincent Prat, em *NURBS Curves and Surfaces Tutorial* [PRATwp]. Considere-se a figura 3.9. Partimos do produto final do algoritmo descrito na secção 3.2.1, i.e., partimos da grelha bidimensional de instâncias da classe *SurfacePoint*, em que cada instância nos indica se o ponto é interior ou exterior à superfície e procedemos como a seguir se descreve.

O primeiro passo consiste na identificação dos *pontos fronteira*. Por *ponto fronteira* entenda-se qualquer ponto interior que tem pelo menos um ponto adjacente que seja exterior. No exemplo da figura 3.9, os pontos fronteira encontram-se assinalados com uma cruz. A identificação dos pontos fronteira é realizada com um algoritmo que, de forma iterativa para cada ponto da grelha, verifica se pelo menos um dos oito pontos adjacentes do ponto em processamento tem um estado diferente (indicado pelo atributo *trimmed* da classe *SurfacePoint*). Em caso afirmativo, o ponto é considerado um *ponto fronteira*. Este algoritmo é válido para todos os pontos com excepção da primeira linha, primeira coluna, última linha e última coluna. Nestes casos cada ponto tem apenas cinco pontos adjacentes.

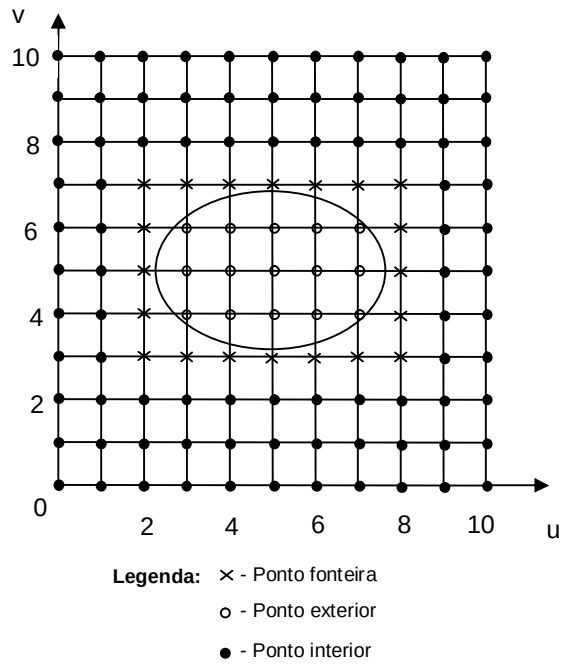

Figura 3.9 – Algoritmo de Triangulação

Num segundo passo, para cada ponto fronteira (F) vamos identificar o ponto da *trimming curve* (T) mais próximo de F, e substituir as coordenadas de F pelas coordenadas de T. Recorde-se que este processamento é efectuado no espaço paramétrico bidimensional de coordenadas *(u, v)*. Recorde-se, também, que as *trimming curves* são aproximadas por segmentos de recta e, portanto, não são mais do que polígonos orientados, em que os vértices são os pontos extremos dos segmentos de recta que as aproximam. Assim sendo, este algoritmo é tão simples quanto calcular a distância entre os pontos que definem a *trimming curve* e o ponto fronteira em processamento.

O terceiro e último passo do algoritmo consiste na produção de triângulos cujos vértices são os pontos interiores da grelha bidimensional, com as coordenadas dos pontos fronteira modificadas de acordo com o descrito anteriormente.

Apresenta-se na figura 3.10 a triangulação produzida para uma das superfícies utilizadas durante a implementação.

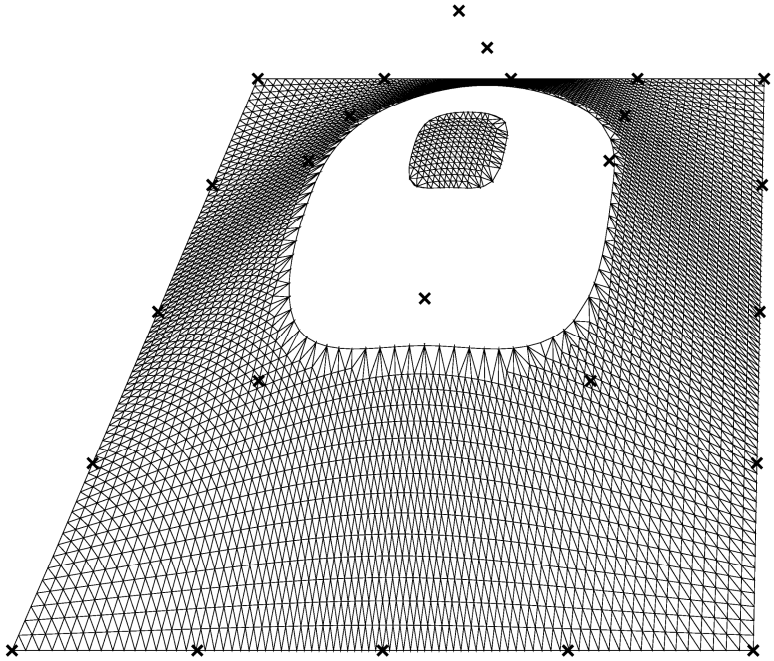

Figura 3-10 – Resultado do algoritmo de triangulação

A construção dos triângulos baseia-se na capacidade que usualmente as bibliotecas gráficas possuem designada por *triangle strips*. Este modo utiliza os vértices enviados para o processador gráfico de forma tal que constrói um triângulo por cada conjunto sequencial de três vértices. Considere-se o exemplo da figura 3.11. De acordo com o modo *triangle strips*, seriam construídos neste exemplo os triângulos $[V_1V_2V_3]$, $[V_2V_3V_4]$, $[V_3V_4V_5]$, $[V_4V_5V_6]$, $[V_5V_6V_7]$, $[V_6V_7V_8]$.

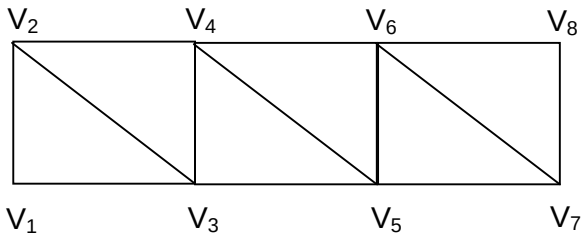

Figura 3.11 – Modo de construção de triângulos designado por *triangle strips*

A triangulação é produzida para cada linha da grelha bidimensional, sendo cada *triangle strip* iniciado na primeira coluna, e sempre que exista uma mudança de um ponto exterior para um ponto interior. É finalizado na situação inversa, ou seja sempre que exista uma mudança de um ponto interior para um ponto exterior, e quando se atinge a última coluna.

Este algoritmo permitiu obter resultados aceitáveis, para as superfícies testadas, para valores de NSL superiores a algumas dezenas. Um valor típico é 60 linhas de varrimento em cada direcção paramétrica. Este valor para NSL é, grosso modo, o valor produzido pelo algoritmo descrito na secção 3.2.2 para o cálculo do valor de incremento do parâmetro, se definirmos para valor de tolerância ε um valor de 0,1.

Este algoritmo de triangulação é um algoritmo simples mas essa simplicidade tem um preço. O algoritmo não produz uma triangulação aceitável em situações em que duas *trimming curves* se encontrem demasiado próximas uma da outra. “Demasiado próximas” neste contexto, significa que não existe nenhum ponto não recortado da superfície entre as duas trimming curves. Acresce que o algoritmo também não suporta o recorte da superfície quando a *trimming curve* está totalmente contida numa única célula. Ambos os problemas podem ser ultrapassados ajustando o valor de incremento do parâmetro para valores menores. Note-se, também, que o algoritmo não produz uma triangulação de *Delaunay*.

3.2.4 Algoritmos para a subdivisão de curvas do tipo NURBS em segmentos de recta

Uma outra questão em aberto, pelo facto de ser suportada apenas de forma parcial pela biblioteca jgeom, é a da subdivisão de uma curva do tipo NURBS em segmentos de recta, i.e., como determinar quais os segmentos recta que devem ser definidos para efectuar a aproximação da curva. Por outras palavras, pretendemos um algoritmo que permita determinar os sucessivos valores do parâmetro t para os quais irão ser avaliados os pontos da curva de forma a definir os vários segmentos de recta (figura 3.11).

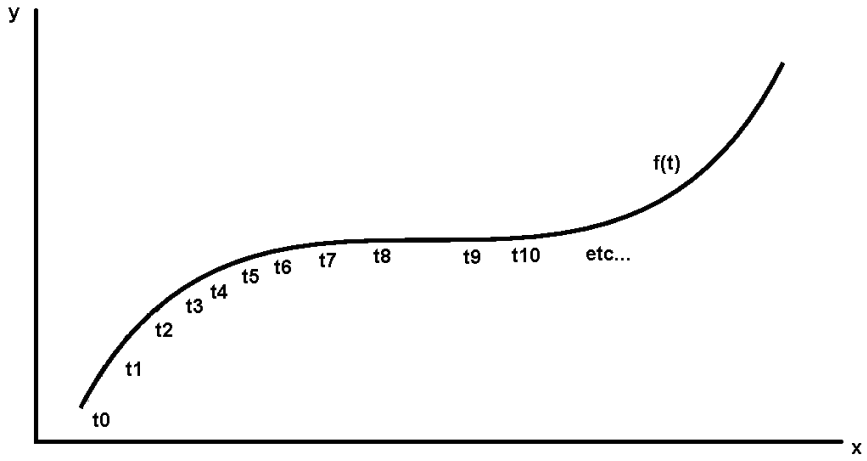

Figura 3.11 – Subdivisão das *trimming curves* em segmentos de recta

Os intervalos no espaço paramétrico entre os sucessivos valores do parâmetro t, i.e., $t_1 - t_0$, $t_2 - t_1$, etc. não têm necessariamente que ser iguais. Podem variar, adaptando-se à curva de forma tal que as suas secções com maior curvatura sejam descritas por um maior número de segmentos de recta, definidos por intervalos paramétricos menores.

A biblioteca jgeom suportava inicialmente apenas intervalos paramétricos iguais. Isto é, no início da avaliação da curva era definido o número de segmentos

de recta – *NS* – que a deveriam aproximar, e o cálculo do intervalo paramétrico era determinado pelo quociente entre toda a extensão do espaço paramétrico ($t_{final} - t_{inicial}$) e *NS*.

No entanto, a API GLU NURBS utilizada pelo OpenGL não obriga, na especificação de *trimming curves*, que seja indicado o número de segmentos de recta que irão aproximar uma curva do tipo NURBS. Este cálculo é efectuado de forma dinâmica pela própria API. Assim, decidimos implementar um algoritmo que permitisse determinar, de forma adaptável, o valor de incremento do parâmetro para uma curva deste tipo.

Começámos por implementar um algoritmo empírico baseado no raio de curvatura e, mais tarde, verificámos a existência de trabalho desenvolvido nesta área que se adaptava na perfeição ao nosso objectivo. Descrevemos, de seguida, ambos os algoritmos.

Algoritmo empírico para subdivisão de uma curva do tipo NURBS em segmentos de recta

Este algoritmo tem por ideia chave a variação do valor de incremento do parâmetro na razão directa do raio de curvatura, i.e, quando diminui o raio de curvatura diminui, também, o valor de incremento do parâmetro. Assim conseguimos obter, para maiores curvaturas[5], intervalos menores para o parâmetro t, o que, intuitivamente, corresponde a uma representação da curva mais satisfatória.

[5] A curvatura define-se como o inverso do raio de curvatura.

Na primeira abordagem que efectuámos, limitámo-nos a utilizar o valor numérico do raio de curvatura de uma curva bidimensional $C(t)=(x(t),y(t))$, dado pela expressão (3.18)

$$\rho=\frac{\sqrt{\left(\left(\frac{dx}{dt}\right)^2+\left(\frac{dy}{dt}\right)^2\right)^3}}{\left|\frac{dx}{dt}\frac{d^2y}{dt^2}-\frac{dy}{dt}\frac{d^2x}{dt^2}\right|}. \qquad (3.18)$$

Depois de implementada esta solução revelou-se inadequada para um representação qualitativamente aceitável da curva. Por esse motivo decidimos relativizar o valor numérico do raio de curvatura dividindo-o pela norma do vector posição em cada ponto da curva.

Adicionalmente, para conseguirmos obter bons resultados em secções da curva onde exista uma mudança brusca do raio de curvatura, implementámos a lógica que a seguir se descreve. Quando o raio de curvatura diminui por um determinado factor (por exemplo, quando diminui para metade), diminuímos também o valor do incremento calculado pelo mesmo factor. Consideremos uma situação em que o algoritmo detecta que no intervalo paramétrico $[t_i,t_{i+1}]$ o raio de curvatura diminui para metade, i.e., $\rho(t_{i+1})=\frac{1}{2}\rho(t_i)$. Então, o algoritmo em vez de avaliar o próximo ponto da curva para o valor do parâmetro t_{i+1} vai fazê-lo para o ponto intermédio entre t_i e t_{i+1} ($factor=2$).

Uma última nota relativa a este algoritmo. Pela expressão (3.18) podemos verificar que para o cálculo do raio de curvatura é necessário o cálculo da primeira e segunda derivadas em ordem ao parâmetro *t*. Como a biblioteca jgeom não tinha

implementado nenhum método para o cálculo das derivadas de uma curva do tipo NURBS, implementámos o algoritmo para o cálculo das mesmas, até uma ordem arbitrária, baseado no algoritmo 2.3 descrito em [PIEG97]. Apresentamos o pseudo-código do mesmo no apêndice I.

Algoritmo baseado na limitação da grandeza *chord error*

Em [YEHS02], [LIUX05] e [LAIJ07] é apresentado um método para o cálculo do valor de incremento do parâmetro para a representação de curvas, baseado na imposição de um limite máximo para a grandeza designada por *chord error*. Este método baseia-se na transformação do parâmetro da curva u numa função do tempo $u(t)$, i.e., transforma a própria função que descreve a curva $C(u)$ numa função composta do tempo $C(u(t))$. Seguidamente, desenvolvendo a função $u(t)$ numa série de Taylor de primeira ordem, deduz-se que o valor do incremento Δu é dado por

$$\Delta u = \left.\frac{du}{dt}\right|_{t=t_i} .(t_{i+1} + t_i) , \qquad \text{com} \quad u_{i+1} = u_i + \Delta u . \tag{3.19}$$

A partir desta consideração, em [YEHS02], deduz-se que

$$\Delta u = \frac{2\sqrt{\rho_i^2 - (\rho_i - \delta_{\max})^2}}{\left\| \frac{dC(u)}{du} \right\|_{u=u_i}} \tag{3.20}$$

sendo $\delta_{\max}$ o valor máximo pretendido para a grandeza "chord error" (constante a definir no algoritmo), ρ_i o raio de curvatura para o valor do parâmetro u_i e

$$\left\| \frac{dC(u)}{du} \right\|_{u=u_i} = \sqrt{\left(\left.\frac{dx}{du}\right|_{u=u_i} \right)^2 + \left(\left.\frac{dy}{du}\right|_{u=u_i} \right)^2 + \left(\left.\frac{dz}{du}\right|_{u=u_i} \right)^2} . \tag{3.21}$$

Assim, é possível calcular o valor do incremento do parâmetro (Δu) com base no raio de curvatura, nas derivadas de primeira ordem e num valor máximo indicado para a grandeza "chord error". Reproduz-se, seguidamente, o pseudo-código do algoritmo implementado:

```
DIF_MAX_MIN = 4;
MAX_CHORD_ERROR = 0.001;
MAX_RADIUS = 100;
```

u → parâmetro da curva;
segnum → número de máximo de segmentos de recta (configurável pelo utilizador);

Point3f → classe que representa um ponto tridimensional no espaço euclidiano.

Point3f[] ders → array com os valores das derivadas para um determinado valor do parâmetro u (ders[0] representa o ponto da curva, ders[1] representa a primeira derivada, ders[2] representa a segunda derivada, etc).

calculaDerivadas(u, k) → função para calcular as derivadas para o valor u do parâmetro até à ordem k;

calculaRaioCurvatura(ders[1], ders[2])→ função para calcular o raio de curvatura;

calculaStep(raio, ders[1], MAX_CHORD_ERROR) → função para calcular o valor do incremento com base na expressão apresentada anteriormente;

Início

```
minStep = (umax - umin) / segnum;
maxStep = minStep * DIF_MAX_MIN;

ArrayList<Point3f> curvePoints = new ArrayList<Point3f>;

u = umin;
while (u<umax) {

      Point3f[] ders = calculaDerivadas(u,2);
      curvePoints.add(ders[0]);

      raio = calculaRaioCurvatura(ders[1], ders[2]);
      if (raio>MAX_RADIUS)
            raio = MAX_RADIUS;

      step = calculaStep(raio, ders[1], MAX_CHORD_ERROR);

      if (step>maxStep) {
```

```
            step = maxStep;
      }

      u = u + step;

}

//Compute last curve point
Point3f[] ders = calculaDerivadas(umax,2);
curvePoints.add(ders[0]);
```

3.3 Implementação de "bindings" para JOGL

Descrevemos nesta secção os novos *packages* implementados na biblioteca jgeom[6] com especial ênfase para o *package* relativo aos novos *evaluators* e, também, para o *package* que implementa o código de *binding* para a API JOGL (figuras 3.12, 3.13 e 3.14).

Implementámos no *package* net.jgeom.jogl.evaluators os algoritmos descritos nas secções anteriores. Assim, temos as classes seguintes:

- TrimSurfaceEvaluator – classe que implementa o cálculo dos pontos de uma *trimmed NURBS surface* mas com uma aproximação do tipo força bruta, i.e., foi a primeira classe a ser implementada e não contempla ainda a lógica subjacente à tabela de arestas activas, limitando-se a suportar várias *trimming curves*;
- TrimSurfaceEvaluatorAET – classe que implementa o cálculo dos pontos de uma *trimmed NURBS surface*. Nesta classe são implementados os algoritmos descritos nas secções 3.2.1, 3.2.2 e 3.2.3;

[6] Iremos mencionar apenas as classes mais relevantes, não incluindo de forma extensiva todas as classes implementadas.

- NurbsCurveEvaluatorCommon – classe abstracta que implementa o método radiusOfCurvature() que permite efectuar o cálculo do raio de curvatura de uma curva dadas as derivadas paramétricas de primeira e segunda ordem;
- AdaptiveNurbsCurveEvaluator – classe que implementa o algoritmo empírico descrito na secção 3.2.4 para a aproximação de uma curva do tipo NURBS por segmentos de recta;
- AdaptiveNurbsCurveEvaluatorChordErrorTaylor – classe que implementa o algoritmo baseado na grandeza *chord error* descrito na secção 3.2.4 para a aproximação de uma curva do tipo NURBS por segmentos de recta.

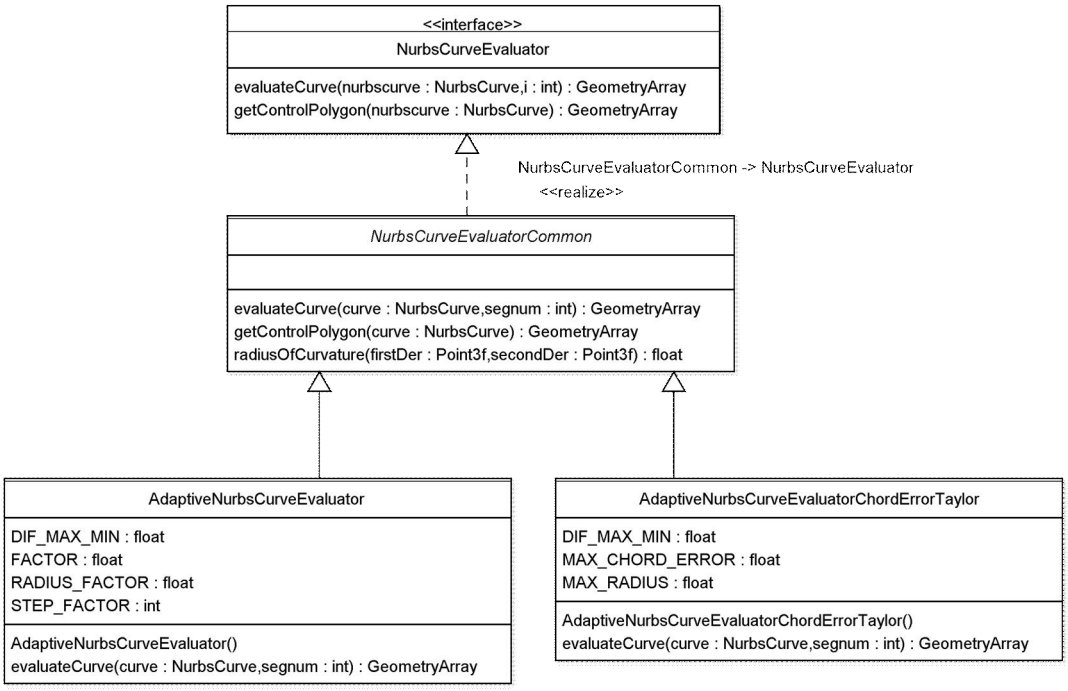

Figura 3.12 – Novas classes do tipo *evaluators* implementadas na biblioteca jgeom

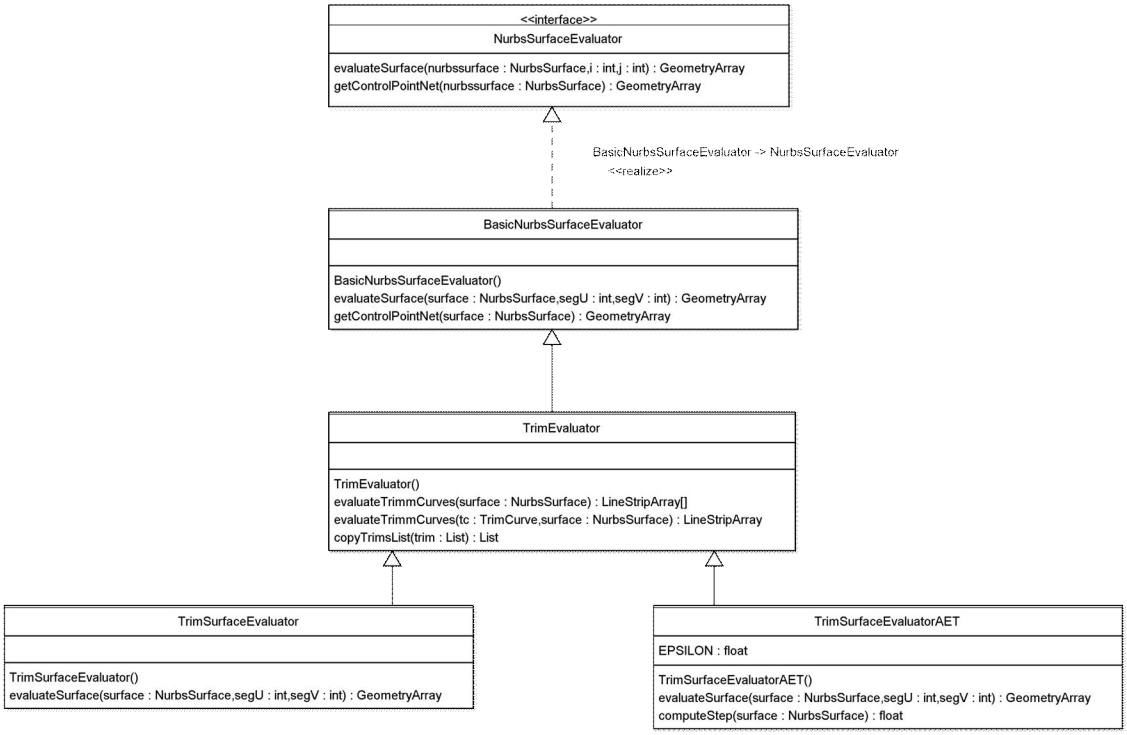

Figura 3.13 – Novas classes do tipo *evaluators* implementadas na biblioteca jgeom

No *package* net.jgeom.jogl.bindings implementámos as classes que estabelecem a relação (*mapping*) entre a biblioteca jgeom e a API JOGL propriamente dita. Assim, temos as seguintes classes (figura 3.14):

- *GLUNurbs* – esta classe herda da classe javax.media.opengl.glu.*GLU* e implementa os métodos relativos a NURBS;
- *Nurbs* – esta classe herda da classe *GLUnurbsImpl* e encapsula os dados relativos a uma superfície do tipo NURBS, em particular uma instância da classe *BasicNurbsSurface* com toda a informação relativa à superfície.

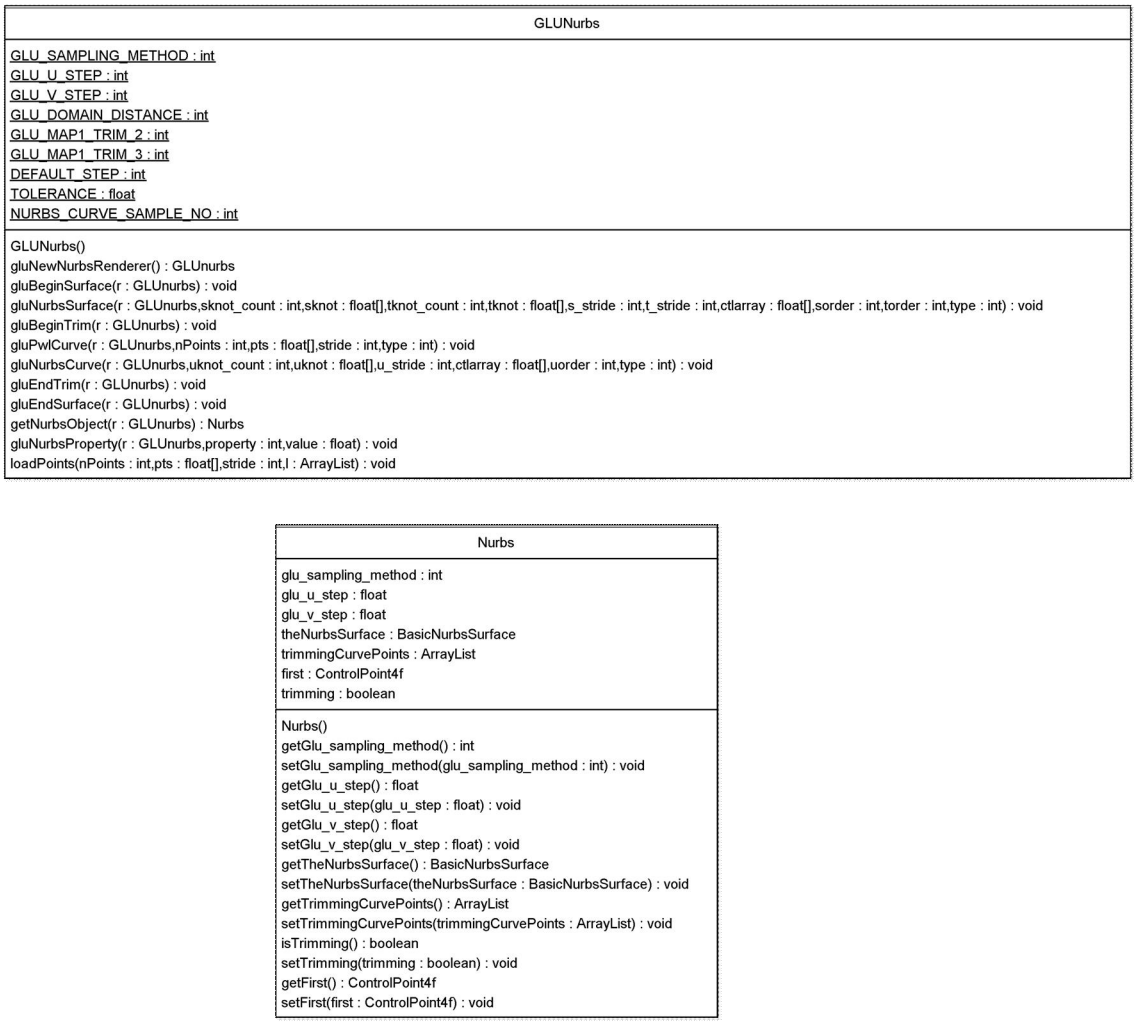

Figura 3.14 – Novas classes de *bindings* implementadas na biblioteca jgeom

Relativamente às funcionalidades disponibilizadas pela API GLU NURBS implementada em C para o OpenGL, não disponibilizámos em Java (através da construção de *bindings* da API jgeom para a tecnologia JOGL) as seguintes:

- Implementação das funcionalidades de *callback* e, em particular, do tratamento de erros;
- Implementação das propriedades da API definidas pela função gluNurbsProperty() com excepção das propriedades GLU_U_STEP e GLU_V_STEP.

3.4 Estudo comparativo dos resultados obtidos com os vários algoritmos

Um dos algoritmos implementados neste trabalho permite-nos representar uma superfície do tipo NURBS recortada por um número arbitrário de *trimming curves*. Este algoritmo baseia-se na utilização de uma tabela de arestas activas, o que aumenta a eficiência do mesmo relativamente a outros algoritmos mais simples.

Propomo-nos, na secção 3.4.1, comparar o algoritmo referido com outro algoritmo implementado inicialmente e que se baseia no cálculo consecutivo dos pontos de intersecção entre as linhas de varrimento e os segmentos de recta das *trimming curves*, tendo por base a equação de uma recta na sua forma explícita $y = mx + b$.

Na secção 3.4.2 iremos comparar outros dois algoritmos implementados com a finalidade de subdividir uma curva do tipo NURBS em segmentos de recta. Pretendemos observar o comportamento destes algoritmos, não numa perspectiva de tempo de execução, mas sim numa perspectiva de número de pontos gerados para a aproximação da curva por segmentos de recta.

3.4.1 Algoritmo para a representação de superfícies do tipo NURBS com um número arbitrário de *trimming curves*

O algoritmo para o cálculo dos pontos de uma superfície do tipo NURBS existente inicialmente na biblioteca jgeom foi modificado, neste trabalho, essencialmente por dois motivos. O primeiro motivo prende-se com o facto de o algoritmo referido não permitir a utilização de mais do que uma *trimming curve*. O segundo motivo está relacionado com o facto de o algoritmo não suportar a triangulação dos pontos da superfície, de forma a aproximar a mesma por uma malha de polígonos.

Numa primeira aproximação para a resolução destes dois problemas, o algoritmo inicialmente existente na biblioteca foi ampliado de forma a calcular os pontos de intersecção entre as linhas de varrimento e os segmentos de recta das várias *trimming curves* com base na equação de uma recta $v = mu + b$, em que v e u são, como habitualmente, os dois parâmetros do espaço paramétrico, m é o declive do segmento de recta e b é o valor do parâmetro v para $u = 0$. O declive m é obtido com base nos pontos terminais do segmento de recta $(u_{\min}, v_{\min})$ e $(u_{\max}, v_{\max})$, através da expressão $m = \frac{\Delta v}{\Delta u} = \frac{v_{\max} - v_{\min}}{u_{\max} - u_{\min}}$. A partir do número de pontos de intersecção calculados para cada linha de varrimento é possível determinar se um determinado ponto do espaço paramétrico deve ou não ser recortado da superfície (secção 3.2.1). Posteriormente esta lógica foi substituída por outra mais eficiente, baseada na utilização de uma tabela de arestas activas (como descrito, também, na secção 3.2.1). O ganho em eficiência obtido está relacionado, precisamente, com o cálculo dos pontos de intersecção. É calculado, previamente, para cada segmento de recta, o inverso do seu declive, e através da expressão para

o cálculo do mesmo, $m = \frac{\Delta v}{\Delta u}$, deduz-se que para incrementos de uma unidade na direcção paramétrica v o incremento em u é dado por $\Delta u = \frac{\Delta v}{m} = \frac{1}{m}$ (com $\Delta v = 1$). Tem-se, então, $u_{curr+1} = u_{curr} + \frac{1}{m}$ dado que $\Delta u = u_{curr+1} - u_{curr}$, ou seja, os pontos de intersecção para a próxima linha de varrimento, u_{curr+1}, são obtidos através de uma única operação de adição (a adição da parcela $\frac{1}{m}$ aos pontos de intersecção da linha de varrimento em processamento, u_{curr}). Tal facto representa um ganho relativamente ao algoritmo inicial, no qual são necessárias uma adição e uma multiplicação (provenientes da equação da recta $v = mu + b$) para o cálculo dos pontos de intersecção. Acresce que este algoritmo ainda tinha, inicialmente, mais um factor de ineficiência que consistia no facto de o declive ser recalculado em cada passo da iteração, não existindo o cálculo único do mesmo para cada segmento de recta no início do algoritmo e posterior utilização durante o cálculo dos pontos de intersecção.

Para comparar a eficiência destes dois algoritmos medimos os tempos de execução dos mesmos para quatro superfícies distintas. Para metade das superfícies (superfícies 1 e 2) utilizámos vectores de nós do tipo *open* enquanto que para a outra metade (superfícies 3 e 4) utilizámos vectores de nós do tipo *periodic*.

Fizemos variar o "número de linhas de varrimento" como variável independente assumindo os valores 50, 100, 200 e 400 (utilizámos sempre o mesmo número de linhas de varrimento para ambas as direcções paramétricas).

Com o intuito de obtermos um valor do erro associado efectuámos três medidas para cada uma das condições de execução e calculámos o valor médio e o desvio padrão das mesmas. Os resultados obtidos estão sumariados no quadro 3.1.

	50 linhas de varrimento					
	Algoritmo Inicial		Tabela de arestas activas		% tempo execução	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Superfície 1	2581	10	2080	19	80,58%	0,89%
Superfície 2	1370	17	1016	1	74,14%	0,90%
Superfície 3	1363	15	999	6	73,30%	1,14%
Superfície 4	2557	11	2097	21	81,99%	0,56%

	100 linhas de varrimento					
	Algoritmo Inicial		Tabela de arestas activas		% tempo execução	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Superfície 1	10234	56	8548	119	83,52%	0,72%
Superfície 2	5750	19	4300	72	74,79%	1,07%
Superfície 3	5720	52	4336	100	75,79%	1,10%
Superfície 4	10209	43	8514	21	83,39%	0,14%

	200 linhas de varrimento					
	Algoritmo Inicial		Tabela de arestas activas		% tempo execução	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Superfície 1	41954	149	36104	166	86,06%	0,50%
Superfície 2	23472	157	18091	163	77,07%	0,27%
Superfície 3	23438	77	18015	41	76,86%	0,29%
Superfície 4	41920	248	36394	184	86,82%	0,11%

	400 linhas de varrimento					
	Algoritmo Inicial		Tabela de arestas activas		% tempo execução	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Superfície 1	164990	5214	141943	4745	86,04%	1,51%
Superfície 2	91717	2993	71575	2228	78,04%	0,49%
Superfície 3	91803	3280	71176	2079	77,54%	0,51%
Superfície 4	163925	5411	142154	4800	86,72%	0,26%

Quadro 3.1 – Tempos de execução dos algoritmos em milisegundos[7]

[7] Estes tempos foram obtidos numa máquina com um processador Intel Core 2 Duo CPU a 2.20GHz com 3GB de memória RAM.

Para cada valor da variável independente “número de linhas de varrimento”, para cada uma das superfícies, medimos o tempo de execução do algoritmo inicial e do algoritmo baseado numa tabela de arestas activas. Apresentamos nas duas últimas colunas a razão entre os tempos de execução do segundo e do primeiro algoritmo (sob a forma de percentagem), bem como o desvio padrão associado. Podemos verificar no quadro que o algoritmo baseado na tabela de arestas activas tem um tempo de execução que varia entre cerca de 73% a cerca de 86% do tempo de execução do algoritmo inicial, o que representa um ganho de eficiência com algum significado. Por outro lado, não se observa uma tendência evidente com o aumento no número de linhas de varrimento, apesar de existir um ligeiro aumento da percentagem quando o número de linhas de varrimento aumenta de 50 para 100 e, posteriormente, de 100 para 200. No entanto, este aumento já não é significativo quando passamos das 200 linhas de varrimento para as 400. Este facto indica que ambos os algoritmos aparentam ter a mesma variação de eficiência em função da variável independente “número de linhas de varrimento” e, após a análise detalhada dos valores absolutos dos tempos de execução, verificamos que para o intervalo [50, 400] ambos os algoritmos se aproximam do comportamento de uma função $O(n^2)$ (quadro 3.2).

x	x^2	$x^2/2$
50	2500	1250
100	10000	5000
200	40000	20000
400	160000	80000

Algoritmo Inicial				
Nº Linhas Varrimento	**Sup 1**	**Sup 2**	**Sup 3**	**Sup 4**
50	2581	1370	1363	2557
100	10234	5750	5720	10209
200	41954	23472	23438	41920
400	164990	91717	91803	163925

Tabela de arestas activas				
Nº Linhas Varrimento	**Sup 1**	**Sup 2**	**Sup 3**	**Sup 4**
50	2080	1016	999	2097
100	8548	4300	4336	8514
200	36104	18091	18015	36394
400	141943	71575	71176	142154

Quadro 3.2 – Tempos de execução dos algoritmos em milisegundos

3.4.2 Algoritmos para a subdivisão de curvas do tipo NURBS em segmentos de recta

Comparámos os dois algoritmos apresentados na secção 3.2.4 aplicados a quatro curvas diferentes com o intuito de obtermos dados sobre o número de pontos calculados para a aproximação das curvas por segmentos de recta. Apresentamos, no quadro 3.3, o resumo dos dados obtidos e, nas figuras 3.15 a 3.18, uma representação das curvas aproximadas por segmentos de recta tendo por base o algoritmo que menos pontos gerou, ou seja, o algoritmo baseado na avaliação da grandeza *chord error*.

	Nº pontos	
	Algoritmo Empírico	**Algoritmo Chord Error**
Curva 1	65	38
Curva 2	75	40
Curva 3	88	42
Curva 4	52	28

Quadro 3.3 – Comparação dos dois algoritmos (número de pontos calculados)

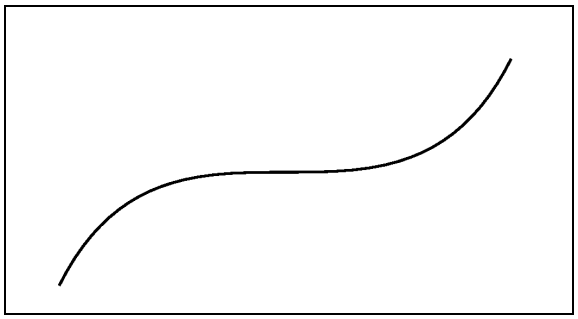

Figura 3.15 – Curva 1

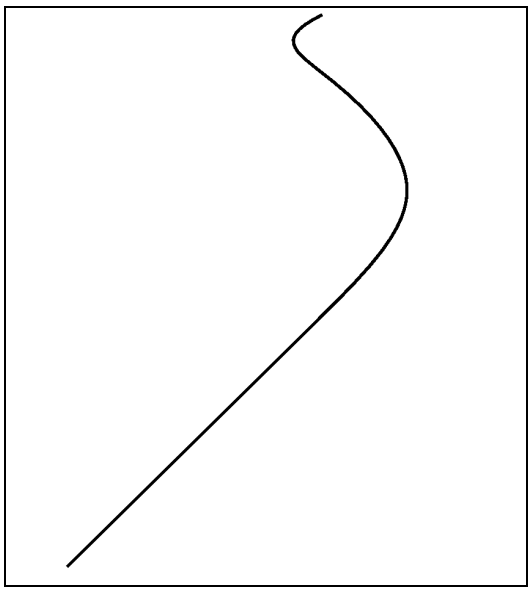

Figura 3.16 – Curva 2

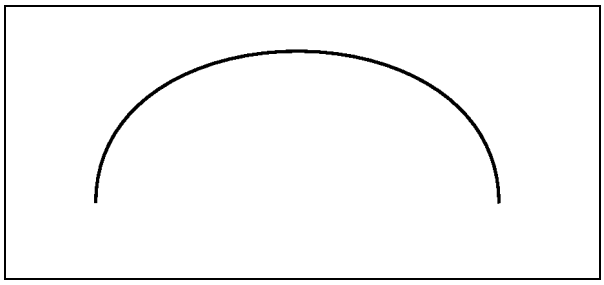

Figura 3.17 – Curva 3

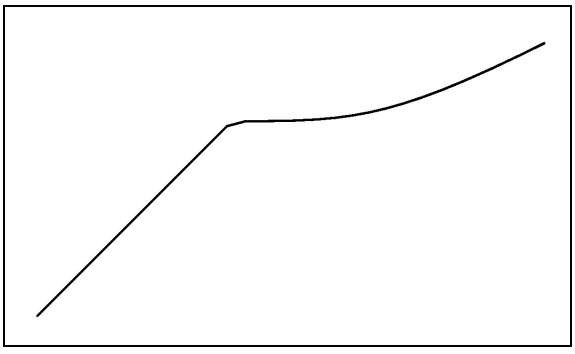

Figura 3.18 – Curva 4

Como se pode verificar pelo quadro o algoritmo baseado na avaliação da grandeza *chord error* produz um menor número de pontos da curva para todos os casos estudados. Tal facto poderá ser útil em aplicações nas quais seja relevante minimizar os dados a transmitir através da rede, como é o caso do projecto MUG [MUG1wp], que implementa uma aplicação de modelação desenhada de forma a que todos os utilizadores colaborem através de um cenário de intranet ou de Internet.

Capítulo 4 – NURBS e a tecnologia Java

Existe, actualmente, uma procura considerável de uma API que permita a representação de curvas e superfícies do tipo NURBS e que possa ser utilizada com tecnologia Java. Tal facto poderá ser facilmente comprovado através, por um lado, do número de entradas (*posts*) em fóruns de discussão especializados que questionam sobre o eventual suporte das APIs JOGL e Java 3D na representação das referidas curvas e superfícies (considerem-se, por exemplo, as referências [SUN1wp], [MAR1wp], [MAR2wp] e [MAR3wp]) e, por outro lado, pelo número considerável de projectos, mais ou menos conseguidos, que permitem a representação deste tipo de curvas e superfícies. Nenhum destes projectos implementa no entanto a representação de curvas e superfícies do tipo NURBS recortadas (ou seja, com *trimming curves*) de forma independente da plataforma. Nas secções seguintes iremos abordar alguns destes projectos com o intuito de analisar, neste capítulo, as várias opções existentes actualmente para a representação de curvas e superfícies do tipo NURBS utilizando tecnologia Java.

4.1 Biblioteca jgeom

A biblioteca jgeom [JGEOwp] revelou-se a biblioteca mais completa, disponível em código aberto, à data de início do trabalho produzido para esta dissertação. Por essa razão decidimos escolhê-la. Apresentava suporte para a representação de curvas e superfícies do tipo NURBS, e permitia também, de forma restrita, a utilização de uma única *trimming curve*. Não permitia no entanto a triangulação dos pontos de uma superfície recortada. Neste trabalho a biblioteca

jgeom foi ampliada para suportar um número arbitrário de *trimming curves* e foi implementado um algoritmo de triangulação para os pontos da superfície.

4.2 Ocnus

A biblioteca desenvolvida pela empresa Ocnus [OCNUwp] permite a representação de curvas e superfícies do tipo NURBS, mas limita-se à sua representação sem recortes, i.e., sem *trimming curves*. Esta biblioteca, não obstante permitir o cálculo dos pontos da superfície (através do algoritmo conhecido por *Oslo Algorithm*), é direccionada principalmente para a produção de código VRML. Para tal utiliza outra API proprietária da empresa *Dimension X* [DIMEwp] denominada *Liquid Reality* [LIQUep].

4.3 OpenGL for Java (GL4Java)

Ao contrário das APIs mencionadas anteriormente que estão directamente implementadas na linguagem de programação Java, independente de plataforma, a API GL4Java [GL4Jwp] utiliza as bibliotecas OpenGL específicas de cada plataforma/sistema operativo. Assim sendo, esta API disponibiliza para a linguagem Java, e através da utilização de JNI (*Java Native Interface* [JVNIwp]), todas as funcionalidades da versão 1.2 do OpenGL e da versão 1.2 da API GLU incluindo a representação de *trimmed NURBS surfaces*. Tal é viabilizado pela utilização de JNI e é, assim, possível usarem-se directamente as bibliotecas OpenGL implementadas em C fazendo um mapeamento para classes Java.

Este projecto foi, no entanto, descontinuado a partir de 7 de Setembro de 2007 e os seus autores recomendam a utilização de JOGL.

4.4 Lightweight Java Game Library (LWJGL)

Esta API ([LWJGwp]) é direccionada essencialmente para o desenvolvimento de jogos em tecnologia Java. À semelhança da API anterior (GL4Java), a LWJGL utiliza as bibliotecas específicas de cada plataforma, para disponibilizar as funcionalidades do OpenGL e de outras APIs como, por exemplo, a OpenAL (*Open Audio Library*). No entanto, não disponibiliza ainda as funcionalidades relativas a NURBS devido a dificuldades encontradas na implementação das funções de *callback* do OpenGL [LWJ1wp].

4.5 MUG Nurbs API

O projecto MUG [MUG1wp] (*Multi-User Groups for Conceptual Understanding and Prototyping*), da *Drextel University*, tem por objectivo a implementação de um sistema de suporte ao desenho conceptual de equipamentos no ramo da Engenharia Mecatrónica, num ambiente colaborativo através da utilização de uma LAN (*Local Area Network*) ou da Internet. Este projecto também disponibiliza, sob a forma de um *plug-in* ao sistema, uma API para a representação de NURBS. No entanto esta não suporta a representação de *trimmed NURBS surfaces*.

4.6 JOGL (Java Bindings para OpenGL)

O projecto JOGL [JOGLwp] tem por objectivo implementar o *Java Specification Request (JSR) 231 – Java Bindings for OpenGL* [J231wp]. Como produto deste projecto foi implementada a biblioteca conhecida pelo mesmo nome, JOGL, que é actualmente a biblioteca Java mais amplamente aceite para realizar os *bindings* para OpenGL e para a biblioteca GLU. Esta biblioteca implementa a totalidade das funcionalidades disponibilizadas pelo OpenGL 2.0 e uma parte das funcionalidades disponibilizadas pela biblioteca GLU. Está implementada praticamente na sua totalidade em Java, existindo apenas cerca de 150 linhas de código C escritas por seres humanos, as quais se destinam essencialmente a corrigir erros da implementação C do OpenGL. Todo o restante código C que efectua *bindings* para as bibliotecas OpenGL específicas de plataforma/sistema operativo é gerado automaticamente durante o processo de *build* por uma ferramenta denominada *GlugGen* – [GLUEwp]. A biblioteca JOGL está integrada com os sistemas de gestão de janelas da plataforma java, a saber, AWT [JAWTwp] e Swing.

O suporte para curvas e superfícies do tipo NURBS existe na biblioteca, mas não é possível ainda a representação de superfícies recortadas, i.e, com *trimming curves*.

4.7 Java 3D

A API Java 3D [J3DPwp] permite a criação de gráficos tridimensionais em aplicações Java. Este API baseia-se nos conceitos de Universo Virtual (*Virtual Universe*) e Grafo da Cena (*Scene Graph*). O Universo Virtual é definido por um Grafo da Cena. Por sua vez, o Grafo de Cena é construído pelo programador com, por um lado, os objectos geométricos que vão construir a cena e, por outro, com outro tipo de objectos que definem materiais, condições de iluminação, sons, animação [DAVI05] e as condições de visualização da cena (posição da câmara, tipo de projecção, etc.). A API Java 3D funciona como uma camada (*layer*) superior mais abstracta que pode usar para representação da cena (*rendering* num *layer* inferior) tanto OpenGL como, em alternativa, *Direct X*. A partir da versão 1.5 da API Java 3D já é possível a utilização de JOGL para *rendering* da cena [DAVI07].

A API Java 3D não suporta, no seu *core*, a representação de NURBS. No entanto, a biblioteca jgeom, acima mencionada, disponibiliza *bindings* para Java 3D sendo, assim, possível a representação deste tipo de curvas e superfícies utilizando essa biblioteca.

4.8 Estudo comparativo entre as tecnologias JOGL e Java 3D na representação de NURBS

No que respeita a NURBS a biblioteca JOGL suporta, desde Outubro de 2007, a representação deste tipo de curvas e superfícies mas sem a possibilidade de efectuar recortes na mesma, i.e, não suporta *trimmed NURBS surfaces*. Não é assumido de forma clara no âmbito do projecto JOGL que esta funcionalidade irá ser implementada num futuro próximo, apesar de ser largamente solicitada pela comunidade de programadores que utilizam a API.

Quanto à API Java 3D não suporta no seu *core* a representação de NURBS. No âmbito do projecto jgeom foi desenvolvida uma biblioteca que permite a representação de *trimmed NURBS surfaces*, embora com as restrições de suportar apenas a especificação de uma *trimming curve* e, também, de não representar a superfície por polígonos mas apenas por um conjunto de pontos individuais.

No trabalho desenvolvido para esta dissertação foi ampliada a biblioteca jgeom com o intuito de eliminar as restrições identificadas no parágrafo anterior.

Para concluir este capítulo gostaríamos de salientar os seguintes pontos, resultantes da pesquisa efectuada:

- a procura entre a comunidade de programadores de aplicações 3D, em tecnologia Java, por uma API que permita a representação de *trimmed NURBS surfaces* é, actualmente, bastante significativa;
- não existe ainda uma resposta completa para este problema inteiramente baseada em código Java;
- em diversos documentos [J3DFwp] e fóruns associados aos projectos JOGL e Java3D é considerada a eventualidade de numa próxima versão

das bibliotecas ser incluído suporte para a representação de NURBS. No entanto, este assunto já é discutido há algum tempo e, até ao momento, este suporte ainda não existe;

- a única biblioteca que conseguimos encontrar com suporte para *trimmed NURBS surfaces* implementada em Java foi a biblioteca jgeom.

Gostaríamos, ainda, de salientar que não obstante o facto de o projecto jgeom já existir desde 2004 [GER1wp] e de o mesmo se encontrar referenciado em inúmeras fontes (por exemplo, no livro *Killer Game Programming in Java* [DAVI07]), a biblioteca jgeom não suporta ainda, de forma total, a representação de *trimmed NURBS surfaces*. Pensamos que tal constatação, associada ao facto de existir entre a comunidade uma procura considerável de uma biblioteca que permita a representação deste tipo de curvas e superfícies, é um indicador do elevado grau de dificuldade associado à implementação de tal biblioteca.

Capítulo 5 – Considerações Finais

5.1 Resultados

Com este trabalho pretendemos, de alguma forma, facilitar a adopção da tecnologia Java na representação de curvas e superfícies do tipo NURBS disponibilizando, para esse efeito, uma biblioteca totalmente implementada em Java. Para tal, ampliámos a biblioteca jgeom com algoritmos que consideramos importantes para atingir este objectivo, em particular um algoritmo que permite a utilização de um número arbitrário de *trimming curves* bem como o algoritmo de triangulação. Outros algoritmos importantes que foram também implementados são o algoritmo para o cálculo dinâmico do valor do incremento e o algoritmo adaptativo para a subdivisão de uma curva do tipo NURBS. Para além disso o facto da implementação ser totalmente baseada em Java permite a sua utilização de forma independente de plataforma em bibliotecas, como é o caso do Java3D. A biblioteca implementada também poderá ser utilizada com JOGL dado que foram implementados os *bindings* para JOGL.

5.2 Trabalho Futuro

Como trabalho futuro gostaríamos de melhorar o algoritmo de triangulação tornando-o num algoritmo mais robusto e eficiente. Em particular, o algoritmo apresentado poderia ter um ganho em eficiência se fosse implementado um mecanismo para identificar a *trimming curve* mais próxima de cada ponto fronteira. Assim, seria possível reduzir a procura do ponto mais próximo aos pontos de apenas

essa *trimming curve* (actualmente, o algoritmo procura em todas as *trimming curves* associadas à superfície). Para além disso, gostaríamos de ter a oportunidade de procurar outras soluções, para além da solução da redução do intervalo paramétrico, para as questões identificadas na secção 3.2.3 relativas a *trimming curves* muito próximas e a uma *trimming curve* totalmente contida numa célula.

Ainda como trabalho futuro, está planeada a curto prazo uma dissertação de mestrado, que irá estudar as técnicas de interacção pessoa-computador para uma adequada manipulação interactiva dos parâmetros associados à representação de curvas e superfícies do tipo NURBS. Esse trabalho poderá agora explorar tais parâmetros, mediante aplicações que façam uso da biblioteca implementada no trabalho desenvolvido para a dissertação que aqui apresentamos. Em particular, poderá explorar a manipulação dos parâmetros associados às *trimmed NURBS surfaces*.

Com o intuito de optimizar a interacção pessoa-computador na manipulação dos parâmetros associados a uma superfície do tipo NURBS, Rogers, em [ROGE01], observa o seguinte: um utilizador de uma aplicação CAD quando manipula uma superfície do tipo B-Spline, regra geral, trabalha com um número constante de pontos de controlo, com o mesmo grau da superfície para ambas as direcções paramétricas e com um número constante de linhas isoparamétricas para a representação da mesma. Só ocasionalmente os utilizadores sentem necessidade de modificar algum destes parâmetros. Tipicamente, poderão, por exemplo, aumentar o número de pontos de controlo numa dada direcção paramétrica para obter um maior detalhe numa zona específica da superfície. Baseando-se no princípio de que a manipulação dinâmica da superfície, por parte do utilizador, consiste maioritariamente na manipulação dos pontos de controlo um a um, Rogers

apresenta na página 256 do seu livro *An Introduction to NURBS* [ROGE01] dois métodos para o cálculo incremental dos pontos da superfície (por cálculo incremental entenda-se o cálculo, apenas, da zona da superfície que sofreu modificações). O primeiro método baseia-se na manipulação de apenas um dos pontos de controlo e o segundo baseia-se na manipulação de apenas um dos pesos (*weigthing factors*) associado a um qualquer ponto de controlo. Para chegar às expressões que reflectem as modificações na superfície, Rogers subtrai a equação que representa a superfície no seu estado final da equação que representa a superfície no seu estado inicial.

Cheung, Lau e Li, em [CHEU03], descrevem um algoritmo para a representação de *trimmed NURBS surfaces* deformáveis em tempo-real como, por exemplo, uma animação de uma face humana. De uma forma resumida, o algoritmo baseia-se na ideia principal de manter, para cada superfície, duas estruturas de dados. A primeira consiste no modelo matemático da superfície. A segunda consiste num modelo de polígonos que representam a superfície. À medida que a superfície vai sendo deformada o modelo de polígonos não é gerado novamente do início mas sim actualizado de forma incremental. Este algoritmo permite, também ele, uma optimização da experiência interactiva do utilizador ao manipular de forma dinâmica os parâmetros associados à superfície.

5.3 Uma restrição importante

Não queremos deixar de referir neste capítulo um problema actual no que diz respeito às *trimmed NURBS surfaces.* Este problema consiste na ocorrência de falhas (recortes indesejados) que ocorrem nas fronteiras das superfícies recortadas.

Ao juntarmos duas *trimmed NURBS surfaces* as mesmas irão apresentar algumas falhas de dimensões muito pequenas na sua junção, ou seja junto do recorte imposto pelas *trimming curves*. Este facto advém de uma consequência teórica, como exposto por Sederberg *et al.* em [SEDE08]. Tipicamente, as *trimming curves* são curvas do tipo NURBS, de grau três, definidas no espaço paramétrico. Assim sendo, a sua imagem numa superfície bi-cúbica em $\Re^3$ tem grau ≤ 18 e género zero. No entanto, de uma forma geral, a intersecção entre duas superfícies bi-cúbicas tem grau 324 e género 433. Logo, a curva resultante da intersecção entre duas superfícies bi-cúbicas pode ser apenas aproximada por uma *trimming curve* definida no espaço paramétrico. Tal facto, apesar de aparentemente inócuo dada a pequena dimensão das falhas, é suficiente para que certas aplicações deixem de ser exequíveis. Por exemplo, aplicações de análise de propriedades físicas como o volume ou a transferência de energia sob a forma de calor não funcionam correctamente se o modelo tiver falhas, por muito pequenas que sejam as suas dimensões. Sederberg *et al.*, descrevem uma solução para este problema baseada na substituição da representação de *trimmed NURBS* por *untrimmed T-Splines*.

5.4 Nota de fecho

Num trabalho desta natureza ficamos sempre com a sensação de que haveria muito mais por fazer. No entanto, penso que demos um pequeno passo para alargar à tecnologia Java, uma ferramenta que há muito existe no mundo da computação gráfica em outras linguagens de programação como, por exemplo, o C++. Esperamos, sinceramente, que este caminho possa ser continuado no futuro.

Bibliografia

[ANDE03] Andersson, F., Bezier and B-Spline Technology, Master's Thesis, Umea Universitet, 2003
http://www.cs.umu.se/education/examina/Rapporter/461.pdf

[BEZI98] Bézier, P., A view of the CAD-CAM Development Period, IEEE Annals of the History of Computing, Vol. 20, No. 2, 1998

[BROWwp] Computer Graphics Java Applets, http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/repositoryApplets.html, Brown University

[CAS1wp] Casciola, G., Harley Design Process using NURBS http://www.dm.unibo.it/~casciola/php/public_html/projects/2004/tutorials/Harley/index.html

[CASCwp] Casciola, G., Vários objectos físicos modelados com superfícies do tipo NURBS, University of Bologna
http://www.dm.unibo.it/~casciola/html/xcmodel.html

[CGAFwp] O'Rourke, J., comp.graphics.algorithms FAQ, Computer Graphics Algorithms Frequently Asked Questions
http://www.faqs.org/faqs/graphics/algorithms-faq/

[CHEN06] Chen, J. X., Wegman, E. J., Foundations of 3D Graphics Programming Using JOGL and Java 3D, Springer, 2006

[CHEU03] Cheung, G., Lau, R., Li. F., Incremental Rendering of Deformable Trimmed NURBS Surfaces, Proceedings of the ACM symposium on Virtual Reality software technology, pp. 48-55, 2003

[DAVI05] Davison, A., Killer Game Programming in Java 1st Ed, Ch. 14, O'Reilly, 2005

[DAVI07] Davison, A. Pro Java 6 3D Game Development: Java 3D, JOGL, Jinput and JOAL APIs (Expert's Voice in Java), pp. 3, Kindle Edition, 2007

[DIMEwp] Dimension X's Liquid Reality http://www.microsoft.com/presspass/press/1997/may97/dmsnxpr.mspx

[ECKEwp] Eckert, R., Web Page, http://www.cs.binghamton.edu/~reckert/460/fillalgs.htm, State University of New York, Binghamton Algorítmo de preenchimento de polígonos (scanline fill polygon).

[EDINwp] Edinformatics.com – The Interactive Library, http://www.edinformatics.com/il/il_math.htm *Site* com applets interactivos de diferentes áreas utilizado para a pesquisa de aplicações sobre raios de curvatura, no momento da implementação dos algoritmos relacionados com raios de curvatura

[FOLE96] Foley, J., van Dam A., Feiner, S., Hughes, J., Computer Graphics, Principles and Practice 2nd Ed, Addison-Wesley, 1996

[FORR98] Forrest, R., The Emergence of NURBS, IEEE Annals of the History of Computing, Vol. 20, No. 2, 1998

[GER1wp] Gerber, S., Autor do projecto jgeom – Curriculum Vitae Neste documento é mencionada a data de início do projecto jgeom http://www.cs.utah.edu/~sgerber/cv_en.pdf

[GERBwp] Gerber, S., Projects Page, School of Computing, University of Utah, http://www.cs.utah.edu/~sgerber/projects/index.htm
Página pessoal do autor de biblioteca jgeom com informação adicional sobre o projecto

[GL41wp] OpenGL for Java Open Source Project http://gl4java.sourceforge.net/

[GL4Jwp] OpenGL for Java Open Source Project http://sourceforge.net/projects/gl4java

[GLUEwp] GlueGen – ferramenta para a geração automática de código JNI para a execução de código C a partir de código Java https://gluegen.dev.java.net/

[GNULwp] GNU General Public Licenses http://www.gnu.org/copyleft/gpl.html
Informação sobre as licenças de software relevantes para a biblioteca jgeom

[HILL01] Hill, F. S., Computer Graphics Using Open GL 2nd Ed., Prentice Hall, 2001

[J231wp] JSR-231 Java Binding for the OpenGL API http://jcp.org/en/jsr/detail?id=231

[J3DFwp] JavaOne 2007, Java 3D BOF, Future Possibilities https://java3d.dev.java.net/j3dbof07/j3dbof07.pdf

[J3DPwp] Java 3D Project https://java3d.dev.java.net/
Página do projecto Java3D

[J3DTwp] Java 3D Tutorial

http://java.sun.com/developer/onlineTraining/java3d/index.html

Tutorial de Java3D com informação técnica sobre esta tecnologia

[JAWTwp] Java Abstract Window Toolkit (AWT)

http://java.sun.com/javase/6/docs/technotes/guides/awt/

[JGEOwp] jgeom Project https://jgeom.dev.java.net/

Página do projecto jgeom

[JGFRwp] JavaGaming.org Forums Java

http://www.javagaming.org/index.php/topic,15594.msg137528/

topicseen.html#msg137528

Página web com informação sobre a release do jogl onde foi

incluído o código migrado por Tomas Hrasky

[JVNIwp] Java Native Interface

http://java.sun.com/j2se/1.3/docs/guide/jni/

[JOGLwp] JOGL Project https://jogl.dev.java.net/

[KLEI97] Klein, R., Construction of the Constrained Delaunay Triangulation of a

Polygonal Domain, CAD Systems Development: Tools and

Methods, pp. 313-326, 1997

[KRAAwp] Kraus, M., Applets, http://wwwvis.informatik.uni-stuttgart.de/

~kraus/LiveGraphics3D/cagd/index.html, Universität Stuttgart

[KRAUwp] Kraus, M., Homepage, http://www.vis.uni-stuttgart.de/~kraus/,

Universität Stuttgart

[KUMA05] Kumar, S., Manocha, D.,

Efficient rendering of trimmed NURBS Surface,

Computer-Aided Design, vol. 27, no. 7, pp. 509-521, 1995

[LAIJ07] Lai, J., Lin, K., Tseng, S., Ueng, W., On the development of a parametric interpolator with confined chord error, feedrate, acceleration and jerk, International journal of advanced manufacturing technology, DOI 10.1007/s00170-007-0954-7, 2007

[LG3Dwp] Kraus, M., LiveGraphics3D, http://www.vis.uni-stuttgart.de/~kraus/LiveGraphics3D/, Universität Stuttgart

[LIQUwp] Liquid Reality VRML Java API http://www.vrmlsite.com/sep96/spotlight/lr/lr.html

[LIUX05] Lui, X., Ahmad, F., Yamazaki, K., Mori, M., Adaptive interpolation scheme for NURBS curves with the integration of machining dynamics, International Journal of Machine Tools & Manufacture, Vol. 45, no. 4-5, pp. 433-444, 2005

[LUKE93] Luken, W. L., Cheng, F., Rendering trimmed NURB surfaces, Computer Science Research Report 18669(81711), IBM Research Division, 1993

[LUKE96] Luken, W. L., Tessellation of trimmed NURB surfaces, Computer Aided Geometric Design, vol. 13, no. 2, pp. 163-177, 1996

[LWJ1wp] About LWJGL support to Nurbs http://lwjgl.org/forum/index.php/topic,1695.0.html

[LWJGwp] Lightweight Java Game Library (LWJGL) http://www.lwjgl.org/

[MAR1wp] Mark Mail Forum Post Re: Annoucement: Java 3D plans http://markmail.org/search/?q=java%20nurbs#query:java%20nurbs+page:1+mid:ibte2vifcuiuxg3g+state:results

[MAR2wp] Mark Mail Forum Post Support for Nurbs/Splines
http://markmail.org/search/?q=java%20nurbs#query:java%20nurbs+page:1+mid:qhpawmah3nrd3fuz+state:results

[MAR3wp] Mark Mail Forum Post net.java.dev.java3d.interest
http://markmail.org/search/?q=java%20nurbs#query:java%20nurbs+page:1+mid:3o2uuyg4gwujpajv+state:results

[MART85] Martinho, E. J. C., Oliveira, J., Fortes, M. A., Matemática para o estudo da Física, Fundação Calouste Gulbenkian, 1985

[MUG1wp] MUG Multi-User Groups for Conceptual Understanding and Prototyping
http://gicl.cs.drexel.edu/software/MUG/desdoc/

[NGCKwp] Ng, C.K., Learn Physics using Java,
http://www.ngsir.netfirms.com/englishVersion.htm

[OCNUwp] Ocnus Nurbs Curves & Surfaces
http://www.ocnus.com/NURBS/index.html

[OPGLwp] OpenGL, http://www.opengl.org

[OROU98] O'Rourke, J., Computational Geometry in C,
Cambridge University Press, 1998

[PETE99] Peter, I., Gumhold, S., Teaching Computer Graphics with Java 3D, University of Tübigen, Germany, 1999

[PIE1wp] Piegl, L. A., Research web page, http://www.cse.usf.edu/~lap/

[PIE2wp] Piegl, L. A., http://www.piegl.com/

[PIE293] Piegl, L.A., Richard, A.M., Algorithm and data structure for triangulating multiply connected polygonal domains,
Computer & Graphics, vol. 17, no. 5, pp 563-574, 1993

[PIEG93] Piegl, L.A., Fang, T.P., Delaunay Triangulation Using a Uniform Grid, IEEE Computer Graphics & Applications, vol. 13, no. 3, pp 36-47, 1993

[PIEG95] Piegl, L., Richard, A., Tessellating trimmed NURBS surfaces, Computer-Aided Design, vol. 27, no. 1, pp. 15-26, 1995

[PIEG97] Piegl, L., Tiller, W., The NURBS Book, Springer, 1997

[PIEG98] Piegl, L., Tiller, W., Geometry-based triangulation of trimmed NURBS surfaces, Computer-Aided Design, vol. 30, no. 1, pp. 11-28, 1998

[PRATwp] Prat, V., NURBS Curves and Surfaces Tutorial, http://www.flipcode.com/archives/NURBS_Curves_Surfaces.shtml, 2001

[PREP85] Preparata, F., Shamos, M., Computational Geometry, An Introduction, Springer-Verlag, 1985

[RAMA02] Ramakrishnan, C., An Introduction to NURBS and OpenGL, University of California, Santa Barbara, 2002

[ROCK89] Rockwood, A., Heaton, K., Davis, T., Real-Time Rendering of Trimmed Surfaces, Computer Graphics, Vol. 23, no. 3, pp. 107-116, 1989

[ROGE01] Rogers, David F, An Introduction to NURBS With Historical Perspective, Morgan Kaufmann Publishers, 2001

[ROHA97] Java API para a representação de curvas e superfícies do tipo NURBS (não suporta *trimming curves*) http://www.ocnus.com/NURBS/index.html

[SEDE08] Sederberg, T., Finnigan, G., Li, X., Lin, H., Ipson, H., Watertight Trimmed NURBS, ACM SIGGRAPH 2008, 79, 2008

[SILV01] Silva, A., Videira, C., UML Metodologias e Ferramentas CASE, 1ª ed., Centro Atlântico, 2001

[SHNEwp] Shneiderman, Ben, http://www.cs.umd.edu/~ben/, University of Maryland

[SUN1wp] How about the Open Inventor for Java? http://forums.sun.com/thread.jspa?messageID=2583076

[SUTHwp] Sutherland, M, fun@learning.physics, http://www.scar.utoronto.ca/~pat/fun/applets.html

[YAJOwp] YAJOGLB Yet Another Java OpenGL Binding http://home.earthlink.net/~rzeh/YAJOGLB/doc/YAJOGLB.html

[YEHS02] Yeh, Syh-Shiuh, Hsu, Pau-Lo, Adaptive-feedrate interpolation for parametric curves with a confined chord error, Computer-Aided Design, Vol. 34, no. 3, pp. 229-237, 2002

Apêndices

I – Algoritmo para o cálculo das derivadas de ordem n de uma curva ou superfície do tipo NURBS

O algoritmo seguinte calcula as derivadas das funções base de uma curva ou superfície do tipo NURBS até ao grau n (com $n \leq p$, sendo p o grau das funções).

Input

span -> intervalo paramétrico;

u -> valor do parâmetro

grade -> ordem da derivada

Output

ders[k][j]-> vector bidimensional que contém as derivadas de ordem k para a função $N_{span-p+j,p}$

```
public float[][] dersBasisFuns(
                  int span, float u, int grade)
    {
        float ders[][] = new float[grade + 1][degree + 1];
        //Guardar as funções de base e diferenças de nós
        float ndu[][] = new float[degree + 1][degree + 1];
        ndu[0][0] = 1.0F;
        float left[] = new float[degree + 1];
        float right[] = new float[degree + 1];
        for(int j = 1; j <= degree; j++)
        {
            left[j] = u - knots[(span + 1) - j];
            right[j] = knots[span + j] - u;
            float saved = 0.0F;
            for(int r = 0; r < j; r++)
            {
```

```
            //triângulo inferior
            ndu[j][r] = right[r + 1] + left[j - r];
            float temp = ndu[r][j - 1] / ndu[j][r];
            //triângulo superior
            ndu[r][j] = saved + right[r + 1] * temp;
            saved = left[j - r] * temp;
        }

        ndu[j][j] = saved;
    }

    //devolver as funções de base no índice 0
    for(int j = 0; j <= degree; j++)
        ders[0][j] = ndu[j][degree];

    //Calcular as derivas
    int r;
    for(r = 0; r <= degree; r++)
    {
        int s1 = 0;
        int s2 = 1;
        //guarda (alternadamente) as duas últimas linhas calculadas
        //a(k,j) e a(k-1,j)
        float a[][] = new float[2][degree + 1];
        a[0][0] = 1.0F;
        //loop para calcular a derivada de ordem k
        for(int k = 1; k <= grade; k++)
        {
            float d = 0.0F;
            int rk = r - k;
            int pk = degree - k;
            if(r >= k)
            {
                a[s2][0] = a[s1][0] / ndu[pk + 1][rk];
                d = a[s2][0] * ndu[rk][pk];
```

```
            }
            int j1;
            if(rk >= -1)
                j1 = 1;
            else
                j1 = -rk;

            int j2;
            if((r - 1) <= pk)
                j2 = k - 1;
            else
                j2 = degree - r;

            int j;
            for(j = j1; j <= j2; j++)
            {
                a[s2][j] =
                    (a[s1][j] - a[s1][j - 1]) / ndu[pk + 1][rk + j];
                d += a[s2][j] * ndu[rk + j][pk];
            }
            if(r <= pk)
            {
                a[s2][k] = -a[s1][k - 1] / ndu[pk + 1][r];
                d += a[s2][k] * ndu[r][pk];
            }
            ders[k][r] = d;
            j = s1;
            s1 = s2;
            s2 = j;
        }
    }
    //Multiplicar pelos factores correctos
    r = degree;
    for(int k = 1; k <= grade; k++)
    {
```

```
            for(int j = 0; j <= degree; j++)
                ders[k][j] *= r;
            r *= degree - k;
        }
        return ders;
    }
```