Curvas Paramétricas - Implementação da Curva de Bézier

João Paulo de Castro Bento Pereira[1]

[1]Aluno de Ciência da Computação, ICEI, PUC Minas
[1]jpcbpereira@sga.pucminas.br

Resumo

Na disciplina de Computação Gráfica, foi lecionado sobre as curvas paramétricas. Neste documento, a curva paramétrica utilizada para implementação e exemplificação é a Curva de Bézier, na qual utiliza de pontos de controle para a interpolação dos pontos intermediários para a curva. A implementação foi realizada na linguagem Python, com uma interface gráfica utilizando a biblioteca Tkinter. O intuito deste trabalho é gerar conhecimento sobre o tema, e também demonstrar a aplicação para desenho de curvas de Bézier.

Introdução

As Curvas de Bézier são curvas paramétricas no plano ou no espaço, são invariantes a transformações, não oscila mais que seu polígono de controle, estando sempre contida dentro do fecho convexo deste polígono.[1] As curvas de Bézier foram desenvolvidas independentemente, ou seja, criada por mais de uma pessoa sem contribuição entre elas. Seus criadores foram, Paul de Casteljau (1959), e Pierre Etienne Bézier(1962). Casteljau trabalhava para Citroën, enquanto Bézier para Renault, e o grande motivador foi para auxiliar no design e fabricação assistida por computador. A curva, embora ter sido desenvolvida primeiramente por Casteljau, ficou com o nome de Pierre devido protocolos de informações da Citroën que decidiu manter como segredo industrial a pesquisa realizada por Casteljau, com isso a Curva ficou mais famosa por Pierre Bézier. Hoje a Curva de Bézier é amplamente implementada em diversas áreas da computação gráfica para modelagem de curvas suaves, muito utilizadas em animações, design de interface e produção de fontes.[2]

Abordagem

Para a implementação das curvas de Bézier, sendo Bézier Linear, Bézier Quadrática e Bézier Cúbica, foi notado que a curva realiza uma média ponderada, ou até mesmo um balanceamento entre os pontos(ou retas) e assim os pesos de cada ponto é definido de forma que quanto mais um deles pesa no resultado, menos o outro influencia.[3] Bézier Linear (entre dois pontos) é calculado pelo ponderamento de $P0$ e $P1$, Bézier Quadrática, entre 3 pontos, sendo eles formando duas Retas $R1$($P0$ e $P1$) e $R2$($P1$ e $P2$) é obtida agora pela ponderação das retas, e consequentemente Bézier Cúbica gerando analogicamente duas curvas $C1$($P0$, $P1$, $P2$) e $C2$($P1$, $P2$, $P3$). O Balanceamento para cada ponto é feito pela variável T, sendo o Peso de $P0$ = 1 - T, e o peso de $P1 = T$. Mais detalhes a seguir.

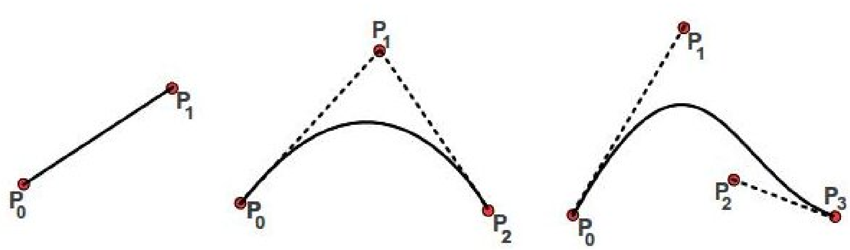

Figura 1. Curvas de Bézier (Linear, Quadrática e Cubica)

Bézier Linear

Bézier Linear, é a implementação das curvas de Bézier que formam uma reta, pois é composto por apenas dois pontos, e, na medida em que um é desbalanceado, o outro balanceia a seguinte equação:

$$P(t) = (1-t)*P0 + t*P1 \tag{1}$$

Para $t \in [0,1]$, e $P0$ e $P1$ os pontos de controle.

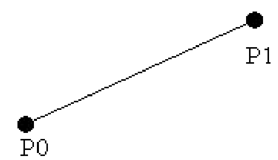

Figura 2. Bézier Linear[3]

Bézier Quadrática

Como dito, a curva de Bézier quadrática, é feita pela interpolação e balanceamento entre os pontos de controle, $P0$, $P1$ e $P2$. A abordagem é semelhante a Bézier Linear, mas neste caso, utiliza-se as Retas $R1$ e $R2$ para os cálculos, ficando assim:

$$R1 : (1-t)*P0 + t*P1$$
$$R2 : (1-t)*P1 + t*P2$$

Aplicando agora a mesma abordagem para gerar a curva $C1$ fica:

$$C1 : (1-t)*R1 + t*R2$$
$$C1(t) = (1-t)^2*P0 + 2*(1-t)*t*P1 + t^2*P2$$

Para $t \in [0,1]$, e $P0$, $P1$ e $P2$ os pontos de controle.

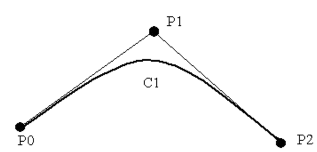

Figura 3. Bézier Quadrática[3]

Bézier Cúbica

Por fim, a Bézier Cúbica é composta por 4 pontos de controle, sendo $P0$, $P1$, $P2$ e $P3$. A obtenção da Curva de Bézier Cúbica segue o mesmo princípio, sendo $C1$ obtida por $P0$, $P1$ e $P2$, e $C2$ obtida por $P1$, $P2$ e $P3$. Consequentemente, $C3$ pode ser obtido por:

$$C3 = (1-t)*C1 + t*C2 \tag{2}$$

Desenvolvendo $C1$ e $C2$ nesta equação, obtemos a equação geral da Curva de Bézier Cúbica para 4 Pontos de Controle:

$$C3(t) = (1-t)^3*P0 + 3*t*(1-t)^2*P1 + 3*t^2*(1-t)*P2 + t^3 + *P3 \tag{3}$$

Para $t \in [0,1]$, e $P0$, $P1$, $P2$ e $P3$ os pontos de controle.

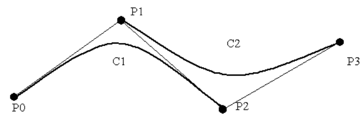

Figura 4. Bézier Cúbica.[3]

Implementação

A implementação das curvas, foi realizada utilizando a linguagem de programação Python(3.6.7). A implementação consiste de uma interface gráfica, construída pela biblioteca Tkinter (https://docs.python.org/3/library/tkinter.html), e para gerar uma quantidade arbitrária de valores entre 0 e 1, que são utilizados como valores de t nas equações, foi utilizado a biblioteca NumPy (http://www.numpy.org/). Ambas bibliotecas não necessitam de uma instalação extra, sendo assim apenas necessário a instalação da própria linguagem.

Tkinter

A interface gráfica é formada por 4 Botões, sendo eles 'Bézier Linear', 'Bézier Quadrática', 'Bézier Cúbica' e por fim 'Limpar'. Cada botão é uma das funcionalidades diferentes da implementação, sendo responsável por permitir o usuário testar cada algoritmo de bézier implementado. A seguir está o código responsável pela configuração de todo o ambiente gráfico:

```
# Classe Paint principal para lidar com a janela de interacao com o usuario
class Paint(object):
    def __init__(self):
        self.root = Tk()
        self.bezier_linear_button = Button(self.root, text='Bezier Linear',
        command=self.use_bezier_linear)
        self.bezier_linear_button.grid(row = 0, column = 0)

        self.bezier_quadratic_button = Button(self.root, text='Bezier Quadratica',
        command=self.use_bezier_quadratic)
        self.bezier_quadratic_button.grid(row = 0, column = 1)

        self.bezier_cubic_button = Button(self.root, text='Bezier Cubica',
        command=self.use_bezier_cubic)
        self.bezier_cubic_button.grid(row = 0, column = 2)

        self.clear_button = Button(self.root, text='Limpar', command=self.limpar)
        self.clear_button.grid(row = 0, column = 3)

        self.stop_button = Button(self.root, text="Stop")
        #definir canvas (area de desenho do usuario)
        self.c = Canvas(self.root, bg='white', width=800, heigh=600)
        self.c.grid(row=1,columnspan=4)
        self.root.title("GC - Curvas de Bezier - 562874 - Joao Castro")

        self.setup()

        self.root.mainloop()
    #Inicializando os elementos do programa como x e y, espessura da linha desenhada
    e outras informacoes
    def setup(self):
        self.x = None
        self.y = None
        self.line_width = 5
        self.color = 'black'
```

```
        self.active_button = self.bezier_linear_button
        #quando o mouse e' clicado, ele chama o metodo paint para pegar o x e y do local
        self.c.bind('<Button-1>', self.paint)
        #quando o mouse e liberado, ele limpa o x e y para que um novo possa ser atribuido
        self.c.bind('<ButtonRelease-1>', self.reset)
    # metodo para o botao limpar, que zera a lista de pontos
    e limpa todos os elementos do canvas
    def limpar(self):
        self.c.delete("all")
        global lista_pontos
        lista_pontos = []
    #Funcoes use_* servem para ativar e desativar os botoes do menu, deixando os afundados
    quando clicados e os demais normais.
    def use_bezier_linear(self):
        self.activate_button(self.bezier_linear_button)

    def use_bezier_quadratic(self):
        self.activate_button(self.bezier_quadratic_button)

    def use_bezier_cubic(self):
        self.activate_button(self.bezier_cubic_button)
    #Funcao responsavel por deixar o botao como sunken ou raised, ou seja, ativo ou nao.
    def activate_button(self, some_button):
        self.active_button.config(relief=RAISED)
        some_button.config(relief=SUNKEN)
        self.active_button = some_button
    #metodo chamado para mostrar os pontos clicados pelo usuario
    e chamar os metodos de bezier
    def paint(self, event):
        global lista_pontos
        self.x = event.x
        self.y = event.y
        if(self.active_button != self.stop_button):
            #Colocando o novo ponto clicado como ponto de controle de bezier
            lista_pontos.append((self.x,self.y))
            self.c.create_oval(self.x-1, self.y-1, self.x+1, self.y+1, width=5,
            fill= 'blue', outline='blue') #Plotando uma bolinha para o ponto de controle
            coordinates = "(%s,%s)" %(self.x, self.y)
            self.c.create_text(self.x, self.y-15, text = coordinates,
            fill='blue' ) #texto de coordenadas x,y do ponto clicado
        #caso a lista ja possua 3 pontos, chama o metodo bezier,
        e bloqueia os cliques futuros e assim por diante..
        if len(lista_pontos) == 3 and self.active_button == self.bezier_quadratic_button:
            self.activate_button(self.stop_button)
            self.bezier()
        elif len(lista_pontos) == 4 and self.active_button == self.bezier_cubic_button:
            self.activate_button(self.stop_button)
            self.bezier()
        elif len(lista_pontos) == 2 and self.active_button == self.bezier_linear_button:
            self.activate_button(self.stop_button)
            self.bezier()
        else: pass
```

Método Bézier

Para calcular as diferentes formas de bézier, o algoritmo verifica o número de pontos existentes na lista de pontos, e se determinado botão está acionado, com isso ele utiliza da biblioteca NumPy para gerar 1000 valores diferentes entre 0 e 1 para representar t, e chama cada um dos métodos de bézier para gerar a curva.

```
#Metodo de chamada para o calculo de bezier,
 def bezier(self):
     global lista_pontos
     if len(lista_pontos) == 2: #bezier linear
         pontos = np.linspace(0,1, num=1000)
         for t in pontos:
             x, y = self.linear_bezier(lista_pontos[0], lista_pontos[1], t)
             self.c.create_oval(x-1, y-1, x+1, y+1, width=0, fill= 'red', outline='red')
         lista_pontos = []
     elif len(lista_pontos) == 3:#bezier quadratico
         pontos = np.linspace(0,1, num=1000)
         for t in pontos:
             x, y = self.quadratic_bezier(lista_pontos[0], lista_pontos[1],
             lista_pontos[2], t)
             self.c.create_oval(x-1, y-1, x+1, y+1, width=0, fill= 'red', outline='red')
         lista_pontos = []
     else:
         pontos = np.linspace(0,1, num=1000) #bezier cubico
         for t in pontos:
             x, y = self.cubic_bezier(lista_pontos[0], lista_pontos[1],
             lista_pontos[2],
             lista_pontos[3], t)
             self.c.create_oval(x-1, y-1, x+1, y+1, width=0, fill= 'red', outline='red')
         lista_pontos = []


 #Metodo para calcular a curva de bezier linear usando a seguinte formula geral
 # P(t) = (1 - t)P0 + tP1
 def linear_bezier(self, p0, p1, t):
     xp0, yp0 = p0
     xp1, yp1 = p1
     xp0 = (1-t) * xp0
     yp0 = (1-t) * yp0
     xp1 = t * xp1
     yp1 = t * yp1
     return (xp0 + xp1, yp0 + yp1)

 #Metodo para calcular a curva de bezier quadratica usando a segunte formula geral
 # C1(t) = (1 - t)^2*P0 + 2t(1 - t)P1 + t^2*P2
 def quadratic_bezier(self, p0, p1, p2, t):
     xp0, yp0 = p0
     xp1, yp1 = p1
     xp2, yp2 = p2
     a = (1-t)**2
     t2 = t**2
     xp0 = a * xp0
     yp0 = a * yp0
     xp1 = 2 * t * (1-t) * xp1
     yp1 = 2 * t * (1-t) * yp1
     xp2 = t2 * xp2
     yp2 = t2 * yp2
```

```python
        return (xp0 + xp1 + xp2, yp0 + yp1 + yp2)

    # Metodo pra calcular a curva de bezier cubica usando a seguinte formula geral
    # C3(t) = (1 - t)^3 * P0 + 3t(1 - t)^2*P1 + 3t^2*(1 - t)P2 + t^3*P3
    def cubic_bezier(self, p0, p1, p2, p3, t):
        xp0, yp0 = p0
        xp1, yp1 = p1
        xp2, yp2 = p2
        xp3, yp3 = p3
        a = (1-t)**3
        b = (1-t)**2
        t2 = t**2
        t3 = t**3
        xp0 = a * xp0
        yp0 = a * yp0
        xp1 = 3 * t * b * xp1
        yp1 = 3 * t * b * yp1
        xp2 = 3 * t2 * (1-t) * xp2
        yp2 = 3 * t2 * (1-t) * yp2
        xp3 = t3 * xp3
        yp3 = t3 * yp3
        return (xp0 + xp1 + xp2 + xp3, yp0 + yp1 + yp2 + yp3)
```

Resultados

Foi realizado testes para Bézier Linear, Bézier Quadrática e Bézier Cúbica. Cada um executado separadamente, e todos realizados juntos.

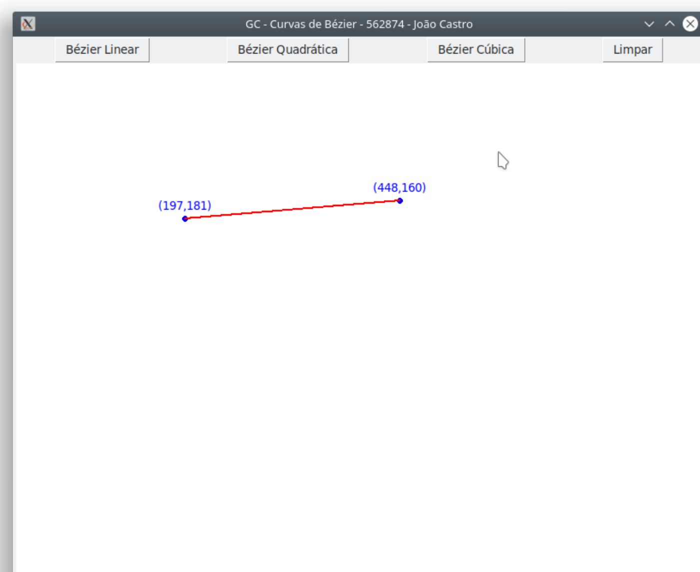

Figura 5. Teste-1: Bézier Linear.

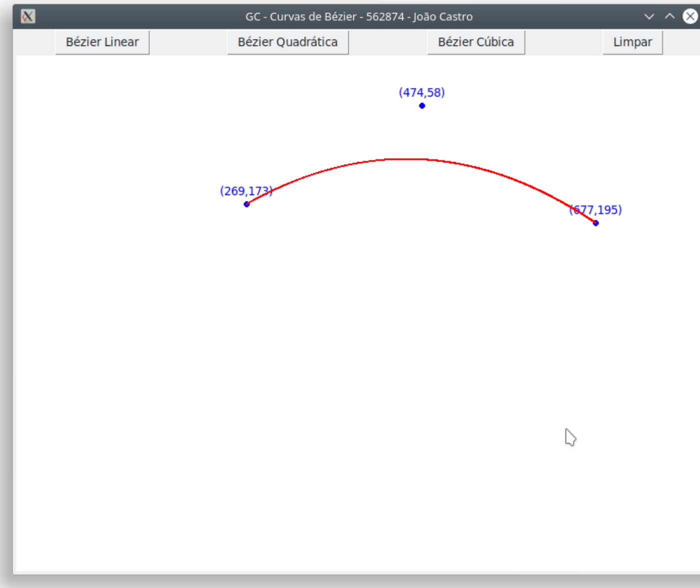

Figura 6. Teste-2: Bézier Quadrática.

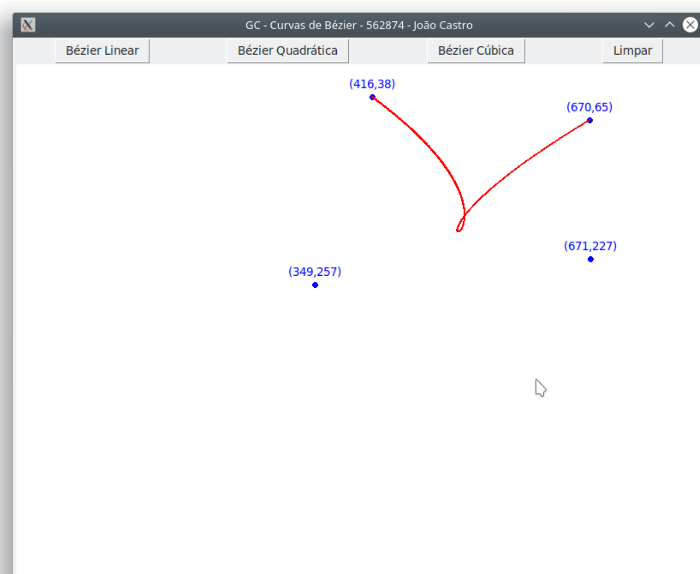

Figura 7. Teste-3: Bézier Cúbica.

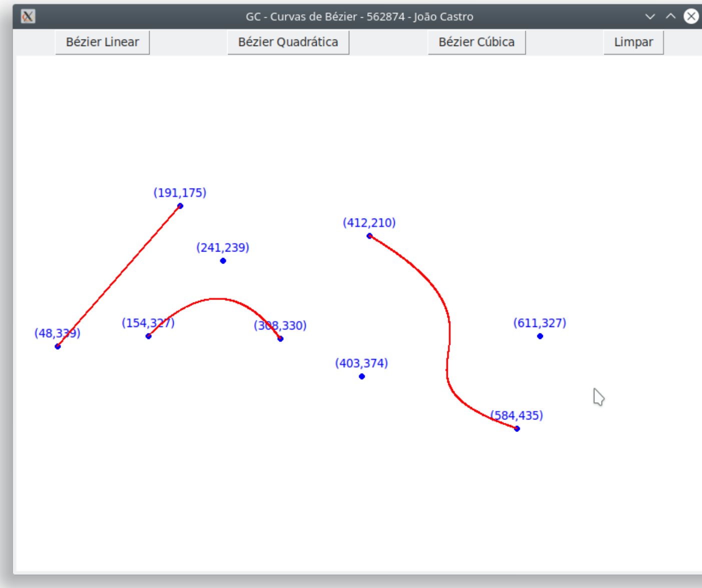

Figura 8. Teste-4: Todas as implementações.

Comentários Finais

Por fim, tendo em vista as imagens como resultados, e os comentários realizados acerca da abordagem proposta e implementada, pode-se dizer que são resultados satisfatórios, pois o aprendizado durante o processo foi suficiente e esclarecedor, e a implementação foi realizada com sucesso. As Curvas de Bézier auxiliam diversas aplicações gráficas no dia a dia, e se tornou uma das curvas paramétricas mais utilizadas. Este trabalho auxiliou na construção teórica e prática do conhecimento trabalhado na disciplina, e também na nova experiência de programação em uma linguagem antes não experimentada.

Referências

1. Biezuner, U. R. J. & de Jesus, B. F. R. Curvas de bézier. .
2. da Mota, R. R. Material didático - computação gráfica (2018).
3. Pinho, P. M. S. Computação gráfica - curvas paramétricas.