Instituto Nacional de Matemática Pura e Aplicada

Curvas B-Spline e de Bézier Aplicadas a CADG

Autor: **Dyego Soares de Araújo**

Orientador: **José María Espinar Garcia**

Rio de Janeiro
Fevereiro de 2016

Dedico esse trabalho a todos aqueles que alteraram os rumos da minha vida

Agradecimentos

Agradeço inicialmente a minha Família. Agradeço aos meus Pais, por todo o apoio, todo o carinho e todo o amor que eles sempre tiveram para comigo. Agradeço ao meu irmão por ter sempre estado ai quando eu necessitei.

Agradeço a todos os professores que passaram por minha vida, tanto os do IMPA quanto os de minha antiga universidade, e de minha escola. Agradeço em especial a meu orientador José Espinar, e ao professor Luis Velho pela ajuda e orientação com esse trabalho.

Agradeço também aos meus amigos aqui no IMPA, que me ensinaram um universo de coisas que vai muito além da matemática. Agradeço em especial a Juan Sebastian, Diana Carolina, Brian Trianna, entre outros, por toda a amizade que me ofereceram.

Por fim, deixo um agradecimento especial para aquela que fez da minha estadia no Rio de Janeiro uma história feliz. Agradecerei sempre por você ter aparecido em minha vida, e te agradeço ainda mais porque sem você eu não teria tido forças para seguir.

Agradeço, por fim, ao CNPQ pelo apoio financeiro e ao IMPA enquanto instituição por tudo que me ofereceu.

Resumo

RESUMO EM PORTUGUES

Palavras Chave: CAGD, Bezier Curves, B-Spline, de Casteljau

Abstract

RESUMO EM INGLÊS

Key words: CAGD, Bezier Curves, B-Spline, de Casteljau

Sumário

Introdução

A modelagem de formas geométricas sempre foi um problema difícil. Diversos são os exemplos de aplicações que necessitam de ferramentas de design de formas geométricas. O Design Gráfico Auxiliado por Computador (CAGD) é uma área da Ciência da Computação que surgiu para dar uma resposta a esse problema.

O CAGD lida com descrições matemáticas de objetos geométricos, para uso em computadores. Ele se utiliza de várias áreas interessantes da matemática, como a Geometria Diferencial, a Geometria Algébrica, Análise Numérica e Teoria das Aproximações, entre outras. Se desenvolveu ativamente desde a década de 50, impulsionado principalmente pela industria Automobilística e Aeronáutica, e continua com pesquisa ativa até os dias de hoje.

Essa dissertação tratará a respeito das curvas de Bézier e as curvas B-Spline.

Embora levem o nome de Bézier, essas curvas foram inicialmente descobertas por Paul de Casteljau, quando trabalhava para a Citroën. Sua idéia consistia em trabalhar com "courbes et pôles", ou "Polígono de Controle", para a definição das curvas. Ele também descobriu o algorítmo que leva seu nome, e diversas propriedades interessantes dessas curvas. Entretanto, por questões de confidencialidade, ele foi impedido de publicar seu trabalho. Este foi descoberto apenas na década de 70, por W. Boehm, que lhe deu o devido crédito e popularizou o nome 'Algorítmo de de Casteljau'. Após se aposentar da Citroën, de Castejau passou a publicar diversos artigos na área e em 2012 recebeu o Prêmio Bézier por suas contribuições no desenvolvimento do ramo.

Piérre Bézier, enquanto trabalhava para a Renault, também reconheceu a necessidade de se encontrar uma representação matemática adequada para modelar peças automobilisticas e carros, utilizando o computador. Nessa época, estava entrando em voga o Controle Numérico. Essa técnica de manufatura utilizava intruções numéricas transmitidas pelo computador para controlar as máquinas de produção. Bézier reconheceu a necessidade de uma reformulação na maneira com que se fazia design. Embora partindo de uma idéia diferente da de de Casteljau, Bézier chegou em uma formulação equivalente. Ele e sua equipe desenvolveram um sistema CAD chamado UNISURF, utilizado na Renault. Esse sistema contava com curvas e superfícies de Bézier como estruturas primitivas, e influenciou alguns sistemas posteriores.

As funções Spline aparecem pela primeira vez com esse nome em um paper de Schoenberg 1946[14]. Boa parte das propriedades que tornam as Splines atrativas para o design de curvas foram descobertas por Carl de Boor, enquanto trabalhava para a General Motors. Lá ele descobriu o Algorítmo de de Boor, que permite a avaliação numerica dessas curvas de maneira estável. Mais tarde, de Boor se tornaria um dos principais pesquisadores a propor o uso de B-Splines em teoria da Aproximação. Seu livro [5] até hoje é descrito como um dos melhores no assunto.

O uso de B-Splines Paramétricas como ferramenta de design foi proposto por Richard Riesenfeld em sua tese de Doutorado[13], ao observar que o algorítmo de de Boor era uma generalização natural para o algorítmo de de Casteljau. Isso implicava que um sistema que contivesse B-Splines naturalmente continha as curvas de Bézier. Diversos artigos foram publicados desde então, simplificando o tratamento dessas curvas, entre eles os algorítmos de inserção de nós por

W. Boehm[2], o algorítmo de Oslo por Riesenfeld, Cohen e Lyche[3], e o princípio Blossoming, descoberto de maneira independente por Lyle Ramshaw[12] e de Casteljau[6].

As curvas B-Spline admitem a generalização para NURBS (Non-Uniform Rational B-Splines). Essa generalização é capaz de representar não só as curvas B-Spline, mas também as curvas cônicas. Desde os primórdios do CAGD, o uso de cônicas sempre foi muito difundido. Permitir tratar tanto cônicas como curvas B-Spline e de Bézier de maneira unificada conferiu às NURBS o status de curva padrão na maioria dos sistemas CAD/CAM.

Um relato histórico completo do desenvolvimento do CADG, em especial das Curvas e Superfícies se encontra detalhado em [9]. Para relatos mais pessoais, temos o divertido relato pessoal de Paul de Casteljau a respeito de seu tempo na Citroën[7] e o artigo de Piérre Bézier a respeito de sua visão pessoal do campo do Design Geométrico [1]

Essa dissertação visa ser um texto introdutório no assunto. Ela foi baseada primariamente nos livros introdutórios no assunto [4], [8] e [10].

Capítulo 1

Preliminares

1.1 Representação Computacional de Curvas

O objetivo central do CAGD é a modelagem de formas geométricas em termos que possam ser compreendidos, armazenados e manipulados por um computador digital. Para compreender um pouco melhor as ideias existentes por tras do CADG, consideremos o problema do design de curvas sob a perspectiva do designer.

Inicialmente, um designer tem uma idéia abstrata do traço de uma curva. Em seguida, essa ideia deve ser transmitida em forma de dados para o computador. Para isso, o designer deverá fornecer alguma informação codificada ao computador, que, ao ser processada, fornecerá a curva. Essa curva será armazenada no computador. Em seguida, o designer pode solicitar uma visualização dessa curva, para saber se ela corresponde a sua ideia original. Em caso afirmativo, segue-se para o design das demais curvas do projeto. Em caso negativo, é interessante que ele tenha disponíveis ferramentas de edição da curva, e que seja capaz de obter a forma que deseja em algumas iterações.

Duas escolhas principais devem ser feitas para guiar esse processo:

- Como codificar a informação que será alimentada ao computador?
- Como codificar a curva para armazená-la no computador?

Várias respostas são possíveis para essas perguntas. Examinemos algumas delas:

1.1.1 Codificação de Entrada

Para alimentar a entrada do computador, possuímos basicamente duas técnicas disponíveis: A amostragem de traço e a inserção de pontos isolados.

Amostragem de traço

A amostragem de traço consistiria na leitura direta do traço de uma curva. Essa leitura deve ser feita por algum dispositivo de captura, como um mouse, um trackpad, um tablet ou mesmo um scanner. Como o computador não poderia lidar com todos os não enumeráveis pontos da curva contínua, algum processo de discretização deve ser conduzido, resultando em um subconjunto discreto na curva. Esse conjunto pode, em seguida, passar por algum processamento a fim de ser transformado em uma curva. A ferramenta Pencil, no Microsoft Paint® exemplifica esse mecanismo de entrada.

Esse método apresenta vantagens e desvantagens. A vantagem principal está na maneira intuitiva com a qual a curva é transmitida. Em geral, o designer está acostumado a fazer desenhos, de modo que essa parece a maneira mais natural.

Existem no entanto algumas desvantagens com esse tipo de entrada. Note que o conjunto de dados extraidos é um conjunto discreto. Para poder aplicar ferramentas de análise clássica, de modo a extrair informações geométricas da curva seria necessário algum tipo de processamento Essa não é uma restrição muito grave ao design, dado o poder computacional dos computadores modernos.

Um problema mais grave, inerente ao desenho a mão livre, esta na falta precisão. A correta transmissão de uma curva para o computador fica então sujeita a habilidade do designer e qualidade do hardware utilizado. Deste modo, este tipo de entrada apresenta complicações com o traço de curvas que definem objetos que requerem algum tipo de precisão, como plantas arquitetônicas ou especificações técnicas de um produto. O problema se agrava se o mesmo programa for utilizado para o design de produtos e para o controle de máquinas de fabricação. Deste modo, em aplicações cujo foco é a precisão, a amostragem de traço pode não ser a maneira mais adequada. Além disso, embora essa abordagem seja bastante útil para o esboço de curvas planas, apresenta algumas complicações no design de curvas 3-D.

As ferramentas matemáticas necessárias pra transformar um traço amostrado em uma curva geométrica fazem parte do ramo conhecido como Teoria das Aproximações. Originalmente, as curvas Spline surgiram no contexto dessa Teoria.

Inserção de Pontos Isolados

A inserção de pontos isolados consiste em alimentar o sistema com instruções, pontos individuais, e ocasionalmente valores escalares. O traço da curva é então definido considerando-se as instruções e relações entre esses pontos. Por exemplo, poderia-se utilizar uma instrução *RETA*, que requisite do usuário dois pontos e defina a reta que passa por eles. A maioria dos sistemas CAD, tais como o Autodesk AutoCAD® utilizam esse método de entrada.

Uma vantagem esse tipo de codificação está na interatividade. Visto que as curvas em geral são definidas por pontos, é possível efetuar um ajuste iterativo nos pontos até se obter a curva mais próxima da imaginada.

Entretanto, a maior força desse modo de inserção está na precisão. Todos os pontos envolvidos podem ser inseridos com a precisão que seja necessária, visto que estão codificados como números em um sistema cartesiano.

Um sistema simples poderia dispor apenas de retas, circunferências e curvas cônicas. Entretanto, para aplicações mais elaboradas, apenas essas curvas se mostram insuficientes. Se faz necessário dispor de curvas menos restritas, que permitam traços mais livres. Tais curvas devem ser complexas o bastante para permitir maior liberdade no traço, mas devem ser codificadas no sistema de maneira intuitiva o bastante para que o designer não tenha necessidade de conhecer matemática avançada para poder utiliza-las. Historicamente, as curvas de Bézier foram desenvolvidas com esse objetivo em mente.

1.1.2 Codificação Interna

As imagens digitais podem ser armazenadas no computador de duas formas distintas. Imagens BitMap ou Gráficos Vetoriais. As representações BitMap consistem em matrizes de amostragem da informação de cor da imagem. Embora essa maneira seja a mais popular de se armazenar uma imagem, para nossos propósitos ela se mostra bastante inadequada. Isso se dá porque há

uma grande dificuldade em se recuperar as informações geométricas das curvas contidas na imagem. Além disso, uma vez armazenada, essa representação torna bastante difícil subsequentes interações com a curva.

A outra representação possível é por meio de Gráficos Vetoriais. Nessa representação, ao invés de amostrar pontos da imagem serão armazenadas representações dos objetos geométricos em si. Nessa representação, podemos traçar objetos geométricos ideais, como linhas e superfícies sem espessura. Estamos interessados em compreender esse tipo de representação. Note que, uma imagem Vetorial é um pouco mais abstrata que uma imagem BitMap. Em geral, para a visualização dessas imagens é necessário um certo processamento para se obter um objeto visualizável. Esse processo é conhecido como Renderização.

Observe que, mesmo escolhida a representação da imagem por meio de gráficos vetoriais, fica a questão a respeito de como representar a curva. Existem diversas ferramentas matemáticas capazes de representar a curva. Em CAGD essas representações usualmente se enquadram em duas categorias: a forma implícita e a forma paramétrica.

Forma Implícita

Definimos como $\mathbb{A}$ o espaço ambiente que vamos trabalhar. Em geral, $A = \mathbb{R}^2$ ou $\mathbb{R}^3$. Uma curva será definida implicitamente como o conjunto de raízes de uma função $f : \mathbb{A} \to \mathbb{R}$, no caso bidimensional ou $f : \mathbb{A} \to \mathbb{R}^2$ no caso tridimensional. Em geral, estaremos interessados em funções f do tipo polinomiais. O ramo da matemática associado ao problema de estudar as propriedades geométricas das raízes de funções polinomiais é a Geometria Algébrica.

Essa representação apresenta uma utilidade clara: Dado um ponto $P \in \mathbb{A}$ é tarefa simples saber se esse é um ponto da curva ou não. Basta avaliar $f(P)$ e verificar se o valor é nulo. Algumas operações de conjunto como união e interseção também possuem representação clara quando os objetos se encontram representados na forma implícita.

Apesar disso, essa representação possui a séria desvantagem de qu obter pontos do objeto geométrico representado não é tarefa trivial. Frequentemente faz-se necessário o cálculo de raízes polinomiais e obtenção de soluções de sistemas não lineares.

Outro problema com a representação implícita está na dificuldade de extrair pedaços da curva. Por exemplo, a forma implícita de uma circunferência centrada na origem de raio 1 seria dada por $f(x, y) = x^2 + y^2 - 1$. Entretanto, não está claro como extrair dessa representação uma representação para um arco de circunferência.

Forma Paramétrica

Nesta seção, I, J denotarão intervalos de $\mathbb{R}$. Uma curva é definida parametricamente como uma função $f : I \to \mathbb{A}$. Os pontos da curva são definidos como a imagem $f(I) \subset \mathbb{A}$. Isto é, um ponto $P \in \mathbb{A}$ se e somente se existe $t \in I$ tal que $P = f(t)$. Por razões computacionais, o espaço de funções utilizado costuma ser o espaço das Funções Polinomiais, ou o espaço das Funções Racionais. Ocasionalmente, o espaço dos Polinômios Trigonométricos e Exponenciais também pode ser utilizado.

Essa forma de representação de objetos apresenta duas vantagens claras sobre a forma Implícita. Primeiramente, é tarefa bastante simples obter pontos das curvas representadas, bastando calcular $f(t)$ para valores $t \in I$. Portanto, objetos representados nessa forma possuem uma clara vantagem no processo de renderização. A representação paramétrica também permite a facil extração de pedaços de um objeto representado. Por exemplo, $f : [-1, 1] \to \mathbb{A}$ dada por $f(t) = (t, t^2)$ representa um trecho de parábola, podemos extrair metade da parábola restringindo f a $[0, 1]$, conforme a figura abaixo:

Por essas razões, a forma paramétrica ainda hoje é a forma mais popular de representação de objetos em CAGD. Boa parte da pesquisa inicial foi orientada tendo essa forma de representação em mente. Entretanto, existem diversas aplicações para a forma implícita. O leitor interessado pode consultar [11].

1.2 Formalização do Problema

Nesse documento, estaremos interessados em estudar a geração de curvas paramétricas obtidas a partir de um número finito de pontos. Idealmente, deseja-se que essas curvas possuam flexibilidade o bastante para representar qualquer traço que um designer possa necessitar. Deseja-se também uma relação intuitiva entre as informações solicitadas do usuário e a curva final obtida.

Para melhor orientar a discussão, faremos a seguinte definição

Definição 1.1 (Algoritmo de Geração de Curvas)**.** Definimos Algoritmo de Geração de Curvas (AGC) a $n+1$ pontos como uma função

$$\mathbf{Curva}_n : \mathbb{A}^{n+1} \times \mathbb{P} \to \mathcal{C}(\mathbb{A})$$

onde $\mathbb{A}^{n+1}$ é o espaço de sequencias finitas de $n+1$ pontos de $\mathbb{A}$, $\mathbb{P} \subset \mathbb{R}^m$ representa o espaço de parâmetros auxiliares requisitados do usuário e $\mathcal{C}(\mathbb{A})$ é o espaço das funções paramétricas $f : I \to \mathbb{A}$ onde $I = [a, b]$ é algum intervalo da reta. O motivo para utilizarmos $\mathbf{Curva}_n$ para $n+1$ pontos é mera conveniência notacional.

Visto da perspectiva computacional, um AGC é uma função da forma:

Algorithm 1: Algorítmo de Geração de Curvas

1 function $\mathbf{Curva}_n$ $(\mathbf{P}, \mathbf{W}, t)$;
Input : $\mathbf{P}$ = Vetor contendo $n+1$ Pontos
$\mathbf{W}$ = Vetor de Parâmetros
t = ponto pertencente ao intervalo
Output: $p(t) \in \mathbb{A}$

Essa dissertação visa estudar AGCs, bem como algumas de suas propriedades.

1.3 Interpolação Polinomial

Para ilustrar a análise que será feita nos capítulos seguintes, nesta seção será proposto um AGC baseado na teoria das interpolações polinomiais. Isso servirá para definir algumas propriedades que podem ser satisfeitas, bem como definir um ponto de comparação para os demais AGCs.

A interpolação polinomial pode ser vista como a solução para o seguinte problema:

Problema 1 (Problema da Interpolação Polinomial para Curvas)**.** Dado um conjunto de pares $\{(P_i, t_i)\}_{i=0}^{n}$, onde $P_i \in \mathbb{A}$ e $t_i \in \mathbb{R}$, encontrar uma curva polinomial $p : [t_0, t_n] \to \mathbb{A}$ de grau n que satisfaça $p(t_i) = P_i$.

Nos termos definidos acima, desejamos obter uma família de AGCs (que denotaremos por $\{\mathbf{InterP}_n\}_{n=0}^{\infty}$) tal que $\mathbf{InterP}_n(\{P_i\}_{i=0}^{n}, \{t_i\}_{i=0}^{n}) = f : [t_0, t_n] \to \mathbb{A}$ satisfaz a propriedade $f(t_i) = P_i$.

Esse é um problema clássico de Análise Numérica. Entretanto, vamos explorar a sua solução seguindo uma linha diferente da que é usualmente exposta nos livros textos no tema. O motivo para isso é exemplificar a maneira como serão definidas as curvas nos capítulos seguintes. Para o leitor interessado, vale a pena consultar o capítulo 7 de [8].

1.3.1 Algorítmo de Aitken

Para obter uma solução geral do problema a n pontos, pode ser util entender o problema para valores pequenos de n. Considere o exemplo abaixo:

Exemplo 1 (AGC por Interpolação Polinomial para $n = 2$)**.** Caso fossem fornecidos apenas 2 pontos $\{P_1, P_2\}$ e dois valores $\{t_1, t_2\}$, a solução óbvia é considerar a reta $r(t)$ que satisfaz $r(t_1) = P_1$, $r(t_2) = P_2$. Então, temos:

$$\mathbf{InterP}_1(\{P_1, P_2\}, \{t_1, t_2\}) = r : [t_1, t_2] \to \mathbb{A}$$

onde

$$r(t) = \frac{t - t_2}{t_1 - t_2} P_1 + \frac{t - t_1}{t_2 - t_1} P_2$$

[Imagem]

A ideia para solucionar esse problema será generalizar esse método de maneira recursiva para obter uma curva polinomial que atenda os requisitos. Buscamos então um algoritmo construtivo capaz de relacionar os pontos e os parâmetros de maneira geométrica. Novamente, consideremos uma situação-exemplo antes de partir para uma solução geral:

Exemplo 2 (AGC por Interpolação Polinomial para $n = 3$)**.** Buscamos então definir:

$$\mathbf{InterP}_2(\{P_0, P_1, P_2\}, \{t_0, t_1, t_2\})$$

.

Como já sabemos resolver o problema para $n = 2$, consideremos $f_0 = \mathbf{InterP}_1(\{P_0, P_1\}, \{t_0, t_1\})$ e $f_1 = \mathbf{InterP}_1(\{P_1, P_2\}, \{t_1, t_2\})$.

A ideia é utilizar essas duas funções para construir uma função que interpole os 3 pontos. Considere a função:

$$f(t) = \frac{t - t_2}{t_0 - t_2} f_0(t) + \frac{t - t_0}{t_2 - t_0} f_1(t)$$

Note que:

$$\begin{array}{rcccl} f(t_0) & = & f_0(t_0) & = & P_0 \\ f(t_2) & = & f_1(t_2) & = & P_2 \\ f(t_1) & = & \frac{t_1 - t_2}{t_0 - t_2} f_0(t_1) + \frac{t_1 - t_0}{t_2 - t_0} f_1(t_1) & = & P_1 \end{array}$$

[imagem]

Observe que a curva resultante é, de fato, um polinômio de grau 2. Isso se dá porque f_0, f_1 são curvas de grau 1, e o procedimento realizado apenas aumenta o grau da curva em 1.

Para generalizar para um número maior de pontos, procederemos com uma definição indutiva. Suponha indutivamente que definimos o AGC $\mathbf{InterP}_n$. Para definir $\mathbf{InterP}_{n+1}$ fazemos:

$$\mathbf{InterP}_{n+1}(\{P_i\}_{i=0}^{n+1}, \{t_i\}_{i=0}^{n+1}) = f : [t_0, t_n + 1] \to \mathbb{A}$$

onde

$$f(t) = \frac{t - t_{n+1}}{t_0 - t_{n+1}} f_0(t) + \frac{t - t_0}{t_{n+1} - t_0} f_1(t)$$

e

$$f_0 = \mathbf{InterP}_n(\{P_i\}_{i=0}^n, \{t_i\}_{i=0}^n), \quad f_1 = \mathbf{InterP}_n(\{P_i\}_{i=1}^{n+1}, \{t_i\}_{i=1}^{n+1})$$

Proposição 1.1. *A curva f é uma interpolação polinomial para os pares $\{P_i, t_i\}_{i=0}^n$ de grau $n+1$*

Demonstração. Observe que, como f_0, f_1 tem por hipótese de indução grau n, segue que f tem grau $n+1$.

Para provar que se trata de uma interpolação, utilizamos o fato de que f_0 e f_1 também são interpolações. Portanto, se $i \neq 0, n$ temos

$$\begin{aligned} f(t_i) &= \tfrac{t_i - t_{n+1}}{t_0 - t_{n+1}} f_0(t_i) + \tfrac{t_i - t_0}{t_{n+1} - t_0} f_1(t_i) \\ &= \tfrac{t_i - t_{n+1}}{t_0 - t_{n+1}} P_i + \tfrac{t_i - t_0}{t_{n+1} - t_0} P_i \\ &= P_i \end{aligned}$$

Para $i = 0$ temos

$$\begin{aligned} f(t_0) &= \tfrac{t_0 - t_{n+1}}{t_0 - t_{n+1}} f_0(t_0) + \tfrac{t_0 - t_0}{t_{n+1} - t_0} f_1(t_0) \\ &= P_0 \end{aligned}$$

Um argumento análogo mostra que $f(t_{n+1}) = P_{n+1}$

□

Apresentamos o pseudocódigo para uma implementação do algoritmo de Aitken.

Algorithm 2: Algorítmo de Aitken para Interpolação Polinomial

```
1 function InterP (P, T, t);
  Input  : P = Vetor contendo n + 1 Pontos
           T = Vetor contendo n + 1 valores reais
           t = ponto pertencente ao intervalo T[0], T[n]
  Output: p_t ∈ 𝔸
2 if tamanho(P) = 2 then
3 |  return p_t = (t − T[1])/(T[0] − T[1]) ∗ P[0] + (t − T[0])/(T[1] − T[0]) ∗ P[1];
4 else
5 |  V_0 = InterP(P[1 : end], T[1 : end], t);
6 |  V_1 = InterP(P[0 : end − 1], T[0 : end − 1], t);
7 |  return p_t = (t − T[end])/(T[0] − T[end]) ∗ V_0 + (t − T[0])/(T[end] − T[0]) ∗ V_1;
8 end
```

1.3.2 Método de Lagrange

Agora que dispomos de um procedimento construtivo, desejamos obter uma fórmula fechada para a curva. Necessitamos de uma fórmula para poder provar algumas de suas propriedades, que não são tão óbvias apenas com o algoritmo.

Para calcular essa forma fechada, vamos tentar buscar uma solução direta para o problema. Para isso, vamos buscar no espaço de Polinômios de grau n, $\mathcal{P}^n$, uma base de polinômios que dependa apenas de $\{t_i\}_{i=0}^n$, digamos $\{L_i^n\}_{i=0}^n$ tal que a solução do problema 1 tenha a forma:

$$f(t) = \sum_{i=0}^{n} P_i L_i^n(t)$$

Observe que uma interpretação possível para essa fórmula é a seguinte: Fixado $t \in I$, podemos interpretar $L_i^n(t)$ como a contribuição do ponto P_i no valor final de $f(t)$. Portanto, essas serão chamadas Funções Misturadoras (Blending Functions em ingles) para o problema. Em geral, estaremos interessados em encontrar as Funções Misturadoras associadas aos modelos de curvas que propormos.

A propriedade fundamental que esses polinômios devem satisfazer é $L_i(t_j) = \delta_{ij}$, onde δ_{ij} denota o delta de Kronecker. Sabendo que se tratam de polinômios de grau n, pelo Teorema Fundamental da Álgebra, devem haver no máximo n raízes. Essas n raízes são dadas por $\{t_j \, j \neq i\}$. Após uma normalização para obter $L_i(t_i) = 1$, obtemos os polinômios:

$$L_i(t) = \prod_{j \neq i}^{n} \left(\frac{t - t_j}{t_i - t_j} \right)$$

Esses polinômios são conhecidos como funções misturadoras de Lagrande (Lagrange's Blending Functions). Vejamos, por exemplo as funções associadas ao conjunto $\{0, 1, 2, 3\}$

[Imagem]

Devemos agora provar que $f(t) = \sum_{i=0}^{n} P_i L_i^n(t)$ de fato coincide com a função fornecida por $\mathbf{InterP}_n(\{P_i\}_{i=0}^n, \{t_i\}_{i=0}^n)$. Para isso, necessitaremos do seguinte lema:

Lema 1.2. *As funções $\{L_i^n(t)\}_{i=0}^n$ formam uma base para o espaço de polinômios $\mathcal{P}^n$.*

Demonstração. É fato conhecido que o espaço $\mathcal{P}^n$ tem dimensão $n+1$. Como existem $n+1$ funções de Lagrange de grau n, resta verificar que estas são Linearmente Independentes. Para isso, suponha que $\sum_{i=0}^{n} c_i L_i^n(t) = 0$ para algum conjunto de escalares $\{c_i\}_{i=0}^n$. Aplicando a função em t_j, obtemos

$$\sum_{i=0}^{n} c_i L_i^n(t_j) = c_j = 0$$

Segue que $c_j = 0 \, \forall j$ de onde segue que essas funções são linearmente independentes.

□

Com o auxílio desse lema, provamos:

Proposição 1.3. $\mathbf{InterP}_n(\{P_i\}_{i=0}^n, \{t_i\}_{i=0}^n)(t) = \sum_{i=0}^{n} P_i L_i^n(t)$

Demonstração. Seja $f = \mathbf{InterP}_n(\{P_i\}_{i=0}^n, \{t_i\}_{i=0}^n)$. Como as funções de Lagrange são uma base de $\mathcal{P}^n$ podemos escrever esse polinômio como

$$f(t) = \sum_{i=0}^{n} C_i L_i^n(t)$$

Sabendo que o polinômio de Atkin passa por P_j no valor t_j, obtemos:

$$P_j = f(t_j) = \sum_{i=0}^{n} C_i L_i^n(t_j) = C_j$$

Assim, finalmente, encontramos a forma fechada $f(t) = \sum_{i=0}^{n} P_i L_i^n(t)$ Provando a proposição.

□

Como podemos aplicar essa demonstração a qualquer polinômio que satisfaça as condições de interpolação, obtemos

Corolário 1.4 (da Demonstração)**.** *A solução para o problema de Interpolação Polinomial é única.*

1.3.3 Propriedades da Interpolação Polinomial

Nessa seção verificaremos algumas das possíveis propriedades que podem ser satisfeitas pela interpolação polinomial. Utilizaremos essa seção para definir algumas das propriedades que podem ou não ser satisfeita por AGCs. Isso será útil para estabelescer critérios de comparação.

Invariância por Transformações Afins

As transformações afins são a família de transformações de uso mais frequente em CAGD. Para compreender essas família de transformações, definimos primeiramente o conceito de combinação afim, ou baricêntro.

Definição 1.2 (Combinação Afim)**.** Dados $\mathtt{P} = \{P_i\}_{i=1}^k$, uma combinação afim é uma combinação linear $P = \sum_{i=1}^k \lambda_i P_i$ que satisfaz a restrição $\sum_{i=1}^k \lambda_i = 1$. O conjunto de todas as combinações afins de $\mathtt{P}$ é chamado fecho afim, e denotado por $\mathfrak{F}(\mathtt{P})$

Com o uso de combinações afins, possuimos uma maneira natural para definir retas e planos que não passam pela origem. Por exemplo, uma reta seria definida por $\mathfrak{F}(\{P_1, P_2\})$.

Análogo ao conceito de Transformação Linear (aquela que preserva combinações lineares), definimos o conceito de Transformação Afim.

Definição 1.3. Uma transformação $T : \mathbb{A} \to \mathbb{A}$ é dita ser afim se, para toda combinação afim $P = \sum_{i=1}^k \lambda_i P_i$ tem-se

$$T\left(\sum_{i=1}^{k} \lambda_i P_i\right) = \sum_{i=1}^{k} \lambda_i T(P_i)$$

Observe que toda transformação linear é uma transformação afim. Entretanto, existem transformações afins que não são lineares, como por exemplo a translação. De fato, é possível provar que toda transformação afim pode ser decomposta em uma transformação linear e uma translação.

Quanto a nossas curvas, seria interessante exigir delas invariância com respeito a transformações afins. Definimos:

Propriedade 1 (Invariancia por Transformações Afins)**.** *Um AGC* $\mathbf{Curva}_n$ *com parâmetros* $\boldsymbol{Pr}$ *é dito possuir a propriedade de invariância por transformações afins se, dada qualquer transformação afim* $T : \mathbb{A} \to \mathbb{A}$ *tem-se* $T(\mathbf{Curva}_n(\{P_i\}_{i=0}^n, \boldsymbol{Pr})) = \mathbf{Curva}(\{T(P_i)\}_{i=0}^n, \boldsymbol{Pr})$.

Em outras palavras, é equivalente calcular a curva para em seguida efetuar a transformação afim ou calcular a transformação afim sobre os pontos e apenas então calcular a curva.

Para provar a invariância afim, podemos observar o AGC sob a perspectiva do algoritmo de Aitken. Todos os passos intermediários desse algoritmo consistem em combinações afins. Como T é invariante por combinações afins, segue que o AGC é também invariante por essas transformações. No entanto, provaremos essa propriedade de outra maneira. Para isso utilizaremos o seguinte lema:

Lema 1.5. *As funções de Lagrange de grau n somam 1. Isto é,* $\sum_{i=0}^{n} L_i^n(t) = 1 \, \forall t$

Demonstração. Considere $f = \mathbf{InterP}_n(\{P\}_{i=0}^n, \{t_i\}_{i=0}^n)$ o problema de interpolação com todos os pontos iguais. Pelo que vimos acima, $f(t) = \sum_{i=0}^{n} P L_i^n(t)$. Entretanto, $g(t) = P$ constante também soluciona o problema. Pela unicidade da solução, obtemos:

$$\left(\sum_{i=0}^{n} L_i^n(t) \right) P = P$$

.

Logo $\sum_{i=0}^{n} L_i^n(t) = 1$. □

Observe que, deste fato, descobrimos que pontos da curva de Interpolação são combinações afins dos pontos interpolados. Portanto, como as transformações afins preservam combinações afins, segue que $T\left(\sum_{i=0}^{n} P_i L_i^n(t)\right) = \sum_{i=0}^{n} T(P_i) L_i^n(t) \, \forall t$. Logo a interpolação polinomial possui invariância afim.

Propriedade do Fecho Convexo

Iniciemos definindo combinações convexas.

Definição 1.4 (Combinação Convexa)**.** Uma combinação afim $P = \sum_{i=1}^{k} \lambda_i P_i$ é dita ser uma combinação convexa se, além de satisfazer $\sum_{i=1}^{k} \lambda_i = 1$, satisfaz também $\lambda_i > 0 \forall i$. O conjunto de todas as combinações convexas de $\mathtt{P} = \{P\}_{i=0}^n$ é chamado fecho afim, e denotado por $\mathfrak{K}(\mathtt{P})$

Intuitivamente, podemos pensar nos pontos $\{P_i\}_{i=0}^n$ como uma curva poligonal. Deste modo, o conjunto de combinações convexas desses pontos, denominado fecho convexo corresponde ao "interior" da poligonal.

Uma vez dispondo do conceito de fecho convexo, podemos definir:

Propriedade 2 (Propriedade do Fecho Convexo)**.** *Um AGC* $\mathbf{Curva}_n$ *com parâmetros* $\boldsymbol{Pr}$ *é dito satisfazer a propriedade do fecho convexo se* $(f : I \to \mathbb{A}) = \mathbf{Curva}_n(\{P_i\}_{i=0}^n, \boldsymbol{Pr})$ *satisfaz* $f(I) \subset \mathfrak{K}(\{P_i\}_{i=1}^n)$.

Em geral, a propriedade do fecho convexo e a propriedade de interpolação não costumam ocorrer juntas. Deste modo, não é uma surpresa muito grande descobrir que essa é uma propriedade não satisfeita por $\mathbf{InterP}_n$. Por exemplo, na figura abaixo temos um exemplo que mostra que, em geral a interpolação polinomial não satisfaz essa propriedade.

[Imagem]

Precisão Linear

Sendo a reta a forma geométrica mais elementar, é interessante que o esquema de traçado da curva, caso alimentado com pontos que se encontram sobre uma linha reta, forneça uma reta. Definamos essa propriedade:

Propriedade 3 (Precisão Linear)**.** *Seja* $\mathbf{Curva}_n$ *um AGC com parâmetros* $\boldsymbol{Pr}$*, e* $\{P_i\}_{i=0}^n$ *pontos contidos em uma reta* $R \subset \mathbb{A}$*.* $\mathbf{Curva}_n$ *é dita satisfazer a propriedade da Precisão Linear se* $(f : I \to \mathbb{A}) = \mathbf{Curva}(\{P_i\}_{i=1}^n, \{t_i\}_{i=0}^n)$ *satisfaz* $f(I) \subset R$.

Vamos utilizar o Algorítmo de Aitken para provar que a interpolação polinomial satisfaz essa propriedade.

Proposição 1.6. *A interpolação linear possui a propriedade da precisão linear.*

Demonstração. Faremos uma prova por indução.

Observe que o traço de $\mathbf{InterP}_1(\{P_0, P_1\}, \{t_0, t_1\})$ sempre está contido na reta gerada por P_0, P_1. Suponha então que $\mathbf{InterP}_{n-1}$ possui a propriedade da precisão linear. Dados então $\{P_i\}_{i=0}^{n} \subset R \subset \mathbb{A}$ onde R é uma reta. Consideremos $f_0(t) = \mathbf{InterP}_{n-1}(\{P_i\}_{i=0}^{n-1}, \{t_i\}_{i=0}^{n-1})$, $f_1(t) = \mathbf{InterP}_{n-1}(\{P_i\}_{i=1}^{n}, \{t_i\}_{i=1}^{n})$ e $t \in [t_0, t_n]$. Como $f(t)$ é dado por uma combinação afim entre $f_0(t)$ e $f_1(t)$, segue que $f(t) \in R$. Portanto, $\mathbf{InterP}_n$ também possui precisão linear. □

Diminuição da Variação

Um dos principais problemas da interpolação polinomial é o fato de ela introduzir oscilações entre os pontos de controle, tornando o design com essas curvas tarefa quase impossível. Uma explicação intuitiva para esse fato é que as curvas polinômiais possuem um número finito de graus de liberdade. A exigência de que essas curvas passem por determinados pontos "consome" esses graus de liberdade forçando a curva a possuir oscilações indesejadas entre os pontos de controle.

Não é tarefa fácil definir a propriedade para um AGC de não possuir oscilações. Utilizaremos a definição de Shoenberg dada no contexto de Teoria das Aproximações.

Proposição 1.7 (Propriedade da Diminuição da Variação)**.** *Seja $g : J \to \mathbb{A}$ uma curva parametrizada contínua, e $\{P_i\}_{i=1}^{n} \subset f(I)$. Um AGC $\mathbf{Curva}_n$ com parâmetros $\boldsymbol{Pr}$ é dito possuir a Propriedade da Diminuição da Variação se $f : I \to \mathbb{A} = \mathbf{Curva}_n(\{P_i\}_{i=1}^{n}, \{t_i\}_{i=1}^{n})$ satisfaz a propriedade de que para qualquer hiperplano $H \subset \mathbb{A}$ tem-se $\#(f(I) \cap H) \leq \#(g(J) \cap H)$.*

Observe que essa propriedade interpreta o AGC como um aproximante para uma determinada curva g. O hiperplano na definição funcionaria como um medidor de oscilações. Intuitivamente, quanto mais interseções com o hiperplano, maior o número de oscilações. Outra maneira de interpretar essa propriedade é pensar que, se $\mathbf{Curva}_n$ a possui, para qualquer hiperplano H e qualquer conjunto de pontos, curva f gerada possui menos interseções com esse plano do que qualquer curva contínua que passe por esses pontos.

Essa propriedade não é satisfeita pelo esquema de aproximação polinomial. De fato, quando se utilizam muitos pontos, um fenômeno conhecido é o aumento excessivo de ondulações na curva, conforme vemos na imagem:

[Imagem]

De fato, a variação acrescentada pelo esquema de interpolação polinomial é um problema tão grave que esse AGC é muito pouco utilizado em aplicações práticas.

Para exemplificar um AGC que possui a propriedade da diminuição da variação, considere $\mathbf{LoP}_n : \mathbb{A}^{n-1} \to \mathcal{C}(\mathbb{A})$ AGC sem parâmetros que mapeia o conjunto ordenado de pontos na curva linear por partes que une esses pontos (**LoP** de "Ligue os Pontos").

É um fato sabido que um segmento de reta só pode intersectar um hiperplano apenas uma vez (exceto no caso em que o segmento está contida no dito hiperplano). Também é um fato conhecido que, se o segmento de reta que une P_1 e P_2 possui interseção com um hiperplano H, então qualquer curva contínua que una esses dois pontos também possui uma interseção com H. (Segue do fato de que H divide $\mathbb{A}$ em 2 componentes conexas por caminhos). Portanto, não é dificil ver que $\mathbf{LoP}_n$ é um AGC que satisfaz a propriedade da Diminuição da Variação.

Convergência

Também no contexto de Teoria das Aproximações, podemos definir a propriedade de Convergência. Essa propriedade não é uma propriedade a ser satisfeita por um AGC $\mathbf{Curva}_n$ em si,

mas sim por uma família de AGCs $\{\mathbf{Curva}_n\}_{n=1}^{\infty}$.

Proposição 1.8 (Convergência)**.** *Seja $g : J \to \mathbb{A}$. Seja $\{\{P_i^n\}_{i=0}^n\}_{n=1}^{\infty} \subset g(J)$ uma sequência de subconjuntos finitos tal que $\{P_i^n\}_{i=0}^n \subset \{P_i^{n+1}\}_{i=0}^n$. Podemos interpretar essa sequência de sequências de pontos como uma amostragem de g que vai sendo sucessivamente refinada, ponto a ponto. Uma família de AGCs $\{\mathbf{Curva}_n\}_{n=1}^{\infty}$ é dita ser convergente se $(f_n : I_n \to \mathbb{A}) = \mathbf{Curva}_n(\{P_i^n\}_{i=0}^n, \mathrm{Pr}_n)$ satisfaz $(f_n(I_n)) \to g(J)$ (isto é, a distância de Hausdorff dos conjuntos converge para 0).*

Intuitivamente, isso quer dizer que, dada uma curva qualquer, se aplicarmos sucessivamente os AGCs da família em cada vez mais pontos da curva, no limite seríamos capazes de recuperar a curva. Ou, de maneira menos precisa ainda, quanto mais pontos de uma curva utilizados, mais próximos estaremos dessa curva.

Novamente, a interpolação polinomial não possui essa propriedade. O primeiro a provar esse fato foi Carl Runge. Em 1902 ele forneceu uma função (que desde então é conhecida como função de Runge) para a qual a interpolação de pontos igualmente espaçados faz com que a distância entre o polinômio interpolante e a função interpolada cresça indefinidamente.

[função de Runge]

Uma família de AGCs que possui a propriedade de convergência é a família LoP_n.

Capítulo 2

Curvas de Bézier

No capítulo anterior, propusemos a família de AGCs $\{\mathbf{InterP}_n\}_{n=1}^{\infty}$. Essa família apresentava algumas propriedades interessantes, como a possibilidade de predeterminação de pontos por onde a curva deve obrigatoriamente passar. O preço a ser pago por essa condição, no entanto, eram oscilações inesperadas ao longo do traço da curva. Em particular, vimos que essa família não possuia as propriedades de Diminuição da Variação nem de Convergência.

O objetivo nesse capítulo é definir uma família de AGCs que possua essas propriedades. Para isso, não é razoável esperar que possamos controlar pontos individuais da curva. Estaremos mais preocupados em controlar o formato da curva como um todo.

No entanto, é razoável pedir interpolação ao menos nos pontos finais da curva. Isso é necessário porque frequentemente o usuário irá necessitar "colar" diferentes curvas, e para obter continuidade é importante ter o controle exato da posição dos pontos extremos. Definimos essa propriedade

Propriedade 4 (Interpolação de Extremos)**.** *Um AGC* $\mathbf{Curva}_n$ *com parâmetros* $\boldsymbol{Pr}$ *é dita satisfazer a propriedade de interpolação de extremos se* $(f : [a, b] \to \mathbb{A}) = \mathbf{Curva}_n(\{P_i^n\}_{i=0}^n, \boldsymbol{Pr})$ *satisfaz* $f(a) = P_0, f(b) = P_n$

Para encontrar um novo AGC, seguiremos um procedimento análogo ao utilizado para definir $\mathbf{InterP}_n$. Vamos buscar um algoritmo construtivo que nos forneça uma curva, buscar uma formula fechada para a curva e a partir dai provar propriedades a respeito dessa curva.

2.1 Algoritmo de de Casteljau

Novamente, iniciaremos trataremos de analisar os casos mais simples antes de conseguir uma generalização.

Exemplo 3 (AGC de Bézier para $n = 2$)**.** Nos são fornecidos dois pontos $\{P_0, P_1\}$ e com eles devemos construir curva. A ideia mais intuitiva seria parametrizar a reta que liga P_0 a P_1. Na falta de informação adicional, parametrizaremos essa reta sobre o intervalo $[0, 1]$. Teremos então

$$\mathbf{Bezier}_1(\{P_0, P_1\})(t) = (1 - t)P_0 + tP_1$$

Esse caso não acrescentou nada de muito novo. Verifiquemos então para $n = 3$

Exemplo 4 (AGC de Bézier para $n = 3$)**.** Agora dispomos de 3 pontos $\{P_0, P_1, P_2\}$ e gostaríamos de encontrar uma curva. Podemos utilizar a mesma abordagem de "dividir e conquistar" utilizada anteriormente. Ou seja, podemos fazer

$$p_0^1(t) = \mathbf{Bezier}_1(P_0, P_1), \qquad p_1^1(t) = \mathbf{Bezier}_1(P_1, P_2)$$

e finalmente

$$\mathbf{Bezier}_2(P_0, P_1, P_2) = p_0^2(t) = (1-t)p_0^1(t) + tp_1^1(t)$$

A ideia aqui consiste em interpolações lineares encaixadas. Ilustremos o processo para calcular um ponto da curva. Fixemos $\tilde{t} \in [0,1]$, e definamos $P_i^0 = P_i \quad i = 0,1,2$

- Calculamos o ponto $P_0^1 = (1-\tilde{t})P_0^0 + \tilde{t}P_1^0$
- Calculamos o ponto $P_1^1 = (1-\tilde{t})P_1^0 + \tilde{t}P_2^0$
- Calculamos o ponto $P_0^2 = (1-\tilde{t})P_0^1 + \tilde{t}P_1^1$
- Esse ponto pertence a curva!

[Imagem]
Podemos organizar esse pontos na árvore:

$$\begin{array}{ccc} P_0^0 & & \\ & P_0^1 & \\ P_1^0 & & P_0^2 \\ & P_1^1 & \\ P_2^0 & & \end{array}$$

Onde cada dois elementos sucessivos em uma coluna geram um elemento da coluna seguinte por meio de interpolação linear.

Examinemos essa ideia aplicada a 4 pontos

Exemplo 5 (AGC de Bézier para $n = 3$)**.** Fixados 4 pontos $\{P_0, P_1, P_2, P_3\}$ vamos buscar uma curva utilizando a mesma abordagem. Podemos utilizar o algoritmo anterior sobre os 3 primeiros e sobre os 3 últimos pontos, obtendo:

$$p_0^2(t) = \mathbf{Bezier}_2(P_0, P_1, P_2), \qquad p_1^2(t) = \mathbf{Bezier}_2(P_1, P_2, P_3)$$

e finalmente

$$\mathbf{Bezier}_3(P_0, P_1, P_2, P_3) = p_0^3(t) = (1-t)p_0^2(t) + tp_1^2(t)$$

Conduzindo uma análise semelhante a feita anteriormente para o caso $n = 3$, encontramos a árvore de avaliação:

$$\begin{array}{ccccc} P_0^0 & & & & \\ & P_0^1 & & & \\ P_1^0 & & P_0^2 & & \\ & P_1^1 & & P_0^3 & \\ P_2^0 & & P_1^2 & & \\ & P_2^1 & & & \\ P_3^0 & & & & \end{array}$$

Onde cada coluna é obtida a partir de uma interpolação linear dos vizinhos da coluna anterior. Com isso estamos prontos para formular a fórma indutiva da curva de Bézier. Definimos indutivamente:

$$\begin{cases} \mathbf{Bezier}_1(P_0)(t) = P_0; \\ \mathbf{Bezier}_{n+1}(\{P_i\}_{i=0}^{n+1})(t) = (1-t)\mathbf{Bezier}_n(\{P_i\}_{i=0}^{n})(t) + t\mathbf{Bezier}_n(\{P_i\}_{i=1}^{n+1})(t) \end{cases}$$

Algorithm 3: Algorítmo de de Casteljau

```
function Bezier (P, t);
Input  : P = Vetor contendo n + 1 Pontos
         t = ponto pertencente ao intervalo [0, 1]
Output: p_t ∈ 𝔸
if tamanho(P) = 2 then
    return p(t) = (1 − t)P[0] + tP[1];
else
    V_0 = Bezier(P[1 : end], t);
    V_1 = Bezier(P[0 : end − 1], t);
    return p_t = (1 − t)V_0 + tV_1;
end
```

Podemos então definir o algoritmo recursivo de de Casteljau:

De posse do algoritmo, ficamos agora com a tarefa de encontrar uma fórmula fechada para a curva. Conforme vimos no capítulo anterior, possuir uma expressão em termos de Funções Misturadoras simplifica o trabalho de provar propriedades da curva. Deste modo, estamos interessados em buscar as funções misturadoras associadas ao Algoritmo de de Casteljau.

2.2 Polinômios de Bernstein

Diferente da abordagem do problema de interpolação polinomial, onde a unicidade da solução para a interpolação foi chave para encontrar as funções misturadoras, seguiremos uma rota diferente nas curvas de Bézier. Vamos primeiro definir indutivamente uma família de polinômios com propriedades interessantes. Em seguida, encontrar uma fórmula fechada para esses polinômios para somente então mostrar que esses polinômios são os polinômios misturadores associados ao algoritmo de de Casteljau.

Definição 2.1. Definimos a seguinte família de polinômios:

$$\theta_{k,n}(t) = \begin{cases} 1 & n = 0; \\ t\theta_{k-1,n-1}(t) + (1-t)\theta_{k,n-1}(t) & 0 < k < n; \\ (1-t)\theta_{0,n-1} & k = 0, n > 0; \\ t\theta_{n-1,n-1} & n = k > 0 \end{cases}$$

Antes de começar a trabalhar com esses polinômios, é bom compreender um pouco como eles funcionam. Observe que eles possuem 2 índices, k, n. Pode-se pensar que, fixado n segundo índice, existem n polinômios $\theta_{0,n}(t), \theta_{1,n}(t), \cdots, \theta_{n,n}(t)$ indexados pelo k primeiro índice. Em uma tabela, podemos gerar os primeiros polinômios:

$$\begin{array}{lllll} & & & & \theta_{0,3}(t) = (1-t)^3 \\ & & \theta_{0,2}(t) = (1-t)^2 & & \\ & \theta_{0,1}(t) = 1-t & & & \theta_{1,3}(t) = 3t(1-t)^2 \\ \theta_{0,0}(t) = 1 & & \theta_{1,2}(t) = 2t(1-t) & & \\ & \theta_{1,1}(t) = t & & & \theta_{2,3}(t) = 3t^2(1-t) \\ & & \theta_{2,2}(t) = t^2 & & \\ & & & & \theta_{3,3}(t) = t^3 \end{array}$$

Observando esses primeiros polinômios, é razoável a hipótese de que $\theta_{k,n} = \binom{n}{k} t^k (1-t)^{n-k}$. Provemos essa hipótese por indução

Proposição 2.1. $\theta_{k,n} = \binom{n}{k} t^k (1-t)^{n-k}$

Demonstração. Conforme observado na tabela, a base para indução já está provada. Assuma então válido para todos os pares $(k, n-1)$ onde $k = 0, \cdots, n-1$. Provemos então para os pares (k, n). Temos que dividir em 3 casos.

Para $k = 0$, temos

$$\theta_{0,n}(t) = (1-t)\theta_{0,n-1}(t) = (1-t)\binom{n-1}{0} t^0 (1-t)^{n-1-0} = (1-t)^n = \binom{n}{0} t^0 (1-t)^{n-0}$$

Para $k = n$, temos

$$\theta_{n,n}(t) = t\theta_{n-1,n-1}(t) = t\binom{n-1}{n-1} t^{n-1} (1-t)^{n-1-(n-1)} = t^n = \binom{n}{n} t^n (1-t)^{n-n}$$

Para $0 < k < n$, temos

$$\begin{aligned} \theta_{k,n}(t) = & \; t\theta_{k-1,n-1}(t) + (1-t)\theta_{k,n-1}(t) \\ = & \; t\tbinom{n-1}{k-1} t^{k-1} (1-t)^{n-1-(k-1)} + (1-t)\tbinom{n-1}{k} t^k (1-t)^{n-1-k} \\ = & \; \left(\tbinom{n-1}{k-1} + \tbinom{n-1}{k}\right) t^k (1-t)^{n-k} \\ = & \; \tbinom{n}{k} t^k (1-t)^{n-k} \end{aligned}$$

□

Observe que esses polinômios satisfazem uma relação semelhante a aquela que definiu o algoritmo. Eles são conhecidos como **Polinômios de Bernstein**. Eles aparecem em diversas aplicações na matemática, das quais destacamos duas em especial.

Considere n provas de Bernoulli independentes, com probabilidade p de sucesso e $(1-p)$ de fracasso. A probabilidade de obtenção de exatos k sucessos é dada por $\theta_{n,k}(p)$.

O Teorema da Aproximação de Weierstrass, um dos mais importantes teoremas em teoria da Aproximação, afirma que dada uma função contínua definida em um intervalo fechado $[a, b]$ é possível conseguir um polinômio que aproxima arbitrariamente essa função. Entretanto, em sua prova original, fica a questão a respeito de como obter tal polinômio. Sergei Bernstein, fazendo uso dos polinômios que agora levam seu nome, encontrou uma maneira de construir tal aproximação.

Em CADG, Paul de Casteljau percebeu cedo a conexão entre esses polinômios e o algoritmo que leva seu nome. Verifiquemos essa conexão.

Proposição 2.2. *Seja* $p(t) = \mathbf{Bezier}_n(\{P_i\}_{i=0}^n)$. *Então* $p(t) = \sum_{i=0}^n P_i \theta_{i,n}(t)$

Demonstração. Como definimos $\mathbf{Bezier}_n$ de maneira indutiva, indução é a maneira para verificar essa afirmação. Observe que $\mathbf{Bezier}_0(P_0)(t) = P_0$. Como $p(t) = P_0\theta_{0,0}(t) = P_0$, a identidade se verifica para $n = 0$. Suponha então que provamos a relação para $\mathbf{Bezier}_{n-1}$. Obtemos então:

$$\mathbf{Bezier}_n(\{P_i\}_{i=0}^n)(t) = p(t) = (1-t)\mathbf{Bezier}_{n-1}(\{P_i\}_{i=0}^{n-1}) + t\mathbf{Bezier}_{n-1}(\{P_i\}_{i=1}^n)$$

Substituindo a hipótese de indução, obtemos:

$$p(t) = (1-t)\sum_{i=0}^{n-1} P_i\theta_{i,n-1}(t) + t\sum_{i=0}^{n-1} P_{i+1}\theta_{i,n-1}(t)$$

Reorganizando, obtemos:

$$p(t) = P_0(1-t)\theta_{0,n-1}(t) + \sum_{i=1}^{n-1} P_i((1-t)\theta_{i,n-1}(t) + t\theta_{i-1,n-1}(t)) + P_n t\theta_{n-1,n-1}(t)$$

Observando as definições recursivas das funções $\theta_{k,n}$, encontramos:

$$p(t) = P_0\theta_{0,n}(t) + \sum_{i=1}^{n-1} P_i\theta_{i,n}(t) + P_n\theta_{n,n}(t) = \sum_{i=0}^{n} P_i\theta_{i,n}(t)$$

Provando a afirmação. □

2.3 Blossoms

Antes de partir para a demonstração das propriedades da Curva de Bézier utilizando a expansão de Bernstein, vale a pena mencionar uma maneira alternativa de se abordar essas curvas. Essa maneira foi proposta por de Casteljau e por Lyle Ramshaw, e por este último batizada de princípio Blossom. A ideia é escrever o polinômio resultante em sua forma polar. Definamos a forma polar de um polinômio. Para isso, necessitamos da seguinte definição:

Definição 2.2 (Aplicação Multiafim)**.** Uma aplicação multiafim é uma função $F : \mathbb{R}^n \to \mathbb{R}$ que é afim em cada variável. Isto é,

$$F(x_1, x_2, \cdots, \lambda_1 x_i + \lambda_2 x_i', \cdots, x_n) = \lambda_1 F(x_1, x_2, \cdots, x_i, \cdots, x_n) + \lambda_2 F(x_1, x_2, \cdots, x_i', \cdots, x_n)$$

Onde $\lambda_1 + \lambda_2 = 1$

Uma aplicação multiafim é dita ser simétrica se

$$F(x_1, \cdots, x_i, \cdots, x_j, \cdots, x_n) = F(x_1, \cdots, x_j, \cdots, x_i, \cdots, x_n)$$

para quaisquer i, j.

Definição 2.3 (Forma Polar)**.** Seja $p(x) = \sum_{i=0}^{n} a_i x^i$ um polinômio de grau n. A forma polar de p é uma aplicação multiafim simétrica $P : \mathbb{R}^n \to \mathbb{R}$ que satisfaz $P(t, t, t, \cdots, t) = p(t)$. O polinômio $p(t)$ é dito **polinômio diagonal** da forma polar P.

Para encontrar a forma Blossom do Algoritmo de de Casteljau observamos que ele consiste quase que exclusivamente de combinações afins. O procedimento que vamos seguir para encontrar essa forma é considerar essas combinações afins utilizando variáveis distintas.

Exemplo 6 (Forma Blossom de $\mathbf{Bezier}_2$)**.** O algoritmo para $\mathbf{Bezier}_2(P_0, P_1, P_2)$ requisitava o traçado das retas $p_0^1(t)$ entre P_1 e P_2, e $p_1^1(t)$ entre P_1 e P_2. Para essas retas, reservaremos a variável t_1. Assim, obtemos a seguinte função:

$$P_0^2(t_1, t_2) = (1-t_2)p_0^1(t_1) + t_2 p_1^1(t_1)$$

Em geral, de maneira recursiva, se assumimos que possuímos uma forma Blossom $\mathbf{Bezier}_n$ escrita como $P_0^n(t_1, \cdots, t_n)$, obtemos então a forma Blossom para $\mathbf{Bezier}_{n+1}$:

$$P_0^{n+1}(t_1, \cdots, t_{n+1}) = (1 - t_{n+1})P_0^n(t_1, \cdots, t_n) + t_{n+1}P_1^n(t_1, \cdots, t_n)$$

Não é dificil verificar que, construido dessa forma, $P_0^n(t, \cdots, t)$ fornece a curva de Bézier, visto que sua definição segue a mesma estrutura da definição de $\mathbf{Bezier}_n$. A multiafinidade segue do fato de que P_0^n é construida fazendo uso de combinações afins.

Dada uma forma polar qualquer, $F(t_1, \cdots, t_n)$, desejamos encontrar o polinômio diagonal associado, $f(t) = F(t, \cdots, t)$. para isso, seguiremos o seguinte procedimento:

- Utilizando a multiafinidade aplicada a t_1, escrevemos:

$$\begin{aligned} F(t, t_2, \cdots, t_n) = &\ F(0(1-t) + 1(t), t_2, \cdots, t_n) \\ = &\ (1-t)F(0, t_2, \cdots, t_n) + (1-t)F(1, t_2, \cdots, t_n) \end{aligned}$$

- Denominamos $F(0, t_2, \cdots, t_n) = F_0^1(t_2, \cdots, t_n)$, $F(1, t_2, \cdots, t_n) = F_1^1(t_2, \cdots, t_n)$. Assim, obtemos:

$$F(t, t_2, \cdots, t_n) = (1-t)F_0^1(t_2, \cdots, t_n) + (1-t)F_1^1(t_2, \cdots, t_n)$$

- Observe que reduzimos o problema de calcular o polinômio diagonal de uma aplicação multiafim a n variáveis a calcular o polinômio diagonal de 2 aplicações multiafins a $n-1$. Podemos seguir reduzindo cada um desses polinômios utilizando o mesmo princípio.

Organizando em uma árvore, obtemos:

$$\begin{array}{ccccccc} & & & & & \cdots & F(0,0,\cdots,0) \\ & & & & F(0,0,0,t_4,\cdots,t_n) & & \\ & & F(0,0,t_3,\cdots,t_n) & & & \cdots & F(1,0,\cdots,0) \\ & F(0,t_2,\cdots,t_n) & & & F(0,0,1,t_4,\cdots,t_n) & & \\ F(t_1,\cdots,t_n) & & F(0,1,t_3,\cdots,t_n) & & & \cdots & \vdots \\ & F(1,t_2,\cdots,t_n) & & & F(0,1,1,t_4,\cdots,t_n) & & \\ & & F(1,1,t_3,\cdots,t_n) & & & \cdots & \vdots \\ & & & & F(1,1,1,t_4,\cdots,t_n) & & \\ & & & & & \cdots & F(1,1,\cdots,1) \end{array}$$

Observe que essa estrutura é exatamente a estrutura do algoritmo de de Casteljau aplicado aos pontos $F(0,0,\cdots,0), F(1,0,\cdots,0), \cdots, F(1,1,\cdots,1)$. Portanto, observe que a estrutura Blossom codifica em si esse algoritmo. Mais ainda, dada uma forma polar $F(t_1, \cdots, t_n)$ utilizando essa observação, notamos que o polinômio diagonal desta nada mais é do que a curva de Bézier associada aos pontos de controle citados acima. Isso pode ser útil para extrair o polígono de controle de uma curva polinomial não traçada pelo algoritmo de Bezier.

2.4 Propriedades das Curvas de Bézier

Dispomos agora de 3 ferramentas distintas para provar as propriedades dos AGCs $\mathbf{Bezier}_n$. Podemos utilizar o algoritmo recursivo de de Casteljau, podemos utilizar a forma explícita em polinômios de Bernstein ou a forma Blossom. É interessante notar que todas as propriedades admitem provas utilizando cada uma das 3 ferramentas. Para essa dissertação, no entanto, será utilizado primariamente a forma de Polinômios de Bernstein, por ser uma forma de mais fácil visualização.

2.4.1 Interpolação de Extremos

O AGC $\mathbf{Bezier}_n$ possui essa propriedade. A verificação desse fato depende de um lema simples

Lema 2.3. $\theta_{k,n}(0) = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases}$ *e* $\theta_{k,n}(1) = \begin{cases} 1 & k = n \\ 0 & k \neq n \end{cases}$

Demonstração. Basta utilizar a expressão para $\theta_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}$. Obviamente $\theta_{k,n}(0) = 0$ sempre que $k > 0$ e $\theta_{k,n}(0) = 1$ se $k = 0$. Conclusão análoga segue para $\theta_{k,n}(1) = 0$ se $k < n$ e $\theta_{k,n} = 1$ se $k = n$. □

De posse desse lema a respeito das funções $\theta_{k,n}$, se torna tarefa simples provar:

Proposição 2.4. *O AGC* $\mathbf{Bezier}_n$ *possui a propriedade de interpolação de extremos.*

Demonstração. Sabendo que $\mathbf{Bezier}_n(\{P_i\}_{i=1}^n)(t) = p(t) = \sum_{k=0}^n P_k \theta_{k,n}(t)$, temos apenas que verificar que $p(0) = P_0$ e $p(1) = P_n$. Calculando, obtemos

$$p(0) = \sum_{k=0}^{n} P_k \theta_{k,n}(0) = P_0$$

$$p(1) = \sum_{k=0}^{n} P_k \theta_{k,n}(1) = P_n$$

Por causa do lema provado. □

2.4.2 Invariancia por Transformações Afins

Caso utilizassemos a forma Blossom para o polinômio resultante do algoritmo de de Casteljau, essa propriedade seguiria de maneira quase imediata. Isso se dá porque a forma Blossom é composta exclusivamente de combinações afins, e aplicações afins por definição comutam com essas. Entretanto, provaremos a invariancia utilizando a expansão de Bernstein. Provemos um lema adicional sobre as funções de Bernstein

Lema 2.5. *Os polinômios de Bernstein de grau n somam 1, isto é,* $\sum_{i=0}^n \theta_{i,n}(t) = 1$

Demonstração. Segue de uma aplicação direta do teorema do binômio de Newton.

$$\begin{array}{rl} \sum_{i=0}^n \theta_{i,n}(t) = & \sum_{i=0}^n \binom{n}{i} t^i (1-t)^{n-i} \\ = & (t + (1-t))^n = 1 \end{array}$$

□

Novamente, o significado desse lema é o de que todo ponto da curva é gerado como uma combinação afim de pontos do polígono de controle. Utilizando esse fato, a proposição se torna bastante clara.

Proposição 2.6. *O AGC* $\mathbf{Bezier}_n$ *é invariante por Transformações Afins*

Demonstração. Seja $T : \mathbb{A} \to \mathbb{A}$ uma transformação Afim. Se $\mathbf{Bezier}_n(\{P_i\}_{i=0}^n) = p(t)$, Temos:

$$T(p(t)) = T\left(\sum_{i=0}^{n} P_i \theta_{i,n}(t)\right) = \sum_{i=0}^{n} T(P_i)\theta_{i,n}(t) = \mathbf{Bezier}_n(\{T(P_i)\}_{i=0}^n)(t)$$

provando a afirmação. □

2.4.3 Precisão Linear

Essa, talvez, seja a única propriedade de $\mathbf{Bezier}_n$ que não provaremos utilizando a expressão em polinômios de Bernstein. Embora seja possivel construir uma prova que faça unicamente uso desses polinômios, ela seria muito envolvida e complicada, isto é, fugiria aos propósitos dessa dissertação. Provemos:

Proposição 2.7. *O AGC* $\mathbf{Bezier}_n$ *possui a propriedade da Precisão Linear.*

Demonstração. Lembremos que $\mathbf{Bezier}_n(\{P_i\}_{i=0}^n) = (1-t)\mathbf{Bezier}_{n-1}(\{P_i\}_{i=0}^{n-1}) + t\mathbf{Bezier}_{n-1}(\{P_i\}_{i=1}^n)$. Suponha por indução que se todos os P_i estão em um segmento reta então $\mathbf{Bezier}_j$ parametriza esse segmento, para todo $j \leq n-1$. Então, se os P_i estão todos sobre um segmento de reta $\mathbf{Bezier}_{n-1}(\{P_i\}_{i=0}^{n-1})$ parametriza um trecho desse segmento, $\mathbf{Bezier}_{n-1}(\{P_i\}_{i=1}^n)$ parametriza outro de modo que $\mathbf{Bezier}_n(\{P_i\}_{i=0}^n)$ contém apenas pontos desse segmento. A propriedade da precisão linear segue dai. □

2.4.4 Propriedade do Fecho Convexo

As propriedades provadas anteriormente coincidiam com as propriedades do AGC $\mathbf{InterP}_n$. A partir de agora buscamos provar as propriedades que distinguem esses dois AGCs. A primeira delas é a propriedade do fecho convexo, satisfeita por $\mathbf{Bezier}_n$ mas não por $\mathbf{InterP}_n$. Verifiquemos esse fato. Novamente, ele pode ser extraido de alguma propriedade fundamental das funções misturadoras (polinômios de Bersntein).

Lema 2.8. *Os polinômios de Bernstein satisfazem* $\theta_{k,n}(t) \geq 0 \quad \forall 0 \leq t \leq 1$

Demonstração. Observe que $\theta_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}$. Se $t > 0$ então $t^k > 0$. Se $t < 1$ então $(1-t)^{n-k} > 0$. Logo $0 \leq t \leq 1$ tem-se $\theta_{k,n}(t) \geq 0$ □

Portanto, em adição ao lema anterior, obtemos que os pontos da curva de Bézier não só são combinações afins dos pontos do polígono de controle, mas também são combinações convexas. Portanto, obviamente, a Curva de Bézier está inteiramente contida no fecho convexo dos pontos. Daí, obtemos a proposição:

Proposição 2.9. *O AGC* $\mathbf{Bezier}_n$ *possui a propriedade do Fecho Convexo.*

2.4.5 Controle Pseudo-Local

Uma propriedade que é bastante útil a um AGC é a propriedade de controle local. A intuição por trás dessa propriedade é a de que um ponto controla apenas uma vizinhança. Essa é uma propriedade bem valiosa para um AGC, visto que possibilita editar a curva por trechos, sem alterar globalmente a curva em cada edição local.

As curvas de Bézier não possuem controle local. De fato, é impossível para um AGC puramente polinomial possuir essa propriedade. Entretanto, no que diz respeito a controle local, as curvas de Bézier possuem uma propriedade bem próxima.

Definição 2.4 (Função Unimodal)**.** Uma função é dita ser unimodal se ela possui apenas um ponto de máximo dentro de seu domínio.

Um exemplo de função unimodal clássico na matemática é a função gaussiana. Pode-se interpretar a unimodalidade como uma forma de concentração. Uma função unimodal possui apenas um foco de concentração. Baseado nessa intuição, podemos definir:

Definição 2.5 (Controle Pseudo-Local)**.** Um AGC $\mathbf{Curva}_n$ com parâmetros $\mathbf{Pr}$ é dita possuir Controle Pseudo-Local se $p(t) = \mathbf{Curva}_n(\{P_i\}_{i=0}^n, \mathbf{Pr})$, em sua expressão em funções misturadoras:

$$p(t) = \sum_{i=0}^{n} P_i F_i(t)$$

é tal que essas funções misturadoras $F_i(t)$ são unimodais.

Observe que tudo o que pode ser dito a respeito dos AGCs, uma vez fixados os parâmetros Pr, está codificado nas funções misturadoras. Essa visão de funções misturadoras é útil na medida em que torna a análise dos AGCs independente dos pontos. Uma função misturadora F_i pode ser interpretada como a "Função de Influência" de um determinado ponto P_i.

Requisitar unimodalidade da "Função de Influência" equivale a idéia intuitiva de que, ao mover o ponto P_i, as alterações da curva estarão concentradas apenas em torno do (único) máximo dessa função.

Procedamos então provar a propriedade de Controle Pseudo-Local para as Curvas de Bézier. Em essência, o trabalho se reduz a provar a proposição:

Proposição 2.10. *Os polinômios de Bernstein (Funções Misturadoras para a* $\mathbf{Bezier}_n$*) são unimodais.*

Demonstração. Visto que os polinômios são funções diferenciáveis, tudo o que devemos verificar é que $\theta'_{k,n}(t)$ possui exatamente uma raiz no interior de $[0,1]$ Calculando, obtemos:

$$\theta_{k,n}(t) = \binom{k}{n} t^k (1-t)^{n-k}$$

$$\begin{aligned}
\theta'_{k,n}(t) = & \binom{n}{k}\left(kt^{k-1}(1-t)^{n-k} - (n-k)t^k(1-t)^{n-k-1}\right) \\
= & \binom{n}{k} t^{k-1}(1 - t^{n-k-1}(kt - (n-k)(1-t)) \\
= & \binom{n}{k} t^{k-1}(1 - t^{n-k-1}(k - nt)
\end{aligned}$$

Os pontos extremos são os pontos em que a derivada se anula. Isso ocorre apenas se $t = 0$, $t = 1$ ou $t = \frac{k}{n}$. Como a derivada possui apenas uma raiz interior a $[0,1]$ segue que essa raiz é o ponto de máximo da curva. Portanto, $\theta_{k,n}$ é unimodal e possui seu máximo em $\frac{k}{n}$. □

2.4.6 Diminuição da Variação

O incremento na variação é a propriedade que torna inviável o uso de $\mathbf{InterP}_n$ para o design de curvas. Portanto, é importante verificar que $\mathbf{Bezier}_n$ não sofre do mesmo problema. Para verificar essa propriedade, vamos primeiramente desenvolver uma ferramenta que será útil em si mesma, a Elevação de Grau.

Elevação de Grau para a Curva de Bezier

Considere, para propósito dessa discussão, $p(t) = \mathbf{Bezier}_n(\{P_i\}_{i=0}^n)(t)$ um polinômio gerado por $n+1$ pontos de controle. Nosso objetivo é obter $n+2$ pontos de controle $\{Q_j\}_{j=0}^{n+1}$ capazes de gerar **o mesmo polinômio**. Embora o nome pareça sugerir outra coisa, não estamos interessados em elevar o grau do polinômio $p(t)$, mas sim representá-lo em uma base que contenha polinômios de grau mais elevado. Em outras palavras, nosso objetivo é:

$$p(t) = \mathbf{Bezier}_{n+1}(\{Q_j\}_{j=0}^{n+1})(t)$$

Para isso, utilizaremos a expressão em polinômios de Bernstein:

$$p(t) = \sum_{i=0}^{n} P_i \theta_{i,n}(t) = \sum_{j=0}^{n+1} Q_j \theta_{j,n+1}(t)$$

A rota mais fácil para enxergar a relação entre $\{P_i\}_{i=0}^{n}$ e $\{Q_j\}_{j=0}^{n+1}$ é tentar construir os valores de Q_j a partir de P_i. Para isso, necessitamos das seguintes propriedades dos polinômios de Bernstein:

Lema 2.11.

$$\theta_{k,n}(t) = \frac{n}{k} t \theta_{k-1,n-1}(t)$$

,

$$\theta_{k,n}(t) = \frac{n}{n-k}(1-t)\theta_{k,n-1}(t)$$

Demonstração. As propriedades seguem da expansão binomial dos polinômios de Bernstein. Note que:

$$\theta_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k} = t \left(\frac{n}{k}\right) \binom{n-1}{k-1} t^{k-1} (1-t)^{n-1-(k-1)} = \frac{n}{k} t \theta_{k-1,n-1}(t)$$

Analogamente, segue:

$$\theta_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k} = (1-t) \left(\frac{n}{n-k}\right) \binom{n-1}{k} t^k (1-t)^{(n-1)-k} = \frac{n}{n-k}(1-t)\theta_{k,n-1}(t)$$

□

Utilizando esse resultado, encontramos:

Lema 2.12.

$$\theta_{k,n}(t) = \frac{k+1}{n+1}\theta_{k+1,n+1}(t) + \frac{n+1-k}{n+1}\theta_{k,n+1}(t)$$

Demonstração. Iniciamos com a observação:

$$\theta_{k,n}(t) = (1-t)\theta_{k,n}(t) + t\theta_{k,n}$$

Deslocando os índices dos resultados obtidos no lema anterior, obtemos as identidades:

$$(1-t)\theta_{k,n}(t) = \frac{k+1}{n+1}\theta_{k+1,n+1}(t)$$

$$t\theta_{k,n}(t) = \frac{n-k+1}{n+1}\theta_{k,n+1}(t)$$

Substituindo essas identidades, obtemos:

$$\theta_{k,n}(t) = \frac{k+1}{n+1}\theta_{k+1,n+1}(t) + \frac{n-k+1}{n+1}\theta_{k,n+1}(t)$$

□

De posse desse lema, somos capazes de obter os Q_j a partir dos P_i. Obtemos por fim a proposição:

Proposição 2.13.

$$\mathbf{Bezier}_n(\{P_i\}_{i=0}^n) = \mathbf{Bezier}_{n+1}(\{Q_j\}_{j=0}^{n+1})$$

onde

$$Q_j = \frac{jP_{j-1} + (n+1-j)P_j}{n+1}$$

Demonstração. Basta utilizar a relação obtida no lema para a expansão de $\mathbf{Bezier}_n(\{P_i\}_{i=0}^n)$. Temos então:

$$\begin{array}{rcl} p(t) = & \sum_{i=0}^n P_i\theta_{i,n}(t) \\ = & \sum_{i=0}^n P_i\left(\frac{i+1}{n+1}\theta_{i+1,n+1}(t) + \frac{n-i+1}{n+1}\theta_{i,n+1}(t)\right) \\ = & \sum_{i=1}^n P_{i-1}\frac{i}{n+1}\theta_{i,n+1}(t) + \sum_{i=0}^n P_i\frac{n-i+1}{n+1}\theta_{i,n+1}(t) \\ = & P_0\theta_{0,n+1}(t) + \sum_{i=1}^n \left(\frac{iP_{i-1}+(n-i+1)P_i}{n+1}\right)\theta_{i,n+1} + P_{n+1}\theta_{n+1,n+1} \end{array}$$

Definimos $Q_0 = P_0$, $Q_{n+1} = P_n$ e $Q_i = \frac{iP_{i-1}+(n-i+1)P_i}{n+1}$. Observe que, embora definidos a parte, Q_0 e Q_{n+1} também satisfazem a relação proposta. Por fim, obtemos:

$$p(t) = \sum_{i=0}^{n+1} Q_i\theta_{i,n+1}$$

provando a proposição. □

Elevação de Grau Repetida

Com a ferramenta obtida anteriormente somos capazes de elevar em o grau da curva de Bezier de 1 em 1. Gostaríamos, no entanto, de conseguir uma elevação de grau repetida, isto é, elevar o grau em k utilizando um único passo. Provemos que isso é realizado conforme a proposição a seguir:

Proposição 2.14.

$$\mathbf{Bezier}_n(\{P_i\}_{i=1}^n) = \mathbf{Bezier}_{n+k}(\{Q_j\}_{j=0}^{n+k})$$

Onde

$$Q_j = \sum_{i=0}^n P_i\binom{n}{i}\frac{\binom{k}{j-i}}{\binom{n+k}{j}}$$

Demonstração. Iniciemos verificando a afirmação para $k = 1$. Temos:

$$\begin{array}{rcl} Q_j = & \sum_{i=0}^n P_i\binom{n}{i}\frac{\binom{1}{j-i}}{\binom{n+1}{j}} \\ = & P_{j-1}\binom{n}{j-1}\frac{\binom{1}{1}}{\binom{n+1}{j-1}} + P_j\binom{n}{j}\frac{\binom{1}{0}}{\binom{n+1}{j}} \\ = & P_{j-1}\frac{j}{n+1} + P_j\frac{n-j+1}{n+1} \end{array}$$

Que coincide com a elevação de grau encontrada anteriormente.

Suponha, então, que essa fórmula produza a k-ésima elevação de grau correta. Utilizando o procedimento anterior, encontremos a elevação de grau $k+1$. Construimos inicialmente a k-ésima elevação de grau:

$$\mathbf{Bezier}_n(\{P_i\}_{i=0}^n) = \mathbf{Bezier}_{n+k}(\{R_l\}_{l=0}^{n+k})$$

Para elevar em mais 1 o grau dos R_j, encontramos:

$$\mathbf{Bezier}_{n+k}(\{R_l\}_{l=0}^{n+k}) = \mathbf{Bezier}_{n+k+1}(\{Q_j\}_{j=0}^{n+k+1})$$

onde os pontos Q_j que satisfazem:

$$Q_j = R_{j-1}\frac{j}{n+k+1} + R_j\left(\frac{n+k+1-j}{n+k+1}\right)$$

Aplicando a hipótese de indução sobre os pontos R_j, obtemos:

$$\begin{aligned} Q_j &= \sum_{i=0}^{n} P_i\binom{n}{i}\left(\frac{j}{n+k+1}\frac{\binom{k}{j-1-i}}{\binom{n+k}{j-1}} + \frac{n+k+1-j}{n+k+1}\frac{\binom{k}{j-i}}{\binom{n+k}{j}}\right) \\ &= \sum_{i=0}^{n} P_i\binom{n}{i}\left(\frac{n+k+1-j}{n+k+1}\frac{\binom{k}{j-1-i}}{\binom{n+k}{j}} + \frac{n+k+1-j}{n+k+1}\frac{\binom{k}{j-i}}{\binom{n+k}{j}}\right) \\ &= \sum_{i=0}^{n} P_i\binom{n}{i}\left(\frac{n+k+1-j}{n+k+1}\right)\frac{\binom{k+1}{j-i}}{\binom{n+k}{j}} \\ &= \sum_{i=0}^{n} P_i\binom{n}{i}\frac{\binom{k+1}{j-i}}{\binom{n+k+1}{j}} \end{aligned}$$

Provando por indução o enunciado. □

Convergência do Polígono de Controle

Resolvida essa questão a respeito de como elevar o grau de uma curva de Bézier, vamos mostrar um resultado um pouco mais profundo a respeito dessa operação. Foi conjecturado por Forrest e efetivamente provado por Farin que, elevando sucessivamente o grau da curva obtem-se no limite um polígono de controle converge pra curva. Para enunciar corretamente esse resultado, necessitamos as seguintes definições:

Definição 2.6 (Mapa Elevação de Grau)**.** Definimos a função $E_k : \mathbb{A}^n \to \mathbb{A}^{n+k}$ que mapeia os pontos $\{P_i\}_{i=0}^{n}$ em $\{Q_j\}_{j=0}^{n+k}$ conforme o metodo de elevação de grau citado acima.

Definição 2.7 (AGC "Ligue os Pontos")**.** No capítulo anterior, definimos informalmente o AGC $\mathbf{LoP}_n$ que recebe n pontos e os une por segmentos de reta. Definimos agora com mais precisão:
$\mathbf{LoP}_n(\{P_i\}_{i=0}^{n}) = p : [0,1] \to \mathbb{A}$ onde,

$$p(t) = \begin{cases} P_i & ; t = \frac{i}{n} \\ P_{i-1}n\left(\frac{i}{n} - t\right) + P_i n\left(t - \frac{i-1}{n}\right) & ; \frac{i-1}{n} < t < \frac{i}{n} \end{cases}$$

De posse dessas definições, podemos finalmente enunciar o resultado:

Proposição 2.15.

$$\mathbf{Bezier}_n(\{P_i\}_{i=0}^{n}) = \lim_{k\to\infty} \mathbf{LoP}_{n+k}(E_k(\{P_i\}_{i=0}^{n}))$$

uniformemente.

Demonstração. Para provar esse resultado, sejam

$$p(t) = \mathbf{Bezier}_n(\{P_i\}_{i=0}^{n})$$

$$p_k(t) = \mathbf{LoP}_{n+k}(E_k(\{P_i\}_{i=0}^{n}))$$

e fixe $t_0 \in [0,1]$. Nosso objetivo primário é provar a convergencia pontual $\lim_{k\to\infty} p_k(t_0) = p(t_0)$. Como convergência pontual sobre um compacto implica convergência uniforme, a prova da convergência pontual basta.

Além disso, ao invés de calcular $p_k(t_0)$, vamos simplificar ainda mais o problema, calculando $p_k\left(\frac{i_k}{n+k}\right)$. onde o valor de i_k é selecionado de modo que a fração seja o mais próxima de t_0 possível. Deste modo, vamos obter uma sequência de frações que satisfaz $\lim_{k\to\infty} \frac{i_k}{n+k} = t_0$ Isso é feito para podermos trabalhar exclusivamente com os pontos gerados, e não com a interpolação linear entre 2 pontos dados. Assim, após as simplificações, resulta que nosso objetivo é provar

$$\lim_{k\to\infty} p_k\left(\frac{i_k}{n+k}\right) = p(t_0)$$

Observemos agora que, conforme a definição de $\mathbf{LoP}_n$, temos que $p_k\left(\frac{i_k}{n+k}\right) = Q_{i_k}$ onde $E_k(\{P_j\}_{j=0}^n) = \{Q_i\}_{i=0}^{n+k}$. Sabemos que $Q_{i_k} = \sum_{j=0}^n P_j \binom{n}{j} \frac{\binom{k}{i_k-j}}{\binom{n+k}{i_k}}$

$$\lim_{k\to\infty} p_k\left(\frac{i_k}{n+k}\right) = \lim_{k\to\infty} Q_{i_k} = \lim_{k\to\infty} \sum_{j=0}^{n} P_j \binom{n}{j} \frac{\binom{k}{i_k-j}}{\binom{n+k}{i_k}}$$

Utilizando a desigualdade de Stirling para números binomiais, pode-se demonstrar que:

$$\lim_{\frac{i_k}{n+k}\to t_0} \frac{\binom{k}{i_k-j}}{\binom{n+k}{i_k}} = t_0^j (1-t_0)^{n-j}$$

Esse é um resultado familiar em probabilidade, e é utilizado para provar que a distribuição hipergeométrica converge para a distribuição binomial. Utilizando esse resultado, obtemos finalmente:

$$\lim_{k\to\infty} p_k\left(\frac{i_k}{n+k}\right) = \sum_{j=0}^{n} P_j \binom{n}{j} t_0^j (1-t_0)^{n-j}$$

Que é exatamente a expressão de $p(t_0)$ em polinômios de Bernstein. □

Diminuição da Variação

Embora o resultado anterior seja bastante interessante do ponto de vista teórico, ele não tem muita utilidade prática. Pode-se demonstrar que embora, de fato, o polígono de controle convirja para a curva, essa convergência é muito lenta para aplicações práticas. Com isso, queremos dizer que para obter uma boa aproximação da curva real, é necessário uma elevação muito grande no grau do polígono de controle, e o custo computacional para esse feito não justifica esse como um bom método de renderização para a curva.

Essa técnica possui duas aplicações principais em CAGD. A primeira consiste em aumentar o número de pontos de controle de uma dada curva. Isto é, para uma aproximação inicial do traço que deseja representar, um designer pode crer que uma curva de Bézier com n pontos de controle seja o suficiente. Após algumas iterações com essa, no entanto, pode chegar a conclusão que necessita de mais alguns graus de liberdade. Ele pode então utilizar a elevação de grau para obter outros pontos que o permitem controlar de maneira mais refinada o traçado da curva. No último capítulo dessa dissertação veremos outra maneira com a qual pode-se conseguir mais pontos de controle para uma curva de Bézier, chamado de Subdivisão.

Outra utilidade para essa técnica, agora muito mais teórica é provar que $\mathbf{Bezier}_n$ possui a propriedade da diminuição da variação. Lembre-se que afirmou-se anteriormente que o AGC $\mathbf{LoP}_n$ possuia a propriedade da Diminuição da Variação. A prova de que $\mathbf{Bezier}_n$ possui essa propriedade consiste em enxerga-la como o limite calculado acima. Como se diminui a variação a cada passo de $\mathbf{LoP}_{n+k}$ para $\mathbf{LoP}_{n+k+1}$ conclui-se então a proposição:

Proposição 2.16. $\mathbf{Bezier}_n$ *possui a propriedade da Diminuição da Variação.*

2.4.7 Convergência

A propriedade da convergência, conforme haviamos mencionado no capítulo anterior, segue da Teoria da Aproximação de Funções. Os polinômios de Bernstein foram por ele utilizados neste contexto. Utilizando o trabalho dele, provaremos que o AGC $\mathbf{Bezier}_n$ possui a propriedade da convergência.

Considere, pois, uma função contínua $f : [0, 1] \to \mathbb{A}$. Faremos a seguinte definição:

Definição 2.8 (Aproximação de Bernstein)**.** Definimos a n-ésima aproximação de Bernstein de uma função f como:

$$B_n(f, t) = \sum_{i=0}^{n} f\left(\frac{k}{n}\right)\theta_{i,n}(t);$$

Sergei Bernstein provou o seguinte teorema:

Teorema 2.17 (Teorema de Bernstein)**.** *Se* $f : [0, 1] \to \mathbb{A}$ *é contínua então* $\lim_{n\to\infty} B_n(f, t) = f(t)$ *com convergência uniforme. Mais ainda, se* $f \in C^{(i)}[0, 1]$ *então* $f^{(i)}(t) = \lim_{n\to\infty} B_n^{(i)}(f, t)$ *também uniformemente.*

A prova para esse teorema requer ferramentas um pouco mais avançadas, fora do escopo dessa dissertação. De todo modo, esse teorema traz como corolário o resultado que necessitamos:

Corolário 2.18. *O AGC* $\mathbf{Bezier}_n$ *possui a propriedade de Convergencia.*

2.5 Tópicos adicionais

Nesta seção, incluimos alguns tópicos que possuem interesse próprio, mas que não se encaixam na discussão anterior.

2.5.1 Algoritmo de Hohner

Embora o Algoritmo de de Casteljau possua várias propriedades interessantes, e serviu para construir toda a teoria das Curvas de Bezier aqui expostas, em termos práticos ele segue sendo um algoritmo relativamente ineficiente. Por ser recursivo, consome-se muita memória para calcular a curva. Embora isso não torne seu uso proibitivo, existem algoritmos de avaliação da curva de Bézier mais eficientes. Mostraremos nessa seção um algoritmo de avaliação.

A ideia consiste em realizar uma avaliação direta da curva utilizando os Polinômios de Bernstein. Recordemos que $\mathbf{Bezier}_n(\{P_i\}_{i=1}^n)(t) = p(t)$ então temos:

$$p(t) = \sum_{i=0}^{n} P_i\theta_{i,n}(t) = \sum_{i=0}^{n} P_i\binom{n}{i}t^i(1-t)^{n-i}$$

Por conveniência, chamaremos $(1-t) = s$. Abriremos então a soma:

$$p(t) = \sum_{i=0}^{n} P_i \binom{n}{i} t^i s^{n-i} = P_0 \binom{n}{0} t^0 s^n + P_1 \binom{n}{1} t^1 s^{n-1} + \cdots P_{n-1} \binom{n}{n-1} t^{n-1} s^1 + P_n \binom{n}{0} t^n s^0$$

A ideia consiste em uma reorganização inteligente dos termos, de modo a economizar multiplicações. Deste modo, obtemos:

$$p(t) = P_n \binom{n}{n} t^n + s\left(P_{n-1}\binom{n}{n-1} t^{n-1} + s\left(P_{n-2}\binom{n}{n-2} t^{n-2} + \cdots \left(P_1 \binom{n}{1} t^1 + s\left(P_0 \binom{n}{0} t^0\right)\right)\cdots\right)\right)$$

Utilizando esse esquema de avaliação, economiza-se diversas multiplicações. Para tornar o algoritmo ainda mais rápido, fazemos uso da relação:

$$\binom{n}{j} = \frac{n-j-1}{j}\binom{n}{j-1}$$

Com isso, obtemos:

Algorithm 4: Algorítmo de Hohner

```
1  function Hohner (P, t);
   Input  : P = Vetor contendo n + 1 Pontos
            t = ponto pertencente ao intervalo [0, 1]
   Output: p_t ∈ 𝔸
2  s := (1 − t)
3  B := 1
4  N := length(P)
5  p_t = B ∗ P[0]
6  For i = 1 to N
7     B := B ∗ (N − i + 1)/i
8     p_t := P[i] ∗ B ∗ t^i + s ∗ p_t
9  EndFor
10 return p_t
```

2.5.2 Derivada da Curva de Bézier

Dada uma curva $p(t) = \mathbf{Bezier}_n(\{P_i\}_{i=0}^n)(t)$, estamos interessados em obter a curva derivada, $p'(t)$. Antes de efetuar os cálculos, é válida uma pequena digressão a respeito do significado dessa operação.

O objetivo da operação de derivação é obter o vetor tangente à curva em determinado ponto. Então, estritamente falando, a curva derivada é um objeto cuja natureza é diferente da curva primitiva. O traçado da curva derivada se chama Hodográfico, e possui algumas aplicações em física.

Fazendo algumas contas, obtemos uma relação recursiva para a derivada da curva de Bézier.

$$\begin{aligned}
\theta'_{k,n}(t) = & \binom{n}{k}\left(kt^{k-1}(1-t)^{n-k} - (n-k)t^k(1-t)^{n-k-1}\right) \\
= & \; n\binom{n-1}{k-1}t^{k-1}(1-t)^{(n-1)-(k-1)} - n\binom{n-1}{k}t^k(1-t)^{(n-1)-k} \\
= & \; n\left(\theta_{k-1,n-1}(t) - \theta_{k,n-1}(t)\right)
\end{aligned}$$

Com isso, podemos construir a curva derivada:

Proposição 2.19. *A curva derivada é dada por:*

$$p'(t) = \sum_{i=0}^{n-1} Q_i \theta_{i,n-1}(t)$$

Onde $Q_i = n(P_{i+1} - P_i)$

Demonstração. Sabemos que:

$$p'(t) = \sum_{i=0}^{n} P_i \theta'_{k,n}(t)$$

Utilizando a observação anterior, obtemos:

$$\begin{array}{rl} p'(t) = & \sum_{0=1}^{n} P_i n \left(\theta_{i-1,n-1}(t) - \theta_{i,n-1}(t)\right) \\ = & \sum_{i=1}^{n} P_i n \theta_{i-1,n-1}(t) - \sum_{i=0}^{n} P_i n \theta_{i,n-1}(t) \\ = & \sum_{i=0}^{n-1} P_{i+1} n \theta_{i,n-1}(t) - \sum_{i=0}^{n} P_i n \theta_{i,n-1}(t) \\ = & \sum_{i=0}^{n-1} n(P_{i+1} - P_i) \theta_{i,n-1}(t) \\ = & \sum_{i=0}^{n-1} Q_i \theta_{i,n-1}(t) \end{array}$$

Onde $Q_i = n(P_{i+1} - P_i)$. □

Denotando por $\Delta(\{P_i\}_{i=0}^n) = \{Q_j\}_{j=0}^{n-1}$ operador de diferenças divididas, obtemos que:

$$\mathbf{Bezier}'_n(\{P_i\}_{i=0}^n) = \mathbf{Bezier}_{n-1}(\Delta(\{P_i\}_{i=0}^n))$$

Dessa maneira, podemos construir derivadas de ordem superior apenas repetindo essa observação. Com isso, obtemos a proposição:

Proposição 2.20. *Se* $p(t) = \mathbf{Bezier}_n(\{P_i\}_{i=0}^n)$ *então* $p^{(k)}(t) = \mathbf{Bezier}_{n-k}(\Delta^k(\{P_i\}_{i=0}^n))$

2.5.3 Colagem de Curvas

Um grande problema com as curvas de Bézier é o fato de que, quanto maior o número de pontos de controle, maior o grau do polinômio obtido. Em geral não é eficiente trabalhar com polinômios de graus muito altos visto que isso acarreta em um custo computacional elevado nas operações. Para a avaliação de poucas curvas isso não é um problema muito grave. Entretanto, caso hajam milhares de curvas de Bézier em um determinado objeto gráfico (como por exemplo, uma página de um documento de texto onde as fontes são codificadas com curvas de Bézier), curvas com o grau elevado podem vir a se tornar um problema.

Para obter flexibilidade de Design e ainda assim se manter trabalhando com curvas de grau baixo, podemos utilizar várias curvas de Bézier no traçado de uma curva individual. Desejamos, no entanto, obter continuidade e idealmente algum grau de suavidade na união dessas curvas. Portanto, nosso objetivo nesta seção é analisar como unir duas curvas distintas de modo a gerar uma única curva que seja suave no ponto de uniao.

Para formalizar um pouco a discussão, seja $p(t) = \mathbf{Bezier}_n(\{P_i\}_{i=0}^n)$ e $q(t) = \mathbf{Bezier}_n(\{Q_j\}_{j=1}^n)$. Nosso objetivo inicial é obter: $p(1) = q(0)$. Conforme mencionamos anteriormente, as curvas de Bézier possuem a propriedade da interpolação de extremos. Portanto, para atingir esse objetivo, basta assegurar-se que $P_n = Q_0$.

[Imagem]

O problema começa a ficar interessante caso estejamos interessados em suavidade, isto é, caso estejamos interessados em $p'(1) = q'(0)$. Para isso, utilizaremos a seção anterior.

Capítulo 3

B-Splines

Capítulo 4

Aplicações

Apêndice A

Apêndice

Referências Bibliográficas

[1] Pierre E. Bézier, *A personal view of progress in computer aided design*, SIGGRAPH Comput. Graph. **20** (1986), no. 3, 154–159.

[2] Wolfgang Boehm, *Inserting new knots into b-spline curves*, Computer-Aided Design **12** (1980), no. 4, 199 – 201.

[3] Elaine Cohen, Tom Lyche, and Richard Riesenfeld, *Discrete b-splines and subdivision techniques in computer-aided geometric design and computer graphics*, Computer Graphics and Image Processing **14** (1980), no. 2, 87 – 111.

[4] Elaine Cohen, Richard F. Riesenfeld, and Gershon Elber, *Geometric modeling with splines : an introduction*, A. K. Peters, Natick (Mass.), 2001.

[5] Carl De Boor, *A practical guide to splines*, Applied mathematical sciences, Springer-Verlag, New York, 2001.

[6] Paul de Faget de Casteljau, *Polynomials, {POLar} forms, and interpolation*, Mathematical Methods in Computer Aided Geometric Design {II} (Tom LycheLarry L. Schumaker, ed.), Academic Press, 1992, pp. 57 – 68.

[7] ———, *De casteljau's autobiography: My time at citroën*, Computer Aided Geometric Design **16** (1999), no. 7, 583 – 586.

[8] Gerald Farin, *Curves and surfaces for computer aided geometric design: A practical guide*, Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[9] Gerald Farin, *History of curves and surfaces in cagd*, 1993.

[10] Jean Gallier, *Curves and surfaces in geometric modeling: Theory and algorithms*, 2000.

[11] Jai Menon, Brian Wyvill, Chandrajit Bajaj, Jules Bloomenthal, Baining Guo, John Hart, Geo Wyvill, and Chandrajit Bajaj, *Implicit surfaces for geometric modeling and computer graphics speaker biographies*, 1996.

[12] Lyle Ramshaw, *Theoretical foundations of computer graphics and cad*, ch. Béziers and B-splines as Multiaffine Maps, pp. 757–776, Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[13] Richard Franklin Riesenfeld, *Applications of b-spline approximation to geometric problems of computer-aided design.*, Ph.D. thesis, Syracuse, NY, USA, 1973, AAI7408299.

[14] I. J. Schoenberg, *I. j. schoenberg selected papers*, ch. Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions, pp. 3–57, Birkhäuser Boston, Boston, MA, 1988.