

**Artur Filipe Ewald Wuerges, Rodrigo Braz Monteiro, Wilson Tadeu
Ferreira Müller Júnior**

*Utilizando curvas de Bézier para renderizar
superfícies curvas em tempo real*

Florianópolis

2007

**Artur Filipe Ewald Wuerges, Rodrigo Braz Monteiro, Wilson Tadeu
Ferreira Müller Júnior**

*Utilizando curvas de Bézier para renderizar
superfícies curvas em tempo real*

Trabalho final entregue para a disciplina de
Geometria Analítica, no curso de Ciência da
Computação.

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis

2007

1 Introdução

Até o começo dos anos 80, a computação gráfica era uma área pequena e pouco explorada da computação. Faltava o *hardware* e o *software* para tornar esta tecnologia acessível às massas (FOLEY et al., 1990). Com o surgimento de computadores que vinham com *display* embutido (Xerox Star, por exemplo), começou a se tornar viável a criação de interfaces gráficas.

O desenvolvimento da computação gráfica continuou, e atingiu-se um ponto em que ela está presente não apenas nas interfaces criadas para o usuário de computadores, como também em áreas como cinema, medicina, ciência e engenharia. Ao mesmo tempo, vem-se exigindo cada vez mais qualidade e velocidade dos gráficos utilizados. Por isso, esta continua sendo uma área de interesse para a pesquisa em Ciência da Computação.

Em simulações computacionais que apresentam um ambiente gráfico tridimensional, como aplicativos de *Computer-Aided Design* (CAD), jogos eletrônicos e simuladores, os elementos gráficos devem ser processados e exibidos em tempo real. O processo no qual as curvas e elementos matemáticos que descrevem a superfície a ser exibida são convertidas em imagens bidimensionais (*bitmaps*) é chamado de renderização (*rendering*).

Em aplicativos que devem ser executados em tempo real, é importante que o algoritmo de renderização seja capaz de mostrar a maior quantidade possível de detalhes no menor tempo possível, de forma a manter uma taxa de atualização que permita movimentos fluídos - pelo menos 30 quadros por segundo, embora resultados muito melhores sejam obtidos a partir de 60 quadros por segundo. A principal preocupação é portanto com a velocidade na qual os gráficos são exibidos, mantendo o foto-realismo em segundo plano.

Tradicionalmente, jogos e simuladores que apresentam gráficos tridimensionais costumam apresentar estes gráficos como conjuntos de triângulos definidos em tempo de design. Estes polígonos são armazenados em alguma estrutura de dados e, durante a execução do jogo, são acessados e mostrados ao usuário através de um processo chamado rasterização (*rasterisation*), onde os vetores que representam os vértices são multiplicados por matrizes de transformação 4x4 e então transformados em pixels.

Com o advento de máquinas mais potentes e componentes de *hardware* especializados em processamento de gráficos tridimensionais, o aproveitamento desta capacidade extra, de acordo com esta abordagem, se daria pelo aumento do número de polígonos. O incremento do número de polígonos, de acordo com Kruger (1999), implica em alguns problemas:

- a) Com mais polígonos, o espaço em disco necessário para armazená-los aumenta, assim

como o tempo necessário para acessá-los;

- b) os gráficos não podem ser melhorados em tempo real. Parte-se de um desenho ótimo e, se necessário, são feitos ajustes para melhorar seu desempenho em máquinas mais lentas. Não é possível, porém, melhorar o desenho para aumentar a qualidade dos gráficos em computadores mais poderosos;
- c) incapacidade de modelar naturalmente formas curvas: a única maneira de obter um formato aproximado ao de uma curva é através do uso de um grande número de polígonos.

Uma das maneiras de minimizar estes problemas é utilizar *patches* de Bézier, que são obtidos utilizando-se uma curva de Bézier para gerar os pontos de controle de uma segunda curva de Bézier. Com isso, tem-se uma equação de curva com duas variáveis paramétricas. O número de pontos de controle dependerá do tipo de *patch* - 9 para biquadrático, 16 para bicúbico etc.

Outra vantagem de se utilizar *patches* de Bézier ao invés de malhas triangulares manuais é a geração de vetores normais, necessários para a iluminação da superfície. Tradicionalmente, a intensidade luminosa é computada calculando-se o produto escalar entre o vetor raio de luz e o vetor normal à superfície no ponto. Em malhas tradicionais, os vetores normais devem ser ou “adivinhados” baseados nos vértices ao redor do ponto de interesse, ou manualmente definidos pelo artista. Com os *patches* de Bézier, basta derivar a equação paramétrica da superfície para se obter a equação paramétrica das normais, permitindo o cálculo rápido e preciso do vetor normal em qualquer ponto da superfície.

1.1 Objetivos

O objetivo geral é criar um algoritmo que, a partir de um conjunto de pontos de controle, retorne um conjunto de triângulos que represente o *patch* de Bézier correspondente. Tendo em vista este objetivo geral, foram definidos os seguintes objetivos específicos:

- a) Revisar a literatura existente sobre renderização e malhas poligonais geradas a partir de curvas de Bézier;
- b) Apresentar a fundamentação teórica das curvas de Bézier, incluindo suas características e representação matemática;
- c) Com base nas informações levantadas nas etapas anteriores, elaborar o algoritmo desejado.

2 Curvas de Bézier

Através dos séculos, uma régua e um par de compassos permaneceram como os únicos dispositivos práticos para a abstração de figuras geométricas do mundo real para desenhos e modelos. Consequentemente, linhas retas e arcos circulares eram os únicos objetos geométricos que podiam ser fiel e confiavelmente reproduzidos em papel. Linhas mais complexas precisavam de árduos cálculos (em desenhos matemáticos) ou eram simplesmente desenhados à mão (em artes visuais ou design).

No século XX, contudo, essa deficiência provou ser um obstáculo real, especialmente no desenho industrial onde linhas e superfícies curvilíneas complexas devem ser definidas com precisão, de modo a ser uniformemente reproduzidas em plástico ou metal. Quando você pensa em curvilinearidade em tecnologia, provavelmente uma das primeiras coisas que vêm à mente são os contornos suaves dos automóveis modernos. De fato, foi para a Renault, empresa francesa de construção de carros, que Pierre Bézier desenvolveu, nos anos 60, uma nova ferramenta de desenho baseada em uma grande variedade de versáteis curvas matemáticas.

Essas curvas, chamadas (em homenagem a seu inventor) de curvas de Bézier, são agora familiares para qualquer usuário de programas de desenho vetorial. A mais comumente usada é a curva de Bézier de terceira ordem, que é totalmente definida por quatro pontos: dois pontos finais (*endpoints*) e dois pontos de controle (*control points*) que não ficam na curva em si, mas definem sua forma.

Essa simples construção é surpreendentemente versátil. Uma curva de Bézier pode ser lisa, mas pode ter curvas acentuadas e formar laços. É também capaz de aproximar-se de outros tipos de curva. Por exemplo, se não é possível desenhar um círculo absolutamente exato com curvas de Bézier, é possível aproximá-las de um quarto de círculo (isto é, um arco de 90°) com um erro menor que 0,06%. Que é certamente insignificante na maioria dos casos práticos (KIRSANOV, 1999).

Já que uma curva possui infinitos pontos, não é possível calcular a posição de todos para exibí-los na tela de um computador. De fato, mesmo aproximando-se por pixels, o tempo necessário seria alto demais, e o benefício pequeno demais. Portanto, aproxima-se a curva por segmentos de reta, que são fáceis de rasterizar, tomando-se pontos arbitrários ao longo da mesma e conectando-os por segmentos de reta. A proximidade entre a aproximação e a curva real é inversamente proporcional à distância entre estes pontos, de modo que pode-se escolher entre precisão matemática e velocidade de processamento.

2.1 Curva linear de Bézier

Dados dois pontos P_1 e P_2 , uma curva linear de Bézier é apenas uma linha reta entre esses dois pontos. A curva é dada por (WIKIPEDIA, 2007a):

$$B(t) = P_1 + (P_2 - P_1)t = (1 - t)P_1 + tP_2, \quad t \in [0, 1] \quad (1)$$

Esta “curva”, que na realidade é uma reta, é equivalente à interpolação linear.

2.2 Curva quadrática de Bézier

Uma curva quadrática de Bézier é caminho traçado pela função $B(t)$, dados os pontos P_1 , P_2 e P_3 :

$$B(t) = (1 - t)^2P_1 + 2t(1 - t)P_2 + t^2P_3, \quad t \in [0, 1] \quad (2)$$

2.3 Curva cúbica de Bézier

Quatro pontos P_1 , P_2 , P_3 e P_4 no plano ou no espaço tridimensional definem uma curva cúbica de Bézier. A curva começa em P_1 indo em direção a P_2 e chega a P_4 vindo da direção de P_3 . Normalmente, ela não passa através de P_2 ou P_3 ; esse pontos só estão lá para definir a direção. A distância entre P_1 e P_2 determina “quão longe” a curva se move em direção a P_3 antes de virar em direção a P_4 .

Na figura 1, tem-se um exemplo de curva de Bézier de cúbica (ou de terceira ordem). Suas extremidades são P_1 e P_4 , e seus pontos de controle são P_2 e P_3 . A curva em si é definida pela seguinte equação paramétrica (WIKIPEDIA, 2007a):

$$P(t) = P_1(1 - t)^3 + 3P_2(1 - t)^2t + 3P_3(1 - t)t^2 + P_4t^3, \quad t \in [0, 1] \quad (3)$$

Esta fórmula também pode ser escrita como:

$$P(t) = \sum_{i=0}^3 \binom{3}{i} P_{i+1} (1 - t)^{3-i} t^i \quad (4)$$

A curva de Bézier de grau n pode ser generalizada da seguinte forma. Dados os pontos P_0 , P_1 , ..., P_n , a curva de Bézier é:

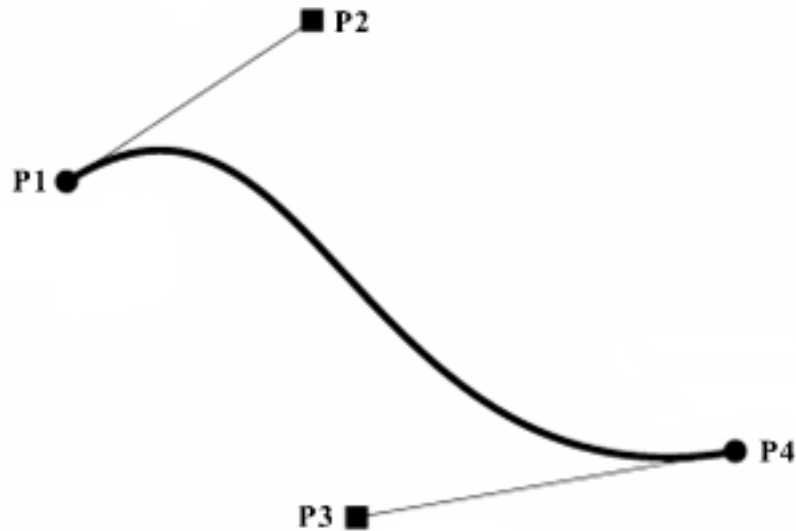


Figura 1: Curva de Bézier de terceira ordem. Adaptado de: <http://www.webreference.com/dlab/9902/bezier.html>

$$B(t) = \sum_{i=0}^n \binom{n}{i} P_i (1-t)^{n-i} t^i = P_0 (1-t)^n + \binom{n}{1} P_1 (1-t)^{n-1} t + \dots + P_n t^n, \quad t \in [0, 1] \quad (5)$$

Esta fórmula pode ser expressa recursivamente, como segue: deixe $B_{P_0 P_1 \dots P_n}$ denotar a curva de Bézier determinada pelos pontos P_0, P_1, \dots, P_n . Então:

$$B(t) = B_{P_0 P_1 \dots P_n} t = (1-t) B_{P_0 P_1 \dots P_{n-1}} t + t B_{P_1 P_2 \dots P_n} t \quad (6)$$

Traduzindo em palavras, a curva de Bézier de grau n é a interpolação linear entre duas curvas de Bézier de grau $n-1$.

2.4 Superfícies de Bézier

Superfícies de Bézier são a extensão bidimensional das curvas de Bézier. Enquanto que as curvas são equações paramétricas de uma coordenada, as superfícies possuem duas coordenadas (tipicamente $u, v \in [0, 1]$). Estas curvas nada mais são do que interpolações entre várias curvas de Bézier, de forma que as curvas determinam suas extremidades, e os pontos no “interior” da curva são calculados fazendo-se uma média ponderada dos pontos de extremidade.

Da mesma forma em que não é praticável calcular todos os pontos de uma curva, a superfície deve ser aproximada por figuras geométricas mais simples. Tipicamente, triângulos são utilizados, por possuírem propriedades geométricas vantajosas para processamento de imagens tridimensionais. Estes triângulos são gerados calculando-se vértices ao longo da superfície, em intervalos arbitrários, de forma que eles podem aproximar a curva real a qualquer margem de erro finita e não-nula.

2.5 Aplicações

Em um jogo, as aplicações das curvas de Bézier não se limitam à modelagem de objetos e superfícies. Elas também podem ser utilizadas em conjunto com técnicas de inteligência artificial para permitir que uma entidade controlada pelo computador se movimente pelo cenário. Basta estabelecer o ponto de origem e o destino como extremidades do caminho da entidade, e então definir dois pontos de controle. Alterando-se estes pontos de controle de acordo com algum algoritmo específico, pode-se obter um trajeto curvo, mais suave e realista (HUI; PRAKASH; CHAUDHARI, 2004).

Em tipografia, as fontes *TrueType* possuem caracteres formados por linhas retas e por curvas de Bézier quadráticas (WIKIPEDIA, 2007b). Sendo quadrática, cada curva possui um ponto de controle. A letra S, por exemplo, pode ser formada por duas curvas quadráticas. Por isso, é possível redimensionar uma fonte sem perder a qualidade da imagem; se as fontes fossem desenhadas *pixel* por *pixel* e depois tivessem seu tamanho aumentado, acabariam ficando “quadriculadas”, como quando se visualiza uma foto usando um *zoom* excessivo. Como as curvas de Bézier possuem uma formulação matemática precisa, elas podem ser redimensionadas sem perda de informação.

As aplicações destas curvas não se limitam à Ciência da Computação. O *design*, de maneira geral, beneficiou-se com o desenvolvimento de curvas paramétricas; na verdade, Pierre Bézier desenvolveu sua técnica por uma solicitação da Renault, uma montadora francesa. Atualmente, curvas suaves são muito comuns no *design* de automóveis. Além disso, é possível encontrar logomarcas de empresas que utilizam-nas (figura 2) (KIRSANOV, 1999).

3 Algoritmo proposto

A seguir, propomos um simples algoritmo capaz de, dados 16 pontos de controle (para as 4 curvas cúbicas localizadas ao longo das laterais), construir uma malha triangular com



Figura 2: Logomarca desenhada com o auxílio de curvas de Bézier. Fonte: <http://www.webreference.com/dlab/9902/bezier.htm>

aproximação arbitrária à superfície bicúbica. O algoritmo possui complexidade $O(n^2)$, que é a melhor possível, neste caso.

1. O usuário (ou rotina superior) determina o número n de subdivisões lineares que serão utilizadas. Este passo é necessário, pois precisamos gerar um número finito de triângulos, que será dado por $2n^2$.
2. Para cada vértice da malha, suas coordenadas em termo do quadrado unitário são determinadas, isto é, as coordenadas são normalizadas para $[0, 1]$. Existem várias técnicas para se determinar a melhor maneira de posicionar os pontos, dependendo do grau de curvatura ao longo da superfície, mas isso está além do escopo deste trabalho. Para nossos propósitos, servirá fazer uma divisão uniforme da superfície. Isto pode ser feito seguinte fragmento de código em C++:

```
for (int i=0;i<n;i++) {  
    for (int j=0;j<n;j++) {  
        float u = (float)i/(n-1);  
        float v = (float)j/(n-1);  
  
        // Determinar coordenadas aqui  
    }  
}
```

}

Note que, uma vez que i e j variam de 0 a n , u e v irão variar no intervalo $[0, 1]$, de modo que o ponto (u, v) esteja contido em um quadrado de vértices $(0, 0)$, $(0, 1)$, $(1, 1)$ e $(1, 0)$.

3. Cada ponto da forma (u, v) é então convertido para um ponto no espaço tri-dimensional, segundo a seguinte fórmula (FARIN, 2002):

$$\mathbf{p}(u, v) = \sum_{i=0}^3 \sum_{j=0}^3 \binom{3}{i} u^i (1-u)^{3-i} \binom{3}{j} v^j (1-v)^{3-j} P_{i,j} \quad (7)$$

Onde $P_{i,j}$ são os pontos de controle. Note que a fórmula acima é uma simples interpolação bilinear entre as quatro curvas, onde o peso é a proximidade com cada uma delas.

4. Por fim, pode-se calcular o vetor normal à superfície em qualquer um dos vértices, o que é necessário para a computação de iluminação, que requer um produto escalar entre o vetor normal no ponto e o vetor raio de luz. Para se obter este vetor, utiliza-se a derivada parcial da equação acima em relação a u , e, em seguida, em relação a v . Isto nos dá dois vetores tangentes à superfície, T_u e T_v :

$$T_u = \sum_{i=0}^3 \sum_{j=0}^3 \binom{3}{i} \frac{\partial}{\partial u} u^i (1-u)^{3-i} \binom{3}{j} v^j (1-v)^{3-j} P_{i,j} \quad (8)$$

$$T_v = \sum_{i=0}^3 \sum_{j=0}^3 \binom{3}{i} u^i (1-u)^{3-i} \binom{3}{j} \frac{\partial}{\partial v} v^j (1-v)^{3-j} P_{i,j} \quad (9)$$

5. Dados os vetores T_u e T_v , pode-se calcular o vetor N_{uv} , unitário e normal à superfície, com a simples equação:

$$N_{uv} = \frac{T_u \times T_v}{\|T_u\| \|T_v\|} \quad (10)$$

6. O seguinte fragmento de código é portanto uma versão simples do algoritmo, utilizando OpenGL. Note que o cálculo de vetores normais foi omitido por brevidade.

```
Ponto3D p[4][4]; // Pontos de controle, supostamente preenchidos an
const int n = 50;
Ponto3D v[n][n];

// Calcula todos os pontos
for (int a=0; a<n; a++) {
```

```

for (int b=0;b<n;b++) {
    // Calcula u e v
    float u = (float)a/(n-1);
    float v = (float)b/(n-1);

    // Determina as coordenadas do ponto
    v[a][b] = 0;
    for (int i=0;i<4;i++) {
        for (int j=0;j<4;j++) {
            v[a][b] = binom(3,i) * pow(u,i) * pow(1-u,3-i) * binom(3,j)
        }
    }
}

// Envia os pontos para o OpenGL
glBegin(GL_TRIANGLES);
for (int i=0;i<n-1;i++) {
    for (int j=0;j<n-1;j++) {
        // Envia dois triângulos para formar um quadrilátero
        // Triângulo 1
        glVertex3f(v[i][j].x,v[i][j].y,v[i][j].z);
        glVertex3f(v[i+1][j].x,v[i+1][j].y,v[i+1][j].z);
        glVertex3f(v[i][j+1].x,v[i][j+1].y,v[i][j+1].z);

        // Triângulo 2
        glVertex3f(v[i+1][j].x,v[i+1][j].y,v[i+1][j].z);
        glVertex3f(v[i][j+1].x,v[i][j+1].y,v[i][j+1].z);
        glVertex3f(v[i+1][j+1].x,v[i+1][j+1].y,v[i+1][j+1].z);
    }
}
glEnd();

```

7. Este algoritmo admite mudanças em tempo real nos pontos de controle. Caso isso não seja necessário, o algoritmo pode pré-calculá-la toda a malha triangular de antemão, evitando a

necessidade de se re-calcular todas as coordenadas a cada atualização da tela, o que pode se tornar caro demais para algumas aplicações.

4 Considerações finais

Na introdução deste trabalho, foram estabelecidos três objetivos: revisar a literatura sobre renderização a partir de curvas de Bézier, estabelecer um referencial teórico em relação à modelagem matemática destas curvas, e por fim desenvolver um algoritmo capaz de entender esta modelagem matemática e transformá-la em um conjunto de triângulos, que são então enviados para o OpenGL (que é uma biblioteca específica para gráficos tridimensionais).

Foi possível perceber, na revisão da literatura, que as curvas de Bézier estão presentes em várias áreas da Computação e da indústria de jogos. Não servem apenas para criar gráficos, mas também para auxiliar a inteligência artificial, por exemplo. São utilizadas também no design, e não é difícil imaginar as possibilidades criadas pela redução de uma curva a uma simples equação matemática cujos parâmetros são alguns poucos pontos.

Finalmente, apresentamos um simples algoritmo capaz de utilizar equações paramétricas bicúbicas de Bézier para criar superfícies arbitrárias e exibí-las em três dimensões utilizando a biblioteca gráfica OpenGL. Este algoritmo prova que, apesar de poderosas, estas superfícies são bastante simples de se implementar e possuem muitas propriedades matemáticas que as tornam interessantes em várias áreas.

Existem muitas áreas nas quais este trabalho poderia ser aprofundado, como estudos matemáticos de propriedades destas superfícies, assim como o aprimoramento do algoritmo, para ser mais rápido, mais preciso ou mais versátil. Assim, este trabalho é apenas uma porta de entrada para o vasto mundo das curvas e superfícies de Bézier.

Referências Bibliográficas

FARIN, G. *Curves and Surfaces for CAGD*. 5. ed. [S.l.]: Academic Press, 2002.

FOLEY, J. D. et al. *Computer graphics: principles and practice*. Reading: Addison-Wesley, 1990.

Game AI: artificial intelligence for 3D path finding.

KIRSANOV, D. Nonlinear design. fev. 1999.

KRUGER, G. Curved surfaces using bézier patches. *Gamasutra*, jun. 1999.

WIKIPEDIA. *Bézier Curve*. 2007. Acesso em: <20 nov. 2007>. Disponível em: <http://en.wikipedia.org/wiki/Bezier_Curve>.

WIKIPEDIA. *TrueType*. 2007. Acesso em: <20 nov. 2007>. Disponível em: <<http://en.wikipedia.org/wiki/TrueType>>.