Trabalho Prático
Computação Gráfica - Curva Bezier

Guilherme Afonso Elifas Gibosky[1]

[1]Instituto de Ciências Exatas e Informática –
Pontifícia Universidade Católica de Minas Gerais (PUCMG)
Belo Horizonte – MG – Brazil

guilherme.gibosky@gmail.com

1. Introdução

A curva de Bézier é uma curva definida matematicamente usada em aplicações gráficas que é definida por quatro pontos: a posição inicial e a posição de final, que são as bases, P0 e P3 e dois pontos intermediários separados, isto é, P1 e P2. Devido a combinação de ”simplicidade” e eficiência, as curvas de Bézier são frequentemente usadas em computação gráfica, animação, modelagem etc.

Este trabalho se divide em: 1-Introdução, 2- Estrutura do Código, 3- Definição Matemática, 4- Conclusão, Bibliografia e Referencias

2. Estrutura do Código

2.1. Linguagem

A implementação escolhida foi para a linguagem C, devido a facilidade com a mesma, além de ser mais fácil visualizar a estrutura, diferente de linguagens mais fechadas como Java.

Mas devido a simplicidade, foi necessário usar um pacote (lSDL2) para acompanhar o *mouse* do usuário, assim como plotar na em uma tela dedicada os pontos e a variação da curva

2.2. Utilização

Para utilizar a implementação, é necessário instalar o pacote lSDL2. No ambiente *UNIX* a instalação se dá através do comando:

sudo apt-get install libsdl2-dev -f

Em distribuições que não usam apt, basta colocar o nome do pacote, libsdl2-dev, no campo do pacote do gerenciador de dependência.

No ambiente Windows, a inclusão de pacotes depende da IDE utilizada.

Pode acontecer de, dependendo da distribuição Linux, de dependências estarem incompletas, nesse caso a sequencia de comandos abaixo pode resolver o problema:

- sudo apt-get dist-upgrade -f
- sudo apt-get install -f
- sudo apt-get install libsdl2-dev -f

A implementação inicia o programa preparando a tela aonde vai ser mostrado os pontos e a curva, de modo que ele define a cor do fundo e as cores e formatos dos elementos.

Quando a tela está pronta o usuário tem a possibilidade de escolher 4 pontos, através do *click* do *mouse*, para que o algoritmo produza a curva. No 4º *click*, o programa entra no modo ”acompanhamento”, que recalcula a curva, quando o usuário muda a localização do último ponto selecionado, através do movimento do *mouse*.

Caso o usuário de um *click* novamente na tela no modo ”acompanhamento”, o programa reinicia a escolha dos pontos para o calculo da curva.

2.3. Organização

As principais funções são as listadas abaixo:

drawCircle())

Função que desenha os círculos no *canvas*.

circleBres())

Função que usa Bresenham para desenhar os círculos que funcionam como os pontos de controle da aplicação. Chama a função *drawCircle()*;

bezierCurve())

Função principal da aplicação. Responsável por calcular a curva de Bézier utilizando os pontos de controle passados pelo usuário. É um implementação da função matemática, descrita em .

main())

Chamada de inicio da aplicação e de controle do comportamento do usuário. Nela que está a contagem dos pontos escolhidos, a reinicialização das distribuição dos pontos de controle e o modo ”acompanhamento”.

3. Definição Matemática

A curva simplesmente baseia seu cálculo no Binômio de Newton para a resolução de seus coeficientes e é resolvida facilmente através de:

$$\mathrm{P(u)} = \sum_{i=0}^{n} P_i B_i^n(u)$$

Vamos definir nossa curva cúbica de Bézier matematicamente. Então um id de curva Bézier definido por um conjunto de pontos de controle P0 a Pn onde n é chamado de ordem (n = 1 para linear, n = 2 para quadrático, etc.). O primeiro e último pontos de controle são sempre os pontos finais da curva; no entanto, os pontos de controle intermediários (se houver) geralmente não estão na curva. Para a ordem da curva cúbica de Bézier (n) do polinômio é 3, o índice (i) varia de i = 0 para i = n, isto é, 3 e u varia entre 0 e 1.

A função que define a curva de Bézier é:

$$P(u) = P_0 B_0^3(u) + P_1 B_1^3(u) + P_2 B_2^3(u) + P_3 B_3^3(u)$$

Assim, expandindo essa definição e a distribuição binomial, chegasse a seguinte formula:

$$P(u) = (1-u)^3 P_0 + 3u^1(1-u)^2 P_1 + 3(1-u)^1 u^2 P_2 + u^3 P_3$$

$$P(u) = \{x(u), y(u)\}$$

Assim o calculo pode ser parametrizado nas equações abaixo:

$$x(u) = (1-u)^3 x_0 + 3u^1(1-u)^2 x_1 + 3(1-u)^1 u^2 x_2 + u^3 x_3$$

$$y(u) = (1-u)^3 y_0 + 3u^1(1-u)^2 y_1 + 3(1-u)^1 u^2 y_2 + u^3 y_3$$

O fragmento de código abaixo, mostra a implementação da equação parametrizada:

```
void bezierCurve(int x[] , int y[])
{
    double xu = 0.0 , yu = 0.0 , u = 0.0 ;
    int i = 0 ;
    for(u = 0.0 ; u <= 1.0 ; u += 0.0001)
    {
        xu = pow(1-u,3)*x[0]+3*u*pow(1-u,2)*x[1]+3*pow(u,2)*(1-u
            )*x[2]
             +pow(u,3)*x[3];
        yu = pow(1-u,3)*y[0]+3*u*pow(1-u,2)*y[1]+3*pow(u,2)*(1-u
            )*y[2]
             +pow(u,3)*y[3];
        SDL_RenderDrawPoint(renderer , (int)xu , (int)yu) ;
    }
}
```

4. Conclusão

O trabalho pratico foi interessante pois permitiu o aprendizado dinâmico, pois ao visualizar como a curva muda de acordo com a mudança do controle é possível incorporar a parte teórica matemática e juntamente com a pesquisa feita para o trabalho, como ela é utilizada e por que essa técnica é amplamente utilizada em computação gráfica.

References

https://pt.khanacademy.org/partner-content/pixar/animate/ball/v/animation3. Acesso: 18 nov. 2018.

https://pt.wikipedia.org/wiki/curva_de_b Acesso: 18 nov. 2018.

http://www.mat.ufmg.br/ rodney/notas_de_aula/bezier.pdf. Acesso: 10 nov. 2018.