UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

CARRERA PROFESIONAL DE INGENIERIA INFORMATICA Y DE SISTEMAS

COMPUTACION GRAFICA 2

DOCENTE: M.SC. HECTOR E. UGARTE R.

HOJA DE EJERCICIOS 4: CURVAS Y CURVAS BEZIER

1. COMPETENCIAS

- El estudiante conoce el concepto de curva.
- El estudiante calcula y grafica curvas Bezier.

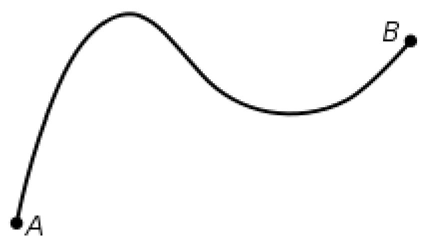
2. MARCO TEORICO

Curvas

La curva es una línea continua de una dimensión, que puede variar de dirección paulatinamente. Ejemplos sencillos de curvas cerradas simples son la elipse, la circunferencia o el óvalo, el cicloide; ejemplos de curvas abiertas, la parábola, la hipérbola y la catenaria y una infinidad de curvas estudiadas en la geometría analítica plana. La recta asume el caso límite de una circunferencia de radio de curvatura infinito y de curvatura 0.

Las curvas se pueden expresar de las formas:

1. En forma implícita… $F(x,y)=0$ Ejemplo $\sqrt{x^2+y^2}=e^{xy}$
2. En forma explícita… $y=f(x)$. Ejemplo: $y=\dfrac{3x^2+5x-7}{x^2}$ función racional.
3. En forma paramétrica ... $x=x(t), y=y(t)$ Ejemplo: $x=a\cdot lnt, y=\dfrac{a}{2}(2t+\dfrac{3}{t+1})$ paramétro : t.



Curvas Bezier

Se denomina curvas de Bézier a un sistema que se desarrolló hacia los años 1960 para el trazado de dibujos técnicos, en el diseño aeronáutico y en el de automóviles. Su denominación es en honor a Pierre Bézier, quien ideó un método de descripción matemática de las curvas que se comenzó a utilizar con éxito en los programas de CAD.

Los casos posibles serian:

Curvas lineales de Bézier

Dados los puntos P_0 y P_1, una curva lineal de Bézier es una línea recta entre los dos puntos. La curva viene dada por la expresión:

$$\mathbf{B}(t) = \mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)t = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \text{ , } t \in [0,1].$$

Curvas cuadráticas de Bézier

Una curva cuadrática de Bézier es el camino trazado por la función B(t), dados los puntos: P_0, P_1, y P_2,

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2 \text{ , } t \in [0,1].$$

Curvas cubicas

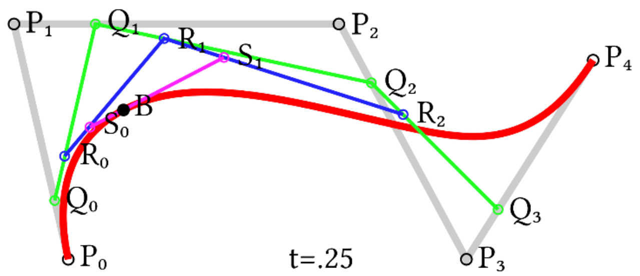
Cuatro puntos del plano o del espacio tridimensional, P_0, P_1, P_2 y P_3 definen una curva cúbica de Bézier. La curva comienza en el punto P_0 y se dirige hacia P_1 y llega a P_3 viniendo de la dirección del punto P_2. Usualmente, no pasará ni por P_1 ni por P_2. Estos puntos solo están ahí para proporcionar información direccional. La distancia entre P_0 y P_1 determina «qué longitud» tiene la curva cuando se mueve hacia la dirección de P_2 antes de dirigirse hacia P_3.

La forma paramétrica de la curva es:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3 \text{ , } t \in [0,1].$$

Generalizando, tenemos:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} \mathbf{P}_i (1-t)^{n-i} t^i = \mathbf{P}_0(1-t)^n + \binom{n}{1}\mathbf{P}_1(1-t)^{n-1}t + \cdots + \mathbf{P}_n t^n \text{ , } t \in [0,1].$$



Algoritmo de De Casteljau

El algoritmo de De Casteljau es un método recursivo para evaluar polinomios en forma de Bernstein o curvas de Bézier, llamado así por su inventor Paul de Casteljau. El algoritmo de De Casteljau también se puede utilizar para dividir una sola curva de Bézier en dos curvas de Bézier en un valor de parámetro arbitrario.

Aunque el algoritmo es más lento para la mayoría de las arquitecturas en comparación con el enfoque directo, es numéricamente más estable.

Una curva Bezier B (de grado n con puntos de control $\beta_0, \ldots, \beta_n$ se puede escribir en la forma de Bernstein como sigue:

$$B(t) = \sum_{i=0}^{n} \beta_i b_{i,n}(t),$$

Donde b es un polinomio de Bernestein:

$$b_{i,n}(t) = \binom{n}{i}(1-t)^{n-i}t^i.$$

La curva en el punto t_0, se puede evaluar con las relaciones recurrentes:

$$\beta_i^{(0)} := \beta_i, \;\; i = 0, \ldots, n$$
$$\beta_i^{(j)} := \beta_i^{(j-1)}(1 - t_0) + \beta_{i+1}^{(j-1)} t_0, \;\; i = 0, \ldots, n - j, \;\; j = 1, \ldots, n$$

Luego la evaluación de B en el punto t_0, puede ser evaluado en $\binom{n}{2}$ operaciones.

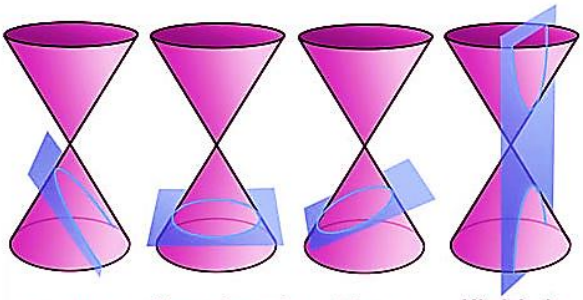
El resultado B(t0) esta dado por: $B(t_0) = \beta_0^{(n)}$.

Mas aun la curva Bezier B se puede dividir en el punto t0 en dos curvas con respectivos puntos de control:

$$\beta_0^{(0)}, \beta_0^{(1)}, \ldots, \beta_0^{(n)}$$
$$\beta_0^{(n)}, \beta_1^{(n-1)}, \ldots, \beta_n^{(0)}$$

3. PRACTICA

A. Hay curvas que resultan de la intersección de un plano con un cono, estas se llaman curvas cónicas y son:



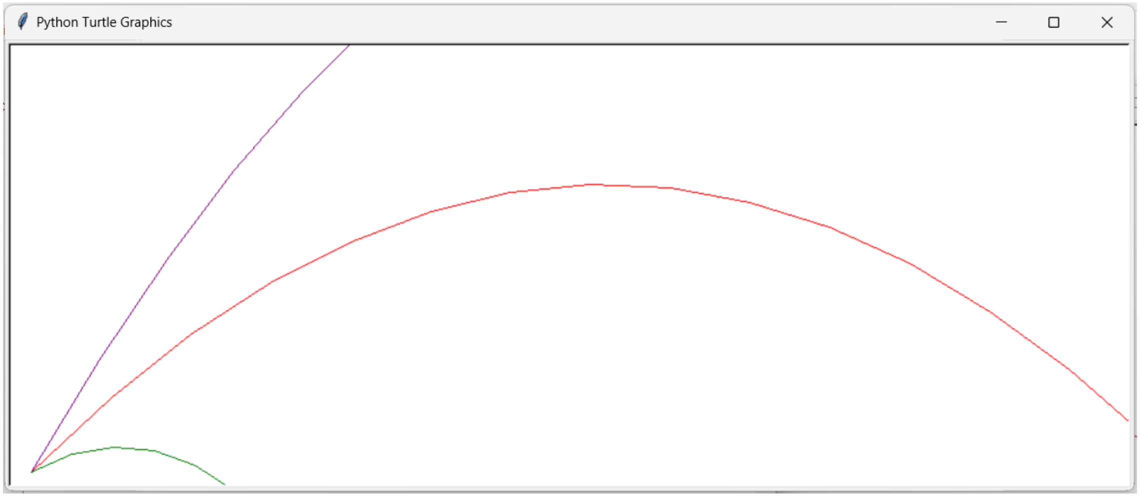
El siguiente código implementa una parábola utilizando la librería turtle y la simulación del disparo de un proyectil en un ángulo y potencia determinada:

```
from turtle import *
import math
import random

G = 9.80665
origen_x = -480
origen_y = -180

def crear_tortuga():
    proyecto = Turtle(shape='turtle')
    proyecto.hideturtle()
    proyecto.penup()
    proyecto.goto(origen_x, origen_y)
    proyecto.pendown()
    proyecto.speed(0)
    proyecto.left(45)
    proyecto.showturtle()
    return proyecto

def dibujar_puntos(turtle):

    angulo = int(input("Ingrese el angulo en grados: "))
    potencia = int(input("Ingrese la potencia: "))

    for tiempo in range(1, 200):
        x = potencia * math.cos(math.radians(angulo)) * tiempo + origen_x
```

```
        y = potencia * math.sin(math.radians(angulo)) * tiempo - (((tiempo ** 2) * G) / 2)
+ origen_y
        turtle.goto(x, y)
window = Screen()
window.setup(1000, 400)

for _ in range(3):
    mi_tortuga = crear_tortuga()
    mi_tortuga.color(random.choice(['red', 'green', 'blue', 'purple', 'black']))
    dibujar_puntos(mi_tortuga)
window.exitonclick()
```

Salida:

```
Ingrese el angulo en grados: 45
Ingrese la potencia: 100
Ingrese el angulo en grados: 60
Ingrese la potencia: 120
Ingrese el angulo en grados: 30
Ingrese la potencia: 42
```



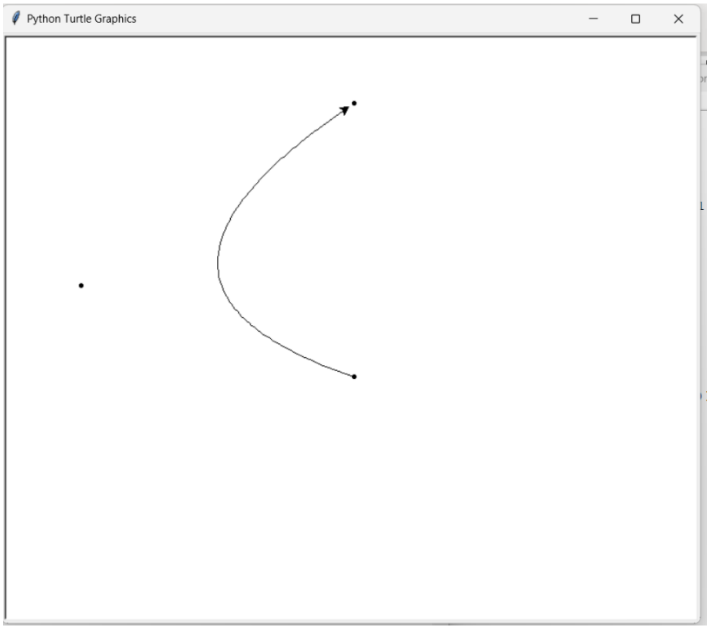
B. De manera similar, implementa el grafico de una hipérbola.

C. El siguiente código implementa curvas cuadráticas de Bezier:

```
from turtle import *

p0 = Vec2D(0, -50)
p1 = Vec2D(-300, 50)
p2 = Vec2D(0, 250)

b = lambda t: (1 - t)**2 * p0 + 2*(1 - t)*t * p1 + t**2 * p2
tortuga = Turtle()
tortuga.penup()

for posicion in [p2, p1, p0]:
    tortuga.goto(posicion)
    tortuga.dot()

tortuga.pendown()
t = 0

while t <= 1:
    posicion = b(t)
```

```
    tortuga.setheading(tortuga.towards(posicion))
    tortuga.goto(posicion)
    t += 0.01

screen = Screen()
screen.exitonclick()
```

Salida:



D. Modifica el código anterior de tal manera que implemente también curvas cubicas y que los puntos sean ingresados desde el Terminal.
E. El siguiente código implementa el Algoritmo de De Casteljau, verifica su correcto funcionamiento además entiende y agrega todos los comentarios posibles al código:

```
from turtle import *
import random

def dibujarLineas(dataset):
    pantalla.colormode(255)
    colorPunto = (random.randint(0,255), random.randint(0,255),
                  random.randint(0,255))
    colorLinea = (random.randint(0,255), random.randint(0,255),
                   random.randint(0,255))
    i = 0
    while i < len(dataset)-1:
        pen.color(colorLinea)
        pen.goto(dataset[i][0], dataset[i][1])
        pen.pendown()
        pen.goto(dataset[i+1][0], dataset[i+1][1])
        pen.penup()
        t_pos = deCasteljau(dataset[i][0], dataset[i][1],
                            dataset[i+1][0], dataset[i+1][1])
        pen.goto(t_pos[0], t_pos[1])
        pen.color(colorPunto)
        pen.dot(5)

        i += 1

def deCasteljau(x1, y1, x2, y2):
```

```python
    t_x = (x1 * (1-funcionT)) + (x2*funcionT)
    t_y = (y1 * (1-funcionT)) + (y2*funcionT)
    t_plot = (t_x, t_y)
    t_coords.append(t_plot)
    return t_plot

pantalla = Screen()
pantalla.screensize(1500,1000)
pantalla.title("Algoritmo de deCasteljau")
pen = Turtle()
pen.speed(5)
pantalla.colormode(255)
pen.penup()
pen.pensize(2)

funcionT = 0.85
t_coords = []
puntosControl = [[(0,0), (200,100), (400, 0)],
                 [(0,0), (100,200), (300,200),
                  (400,0)],
                 [(-100,0), (0, 75), (200,120),
                  (350,75), (400,0)],
                [(-100,0), (0, 100), (100,200),
                 (200,200), (300, 100), (400,0)]]

for poligono in puntosControl:
    for puntocontrol in poligono:
        pen.goto(puntocontrol[0], puntocontrol[1])
        pen.dot(5)

    for punto in range(0, len(poligono)+1):
        dibujarLineas(poligono)
        poligono = t_coords
        t_coords = []
    pantalla.clearscreen()
```