Modelagem geológica por simplóides de Bézier

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Lucas B. Freitas e aprovada pela Banca Examinadora.

Campinas, 17 de dezembro de 2010.

Jorge Stolfi (Orientador)

Martin Tygel (Co-orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Freitas, Lucas Batista
F884m Modelagem geológica por simplóides de Bézier/Lucas Batista Freitas-- Campinas, [S.P. : s.n.], 2010.

Orientador : Jorge Stolfi; Martin Tygel.
Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Spline, Teoria do. 2.Gráfica por computador. 3.Projeto assistido por computador. I. Stolfi, Jorge. II.Tygel, M.. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Título em inglês: Geological modeling using Bézier simploids

Palavras-chave em inglês (Keywords): 1. Spline theory. 2. Computer graphics. 3. Computer-aided design.

Área de concentração: Computação gráfica

Titulação: Doutor em Ciência da Computação

Banca examinadora: Prof. Dr. Jorge Stolfi (IC – UNICAMP)
Profa. Dra. Maria Cristina de Castro Cunha (IMECC – UNICAMP)
Prof. Dr. Philippe Remy Bernard Devloo (FEC – UNICAMP)
Prof. Dr. Eduardo Filpo Ferreira da Silva (PETROBRÁS)
Prof. Dr. Jessé Carvalho Costa (Depto. De Geofisica - UFPA)

Data da defesa: 17/12/2010

Programa de Pós-Graduação: Doutorado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 17 de dezembro de 2010, pela Banca examinadora composta pelos Professores Doutores:

Profª. Drª. Maria Cristina de Castro Cunha
IMECC / UNICAMP

Prof. Dr. Philippe Remy Bernard Devloo
FEC / UNICAMP

Prof. Dr. Eduardo Filpo Ferreira da Silva
PETROBRAS

Prof. Dr. Jessé Carvalho Costa
Departamento de Geofísica / UFPA

Prof. Dr. Jorge Stolfi
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Modelagem geológica por simplóides de Bézier

Lucas B. Freitas[1]

Dezembro de 2010

Banca Examinadora:

- Jorge Stolfi (Orientador)
- Dr. Eduardo Filpo Ferreira da Silva
  Petrobras
- Prof. Dr. Jessé Carvalho Costa
  CPGf – UFPa
- Profa. Dra. Maria Cristina Cunha
  IMECC – Unicamp
- Prof. Dr. Philippe Devloo
  FEC – Unicamp
- Prof. Dr. Luiz Carlos Pacheco Velho
  IMPA (Suplente)
- Profa. Dra. Anamaria Gomide
  IC – Unicamp (Suplente)
- Prof. Dr. Hélio Pedrini
  IC – Unicamp (Suplente)

[1]Suporte financeiro de: Bolsa da FAPESP (processo 2006/50344-4) 2006–2008.

Resumo

A exploração e monitoramento de um reservatório de petróleo ou gás natural exige conhecimento bastante detalhado das estruturas geológicas da região de interesse. A representação matemática e computacional desse conhecimento é um *modelo geofísico*.

Nesta tese descrevemos um sistema geral para modelagem geofísica baseado em elementos finitos polinomiais de graus arbitrários. Adotamos uma abordagem comum na indústria, em que a geometria e as propriedades das formações geológicas são representadas por funções definidas por partes, ou *splines*, que consistem da justaposição de tais elementos. Neste contexto, apresentamos contribuições teóricas e computacionais.

A principal contribuição teórica é uma teoria unificada dos elementos simploidais de Bézier, que incluem os tipos de elementos finitos mais comuns na modelagem por malhas—tais como arcos de Bézier, retalhos de Bézier triangulares e retangulares, blocos de Bézier tetraédricos, prismáticos e hexaédricos, e suas generalizações para dimensões arbitrárias, com graus independentes em cada eixo e cada componente. Como parte testa teoria, desenvolvemos fórmulas genéricas explícitas para conversão entre estes vários tipos de blocos, bem como diferenciação, reparametrização afim e elevação de grau.

As contribuições computacionais desta tese incluem a implementação dessa teoria na forma de uma biblioteca (BezEl) que permite a representação e manipulação eficiente de malhas de elementos de Bézier simplodais com dimensões e graus arbitrários. Outra contribuição original desta tese é uma metodologia para realizar o traçado eficiente de raios em malhas de elementos simploidais.

Abstract

The exploration and monitoring of a hydrocarbon reservoir demand a very detailed knowledge about the geological structures of the target area. The mathematical and computation representation of this knowledge is a *geophysical model*.

In this thesis, we describe a general system for geophysical modeling based on polynomial finite elements of arbitrary degree. We adopted an approach that is popular in industry, whereby both the geometry and the physical properties of the geological formations are represented by piecewise-defined functions, or *splines*, that are obtained by the assembly of many such elements. In this context, we present both theoretical and computational contributions.

The main theoretical contribution is a unified theory of *simploidal Bézier elements*, which include the element types most common in mesh based modeling – such as Bézier arcs, triangular and rectangular Bézier patches, tetrahedral, prismatic and hexahedral Bézier blocks, and their generalizations to arbitrary dimensions with independent degrees on each axis and each component. As part of this theory, we developed general explicit formulas for the conversion between these various block types, as well as differentiation, affine reparametrization and degree raising.

The computational contributions of this thesis include the implementation of this theory as a library (BezEl) that allows efficient representation and manipulation of meshes of simploidal Bézier elements with arbitrary dimension and degree. Another original contribution of this thesis is a methodology for performing efficient ray tracing in meshes of such simploidal elements.

Agradecimentos

Antes de tudo, agradeço a Deus por ter me concedido esta graça. Agradeço também meus orientadores, o prof. Jorge Stolfi e o prof. Martin Tygel, por terem me guiado nesta difícil trajetória e a FAPESP, ao Instituto de Computação e ao Laboratório de Geofísica Computacional pelo suporte financeiro e pela infra-estrutura que me foi disponibilizada.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A exploração e monitoramento de um reservatório de petróleo ou gás natural exige conhecimento bastante detalhado das estruturas geológicas da região, num volume que pode cobrir centenas de quilômetros quadrados de área e vários quilômetros de profundidade. Essas informações são fundamentais tanto para a localização de novas jazidas petrolíferas, quanto para o melhor aproveitamento das jazidas existentes.

1.1 Modelagem Geofísica

Esses parâmetros incluem a densidade, velocidade de propagação de ondas sísmicas, porosidade, conteúdo de óleo, água e gás, etc. Um *sistema de modelagem geofísica* é uma coleção de software para criação, manipulação e uso de tais modelos.

Nesta tese descrevemos um sistema geral para modelagem geofísica. Como é usual na maioria de tais sistemas, a geometria e as propriedades das formações geológicas são representadas por *splines*, as quais consistem da justaposição de diversos blocos de geometria simples deformados por funções polinomiais. Estes incluem *blocos simpliciais* (tetraedros ou triângulos), os *blocos tensoriais* (hexaedros ou retângulos) e os *blocos simploidais* que são uma generalização dos dois anteriores incluindo, por exemplo, prismas de base triangular.

O sistema desenvolvido nesta tese permite a utilização de blocos de todos estes tipos em um mesmo modelo. Neste trabalho, desenvolvemos uma teoria matemática geral para blocos simploidais, que trata blocos diferentes domínios, graus e dimensões de maneira uniforme. Esta análise teórica inclui fórmulas gerais explícitas para elevação de grau, diferenciação e conversão entre os diversos tipos de splines simploidais.

Para validação da teoria, implementamos uma biblioteca na linguagem C++, que denominamos BezEl, que permite construir e manipular modelos geológicos formados por blocos simploidais gerais.

Uma novidade desta biblioteca é um mecanismo genérico para implementar restrições entre os parâmetros dos blocos, por exemplo para restringir a geometria de um bloco ou para impor continuidade entre blocos adjacentes. Este mecanismo é geral o bastante para permitir colagens *não-conformes*, onde uma única face de um bloco corresponde à união de duas ou mais faces de blocos adjacentes

Além de geral, a implementação de cada bloco permite a representação econômica de blocos com geometria restrita, por exemplo, quando vários blocos similares são usados para formar uma malha uniforme.

Embora a modelagem geofísica tenha sido a motivação original e orientadora deste trabalho, as ferramentas são potencialmente úteis em outras áreas como engenharia civil, mecânica, hidráulica, aeronáutica, metereologia, oceanografia, astrofísica, animação, etc.

No próximo capítulo, descrevemos as principais abordagens para representação de modelos geológicos. O restante da tese consiste de duas partes: a formulação matemática e a validação computacional. Na primeira parte, descrevemos os polinômios de Bernstein (capítulo 3), os elementos de Bézier tensoriais (capítulo 5), simpliciais (capítulo 6) e a conversão entre eles (capítulo 7). Em seguida, formalizamos os elementos simploidais (capítulo 8) e sua reparametrização afim (capítulo 10).

Na segunda parte, descrevemos os aspectos gerais da biblioteca BezEl (capítulo 11), detalhamos o uso de restrições nesta biblioteca (capítulo 12), e descrevemos um exemplo de utilização da mesma, consistindo de um editor gráfico de modelos geológicos bidimensionais (capítulo 14). Finalmente, descrevemos uma metodologia para traçamento de raios eficiente em malhas de elementos simploidais (capítulo 15) que implementamos como um protótipo em Java Applet.

Capítulo 2

Modelos geológicos

Para muitos fins, a geologia de uma região pode ser modelada adequadamente por uma coleção de *fácies* — regiões tridimensionais, dentro das quais as propriedades do meio variam suavemente — separadas por *interfaces* — superfícies onde as propriedades do meio apresentam alguma descontinuidade.

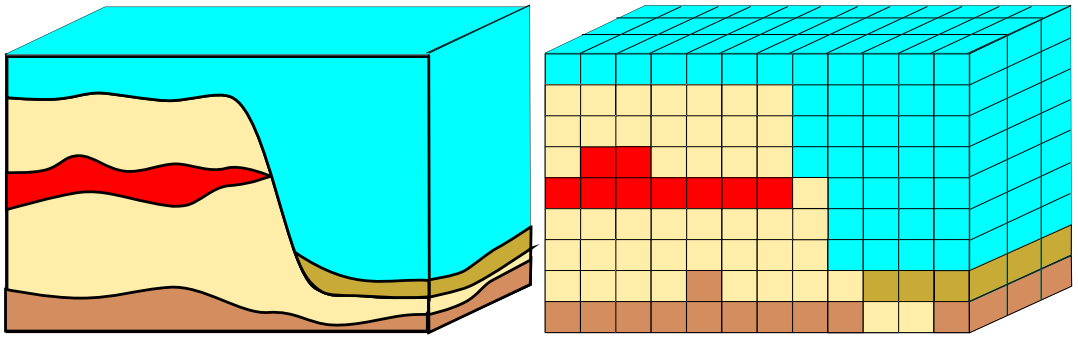
Um modelo geofísico deve capturar os aspectos relevantes tanto da geometria das interfaces quanto da litologia (composição e propriedades físicas) das fácies. Para simulação sísmica por traçado de raios, por exemplo, o sistema de modelagem deve ser capaz de representar interfaces como superfícies suaves (C^1 ou C^2) e deve também ser capaz de modelar variação suave de propriedades físicas dentro de cada fácie.

2.1 Principais abordagens

Há três grandes categorias de sistemas de modelagem geofísica em uso corrente: (a) sistemas baseados em grade uniforme, (b) sistemas baseados em interfaces e (c) sistemas baseados em malhas tridimensionais.

2.1.1 Modelos baseados em grade uniforme

Uma técnica clássica para modelagem geofísica é cobrir a região de interesse com uma grade densa e uniforme, armazenando as propriedades físicas de interesse em cada nó da grade. Veja a figura 2.1. Estes modelos são simples e extremamente flexíveis, mas não conseguem modelar precisamente descontinuidades abruptas e intrusões estreitas. Por esta razão, são mais utilizados em aplicações de imageamento baseado na equação da onda (*wave equation methods*), que não requerem elevada precisão na representação das propriedades físicas.



(a) (b)

Figura 2.1: (a) Ilustração esquemática da geologia de uma região e (b) modelagem da mesma por uma grade uniforme.

2.1.2 Modelos baseados em interfaces

Em sistemas baseados em interfaces geológicas, as fronteiras entre fácies são modeladas explicitamente por malhas bidimensionais. A geometria das fácies é definida indiretamente, como sendo a partição da região de interesse definida pelas interfaces. Sua grande vantagem sobre os modelos baseados em grade uniforme é a capacidade de representar descontinuidades abruptas de forma precisa.

Dentre os sistemas baseados em interfaces, os mais simples permitem apenas modelos do tipo "bolo de camadas" (*layer cake*), onde a profundidade z de cada interface é considerada uma função da posição horizontal (x, y). Esta abordagem permite modelar sucessões de camadas de formas e espessuras variáveis, mas não permite modelar geologias mais complexas, como camadas dobradas sobre si mesmas (*cavalgamentos*), falhas, intrusões, etc.

Outros sistemas suportam interfaces e fácies com geometrias mais complexas. Um exemplo é o sistema descrito por Vinje [26]. Nesse sistema, cada interface é implementada por uma malha triangular onde as posições dos nós são dados do modelo. Veja a figura 2.2. Informações adicionais sobre as interfaces (como normal e curvatura) são estimadas pelo sistema e armazenadas nos nós da malha [20]. As propriedades físicas dentro de cada fácie são modeladas por funções das coordenadas x, y, z do ponto, representadas por splines tricúbicas.

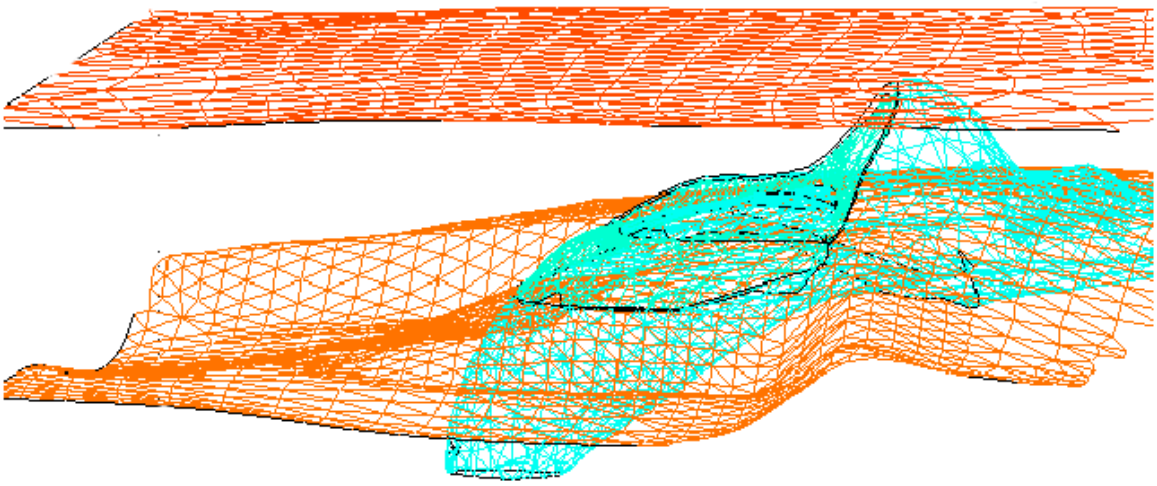


Figura 2.2: Modelo baseado em interfaces criado no sistema de modelagem do software Norsar.

2.1.3 Modelos de malhas tridimensionais

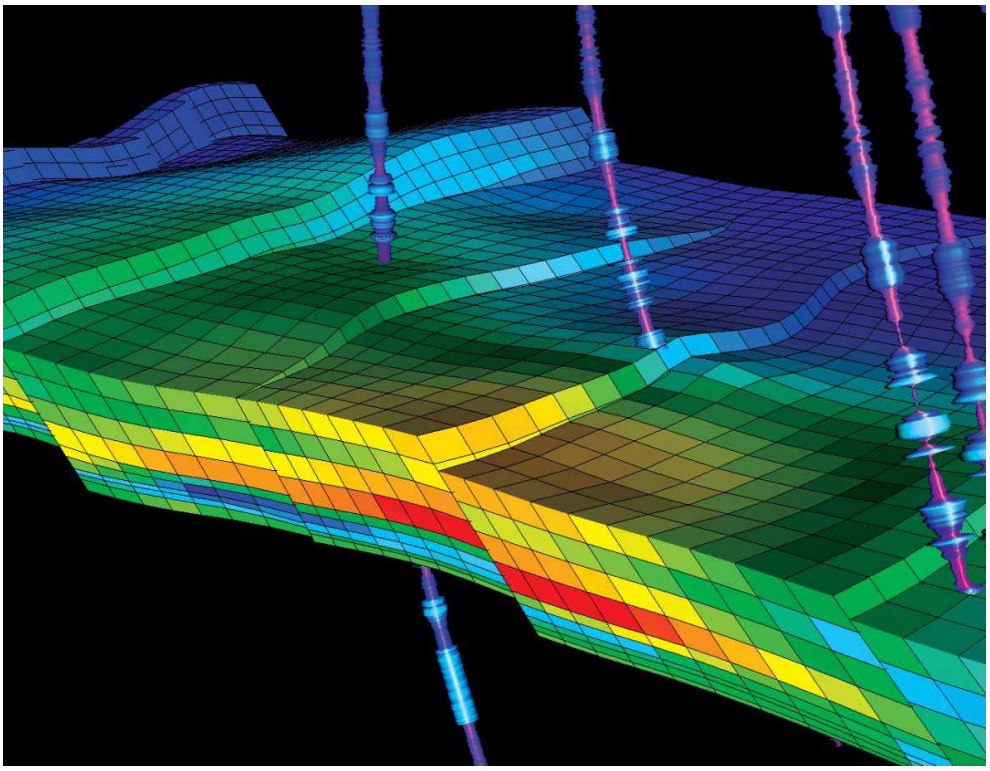
Nos sistemas baseados em malhas tridimensionais, a região de interesse é particionada *células* ou *blocos*, cuja geometria relativamente simples é descrita por um pequeno número de parâmetros (ver, por exemplo Konig [16] ou Wang [28]). Combinando um número suficiente de blocos pode-se representar geologias arbitrariamente complexas.

Para serem vantajosos, estes sistemas precisam suportar blocos com facetas não planas, que permitam modelar interfaces suaves. Uma escolha comum são blocos polinomiais, onde cada bloco é a imagem de algum sólido geométrico simples por uma função polinomial.

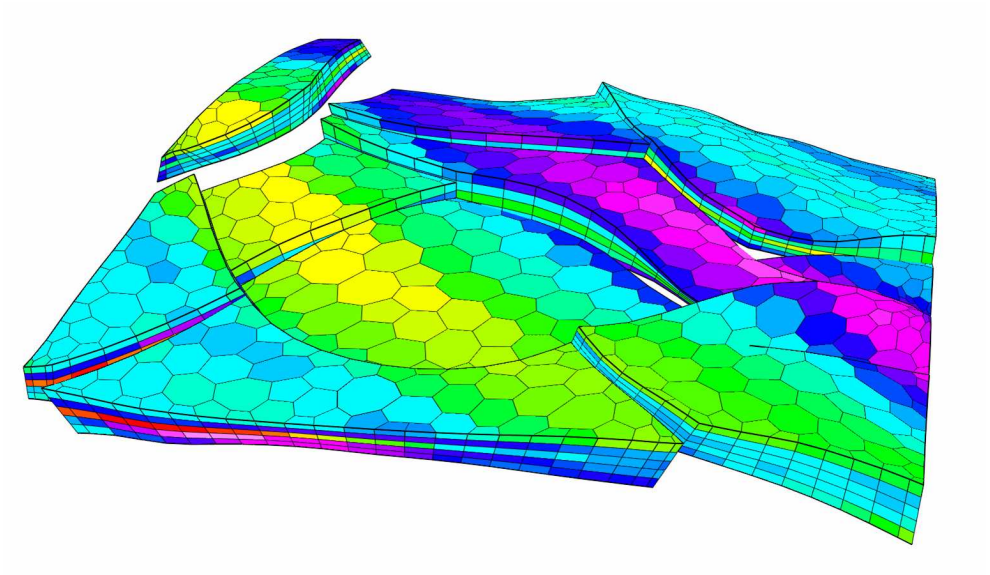
Um exemplo desta categoria é o sistema descrito por Meng e Bleinstein [21]. Esse sistema usa blocos tetraédricos de grau 3 que preenchem a região de interesse. Cada fácie é modelada por um subconjunto dos tetraedros, e cada interface é a coleção dos retalhos triangulares que separam duas fácies.

Outro exemplo é o sistema PZ de Devloo [7] cujos blocos são generalizações dos blocos de Coons [4], definidos por interpolação de pontos, curvas e superfícies paramétricas.

Dependendo dos tipos de blocos permitidos, sistemas baseados em malhas tridimensionais permitem modelar estruturas geológicas de forma arbitrária com continuidade e suavidade. Normalmente eles permitem construir modelos com milhões de blocos, mas geralmente restringem os tipos de blocos a um único tipo de elemento, simplicial ou tensorial. A figura 2.3 mostra dois exemplos de modelos geológicos criados usando softwares comerciais.



(a)



(b)

Figura 2.3: Exemplos de modelos construídos por softwares comerciais: (a) Modelo criado usando earthVision, da empresa Dynamic Grahics que usa apenas por blocos hexaédricos tricúbicos. (b) Modelo criado usando goCAD, da empresa Paradigm, formado por por elementos prismáticos de bases variadas.

2.2 Nossa abordagem

2.2.1 Blocos simploidais

Nosso sistema de modelagem é baseado em malhas tridimensionais. Cada elemento da malha é um *bloco simploidal*, que consiste em um conjunto de funções polinomiais que descrevem não apenas a geometria do bloco como também uma coleção arbitrária de propriedades físicas (densidade, elasticidade, etc) no seu interior. A figura 2.4 mostra um exemplo de modelo geológico representado utilizando nossa abordagem que consiste de mais de 645 mil blocos hexaédricos de grau 1.

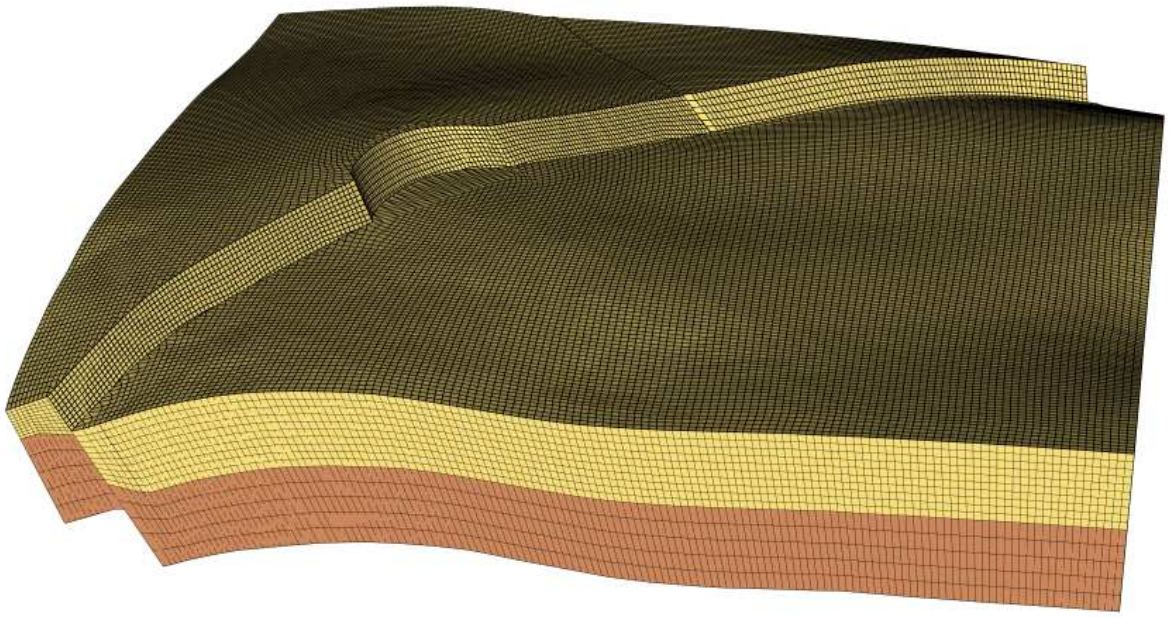


Figura 2.4: Modelo geológico representado utilizando nossa abordagem.

Tanto a geometria quanto a física são definidas em função do mesmo sistema de *coordenadas locais*, que são coordenadas baricêntricas relativas ao *domínio* do bloco (por exemplo um tetraedro regular ou um cubo unitário). A geometria é definida por três funções X,Y e Z, que fornecem as *coordenadas reais* na região de interesse de cada ponto do bloco, dadas suas coordenadas locais U. As propriedades físicas desse mesmo ponto são dadas por funções adicionais das coordenadas locais U. Neste ponto, nossa abordagem difere de vários sistemas, como o de Vinje [26], onde as propriedades físicas são modeladas como funções das coordenadas reais X,Y e Z.

Nossa abordagem tem a vantagem de que alterações na geometria das fácies acarretam

automaticamente as alterações adequadas nas propriedades. Veja figura 2.5. Por outro lado, esta abordagem é inconveniente quando é necessário determinar as propriedades físicas a partir das coordenadas X,Y e Z; como ocorre, por exemplo, na simulação sísmica. No capítulo 15, apresentaremos um método para contornar esta limitação.

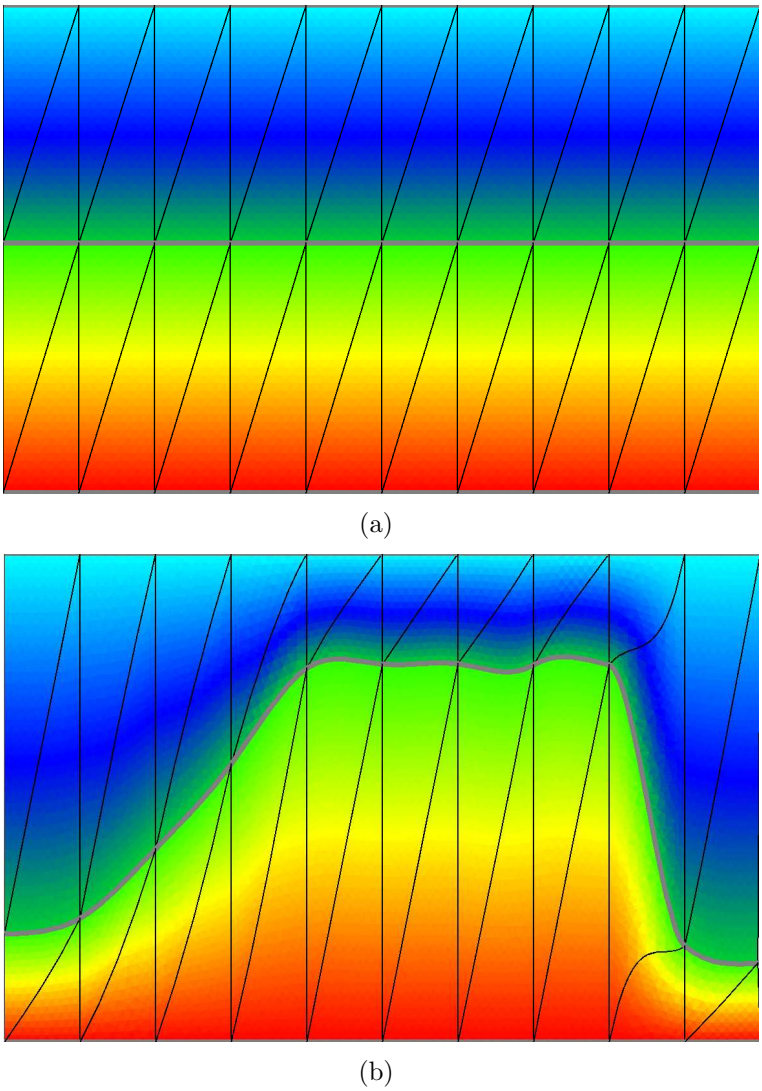
Os blocos simploidais permitidos na nossa abordagem incluem tetraedros, hexaedros e prismas triangulares com paredes curvas de grau arbitrário. Eles não incluem blocos mais complexos, como octaedros, pirâmides de base quadrada, prismas hexagonais, etc. (Esta limitação pode ser contornada utilizando dois ou mais blocos simploidais para modelar estas outras formas.) A figura 2.6 mostra um exemplo de modelo geológico que combina blocos hexaédricos de grau 3 e blocos tetraédricos de grau 1.

2.2.2 Informações topológicas

No nosso sistema, um modelo contém, além da coleção e blocos, uma *relação de adjacência* que descreve a topologia da malha e, para cada par de blocos adjacentes, um *mapeamento de colagem* que relaciona as coordenadas locais dos dois blocos na faceta comum.

2.2.3 Restrições intra- e inter-blocos

Cada modelo também inclui, opcionalmente, um conjunto de restrições sobre os parâmetros dos blocos que o compõem. Estas restrições podem incluir, por exemplo, condições de continuidade (restrições *inter-blocos*), que garantem que não haja frestas ou sobreposição entre blocos vizinhos, que as interfaces possuam derivadas contínuas, ou que as propriedades físicas variam suavemente de um bloco para outro da mesma fácie. Outros tipos de restrições (restrições *intra-blocos*) podem ser utilizadas para eliminar parâmetros desnecessários na descrição de cada bloco. Estas restrições podem ser utilizadas para garantir que as condições acima sejam satisfeitas durante a construção e edição interativa do modelo.



(a)

(b)

Figura 2.5: Ilustração da abordagem acoplada para modelagem de geometria e propriedades físicas. (a) Modelo inicial e (b) modelo depois de alterar a geometria.

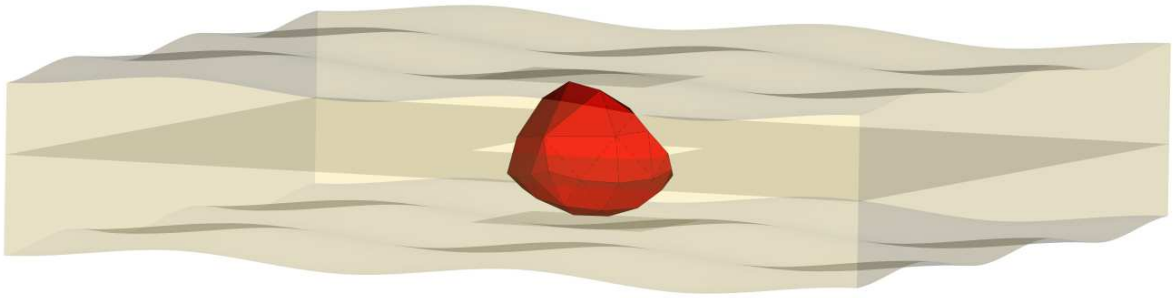


Figura 2.6: Modelo geológico construído com nossa biblioteca BezEl, usando blocos hexaédricos e tetraédricos.

Capítulo 3

Polinômios de Bernstein

Neste capítulo, revisamos conceitos relacionados aos polinômios de *Bernstein-Bézier univariados*, e definimos o conceito de *arcos de Bézier*.

3.1 A base de Bernstein univariada

Para qualquer $g \in \mathbb{N}$, a *base de Bernstein univariada de grau g* consiste dos polinômios B_i^g, para $i = 0, \cdots, g$, definidos por:

$$B_i^g(z) = \binom{g}{i} z^i \, (1-z)^{g-i} \tag{3.1}$$

para todo $z \in \mathbb{R}$ [1]. Na teoria de Bézier é conveniente introduzir a notação $\bar{z} = 1 - z$. Nesta notação, a fórmula 3.1 fica

$$B_i^g(z) = \binom{g}{i} z^i \, \bar{z}^{\, g-i} \tag{3.2}$$

Veja a figura 3.1.

Note que a fórmula (3.2) é o termo geral da fórmula de Newton para potência g do binômio $(z + \bar{z})^g$. Os polinômios de Bernstein são importantes também na teoria da probabilidade: se z é a probabilidade de um evento E, então $B_i^g(z)$ é a probabilidade de E ocorrer exatamente i vezes em g tentativas.

Dentre as propriedades dos polinômios de Bernstein, destacamos:

P1: Os polinômios são não-negativos no intervalo $[0, 1]$

P2: Cada polinômio tem apenas um máximo no intervalo $[0, 1]$, em $z = i/g$

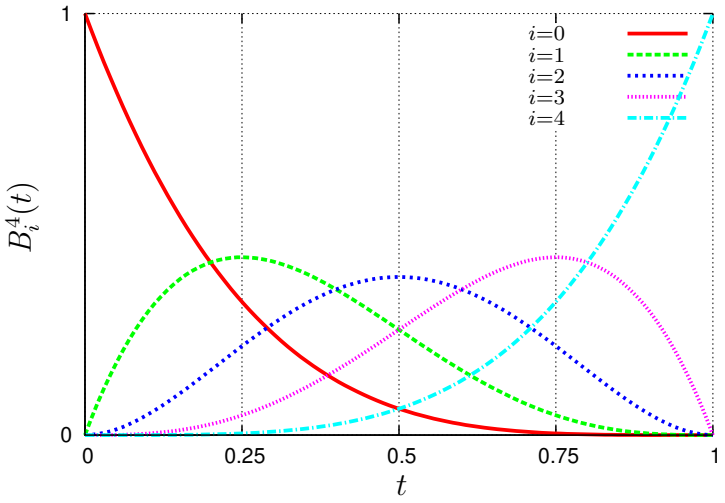


Figura 3.1: Elementos da base de Bernstein univariada de grau 4

P3: Os polinômios de grau g formam uma partição da unidade:

$$\sum_{i=0}^{g} B_i^g(z) = 1 \quad \text{para todo } z \in \mathbb{R}.$$

P4: Todos os polinômios são nulos em $z = 0$, exceto B_0^g.

P5: Todos os polinômios são nulos em $z = 1$, exceto B_g^g.

3.2 Representação de Bézier para polinômios

A *representação de Bézier* de uma função polinomial f de grau g de $\mathbb{R}$ para $\mathbb{R}$ é sua expansão em termos da base de Bernstein univariada, ou seja

$$f(z) = \sum_{i=0}^{g} b_i B_i^g(z) \tag{3.3}$$

onde $b_0, \cdots, b_g$ são números reais, chamados *ordenadas de Bézier* ou *coeficientes de Bézier* da função f. Em conseqüência da propriedade P3, o valor de $f(z)$, para qualquer $z \in [0, 1]$, está no intervalo $[b_{\min}, b_{\max}]$, onde $b_{\min}$ é o menor dos coeficientes b_i, e $b_{\max}$ o maior deles.

É comum associar cada coeficiente de Bézier b_i de B_i^g à sua *posição nominal* i/g, o valor de z onde $B_i^g(z)$ atinge seu máximo no intervalo $[0, 1]$.

3.3 Elevação de grau

A *fórmula de elevação de grau* (3.4) proposta por Trump [24] nos permite expressar um polinômio de Bernstein $B_i^g(z)$ de grau g como uma combinação linear de polinômios $B_j^h(z)$ de qualquer grau prescrito $h \geq g$.

$$B_i^g(z) \;=\; \sum_{k=i}^{i+h-g} \frac{\binom{k}{i}\binom{h-k}{g-i}}{\binom{h}{g}} B_k^h(z) \tag{3.4}$$

Portanto, para converter um polinômio f de sua representação em termos de B^g para sua representação em termos de B^h basta multiplicar seu vetor de coeficientes de Bézier $(b_0, \cdots, b_g)$ pela matriz M de $(g+1)$ linhas e $(h+1)$ colunas, onde

$$M_{ik} = \begin{cases} \binom{h}{g}^{-1}\binom{k}{i}\binom{h-k}{g-i} & \text{se } i \leq k \text{ e } i \geq k-(h-g) \\ 0 & \text{caso contrário} \end{cases}$$

3.4 Diferenciação

A derivada de ordem r do polinômio de Bernstein univariado B_i^g é dada por

$$\partial^r B_i^g(z) \;=\; \partial^r \left(\binom{g}{i} z^i (1-z)^{g-i} \right)$$

Utilizando a fórmula de Leibnitz para a derivada do produto [15], temos que

$$\partial^r B_i^g(z) \;=\; \binom{g}{i} \sum_{j=0}^{r} \binom{r}{j} \left(\partial^j z^i\right) \left(\partial^{r-j}(1-z)^{g-i}\right)$$

Note que o termo dentro do somatório se anula sempre que $j > i$ ou $r - j > g - i$, pois a j-ésima derivada de um polinômio de grau i é zero. Então,

$$\begin{aligned} \partial^r B_i^g(z) &= \frac{g!}{i!(g-i)!} \sum_{\substack{j=0 \\ j \geq r-g+i \\ j \leq i}}^{r} \binom{r}{j} \frac{i!}{(i-j)!} z^{i-j} \frac{(g-i)!}{(g-i-r+j)!} (1-z)^{g-i-r+j} (-1)^{r-j} \\ &= \frac{g!}{(g-r)!} \sum_{\substack{j=0 \\ j \geq r-g+i \\ j \leq i}}^{r} \binom{r}{j} (-1)^{r-j} \binom{g-r}{i-j} z^{i-j} (1-z)^{g-i-r+j} \\ &= \sum_{\substack{j=0 \\ j \geq r-g+i \\ j \leq i}}^{r} \frac{g!}{(g-r)!} \binom{r}{j} (-1)^{r-j} B_{i-j}^{g-r}(z) \end{aligned} \tag{3.5}$$

Como conseqüência desta fórmula, temos

$$\partial B_i^g(0) = \begin{cases} -1 & \text{se } i = 0 \\ 1 & \text{se } i = 1 \\ 0 & \text{c.c.} \end{cases} \quad \text{e} \quad \partial B_i^g(1) = \begin{cases} -1 & \text{se } i = g \\ 1 & \text{se } i = g-1 \\ 0 & \text{c.c.} \end{cases}$$

Mais geralmente,

$$\partial^r B_i^g(0) = \begin{cases} \displaystyle\sum_{\substack{j=0 \\ j \geq r-g+i \\ j \leq i}}^{r} \frac{g!}{(g-r)!} \binom{r}{j} (-1)^{r-j} & \text{se } i \leq r \\ 0 & \text{c.c.} \end{cases}$$

e

$$\partial^r B_i^g(1) = \begin{cases} \displaystyle\sum_{\substack{j=0 \\ j \geq r-g+i \\ j \leq i}}^{r} \frac{g!}{(g-r)!} \binom{r}{j} (-1)^{r-j} & \text{se } i \geq r \\ 0 & \text{c.c.} \end{cases}$$

3.5 Arco de Bézier

A representação de Bézier para polinômios é muito usada em computação gráfica para descrever curvas suaves de forma arbitrária. Este uso foi introduzido por P.Bézier na década de 60 para controle numérico e projeto de carrocerias de automóvel [1]. Formalmente, definimos uma *curva de Bézier de grau g em* $\mathbb{R}^m$ como uma função $F : \mathbb{R} \to \mathbb{R}^m$ cujas m componentes $F_0, F_1, \cdots F_{m-1}$ são polinômios de grau g, definidos em termos da representação de Bézier. A restrição da curva ao intervalo $[0, 1]$ é chamada de *arco de Bézier*.

Os $g + 1$ coeficientes de Bézier das m componentes do arco podem ser interpretados como as coordenadas de $g + 1$ pontos do $\mathbb{R}^m$, chamados de *pontos de controle* ou *pontos de Bézier* do arco. Mais especificamente, para qualquer $i = \{0, \cdots, g\}$, o ponto de controle $P_i \in \mathbb{R}^m$ é tal que sua j-ésima coordenada é o coeficiente de Bézier de índice i da componente F_j. A poligonal que liga os pontos de Bézier P_i, na ordem crescente de índice i, é chamada de *poligonal de Bézier* do arco. Veja a figura 3.2.

Note que os pontos $F(z)$ de arco de Bézier F estão totalmente contidos no fecho convexo dos pontos de controle de F. Esta propriedade dos arcos de Bézier é conseqüência direta da propriedade P3 dos polinômios de Bernstein univariados. Outra propriedade importante é que os pontos P_0 e P_g são os extremos do arco, isto é $F(0) = P_0$ e $F(1) = P_g$. Além disso, o vetor velocidade da curva no início $\partial F(0)$ é múltiplo do vetor $P_1 - P_0$, e no fim $\partial F(1)$ é um múltiplo de $P_g - P_{g-1}$. Mais geralmente, a r-ésima derivada inicial $\partial^r F(0)$ depende

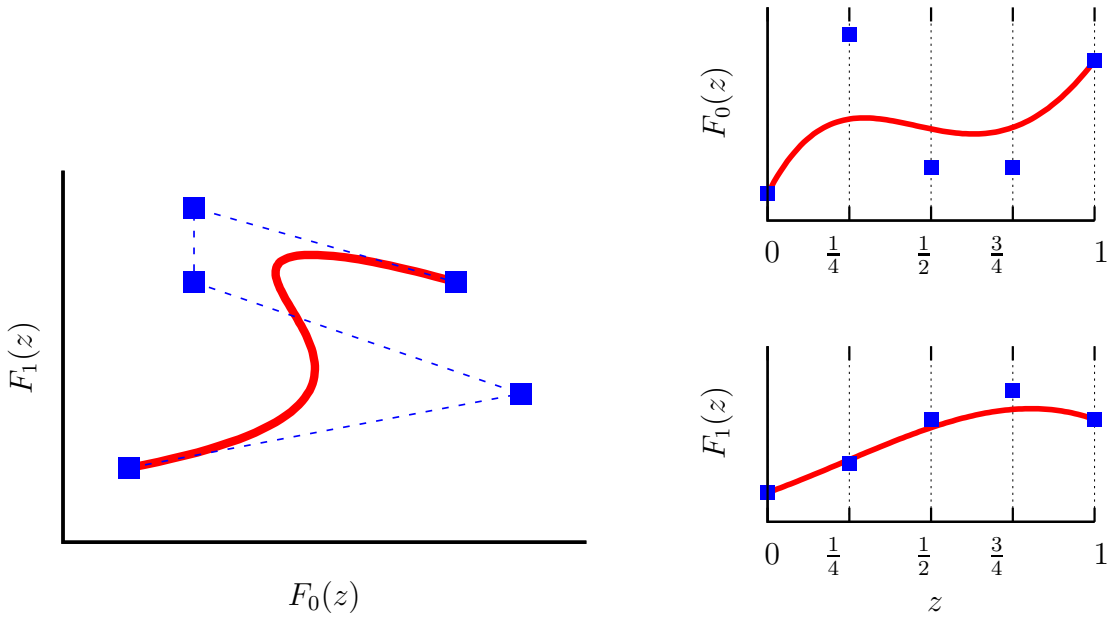


Figura 3.2: Um arco de Bézier F de grau 4 no $\mathbb{R}^2$ (esquerda em vermelho) mostrando seus pontos de controle de Bézier (em azul) e a poligonal de Bézier (tracejada); e os gráficos de suas componentes $F_0(z) = 2z^4 + 3z$ e $F_1(z) = z^4 + z^2$ (direita) mostrando os respectivos coeficientes de Bézier (pontos azuis).

apenas dos pontos $P_0, P_1, \cdots, P_r$, e a final $\partial^r F(1)$ apenas dos pontos $P_g, P_{g-1}, \cdots, P_{g-r}$. Estas propriedades decorrem das propriedades P4 e P5 da base de Bernstein, e simplificam bastante a concatenação de dois ou mais arcos de Bézier para formar um *spline de Bézier* de continuidade arbitrária.

Capítulo 4

Multi-, hiper- e ultra-índices

Neste capítulo, introduzimos uma notação especial para indexação de objetos multi-dimensionais, necessária para os capítulos seguintes.

4.1 Multi-índices

Como DeRose [6], definimos um *multi-índice* como uma tupla $\kappa = (\kappa_0, \cdots, \kappa_d)$ de números naturais. Denotamos por $\mathbb{I}$ o conjunto de todos os multi-índices, e por $\mathbb{I}_d$ o conjunto de todos os multi-índices com $d+1$ componentes, ou seja $\mathbb{N}^{d+1}$. Para quaisquer $d, n \in \mathbb{N}$, tais que $n \leq d$, quaisquer multi-índices $\kappa, \lambda \in \mathbb{I}_d$, qualquer vetor $x \in \mathbb{R}^{d+1}$, e qualquer função F de $\mathbb{R}^{d+1}$ para $\mathbb{R}$, definimos as seguintes operações:

soma e subtração:	$\kappa \pm \lambda \;=\; (\kappa_0 \pm \lambda_0, \kappa_1 \pm \lambda_1, \cdots, \kappa_d \pm \lambda_d)$
divisão:	$\kappa/\lambda \;=\; (\kappa_0/\lambda_0, \cdots, \kappa_d/\lambda_d) \in \mathbb{R}^d$
divisão por um número real:	$\kappa/g \;=\; (\kappa_0/g, \cdots, \kappa_d/g) \in \mathbb{R}^d$
comparação:	$\kappa \le \lambda \;\Leftrightarrow\; (\kappa_0 \le \lambda_0) \wedge (\kappa_1 \le \lambda_1) \wedge \cdots \wedge (\kappa_d \le \lambda_d)$
total:	$\lvert\kappa\rvert \;=\; \kappa_0 + \kappa_1 + \cdots + \kappa_d$
fatorial:	$\kappa! \;=\; \kappa_0!\kappa_1! \cdots \kappa_d!$
coeficiente multinomial:	$\binom{g}{\kappa} \;=\; \dfrac{g!}{\kappa!} \;=\; \dfrac{g!}{\kappa_0! \cdots \kappa_d!}$
multi-combinação:	$\binom{\kappa}{\lambda} \;=\; \dfrac{\kappa!}{(\kappa-\lambda)!\lambda!} = \binom{\kappa_0}{\lambda_0} \cdots \binom{\kappa_d}{\lambda_d}$
potência:	$x^\kappa \;=\; x_0^{\kappa_0} x_1^{\kappa_1} \cdots x_d^{\kappa_d}$
derivada parcial:	$\partial^\kappa F \;=\; \partial_0^{\kappa_0} \cdots \partial_d^{\kappa_d} F$

Quando um multi-índice α é usado como um expoente, como na fórmula de potenciação acima, também usamos o termo *multi-grau*.

Para quaisquer $g \in \mathbb{Z}$ e $d \in \mathbb{N}$ denotamos por $\mathbb{I}_d^g$ o conjunto dos multi-índices $\kappa \in \mathbb{I}_d$ tais que $\lvert\kappa\rvert = g$. (Observe que $\mathbb{I}_d^g = \emptyset$ se $g < 0$.) Denotamos também por $g * d$ a tupla $(g, g, \cdots, g) \in \mathbb{I}_{d-1}$, com d elementos, todos iguais a g.

O uso de multi-índices simplifica diversas fórmulas de álgebra multivariada, cálculo, equações diferenciais parciais, probabilidade, etc. Por exemplo, para quaisquer $d, g \in \mathbb{N}$ e qualquer $x = (x_0, x_1, \cdots, x_d) \in \mathbb{R}^{d+1}$, a fórmula multinomial

$$(x_0 + x_1 + \cdots + x_d)^g = \sum_{\substack{i_0,\cdot,i_d \\ i_0+\cdots+i_d=g}} g! \left(\prod_{j=0}^{d} \frac{x_j^{i_j}}{i_j!} \right)$$

pode ser escrita de forma mais sucinta como

$$(x_0 + x_1 + \cdots + x_d)^g = \sum_{\kappa \in \mathbb{I}_d^g} \frac{g!}{\kappa!} x^\kappa = \sum_{\kappa \in \mathbb{I}_d^g} \binom{g}{\kappa} x^\kappa \tag{4.1}$$

Outro exemplo é a fórmula de Leibnitz para a r-ésima derivada do produto de várias funções. Sejam $F_0(t), \cdots, F_d(t)$, $d+1$ funções de $\mathbb{R}$ para $\mathbb{R}$. Na notação usual, a fórmula é

$$\partial^r \left(\prod_{i=0}^{d} F_i \right) = \sum_{\kappa_0=0}^{r} \cdots \sum_{\substack{\kappa_d=0 \\ \kappa_0+\cdots+\kappa_d=r}}^{r} \prod_{i=0}^{d} \partial^{\kappa_i} F_i$$

Usando multi-índices, a fórmula reduz-se a

$$\partial^r \left(\prod_{i=0}^{d} F_i \right) = \sum_{\kappa \in \mathbb{I}_d^r} \binom{r}{\kappa} \prod_{i=0}^{d} \partial^{\kappa_i} F_i \tag{4.2}$$

4.2 Matrizes irregulares e hiper-índices

Para indexar coeficientes de polinômios simploidais (capítulo 8), necessitamos estender a notação de multi-índice. Para este fim, definimos uma *matriz irregular* como uma tupla $M = (M_0, M_1, \cdots, M_m)$ de tuplas reais (suas *linhas*). Note que, diferentemente de uma matriz ordinária, uma matriz irregular pode conter linhas de tamanhos diferentes; por exemplo $((1,0),(2,5,3))$ ou

$$\begin{pmatrix} 1 & 0 & \\ 2 & 5 & 3 \end{pmatrix} \tag{4.3}$$

Dado um multi-índice $\delta \in \mathbb{I}_m$, denotamos por $\mathbb{M}_\delta$ o conjunto de todas as matrizes irregulares tais que sua i-ésima linha tem $\delta_i + 1$ componentes. Por exemplo, a matriz (4.3) é um elemento de $\mathbb{M}_{(1,2)}$.

Um *hiper-índice* é uma matriz irregular de números naturais; ou seja, uma tupla $\Lambda = (\Lambda_0, \Lambda_1, \cdots, \Lambda_m)$ de multi-índices, a exemplo de (4.3) acima. Note que um multi-índice pode ser visto como um hiper-índice de uma única linha. (O termo hiper-índice já foi usado por DeRose [6], mas apenas para matrizes regulares).

Dado um multi-índice $\delta \in \mathbb{I}_m$, denotamos por $\mathbb{H}_\delta$ o conjunto de todos os hiper-índices Λ tais que cada linha Λ_i pertence a $\mathbb{I}_{\delta_i}$. Ou seja, $\mathbb{H}_\delta$ é o subconjunto de $\mathbb{M}_\delta$ cujos elementos são naturais. No caso específico de $\delta = n * (m+1)$, ou seja, quando todas as linhas Λ_i tem o mesmo número de componentes $\delta_i + 1 = n + 1$, abreviamos $\mathbb{H}_\delta$ por $\mathbb{H}_{m,n}$. Neste caso, um elemento de $\mathbb{H}_{m,n}$ é uma matriz ordinária (retangular) de números naturais com $m+1$ linhas e $n+1$ colunas.

Dados números naturais m e s, multi-índices $\kappa, \lambda, \delta \in \mathbb{I}_m$, hiper-índices $\Lambda, \Omega \in \mathbb{H}_\delta$, e duas matrizes irregulares $U, T \in \mathbb{M}_\delta$, definimos as operações

divisão por um multi-índice:	$\Lambda/\kappa \ = (\Lambda_0/\kappa_0, \ \cdots \ , \Lambda_m/\kappa_m) \in \mathbb{M}_\delta$
comparação:	$\Lambda \leq \Omega \ \Leftrightarrow \ (\Lambda_0 \leq \Omega_0) \wedge (\Lambda_1 \leq \Omega_1) \wedge \cdots \wedge (\Lambda_m \leq \Omega_m)$
soma das colunas:	$\lvert\Lambda\rvert = \ (\lvert\Lambda_0\rvert, \cdots , \lvert\Lambda_m\rvert) \in \mathbb{I}_m$
total:	$\lvert\lvert\Lambda\rvert\rvert = \ \lvert(\lvert\Lambda\rvert)\rvert \in \mathbb{N}$
fatorial:	$\Lambda! \ = \ \Lambda_0!\Lambda_1!\cdots\Lambda_m!$
hiper-combinação:	$\binom{\Lambda}{\Omega} \ = \ \dfrac{\Lambda!}{\Omega!(\Lambda-\Omega)!} \ = \ \binom{\Lambda_0}{\Omega_0}\cdots\binom{\Lambda_d}{\Omega_d}$
potência:	$T^\Lambda \ = \ \prod_{i=0}^{m} \prod_{j=0}^{\delta_i} T_{i,j}^{\Lambda_{i,j}}$

Ademais, se $\delta \in \mathbb{I}_m$ é tal que $\delta_i \geq (m-1)$, e Λ é um hiper-índice pertencente a $\mathbb{H}_\delta$ (ou seja, cada linha tem ao menos m elementos), denotamos por $\mathrm{diag}(\Lambda)$ a diagonal de Λ, ou seja, o multi-índice $(\Lambda_{00}, \Lambda_{11}, \cdots , \Lambda_{mm}) \in \mathbb{I}_m$.

Dados $m, n \in \mathbb{N}$, e multi-índices $\delta, \alpha \in \mathbb{I}_m$ denotamos por $\mathbb{H}_\delta^\alpha$ o conjunto de todos os hiper-índices Λ tais que o total $\lvert\Lambda_i\rvert$ de cada linha Λ_i é α_i. Além disso, denotamos por $\mathbb{H}_{m,n}^{\alpha,\beta}$ o conjunto de todos o hiper-índices (retangulares) $\Omega \in \mathbb{H}_{m,n}^\alpha$ tais que a soma da j-ésima coluna é β_j.

Da mesma forma que multi-índices, os hiper-índices nos permitem simplificar muitas fórmulas da álgebra multi-afim. Por exemplo, a expansão do produto de potências de multinômios

$$(x_{00} + x_{01} + x_{02})^5 (x_{10} + x_{11})^3 (x_{20} + x_{21} + x_{22} + x_{23})^2$$

utilizando a notação de hiper-índices, pode ser escrito como

$$\sum_{\Lambda \in \mathbb{H}_{(2,1,3)}^{(5,3,2)}} \binom{(5,3,2)}{\Lambda} \begin{pmatrix} x_{00} & x_{01} & x_{02} & \\ x_{10} & x_{11} & & \\ x_{20} & x_{21} & x_{22} & x_{23} \end{pmatrix}^\Lambda$$

Outros exemplos de uso de hiper-índices serão vistos no capítulo 8.

4.3 Ultra-índices

Para as fórmulas de reparametrização de polinômios simploidais (capítulo 10), necessitamos estender ainda mais a notação de índices. Para este fim, definimos um *tensor irregular* como uma tupla $T = (T_0, T_1, \cdots , T_p)$ onde cada T_i, chamado de *plano i de T*, é uma matriz irregular. Denotamos por $\mathbb{T}_p$ o conjunto de todos os tensores irregulares de $p+1$ planos.

Um *ultra-índice* é um tensor irregular de números naturais; ou seja, uma tupla $\mathbf{\Lambda} = (\mathbf{\Lambda}_0, \mathbf{\Lambda}_1, \cdots, \mathbf{\Lambda}_p)$ de hiper-índices. Observe que um hiper-índice por ser visto como um ultra-índice de um único plano.

Para todo $m, p \in \mathbb{N}$ e todo $\delta \in \mathbb{I}_m$, denotamos por $\mathbb{U}_{p,\delta}$ o conjunto de todos os ultra-índices que consistem de $p+1$ hiper-índices, todos em $\mathbb{H}_\delta$. As operações de hiper-índices podem ser estendidas para ultra-índices. Em particular, definimos a *soma dos planos* de um ultra-índice $\mathbf{\Lambda} \in \mathbb{U}_{p.\delta}$, como o hiper-índice $|\mathbf{\Lambda}| = \mathbf{\Lambda}_0 + \mathbf{\Lambda}_1 + \cdots \mathbf{\Lambda}_p$. Definimos também o *fatorial* de um ultra-índice $\mathbf{\Lambda} \in \mathbb{U}_{p.\delta}$, denotado por $\mathbf{\Lambda}!$, como o produto $\mathbf{\Lambda}_0!\mathbf{\Lambda}_1! \cdots \mathbf{\Lambda}_p!$

Dados $m, p \in \mathbb{N}$, uma multi-dimensão $\delta \in \mathbb{I}_m$ e um hiper-índice Ω, denotamos por $\mathbb{U}^{\Omega}_{p,\delta}$ o conjunto de todos os ultra-índices $\mathbf{\Lambda}$ tais que $|\mathbf{\Lambda}| = \Omega$.

Capítulo 5

Elementos de Bézier tensoriais

Neste capítulo, revisamos conceitos relacionados aos polinômios de *Bernstein-Bézier tensoriais*; e definimos os *elementos de Bézier tensoriais* utilizados por P. Bézier na década de 1960 [1].

5.1 Polinômios tensoriais

Definimos o espaço dos *polinômios tensoriais de dimensão* $d \in \mathbb{N}$ *e multi-grau* $\alpha \in \mathbb{I}_{d-1}$, denotado por $\mathcal{P}_d^\alpha$, como o espaço vetorial de todas as funções f de $\mathbb{R}^d$ para $\mathbb{R}$, tais que $f(x)$ é um polinômio de grau α_i em cada coordenada x_i do argumento, quando são fixadas todas as demais coordenadas x_j com $j \neq i$. Por exemplo, a função f de $\mathbb{R}^2$ para $\mathbb{R}$ com fórmula

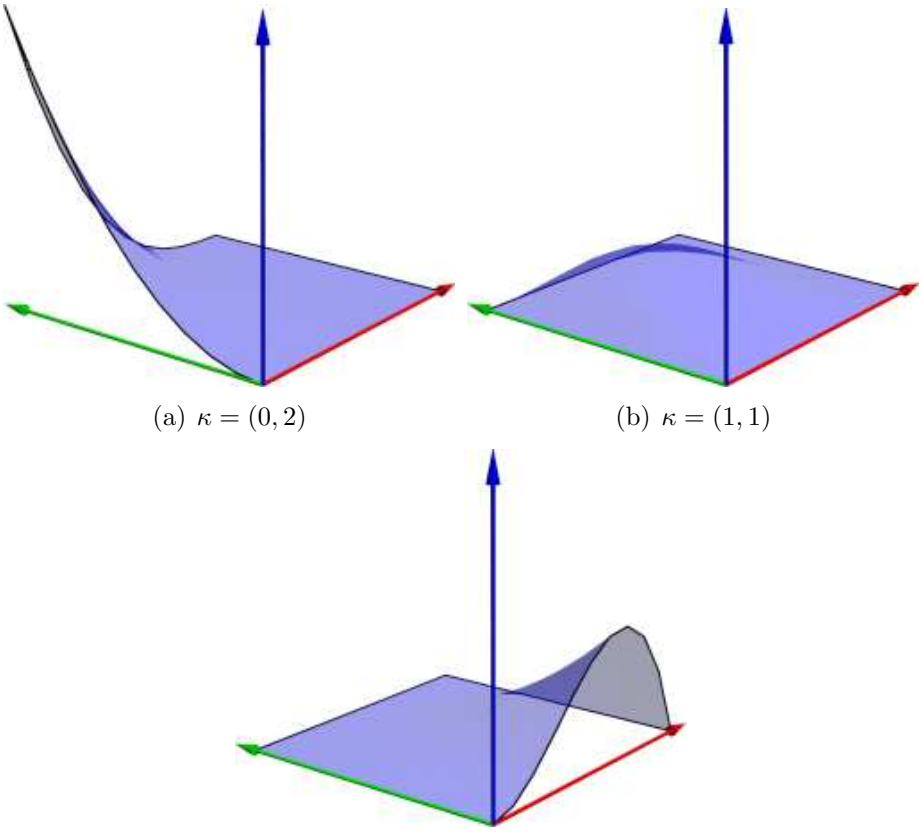
$$\begin{aligned} f(x) &= 4x_0^3 + 5x_1 - 3x_0x_1^2 + 5x_0x_1 \\ &= (4)x_0^3 + (-3x_1^2 + 5x_1)x_0 + (5x_1) \\ &= (-3x_0)x_1^2 + (5x_0 + 5)x_1 + (4x_0^3) \end{aligned}$$

é um polinômio tensorial de multi-grau $(3, 2)$. Note que $f(x)$ não pode ser escrito como o produto de dois polinômios $f'(x_0)$ e $f''(x_1)$, cada um dependendo apenas de uma coordenada.

A *base canônica* de $\mathcal{P}_d^\alpha$ é a lista de todos os monômios em d variáveis $x_0, \cdots, x_{d-1}$ que possuem grau no máximo α_i em cada variável x_i. Por exemplo, para $d = 2$ e $\alpha = (1, 2)$, a base canônica é 1, x_0, x_1, x_0x_1, x_1^2 e $x_0x_1^2$. A dimensão de $\mathcal{P}_d^\alpha$ é $\prod_{i=0}^{d-1}(\alpha_i + 1)$.

5.1.1 A base de Bernstein tensorial

Dado $d \in \mathbb{N}$ e um multi-grau $\alpha \in \mathbb{I}_{d-1}$, a *base de Bernstein tensorial* de $\mathcal{P}_d^\alpha$, consiste dos polinômios tensoriais $B_\kappa^\alpha(u)$, para todos os multi-índices $\kappa \in \mathbb{I}_{d-1}$ tais que $\kappa \leq \alpha$, cada



(a) $\kappa = (0, 2)$ (b) $\kappa = (1, 1)$

(c) $\kappa = (2, 0)$

Figura 5.1: Alguns dos 12 polinômios B^α_κ da base de Bernstein tensorial de $\mathcal{P}_2^{(3,2)}$. Os eixos vermelho, verde e azul representam x_0, x_1 e $B^{(3,2)}_\kappa(x_0, x_1)$.

um definido por

$$B^\alpha_\kappa(x) \;=\; \prod_{i=0}^{d-1} B^{\alpha_i}_{\kappa_i}(x_i) \tag{5.1}$$

para todo $x \in \mathbb{R}^d$. Veja figura 5.1.

Os polinômios de Bernstein tensoriais $B^\alpha_\kappa(u)$ compartilham varias propriedades dos polinômios de Bernstein univariados $B^g_i(z)$, incluindo:

P1′: Os polinômios B^α_κ são não-negativos no hipercubo $[0, 1]^d$

P2′: Cada polinômio tem apenas um máximo no hipercubo $[0, 1]^d$, no ponto $\kappa/\alpha \in \mathbb{R}^d$.

P3′: Os polinômios B^α_κ de mesmo multi-grau α formam uma partição da unidade:

$$\sum_{\substack{\kappa \in \mathbb{I}_{d-1} \\ \kappa \le \alpha}} B^\alpha_\kappa(x) = 1 \;\text{ para todo } x \in \mathbb{R}^d.$$

5.1.2 A representação de Bézier de polinômios tensoriais

Seja f uma função polinomial tensorial de multi-grau $\alpha \in \mathbb{I}_{d-1}$ de $\mathbb{R}^d$ para $\mathbb{R}$. A representação de Bézier de f é sua expansão em termos da base de Bernstein tensorial, a saber

$$f(x) = \sum_{\substack{\kappa \in \mathbb{I}_{d-1} \\ \kappa \leq \alpha}} b_\kappa B^\alpha_\kappa(x) \tag{5.2}$$

onde cada b_κ é um número real, o *coeficiente de Bézier* (*tensorial*) de multi-índice κ de f.

Para fins de visualização, costuma-se definir a *posição nominal* de cada coeficiente de Bézier b_κ como o ponto $\kappa/\alpha = (\kappa_0/\alpha_0, \cdots, \kappa_{d-1}/\alpha_{d-1}) \in \mathbb{R}^d$, onde esse polinômio atinge o máximo. Veja a figura 5.1.

5.1.3 Elevação de grau

Se β e α são dois multi-graus pertencentes a $\mathbb{I}_{d-1}$ com $\alpha \leq \beta$, então a aplicação repetida da fórmula de elevação de grau (3.4) garante que cada polinômio B^α_λ de multi-grau α pode ser expresso como uma combinação linear de polinômios B^β_κ de multi-grau β. A saber,

$$B^\alpha_\lambda(x) = \sum_{\substack{\kappa \in \mathbb{I}_{d-1} \\ \kappa \geq \lambda - (\beta - \alpha) \\ \kappa \leq \alpha \\ \kappa \leq \lambda}} \binom{\beta}{\alpha}^{-1} \binom{\lambda}{\kappa} \binom{\beta - \lambda}{\alpha - \kappa} B^\beta_\kappa(x)$$

para todo $x \in \mathbb{R}^d$.

5.1.4 Diferenciação

Pelas fórmulas (5.1) e (3.5), a i-ésima derivada parcial de ordem r de B^α_κ é

$$\begin{aligned} \partial^r_i B^\alpha_\kappa(x) &= \partial^r_i \left(B^{\alpha_i}_{\kappa_i}(x_i) \prod_{\substack{j=0 \\ j \neq i}}^{d-1} B^{\alpha_j}_{\kappa_j}(x_j) \right) \\ &= \left(\sum_{\substack{p=0 \\ p \geq r - \alpha_i + \kappa_i \\ p \leq \kappa_i}}^{r} \frac{\alpha_i!}{(\alpha_i - r)!} \binom{r}{p} (-1)^{r-p} B^{\alpha_i - r}_{\kappa_i - p}(x_i) \right) \prod_{\substack{j=0 \\ j \neq i}}^{d} B^{\alpha_j}_{\kappa_j}(x_j) \end{aligned}$$

Portanto, para qualquer $\lambda \in \mathbb{I}_d$, a derivada mista de ordem λ_i em cada coordenada x_i é

$$\begin{aligned}\partial^\lambda B^\alpha_\kappa(x) = \partial_0^{\lambda_0} \cdots \partial_d^{\lambda_d} B^\alpha_\kappa(x) &= \prod_{i=0}^{d-1} \left(\sum_{\substack{p=0 \\ p \geq \lambda_i - \alpha_i + \kappa_i \\ p \leq \kappa_i}}^{r} \frac{\alpha_i!}{(\alpha_i - \lambda_i)!} \binom{\lambda_i}{p} (-1)^{\lambda_i - p} B^{\alpha_i - \lambda_i}_{\kappa_i - p}(x_i) \right) \\ &= \sum_{\substack{\mu \in \mathbb{I}_d \\ \mu \geq \lambda - \alpha + \kappa \\ \mu \leq \kappa}} \frac{\alpha!}{(\alpha - \lambda)!} \binom{\lambda}{\mu} (-1)^{|\lambda| - |\mu|} B^{\alpha - \lambda}_{\kappa - \mu}(x) \qquad (5.3)\end{aligned}$$

Observe que $\partial^\lambda B^\alpha_\kappa$ é um polinômio tensorial de multi-grau $\beta = \alpha - \lambda$ se $\lambda \leq \alpha$. Caso contrário (se $\lambda_i > \alpha_i$ para algum i), a derivada é o polinômio nulo pois o somatório fica vazio.

5.1.5 Derivada direcional

Se $\mathbb{X}$ é um conjunto e $\mathcal{E}$ é uma expressão que envolve uma variável livre x, escrevemos

$$(x : \mathbb{X} \to \mathcal{E}) \qquad (5.4)$$

para a função de domínio $\mathbb{X}$ que mapeia cada $x \in \mathbb{X}$ para o valor correspondente de $\mathcal{E}$.

Seja ξ um vetor do $\mathbb{R}^d$ e f uma função de $\mathbb{R}^d$ para $\mathbb{R}$. A *derivada (direcional) de ordem r de f na direção ξ* é definida por

$$(D^r_\xi f)(x) = \left[\partial^r \left(t : \mathbb{R} \to f(x + t\xi)\right)\right](0).$$

A derivada direcional de ordem r do polinômio de Bernstein tensorial B^α_κ na direção ξ é portanto

$$(D^r_\xi B^\alpha_\kappa)(x) = \left[\partial^r \left(t : \mathbb{R} \to \prod_{i=0}^{d-1} B^{\alpha_i}_{\kappa_i}(x_i + t\xi_i)\right)\right](0)$$

Aplicando a fórmula de Leibniz (4.2), temos

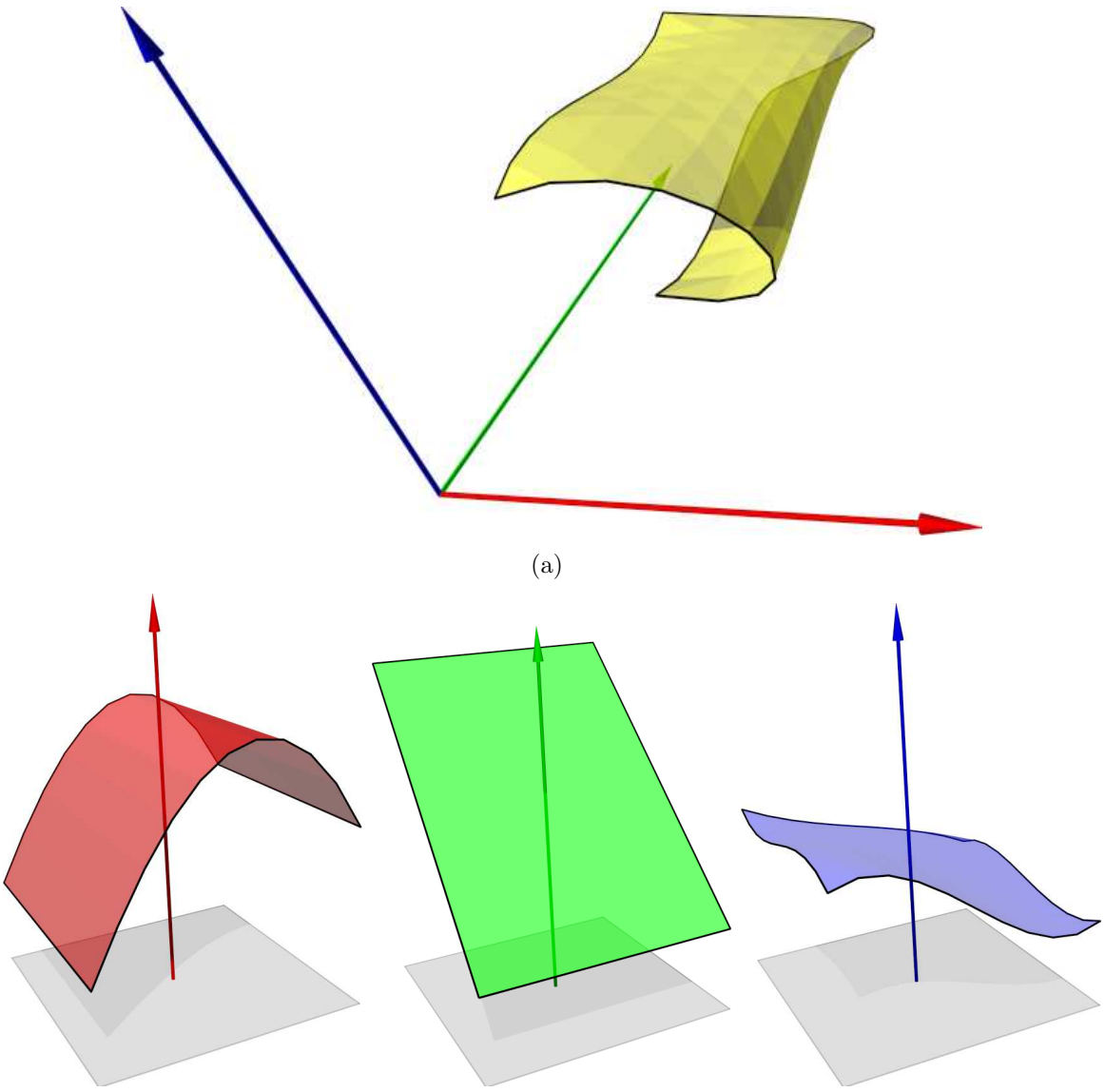
$$\begin{aligned}(D^r_\xi B^\alpha_\kappa)(x) &= \sum_{\lambda \in \mathbb{I}^r_{d-1}} \frac{r!}{\lambda!} \prod_{i=0}^{d-1} \left[\partial^{\lambda_i} \left(t : \mathbb{R} \to B^{\alpha_i}_{\kappa_i}(x_i + t\xi_i)\right)\right](0) \\ &= \sum_{\lambda \in \mathbb{I}^r_{d-1}} \frac{r!}{\lambda!} \prod_{i=0}^{d-1} \xi_i^{\lambda_i} (\partial^{\lambda_i} B^{\alpha_i}_{\kappa_i})(x_i) \\ &= \sum_{\lambda \in \mathbb{I}^r_{d-1}} \frac{r!}{\lambda!} \xi^\lambda (\partial^\lambda B^\alpha_\kappa)(x) \qquad (5.5)\end{aligned}$$

para todo $x \in \mathbb{R}^d$. Observe que $(D^r_\xi B^\alpha_\kappa)$ é um polinômio tensorial cujo multi-grau β não excede α. No caso específico em que ξ é paralelo ao eixo de coordenadas i do domínio, $\beta_i = \max\{0, \alpha_i - r\}$ e $\beta_j = \alpha_j$ para todo $j \neq i$.

5.2 Elementos tensoriais de Bézier

Assim como polinômios univariados de Bézier são usados para modelar curvas arbitrárias, os polinômios tensoriais de Bézier são usados para modelar superfícies e volumes. Para este fim, define-se um *elemento tensorial de Bézier de dimensão d e multi-grau* $\alpha \in \mathbb{I}_{d-1}$ *em* $\mathbb{R}^m$ como sendo uma função $F : [0,1]^d \to \mathbb{R}^m$ cujas m componentes são polinômios do espaço $\mathcal{P}^\alpha_d$ definidos em termos da representação de Bézier. Note que um arco de Bézier é um caso particular de elemento tensorial de Bézier (com $d = 1$). Os outros casos de maior importância em modelagem geométrica são o *retalho tensorial* ($d = 2$) e o *bloco tensorial* ($d = 3$) de Bézier. Veja a figura 5.2.

De forma análoga aos arcos de Bézier, os m coeficientes de Bézier de um mesmo elemento F com mesmo multi-índice κ em cada componente podem ser vistos como as coordenadas de um ponto P_κ do $\mathbb{R}^m$, o *ponto de controle* de Bézier de índice κ. A *grade de controle* de um elemento de Bézier é formada pelos seus pontos de controle e pelos segmentos de reta que ligam cada ponto P_κ aos pontos P_λ tais que apenas um dos índices λ_i difere do correspondente índice κ_i, e em apenas uma unidade. Veja a figura 5.3. Em decorrência da propriedade P3′, os pontos $F(x)$ de um elemento tensorial F estão totalmente contidos no fecho convexo dos pontos de controle de F [9].



(a)

(b) (c) (d)

Figura 5.2: (a) Um elemento tensorial de Bézier F de dimensão 2 (retalho) e multi-grau $(2, 3)$ no $\mathbb{R}^3$; e suas componentes (b) F_0, (c) F_1 e (d) F_2.

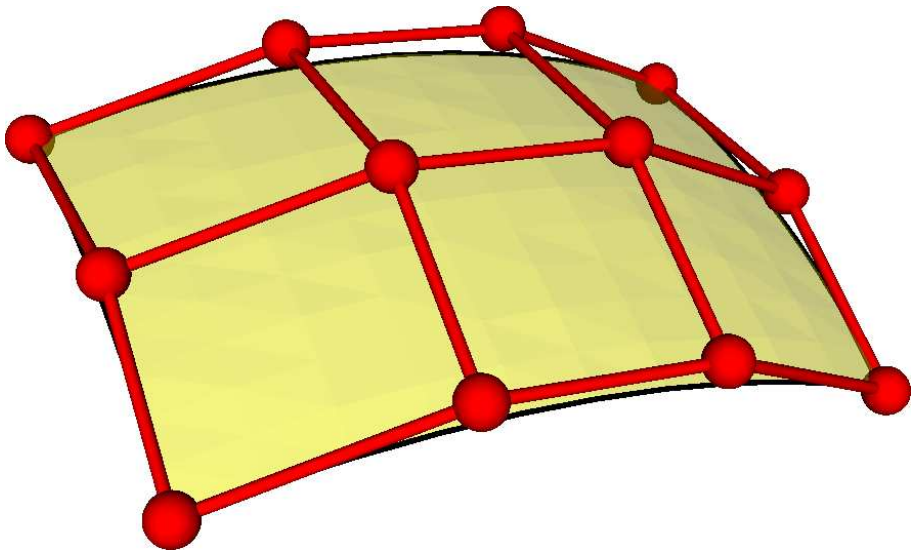


Figura 5.3: A grade de controle (em vermelho) de um retalho tensorial de Bézier de dimensão 2 e multi-grau $(3, 2)$(amarelo).

Capítulo 6

Elementos de Bézier simpliciais

Neste capítulo, revisamos conceitos relacionados aos polinômios de *Bernstein-Bézier simpliciais* e definimos os *elementos de Bézier simpliciais*.

6.1 Espaços afins canônicos

Definimos o *espaço afim canônico de dimensão* $d \in \mathbb{N}$, denotado por $\mathbb{A}^d$, como o conjunto

$$\mathbb{A}^d = \{u \in \mathbb{R}^{d+1} \mid \sum_{i=0}^{d} u_i = 1\}. \tag{6.1}$$

Note que os pontos de $\mathbb{A}^d$ constituem um hiperplano d-dimensional do $\mathbb{R}^{d+1}$ que corta cada eixo na coordenada 1. Veja a figura 6.1. Note também que $\mathbb{A}^0$ possui um único ponto, a 1-tupla (1); e que $\mathbb{A}^{d+e}$ *não* é o mesmo que $\mathbb{A}^d \times \mathbb{A}^e$.

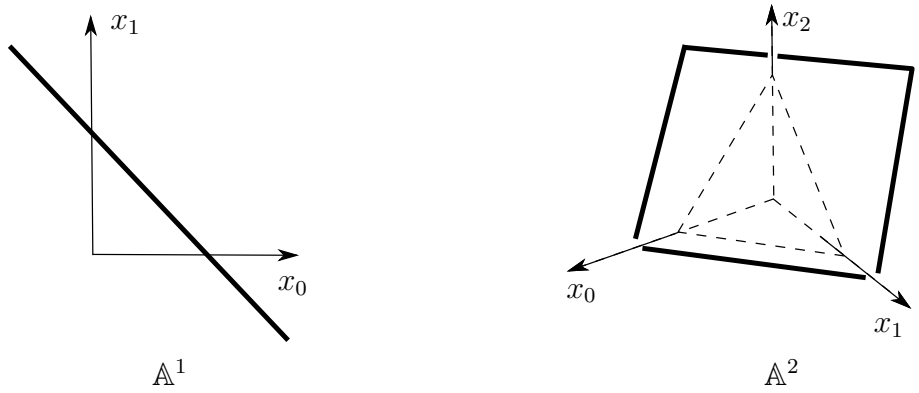


Figura 6.1: Espaços afins $\mathbb{A}^1$ e $\mathbb{A}^2$.

Definimos também o *espaço tangente afim canônico de dimensão* $d \in \mathbb{N}$, denotado por $\mathbb{V}^d$, como o espaço tangente de $\mathbb{A}^d$, isto é o conjunto de todos os vetores de $\mathbb{R}^{d+1}$ que são

diferenças de dois pontos de $\mathbb{A}^d$. Mais especificamente,

$$\mathbb{V}^d = \{\xi \in \mathbb{R}^{d+1} \mid \sum_{i=0}^{d} \xi_i = 0\}. \tag{6.2}$$

Note que $\mathbb{V}^{d+e}$ não é o mesmo que $\mathbb{V}^d \times \mathbb{V}^e$.

6.1.1 Simplexos canônicos

Definimos o *simplexo canônico de dimensão* $d \geq 0$ como o conjunto $\mathbb{K}^d$ dos pontos de $\mathbb{A}^d$ que tem todas as coordenadas não-negativas; ou seja

$$\mathbb{K}^d = \{(u_0, \cdots, u_d) \in \mathbb{R}^{d+1} \mid \sum_{i=0}^{d} u_i = 1 \wedge \bigwedge_{i=0}^{d} u_i \geq 0\} \tag{6.3}$$

Os simplexos canônicos de dimensão 1, 2 e 3 são um segmento em $\mathbb{R}^2$, um triângulo eqüilátero em $\mathbb{R}^3$ e um tetraedro regular em $\mathbb{R}^4$, ilustrados na figura 6.2.

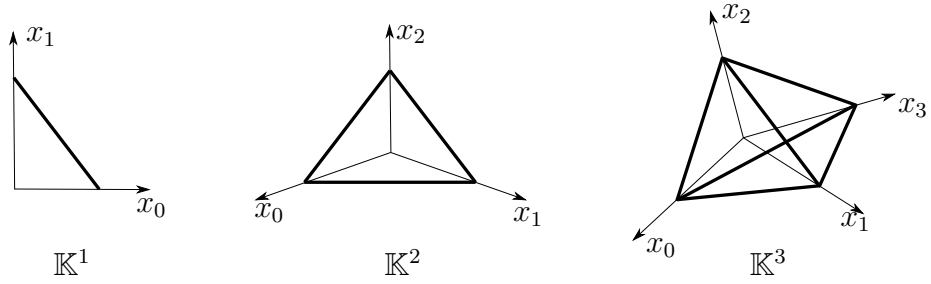


Figura 6.2: Simplexos canônicos.

6.1.2 Facetas

Definimos a *faceta* j do simplexo canônico $\mathbb{K}^d$, denotada por $\mathbb{K}^d|_j$, como sendo o conjunto de pontos $u \in \mathbb{K}^d$ tais que $u_j = 0$. Note que $\mathbb{K}^d|_j$ é congruente ao simplexo $\mathbb{K}^{d-1}$.

6.2 Polinômios simpliciais

Definimos o espaço dos *polinômios simpliciais de dimensão* $d \in \mathbb{N}$ *e grau* $g \in \mathbb{N}$, denotados por $\mathcal{P}_d^g$, como o conjunto das funções f de $\mathbb{A}^d$ para $\mathbb{R}$ tais que f pode ser expresso como um polinômio de grau g nas $d+1$ coordenadas do argumento u. Por exemplo,

$$f(u) = 3u_0u_1^2 + 2u_1^3 \tag{6.4}$$

é um elemento de $\mathcal{P}_2^2$. Verifica-se que a dimensão de $\mathcal{P}_d^g$ é $\binom{d+g}{g}$ [5].

Uma vez que as coordenadas do argumento somam 1, qualquer monômio pode ser multiplicado por qualquer potência de $(u_0 + \cdots + u_d)$ sem afetar o valor de f. Portanto, o mesmo polinômio pode ser escrito de várias formas. O polinômio f da fórmula (6.4), por exemplo, pode ser escrito também como

$$f(u) = 3u_0^2u_1^2 + 2u_0u_1^3 + 3u_0u_1^3 + 2u_1^4$$

Se f é um polinômio de grau g, podemos obter uma fórmula única (para cada g) exigindo que o polinômio seja *homogêneo*, isto é que todos os monômios tenham o mesmo grau total. Para colocar uma coleção arbitrária de monômios nesta forma, basta multiplicar todo monômio com grau total $k < g$ por $(u_0 + \cdots + u_d)^{g-k}$.

6.2.1 A base de Bernstein simplicial

Dados $d \in \mathbb{N}$ e $g \in \mathbb{N}$, a *base de Bernstein simplicial* de $\mathcal{P}_d^g$ consiste dos polinômios simpliciais B_κ^g de $\mathbb{A}^d$ para $\mathbb{R}$, para todo $\kappa \in \mathbb{I}_d^g$, onde

$$B_\kappa^g(u) = \binom{g}{\kappa} u^\kappa \quad \text{para todo } u \in \mathbb{A}^d \tag{6.5}$$

Veja figura 6.3.

Note que, no caso específico onde $d = 1$, a fórmula (6.5) para $B_\kappa^g(u)$ é equivalente à fórmula (3.2) do polinômio de Bernstein univariado $B_i^g(z)$, com $u_0 = z$, $u_1 = \bar{z} = 1 - z$, $\kappa_0 = i$ e $\kappa_1 = g - i$.

Observe também que a fórmula (6.5) está relacionada com a fórmula de Newton para a potência g do multinômio $(u_0 + u_1 + \cdots + u_d)$. Em teoria estatística, esta fórmula define a *distribuição de probabilidade multinomial*. Suponha que um experimento pode ter $d + 1$ resultados mutuamente exclusivos $E_0, \cdots, E_d$ sendo que cada E_i ocorre com probabilidade u_i. Então, $B_\kappa^g(u)$ é a probabilidade de, em g tentativas, o evento E_0 ocorrer κ_0 vezes, E_1, κ_1 vezes, ..., e E_d κ_d vezes.

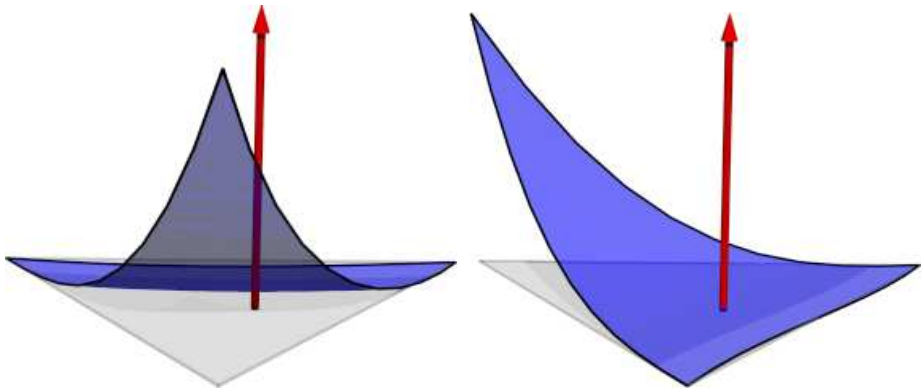
As propriedades de polinômios univariados têm análogos para simpliciais:

P1″: Os polinômios são não-negativos no simplexo $\mathbb{K}^d$

P2″: Cada polinômio tem apenas um máximo no simplexo de $\mathbb{K}^d$, no ponto $\kappa/g \in \mathbb{A}^d$.

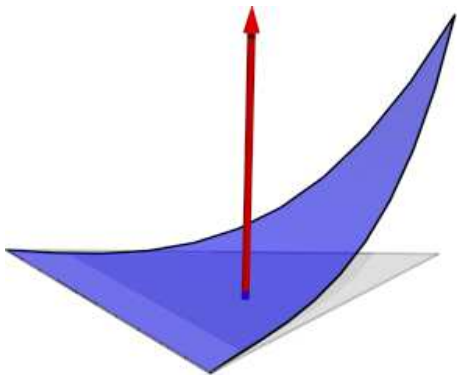
P3″: Os polinômios B_κ^g de mesmo grau g formam uma partição da unidade:

$$\sum_{\kappa \in \mathbb{I}_d^g} B_\kappa^g(u) = 1 \quad \text{para qualquer } u \in \mathbb{K}^d \tag{6.6}$$

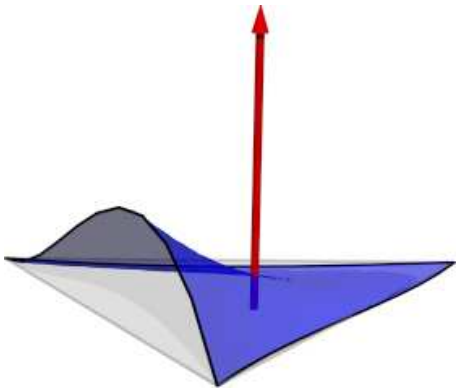


(a) $B^3_{(3,0,0)}(u)$

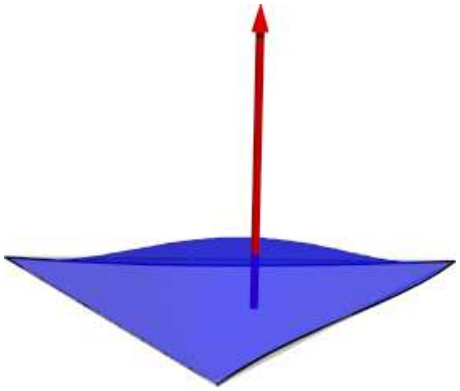
(b) $B^3_{(0,3,0)}(u)$



(c) $B^3_{(0,0,3)}(u)$



(d) $B^3_{(2,1,0)}(u)$



(e) $B^{(3)}_{(1,1,1)}(u)$

Figura 6.3: Polinômios da base de Benstein simplicial de $\mathcal{P}^3_2$.

6.2.2 A representação de Bézier de polinômios simpliciais

A representação de Bézier de uma função polinomial simplicial f de grau g de $\mathbb{A}^d$ para $\mathbb{R}$ é sua expansão em termos da base de Bernstein simplicial de grau g; a saber,

$$f(u) = \sum_{\kappa \in \mathbb{I}_d^g} b_\kappa B_\kappa^g(u) \tag{6.7}$$

onde cada b_κ é um número real, o *coeficiente* (*simplicial*) *de Bézier* de f. Como no caso tensorial, a *posição nominal* do coeficiente b_κ é $\kappa/g = (\kappa_0/g, \cdots, \kappa_d/g) \in \mathbb{A}^d$.

6.2.3 Elevação de grau

A fórmula de elevação de grau de polinômios de Bernstein univariados (3.4) pode ser generalizada para polinômios simpliciais. Especificamente, para quaisquer graus $g, h \in \mathbb{N}$ tais que $g \leq h$, qualquer dimensão $d \in \mathbb{N}$ e qualquer multi-índice $\kappa \in \mathbb{I}_d^g$, temos

$$B_\kappa^g(u) = \frac{g!}{\kappa!} u^\kappa \quad = \quad \binom{h}{g}^{-1} \sum_{\substack{\mu \in \mathbb{I}_d^h \\ \kappa \leq \mu}} \binom{\mu}{\kappa} B_\mu^h(u) \tag{6.8}$$

Uma demonstração desta fórmula foi publicada por Trump [24].

6.2.4 Derivada direcional

A derivação de funções definidas em $\mathbb{A}^d$ é mais complicada do que em $\mathbb{R}^d$. Como as coordenadas do domínio devem somar 1, não é possível alterar o valor de uma coordenada mantendo as demais fixas; e portanto o conceito de derivada parcial não faz muito sentido. Por esta razão, para funções definidas no $\mathbb{A}^d$ é mais conveniente trabalhar com derivadas direcionais.

Seja F uma função de $\mathbb{A}^d$ para $\mathbb{R}$ e ξ um vetor de $\mathbb{V}^d$. Definimos a *r-ésima derivada* (*direcional*) *de F na direção ξ* como sendo a função $D_\xi^r F$, de $\mathbb{A}^d$ para $\mathbb{R}$, definida por

$$(D_\xi^r F)(u) = \left[\partial^r (s : \mathbb{R} \to F(u + s\xi))\right](0) \quad \text{para todo } u \in \mathbb{A}^d. \tag{6.9}$$

Aplicando esta definição ao polinômio de Bernstein (6.5), temos

$$(D_\xi^r B_\kappa^g)(u) \quad = \quad \left[\partial^r \left(t : \mathbb{R} \to \frac{g!}{\kappa!}(u + t\xi)^\kappa\right)\right](0) \tag{6.10}$$

Usando a fórmula de Leibnitz (4.2),

$$(D_\xi^r B_\kappa^g)(u) \quad = \quad \frac{g!}{\kappa!} \sum_{\lambda \in \mathbb{I}_d^r} \frac{r!}{\lambda!} \prod_{i=0}^{d} \left[\partial^{\lambda_i} \left(t : \mathbb{R} \to (u_i + t\xi_i)^{\kappa_i}\right)\right](0) \tag{6.11}$$

Note que o p produtório se anula quando $\lambda_i > \kappa_i$ para algum i, uma vez que a derivada de ordem $\lambda_i + 1$ de um polinômio de grau κ_i é zero. Portanto podemos restringir o somatório aos multi-índices λ tais que $\lambda \leq \kappa$.

$$\begin{aligned}
(D^r_\xi B^g_\kappa)(u) &= \frac{g!}{\kappa!} \sum_{\substack{\lambda \in \mathbb{I}^r_d \\ \lambda \leq \kappa}} \frac{r!}{\lambda!} \prod_{i=0}^{d} \frac{\kappa_i!}{(\kappa_i - \lambda_i)!} u_i^{\kappa_i - \lambda_i} \xi_i^{\lambda_i} \\
&= \frac{g!}{\kappa!} \sum_{\substack{\lambda \in \mathbb{I}^r_d \\ \lambda \leq \kappa}} \frac{r!}{\lambda!} \frac{\kappa!}{(\kappa - \lambda)!} u^{\kappa - \lambda} \xi^\lambda \\
&= \frac{g!}{(g-r)!} \sum_{\substack{\lambda \in \mathbb{I}^r_d \\ \lambda \leq \kappa}} \frac{r!}{\lambda!} \xi^\lambda \frac{(g-r)!}{(\kappa - \lambda)!} u^{\kappa - \lambda} \\
&= \frac{g!}{(g-r)!} \sum_{\substack{\lambda \in \mathbb{I}^r_d \\ \lambda \leq \kappa}} \frac{r!}{\lambda!} \xi^\lambda B^{g-r}_{\kappa - \lambda}(u)
\end{aligned}$$

Estendendo a definição do polinômio de Bernstein (6.5) para vetores de $\mathbb{V}^d$, podemos escrever o fator

$$\frac{r!}{\lambda!} \xi^\lambda$$

como

$$B^r_\lambda(\xi).$$

Portanto temos que

$$(D^r_\xi B^g_\kappa)(u) = \frac{g!}{(g-r)!} \sum_{\substack{\lambda \in \mathbb{I}^r_d \\ \lambda \leq \kappa}} B^r_\lambda(\xi) \;\; B^{g-r}_{\kappa - \lambda}(u) \tag{6.12}$$

Observe que a derivada se anula (porque o somatório é vazio) se $r > g$. A equação (6.12) generaliza a apresentada por Farin para o caso bidimensional [8]. Se introduzimos $\mu = \kappa - \lambda$, obtemos a fórmula alternativa

$$(D^r_\xi B^g_\kappa)(u) = \frac{g!}{(g-r)!} \sum_{\mu \in \mathbb{I}^{g-r}_d} B^r_{\kappa - \mu}(\xi) \;\; B^{g-r}_\mu(u) \tag{6.13}$$

6.3 Elementos simpliciais de Bézier

Um *elemento simplicial de Bézier de dimensão d e grau g em* $\mathbb{R}^m$ é uma função $F : \mathbb{K}^d \to \mathbb{R}^m$ cujas m componentes $F_0, F_1, \cdots, F_{m-1}$ são polinômios do espaço $\mathcal{P}^g_d$ definidos em termos da representação de Bézier. Note que o arco de Bézier é um caso particular de

elemento simplicial de Bézier (com $d = 1$). Os outros casos de maior importância para modelagem geométrica são o *retalho simplicial* (*ou triângulo*) *de Bézier* ($d = 2$) e o *bloco simplicial* (*ou tetraedro*) *de Bézier* ($d = 3$). Veja a figura 6.4.

Os m coeficientes de Bézier de mesmo multi-índice κ em cada componente F_i podem ser vistos como as coordenadas de um ponto P_κ do $\mathbb{R}^m$, o *ponto de controle de Bézier* de multi-índice κ do elemento. É conveniente associar o ponto P_κ ao ponto $u = \kappa/g$ do domínio $\mathbb{K}^d$. Similarmente ao caso da curva e do elemento tensorial, segue da propriedade P3″ que a imagem de um elemento simplicial F está totalmente contida no fecho convexo dos pontos de controle de F [9]. A *grade de controle* do elemento é formada pelos pontos de controle P_κ e pelos segmentos que ligam todos os pares de pontos P_κ e P_λ tais que κ e λ diferem em exatamente dois índices i e j, com $\kappa_i - \lambda_i = +1$ e $\kappa_j - \lambda_j = -1$. Veja a figura **??**.

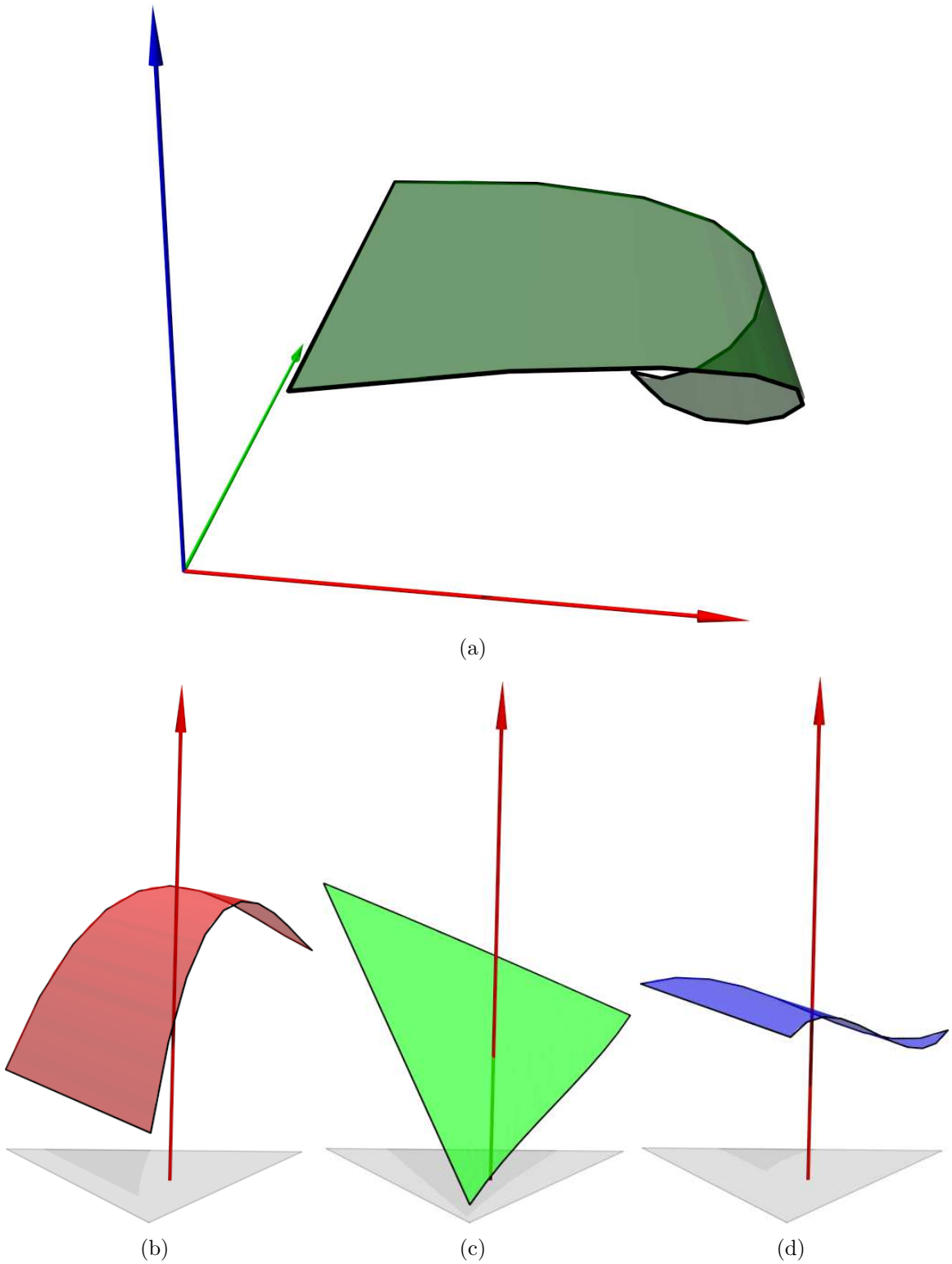


Figura 6.4: (a) Um elemento simplicial de Bézier F de dimensão 2 (retalho triangular) e grau 3 no $\mathbb{R}^3$; e suas componentes (b) F_0, (c) F_1 e (d) F_2.

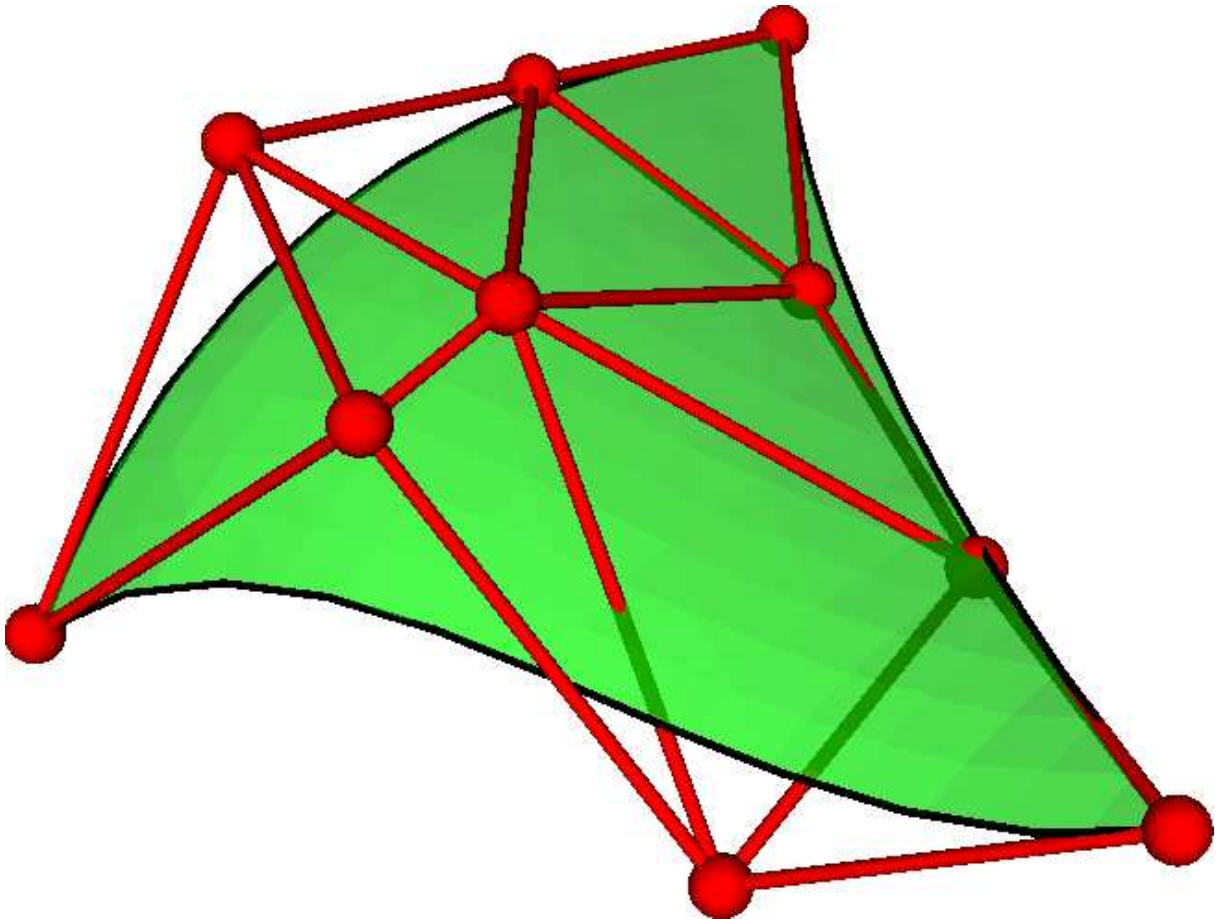


Figura 6.5: Um elemento simplicial de Bézier de dimensão 2 e grau 3, e sua grade de controle (em vermelho).

Capítulo 7

Conversão tensorial/simplicial

Em modelagem geométrica, freqüentemente surge a necessidade de converter elementos tensoriais para simpliciais e vice versa, por exemplo para obter sub-retalhos triangulares de retalhos retangulares, ou vice-versa. Estas conversões também são necessárias para impor restrições de continuidade entre um spline definido em uma malha retangular e um spline definido em uma malha triangular. Veja a figura 7.1.

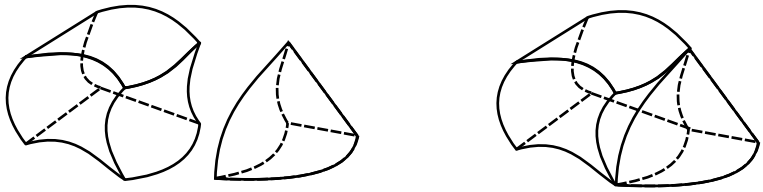


Figura 7.1: Colando um retalho 3D tensorial a um retalho tetraédrico.

Neste capítulo, apresentamos fórmulas para conversão entre estas duas representações de polinômios multivariados. Estes resultados generalizam, para dimensão arbitrária, a conversão de retalhos retangulares para triangulares de Goldmann [11], Hu [14] e Lasser [18], e a conversão de retalhos triangulares para retangulares de Brueckner [3], Hu [13] [14]e Lasser [17].

7.1 Mapeamento canônico tensorial/simplicial

Definimos o *mapeamento canônico* de $\mathbb{A}^d$ para $\mathbb{R}^d$ como a função η_d tal que se $x = \eta_d(u)$, então $x_i = u_i$ para $i \in \{0, \cdots, d-1\}$. Veja as figuras 7.2 e 7.3.
Dizemos que um polinômio f'', definido no espaço $\mathbb{A}^d$, é a *forma simplicial canônica* de um polinômio tensorial f', definido no espaço $\mathbb{R}^d$, se para todo $u \in \mathbb{A}^d$, $f''(u) = f'(\eta_d(u))$. Nesse caso, dizemos que f' é a *forma tensorial canônica* de f''. Veja a figura 7.4.

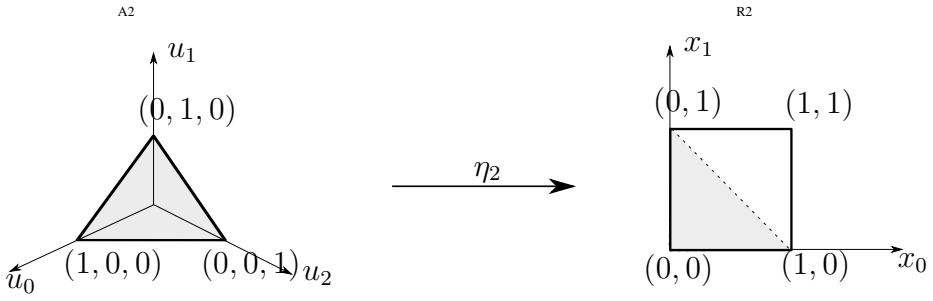


Figura 7.2: O mapeamento canônico η_2.

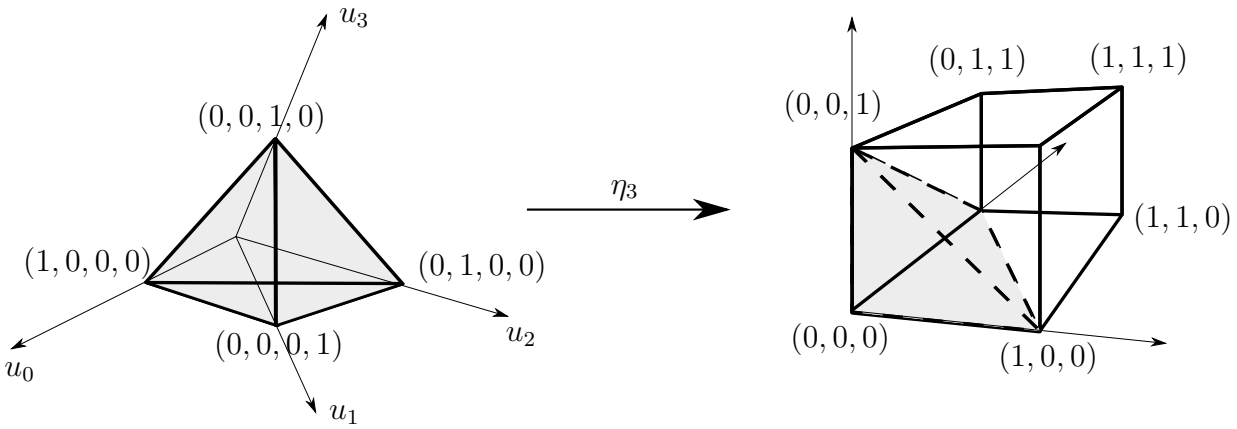


Figura 7.3: O mapeamento canônico η_3.

Nosso objetivo, neste capítulo, é desenvolver fórmulas explícitas para conversão de um polinômio tensorial f' para sua forma simplicial canônica f'', e vice versa, na representação de Bézier. Mais especificamente, dados os coeficientes de Bézier b'_σ de um polinômio tensorial f' de multigrau $\alpha \in \mathbb{I}_d$,

$$f'(x) = \sum_{\substack{\sigma \in \mathbb{I}_d \\ \sigma \leq \alpha}} b'_\sigma B^\alpha_\sigma(x)$$

desejamos encontrar os coeficientes b''_κ de sua simplicial canônica f'',

$$f''(u) = \sum_{\kappa \in \mathbb{I}^g_d} b''_\kappa B^g_\kappa(u)$$

onde $g = |\alpha|$. A relação entre os coeficientes b'_σ e b''_κ pode ser resumida por uma matriz de mudança de base $M_{\kappa\sigma}$:

$$b''_\kappa = \sum_{\sigma \in \mathbb{I}^g_d} M_{\kappa\sigma} b'_\sigma$$

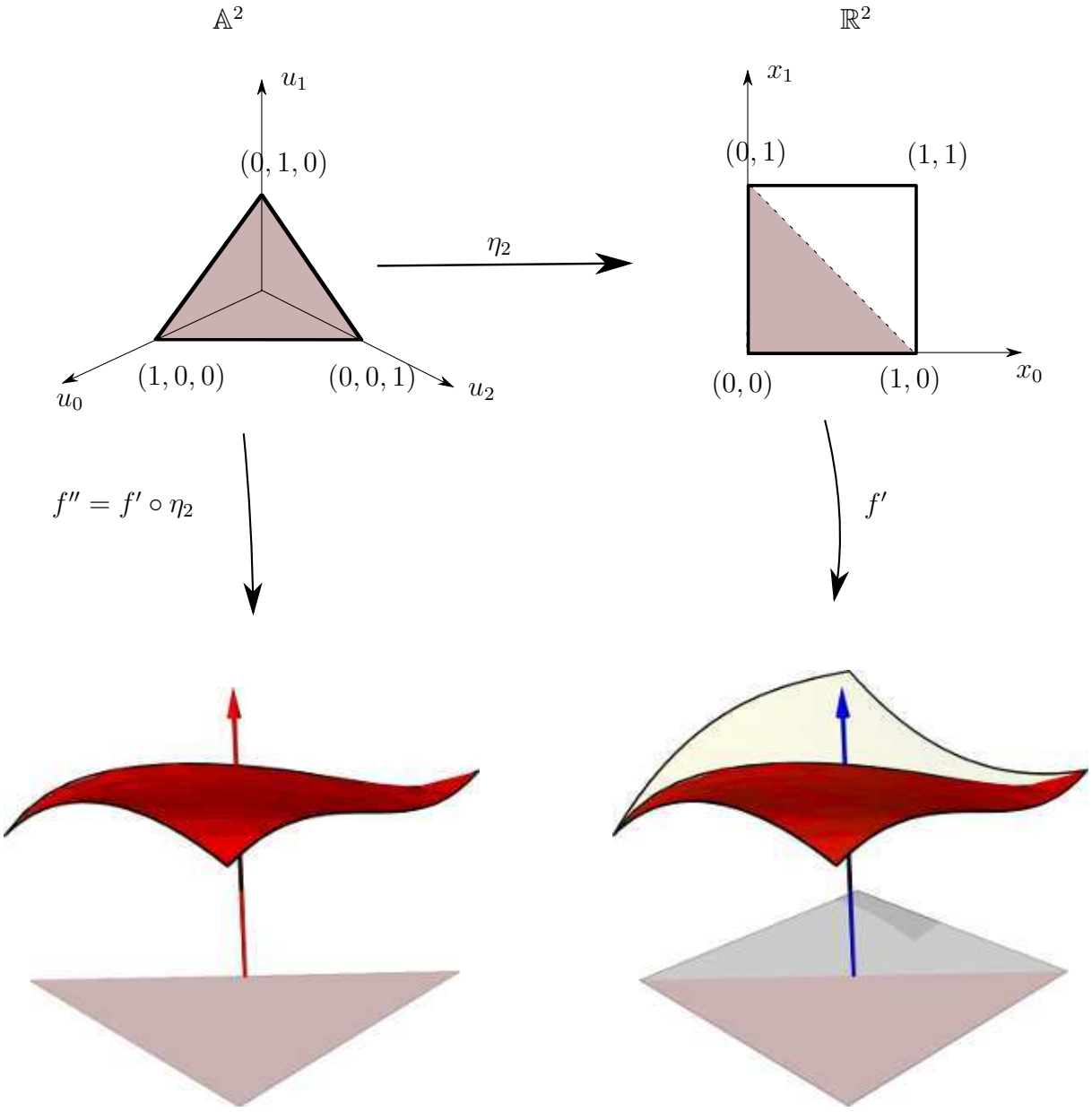


Figura 7.4: O mapeamento canônico η_2 e a relação entre um polinômio tensorial f', definido no $\mathbb{R}^2$, e sua forma simplicial canônica f''.

7.2 Conversão de tensorial para simplicial

Pela definição (5.1), dados $\alpha \in \mathbb{I}_{d-1}$ e $\mu \in \mathbb{I}_{d-1}$ tal que $\mu \leq \alpha$, a forma simplicial canônica do polinômio de Bernstein tensorial B^{α}_{σ} pode ser escrita como

$$B^{\alpha}_{\sigma}(\eta_d(u)) \;=\; \prod_{j=0}^{d-1} B^{\alpha_j}_{\sigma_j}(u_j) \;\text{ para todo } u \in \mathbb{A}^d \tag{7.1}$$

Além disso, cada polinômio de Bernstein univariado $B^g_i(u_j)$ no lado direito da fórmula (7.1) pode ser descrito como uma combinação linear de polinômios de Bernstein simpliciais definidos em $\mathbb{A}^d$. Mais especificamente

$$\begin{aligned}
B^g_i(u_j) &= \frac{g!}{i!(g-i)!} u_j^i (1-u_j)^k \\
&= \frac{g!}{i!(g-i)!} u_j^i \left(1 - \left(\sum_{\substack{:p=0\\:j\neq j}}^{d} u_p\right) + \left(\sum_{\substack{:p=0\\:j\neq j}}^{d} u_p\right) - u_j\right)^k \\
&= \frac{g!}{i!k!} u_j^i (u_0 + \cdots + u_{j-1} + u_{j+1} + \cdots + u_d)^k \\
&= \frac{g!}{i!k!} u_j^i \sum_{\substack{\kappa\in\mathbb{I}^k_d\\ \kappa_j=0}} \frac{k!}{\kappa!} u_0^{\kappa_0} \cdots u_{j-1}^{\kappa_{j-1}} u_{j+1}^{\kappa_{j+1}} \cdots u_d^{\kappa_d} \\
&= \sum_{\substack{\kappa\in\mathbb{I}^k_d\\ \kappa_j=0}} \frac{g!}{\kappa!i!} u_0^{\kappa_0} \cdots u_{j-1}^{\kappa_{j-1}}\, u_j^i\, u_{j+1}^{\kappa_{j+1}} \cdots u_d^{\kappa_d} \\
&= \sum_{\substack{\lambda\in\mathbb{I}^g_d\\ \lambda_j=i}} \frac{g!}{\lambda!} u_0^{\lambda_0} \cdots u_d^{\lambda_d} \\
&= \sum_{\substack{\lambda\in\mathbb{I}^g_d\\ \lambda_j=i}} B^g_{\lambda}(u)
\end{aligned} \tag{7.2}$$

Combinando esta equação com a fórmula (7.1), temos

$$B^{\alpha}_{\sigma}(\eta_d(u)) \;=\; \prod_{j=0}^{d-1} \sum_{\substack{\kappa\in\mathbb{I}^{\alpha_j}_d\\ \kappa_j=\sigma_j}} B^{\alpha_j}_{\kappa}(u)$$

Para expandir o produtório acima em termos de somatórios, utilizamos a notação de hiper-índices

$$B^{\alpha}_{\sigma}(\eta_d(u)) \;=\; \sum_{\substack{\Lambda\in\mathbb{H}^{\alpha}_{d-1,d}\\ \mathrm{diag}(\Lambda)=\sigma}} B^{\alpha_0}_{\Lambda_0}(u) \cdots B^{\alpha_{d-1}}_{\Lambda_{d-1}}(u)$$

Agora usamos o fato de que o produto de $m+1$ polinômios de Bernstein simpliciais de graus $\alpha_0, \alpha_1, \cdots, \alpha_m$ pode ser reduzido a um único polinômio de Bernstein simplicial de grau $g = \alpha_0 + \alpha_1 + \cdots + \alpha_m$. Segundo DeRose [6],

$$B^{\alpha_0}_{\Lambda_0}(u) \cdots B^{\alpha_{d-1}}_{\Lambda_{d-1}}(u) = \frac{\alpha!(\Lambda_0 + \cdots + \Lambda_{d-1})!}{\Lambda_0! \cdots \Lambda_{d-1}!|\alpha|!} B^{|\alpha|}_{\Lambda_0+\cdots+\Lambda_{d-1}}(u). \tag{7.3}$$

Então,

$$\begin{aligned} B^{\alpha}_{\sigma}(\eta_d(u)) &= \sum_{\substack{\Lambda \in \mathbb{H}^{\alpha}_{d-1,d} \\ \mathrm{diag}(\Lambda)=\sigma}} \frac{\alpha!(\Lambda_0 + \cdots + \Lambda_{d-1})!}{\Lambda!|\alpha|!} B^{|\alpha|}_{\Lambda_0+\cdots+\Lambda_{d-1}}(u) \\ &= \sum_{\kappa \in \mathbb{I}^{|\alpha|}_d} \sum_{\substack{\Lambda \in \mathbb{H}^{\alpha,\kappa}_{d-1,d} \\ \mathrm{diag}(\Lambda)=\sigma}} \frac{\alpha!\kappa!}{\Lambda!|\alpha|!} B^{|\alpha|}_{\kappa}(u) \end{aligned}$$

Portanto,

$$M_{\sigma\kappa} = \sum_{\substack{\Lambda \in \mathbb{H}^{\alpha,\kappa}_{d-1,d} \\ \mathrm{diag}(\Lambda)=\sigma}} \frac{\alpha!\kappa!}{\Lambda!|\alpha|!}$$

7.3 Conversão de simplicial para tensorial

Seja g, d naturais e κ um multi-índice de $\mathbb{I}^g_d$. As fórmulas a seguir expressam a forma tensorial canônica de um polinômio de Bernstein simplicial B^g_κ, como uma combinação linear de polinômios de Bernstein tensoriais $B^\alpha_\sigma(x)$, onde $\alpha = g^{*d} = (g, \cdots, g) \in \mathbb{I}_{d-1}$.

Para estas fórmulas, denotamos por i o último elemento κ_d de κ, e por $\kappa' \in \mathbb{I}^{g-i}_{d-1}$ o multi-índice que é igual a κ nos seus primeiros $d-1$ elementos (ou seja, $\kappa' = \kappa|_{d-1}$). Observe que $\kappa! = \kappa'!\, i!$ e $u^\kappa = u_d^i \prod_{j=0}^{d-1} u_j^{\kappa'_j}$. Equação (6.5) então se torna

$$\begin{aligned} B^g_\kappa(\eta_d^{-1}(x)) &= \frac{g!}{\kappa!} x_0^{\kappa'_0} x_1^{\kappa'_1} \cdots u_{d-1}^{\kappa'_{d-1}} (1 - x_0 - \cdots x_{d-1})^i \\ &= \frac{g!}{\kappa!} x_0^{\kappa'_0} x_1^{\kappa'_1} \cdots x_{d-1}^{\kappa'_{d-1}} \sum_{\mu \in \mathbb{I}^i_d} \frac{i!}{\mu!} (1-d)^{\mu_d} \prod_{k=0}^{d-1} (1 - x_k)^{\mu_k} \\ &= \sum_{\mu \in \mathbb{I}^i_d} (1-d)^{\mu_d} \frac{g!}{(\mu+\kappa)!} \frac{(\mu_d + i)!}{\mu_d!} \prod_{k=0}^{d-1} B^{\kappa'_k + \mu_k}_{\kappa'_k}(x_k) \end{aligned} \tag{7.4}$$

Introduzindo a variável $j = i - \mu_d$ e substituindo μ_d por $i - j$, equação (7.4) se torna

$$B_\kappa^g(\eta_d^{-1}(x)) \;=\; \sum_{j=0}^{i} \sum_{\nu \in \mathbb{I}_{d-1}^j} (1-d)^{i-j} \frac{g!}{(\nu+\kappa')!(i-j)!} \prod_{k=0}^{d-1} B_{\kappa'_k}^{\kappa'_k+\nu_k}(x_k) \tag{7.5}$$

Substituindo agora ρ por $\nu+\kappa'$ e q por $i-j$, equação (7.5) se torna

$$B_\kappa^g(\eta_d^{-1}(x)) \;=\; \sum_{q=0}^{i} \sum_{\substack{\rho \in \mathbb{I}_{d-1}^{g-q} \\ \rho \geq \kappa'}} (1-d)^q \frac{g!}{\rho! q!} \prod_{k=0}^{d-1} B_{\kappa'_k}^{\rho_k}(x_k) \tag{7.6}$$

Para reduzir a fórmula (7.6) a uma combinação linear de polinômios de Bernstein de mesmo multi-grau α, é necessário aplicar a fórmula (3.4) de elevação de grau de polinômios de Bernstein univariados a cada termo da somatória. Obtemos então

$$B_\kappa^g(\eta_d^{-1}(x)) \;=\; \sum_{\substack{\sigma \in \mathbb{I}_{d-1} \\ \sigma \geq \kappa' \\ \sigma \leq \alpha}} \left(\sum_{\substack{\rho \in \mathbb{I}_{d-1} \\ \rho \geq \kappa' \\ |\rho| \leq g \\ \rho \leq \alpha+\kappa'-\sigma}} (1-d)^{g-|\rho|} \frac{g!(\alpha-\rho)!}{\alpha!(g-|\rho|)!} \binom{\sigma}{\kappa'} \binom{\alpha-\sigma}{\rho-\kappa'} \right) B_\sigma^\alpha(x) \tag{7.7}$$

Portanto temos

$$M_{\sigma\kappa} = \begin{cases} \sum_{\substack{\rho \in \mathbb{I}_{d-1} \\ \rho \leq \kappa' \\ |\rho| \leq g \\ \rho \leq \alpha+\kappa'-\sigma}} (1-d)^{g-|\rho|} \frac{g!(\alpha-\rho)!}{\alpha!(g-|\rho|)!} \binom{\sigma}{\kappa'} \binom{\alpha-\sigma}{\rho-\kappa'} & \text{se } \kappa' \geq \sigma \\ 0 & \text{caso contrário} \end{cases}$$

Capítulo 8

Simplóides de Bézier

Neste capítulo, definiremos a abordagem unificada dos *polinômios simploidais*[6], que generalizam polinômios tensoriais e simpliciais para domínios que são produtos cartesianos de simplexos canônicos denominados *simplóides*. Em particular, descrevemos os *elementos de Bézier simploidais* ou *simplóides de Bézier*, que, analogamente, generalizam a representação de Bézier-DeCasteljau destes polinômios, como o elemento prismático da figura 8.1.

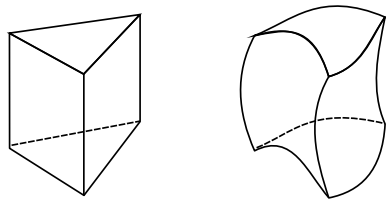


Figura 8.1: O prisma canônico (esquerda) e um exemplo de um bloco de Bézier simploidal (direita).

Neste capítulo, também daremos fórmulas gerais explícitas para elevação para graus arbitrários, bem como para o cálculo de derivadas direcionais de ordem arbitrária.

8.1 Espaço multi-afim canônico

Definimos $\mathbb{A}^\delta$, o *espaço multi-afim canônico de multi-dimensão* $\delta \in \mathbb{I}_m$, como o conjunto

$$\mathbb{A}^\delta = \{U \in \mathbb{M}_\delta : |U_i| = 1, i = 0, \cdots, m\}. \tag{8.1}$$

Note que $\mathbb{A}^\delta = \mathbb{A}^{\delta_0} \times \cdots \times \mathbb{A}^{\delta_m}$ e que, diferentemente do espaço cartesiano, $(\mathbb{A}^m)^{\times n}$ *não* é isomórfico a $\mathbb{A}^{m \cdot n}$ em geral. Quando um multi-índice δ é utilizado para denotar a dimensão de um espaço multi-afim, como na definição acima, o chamamos de *multi-dimensão*.

Definimos também o *espaço vetorial canônico de multi-dimensão* δ, denotado por $\mathbb{V}^\delta$, como o conjunto de diferenças entre pares de pontos do $\mathbb{A}^\delta$; isto é, o subespaço de $\mathbb{M}_\delta$ consistindo das matrizes Ξ tais que $|\Xi_i| = 0$ para todo $i = 0, \cdots, m$.

8.1.1 Simplóides canônicos

Dados $m \in \mathbb{N}$ e $\delta \in \mathbb{I}_m$, definimos o *simplóide canônico de multi-dimensão* δ como o produto cartesiano

$$\mathbb{K}^\delta = \mathbb{K}^{\delta_0} \times \cdots \times \mathbb{K}^{\delta_m}$$

Note que $\mathbb{K}^\delta \subset \mathbb{A}^\delta$. Os simplóides canônicos de dimensão $(1,1)$, $(2,1)$ e $(1,1,1)$ estão ilustrados na figura 8.2. Em particular, os simplexos canônicos $\mathbb{K}^d$ e os hipercubos canônicos

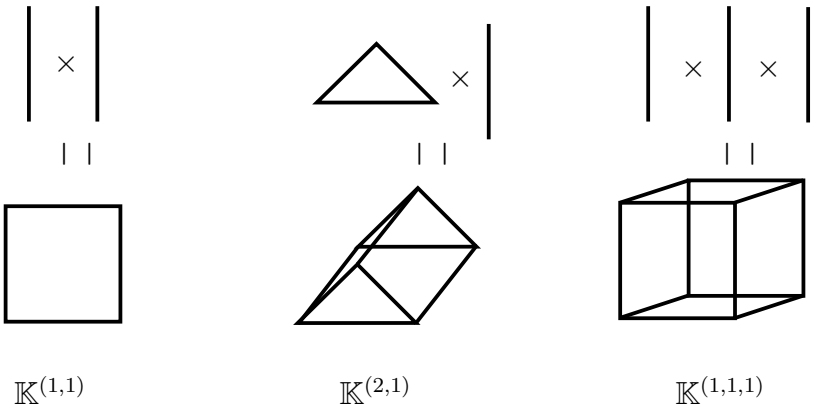


Figura 8.2: Alguns simplóides canônicos.

$[0,1]^d$ de $\mathbb{R}^d$ podem ser vistos como casos particulares de simplóides canônicos.

8.1.2 Facetas

Definimos a faceta de índices i, j de um simplóide canônico $\mathbb{K}^\delta$, para $0 \le i \le m$ e $0 \le j \le \delta_i$, denotada por $\mathbb{K}^\delta|_{i,j}$, como

$$\mathbb{K}^\delta|_{i,j} = \{U \in \mathbb{K}^\delta : U_{ij} = 0\}. \tag{8.2}$$

Observe que $\mathbb{K}^\delta|_{i,j}$ é congruente ao simplóide $\mathbb{K}^\varepsilon$ onde $\varepsilon \in \mathbb{I}_d$ é tal que $\varepsilon_i = \delta_i - 1$ e $\varepsilon_k = \delta_k$ para todo $k \neq i$. Observe também que o conceito de faceta de um simplexo canônica $\mathbb{K}^d$ (seção 6.1.2) é um caso particular desta definição.

8.2 Polinômios simploidais

Definimos o espaço dos *polinômios simploidais de multi-dimensão* $\delta \in \mathbb{I}_m$ *e multi-grau* $\alpha \in \mathbb{I}_m$, denotado por $\mathcal{P}_\delta^\alpha$, como o espaço de todas as funções f de $\mathbb{A}^\delta$ para os reais, tais que para cada $i \in \{0, \cdots, d\}$, $f(U)$ é uma função polinomial simplicial da componente U_i com grau α_i quando as outras componentes são consideradas fixas. Por exemplo, considere a função f de $\mathbb{A}^{(2,1)}$ para $\mathbb{R}$ definida por

$$\begin{aligned} f(U) &= f\begin{pmatrix} U_{00} & U_{01} & U_{02} \\ U_{10} & U_{11} & \end{pmatrix} \\ &= U_{00}^2U_{10}^2U_{11} + U_{01}^2U_{11}^3 + U_{01}^2U_{10}^3 + \\ &\quad + U_{01}U_{02}U_{10}U_{11}^2 + U_{01}U_{02}U_{10}^2U_{11} + U_{02}^2U_{11}^3 \\ &= (U_{10}^2U_{11})U_{00}^2 + (U_{11}^3 + U_{10}^3)U_{01}^2 + \\ &\quad + (U_{10}U_{11}^2 + U_{10}^2U_{11})U_{01}U_{02} + (U_{11}^3)U_{02}^2 \qquad (8.3) \\ &= (U_{01}^2)U_{10}^3 + (U_{00}^2 + U_{01}^2)U_{10}^2U_{11} + \\ &\quad + (U_{01}^2)U_{10}U_{11}^2 + (U_{01}^2 + U_{02}^2)U_{11}^3 \qquad (8.4) \end{aligned}$$

Observe que, pela fórmula (8.3), a função f é um polinômio simplicial de grau 2 da componente $U_0 = (U_{00}, U_{01}, U_{02})$; e pela fórmula (8.4), é um polinômio simplicial de grau 3 da componente $U_1 = (U_{10}, U_{11})$.

Note que qualquer função $f \in \mathcal{P}_\delta^\alpha$ também pertence ao espaço $\mathcal{P}_\delta^\beta$ se $\beta \geq \alpha$. Note ainda que todo polinômio simplicial pode ser visto trivialmente como um polinômio simploidal.

Ademais, todo polinômio tensorial f em $\mathbb{R}^d$ de multi-grau $\alpha \in \mathbb{I}_{d-1}$ pode ser visto como um polinômio simploidal definido no espaço multi-afim $\mathbb{A}^{1*d} = \mathbb{A}^1 \times \cdots \times \mathbb{A}^1$, utilizando o mapeamento ψ_d de $\mathbb{R}^d$ em $\mathbb{A}^{1*d}$ tal que $\psi_d(x_0, \cdots, x_{d-1}) = ((x_0, 1 - x_0), \cdots, (x_{d-1}, 1 - x_{d-1}))$. Por esta mesma correspondência, todo polinômio simploidal de domínio $\mathbb{A}^{1*d}$ e multi-grau $\alpha \in \mathbb{I}_{d-1}$ é essencialmente um polinômio tensorial de domínio $\mathbb{R}^d$ e mesmo multi-grau α.

8.2.1 A base de Bernstein simploidal

Para todo $m \in \mathbb{N}$, toda multi-dimensão $\delta \in \mathbb{I}_m$ e todo multi-grau $\alpha \in \mathbb{I}_m$, definimos a *base de Bernstein simploidal de* $\mathcal{P}_\delta^\alpha$, como consistindo dos polinômios simploidais B_Λ^α, para todo $\Lambda \in \mathbb{H}_\delta^\alpha$, cada um definido como um produto de polinômios de Bernstein simpliciais:

$$B_\Lambda^\alpha(U) = \prod_{i=0}^{m} B_{\Lambda_i}^{\alpha_i}(U_i) \quad \text{para todo } U \in \mathbb{A}^\delta. \qquad (8.5)$$

Note que estes polinômios generalizam os polinômios de Bernstein simpliciais ($\delta \in \mathbb{I}_0$) e polinômios de Bernstein tensoriais ($\delta = 1 * m = (1, 1, \cdots, 1)$).

8.2.2 Representação de Bézier de polinômios simploidais

A *representação de Bézier de uma função polinomial simploidal* f de multi-dimensão $\delta \in \mathbb{I}_d$ e multi-grau $\alpha \in \mathbb{I}_d$ é sua expansão em termos da base de Bernstein simploidal B^α_δ, a saber

$$f(U) = \sum_{\Lambda \in \mathbb{H}^\alpha_\delta} b_\Lambda B^\alpha_\Lambda(U)$$

onde cada b_Λ é um número real, um *coeficiente (simploidal) de Bézier* de f. De forma análoga aos polinômios simpliciais, a posição nominal de cada coeficiente b_Λ é definida como sendo o ponto $\Lambda/\alpha \in \mathbb{A}^\delta$

Por exemplo, a função f definida pela fórmula 8.4 tem os seguintes coeficientes de Bézier:

$$\begin{array}{cccc}
c^{200}_{30} = 0 & c^{200}_{21} = 1 & c^{200}_{12} = 0 & c^{200}_{03} = 0 \\
c^{110}_{30} = 0 & c^{110}_{21} = 0 & c^{110}_{12} = 0 & c^{110}_{03} = 0 \\
c^{020}_{30} = 1 & c^{020}_{21} = 1/3 & c^{020}_{12} = 1/3 & c^{020}_{03} = 1 \\
c^{011}_{30} = 0 & c^{011}_{21} = 0 & c^{011}_{12} = 0 & c^{011}_{03} = 0 \\
c^{002}_{30} = 0 & c^{002}_{21} = 0 & c^{002}_{12} = 0 & c^{002}_{03} = 1 \\
c^{101}_{30} = 0 & c^{101}_{21} = 0 & c^{101}_{12} = 0 & c^{101}_{03} = 0
\end{array}$$

8.2.3 Elevação de grau

A fórmula de elevação de grau (6.8) para polinômios simpliciais pode ser estendida para polinômios simploidais gerais. Mais especificamente, dados um natural m, uma multi-dimensão $\delta \in \mathbb{I}_m$, um multi-grau de origem $\alpha \in \mathbb{I}_m$, um hiper-índice $\Lambda \in \mathbb{H}^\alpha_\delta$ e um

multi-grau de destino $\beta \in \mathbb{I}_m$ tal que $\beta \geq \alpha$,temos

$$\begin{aligned} B^\alpha_\Lambda(U) &= \prod_{i=0}^{m} B^{\alpha_i}_{\Lambda_i}(U) \\ &= \frac{\left(\sum_{\substack{\kappa_0 \in \mathbb{I}^{\beta_0}_{\delta_0} \\ \Lambda_0 \leq \kappa_0}} \binom{\kappa_0}{\Lambda_0} B^{\beta_0}_{\kappa_0}(U)\right)}{\binom{\beta_0}{\alpha_0}} \cdots \frac{\left(\sum_{\substack{\kappa_m \in \mathbb{I}^{\beta_m}_{\delta_m} \\ \Lambda_m \leq \kappa_m}} \binom{\kappa_m}{\Lambda_m} B^{\beta_m \kappa_m}(U)\right)}{\binom{\beta_m}{\alpha_m}} \\ &= \frac{\sum_{\substack{\kappa_0 \in \mathbb{I}^{\beta_0}_{\delta_0} \\ \Lambda_0 \leq \kappa_0}} \cdots \sum_{\substack{\kappa_m \in \mathbb{I}^{\beta_m}_{\delta_m} \\ \Lambda_m \leq \kappa_m}} \binom{\kappa_0}{\Lambda_0} \cdots \binom{\kappa_m}{\Lambda_m} B^{\beta_0}_{\kappa_0}(U) \cdots B^{\beta_m}_{\kappa_m}(U)}{\binom{\beta}{\alpha}} \end{aligned}$$

Introduzindo o hiper-índice Ω cujas linhas são $\kappa_0, \kappa_1, \cdots, \kappa_m$ temos

$$\begin{aligned} B^\alpha_\Lambda(U) &= \binom{\beta}{\alpha}^{-1} \sum_{\substack{\Omega \in \mathbb{H}^\beta_{m,\delta} \\ \Lambda \leq \Omega}} \binom{\Omega_0}{\Lambda_0} \cdots \binom{\Omega_m}{\Lambda_m} B^\beta_\Omega(U) \\ &= \binom{\beta}{\alpha}^{-1} \sum_{\substack{\Omega \in \mathbb{H}^\beta_{m,\delta} \\ \Lambda \leq \Omega}} \binom{\Omega}{\Lambda} B^\beta_\Omega(U) \end{aligned} \tag{8.6}$$

Portanto, dada a representação de Bézier de um polinômio F de multi-grau α

$$F(U) = \sum_{\Lambda \in \mathbb{H}^\alpha_\delta} b'_\Lambda B^\alpha_\Lambda(U)$$

podemos obter os coeficientes do mesmo polinômio em termos da base de multi-grau β

$$F(U) = \sum_{\Omega \in \mathbb{H}^\beta_\delta} b''_\Omega B^\beta_\Omega(U)$$

pela fórmula

$$b''_\Omega = \sum_{\Lambda \in \mathbb{H}^\alpha_\delta} K_{\Omega\Lambda} b'_\Lambda \tag{8.7}$$

onde

$$K_{\Omega\Lambda} = \binom{\beta}{\alpha}^{-1} \binom{\Omega}{\Lambda}$$

8.2.4 Diferenciação

Dados um natural m, uma multi-dimensão $\delta \in \mathbb{I}_m$ e um vetor $\Xi \in \mathbb{V}^\delta$, a derivada de ordem r do polinômio de Bernstein simploidal B^α_Λ na direção Ξ é

$$(D^r_\Xi B^\alpha_\Lambda)(U) \;=\; \left[\partial^r \left(t : \mathbb{R} \to \prod_{i=0}^{m} B^{\alpha_i}_{\Lambda_i}(U_i + t\,\Xi_i)\right)\right](0) \tag{8.8}$$

para qualquer $U \in \mathbb{A}^\delta$. Usando a fórmula de Leibnitz (4.2), obtemos

$$(D^r_\Xi B^\alpha_\Lambda)(U) \;=\; \sum_{\kappa \in \mathbb{I}^r_m} \frac{r!}{\kappa!} \prod_{i=0}^{m} \left[\partial^{\kappa_i}\left(t : \mathbb{R} \to B^{\alpha_i}_{\Lambda_i}(U_i + t\,\Xi_i)\right)\right](0)$$

Mais uma vez, se $\kappa_i > \alpha_i$ para algum i, o termo de dentro do somatório é igual a zero. Utilizando a fórmula (6.12) obtemos

$$\begin{aligned}(D^r_\Xi B^\alpha_\Lambda)(U) \;&=\; \sum_{\substack{\kappa \in \mathbb{I}^r_m \\ \kappa \le \alpha}} \frac{r!}{\kappa!} \prod_{i=0}^{m} \frac{\alpha_i!}{(\alpha_i - \kappa_i)!} \sum_{\substack{\lambda \in \mathbb{I}^{\kappa_i}_{\delta_i} \\ \lambda \le \Lambda_i}} B^{\kappa_i}_\lambda(\Xi_i)\;\; B^{\alpha_i - \kappa_i}_{\Lambda_i - \lambda}(U_i) \\ &=\; \sum_{\substack{\kappa \in \mathbb{I}^r_m \\ \kappa \le \alpha}} \frac{r!}{\kappa!}\, \frac{\alpha!}{(\alpha - \kappa)!} \sum_{\substack{\Psi \in \mathbb{H}^\kappa_\delta \\ \Psi \le \Lambda}} \prod_{i=0}^{m} B^{\kappa_i}_{\Psi_i}(\Xi_i)\;\; B^{\alpha_i - \kappa_i}_{\Lambda_i - \Psi_i}(U_i)\end{aligned}$$

Como na seção 6.2.4, se considerarmos a definição de B^α_Λ

$$B^\alpha_\Lambda(U) = \prod_{i=0}^{m} \frac{\alpha_i!}{\Lambda_i!} U_i^{\Lambda_i}$$

como válida em todo espaço $\mathbb{M}_{m,\delta}$, ao invés de apenas no espaço $\mathbb{A}^\delta$, podemos escrever

$$(D^r_\Xi B^\alpha_\Lambda)(U) \;=\; \sum_{\substack{\kappa \in \mathbb{I}^r_m \\ \kappa \le \alpha}} \frac{r!}{\kappa!}\, \frac{\alpha!}{(\alpha - \kappa)!} \sum_{\substack{\Psi \in \mathbb{H}^\kappa_\delta \\ \Psi \le \Lambda}} B^\kappa_\Psi(\Xi)\;\; B^{\alpha - \kappa}_{\Lambda - \Psi}(U) \tag{8.9}$$

Como pode-se notar, esta equação é a versão simploidal da equação 6.12. Entretanto, uma forma mais adequada para nossos propósitos é obtida realizando a elevação de multi-grau de $\alpha - \kappa$ para α:

$$(D^r_\Xi B^\alpha_\Lambda)(U) \;=\; \sum_{\substack{\kappa \in \mathbb{I}^r_m \\ \kappa \le \alpha}} \frac{r!}{\kappa!}\, \frac{\alpha!}{(\alpha - \kappa)!} \sum_{\substack{\Psi \in \mathbb{H}^\kappa_\delta \\ \Psi \le \Lambda}} B^\kappa_\Psi(\Xi) \sum_{\substack{\Omega \in \mathbb{H}^\alpha_\delta \\ \Omega \ge \Lambda - \Psi}} \binom{\Omega}{\Lambda - \Psi} \binom{\alpha}{\alpha - \kappa} B^\alpha_\Omega(U)$$

que pode ser reorganizada em

$$(D^r_\Xi B^\alpha_\Lambda)(U) \;=\; \sum_{\Omega\in\mathbb{H}^\alpha_\delta} \sum_{\substack{\kappa\in\mathbb{I}^r_m\\ \kappa\le\alpha}} \sum_{\substack{\Psi\in\mathbb{H}^\kappa_\delta\\ \Psi\le\Lambda\\ \Psi\ge\Lambda-\Omega}} r!\binom{\Omega}{\Lambda-\Psi} B^\kappa_\Psi(\Xi)\;\; B^\alpha_\Omega(U) \tag{8.10}$$

Portanto, dados os coeficientes de Bézier b'_Λ de um polinômio F, podemos obter os coeficientes de Bézier b''_Ω da derivada direcional $D^r_\Xi F$ pela fórmula

$$b''_\Omega = \sum_{\Lambda\in\mathbb{H}^\alpha_\delta} K_{\Omega\Lambda} b'_\Lambda \tag{8.11}$$

onde

$$K_{\Omega\Lambda} = \sum_{\substack{\kappa\in\mathbb{I}^r_m\\ \kappa\le\alpha}} \sum_{\substack{\Psi\in\mathbb{H}^\kappa_\delta\\ \Psi\le\Lambda\\ \Psi\ge\Lambda-\Omega}} r!\binom{\Omega}{\Lambda-\Psi}$$

8.3 Elementos de Bézier Simploidais

Um *elemento simploidal de Bézier de multi-dimensão δ e multi-grau $\alpha \in \mathbb{I}_{d-1}$ em $\mathbb{R}^d$* é a uma função $F : \mathbb{K}^\delta \to \mathbb{R}^m$ cujas m componentes são polinômios simploidais do espaço $\mathcal{P}^\alpha_\delta$ definidos em termos da representação de Bézier. Assim como o polinômio simploidal generaliza os polinômios simpliciais e tensoriais, o elemento simploidal de Bézier generaliza os elementos simpliciais e tensoriais. Veja a figura **??**.
Os pontos de controle de um elemento F são definidos de maneira análoga aos dos elementos simpliciais e tensoriais: o coeficiente de Bézier de hiper-índice Λ da componente F_i é a coordenada i do ponto de controle P_Λ de F.

A *grade de controle de Bézier* de F consiste dos seus pontos de controle P_Λ e dos segmentos de reta que ligam todos os pares de pontos P_Λ e P_Ω tais que Λ e Ω diferem em apenas dois elementos i, j da mesma linha k com $\Lambda_{ki} - \Omega_{ki} = +1$ e $\Lambda_{kj} - \Omega_{kj} = -1$.

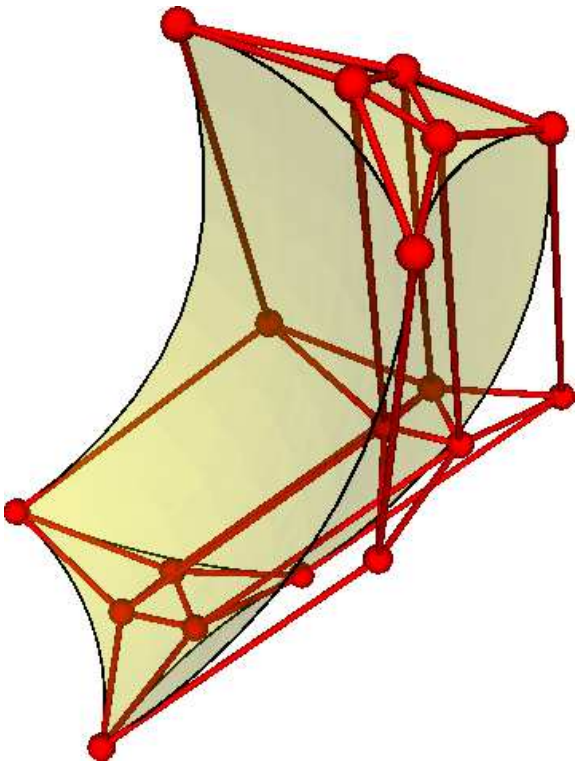


Figura 8.3: Um elemento de Bézier simploidal de multi-dimensão $(1, 2)$ (*um prisma de Bézier*) e multi-grau $(2, 3)$ mostrando seus pontos de controle e sua grade de Bézier (em vermelho).

Capítulo 9

Mapeamentos afins

Um *mapeamento afim* é uma função de um espaço $\mathbb{X} \subset \mathbb{R}^m$ para outro espaço $\mathbb{Y} \subset \mathbb{R}^n$ tal que cada coordenada do resultado é uma função afim (polinomial de primeiro grau) das coordenadas do argumento. Note que transformações lineares de $\mathbb{R}^m$ para $\mathbb{R}^n$ são casos particulares de mapeamentos afins [12].

Nesta tese, mapeamentos afins são importantes para a reparametrização de elementos de Bézier, em particular para descrever a colagem de blocos adjacentes de uma malha.

9.1 Mapeamento afim de $\mathbb{A}^\delta$ para $\mathbb{A}^\varepsilon$

No caso específico onde $\mathbb{X}$ e $\mathbb{Y}$ são espaços multi-afins de multi-dimensões $\delta \in \mathbb{I}_m$ e $\varepsilon \in \mathbb{I}_n$ respectivamente, um mapeamento afim é definido por constantes reais M_{ij}^{rs} onde $r \in \{0, \cdots, n\}$, $s \in \{0, \cdots, \varepsilon_r\}$, $i \in \{0, \cdots, m\}$ e $j \in \{0, \cdots, \delta_i\}$ tais que

$$(\Gamma(U))_{rs} = \sum_{i=0}^{m} \sum_{j=0}^{\delta_i} U_{ij} M_{ij}^{rs} \quad \text{para todo } U \in \mathbb{A}^\delta. \tag{9.1}$$

Note que, para que $\Gamma(U)$ pertença a $\mathbb{A}^\varepsilon$, é necessário que, para todo $r \in \{0, \cdots, n\}$:

$$\sum_{s=0}^{\varepsilon_r} (\Gamma(U))_{rs} = 1$$

Verifica-se que esta condição é equivalente a dizer que para todo $r \in \{0, \cdots, n\}$, existe um $w^r \in \mathbb{A}^m$ tal que

$$\sum_{s=0}^{\varepsilon_r} M_{ij}^{rs} = w_i^r \tag{9.2}$$

para todo $i = \{0, \cdots, m\}$ e todo $j \in \{0, \cdots, \delta_i\}$. Observe que a constante w_i^r define o peso da linha i de U na formação da linha r de $\Gamma(U)$. Observe também que a condição (9.2) implica que o lado esquerdo não depende de j.

Em particular, quando o domínio e/ou contra-domínio são espaços afins (isto é, $\delta = (d)$ e/ou $\varepsilon = (e)$) podemos suprimir o índice i e/ou o índice r dos coeficientes M_{ij}^{rs}, pois estes índices são sempre 0.

9.1.1 Mapeamento afim canônico de $\mathbb{A}^d$ para $\mathbb{A}^{1*d}$

Um caso particular muito comum de mapeamento afim é o mapeamento canônico η_d de $\mathbb{A}^d$ para $\mathbb{R}^d$ (definido na seção 7.1), que pode ser visto como um mapa de $\mathbb{X} = \mathbb{A}^{(d)}$ para $\mathbb{Y} = \mathbb{A}^{1*d}$. Neste caso, as constantes M_j^{rs} são

$$M_0^{rs} = \begin{cases} 1, & \text{se } r = s, \\ 0, & \text{se } r \neq s \end{cases} \quad \text{e} \quad M_1^{rs} = 1 - M_0^{rs} = \begin{cases} 0, & \text{se } r = s, \\ 1, & \text{se } r \neq s \end{cases}$$

9.2 Derivadas de um mapeamento afim

Seja Γ um mapeamento afim de $\mathbb{A}^\delta$ para $\mathbb{A}^\varepsilon$, definido pelas constantes M_{ij}^{rs} onde $r \in \{0, \cdots, n\}$, $s \in \{0, \cdots, \varepsilon_r\}$, $i \in \{0, \cdots, m\}$ e $j \in \{0, \cdots, \delta_i\}$; e seja Ξ uma direção de $\mathbb{A}^\delta$. A derivada (direcional) de Γ na direção Ξ é uma constante (independente de U) dada por

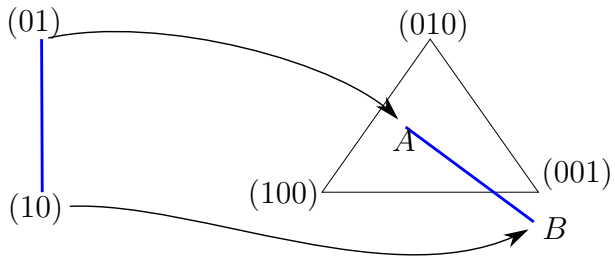
$$((D_\Xi \Gamma)(U))_{rs} = \sum_{i=0}^{m} \sum_{j=0}^{\delta_i} \Xi_{ij} M_{ij}^{rs} \tag{9.3}$$

Observe que, por esta razão, $\Gamma(\mathbb{K}^\delta|_{ij})$ é uma cópia transladada de $\Gamma(\mathbb{K}^\delta|_{ik})$ para todo $i \in \{0, \cdots, m\}$ e para todo $j, k \in \{0, \cdots, \delta_i\}$.

9.3 Exemplos

Neste seção, apresentamos alguns exemplos de mapas afins entre espaços multi-afins de dimensões variadas, e os respectivos coeficientes M_{ij}^{rs}.

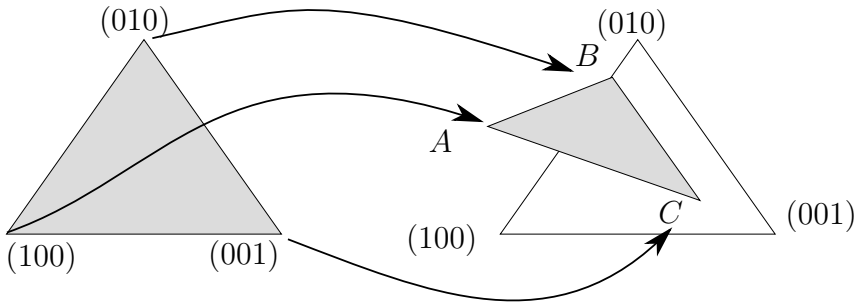
9.3.1 Mapeamento afim de $\mathbb{A}^{(1)}$ para $\mathbb{A}^{(2)}$



$$M_0^0 = 0.4 \quad M_1^0 = 0.8$$
$$M_0^1 = 0.5 \quad M_1^1 = -0.3$$
$$M_0^2 = 0.1 \quad M_1^2 = -0.5$$

Figura 9.1: Exemplo de um mapeamento afim de $\mathbb{A}^{(1)}$ para $\mathbb{A}^{(2)}$ (ou de $\mathbb{A}^1$ para $\mathbb{A}^2$).

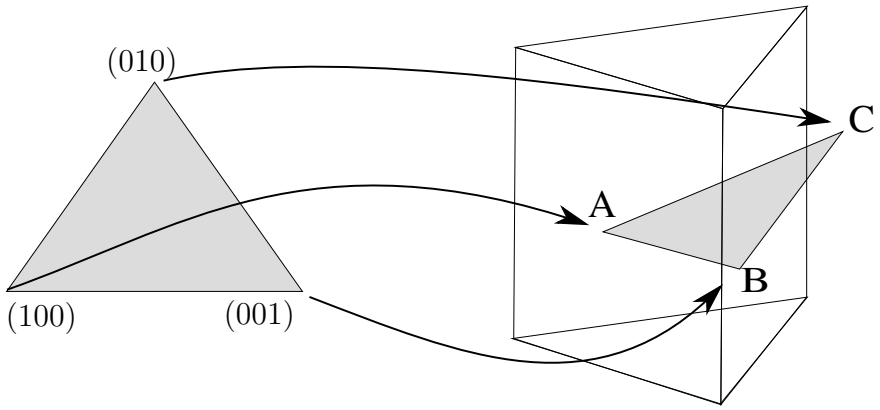
9.3.2 Mapeamento afim de $\mathbb{A}^{(2)}$ para $\mathbb{A}^{(2)}$



$$M_0^0 = 0.7 \quad M_1^0 = 0.2 \quad M_2^0 = 0.2$$
$$M_0^0 = 0.6 \quad M_1^0 = 0.8 \quad M_2^0 = 0.2$$
$$M_0^0 = -0.3 \quad M_1^0 = 0.0 \quad M_2^0 = 0.6$$

Figura 9.2: Exemplo de mapeamento afim de $\mathbb{A}^{(2)}$ para $\mathbb{A}^{(2)}$ (ou de $\mathbb{A}^2$ para $\mathbb{A}^2$).

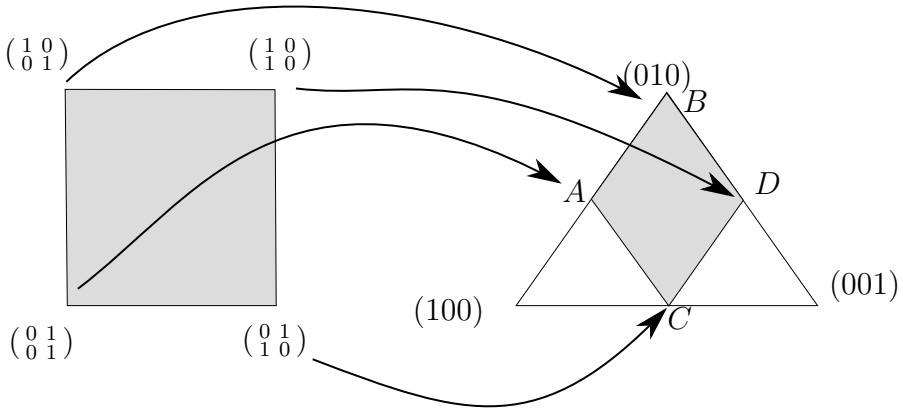
9.3.3 Mapeamento afim de $\mathbb{A}^{(2)}$ para $\mathbb{A}^{(2,1)}$



$$
\begin{array}{lll}
M_0^{0,0} = 0.5 & M_1^{0,0} = 0.1 & M_2^{0,0} = 0.7 \\
M_0^{1,0} = 0.5 & M_1^{1,0} = 0.4 & M_2^{1,0} = -0.4 \\
M_0^{2,0} = 0.0 & M_1^{2,0} = 0.5 & M_2^{2,0} = 0.7 \\
M_0^{0,1} = 0.5 & M_1^{0,1} = 0.4 & M_2^{0,1} = 0.6 \\
M_0^{1,1} = 0.5 & M_1^{1,1} = 0.6 & M_2^{1,1} = 0.4
\end{array}
$$

Figura 9.3: Exemplo de um mapeamento afim de $\mathbb{A}^{(2)}$ (ou $\mathbb{A}^2$) para $\mathbb{A}^{(2,1)}$.

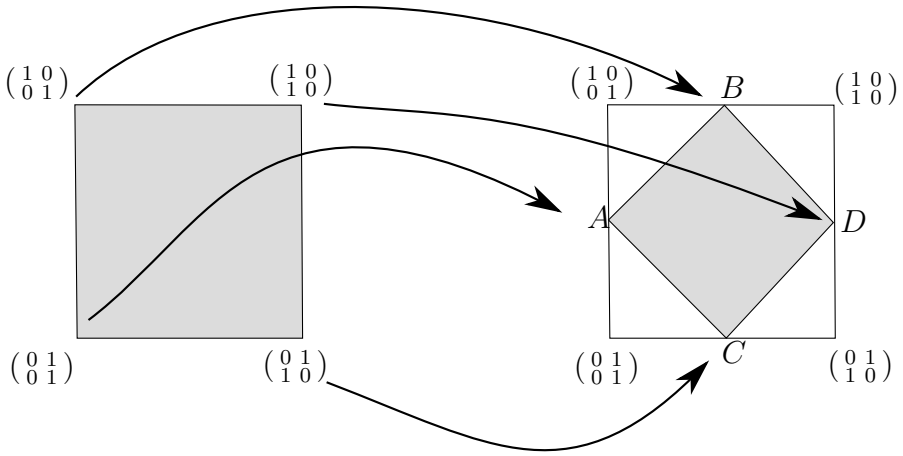
9.3.4 Mapeamento afim de $\mathbb{A}^{(1,1)}$ para $\mathbb{A}^{(2)}$



$$\begin{array}{llll} M^0_{0,0}=0.0 & M^0_{0,1}=0.5 & M^0_{1,0}=0.0 & M^0_{1,1}=0.0 \\ M^1_{0,0}=0.0 & M^1_{0,1}=0.0 & M^1_{1,0}=0.5 & M^1_{1,1}=0.0 \\ M^2_{0,0}=0.5 & M^2_{0,1}=0.0 & M^2_{1,0}=0.0 & M^1_{1,1}=0.5 \end{array}$$

Figura 9.4: Exemplo de um mapeamento afim de $\mathbb{A}^{(1,1)}$ para $\mathbb{A}^{(2)}$ (ou $\mathbb{A}^2$). Note que os segmentos AB e CD, que são imagens das facetas $\mathbb{K}^{1,1}|_{1,0}$ e $\mathbb{K}^{1,1}|_{1,1}$, são paralelas entre si.

9.3.5 Mapeamento afim de $\mathbb{A}^{(1,1)}$ para $\mathbb{A}^{(1,1)}$



$$\begin{array}{llll} M^{0,0}_{0,0} = 0.5 & M^{0,0}_{0,1} = 0.0 & M^{0,0}_{1,0} = 0.5 & M^{0,0}_{1,1} = 0.0 \\ M^{0,1}_{0,0} = 0.0 & M^{0,1}_{0,1} = 0.5 & M^{0,1}_{1,0} = 0.0 & M^{0,1}_{1,1} = 0.5 \\ M^{1,0}_{0,0} = 0.0 & M^{1,0}_{0,1} = 0.5 & M^{1,0}_{1,0} = 0.5 & M^{1,0}_{1,1} = 0.0 \\ M^{1,1}_{0,0} = 0.5 & M^{1,1}_{0,1} = 0.0 & M^{1,1}_{1,0} = 0.0 & M^{1,1}_{1,1} = 0.5 \end{array}$$

Figura 9.5: Exemplo de um mapeamento afim de $\mathbb{A}^{(1,1)}$ para $\mathbb{A}^{(1,1)}$.

Capítulo 10

Reparametrização de elementos de Bézier

Neste capítulo, consideraremos o problema da *reparametrização afim* de elementos simploidais de Bézier.

Este problema inclui vários casos particulares importantes, como a extração de facetas e a subdivisão de um bloco de Bézier (algoritmo de DeCasteljau generalizado). Por exemplo, considere um elemento de Bézier de multi-grau $(5, 7)$ cujo domínio é o prisma canônico, $\mathbb{K}^{(1,2)}$. Veja a figura 10.1.

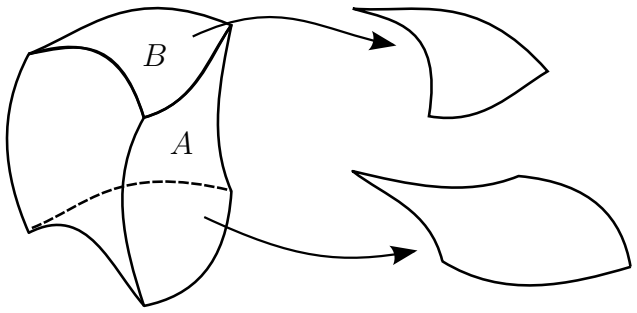


Figura 10.1: Bloco de Bézier de domínio prismático

A extração da face A deste retalho produzirá um retalho de Bézier de domínio $K^{(1,1)}$ e multi-grau $(5, 7)$. A extração da face B, por outro lado, produzirá um retalho de Bézier de domínio $\mathbb{K}^{(2)}$ e multi-grau (7). Esta operação é uma ferramenta importante para a colagem não conforme de retalhos simploidais, como veremos no capítulo 12.

10.1 Reparametrização afim

Seja F uma função de $\mathbb{A}^\delta$ para $\mathbb{R}$, e Γ um mapeamento afim de outro espaço multi-afim $\mathbb{A}^\varepsilon$ para $\mathbb{A}^\delta$. Definimos a *reparametrização de F por* Γ como a composição $F \circ \Gamma$, que é uma função de $\mathbb{A}^2$ para $\mathbb{R}$. Veja a figura 10.2.
Se F é um polinômio simploidal de multigrau α, verifica-se que $G = F \circ \Gamma$ também é um polinômio simploidal, cujo multi-grau β depende de α, δ e ε.

Neste capítulo, consideramos o problema de calcular a representação de Bézier de G a partir da representação de Bézier de F e do mapa Γ. Especificamente, dados os coeficientes de Bézier b'_Λ de um polinômio simploidal $F \in \mathcal{P}^\alpha_\delta$,

$$F(U) = \sum_{\Lambda \in \mathbb{H}^\alpha_\delta} b'_\Lambda B^\alpha_\Lambda(U) \quad \text{para todo } U \in \mathbb{A}^\delta$$

e um mapeamento afim Γ de outro espaço multi-afim $\mathbb{A}^\varepsilon$ para $\mathbb{A}^\delta$, deseja-se encontrar o multi-grau β e os coeficientes b''_Ω tais que

$$G(V) = F(\Gamma(V)) = \sum_{\Omega \in \mathbb{H}^\beta_\varepsilon} b''_\Omega B^\beta_\Omega(V) \quad \text{para todo } V \in \mathbb{A}^\varepsilon$$

Como veremos, os coeficientes b''_Ω são funções lineares dos coeficientes b'_Λ. Portanto a relação entre os coeficientes b'_Λ e b''_Ω pode ser resumida por uma matriz de mudança de base $K^\alpha_{\Omega\Lambda}$:

$$b''_\Omega = \sum_{\Lambda \in \mathbb{H}^\alpha_\delta} K^\alpha_{\Omega\Lambda} b'_\Lambda$$

onde $K^\alpha_{\Omega\Lambda}$ é o coeficiente de B^β_Ω na representação de Bézier de $B^\alpha_\Lambda \circ \Gamma$.

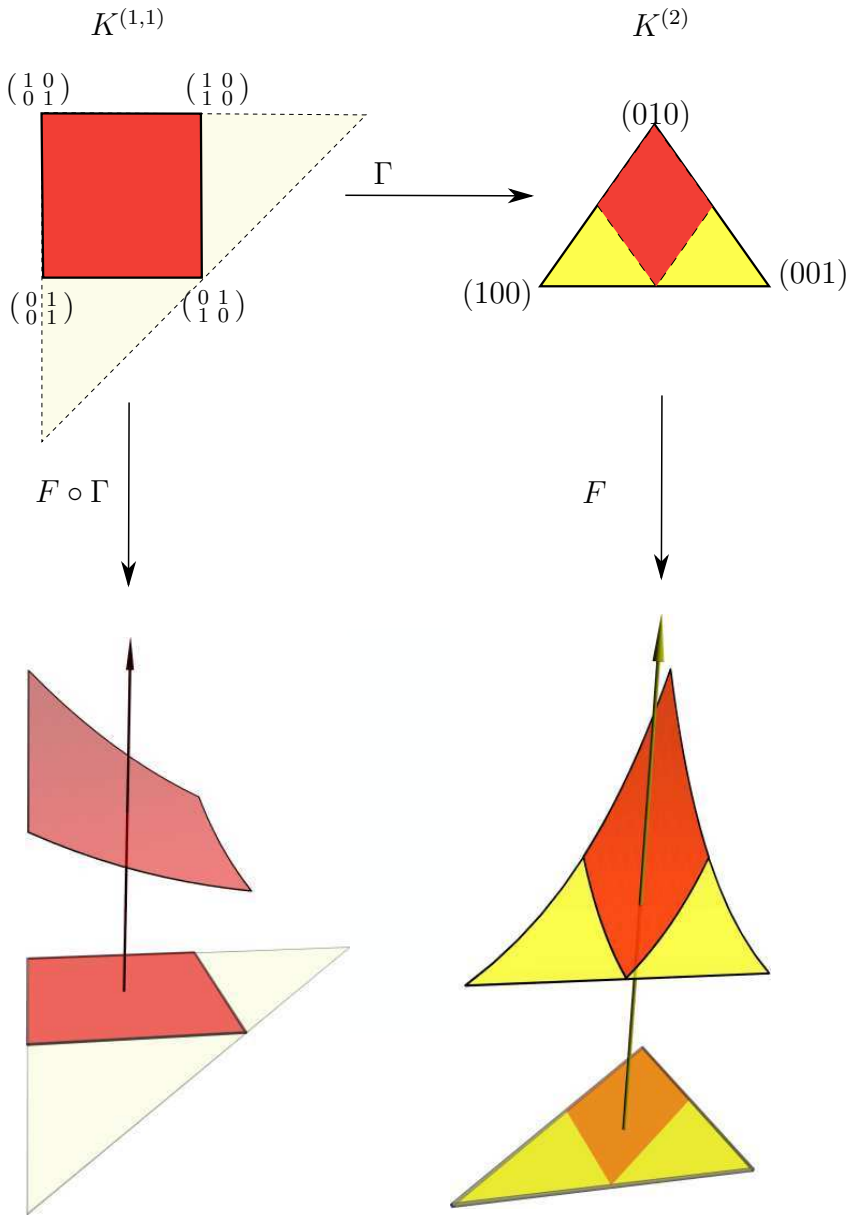


Figura 10.2: Uma função F, de $\mathbb{A}^{(2)}$ para $\mathbb{R}$, restrita ao $\mathbb{K}^{(2)}$ (em amarelo) e sua reparametrização pelo mapeamento afim Γ de $\mathbb{A}^{(1,1)}$ para $\mathbb{A}^{(2)}$, da seção 9.3.4, resultando na função $F \circ \Gamma$, de $\mathbb{A}^{(1,1)}$ para $\mathbb{R}$, restrita ao $\mathbb{K}^{(1,1)}$ (em vermelho).

10.2 Mapeamento de simploidal para simplicial

Consideramos, inicialmente, o caso particular onde Γ é um mapeamento de um espaço multi-afim para um espaço afim. Seja Γ um mapeamento de $\mathbb{A}^\delta$ para $\mathbb{A}^e$ com $\delta \in \mathbb{I}_m$ e $m, e \in \mathbb{N}$. Pelas fórmulas (8.5) e (9.1), para todo grau $g \in \mathbb{N}$ e todo multi-índice $\kappa \in \mathbb{I}_d^g$, a composição $B_\kappa^g \circ \Gamma$ é dada por

$$B_\kappa^g(\Gamma(U)) \;=\; \frac{g!}{\kappa!}\prod_{s=0}^{e}\left(\sum_{i=0}^{m}\sum_{j=0}^{\delta_i} U_{ij}M_{ij}^s\right)^{\kappa_s}$$

Podemos expandir este produtório em um somatório sobre um multi-índice μ, um hiper-índice Ω e um ultra-índice $\mathbf{\Lambda}$. Na fórmula abaixo, M^s é a matriz irregular tal que $(M^s)_{ij} = M_{ij}^r$ para todo $i \in \{0, \cdots, m\}$ e todo $j \in \{0, \cdots, \delta_i\}$:

$$\begin{aligned}
B_\kappa^g(\Gamma(U)) &= \frac{g!}{\kappa!}\sum_{\mu\in\mathbb{I}_m^g}\sum_{\Omega\in\mathbb{H}_\delta^\mu}\sum_{\mathbf{\Lambda}\in\mathbb{U}_{e,\delta}^\Omega}\prod_{s=0}^{e}\frac{\kappa_s!}{\mathbf{\Lambda}_s!}\prod_{i=0}^{m}\prod_{j=0}^{\delta_i}(U_{ij}M_{ij}^s)^{\mathbf{\Lambda}_{s,i,j}} \\
&= \sum_{\mu\in\mathbb{I}_m^g}\sum_{\Omega\in\mathbb{H}_\delta^\mu}\sum_{\mathbf{\Lambda}\in\mathbb{U}_{e,\delta}^\Omega}\frac{g!}{\mathbf{\Lambda}!}\left(\prod_{s=0}^{e}\prod_{i=0}^{m}\prod_{j=0}^{\delta_i}(M_{ij}^s)^{\mathbf{\Lambda}_{s,i,j}}\right)\prod_{i=0}^{m}\prod_{j=0}^{\delta_i}\prod_{s=0}^{e}(U_{ij})^{\mathbf{\Lambda}_{s,i,j}} \\
&= \sum_{\mu\in\mathbb{I}_m^g}\sum_{\Omega\in\mathbb{H}_\delta^\mu}\sum_{\mathbf{\Lambda}\in\mathbb{U}_{e,\delta}^\Omega}\frac{g!}{\mathbf{\Lambda}!}\left(\prod_{s=0}^{e}(M^s)^{\mathbf{\Lambda}_s}\right)\prod_{i=0}^{m}\prod_{j=0}^{\delta_i}U_{ij}^{\Omega_{i,j}} \\
&= \sum_{\mu\in\mathbb{I}_m^g}\sum_{\Omega\in\mathbb{H}_\delta^\mu}\sum_{\mathbf{\Lambda}\in\mathbb{U}_{e,\delta}^\Omega}\frac{g!\Omega!}{\mathbf{\Lambda}!\mu!}\left(\prod_{s=0}^{e}(M^s)^{\mathbf{\Lambda}_s}\right)\prod_{i=0}^{m}\frac{\mu_i}{\Omega_i}U_i^{\Omega_i} \\
&= \sum_{\mu\in\mathbb{I}_m^g}\sum_{\Omega\in\mathbb{H}_\delta^\mu}\left(\sum_{\mathbf{\Lambda}\in\mathbb{U}_{e,\delta}^\Omega}\frac{g!\Omega!}{\mathbf{\Lambda}!\mu!}\prod_{s=0}^{e}(M^s)^{\mathbf{\Lambda}_s}\right)B_\Omega^\mu(U) \qquad (10.1)
\end{aligned}$$

Para a simplificar a notação, podemos escrever a equação (10.1) como

$$B_\kappa^g(\Gamma(U)) \;=\; \sum_{\mu\in\mathbb{I}_m^g}\sum_{\Omega\in\mathbb{H}_\delta^\mu}\frac{g!}{\mu!}\mathcal{G}(e,\kappa,\delta,\Omega,M)B_\Omega^\mu(U) \qquad (10.2)$$

onde M é a coleção das matrizes irregulares M^s para todo $s \in \{0, \cdots, e\}$ e

$$\mathcal{G}(e,\kappa,\delta,\Omega,M) \;=\; \sum_{\mathbf{\Lambda}\in\mathbb{U}_{e,\delta}^\Omega}\frac{\Omega!}{\mathbf{\Lambda}!}\prod_{s=0}^{e}(M^s)^{\mathbf{\Lambda}_s} \qquad (10.3)$$

Note que equação (10.2) generaliza a formula (7.4), que realiza a conversão de polinômios de Bernstein tensoriais para simpliciais.

10.3 Reparametrização afim genérica

Vamos considerar agora o caso geral onde Γ é um mapeamento afim de um espaço multi-afim $\mathbb{A}^{\delta}$ para outro espaço multi-afim $\mathbb{A}^{\varepsilon}$, com $\delta \in \mathbb{I}_m$, $\varepsilon \in \mathbb{I}_n$ e $m, n \in \mathbb{N}$. Pela definição de polinômio de Bernstein simploidal (8.5), para todo multi-grau $\alpha \in \mathbb{I}_n$ e todo hiper-índice $\Lambda \in \mathbb{H}^{\alpha}_{\varepsilon}$, a composição $B^{\alpha}_{\Lambda} \circ \Gamma$ é dada por

$$B^{\alpha}_{\Lambda}(\Gamma(U)) \;=\; \prod_{r=0}^{n} B^{\alpha_r}_{\Lambda_r}((\Gamma(U))_r).$$

Utilizando a equação (10.2) obtemos

$$\begin{aligned}
B^{\alpha}_{\Lambda}(\Gamma(U)) &= \prod_{r=0}^{n} \sum_{\mu \in \mathbb{I}_m^{\alpha_r}} \frac{\alpha_r!}{\mu!} \sum_{\Omega \in \mathbb{H}^{\mu}_{\delta}} \mathcal{G}(\varepsilon_r, \Lambda_r, \delta, \Omega, M^r) B^{\mu}_{\Omega}(U) \\
&= \sum_{\nu \in \mathbb{I}_m^{|\alpha|}} \sum_{\Psi \in \mathbb{H}^{\alpha,\nu}_{n,m}} \sum_{\Upsilon \in \mathbb{H}^{\nu}_{\delta}} \sum_{\mathbf{\Lambda} \in \mathbb{U}^{\Upsilon}_{n,\delta}} \prod_{r=0}^{n} \frac{\alpha_r!}{\Psi_r!} \mathcal{G}(\varepsilon_r, \Lambda_r, \delta, \mathbf{\Lambda}_r, M^r) B^{\Psi_r}_{\mathbf{\Lambda}_r}(U) \\
&= \sum_{\nu \in \mathbb{I}_m^{|\alpha|}} \sum_{\Psi \in \mathbb{H}^{\alpha,\nu}_{n,m}} \sum_{\Upsilon \in \mathbb{H}^{\nu}_{\delta}} \sum_{\mathbf{\Lambda} \in \mathbb{U}^{\Upsilon}_{n,\delta}} \frac{\alpha!}{\Psi!} \left(\prod_{r=0}^{n} \mathcal{G}(\varepsilon_r, \Lambda_r, \delta, \mathbf{\Lambda}_r, M^r) \right) \\
&\qquad \prod_{r=0}^{n} \prod_{i=0}^{m} B^{\Psi_{r,i}}_{\mathbf{\Lambda}_{r,i}}(U_i) \\
&= \sum_{\nu \in \mathbb{I}_m^{|\alpha|}} \sum_{\Psi \in \mathbb{H}^{\alpha,\nu}_{n,m}} \sum_{\Upsilon \in \mathbb{H}^{\nu}_{\delta}} \sum_{\mathbf{\Lambda} \in \mathbb{U}^{\Upsilon}_{n,\delta}} \frac{\alpha!}{\Psi!} \left(\prod_{r=0}^{n} \mathcal{G}(\varepsilon_r, \Lambda_r, \delta, \mathbf{\Lambda}_r, M^r) \right) \\
&\qquad \prod_{i=0}^{m} \prod_{r=0}^{n} B^{\Psi_{r,i}}_{\mathbf{\Lambda}_{r,i}}(U_i)
\end{aligned} \tag{10.4}$$

Utilizando equação (7.3) de produto de polinômios de Bernstein simpliciais, obtemos

$$\begin{aligned} B_\Lambda^\alpha(\Gamma(U)) &= \sum_{\nu\in\mathbb{I}_m^{|\alpha|}}\sum_{\Psi\in\mathbb{H}_{n,m}^{\alpha,\nu}}\sum_{\Upsilon\in\mathbb{H}_\delta^\nu}\sum_{\mathbf{\Lambda}\in\mathbb{U}_{n,\delta}^\Upsilon}\frac{\alpha!}{\Psi!}\left(\prod_{r=0}^{n}\mathcal{G}(\varepsilon_r,\Lambda_r,\delta,\mathbf{\Lambda}_r,M^r)\right) \\ &\qquad \prod_{i=0}^{m}B_{\Upsilon_i}^{\nu_i}(U_i)\frac{\Upsilon_i!}{\nu_i!}\prod_{r=0}^{n}\frac{\Psi_{r,i}!}{\mathbf{\Lambda}_{r,i}!} \\ &= \sum_{\nu\in\mathbb{I}_m^{|\alpha|}}\sum_{\Psi\in\mathbb{H}_{n,m}^{\alpha,\nu}}\sum_{\Upsilon\in\mathbb{H}_\delta^\nu}\sum_{\mathbf{\Lambda}\in\mathbb{U}_{n,\delta}^\Upsilon}\frac{\alpha!\Upsilon!}{\Psi!\nu!}\left(\prod_{r=0}^{n}\mathcal{G}(\varepsilon_r,\Lambda_r,\delta,\mathbf{\Lambda}_r,M^r)\right) \\ &\qquad B_\Upsilon^\nu(U)\prod_{i=0}^{m}\prod_{r=0}^{n}\frac{\Psi_{r,i}!}{\mathbf{\Lambda}_{r,i}!} \\ &= \sum_{\nu\in\mathbb{I}_m^{|\alpha|}}\sum_{\Upsilon\in\mathbb{H}_\delta^\nu}\left(\sum_{\mathbf{\Lambda}\in\mathbb{U}_{n,\delta}^\Upsilon}\frac{\alpha!\Upsilon!}{\mathbf{\Lambda}!\nu!}\prod_{r=0}^{n}\mathcal{G}(\varepsilon_r,\Lambda_r,\delta,\mathbf{\Lambda}_r,M^r)\right)B_\Upsilon^\nu(U) \end{aligned}$$

que pode ser reescrito como

$$B_\Lambda^\alpha(\Gamma(U)) = \sum_{\nu\in\mathbb{I}_m^{|\alpha|}}\sum_{\Upsilon\in\mathbb{H}_\delta^\nu}\frac{\alpha!\Upsilon!}{\nu!}\mathcal{H}(\delta,\varepsilon,\alpha,\Lambda,\Upsilon,M)B_\Upsilon^\nu(U) \tag{10.5}$$

onde

$$\mathcal{H}(\delta,\varepsilon,\alpha,\Lambda,\Upsilon,M) = \sum_{\mathbf{\Lambda}\in\mathbb{U}_{n,\delta}^\Upsilon}\frac{1}{\mathbf{\Lambda}!}\prod_{r=0}^{n}\mathcal{G}(\varepsilon_r,\Lambda_r,\delta,\mathbf{\Lambda}_r,M^r)$$

Na fórmula (10.5), o lado direito envolve polinômios de Bernstein de multi-graus diferentes. Utilizamos então a equação (8.6) para elevar cada multi-grau ν para o multi-grau $\beta = |\alpha| * d$. Obtemos assim

$$\begin{aligned} B_\Lambda^\alpha(\Gamma(U)) &= \sum_{\nu\in\mathbb{I}_m^{|\alpha|}}\sum_{\Upsilon\in\mathbb{H}_\delta^\nu}\frac{\alpha!\Upsilon!}{\nu!}\mathcal{H}(\delta,\varepsilon,\alpha,\Lambda,\Upsilon,M)B_\Upsilon^\nu(U) \\ &= \sum_{\nu\in\mathbb{I}_m^{|\alpha|}}\sum_{\Upsilon\in\mathbb{H}_\delta^\nu}\frac{\alpha!\Upsilon!}{\nu!}\mathcal{H}(\delta,\varepsilon,\alpha,\Lambda,\Upsilon,M)\sum_{\substack{\Omega\in\mathbb{H}_{m,\delta}^\beta\\ \Omega\geq\Upsilon}}\frac{\binom{\Omega}{\Upsilon}}{\binom{\beta}{\nu}}B_\Omega^\beta(U) \\ &= \sum_{\Omega\in\mathbb{H}_{m,\delta}^\beta}\sum_{\nu\in\mathbb{I}_m^{|\alpha|}}\sum_{\substack{\Upsilon\in\mathbb{H}_\delta^\nu\\ \Omega\geq\Upsilon}}\frac{\alpha!\Upsilon!}{\nu!}\frac{\binom{\Omega}{\Upsilon}\mathcal{H}(\delta,\varepsilon,\alpha,\Lambda,\Upsilon,M)}{\binom{\beta}{\nu}}B_\Omega^\beta(U) \end{aligned} \tag{10.6}$$

Portanto, temos

$$K_{\Omega\Lambda}(\Gamma) = \sum_{\nu\in\mathbb{I}_m^{|\alpha|}} \sum_{\substack{\Upsilon\in\mathbb{H}_\delta^\nu \\ \Omega\geq\Upsilon}} \frac{\alpha!\Upsilon!}{\nu!} \frac{\binom{\Omega}{\Upsilon}\mathcal{H}(\delta,\varepsilon,\alpha,\Lambda,\Upsilon,M)}{\binom{\beta}{\nu}} \tag{10.7}$$

10.4 Derivada direcional da reparametrização

Seja F um polinômio simploidal de multi-dimensão δ e Γ um mapeamento afim $\Gamma : \mathbb{A}^\varepsilon \to \mathbb{A}^\delta$. A derivada de $G = F \circ \Gamma$ na direção $\Xi \in \mathbb{V}^\varepsilon$, avaliada no ponto $V \in \mathbb{A}^\varepsilon$, é dada por

$$\begin{aligned} D_\Xi G(V) &= (D_\Xi(F\circ\Gamma))(V) \\ &= [\partial(s:\mathbb{R}\to(F\circ\Gamma)(V+s\Xi))](0) \end{aligned}$$

aplicando a regra da cadeia e a fórmula (9.3) da derivada direcional de Γ, obtemos

$$\begin{aligned} D_\Xi G(V) &= \sum_{r=0}^{m}\sum_{s=0}^{\varepsilon_r}(\partial_{rs}F)(\Gamma(V))\left([\partial(t:\mathbb{R}\to(\Gamma\left(V+t\Xi\right))_{rs})]\left(0\right)\right) \\ &= \sum_{r=0}^{m}\sum_{s=0}^{\varepsilon_r}(\partial_{rs}F)(\Gamma(V))\sum_{i=0}^{m}\sum_{j=0}^{\delta_i}M_{ij}^{rs}\Xi_{ij} \end{aligned}$$

Seja então Θ o vetor de $\mathbb{V}^\varepsilon$ definido por

$$\Theta_{rs} = \sum_{i=0}^{m}\sum_{j=0}^{\delta_i}M_{ij}^{rs}\Xi_{ij}$$

a equação acima pode ser reescrita como

$$D_\Xi G(V) = (D_\Theta F)(\Gamma(V)) \tag{10.8}$$

Verifica-se que as derivadas direcionais de ordem maior também satisfazem a relação (10.8). Mais especificamente, para qualquer ordem r,

$$D_\Xi^r G(V) = D_\Theta^r F(\Gamma(V)) \tag{10.9}$$

Capítulo 11

Aspectos gerais da implementação

As malhas usadas em modelagem geológica podem ter centenas de milhares ou milhões de elementos simploidais de Bézier. Em teoria, a forma (e outras propriedades) de cada bloco podem ser especificada pelos seus pontos de controle de Bézier. Entretanto, as restrições de continuidade, suavidade e geometria, tipicamente presentes nessas malhas, tornam esta representação redundante e muito perdulária em termos de espaço. Neste capítulo, descrevemos as técnicas utilizadas na biblioteca BezEl para representar tais modelos de maneira mais econômica.

11.1 Elementos de Bézier

Um modelo geológico geralmente utiliza um repertório bem pequeno de tipos distintos de blocos de Bézier. Por exemplo, a figura 11.1 mostra duas camadas geológicas compostas por um único tipo de bloco de Bézier (hexaedros), com superfícies superior e inferior onduladas mas com paredes verticais planas, arranjadas de forma que suas projeções no plano horizontal formam uma grade retangular.

Em princípio, podemos modelar este tipo de bloco por um elemento tensorial de Bézier de dimensão 3 e multi-grau $(3, 3, 1)$ no $\mathbb{R}^3$. Ou seja, o domínio é o cubo $\mathbb{K}^{(1,1,1)}$ e cada componente X,Y,Z da imagem é um polinômio nas variáveis u,v e w do domínio com grau 3 em u e em v e grau 1 em w. Nesta representação, são necessários 96 parâmetros para expressar a forma do bloco: 32 coeficientes de Bézier para cada componente X,Y e Z, ou seja, 32 pontos de controle de Bézier, veja a figura 11.2

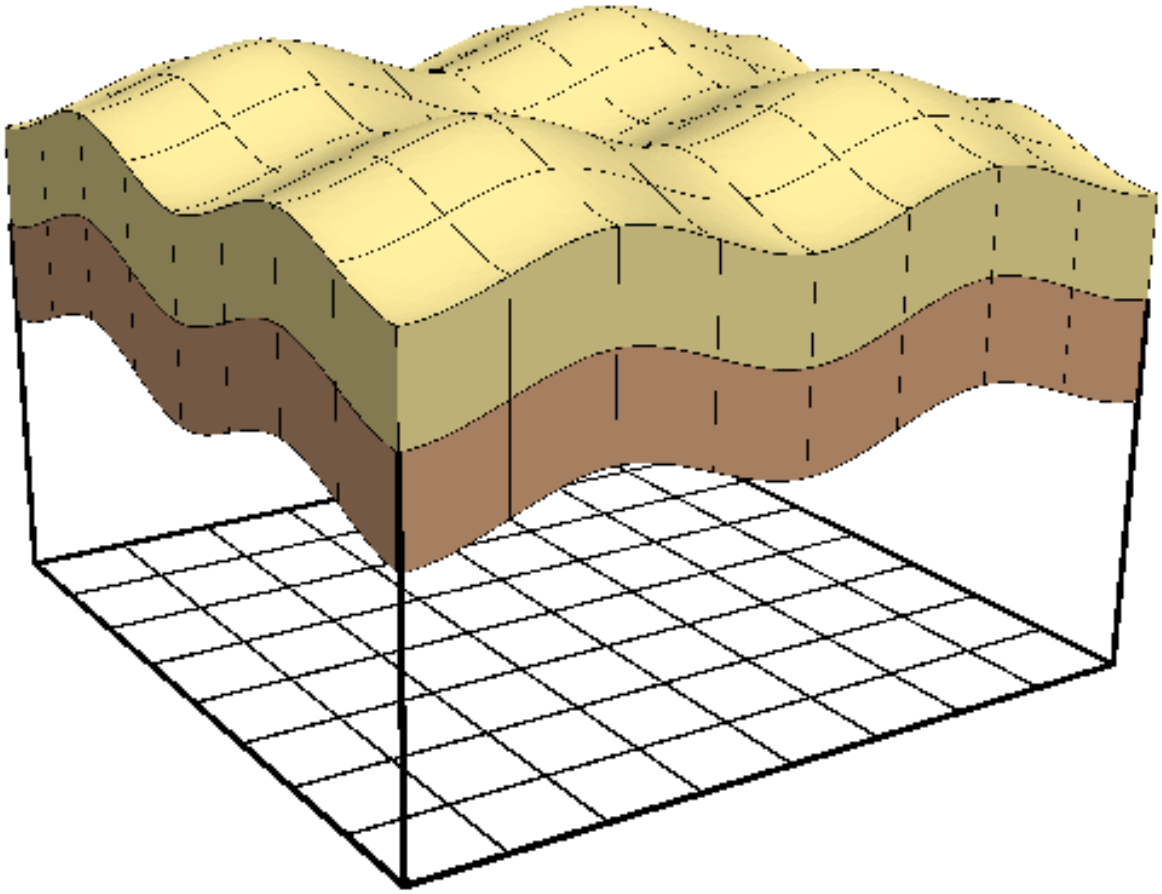


Figura 11.1: Exemplo de modelo geológico.

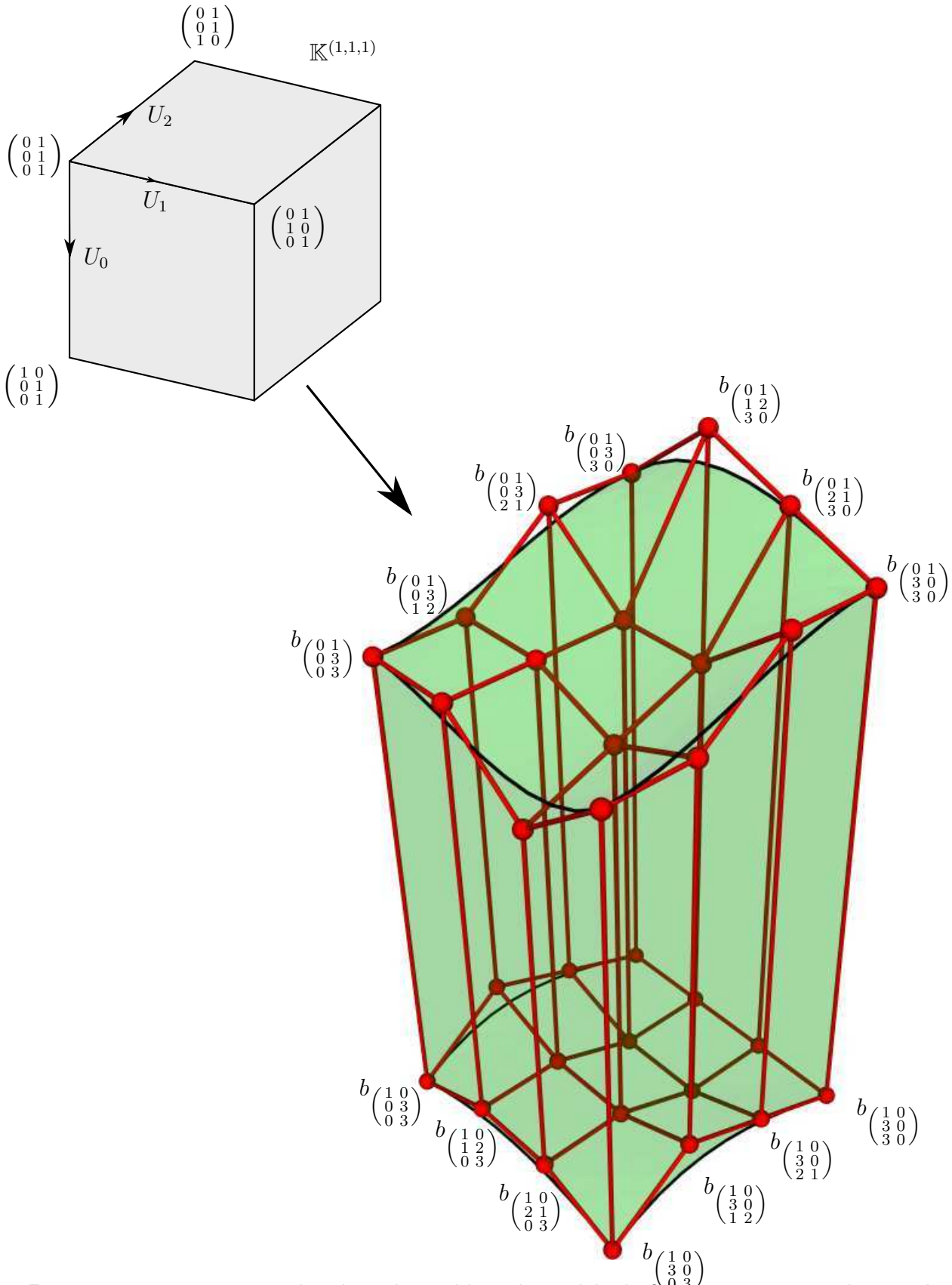


Figura 11.2: Representação de Bézier de um bloco do modelo da figura 11.1, mostrando a grade de controle de Bézier (em vermelho) e, para alguns pontos, os respectivos índices.

11.2 Blocos de Bézier generalizados

Observe que este tipo de bloco é muito mais geral do que o necessário para construir a malha da figura 11.1, pois inclui blocos com paredes curvas e inclinadas, como ilustrado na figura 11.3.

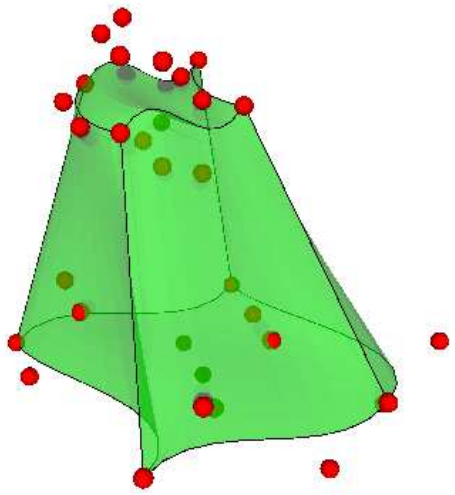


Figura 11.3: Outro exemplo de um bloco de Bézier de multi-dimensão $(1, 1, 1)$ e multi-graus $(3, 3, 1)$.

Uma vez que, nessa malha, a projeção de cada bloco no plano horizontal é sempre um retângulo alinhado com os eixos, a componente X pode ser modelada por uma função linear da coordenada u do domínio apenas. Da mesma forma, a componente Y pode ser uma função linear da coordenada v apenas. Somente a componente Z precisa ser uma função linear de w e cúbica em u e v. Portanto, podemos modelar cada bloco da malha por três funções simploidais X, Y e Z com o mesmo domínio $\mathbb{K}^{(1,1,1)}$ mas multi-graus diferentes — $(1, 0, 0)$,$(0, 1, 0)$ e $(3, 3, 1)$, respectivamente. Veja a figura 11.4.

Para especificar estas funções basta fornecer dois parâmetros para X (as coordenadas x_{min} e x_{max}, que são os coeficientes de Bézier $b^X_{((0,1),(0,0),(0,0))}$ e $b^X_{((1,0),(0,0),(0,0))}$ desta função), dois parâmetros para Y (y_{min} e y_{max}, ou $b^Y_{((0,0),(0,1),(0,0))}$ e $b^Y_{((0,0),(1,0),(0,0))}$) e 32 parâmetros para Z (os coeficientes b^Z_Λ onde $\Lambda \in \mathbb{H}^{(3,3,1)}_{(1,1,1)}$). Veja figura 11.4. Note que estes últimos são os 16 coeficientes da face superior e os 16 coeficientes de Bézier da face inferior. No total, são apenas $2 + 2 + 32 = 36$ parâmetros, uma economia de memória de aproximadamente 63%.

Por conveniência, neste capítulo usaremos a notação $\mathtt{B}.X$, $\mathtt{B}.Y$ e $\mathtt{B}.Z$ para indicar as componentes do bloco de Bézier generalizado $\mathtt{B}$ que representam a geometria; e analogamente para outras componentes que representam as propriedades físicas. Além disso,

denotaremos por `B.P`(U) o ponto de coordenadas (`B.X`(U), `B.Y`(U), `B.Z`(U)). Na biblioteca, estas componentes são identificadas por índices 0,1,2,etc; e acessadas através do método `B.eval(k,U)` que calcula a componente `k` para o ponto `U` do domínio $\mathbb{A}^\delta$

Considerando este exemplo, na biblioteca BezEl, optamos por representar cada célula de uma malha por um *bloco de Bézier generalizado*, que é uma lista de funções polinomiais simploidais de mesmo domínio mas de multi-graus arbitrários e independentes.

Cada bloco é uma instância da classe `Block` e é de um determinado tipo. Este tipo é descrito por um objeto separado (uma instância da classe `BlockKind`), que armazena informações como a dimensão do contra-domínio, a multi-dimensão e os multi-graus. A figura 11.5 ilustra esta representação. Nesta figura, os blocos `B0` e `B1` são do mesmo tipo `C0`. O bloco `B2`, por outro lado, é do tipo `C1`. Note que o número e significado dos parâmetros `params` de cada bloco são determinados pelo seu tipo. Por conveniência, usaremos a notação `B`.α para indicar o parâmetro α do bloco `B`.

Esta separação permite uma representação eficiente de malhas que possuem muitos blocos de um mesmo tipo, como a da figura 11.1. O tipo destes blocos é descrito por uma instância `brickKind` da classe `BlockKind`, com dimensão do contra-domínio, o parâmetro `brickKind.rD`, 3, a multi-dimensão do domínio, `brickKind.dD`, com valor $(1, 1, 1)$ e multigraus, parâmetro `brickKind.deg`, $((1, 0, 0), (0, 1, 0), (3, 3, 1))$.

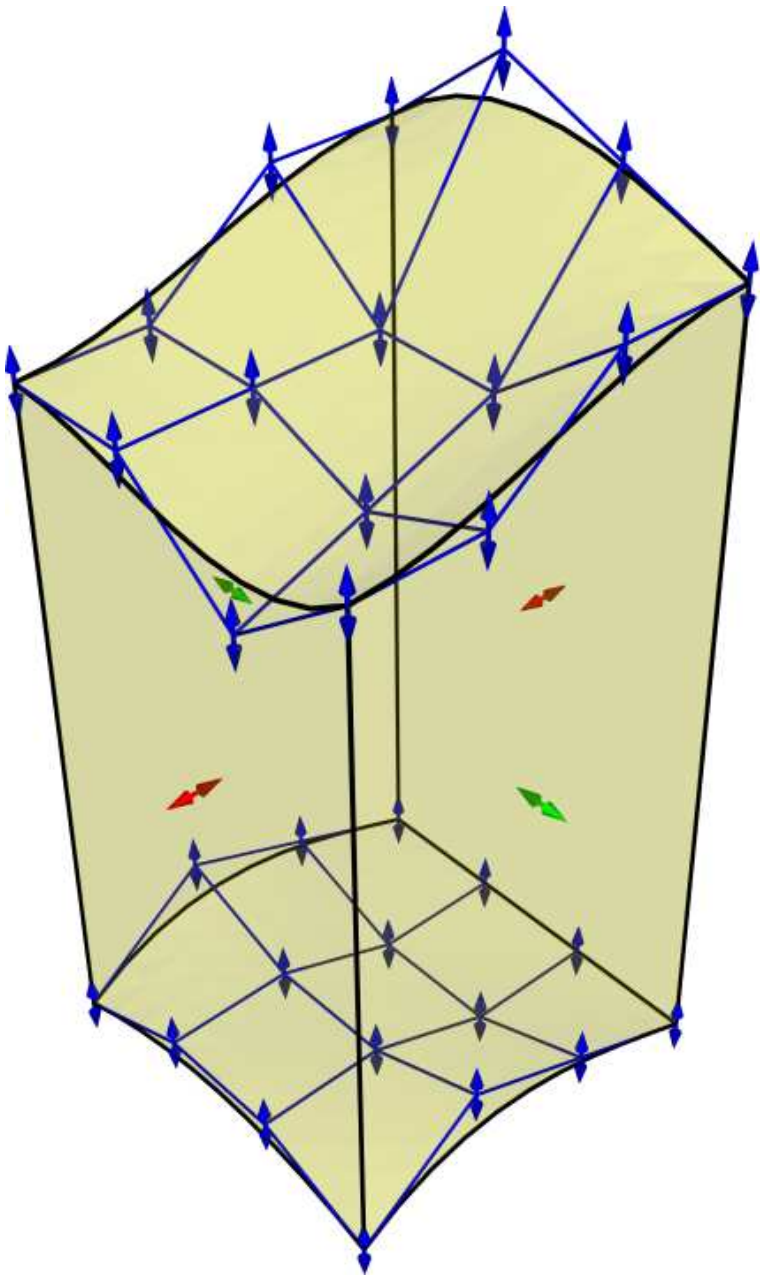


Figura 11.4: Representação mais econômica dos blocos da figura 11.1. Os pontos vermelhos, verdes e azuis representam os coeficientes de Bézier das componentes X,Y e Z respectivamente. Observe que apenas uma coordenada de cada ponto é livre.

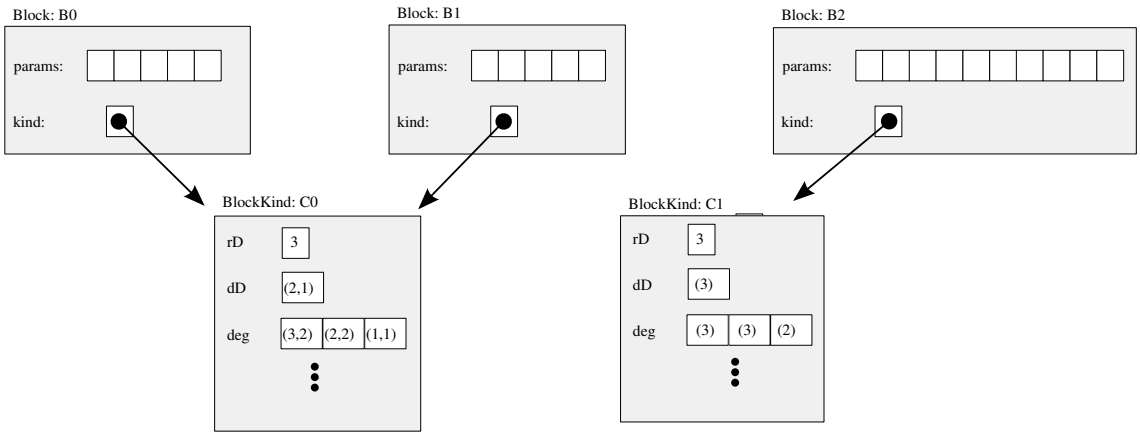


Figura 11.5: Implementação de um bloco de Bézier.

11.3 Compartilhamento de parâmetros

As restrições de continuidade e suavidade implícitas no modelo da figura 11.1 implicam na identidade de diversos de seus coeficientes de Bézier. Por exemplo, todos os blocos de uma fileira devem ter os mesmos coeficientes x_{min} e x_{max} ou y_{min} e y_{max}, dependendo da orientação da fileira. Além disso, os coeficientes de Bézier da face superior da camada de baixo dever ser iguais aos da face inferior da camada de cima. Adicionalmente, se desejarmos que as superfícies de cada camada sejam contínuas, os coeficientes de Bézier que determinam uma aresta superior ou inferior de um bloco devem coincidir com os coeficientes correspondentes nos blocos vizinhos.

Estas situações são muito comuns, e por essa razão a biblioteca BezEl foi projetada para facilitar o compartilhamento de parâmetros. Especificamente, o vetor `B.params` de cada bloco `B` contém apenas *referências* (índices) para seus parâmetros, e não os valores dos mesmos. Os valores dos parâmetros de todos os blocos de um mesmo modelo `G` são armazenados em um único vetor `G.vars` associado ao modelo. Desta forma, diversos blocos do mesmo modelo podem compartilhar o mesmo parâmetro, e a modificação do valor de um parâmetro é automaticamente refletida em todos os blocos que o compartilham. A figura 11.6 ilustra esta estratégia no caso da figura 11.5.

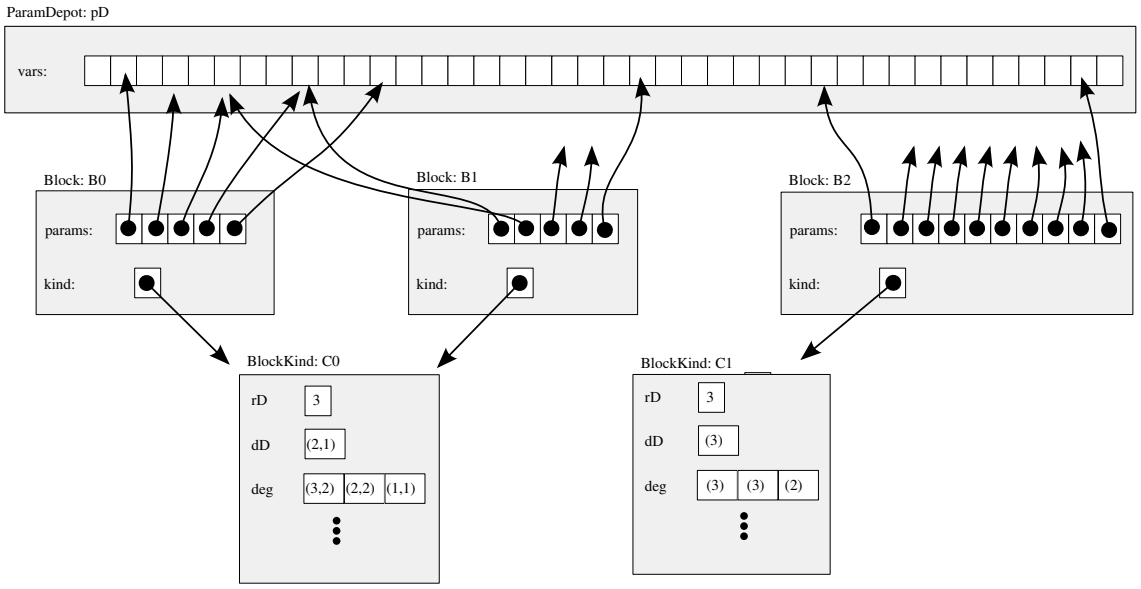


Figura 11.6: Implementação do compartilhamento de parâmetros entre blocos de Bézier

Como exemplo de compartilhamento de parâmetros, vamos descrever aqui o procedimento típico para efetuar a colagem de dois blocos `A` e `B` da malha da figura 11.1 (tipo `brickKind`). Esta operação é uma *colagem conforme*, onde uma faceta inteira de um bloco é identificada com uma faceta de outro bloco. Veja a figura 11.7. O objetivo da

colagem conforme é garantir que não haja frestas nem sobreposições entre os blocos. Esta operação deve ser repetida várias vezes para construir a malha da figura 11.1

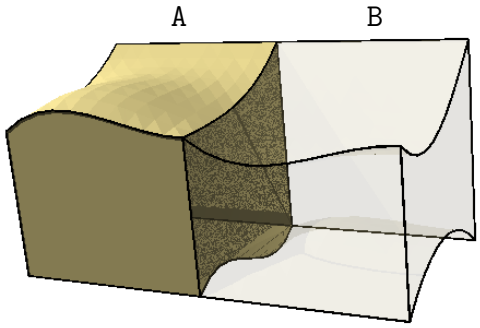


Figura 11.7: Colagem conforme de dois blocos do tipo `brickKind`.

Inicialmente, identificamos os parâmetros de `A` que serão compartilhados pelo bloco `B`. Estes são os parâmetros da faceta $U_1 = 1$ de `A`. Veja a figura 11.8. Como pode ser visto na figura 11.4, estes são os coeficientes de Bézier b^X_Λ, b^Y_Λ e b^Z_Λ das componentes X,Y e Z de `A` cujo hiper-índice Λ é tal que $\Lambda_{1,1} = 0$. Em seguida, para cada parâmetro de `A` a ser compartilhado, determinamos o hiper-índice do coeficiente correspondente no bloco `B`. Este será um dos coeficientes b^X_Ω, b^Y_Ω,b^Y_Ω com $\Omega_{1,0} = 0$. Finalmente, criamos o bloco `B` compartilhando os parâmetros de `A`.

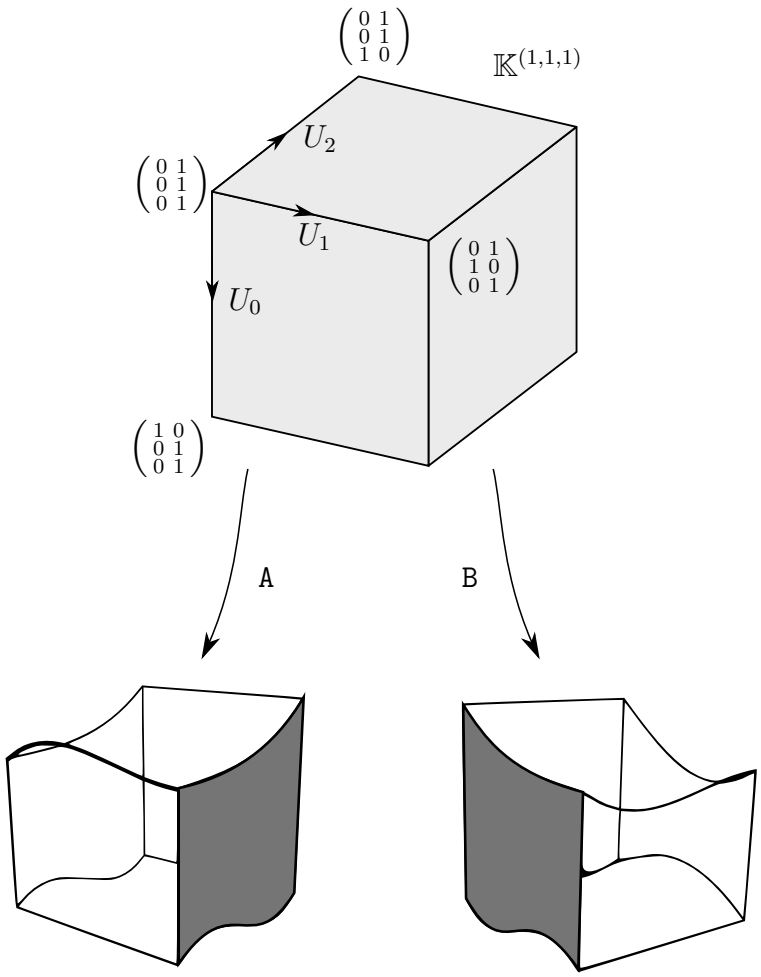


Figura 11.8: Diagrama da colagem conforme de dois blocos que compõem a malha da figura 11.1

11.4 Parâmetros internos e externos

Os 36 coeficientes de Bézier da figura 11.4 constituem uma *representação individual mínima* para o bloco da malha da figura 11.1 no sentido que para descrever a forma de um desses blocos isoladamente são necessários de fato 36 parâmetros. Entretanto, em outras situações, pode ocorrer que os coeficientes de Bézier de um único bloco isolado estão sujeitos a restrições. Neste caso, o bloco pode ser completamente descrito por um número menor de parâmetros.

Por exemplo, considere os blocos utilizados para construir malhas pseudo-cilíndricas como a da figura 11.9, onde cada anel é formado por N blocos simploidais. Cada um destes blocos é limitado por 4 faces planas e duas faces polinomiais cúbicas que aproximam a superfície de um cilindro. Veja figura 11.10. Observe que as componentes X e Y são de multi-grau $(3, 1, 0)$, ou seja, são polinômios de grau 3 em u, 1 em v e independentes de

w. Cada função tem portanto 8 coeficientes de Bézier, b^X_Λ e b^Y_Ω onde $\Lambda, \Omega \in \mathbb{H}^{(3,1,0)}_{(1,1,1)}$. Estes 16 coeficientes podem ser vistos como 8 pontos de controle projetados no plano XY. A componente Z, por outro lado, é de multi-grau $(0, 0, 1)$, ou seja, é independente de u e de v mas é uma função linear de w, e é especificada por 2 coeficientes de Bézier, b^Z_Ψ onde $\Psi \in \mathbb{H}^{(0,0,1)}_{(1,1,1)} = \{((0, 0), (0, 0)(0, 1)), ((0, 0), (0, 0), (1, 0))\}$; estes coeficientes especificam as alturas z_0 e z_1 das facetas inferior e superior do bloco.

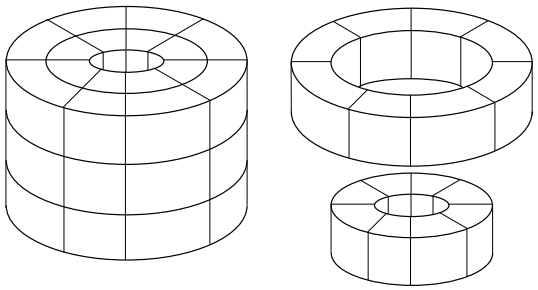


Figura 11.9: Uma malha pseudo-cilíndrica composta por 3 pares de anéis concêntricos, cada um formado por 4 blocos.

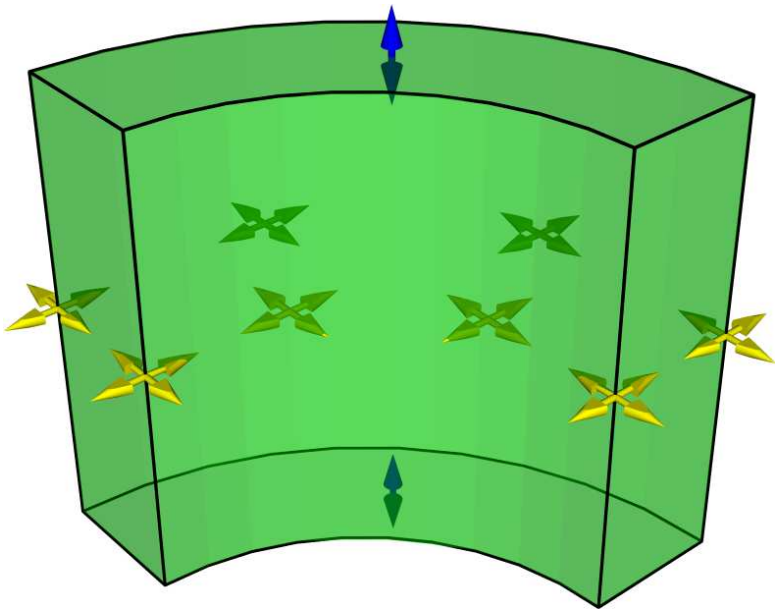


Figura 11.10: Um bloco simploidal para a malha da figura 11.9. Os pontos amarelos são pontos de controle de Bézier das coordenadas X e Y enquanto os pontos azuis são os pontos de controle de Bézier da coordenada Z.

Portanto, este bloco de Bézier generalizado tem $8 + 8 + 2 = 18$ coeficientes de Bézier. Entretanto, se considerarmos o ângulo $\theta = 2\pi/N$ fixo, todos os 18 coeficientes de Bézier

podem ser escritos como funções lineares de 8 parâmetros: as alturas z_0 e z_1, as coordenadas (x_c, y_c) do eixo do cilindro, e as coordenadas (x_0, y_0) e (x_1, y_1) de dois cantos do bloco, projetados no plano $Z = 0$. Veja a figura 11.11.

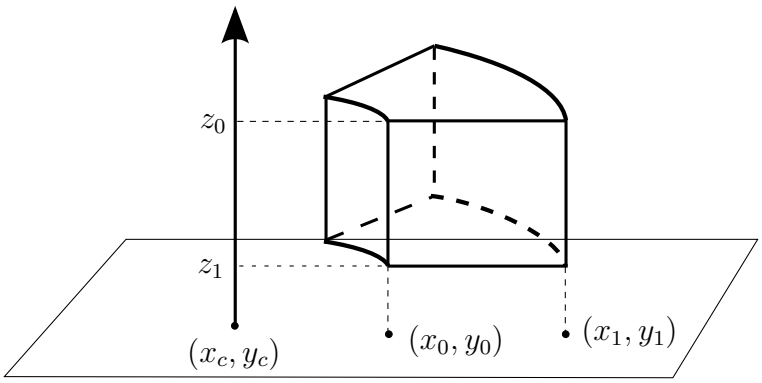


Figura 11.11: Ilustração dos parâmetros que derivam os coeficientes de Bézier de blocos que compõem a malha da figura 11.9.

A relação entre estes 8 parâmetros e os coeficientes de Bézier b^X_Λ, b^Y_Ω e b^Z_Ψ depende do ângulo θ e do método de aproximação escolhido. Em 2006, Riskus publicou um conjunto

de fórmulas para este fim [23]:

$$\begin{pmatrix} b^X_{((0,3),(0,1),(0,0))} \\ b^X_{((1,2),(0,1),(0,0))} \\ b^X_{((2,1),(0,1),(0,0))} \\ b^X_{((3,0),(0,1),(0,0))} \\ b^X_{((0,3),(1,0),(0,0))} \\ b^X_{((1,2),(1,0),(0,0))} \\ b^X_{((2,1),(1,0),(0,0))} \\ b^X_{((3,0),(1,0),(0,0))} \\ b^Y_{((0,3),(0,1),(0,0))} \\ b^Y_{((1,2),(0,1),(0,0))} \\ b^Y_{((2,1),(0,1),(0,0))} \\ b^Y_{((3,0),(0,1),(0,0))} \\ b^Y_{((0,3),(1,0),(0,0))} \\ b^Y_{((1,2),(1,0),(0,0))} \\ b^Y_{((2,1),(1,0),(0,0))} \\ b^Y_{((3,0),(1,0),(0,0))} \\ b^Z_{((0,0),(0,0),(0,1))} \\ b^Z_{((0,0),(0,0),(1,0))} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & \kappa & 0 & 0 & 0 \\ 1 & \kappa & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & \kappa & 0 & 0 \\ 1 & 0 & \kappa & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -\kappa & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & \kappa & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\kappa & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & \kappa & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_c \\ x_0 \\ x_1 \\ y_c \\ y_0 \\ y_1 \\ z_0 \\ z_1 \end{pmatrix}$$

onde o valor de κ depende do ângulo θ. A tabela 11.1 mostra os valores de κ para alguns ângulos.

A biblioteca BezEl também permite este tipo de otimização. Em geral, os parâmetros `B.params` de um bloco `B` não são os coeficientes de Bézier de suas componentes, mas sim uma coleção arbitrária de *parâmetros externos*. Os coeficientes de Bézier (*parâmetros internos do bloco*) são funções afins deles.

A relação entre os parâmetros internos e externos é mais um atributo do tipo do bloco,

θ	κ
30°	0.175534
45°	0.265216
60°	0.357259
90°	0.552284

Tabela 11.1: Tabela com valores de κ para diferentes θ.

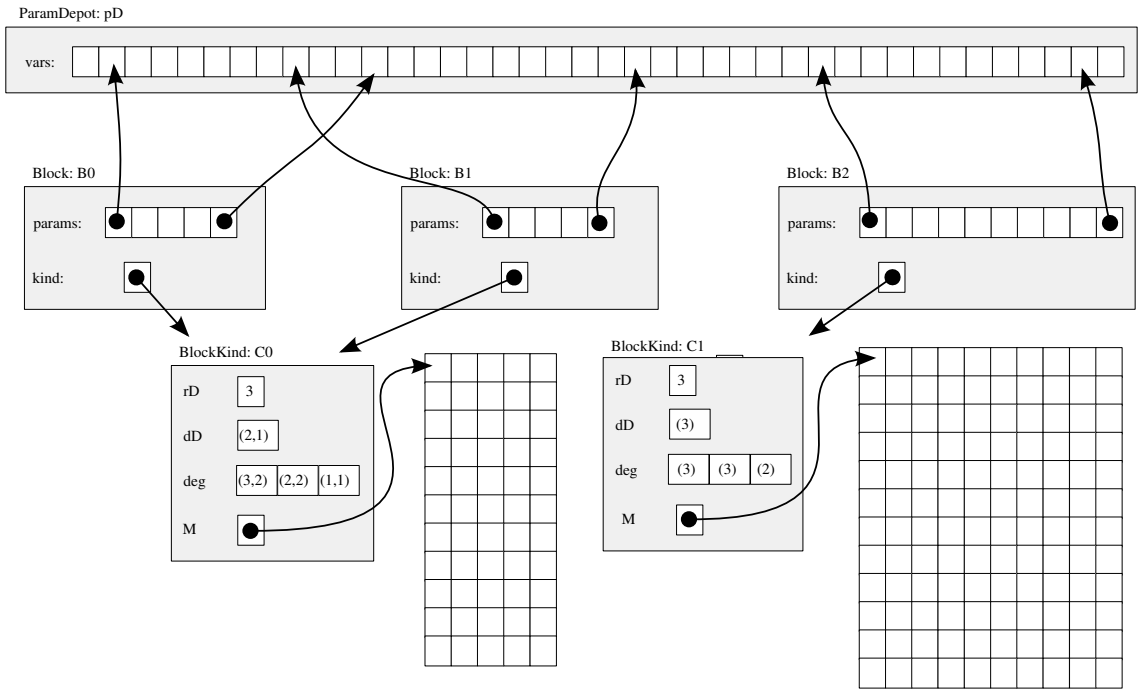


Figura 11.12: Esquema da relação o repositório de parâmetros, um bloco e seu tipo com matrizes de conversão entre parâmetros externos e internos.

representado por uma matriz `M` armazenada no objeto `BlockKind` correspondente. Veja figura 11.12

11.5 Extração de componentes

A representação descrita na seção 11.4 economiza espaço mas não é eficiente quando desejamos utilizar os coeficientes de Bézier de um mesmo bloco `B` diversas vezes, por exemplo na avaliação de `B`.F em muitos pontos do domínio. Para estes casos, utilizamos isoladamente as diversas componentes do bloco, obtidas através do método `B.explicit()`. Este método, que não precisa de argumentos, retorna um vetor de `B.kind.rD` polinômios simploidais, cada um representando uma componente `B`.F_i.

Na biblioteca BezEl, polinômios simploidais são representados por objetos da da classe `BezierSimploidPoly`. Cada instância `P` desta classe contém apenas três atributos: a multi-dimensão `P.domainType`, o multi-grau `P.multiDegree` e um vetor `P.coef` com seus coeficientes de Bézier. O método `P.evaluate` avalia o polinômio em um ponto `U` do domínio, fornecido como argumento.

Ao contrário da representação de blocos, o vetor `P.coef` armazena os valores numéricos dos coeficientes de Bézier. Estes valores são calculados pelo método `B.explicit` a partir dos valores correntes de parâmetros externos e da matriz `B.kind.M`.

11.6 Extração de facetas

Uma operação importante na biblioteca BezEl é a extração de facetas de blocos. Esta operação resulta em um bloco que compartilha parâmetros (externos e internos) com o bloco original. Esta é uma etapa importante em colagens arbitrárias (não conforme) blocos, como veremos no capítulo 12

Considere por exemplo um bloco de Bézier generalizado `B` cujo domínio é o prisma canônico, $\mathbb{K}^{(2,1)}$ de multi-grau $(2,3)$ para X,Y e Z, como o da figura 11.13a. A figura 11.13b mostra a faceta superior deste bloco correspondente à faceta $\mathbb{K}^{(2,1)}|_{(1,0)}$ do domínio. A extração desta faceta do bloco `B` produz um bloco de Bézier de domínio $\mathbb{K}^{(2)}$ e grau (2) para X,Y e Z.

Vamos agora descrever a extração desta faceta utilizando a biblioteca BezEl. Veja a figura 11.14. Esta operação consiste de duas etapas: criação um novo tipo de bloco para a faceta e a criação do bloco que é a faceta propriamente dita. Suponha que o bloco original é descrito pelo objeto `B` e seu tipo é o objeto `prismKind`. Para extrair a faceta da figura 11.13b procedemos da seguinte forma:

1. Criamos um tipo de bloco apropriado, `triangKind`, invocando o método `prismKind->get_FacetKind` cujos argumentos são os índices da faceta desejada, no caso, `(1,0)`.

2. Criamos o bloco `B_top` da faceta, invocando o método `B->get_Facet`. Os argumentos deste método são o tipo do bloco `triangKind` e os índices `(1,0)` da faceta desejada. Veja figura 11.14

A figura 11.13 mostra uma das faceta laterais, correspondente à faceta $\mathbb{K}^{(2,1)}|_{(0,1)}$ do domínio. A extração desta faceta produz um bloco de Bézier de domínio $\mathbb{K}^{(1,1)}$ e grau $(2,3)$ para X,Y e Z. De forma análoga, a extração desta faceta é realizada invocando o método `prismKind->get_FacetKind` com argumentos `(0,1)` para obter o tipo de bloco adequado, digamos `retangKind`; e em seguida, invocando o método `B->get_Facet` com argumentos `retangKind` e `(0,1)` para obter a faceta `B_side`.

Note que os parâmetros externos (e internos) dos novos blocos `B_top` e `B_side` são subconjuntos dos parâmetros externos (e internos) do bloco original `B`. Além disso, os tipos de bloco `triangKind` e `retangKind` podem ser compartilhados por quaisquer outras facetas semelhantes, ou quaisquer outros retalhos de mesmas multi-dimensões e multi-graus.

A figura 11.15 mostra outro exemplo, a extração da faceta lateral interna do bloco da figura 11.10. O procedimento é similar. Seja `B` o bloco original, de tipo `cylSliceKind`. O método `B->get_FacetKind`, invocado com argumentos `(2,0)` fornece um novo tipo de bloco, `cylPlateKind`, adequado para a faceta desejada. Este tipo de bloco é um retalho bidimensional de domínio $\mathbb{K}^{(1,1)}$, com X e Y de multi-grau $(3,0)$ e Z de multi-grau $(0,1)$. O objeto `B_side`, que descreve a faceta, é obtido invocando `B->get_Facet`, com argumentos `cylPlateKind` e `(2,0)`. Veja figura 11.16.

Assim como `cylSliceKind` tem uma relação não trivial entre parâmetros externos e internos, o mesmo ocorre com `cylPlateKind`. Temos 4 coeficientes de Bézier para X, 4 para Y e 2 para Z, num total de 10 parâmetros. Porém, todos estes podem ser descritos como funções lineares de 4 parâmetros externos: o intervalo de Z, z_{min} e z_{max}, e as coordenadas x, y de uma das arestas verticais.

Uma vez que `sliceKind` armazena em `sliceKind->M` as relações entre os parâmetros internos e externos de blocos deste tipo, a criação do tipo `plateKind` requer a especificação destas relações para blocos do tipo da faceta. A matriz `plateKind->M`, que descreve as relações entre os parâmetros externos e internos da faceta de índice $(2,0)$, é uma sub-matriz de `sliceKind->M` obtida selecionando as linhas relacionadas aos hiper-índices

$$\{\Lambda \in \mathbb{H}^{(3,0,1)}_{(1,1,1)} \mid \Lambda_{2,0} \neq 0\}$$

e eliminando as colunas compostas unicamente por zeros.

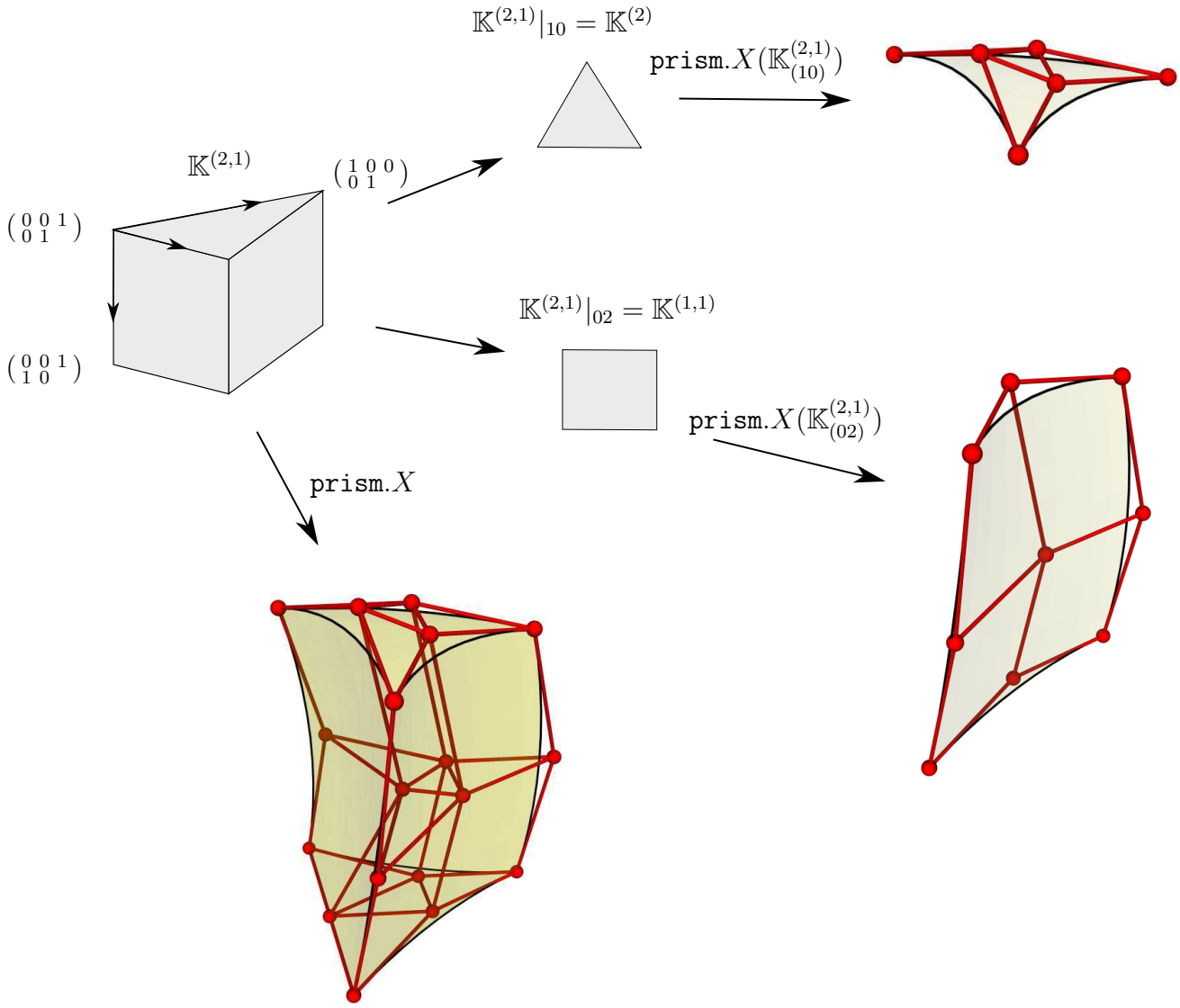


Figura 11.13: Um bloco prismático de multi-grau $(2, 2)$ e suas facetas $(1, 0)$ e $(0, 2)$.

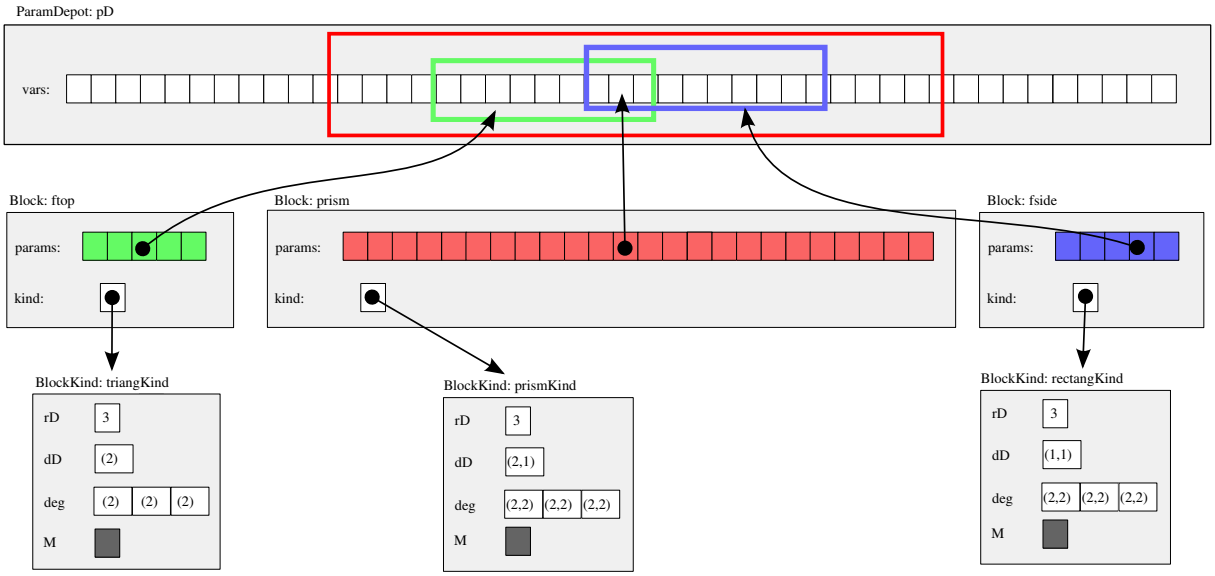


Figura 11.14: Extração de facetas de um prisma da figura 11.13

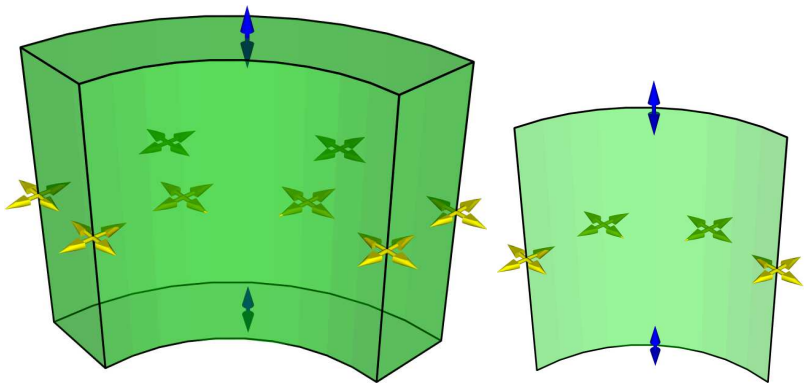


Figura 11.15: Facetas do bloco da malha cilíndrica: (a) Faceta $(0, 0)$, (b)

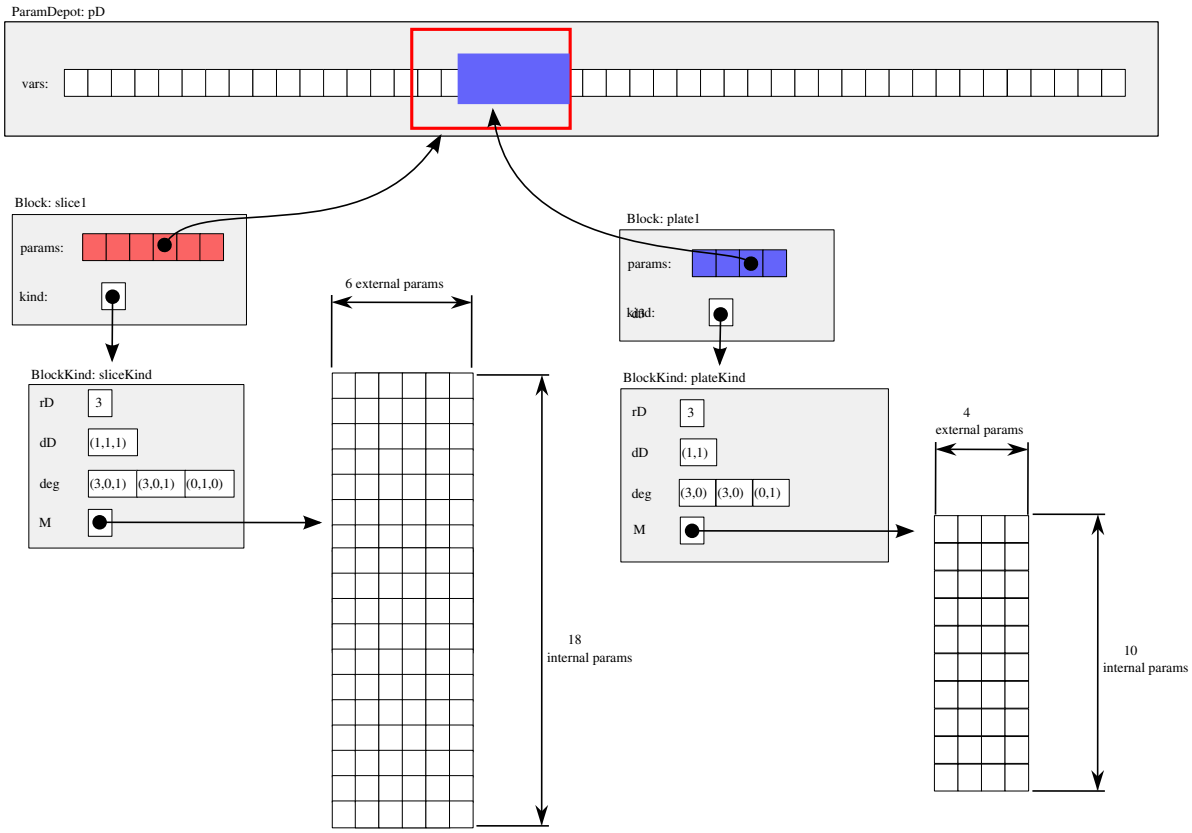
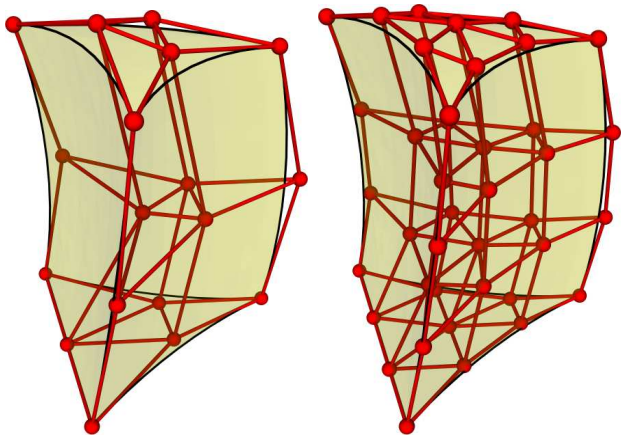


Figura 11.16: Extração de facetas do prisma da figura 11.13

11.7 Elevação de grau

Outra operação muito freqüente em modelagem geométrica é a elevação de grau. No exemplo da figura 11.17, a partir de um elemento prismático de Bézier `B`, de multi-grau $(2,2)$ nas componentes X,Y e Z (figura 11.17a), obtemos um bloco `RB` (figura 11.17b) de mesmo domínio mas de multi-grau $(3,3)$ em todas as componentes que coincide com `B` em todos os pontos do domínio.



(a) (b)

Figura 11.17: Elevação de grau de um bloco prismático de multi-grau $(2,2)$ para $(3,3)$.

A figura 11.18 ilustra a elevação de grau utilizando a biblioteca BezEl. Assim como a extração de facetas, esta operação consiste de duas etapas: criação um tipo de bloco adequado e a criação do bloco de grau maior propriamente dito. Suponha que o bloco da figura 11.17a é descrito pelo objeto `prism22` e seu tipo é o objeto `prism22Kind`. Suponha que não há qualquer restrição sobre a forma do bloco, de modo que seus 54 parâmetros externos são os próprios coeficientes de Bézier das funções X,Y e Z (18 pontos de controle). Para elevar o grau de todas as componentes para $(3,3)$ procedemos da seguinte forma:

1. Criamos um tipo de bloco apropriado, digamos `prism22_33_Kind`, invocando o método `prism22Kind->get_raisedDegreeKind` cujo argumento é um vetor contendo os novos multi-graus, neste caso, `[(3,3),(3,3),(3,3)]`.

2. Criamos o bloco da figura 11.17b `prism22_33`, invocando o método `get_SubBlock` de `prism22` utilizando como argumento o tipo do bloco `prism22_33_Kind`.

O novo bloco tem 120 parâmetros internos (40 pontos de controle). Entretanto, como descrito na seção 8.2.3, os coeficientes de Bézier do novo elemento são funções lineares dos coeficientes do elemento inicial. Portanto, os parâmetros externos do novo bloco são os mesmos do bloco original, ou seja os 54 coeficientes de Bézier. A relação entre os parâmetros

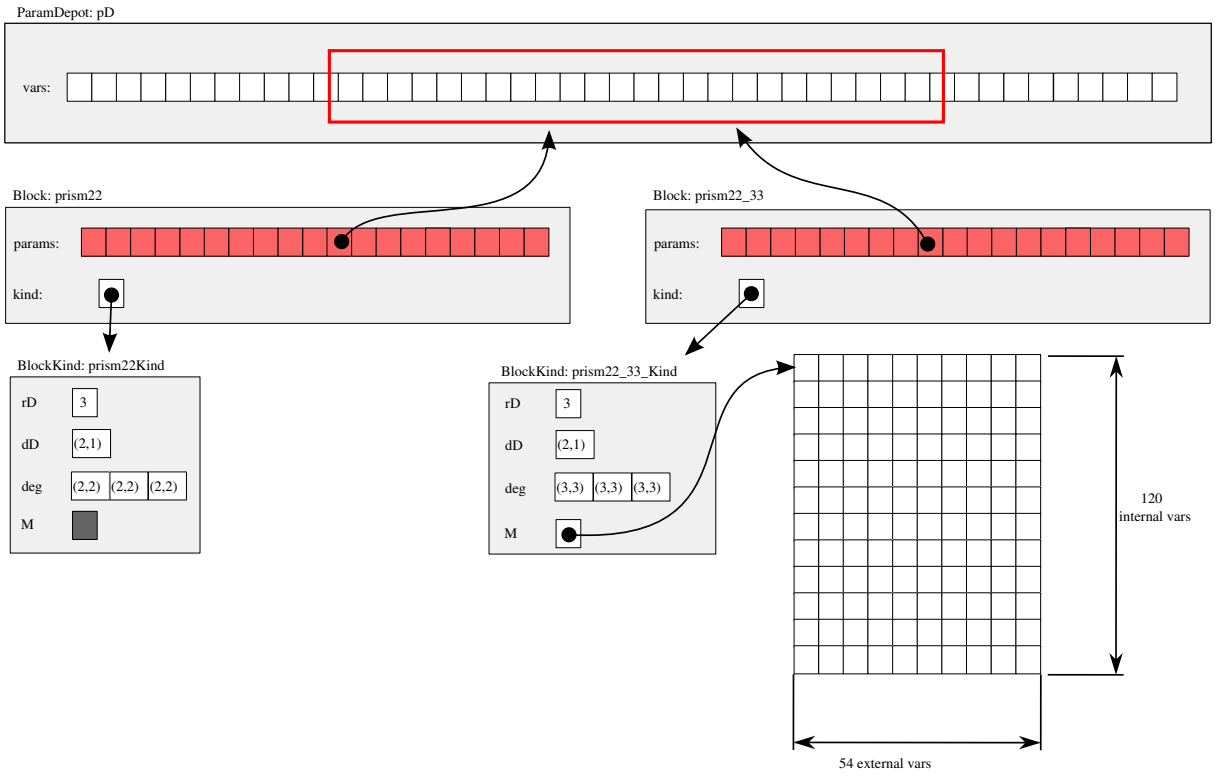
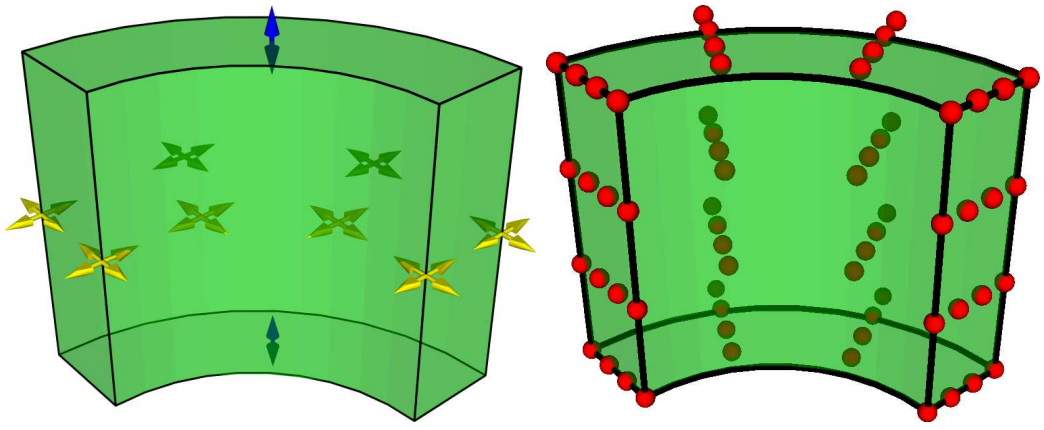


Figura 11.18: Diagrama ilustrativo da elevação de grau de um bloco prismático.

internos e externos do novo elemento são armazenadas na matriz `prism22_33_Kind.M` que tem 120 linhas e 54 colunas e cujos elementos são os coeficientes $K_{\Omega\Lambda}$ definidos na seção 8.2.3.

A figura 11.19 mostra outro exemplo, a elevação de grau do elemento para compor malhas pseudo-cilíndricas descrito na seção 11.4. O procedimento é similar. Seja `cylSlice1` o bloco original, de tipo `cylSliceKind`. O método `cylSliceKind->get_raisedDegreeKind`, invocado utilizando o vetor de multi-graus `[(3,3,3),(3,3,3),(3,3,3)]` como argumento, fornece um novo tipo de bloco, `cylSlice33Kind`, adequado para o bloco desejado. Este tipo de bloco tem como domínio o simplóide canônico $\mathbb{K}^{(1,1,1)}$ e as componentes X,Y e Z são de grau $(3,3,3)$. Além disso, este tipo de bloco é responsável pela conversão dos 6 parâmetros externos do bloco original `cylSlice1` nos seus 18 parâmetros internos (a matriz `cylSliceKind.M`) e a conversão destes nos 192 parâmetros internos do bloco coincidente de grau maior (fórmula (8.7)). Para isto, estas duas operações lineares são combinadas em uma única matriz de 192 linhas por 6 colunas, o atributo `cylSlice33Kind->M`. O objeto `cylSlice33`, que descreve o elemento coincidente de grau maior, é obtido invocando `cylSlice1->get_SubBlock`, utilizando como argumento o novo tipo de bloco `cylSlice33Kind`. Veja figura 11.20.



(a) (b)

Figura 11.19: Elevação de grau de um bloco da malha pseudo-cilíndrica de grau $(0, 3, 3)$ em X,Y e $(1, 0, 0)$ em Z, para $(3, 3, 3)$ em X, Y e Z.

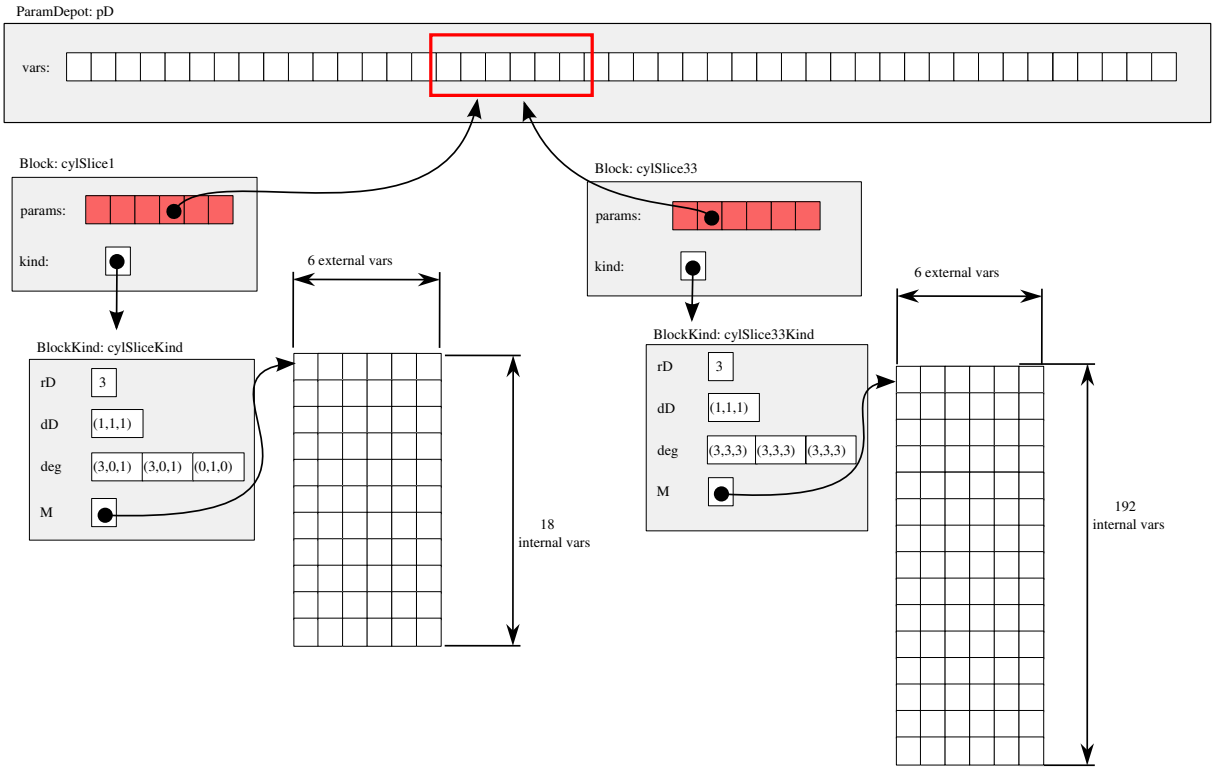


Figura 11.20: Diagrama ilustrativo da elevação de grau de um bloco da malha pseudo-cilíndrica.

11.8 Reparametrização

A função `A`.F de um bloco `A` pode ser reparametrizada através do método `A.subBlock`. Esta operação resulta em um novo bloco cujos parâmetros externos são um subconjunto dos parâmetros externos de `A`, mas, em geral, definido em um domínio diferente (que pode extrapolar do domínio de `A`) e com um conjunto diferente de parâmetros internos.

Como nas operações anteriores, é necessário criar um novo tipo de bloco para cada mapa Γ definido, e depois criar o bloco desejado que terá este tipo. Este novo tipo `reparKind` é criado pelo método `A.kind.get_BlockKind_of_Composition_w_DM` utilizando o mapa Γ como argumento. A matriz `reparKind.M` é a composição da matriz dos coeficientes de Γ (descrita no capítulo 10) com a matriz `A.kind.M`.

Um mapa afim Γ é representado na biblioteca BezEl por uma instância da classe `DomainMapping`, que contém os coeficientes M_{ij}^{rs}(capítulo 9) armazenados na forma de uma matriz irregular (indexada por r, s) de matrizes irregulares (indexadas por i, j).

Note que o mapa Γ não precisa ser sobrejetor. Por exemplo, se `A` é um retalho triangular (domínio $\mathbb{K}^2$) e Γ leva de $\mathbb{K}^1$ para $\mathbb{K}^2$, a reparametrização de `A` por Γ retorna uma curva de Bézier, que corresponde ao segmento de reta $\Gamma(\mathbb{K}^1)$ no espaço $\mathbb{A}^2$.

11.9 Derivada direcional

Outra operação importante implementada pela BezEl é o cálculo da derivada direcional da função `A`.F de um bloco `A`, em uma direção dada Ξ do domínio do bloco. Como descrito no capítulo 8, a derivada direcional de cada componente `A`.F_i também é um polinômio simploidal e, portanto, pode ser vista como a componente `dA`.F_i de um outro bloco `dA` de mesma multi-dimensão.

Para isso, um novo tipo `derivKind` deve ser criado para cada tipo de bloco e cada direção Ξ através do método `A.kind.get_BlockKind_of_DirectionalDerivative`. Os argumentos deste método são a ordem r da derivada e o vetor direção Ξ. Em seguida, o bloco `dA` pode ser criado através do método `A.subBlock`, fornecendo o tipo `derivKind` como argumento.

Capítulo 12

Restrições entre blocos

Outra ferramenta importante para modelagem geométrica, fornecida pela biblioteca BezEl, é a especificação de restrições afins (de 1º grau) entre os parâmetros dos blocos de um mesmo modelo.

Para representar tais restrições, cada modelo `G` da biblioteca BezElpossui, além do vetor `G.vars` de classe `ParamDepot`, uma instância `G.eqs` da classe `EquationDepot`, que é uma lista de equações afins sobre os parâmetros externos armazenados em `G.vars`. Cada elemento desta lista é um objeto da classe `Equation` que representa uma única restrição.

Observe que restrições que envolvem os coeficientes de Bézier de um único bloco isolado podem ser implementadas alternativamente pela matriz do tipo do bloco, como descrito na seção 11.4. Mais precisamente, se o bloco do tipo `B` tem n coeficientes de Bézier e as restrições definem um certo subespaço afim do $\mathbb{R}^n$ de dimensão p, é possível implementar estas restrições escolhendo p parâmetros externos e preenchendo adequadamente a matriz `B.M` (de tamanho $n \times p$). Porém, esta solução exige que um novo tipo de bloco seja criado para cada conjunto diferente de restrições.

12.1 Colagem não-conforme

Um exemplo de uso de restrições é a colagem de dois blocos pseudo-cilíndricos de θ radianos, `cyl_A` e `cyl_B` (veja seção 11.4), de modo que a faceta externa de `cyl_B` coincide com parte da faceta interna de `cyl_A`, como na figura 12.1.

Para que não existam nem frestas nem sobreposição entre os blocos, as restrições que devem ser satisfeitas são facilmente determinadas pela própria definição dos parâmetros externos deste tipo de bloco (ver figura 11.11):

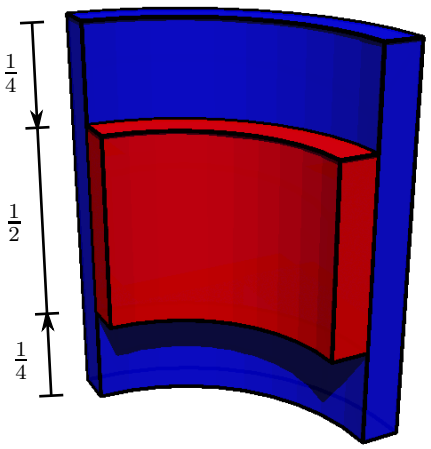


Figura 12.1: Uso de restrições entre parâmetros externos de dois blocos pseudo-cilíndricos: o `cyl_A` (em verde) e o `cyl_B` (em amarelo).

$$\texttt{cyl\_A}.x_1 - \texttt{cylB}.x_0 = 0 \quad (12.1)$$

$$\texttt{cyl\_A}.y_1 - \texttt{cylB}.y_0 = 0 \quad (12.2)$$

$$0.75\texttt{cyl\_A}.z_0 + 0.25\texttt{cyl\_A}.z_1 - \texttt{cylB}.z_0 = 0 \quad (12.3)$$

$$0.25\texttt{cyl\_A}.z_0 + 0.75\texttt{cyl\_A}.z_1 - \texttt{cylB}.z_1 = 0 \quad (12.4)$$

Neste caso, a lista `G.eqs` do modelo teria 4 elementos do tipo `Equation` (veja figura 12.2). Note que os parâmetros externos do bloco, na verdade, residem no vetor `G.vars`.

12.2 Restrições entre parâmetros internos

No exemplo da seção 12.1, as restrições foram formuladas diretamente sobre os parâmetros externos dos blocos. Em algumas situações, é mais natural expressar uma determinada restrição em termos dos parâmetros internos (coeficientes de Bézier). Nestes casos, as matrizes `M` dos tipos dos blocos envolvidos devem ser usadas para substituir cada parâmetro interno presente na equação pela combinação afim equivalente de parâmetros externos.

A inserção de um poço ao modelo da figura 11.1 ilustra uma destas situações. Veja a figura 12.3. Neste exemplo, o poço é composto por 4 blocos do tipo `cylSliceKind` (descrito na seção 11.4) de ângulo $\theta = \pi/2$. Como mostra a figura 12.4, esta operação

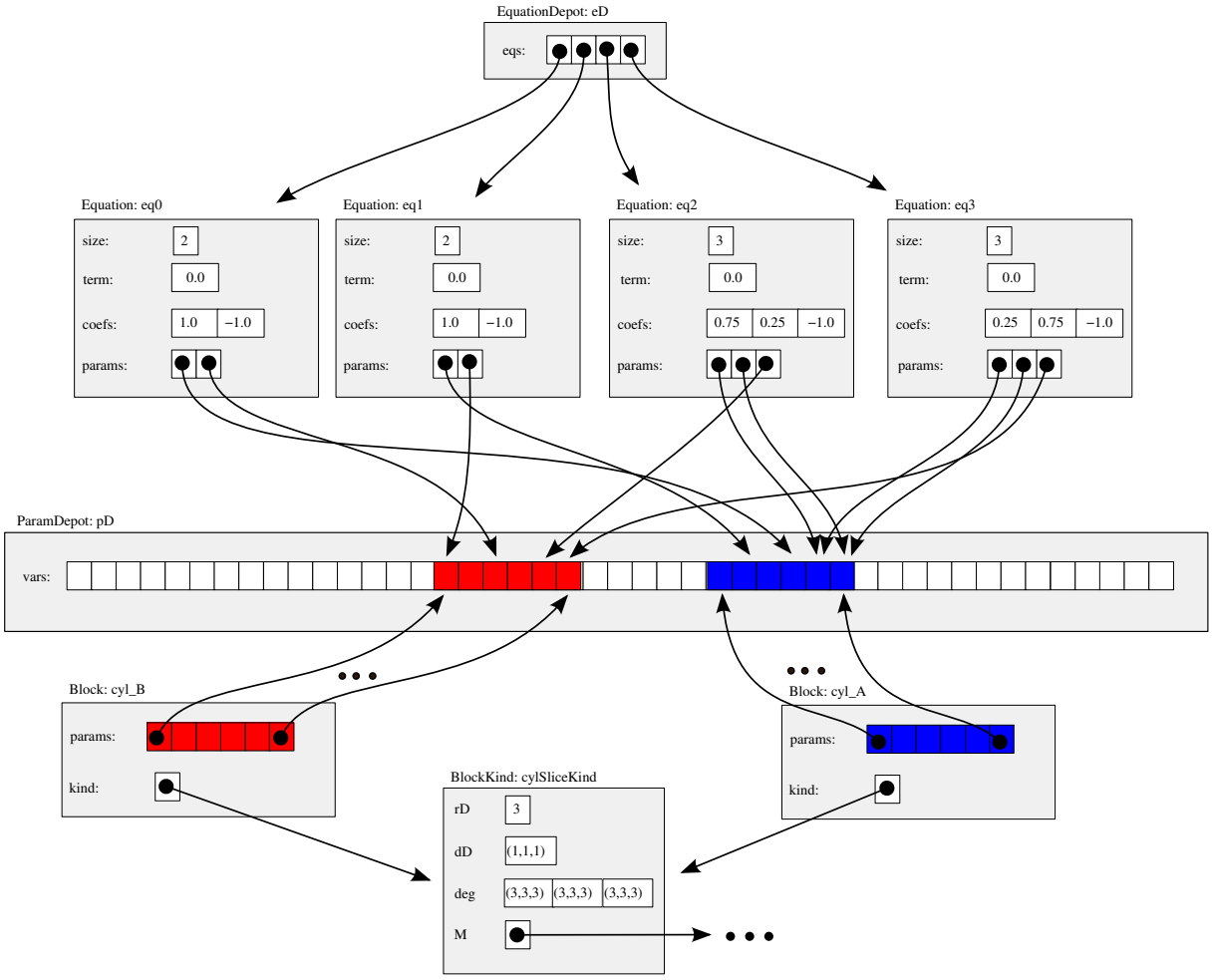


Figura 12.2: Representação das restrições 12.1 a 12.4 na BezEl.

implica em substituir um dos blocos da malha pelos 4 blocos do poço, e mais 4 blocos de transição que preenchem o espaço entre o poço e o resto da malha.

Os blocos de transição também podem ser blocos tensoriais, mas devem ser de tipo diferente do `brickKind`. Para que eles possam se acomodar adequadamente à parede pseudo-cilíndrica do poço, as componentes X e Y deste novo tipo devem ser funções cúbicas das coordenadas U_1 e U_2; mas podem ser independentes de U_0 pois tanto o poço quanto a malha tem paredes verticais. A componente Z, por outro lado, deve ser uma função linear de U_0 e cúbica em U_1 e U_2. Assim, tanto a função X quanto a Y possuem 16 coeficientes de Bézier, b^x_Λ e b^y_Ω onde $\Lambda, \Omega \in \mathbb{H}^{(3,3,0)}_{(1,1,1)}$, enquanto que a componente Z possui 32 coeficientes, b^z_Ψ onde $\Psi \in \mathbb{H}^{(3,3,1)}_{(1,1,1)}$. Vamos supor que este tipo de bloco é descrito pelo objeto `cylJoinKind`, cujos parâmetros externos são os próprios coeficientes de Bézier (isto é, `cylJoinKind.M` é trivial).

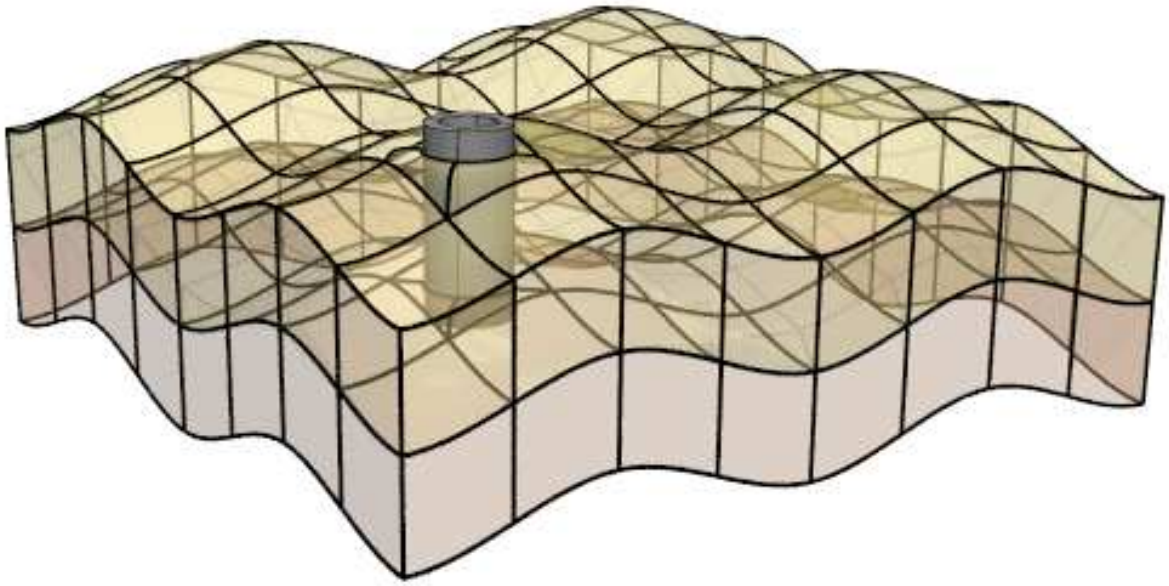


Figura 12.3: Adição de um poço ao modelo da figura 11.1.

Suponha que desejamos colar um bloco `join` deste tipo a um bloco `cyl` do tipo `cylSliceKind` segundo a figura 12.5. Para isso, procedemos da seguinte forma

1. Obtemos a faceta $(1, 1)$ de `join`, que chamaremos de `f_join`. Note que os parâmetros externos de `f_join` são um subconjunto dos parâmetros externos de `join`.

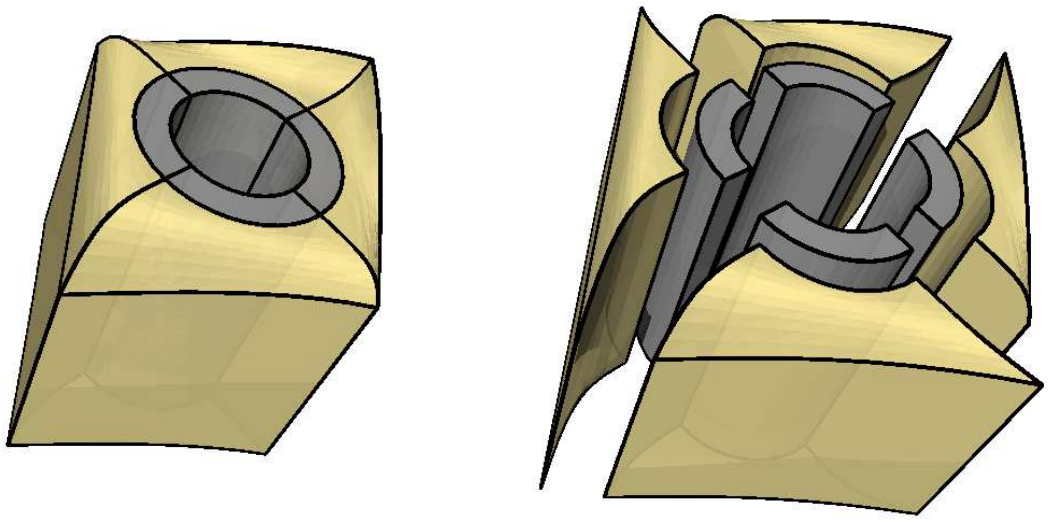
2. Obtemos a faceta $(1, 0)$ do bloco `cyl`, que chamaremos de `f_cyl`.

3. Elevamos o multi-grau da componente Z da faceta `f_cyl` de $(0, 1)$ para $(3, 1)$, obtendo o novo bloco `f_cyl_rD` de mesmo multi-grau da faceta `f_join`. Note que `f_cyl_rD` tem um novo tipo e que seus parâmetros externos são um subconjunto dos parâmetros externos de `cyl`.

4. Para cada componente (X,Y e Z) de `f_join`, e para cada coeficiente de Bézier β dessa componente (ou seja b^x_Λ para X, b^y_Ω para Y e b^z_Ψ para Z), criamos uma restrição no repositório `eqs` do modelo que iguala `f_join`.β ao coeficiente de Bézier `f_cyl_rD`.γ correspondente na outra faceta. Como o tipo de `f_cyl_rD` possui uma matriz `M` não trivial, usamos esta matriz para substituir este coeficiente de Bézier pela combinação apropriada dos parâmetros externos de `f_cyl_rD`.

5. Finalmente, descartamos os blocos auxiliares criados: `f_join`, `f_cyl` e `f_cyl_rD`.

O passo 1 implica em criar (ou se reutilizar) um tipo de bloco específico para a faceta `f_join`. Este tipo de bloco tem domínio $\mathbb{K}^{(1,1)}$ e multigraus $(3, 0)$ para as componentes X



(a) (b)

Figura 12.4: Região em torno do poço.

e Y e $(3,1)$ para a componente Z. Analogamente, o passo 2 implica em criar ou reutilizar um tipo de bloco para a faceta, `f_cylKind`, de domínio $\mathbb{K}^{(1,1)}$ e multigraus $(3,0)$ para as componentes X e Y e $(1,0)$ para a componente Z, cuja matriz `f_cylKind.M` é

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & \kappa & 0 & 0 \\ \kappa & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\kappa & 1 & 0 & 0 \\ -1 & \kappa & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

onde $\kappa = 0.552285$ (veja seção 11.4).

O passo 3 requer outro tipo de bloco, `f_cylKind_rD`, para a faceta `f_cyl_rD`, cuja matriz

é

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & \kappa & 0 & 0 \\ \kappa & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\kappa & 1 & 0 & 0 \\ -1 & \kappa & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

No passo 4, as restrições criadas sobre os parâmetros internos são

$$
\begin{aligned}
\texttt{f\_join}.b^{x}_{((3,0),(0,0)} &= \texttt{f\_cyl\_rD}.c^{x}_{((3,0),(0,0)} \\
\texttt{f\_join}.b^{x}_{((2,1),(0,0)} &= \texttt{f\_cyl\_rD}.c^{x}_{((2,1),(0,0)} \\
\texttt{f\_join}.b^{x}_{((1,2),(0,0)} &= \texttt{f\_cyl\_rD}.c^{x}_{((1,2),(0,0)} \\
\texttt{f\_join}.b^{x}_{((0,3),(0,0)} &= \texttt{f\_cyl\_rD}.c^{x}_{((0,3),(0,0)} \\
\texttt{f\_join}.b^{y}_{((3,0),(0,0)} &= \texttt{f\_cyl\_rD}.c^{y}_{((3,0),(0,0)} \\
\texttt{f\_join}.b^{y}_{((2,1),(0,0)} &= \texttt{f\_cyl\_rD}.c^{y}_{((2,1),(0,0)} \\
\texttt{f\_join}.b^{y}_{((1,2),(0,0)} &= \texttt{f\_cyl\_rD}.c^{y}_{((1,2),(0,0)} \\
\texttt{f\_join}.b^{y}_{((0,3),(0,0)} &= \texttt{f\_cyl\_rD}.c^{y}_{((0,3),(0,0)} \\
\texttt{f\_join}.b^{y}_{((3,0),(0,1)} &= \texttt{f\_cyl\_rD}.c^{y}_{((3,0),(0,1)} \\
\texttt{f\_join}.b^{y}_{((2,1),(0,1)} &= \texttt{f\_cyl\_rD}.c^{y}_{((2,1),(0,1)} \\
\texttt{f\_join}.b^{y}_{((1,2),(0,1)} &= \texttt{f\_cyl\_rD}.c^{y}_{((1,2),(0,1)} \\
\texttt{f\_join}.b^{y}_{((0,3),(0,1)} &= \texttt{f\_cyl\_rD}.c^{y}_{((0,3),(0,1)} \\
\texttt{f\_join}.b^{z}_{((3,0),(1,0)} &= \texttt{f\_cyl\_rD}.c^{z}_{((3,0),(1,0)} \\
\texttt{f\_join}.b^{z}_{((2,1),(1,0)} &= \texttt{f\_cyl\_rD}.c^{z}_{((2,1),(1,0)} \\
\texttt{f\_join}.b^{z}_{((1,2),(1,0)} &= \texttt{f\_cyl\_rD}.c^{z}_{((1,2),(1,0)} \\
\texttt{f\_join}.b^{z}_{((0,3),(1,0)} &= \texttt{f\_cyl\_rD}.c^{z}_{((0,3),(1,0)}
\end{aligned}
$$

que, expressas em termos dos parâmetros externos de `cyl` são

$$\begin{array}{lcll}
\texttt{f_join}.b^{x}_{((3,0),(0,0)} & - & \texttt{cyl}.y_1 & = 0 \\
\texttt{f_join}.b^{x}_{((2,1),(0,0)} & - & (\kappa\texttt{cyl}.x_1 + \texttt{cyl}.y_1) & = 0 \\
\texttt{f_join}.b^{x}_{((1,2),(0,0)} & - & (\texttt{cyl}.x_1 + \kappa\texttt{cyl}.y_1) & = 0 \\
\texttt{f_join}.b^{x}_{((0,3),(0,0)} & - & \texttt{cyl}.x_1 & = 0 \\
\texttt{f_join}.b^{y}_{((3,0),(0,0)} & - & (-\texttt{cyl}.x_1) & = 0 \\
\texttt{f_join}.b^{y}_{((2,1),(0,0)} & - & (-\texttt{cyl}.x_1 + \kappa\texttt{cyl}.y_1) & = 0 \\
\texttt{f_join}.b^{y}_{((1,2),(0,0)} & - & (-\kappa\texttt{cyl}.x_1 + \texttt{cyl}.y_1) & = 0 \\
\texttt{f_join}.b^{y}_{((0,3),(0,0)} & - & \texttt{cyl}.y_1 & = 0 \\
\texttt{f_join}.b^{y}_{((3,0),(0,1)} & - & \texttt{cyl}.z_0 & = 0 \\
\texttt{f_join}.b^{y}_{((2,1),(0,1)} & - & \texttt{cyl}.z_0 & = 0 \\
\texttt{f_join}.b^{y}_{((1,2),(0,1)} & - & \texttt{cyl}.z_0 & = 0 \\
\texttt{f_join}.b^{y}_{((0,3),(0,1)} & - & \texttt{cyl}.z_0 & = 0 \\
\texttt{f_join}.b^{z}_{((3,0),(1,0)} & - & \texttt{cyl}.z_1 & = 0 \\
\texttt{f_join}.b^{z}_{((2,1),(1,0)} & - & \texttt{cyl}.z_1 & = 0 \\
\texttt{f_join}.b^{z}_{((1,2),(1,0)} & - & \texttt{cyl}.z_1 & = 0 \\
\texttt{f_join}.b^{z}_{((0,3),(1,0)} & - & \texttt{cyl}.z_1 & = 0
\end{array}$$

Neste caso, como cada parâmetro de `f_join` aparece em exatamente uma equação com coeficiente 1, todas estas restrições são independentes e compatíveis.

Note que cada ponto do $\mathbb{R}^3$ na faceta compartilhada do modelo tem as mesmas coordenadas locais relativas aos blocos `f_join` e `f_cyl_rD`. Ou seja, a colagem garante que $\texttt{f\_cyl\_rD}.F(U) = \texttt{f\_join}.F(U)$ para todo $U \in \mathbb{A}^{(1,1)}$.

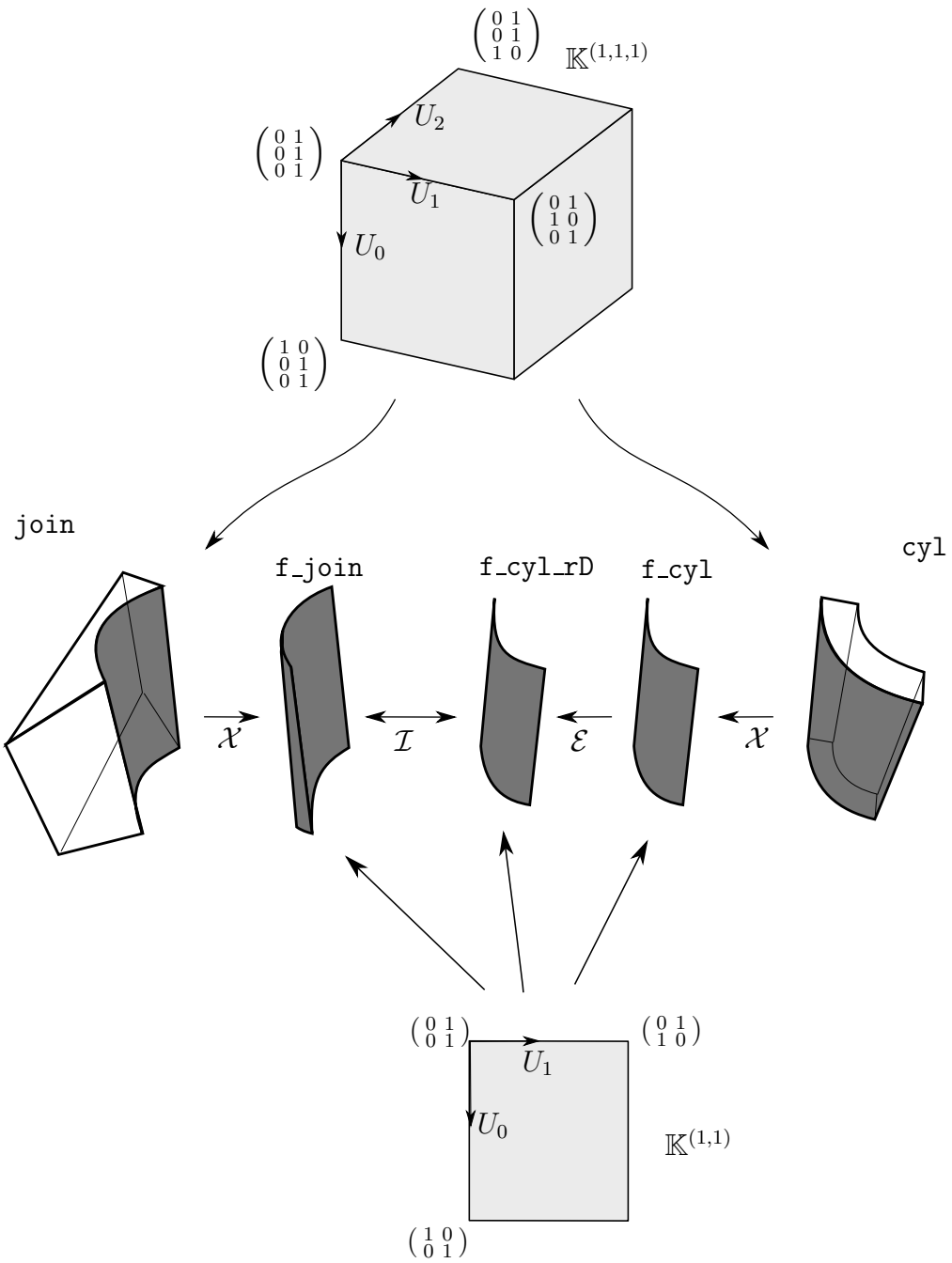


Figura 12.5: Diagrama da colagem conforme de um bloco que compõe o poço com um bloco de transição. Denotamos por $\mathcal{X}$, $\mathcal{E}$ e $\mathcal{I}$ as operações de extração de uma faceta, elevação de grau e identificação de coeficientes de Bézier, respectivamente.

12.3 Colagem não-conforme geral

Em muitas aplicações, a identificação de facetas completas não é suficiente. Freqüentemente, desejamos identificar uma parte de uma faceta de um bloco com parte de uma faceta de outro bloco. Nestes casos, como no exemplo da seção 12.1, a correspondência entre as coordenadas locais de cada ponto compartilhado relativas a cada faceta não é trivial. Na biblioteca BezEl, este tipo de colagem pode ser realizada se a correspondência entre as coordenadas locais for uma bijeção afim.

Mais especificamente, suponha que as facetas em questão são representadas por dois elementos simploidais de Bézier `A` e `B`, de domínios $\mathbb{K}^\delta$ e $\mathbb{K}^\varepsilon$, respectivamente, e queremos identificar $\texttt{A}.F(U)$ com $\texttt{B}.F(V)$, onde U pertence a um subconjunto $\mathbb{X}$ de $\mathbb{A}^\delta$ e V pertence a um subconjunto $\mathbb{Y}$ de $\mathbb{A}^\delta$. Para simplificar, vamos supor que $\mathbb{X}$,$\mathbb{Y}$,$\mathbb{A}^\delta$ e $\mathbb{A}^\varepsilon$ têm a mesma dimensão topológica, e que queremos identificar $\texttt{A}.F(U)$ com $\texttt{B}.F(\Gamma(U))$ onde Γ é um mapa afim de $\mathbb{A}^\delta$ para $\mathbb{A}^\varepsilon$ dado. Note que nem $\mathbb{X}$ nem $\mathbb{Y}$ tem que ser o simplóide canônico correspondente ($\mathbb{K}^\delta$ ou $\mathbb{K}^\varepsilon$ respectivamente). Note também que, nestas condições, as funções $\texttt{A}.F$ e $(\texttt{B}.F) \circ \Gamma$ coincidem em todo o domínio $\mathbb{A}^\delta$, e não apenas no conjunto $\mathbb{X}$.

Por exemplo, suponha que desejamos colar um bloco prismático `prism`, de multi-grau $(2, 2)$, com um bloco tetraédrico `tetra`, de grau (2), como na figura 12.6.

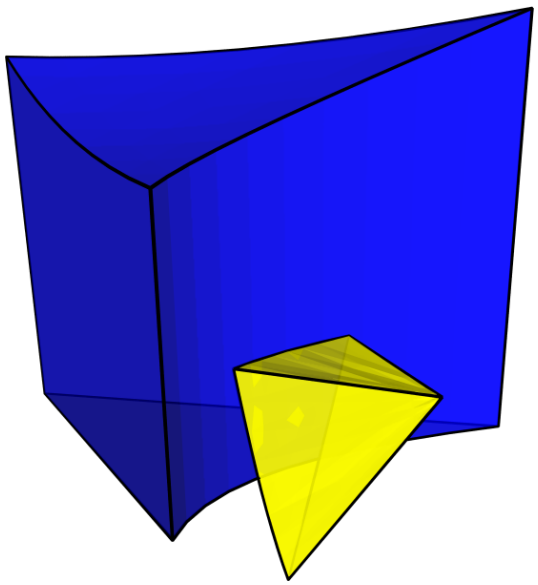


Figura 12.6: Colagem não-conforme de um bloco `prism` de domínio $\mathbb{K}^{(2,1)}$ e grau $(2, 2)$ com um bloco `tetra` de domínio $\mathbb{K}^{(3)}$ e grau (2).

.

Para isso, é necessário definir: (a) as facetas que participarão da colagem, e (b) o mapeamento de colagem Γ. No caso, trata-se do mapa $\Gamma : \mathbb{A}^{(2)} \to \mathbb{A}^{(1,1)}$ ilustrado na figura 12.7.

Para obtermos o conjunto de restrições entre os parâmetros de `prism` e `tetra` que garantem esta colagem, procedemos da seguinte forma:

1. Obtemos a faceta $(0, 2)$ de `prism`, que chamaremos de `f_prism`. Esta faceta é um bloco de domínio $\mathbb{K}^{(1,1)}$ e grau $(2, 2)$.

2. Obtemos a faceta $(0, 0)$ de `tetra`, de domínio $\mathbb{K}^{(2)}$ e grau (2), que chamaremos de `f_tetra`.

3. Reparametrizamos a faceta `f_prism` através da sua composição com o mapeamento Γ para obter o bloco simplicial `f_prism_a2`. Note que, como demonstrado no capítulo 10, este novo bloco tem domínio $\mathbb{K}^{(2)}$ e suas componentes são de grau (4). O objetivo é fazer `f_prism_a2`.F coincidir com `f_tetra`.F em todo $\mathbb{A}^{(2)}$.

4. Elevamos o grau da faceta `f_tetra` de (2) para (4), obtendo o novo bloco `f_tetra_rD`. Esta faceta é um bloco de domínio $\mathbb{K}^{(2)}$ e grau (4), o mesmo que `f_prism_a2`.

5. Adicionamos ao repositório `eqs` do modelo as restrições que igualam cada coeficiente de Bézier de `f_prism_a2` ao coeficiente correspondente de `f_tetra_rD`. Note que, para isso, é necessário utilizar as matrizes `f_prism_a2.kind.M` e `f_tetra_rD.kind.M` para substituir cada coeficiente de Bézier pela combinação dos parâmetros externos apropriada.

6. Finalmente, descartamos os blocos auxiliares criados: `f_prism`, `f_prism_a2`, `f_tetra` e `f_tetra_rD`.

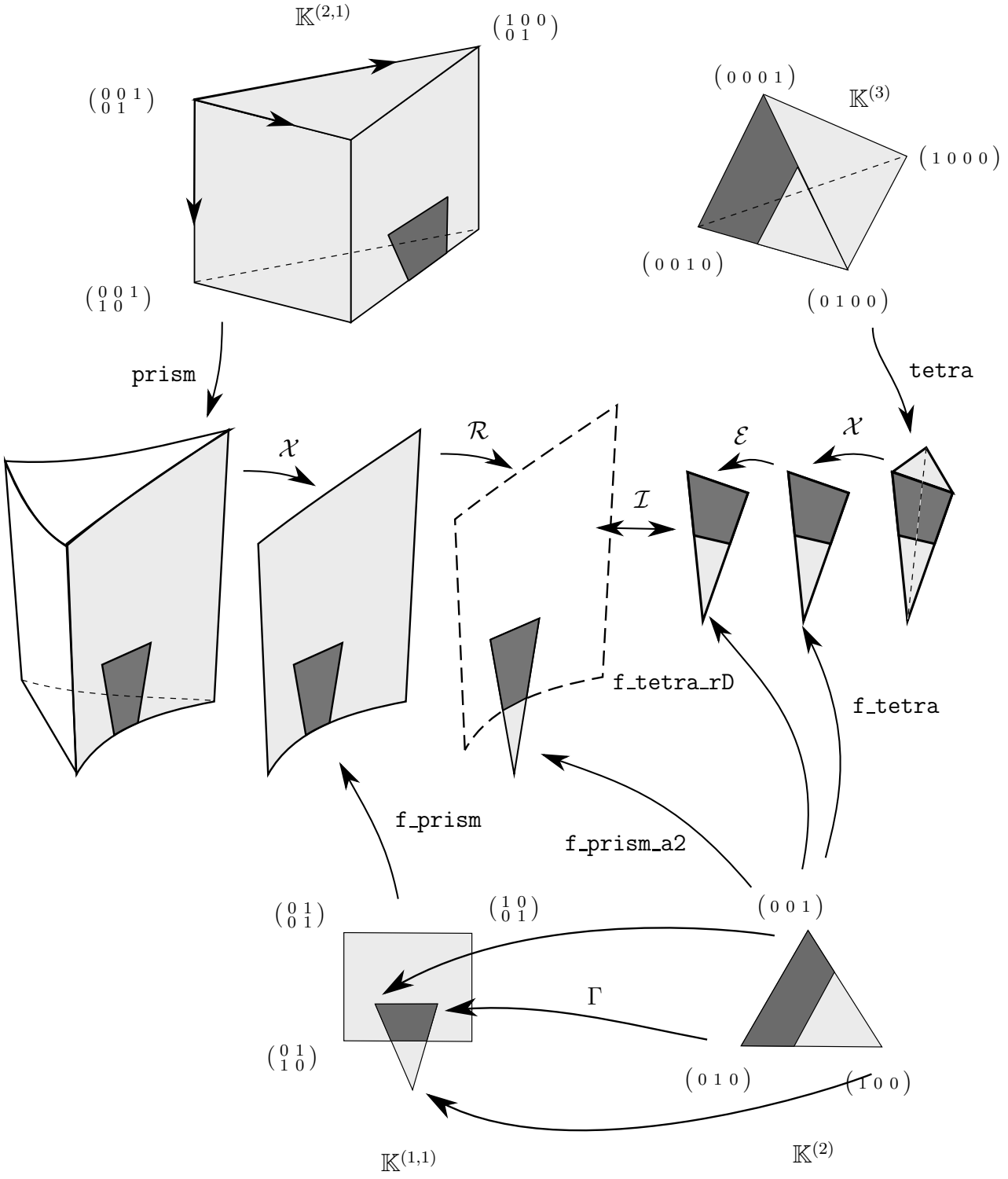


Figura 12.7: Colagem não-conforme. Ilustração dos blocos para colagem, seus domínios e o mapeamento de colagem. Denotamos por $\mathcal{X}$, $\mathcal{E}$, $\mathcal{R}$ e $\mathcal{I}$ as operações de extração de uma faceta, elevação de grau, reparametrização e identificação de coeficientes de Bézier, respectivamente.

12.4 Colagem com imposição de suavidade

Na colagem de dois blocos, frequentemente é necessário impor *restrições de suavidade*, além das restrições de continuidade.

Por exemplo considere dois blocos hexaédricos (domínio $\mathbb{K}^{1,1,1}$) colados por uma faceta como na figura 12.8, com mapa de colagem Γ trivial:

$$\Gamma(U) = \begin{pmatrix} U_{00} - 1 & U_{01} + 1 \\ U_{10} & U_{11} \\ U_{20} & U_{21} \end{pmatrix}.$$

Para conseguir a simples continuidade, basta impor que o ponto $\mathtt{A}.F(U) = \mathtt{B}.F(\Gamma(U))$ para todo U da faceta $\mathbb{K}^{1,1,1}|_{0,1}$ do domínio de $\mathtt{A}$. A colagem terá *suavidade (paramétrica) de primeira ordem* se a derivada em qualquer direção $\Xi \in \mathbb{V}^{1,1,1}$ de $\mathtt{A}.F$ em qualquer ponto U dessa faceta for igual à derivada de $\mathtt{B}.F$, na mesma direção Ξ, no ponto $\Gamma(U)$. Observe que, para direções Ξ paralelas à faceta comum, esta condição decorre automaticamente da condição de continuidade. Observe também que o operador de derivada direcional de primeira ordem D^1_Ξ é linear na direção Ξ. Portanto, se a continuidade já foi assegurada, basta garantir esta condição para uma direção Ξ transversal à faceta.

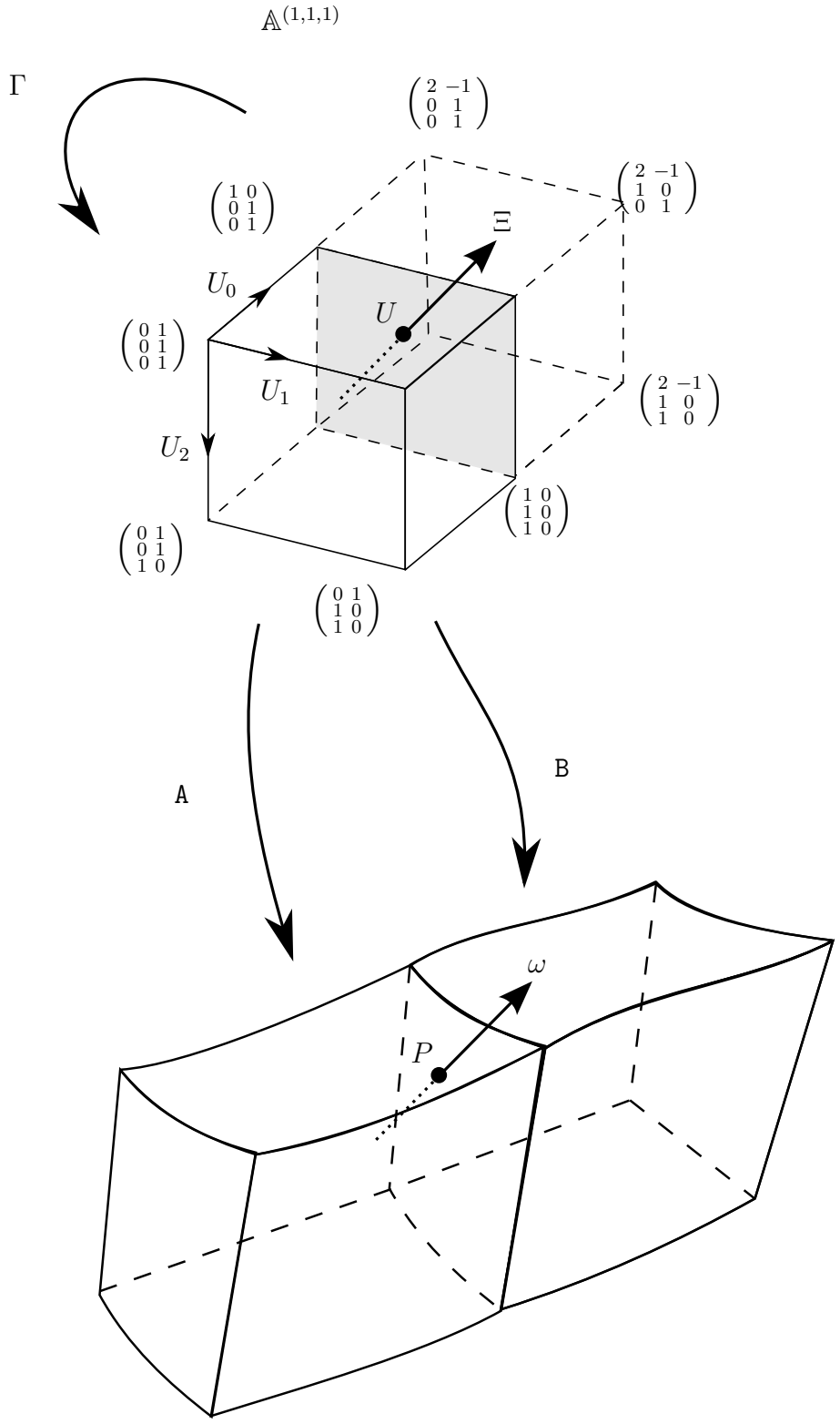


Figura 12.8: Colagem com suavidade. Na figura, o ponto $P \in \mathbb{R}^3$ é $\mathtt{A}.F(U) = \mathtt{B}.F(\Gamma(U))$, e o vetor $\omega \in \mathbb{R}^3$ é $(D_{\Xi}\mathtt{A}.F)(U) = (D_{\Xi}\mathtt{B}.F)(\Gamma(U))$.

Capítulo 13

A biblioteca bezEl

O código da BezEl está disponível para download no link https://bitbucket.org/lbfreitas/bezel

13.1 Estruturas de indexação

A biblioteca inclui classes destinadas a representar multi-,hiper-, e ultra-índices (vetores, matrizes e tensores irregulares de inteiros).

13.1.1 Multi-índices

A representação de multi-índices consiste na classe `MultiIndex`, que possui apenas dois atributos: sua dimensão `dim` e em um vetor `elem` de inteiros de tamanho `dim+1`. Por exemplo, o multi-índice $(2, 1, 3, 7)$ pode ser instanciado com os comandos

```
MultiIndex* m = new MultiIndex(3);
m->elem[0] = 2;
m->elem[1] = 1;
m->elem[2] = 3;
m->elem[3] = 7;
```

Apesar de uma multi-dimensão ou um multi-grau ser apenas um uso específico de um multi-índice, para maior clareza do código, criamos as classes `MultiDimension` e `MultiDegree`, derivadas da classe `MultiIndex`.

Além das operações básicas descritas na seção 4.1, esta classe implementa uma série de métodos para enumerar multi-índices sujeitos a vários tipos de restrições. Estes métodos devolvem um vetor de multi-índices (classe `std::vector<MultiIndex*>`) contendo os multi-índices desejados. A tabela 13.1 relaciona alguns dos métodos de enumeração da classe `MultiIndex`. Nesta tabela, `lambda` e `mu` representam os multi-índices λ e μ respectivamente, e `enum` e `enumAux` são os vetores de multi-índices resultantes.

Tabela 13.1: Enumeração de multi-índices

Conjunto de multi-índices	Implementação
$\{\kappa \in \mathbb{I}_d : \kappa < \lambda\}$	`enum = lambda->getLessThan()`
$\{\kappa \in \mathbb{I}_d : \mu < \kappa < \lambda\}$	`enumAux = lambda->getLessThan();` `enum = mu->selectGreaterFrom(enumAux);`
$\{\kappa \in \mathbb{I}_d^g\}$	`enum = MultiIndex::getMultiIdxSet(g,d)`
$\{\kappa \in \mathbb{I}_d : \lvert\kappa\rvert \leq g\}$	`enum = MultiIndex::getMultiIdxSetLessEq(g,d)`

A ordem dos multi-índices definida por estes métodos é utilizada em todos os contextos em que é necessário para converter um multi-índice em um índice inteiro, por exemplo para armazenar os coeficientes de Bézier de um polinômio simplicial.

13.1.2 Hiper-índices

A classe `HiperIndex` representa os hiper-índices. Sua implementação consiste basicamente em um vetor `line` de multi-índices, um para cada linha do hiper-índice, e em um multi-índice `dim` que define o tamanho de cada linha. Por exemplo, o hiper-índice $((1,0),(2,5,3))$ pode ser representado por uma instância `Lambda` desta classe, com `Lambda.dim` $= (1,2)$. Esta instância pode ser criada pelos comandos

```
MultiIndex* delta = new MultiIndex(1);
delta->elem[0] = 1;
delta->elem[1] = 2;

HyperIndex* Lambda = new HyperIndex(delta);
Lambda->allocate();
Lambda->line[0]->elem[0] = 1;
Lambda->line[0]->elem[1] = 0;
Lambda->line[1]->elem[0] = 2;
Lambda->line[1]->elem[1] = 5;
Lambda->line[1]->elem[2] = 3;
```

Além das operações básicas descritas na seção 4.2, esta classe também implementa métodos para a enumeração de um conjunto de hiper-índices. Mais especificamente, dados dois multi-índices α e δ, representados por `alpha` e `delta`, o conjunto dos hiper-índices $\Lambda \in \mathbb{H}_\delta^\alpha$ pode ser obtido utilizando o método estático

```
std::vector<HyperIndex*> getAllIndexes(MultiIndex* alpha,MultiIndex* delta);
```

O ordem definida por este método é a utilizada em todos os contextos em que é necessário linearizar um conjunto de hiper-índices, por exemplo para armazenar os coeficientes de Bézier de um polinômio simploidal.

13.1.3 Ultra-índices

Os ultra-índices são implementados por instâncias da classe `UltraIndex`. Na biblioteca BezEl, esta classe está restrita a ultra-índices em que todos os planos tem o mesmo formato, isto é, a elementos de $\mathbb{U}_{p,\delta}$, para algum $p \in \mathbb{N}$ e algum $\delta \in \mathbb{I}$. Os atributos desta classe são: o número de planos `nPlanes`, a multi-dimensão (compartilhada) `dim` dos seus planos; e um vetor `plane` de hiper-índices, de tamanho `nPlanes`.

As operações de soma de planos e fatorial são implementadas pelos métodos `sum` e `factorial`, respectivamente. Outro método importante é

`vector<UltraIndex*> getAllIndexes(int p, MultiIndex* d, HyperIndex* s);`

que enumera todos os ultra-índices de `p`+1 planos, cada um de dimensão `d` e cuja soma dos planos é o hiper-índice `s`.

13.2 Espaços multi-afins

Outro conjunto de classes é usado para representar pontos e vetores de espaços multi-afins ($\mathbb{A}^\delta$ e $\mathbb{V}^\delta$), que são matrizes irregulares de números reais. Pontos e vetores de espaços afins são tratados como casos particulares.

Na biblioteca BezEl, todas as coordenadas são instâncias de uma classe `Number`, que pode ser concretizada pelo tipo `double` ou por outros tipos (como racionais exatos).

13.2.1 Matrizes irregulares

Uma matriz irregular é implementada por uma instância da classe `IrregularMatrix`. Seus atributos são a multi-dimensão `mD` da matriz e uma matriz bidimensional irregular `data` de ponteiros para os elementos da matriz, da classe `Number`. Mais especificamente, cada posição `data[i]` aponta para um vetor de referências de números reais de tamanho `mD.elem[i]`. Por exemplo, a matriz irregular `M` $= ((1.3, 0.7), (2.1, 5.7, 3.0))$ pode ser criada pelos comandos

```
MultiDimension* delta = new MultiDimension(1);
delta->elem[0] = 1; delta->elem[1] = 2;

IrregularMatrix* M = new IrregularMatrix(delta);
```

```
M->allocate();
M->data[0][0] = new Number(1.3);
M->data[0][1] = new Number(0.7);
M->data[1][0] = new Number(2.1);
M->data[1][1] = new Number(5.7);
M->data[1][2] = new Number(3.0);
```

13.2.2 Matrizes regulares

Matrizes regulares bidimensionais são representadas por instâncias da classe `Matrix`, derivada da classe `IrregularMatrix`. Além dos atributos herdados, a classe `Matrix` possui os atributos `n_rows` e `n_cols` que indicam o número de linhas e colunas da matriz. Esta classe dispõe de métodos para realizar algumas operações da álgebra matricial como o produto à esquerda (`leftMultiply`), produto à direita (`rightMultiply`), tranposição (`transpose`), etc.

13.2.3 Pontos e vetores de espaços multi-afins

Um ponto de um espaço multi-afim $\mathbb{A}^\delta$ é implementado por uma instância da classe `DomainPoint`. Esta classe, derivada da classe `IrregularMatrix`, tem apenas os atributos herdados desta última. O mesmo vale para vetores de um espaço tangente multi-afim $\mathbb{V}^\delta$, representados por instâncias da classe `DomainVector`.

13.2.4 Mapeamento afim

A classe `DomainMapping` implementa mapeamentos afins entre espaços multi-afins, descritos no capítulo 9. O domínio e o contra-domínio são espaços multi-afins determinados por suas multi-dimensões, armazenadas nos atributos `dD` and `rD`, respectivamente. Os coeficientes do mapeamento, descritos na equação (9.1), são armazenados na forma de uma matriz bidimensional irregular `coef` de instâncias da classe `IrregularMatrix`. Mais especificamente, `coef[r][s]` é a matriz irregular de multi-dimensão `dD` que contém os coeficientes M_{ij}^{rs} para todo $i = 0, \cdots,$ `dD.dim` e todo $j = 0, \cdots,$ `dD.elem[i]`. Uma instância `M` desta classe tem um método `M->map(DomainPoint* U)` que retorna a imagem do ponto `U` pelo mapeamento `M`.

13.3 Polinômios Simploidais

A classe `BezierSimploidPoly` implementa um polinômio simploidal na representação de Bézier. Seus atributos são a multi-dimensão `dD` do seu domínio (uma instância da classe `MultiDimension`), seu multi-grau `deg` (uma instância da classe `MultiDegree`), um vetor `coef` contendo seus coeficientes de Bézier (ponteiros para a classe `Number` e o tamanho `n_coef` deste vetor. A ordem dos coeficientes neste vetor é definida pelo método estático `getAllIndexes` da classe `HyperIndex` (seção 13.1.2). Observe que estes coeficientes não são armazenados no repositório de parâmetros (descrito no capítulo 11). Uma instância `p` desta classe tem os seguintes métodos principais

- `p->evaluate(DomainPoint* U)`

  Este método retorna o valor do polinômio `p` calculado no ponto `U`, que deve ser um ponto do espaço-multi-afim de multi-dimensão `dD`. O resultado é do tipo `Number*`.

- `p->get_Poly_of_Facet(int k, int i)`

  Este método retorna um novo polinômio `q` que corresponde à restrição do polinômio `p` à faceta de índices `k,i` do simplóide que é seu domínio.

- `p->get_Matrix_of_Facet(int k, int i)`

  Este método retorna uma matriz `K` que relaciona os coeficientes de Bézier do polinômio `p` com os do polinômio `q` que corresponde à restrição do polinômio `p` à faceta de índices `k,i` do simplóide que é seu domínio.

- `p->get_Matrix_of_RaisedDegree(MultiDegree* new_deg)`

  Este método retorna uma matriz `K` que relaciona os coeficientes de Bézier do polinômio `p` com os do polinômio `q`, idêntico a `p`, porém representado como um polinômio de grau `new_deg`, que deve ser maior ou igual a `p->deg`.

- `p->get_Matrix_of_DirectionalDerivative(int r, DomainVector* Xi)`

  Este método retorna uma matriz `K` que relaciona os coeficientes de Bézier do polinômio `p` com os do polinômio `q`, que corresponde à r-ésima derivada direcional do polinômio `p` na direção `Xi`, que deve ser um vetor tangente ao espaço multi-afim de dimensão `dD`.

- `p->get_Matrix_of_Composition_w_AffineMap(DomainMapping* dm)`

  Este método retorna uma matriz `K` que relaciona os coeficientes de Bézier do polinômio `p` com os do polinômio `q` que corresponde à composição do polinômio `p` com o mapeamento afim `dm`; ou seja `q->evaluate(U)` é equivalente a `p->evaluate(dm->map(U))`.

A matrix `K`, retornada nos quatro métodos acima é representada por uma instância da classe `Matrix`. O coeficiente `q->coef[i]` pode ser então calculado acumulando o produto `p->coef[j] * K->data[j][i]` para todo $j = 0, \cdots,$ `n_coef`.

13.4 Elementos de Bézier simploidais

Lembramos que um bloco de Bézier simploidal (definido no capítulo 11) representa uma função de um simplóide $\mathbb{K}^\delta$ para um espaço cartesiano $\mathbb{R}^m$, onde cada uma das componentes da imagem é um polinômio simploidal possivelmente de grau distinto.

13.4.1 Tipos de bloco

Como discutido no capítulo 11, a classe `BlockKind`, descreve as características de um conjunto de blocos com um mesmo domínio e contra-domínio, mesmo grau e mesma relação entre parâmetros internos e externos. Os atributos da classe `BlockKind` são a multi-dimensão `dD` $= \delta$ do domínio, a dimensão `rD` $= m$ do contra-domínio , os multi-graus `deg` de cada componente, o número de parâmetros internos `n_int` e externos `n_ext`, e a matriz `M` que converte parâmetros externos em internos. A dimensão do domínio `dD` é uma instância de `MultiDimension` enquanto que a dimensão do contra-domínio `rD` é um inteiro. Os multi-graus `deg` são representados por um vetor de tamanho `rD` de instâncias da classe `MultiDegree`. A matriz `M` é uma instância da classe `Matrix`.

Por exemplo, um tipo de bloco `brickKind` que descreve blocos do que compõem a malha da figura 11.1 pode ser criado pelos comandos

```
MultiDimension* mD = new MultiDimension(2);
mD->elem[0] = 1;  mD->elem[1] = 1;  mD->elem[2] = 1;

MultiDegree* deg[3];
deg[0] = new MultiDegree(2);
deg[0]->elem[0] = 1;
deg[0]->elem[1] = 0;
deg[0]->elem[2] = 0;
deg[1] = new MultiDegree(2);
deg[1]->elem[0] = 0;
deg[1]->elem[1] = 1;
deg[1]->elem[2] = 0;
deg[2] = new MultiDegree(2);
deg[2]->elem[0] = 3;
deg[2]->elem[1] = 3;
deg[2]->elem[2] = 1;
```

```
BlockKind* brickKind = new BlockKind(mD,3,deg);
//  by default:  brickKind->M = NULL
```

Os principais métodos de uma instância `bK` são

- `bK->getInternalVars(int *varId)`

  Este método aplica a matriz `bK->M` de conversão de parâmetros externos para internos. O resultado é um vetor de coeficientes (de tipo `Number*`) representados por uma instância da classe `Matrix` com uma única coluna.

- `bK->get_Kind_of_RaisedDegree(MultiDegree** new_deg)`

  Este método cria um novo tipo de bloco `bK_raised` para descrever blocos com mesmo domínio e contra-domínio do tipo `bK` porém com graus descritos pelo vetor `new_deg` com `bK->rD` instâncias da classe `MultiDegree`. Cada bloco deste tipo terá o mesmo número de parâmetros externos (`bK->n_ext`) mas um número diferente de parâmetros internos. A matriz `bK_raised->M` é calculada de modo que os mesmos parâmetros externos produzirão polinômios idênticos, apesar de formalmente de graus distintos.

- `bK->get_Kind_of_DirectionalDerivative(int r, DomainVector* Xi)`

  Este método cria um novo tipo de bloco `bK_deriv` para descrever blocos com mesmo domínio e contra-domínio do tipo `bK`. Os graus do tipo `bK_deriv`, entretanto, dependem a ordem `r` da derivada e de sua direção `Xi`. Cada bloco deste tipo terá o mesmo número de parâmetros externos (`bK->n_ext`) mas um número diferente de parâmetros internos. A matriz `bK_deriv->M` é calculada de modo que os mesmos parâmetros externos produzirá os coeficientes de Bézier da r-ésima derivada direcional do bloco do tipo `bK`.

- `bK->get_Kind_of_Composition_w_DM(DomainMapping* dm)`

  Este método cria um novo tipo de bloco `bK_dm` para descrever blocos oriundos da composição de blocos do tipo `bK` com o mapeamento afim `dm`. Este novo tipo `bK_dm` tem o mesmo domínio do mapeamento afim `dm->dD` e mesmo contra-domínio do tipo `bK->rD`. Cada bloco do tipo `bK_dm` terá o mesmo número de parâmetros externos (`bK->n_ext`) mas um número diferente de parâmetros internos. A matriz `bK_dm->M` é calculada de modo que os mesmos parâmetros externos produzirá os coeficientes de Bézier da composição.

- `bK->get_Kind_of_Facet(int k, int i)`

- `bK->get_IndexMap_of_Facet(int k, int i)`

  O primeiro método, `bK->get_Kind_of_Facet(int k, int i)`, cria um novo tipo de bloco `bK_facet` para descrever blocos que são facetas de blocos do tipo `bK`. Este novo tipo `bK_facet` tem mesmo contra-domínio de `bK`. Entretanto, seu domínio e graus dependem não apenas do domínio de `bK` mas também da faceta escolhida. Cada bloco deste novo tipo terá um número menor de parâmetros externos. A matriz `bK_facet->M` é calculada de modo que um subconjunto dos parâmetros externos de um bloco do tipo `bK` produzirá os coeficientes de Bézier sua faceta. Este subconjunto é calculado pelo segundo método, `bK->get_IndexMap_of_Facet(int k, int i)`. Este método retorna um vetor com os `bK_facet->n_ext` índices dos parâmetros externos que pertencerão à faceta.

13.4.2 Blocos de Bézier

Como descrito no capítulo 11, cada elemento de Bézier é implementado por uma instância da classe `Block`. Seus atributos são um ponteiro `model` para o modelo ao qual pertence (vide seção 13.5), o índice `bId` do bloco na lista de blocos do modelo (seção 13.5.1), um vetor `varId` contendo os índices de seus parâmetros externos no repositório (seção 13.5.2), e uma referência `kind` para o tipo do bloco (instância de `BlockKind`). Cada instância possui também um atributo `mark`, utilizado temporariamente por algoritmos de percursos e visualização. Observe que os parâmetros internos do bloco não são armazenados permanentemente.

Os métodos mais importantes de uma instância `b` desta classe são

- `b->explicit()`

  Este método converte o bloco em uma lista de polinômios simploidais explícitos, um para cada componente, na forma de um vetor de `b->rD` instâncias da classe `BezierSimploidPoly`. Os coeficientes de Bézier dos polinômios são calculados usando a matriz `b->kind->M`.

- `b->getSubBlock_using_Imap(BlockKind* bK, int *idx_map)`

  Este método cria um novo bloco `c` do tipo `bK` cujos parâmetros externos são um subconjunto dos parâmetros externos do bloco `b`. O subconjunto (e sua ordem) é especificado pelo vetor de inteiros `idx_map`, de tamanho `b->kind->n_ext`, tal que se `idx_map[j]` $= k$, então `c->varId[j]` $=$ `b->varId[k]`.

- `b->getSubBlock(BlockKind* newbK)`

  Este método é um caso particular do método `b->getSubBlock_using_Imap(BlockKind* bK, int *idx_map)`, quando o vetor `idx_map` é a identidade; ou seja, todos os

parâmetros externos do bloco `b` são também parâmetros externos do bloco `c = b->getSubBlock(newbK)`.

13.5 Modelos

Um modelo geológico é representado por um objeto da classe `Model` que possui quatro atributos: uma coleção de blocos de Bézier `blks`, um repositório `pDep` com os parâmetros externos desses blocos, um repositório `eqDep` de equações entre esses parâmetros, e uma estrutura topológica `topo` que especifica as relações de adjacência entre blocos.

13.5.1 Coleção de blocos

A coleção `blks` é uma instância da classe `std::vector` cujos elementos são instâncias da classe `Block`. O índice de cada bloco `b` neste vetor é seu atributo `b->bId`.

13.5.2 Repositório de parâmetros externos

O repositório de parâmetros externos (veja capítulo 11) é implementado por uma instância `pDep` da classe `ParamDepot`. O número de parâmetros do modelo é armazenado no atributo inteiro `pDep.nP`, e os valores correntes destes parâmetros (instâncias da classe `Number`) são armazenados em um vetor `pDep.data` nas posições `[0, ... ,pDep.nP-1]`. Novos parâmetros são acrescentados ao modelo pelo método `reserveNewVars(int qty)`, que estende o vetor `pDep.data` se necessário e retorna um vetor de `qty` inteiros, os índices dos parâmetros reservados. Os parâmetros externos de um bloco `b` são `pDep.data[b.varId[k]]`, para k variando de 0 a `b.kind.n_ext-1`. A atribuição e leitura dos valores de parâmetros armazenados em um repositório `pDep` são implementado pelos métodos

```
pDep->setVar(int varId, Number* value);
pDep->getVar(int varId);
```

13.5.3 Equações entre variáveis

As restrições entre blocos (capítulo 12) são implementadas pela classe `Equation`. Cada instância desta implementa uma restrição linear entre um ou mais parâmetros externos do modelo, armazenados no repositório `pDep`. Uma instância `e` desta classe possui um atributo inteiro `size` que é o número de parâmetros envolvidos na equação; um vetor `varId`, de tamanho `size`, contendo os índices desses parâmetros no repositório; e um vetor `coefs`, também de tamanho `size`, que armazena os coeficientes de cada parâmetro na

equação. O termo independente é representado pelo atributo `independent`. Os métodos

```
e->add(Equation* op);
e->sub(Equation* op);
e->mul(Number* s);
e->div(Number* s);
```

realizam as operações de soma e subtração da equação `op`, e multiplicação e divisão pelo escalar `s` e retornam uma nova equação. Há também um método `e->collapse()`, que cria um anova equação eliminando os termos cujo coeficiente é nulo.

13.5.4 Repositórios de equações

A classe `EquationDepot` implementa um repositório de equações. Seus atributos são: o número `nE` de equações e um vetor `eqs` de tamanho `nE` contendo apontadores para instâncias da classe `Equation`. Equações podem ser adicionadas a um repositório `eDep` uma-a-uma, utilizando o método `eDep->addRestriction` ou em grupos, utilizando o método `eDep->addRestrictionSet`.

13.5.5 Estrutura topológica

A topologia de um modelo `mod` é representada por uma instância `mod->topo` da classe `AdjacencyList`, que implementa listas de adjacências. Além de um ponteiro para o modelo ao qual pertence, esta classe contém um único atributo, o vetor `topo->list` (uma instância de std::vector) tal que `mod.topo.list[`i`]` são as adjacências do bloco `bi = mod.blks[`i`]`. Esta informação é implementada por outros `bi.nfacets` vetores de triplas do tipo `AdjLstElem`, uma para cada faceta do domínio de `bi`. O método `mod->topo->getAdjBlocks(`i`,`k`,`l`)` retorna o vetor de triplas de adjacências do bloco `bi` pela sua (k,l)-faceta. Cada tripla `tbklm = mod->topo->getAdjBlocks(`i`,`k`,`l`)[`m`]` descreve o m-ésimo vizinho do bloco `bi` adjacente à faceta (k,l). O índice `tiklm.adjId` armazena a posição do bloco vizinho em `mod.blks`. O atributo `tiklm.adjFacet` armazena os índices da faceta de `mod.blks[tiklm.adjId]` através da qual se verifica a adjacência; e o mapeamento afim `timlm.dm` relaciona o domínio desta faceta com o domínio da faceta correspondente do bloco `bi`. (Lembre que a biblioteca BezEl supõe que esse mapeamento afim é constante.)

13.6 Classes adicionais

13.6.1 Visualização

A visualização dos blocos de Bézier representados pela bezEl é realizada com o auxílio do traçador de raios POVRay. A classe `Model_vis` é a mais importante nesta tarefa. Uma instância `vis` desta classe representa facetas de blocos por malhas de triângulos planos com transparência, arestas (e grade de controle) por seqüências de cilindros e pontos de controle de Bézier por esferas. O atributo `vis->model` é um ponteiro para o modelo a ser visualizado. Outros atributos são o vetor de strings `vis->color` e o vetor de números reais `vis->transmit`, ambos de tamanho `vis->nPigments`. Mais especificamente, se `b` é um bloco pertencente ao modelo `vis->model`, `b` será pintado com cor `vis->color[b->mark]` e transparência `vis->transmit[b->mark]`. Note que a string de cor deve ter o formato do POVRay, por exemplo `rgb<0.4,0.1,0.0>`.

Os métodos de uma instância `vis` mais importantes são

- `vis->initDomain(int nk11, int nk2)`

  Este método estabelece a resolução da representação das facetas. As facetas de domínio $\mathbb{K}^{(1,1)}$ são divididas em `nk11 * nK11` retângulos, e cada retângulo é dividido em dois triângulos planos. As facetas de domínio $\mathbb{K}^{(2)}$, por outro lado, são divididas em $4^{\texttt{nK2}}$ triângulos.

- `vis->plotPOV(FILE* fp)`

  Este método gera a representação POVRay do modelo `vis->mod` e escreve no arquivo apontado por `fp`. Observe que os blocos deste modelo marcados com valores negativos ou maiores ou iguais a `vis->nPigments` não são exportados para esta representação.

13.6.2 Coletor de lixo

A classe `SmartPointer` implementa um coletor de lixo simples para algumas classes da BezEl como `Number`, `MultiIndex`, `HyperIndex`, etc. A inicialização do coletor é realizada através do método `initSmartPointers()`. O coletor armazena internamente uma pilha de ponteiros `SmartPointer.pointers` de objetos de tipos arbitrários. Após inicializado, cada criação de uma nova instância de uma das classes acima armazena um ponteiro nesta pilha. Os métodos mais importantes são

- `SmartPointer.get_next()`

  Este método retorna o tamanho atual da pilha.

- `SmartPointer.delete_from(int p)`

  Este método remove e destrói todos os objetos da pilha depois da posição `from`.

Estes métodos são usados quando um procedimento cria um número grande de objetos temporários que serão destruídos ao final do procedimento. Neste caso, o procedimento começa executando `int p = SmartPointer.get_next()`, e antes de retornar executa `SmartPointer.delete_from(p)`.

Capítulo 14

Editor de modelos 2D

Para demonstrar a capacidade da biblioteca, implementamos um editor de modelos geofísicos bidimensionais. Este editor consiste em dois programas: (a) o editor de topologia e propriedades físicas e (b) o editor de geometria.

O editor de topologia e propriedades físicas, implementado como um Java applet, utiliza estruturas próprias e grava a descrição do modelo em um arquivo texto. O editor de geometria, implementado em Qt/C++, lê um arquivo texto e reconstrói o modelo utilizando as funções e estruturas de dados da biblioteca BezEl. Este editor permite ajustar as coordenadas verticais de certos pontos de controle de Bézier, preservando as restrições de suavidade de interfaces.

14.1 Editor de topologia e propriedades físicas

O editor de topologia (TopEdit) é usado para definir o número de fácies e interfaces, suas relações de adjacências e uma geometria inicial. A região de interesse é um retângulo com o eixo X representando a posição horizontal e o eixo Z a profundidade. O eixo Y é implícito; supõe-se que todas as propriedades são constantes nesta direção.

Uma seção de edição consiste de cinco etapas.

Na primeira etapa, a região de interesse é dividida por uma coleção de *linhas nodais* verticais, igualmente espaçadas. Veja a figura 14.1. O número de linhas é escolhido pelo usuário, e deve ser tanto maior quanto maior for a complexidade topológica do modelo.

Na segunda etapa, o usuário especifica uma série de interfaces geológicas. Cada interface é uma curva em que a profundidade Z é uma função da coordenada X. A curva pode começar ou terminar num dos lados verticais do modelo, ou na interseção de uma interface anterior com uma linha nodal. O editor exige que cada interface cruze pelo menos duas regiões entre linhas nodais. Veja a figura 14.2.

As linhas nodais dividem cada interface em arcos de Bézier de grau 3, conectados com

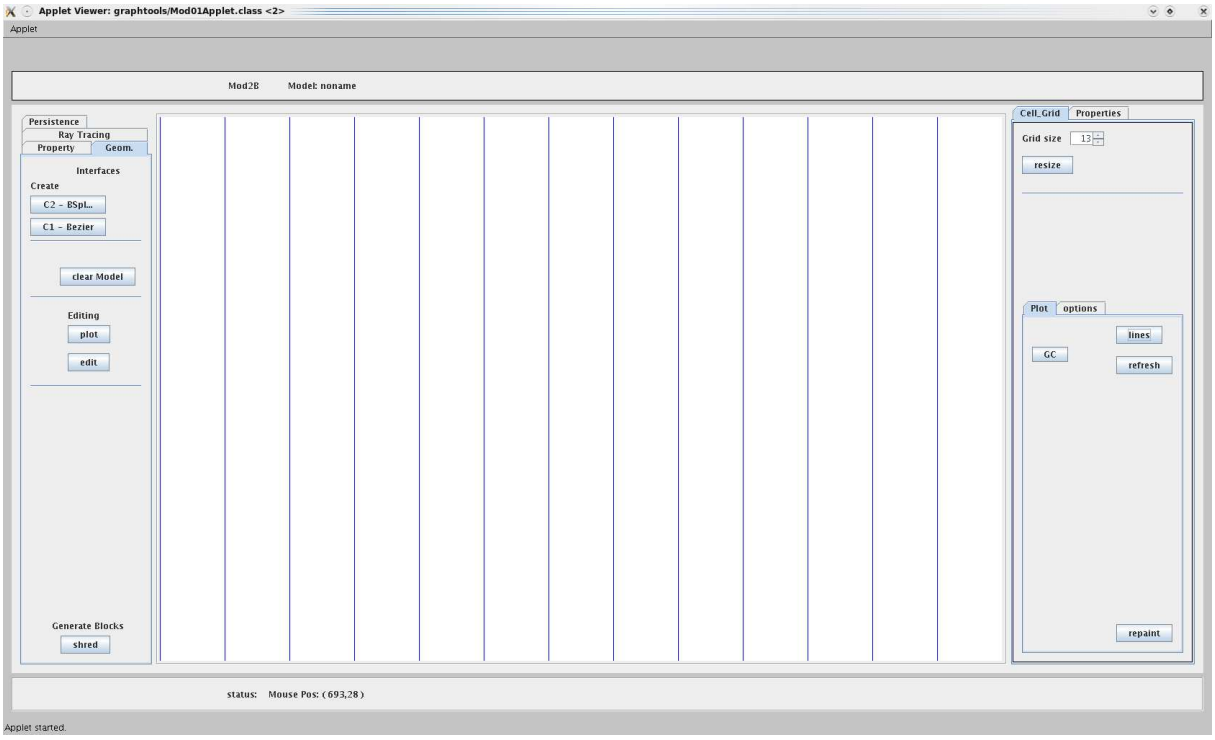


Figura 14.1: Primeira etapa: definição das linhas nodais sobre a região de interesse.

primeira derivada contínua. O usuário pode alterar a coordenada vertical dos pontos de controle de cada arco, e o editor modifica os pontos de controle dos arcos vizinhos de modo a manter a suavidade da interface.

Na terceira etapa, o editor identifica as fácies do modelo. Cada fácie é uma região conexa maximal delimitada superiormente e inferiormente por duas interfaces. Portanto, cada fácie pode ser delimitada à esquerda e à direita por um trecho de linha nodal (incluindo os lados verticais da região) ou pelo ponto de encontro de duas interfaces. Veja a figura 14.3.

Na quarta etapa, o editor particiona automaticamente cada fácie em um conjunto de blocos triangulares de Bézier de grau 3. Os três vértices de cada bloco são intersecções de linhas nodais com interfaces. Uma das arestas é vertical (parte de uma linha nodal) e o vértice oposto a ela está sobre uma das linhas nodais vizinhas. As outras arestas podem ser arcos de interface ou arestas "diagonais" escolhidas automaticamente pelo editor. Veja a figura 14.4.

Na última etapa, o usuário especifica a velocidade de propagação da onda em cada fácie. O usuário pode definir separadamente a velocidade de propagação ao longo das interfaces superior e inferior de cada fácie. Se os dois valores forem diferentes, entende-se que há uma variação suave de velocidade com a profundidade. Veja a figura 14.5.

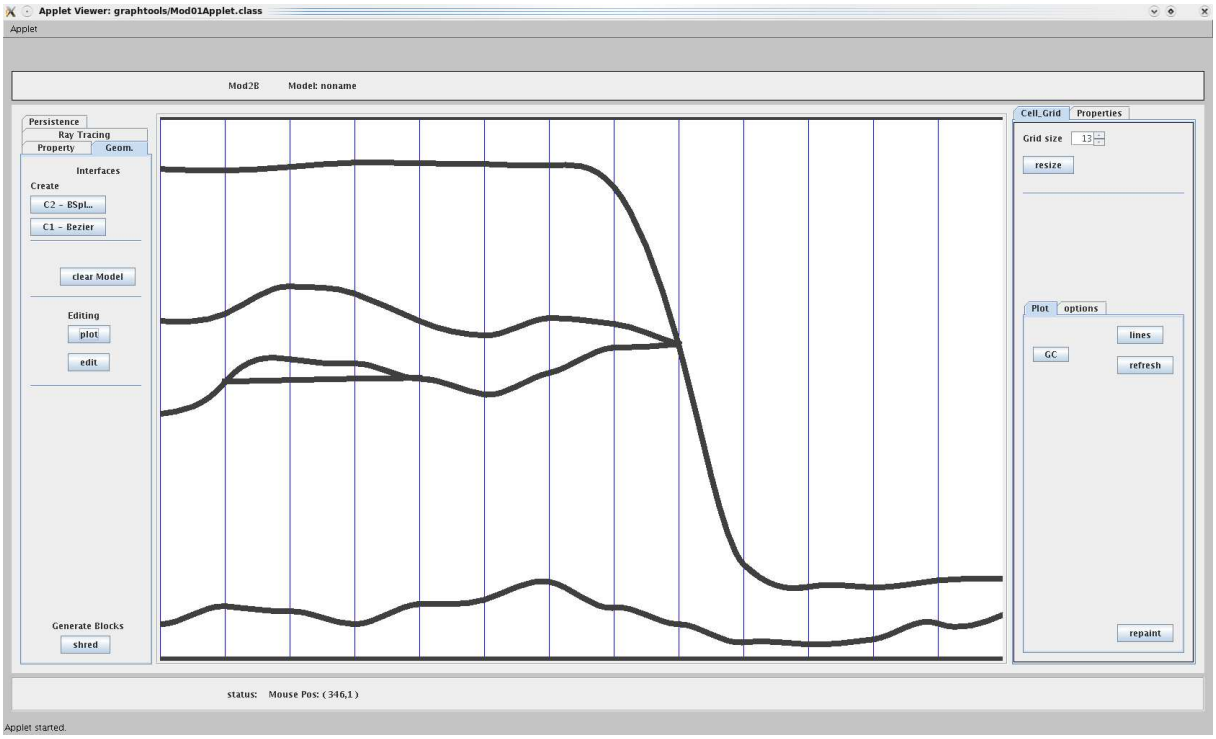


Figura 14.2: Segunda etapa: definição das interfaces (incluindo os limites superior e inferior do modelo)

O editor salva estas informações em um arquivo texto, que especifica o número de blocos, suas relações de adjacência e a correspondência entre as coordenadas locais de blocos adjacentes. Além disso, o arquivo especifica os blocos e as respectivas facetas que compõem cada interface.

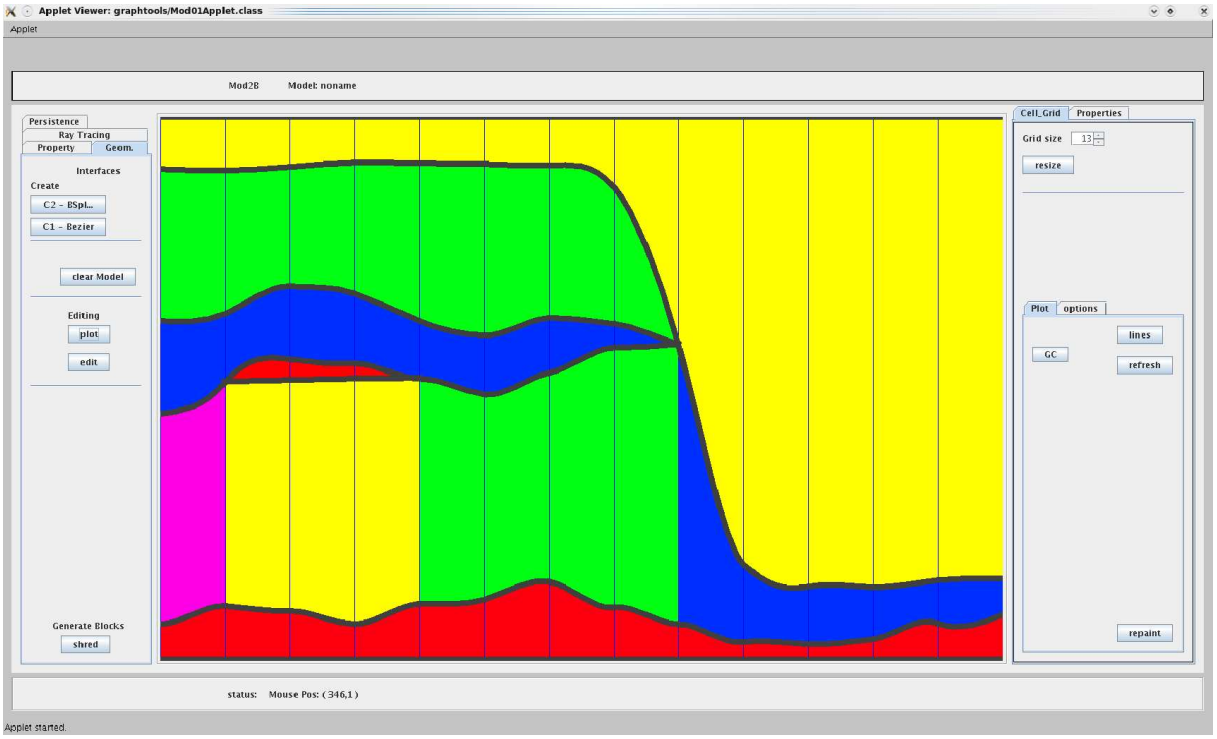


Figura 14.3: Terceira etapa: identificação das fácies (indicadas por cores arbitrárias).

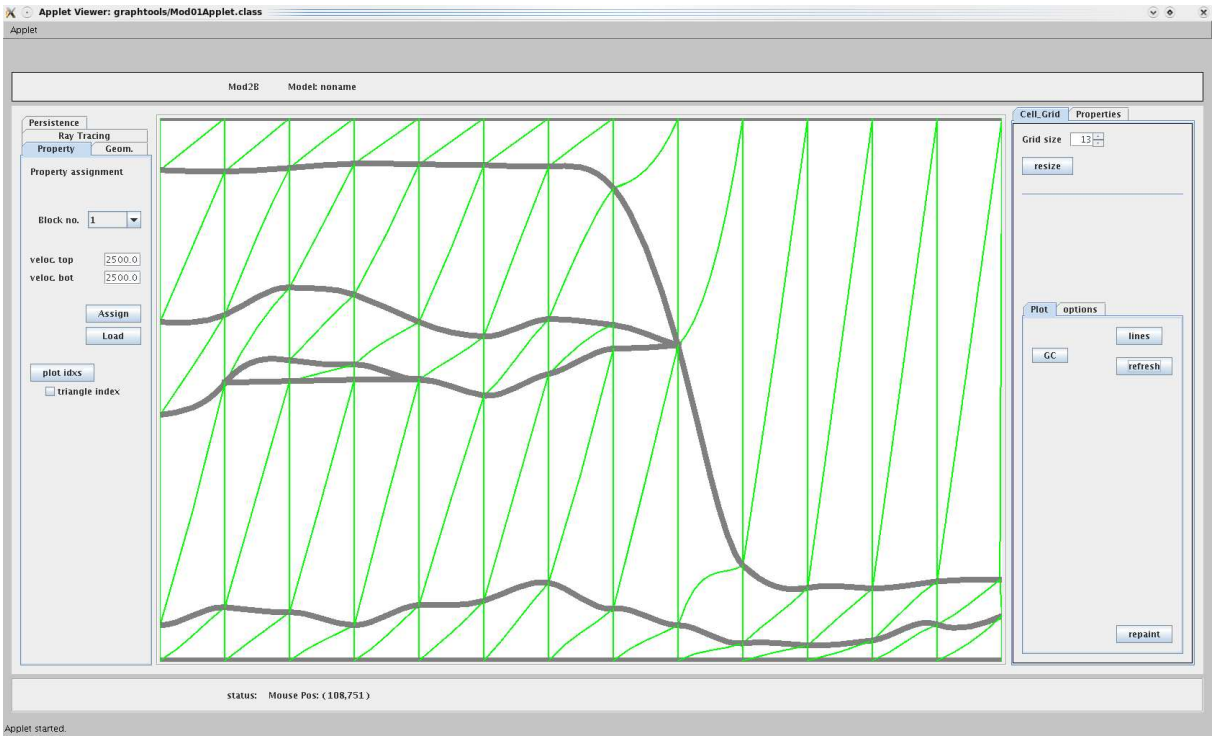


Figura 14.4: Quarta etapa: decomposição das fácies em blocos triangulares.

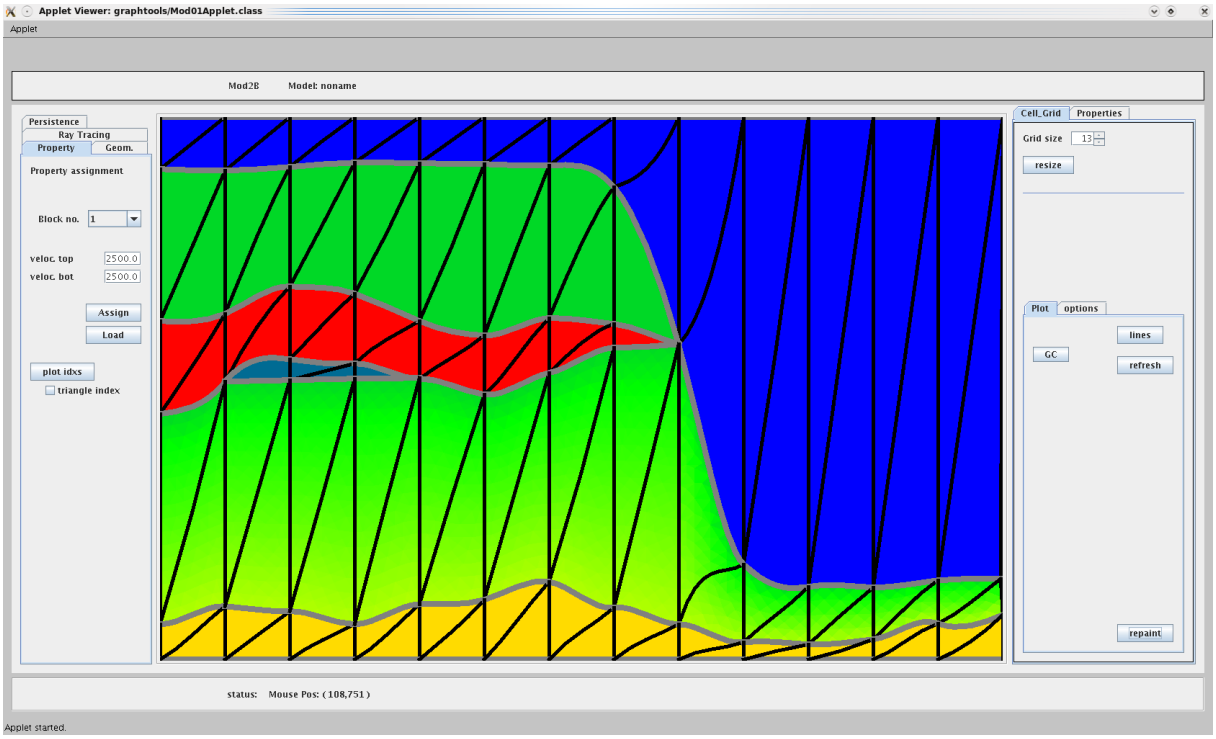


Figura 14.5: Quinta etapa: especificação das velocidades em cada fácie.

14.2 Editor de geometria

O editor de geometria (GeoEdit) lê o arquivo texto gerado pelo editor de topologia e reconstrói o modelo, agora utilizando as estruturas e funções da biblioteca BezEl. A reconstrução do modelo consiste de três etapas.

Na primeira etapa, o editor cria o modelo (classe `Model`) e os blocos da malha. Cada bloco é uma instância da classe `Block` (capítulo 11) que pode ter dois tipos possíveis, `lvKind` e `rvKind`, dependendo da posição da aresta vertical — no lado esquerdo ou direito do bloco, respectivamente. Os dois tipos têm domínio $\mathbb{K}^2$ e três componentes X, Z, e V, de grau (3), (3) e (1) respectivamente. Cada bloco tem 14 parâmetros externos, que são o índice l_N da linha nodal que contém sua aresta, os 10 coeficientes de Bézier da componente Z, e os três coeficientes de Bézier da componente V (que são as velocidades nos vértices do bloco). Cada bloco tem 23 parâmetros internos: 10 coeficientes de Bézier para cada componente X e Z, e três para a componente V. As matrizes de conversão tem a forma

$$\left(\begin{array}{c|c|c|c} D_{10\times 1} & 0_{10\times 10} & 0_{10\times 3} & A_{10\times 1} \\ \hline 0_{10\times 1} & I_{10\times 10} & 0_{10\times 3} & 0_{10\times 1} \\ \hline 0_{3\times 1} & 0_{3\times 10} & I_{3\times 3} & 0_{3\times 1} \end{array}\right)$$

As submatrizes A e D são

$$A = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \pm d/3 \\ \pm d/3 \\ \pm d/3 \\ \pm 2d/3 \\ \pm 2d/3 \\ \pm d \end{pmatrix}. \quad D = \begin{pmatrix} d \\ d \\ \\ \\ \\ \vdots \\ \\ \\ \\ d \end{pmatrix}.$$

onde d é a distância entre as linhas nodais, e o sinal $\pm$ é escolhido de acordo com o tipo (positivo para `lvKind.M` e negativo para `rvKind.M`).

Na segunda etapa, para cada par de blocos adjacentes, o editor acrescenta ao modelo um conjunto restrições que descrevem as adjacências entre os blocos, garantindo a ausência de frestas e superposições. (Estas restrições poderiam ter sido implementadas pelo compartilhamento de parâmetros externos, como descrito na seção 11.3; porém quando o editor foi escrito, esta funcionalidade não havia sido implementada.) Ainda nesta etapa, o editor também acrescenta ao modelo uma série de restrições que garantem a suavidade de cada interface. Estas restrições não são atualmente utilizadas pelo editor, mas poderiam ser utilizadas por outros programas, por exemplo em um algoritmo geral para a determinação automática de elementos finitos [19].

Na terceira etapa, o editor escolhe um subconjunto dos coeficientes de Bézier — os *coeficientes do modelo* — que podem ter sua componente vertical alterada arbitrariamente pelo usuário. Para cada um destes coeficientes, o editor determina também as alterações que devem ser feitas nos parâmetros externos dos blocos de modo a preservar as restrições de continuidade e suavidade. A modificação do coeficiente pelo usuário acarreta a modificação automática dos parâmetros externos de um ou mais blocos conforme estas informações. Veja a figura 14.6.

Observe que a alteração de um ponto de controle no interior de uma fácie não altera a forma da mesma, mas sim a correspondência entre as coordenadas locais U e a coordenada Z. Embora a velocidade V seja sempre a mesma função afim das coordenadas locais U, essa alteração vai modificar a relação entre V e a coordenada Z no interior da fácie.

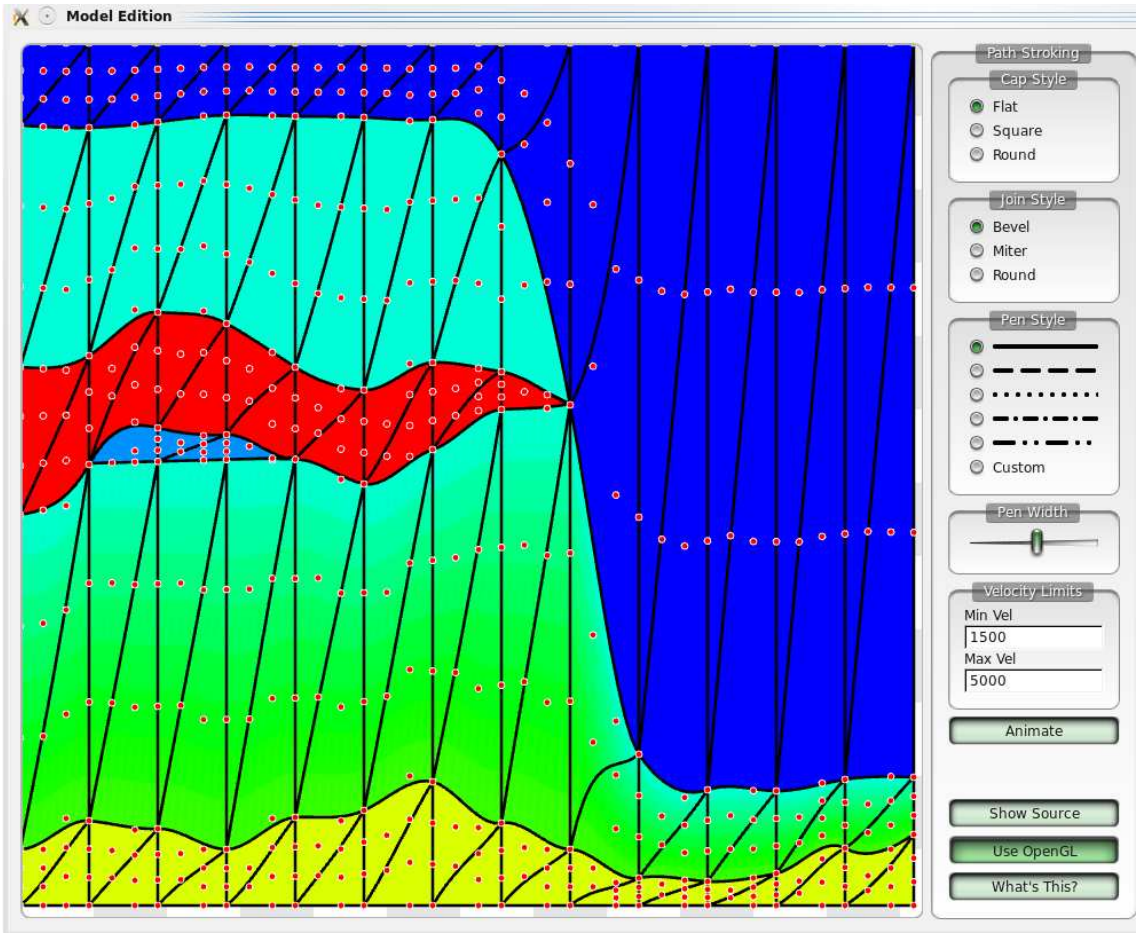


Figura 14.6: Interface do editor de geometria. As coordenadas Z dos pontos vermelhos são os coeficientes do modelo.

Capítulo 15

Simulação sísmica

Uma das ferramentas principais para determinar a estrutura geológica de uma região extensa é a *sísmica de reflexão*, que consiste em aplicar um forte sinal mecânico em um ponto da região de interesse e registrar seus ecos em um conjunto de receptores (geofones em terra ou hidrofones no mar) distribuídos na *região de aquisição*, por exemplo na superfície terrestre, abaixo da superfície ou no fundo do mar, no interior de poços, etc.

Embora o objetivo final seja inverter os dados sísmicos assim obtidos para se conhecer a geologia da sub-superfície, há interesse também no problema direto, a *simulação sísmica*, onde o modelo geofísico é fornecido a priori e a aquisição de dados sísmicos é obtida computacionalmente, resultando em dados sísmicos simulados, conhecidos como *sismogramas sintéticos*.

A simulação sísmica é uma ferramenta poderosa para modelagem, exploração, gerenciamento e monitoramento de reservatórios. Ela ajuda a validar modelos geológicos e interpretar dados sísmicos reais, por exemplo possibilitando a identificação e posicionamento de interfaces e a determinação de propriedades físicas das fácies principais [2].

A simulação sísmica pode ser feita por várias técnicas incluindo solução da equação da onda sobre uma grade regular [27], aproximações baseadas na teoria de raios [25] e de frentes de onda [26]. Dentre estas técnicas de simulação, o traçamento de raios se destaca por sua versatilidade e eficiência computacional, sendo especialmente apropriado para modelos baseados em interfaces ou em malhas tridimensionais.

15.1 Traçado de raios

O traçado de raios se aplica tipicamente à situações onde a excitação sísmica é impulsiva — uma fonte pontual (explosão ou tiro de canhão de ar), localizada em um ponto da região e limitada a um intervalo de tempo muito curto.

15.1.1 Ondas sísmicas

Verifica-se que, em cada instante seguinte ao impulso, a deformação elástica na rocha consiste de uma coleção de *ondas sísmicas*. Cada onda pode ser modelada por uma superfície, a *frente da onda*, que se desloca de maneira contínua através de uma fácie.

Neste trabalho, por simplicidade, nos restringiremos ao traçado de raios em meios *isotrópicos*, nos quais a velocidade de propagação da frente de onda depende unicamente de sua posição, e não da direção de propagação. Nestes meios, dois tipos de onda se propagam de forma independente e com velocidades diferentes. A *onda compressional* ou *onda* **P**, mais rápida, desloca as partículas da rocha na direção da sua propagação. A *onda cisalhante* ou *onda* **S**, mais lenta, desloca as partículas na direção perpendicular à sua propagação.

15.1.2 Equações do raio

Nesse contexto, um *raio* é a trajetória de um ponto da frente de onda que se desloca sempre na direção perpendicular à ela. Esta trajetória pode ser modelada por uma função R dos reais, $\mathbb{R}$, para o $\mathbb{R}^3$, tal que $R(t)$ são as coordenadas do ponto t segundos depois do impulso inicial. Veja a figura 15.1.

Dentro de uma camada, onde as propriedades físicas da rocha variam suavemente, um raio R satisfaz as equações

$$\begin{aligned} \dot{R}(t) &= v(R(t))^2 p(t) \\ \dot{p}(t) &= \frac{1}{v(R(t))}(\vec{\nabla} v)(R(t)). \end{aligned} \tag{15.1}$$

Nestas equações, p é uma função auxiliar de $\mathbb{R}$ para $\mathbb{R}^3$, e $\dot{R}$ e $\dot{p}$ denotam as derivadas de R e de p em relação ao tempo.

A função dada v associa a cada ponto da região de interesse o módulo da velocidade de propagação da onda simulada (**P** ou **S**) naquele ponto. O valor de $p(t)$, chamado de *vagarosidade*, é um vetor perpendicular à frente de onda no instante t e no ponto $R(t)$, cujo módulo é igual a $1/v(R(t))$. A figura 15.1 ilustra estes conceitos.

Observe que, dadas a posição inicial $R(t_0)$ e a vagarosidade inicial $p(t_0)$ de um raio para algum instante $t_0 \geq 0$, as equações (15.1) determinam a posição $R(t)$ no raio para todo $t \geq t_0$. Esta trajetória pode ser calculada numericamente, por exemplo pelo método Runge-Kutta [22].

As equações (15.1) descrevem apenas a posição e a direção do raio. A integração destas equações é chamada de *traçado cinemático do raio*. Porém, cada ponto de uma onda tem também outras propriedades físicas como amplitude, duração e forma do pulso,

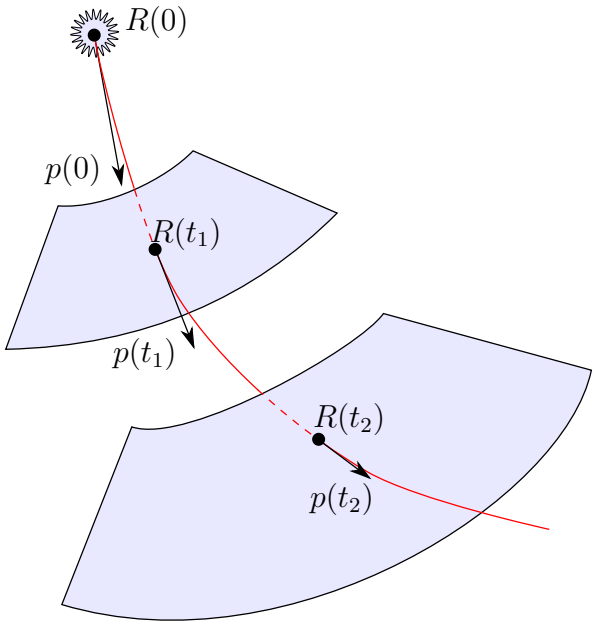


Figura 15.1: Ilustração da propagação de uma onda sísmica resultante de um impulso inicial no instante $t = 0$ e no ponto $R(0)$, mostrando a frente de onda em dois instantes subseqüentes $t = t_1$ e $t = t_2$, e um raio (em vermelho) com direção inicial $p(0)$.

polarização, etc. A evolução destas propriedades ao longo do tempo é descrita por um outro conjunto de equações, cuja interação constitui o *traçado de raios dinâmico*.

15.1.3 Custo da integração em coordenadas cartesianas

A integração das equações do raio (15.1) na forma apresentada é bastante dispendiosa no caso de malhas com elementos curvos, como os usados na nossa biblioteca. Este custo decorre principalmente da necessidade de verificar, a cada passo de interação, se o raio continua no mesmo elemento ou se cruzou sua fronteira. Mais especificamente, suponha que o ponto $R(t)$ do raio está no interior do bloco $\mathtt{B}$, cujo domínio é um simplóide $\mathbb{K}^\delta$, para algum $\delta \in \mathbb{I}_m$ e algum $m \in \mathbb{N}$. Para determinar se o ponto $R(t)$ está dentro do elemento é necessário encontrar o ponto $U(t)$ (do domínio $\mathbb{A}^\delta$) tal que $R(t) = \mathtt{B}.P(U(t))$, onde $\mathtt{B}.P = (\mathtt{B}.P_0, \mathtt{B}.P_1, \mathtt{B}.P_2) = (\mathtt{B}.X, \mathtt{B}.Y, \mathtt{B}.Z)$ são as três componentes de $\mathtt{B}$ que descrevem a geometria do bloco. Uma vez determinado o ponto $U(t)$, podemos verificar se ele está dentro do simplóide $\mathbb{K}^\delta$. A determinação de $U(t)$ a partir de $R(t)$ recai em um sistema de 3 equações polinomiais cuja solução exige métodos iterativos custosos como Newton-Raphson.

O cálculo de $U(t)$ é necessário também quando as propriedades físicas são descritas em termos das coordenadas locais, como no nosso modelo; pois nesse caso precisamos de

$U(t)$ para calcular velocidade de propagação $v(R(t)) = \mathtt{B}.v(U(t))$. Além disso, para obter o gradiente cartesiano $(\nabla v)(R(t))$ precisamos calcular o seu gradiente $(\nabla \mathrm{B}.v)(U(t))$ nas coordenadas locais e traduzir este último para o gradiente em relação às coordenadas X,Y e Z. A alternativa comum de modelar as propriedaes físicas como função das coordenadas cartesianas economizaria esta conversão, mas não eliminaria a necessidade de calcular $U(t)$ para a detecção da fronteira do bloco.

15.1.4 Interação do raio com interfaces

As equações (15.1) são válidas apenas enquanto o raio viaja em um meio onde v varia suavemente com a posição. Quando um raio R atinge uma interface, onde há mudança brusca nas propriedades físicas, ele geralmente se extingue e dá origem a um ou mais novos raios, refletidos e/ou transmitidos. A direção inicial de cada novo raio R' depende da direção final do raio R, da inclinação local da interface, e do contraste de velocidades através da interface (que por sua vez depende dos tipos de onda **S** ou **P**), de acordo com a lei de Snell [25]. Cada novo raio R' é governado pelas equações (15.1) até encontrar outra interface, e assim por diante. Na simulação numérica, costuma-se especificar de antemão o lado da interface (isto é, se o raio é refletido ou transmitido) e o tipo de onda (**S** ou **P**) que deve ser escolhido em cada interface atravessada. Esta seqüência de escolhas é chamada de *assinatura* do raio [25].

Quando o raio atinge a fronteira, o custo de identificar o bloco vizinho é, no máximo, proporcional ao número de vizinhos. A transição do raio, por outro lado, tem custo constante. Uma vez que, em malhas típicas, cada bloco tem um número limitado de vizinhos, o custo do traçado de raios é proporcional ao número total de interfaces atravessadas mais o número total de passos de integração efetivados no interior dos blocos.

15.2 Traçado de raios nas coordenadas locais do bloco

Para evitar a conversão das coordenadas X,Y e Z para coordenadas locais U, nós adaptamos as equações do raio (15.1) de modo a descrever diretamente a evolução de $U(t)$ [10]. Mais precisamente, substituímos o estado do integrador $[R(t), p(t)]$ por $[U(t), p(t)]$, onde $U(t)$ são as coordenadas locais do ponto $R(t)$ em $\mathtt{B}$. Veja a figura 15.2. Note que o vetor vagarosidade $p(t)$ continua sendo representado pelas suas coordenadas Cartesianas físicas. Reescrevemos então o sistema (15.1) como

$$\begin{aligned} \dot{U}(t) &= (W(t))^{-1} \left(\mathtt{B}.v(U(t))\right)^2 p(t) \\ \dot{p}(t) &= (W(t))^{-1} \left(\frac{1}{\mathtt{B}.v(U(t))}\right) G(t) \ , \end{aligned} \tag{15.2}$$

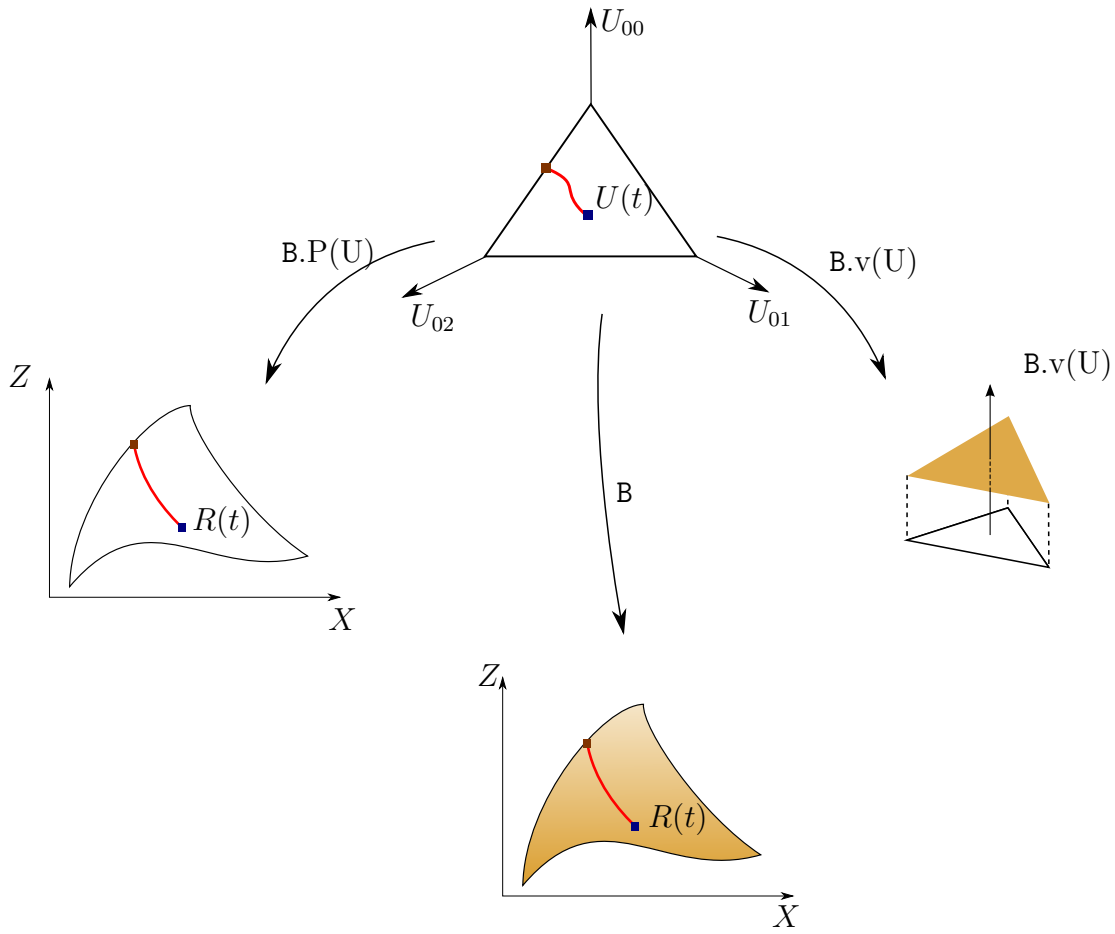


Figura 15.2: Ilustração do traçado de raios nas coordenadas locais de um bloco simploidal bidimensional.

onde $W(t)$ é a matriz jacobiana das funções da forma do bloco B calculada no ponto $U(t)$,

$$W_{ik}(t) = \frac{\partial(\mathtt{B}.P_i)}{\partial U_k}(U(t)). \tag{15.3}$$

e $G(t)$ é o vetor gradiente da função $\mathtt{B}.v$ em relação as coordenadas locais, também calculado no ponto $U(t)$,

$$G_k(t) = \frac{\partial(\mathtt{B}.v)}{\partial U_k}(U(t)) \tag{15.4}$$

Nas equações (15.3) e (15.4), supõe-se que as coordenadas U_{ij} de um ponto U de $\mathbb{A}^\delta$ são renumeradas $U_0, U_1, \cdots, U_{s-1}$, onde $s = |\delta| + m$, em alguma ordem fixa. Observe que cada derivada parcial $\partial(\mathtt{B}.P_i)/\partial U_k$ e $\partial(\mathtt{B}.v)/\partial U_k$, como já mencionado no capítulo 8, é uma função polinomial simploidal, facilmente determinada a partir dos coeficientes de Bézier do bloco B.

Observe que o sistema (15.2) não é válido quando a matriz W é singular. Entretanto, a inexistência de tais singularidades é um requisito natural para um modelo geofísico baseado em células.

15.2.1 Normalização de $U(t)$

Em teoria, se $U(t_0)$ está em $\mathbb{A}^\delta$, isto é a soma de cada linha de $U(t)$ é 1, as equações (15.2) mantém $U(t)$ em $\mathbb{A}^\delta$ para todo $t \geq t_0$. Porém, na prática, erros de arredondamento podem fazer com que $U(t)$ saia de $\mathbb{A}^\delta$. Para corrigir tais desvios é necessário normalizar $U(t)$ após cada passo de integração forçando $\sum_{j=0}^{\delta_i} U_{ij}(t) = 1$.

15.2.2 Custo da integração em coordenadas locais do bloco

A principal vantagem deste método é evitar as iterações de Newton-Raphson para determinar $U(t)$ a partir de $R(t)$. A desvantagem é a necessidade de calcular o jacobiano $W(t)$ e sua inversa, mas estas contas são mais simples do que as iterações de Newton-Raphson.

15.3 Exemplo numérico

Para validar esta nova abordagem, implementamos um módulo na linguagem Java que realiza o traçado de raios nos modelos bidimensionais criados pelo editor TopEdit descrito no capítulo 14. Dados a posição da fonte, a direção inicial de um raio e a sua assinatura, o programa realiza o traçado do raio integrando o sistema (15.2), utilizando o método Runge-Kutta de quarta-ordem [22], até encontrar uma interface. Neste ponto, o programa acompanha um dos novos raios, determinado por uma assinatura previamente fornecida. Adicionalmente, o usuário pode especificar um feixe de raios, fornecendo a abertura em graus do feixe e a separação em graus entre os raios que o compõe.

Para ilustrar o programa, usamos a seguir um modelo geofísico de uma região plana (propagação 2D) com 12km de extensão por 8km de profundidade, mostrado na figura 15.3. Neste teste, a fonte foi localizada na superfície, mais especificamente na posição $X = 4$km e $Z = 0$km. A figura 15.4a ilustra os raios refletidos na base da primeira camada, com direções iniciais variando de $-28°$ a $28°$ com incrementos de $1°$. A figura 15.4b, por sua vez, ilustra os raios refletidos na base da segunda camada, com direções iniciais variando de $-17°$ a $17°$ com mesmo incremento. Em ambas as simulações, usamos integração de RungeKutta com passo de 2ms.

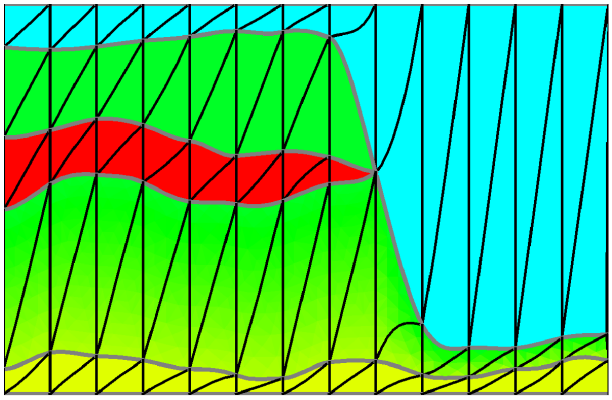
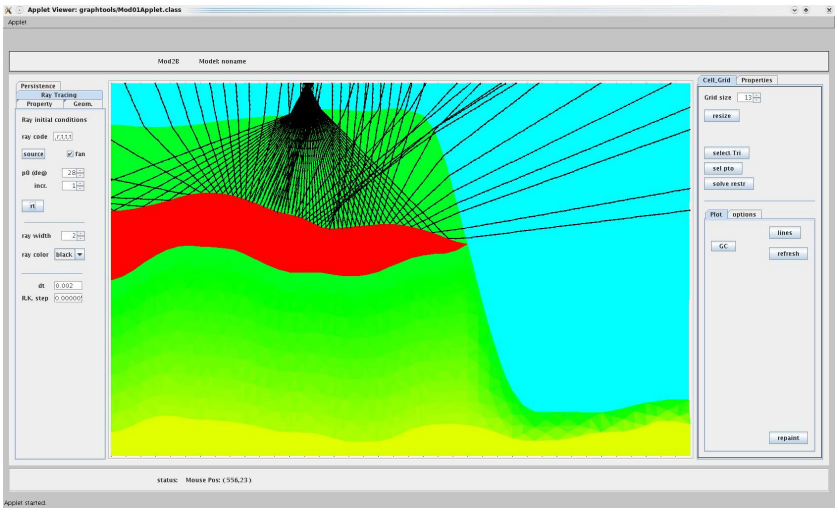
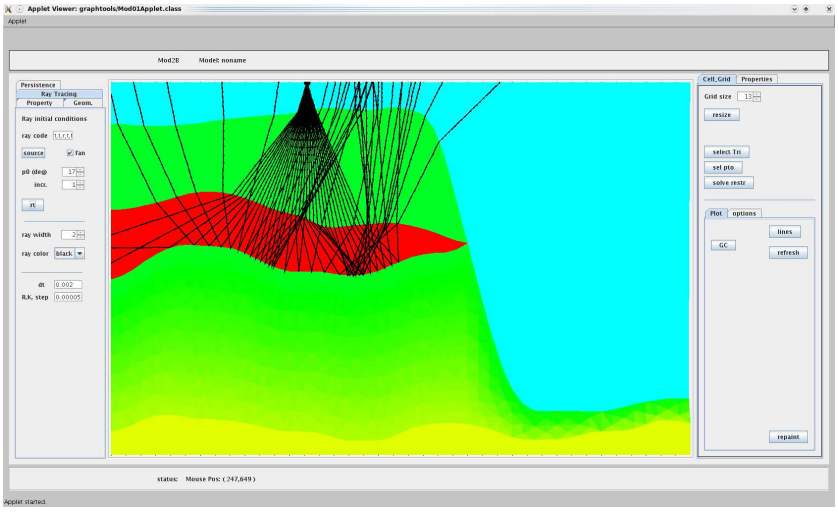


Figura 15.3: Modelo geofísico criado pelo editor TopEdit. As velocidades variam de 1500m/s (ciano) a 5000m/s (vermelho).



(a)



(b)

Figura 15.4: Simulação sísmica por traçamento de raios. Raios refletidos (a) na base da primeira camada e (b) na base da segunda camada.

Capítulo 16

Conclusões

Nesta tese, estudamos a representação de modelos geológicos por malhas de elementos de Bézier simploidais. Esta classe de elementos inclui, como casos particulares, a curva de Bézier, os retalhos triangulares e retangulares de Bézier, e os blocos hexaédricos, tetraédricos e prismáticos de Bézier.

Na primeira parte da tese, revisamos os conceitos de elementos simpliciais e tensoriais e desenvolvemos fórmulas genéricas explícitas para conversão entre estes dois tipos para dimensões e graus arbitrários. Além disso, formalizamos os elementos simploidais de Bézier e apresentamos fórmulas genéricas para elevação de grau, diferenciação e reparametrização por mapeamentos afins constantes. Para este fim, usamos a notação de multi-índices de DeRose [6], estendemos sua notação de hiper-índices para incluir matrizes irregulares, e definimos o novo conceito de ultra-índice. Graças a estes conceitos, conseguimos obter fórmulas sucintas válidas para todos os tipos de blocos e dimensões.

Na segunda parte, aplicamos esta teoria à representação computacional de malhas de elementos simploidais. Descrevemos a biblioteca BezEl, por nós desenvolvida, que permite a representação e manipulação eficiente de malhas de elementos de Bézier simplodais de dimensões e graus arbitrários. A generalidade do número de dimensões destes elementos permite que a malha descreva não apenas a geometria do modelo mas também um conjunto arbitrário de propriedades físicas do seu interior.

Dentre as características originais da BezEl, destacamos o conceito do tipo de bloco e a distinção entre parâmetros internos e externos, que permitem representar de forma econômica malhas regulares e semi-regulares com blocos de graus elevados. Outras contribuições originais desta biblioteca são rotinas genéricas para extração de facetas, reparametrização, elevação de grau e diferenciação. A biblioteca BezEl também proporciona a asserção de restrições lineares arbitrárias sobre os parâmetros dos blocos, que podem ser usadas, por exemplo, para editar a geometria do modelo mantendo a suavidade de interfaces.

Validamos a biblioteca implementando um editor interativo da geometria de um modelo geológico bidimensional (apresentado no capítulo 14).

Outra contribuição original desta tese é uma estratégia para traçamento eficiente de raios em modelos descritos por malhas de elementos de Bézier simploidais (descrita no capítulo 15) . Esta metodologia consiste em efetuar a integração das equações do raio em termos das coordenadas locais de cada bloco em vez das coordenadas X, Y e Z do espaço físico. Desta forma, evita-se a inversão numérica das funções de geometria do bloco. Além disso, a detecção da intersecção do raio com a fronteira do bloco torna-se uma operação trivial.

Referências Bibliográficas

[1] P. Bézier. *Numerical control; mathematics and applications [by] P. Bézier. Appendixes by A. R. Forrest. Translated by A. R. Forrest and Anne F. Pankhurst.* J. Wiley London, New York, 1972.

[2] T.N. Bishop. Tomographic determination of velocity and depth in laterally varying media. *Geophysics*, 50:903–923, 1985.

[3] I Brueckner. Construction of Bézier points of quadrilaterals from those of triangles. *Computer Aided Design*, 1980.

[4] S.A. Coons. Surface patches and b-splines curves. In *in Computer Aided Geometric Design (ed. Barnhill and Riesenfeld)*, pages 1–16. Academic Press, 1974.

[5] C. DeBoor. B–form basics. In *in (Geometric Modeling: Algorithms and New Trends),(ed. Farin)*, pages 131–148. SIAM Publications, 1987.

[6] T.D. DeRose, R.N. Goldman, H. Hagen, and S. Mann. Functional composition algorithms via blossoming. *ACM Trans. Graph.*, 12(2):113–135, 1993.

[7] R.B. Devloo. Pz: An object oriented environment for scientific programming. *Computer Methods in Applied Mechanics and Engineering*, 150:133–153, 1997.

[8] G. Farin. Bézier polynomials over triangles and the construction of piecewise CR polynomials. Technical Report TR91, Brunel University, 1980.

[9] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, third edition, 1992.

[10] L. Freitas, J. Stolfi, and M. Tygel. Fast ray tracing in cellular models. In *Expanded Abstracts, 10th International Congress of the Geophysics Brazillian Society*, 2007.

[11] R Goldman and D. Filip. Conversion from Bézier rectangles to Bézier triangles. *Computer Aided Design*, 1987.

[12] K. Hoffman and R. Kunze. *Linear Algebra*. Prentice Hall, second edition, 1971.

[13] S.-M. Hu. Conversion of a triangular Bézier patch into three rectangular Bézier patches. *Computer Aided Geometric Design*, 13(3):219–226, 1996.

[14] S.-M. Hu. Conversion between triangular and rectangular Bézier patches. *Comput. Aided Geom. Des.*, 18(7):667–671, 2001.

[15] W. Kaplan. *Advanced Calculus*. Addison-Wesley, 1993.

[16] M. Konig. Cell ray tracing for smooth, isotropic media: a new concept based on a generalized analytic solution. *Geophysical Journal International*, 1995.

[17] D. Lasser. Tensor product Bézier surfaces on triangle Bézier surfaces. *Computer Aided Geometric Design*, 19(8):625–643, 2002.

[18] D. Lasser. Triangular subpatches of rectangular Bézier surfaces. *Comput. Math. Appl.*, 55(8):1706–1719, 2008.

[19] A.P. Malheiro and J. Stolfi. Finding minimal bases in arbitrary spline spaces. In *Proceedings of the 22nd Canadian Conference on Computational Geometry (CCCG2010)*, pages 135–138, 2010.

[20] J. L. Mallet, C. Sword, and W. Velten. Computation of smooth second derivatives on irregular triangulated surfaces. In *Expanded Abstracts, 67th SEG Annual Meeting*, pages 1715–1718, 1997.

[21] Z. Meng and N. Bleistein. Wavefront construction (WF) ray tracing in tetrahedral models – application to 3-D traveltime and raypath computations. Technical Report CWP-251, Center for Wave Phenomena, Colorado School of Mines, 1997.

[22] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[23] A. Riskus. Approximation of a cubic Bézier curve by circular arcs and vice versa. *Information Tecnology and Control*, 35(4):371–378, 2006.

[24] W. Trump and H. Prautzsch. Arbitrarily high degree elevation of Bézier representations. *Computer Aided Geometric Design*, 13(5):387 – 398, 1996.

[25] V. Červený. *Seismic Ray Theory*. Cambridge University Press, 2001.

[26] V. Vinje, E. Iversen, H. Gjøystdal, and K. Åstebøl. 3D ray modeling by wavefront construction in open models. *Geophysics*, 64:1912–1919, 1999.

[27] J. Virieux. P-sv wave propagation in heterogeneous media: Velociy-stress finite-difference method. *Geophysics*, 51, 1986.

[28] L. Wang. *Estimation of Multi-valued Green's function by Dynamic Ray tracing and True Amplitude Kirchoff Inversion in 3D-Heterogeneous media*. PhD thesis, Center of Wave Phenomenon, 2000.