

河北大学

《计算机图形学》实验报告

学 院 网络空间安全与计算机学院

学科门类 工学

专 业 计算机科学与技术

学 号 20201202075

姓 名 myp

指导教师 崔振超

## 一、 实验目的

三次Hermit插值曲线函数及Hermit插值曲线制作，参数自拟。在报告中说明参数的意义，及曲线的结果展示。

制作一个Bezier曲线函数，利用此函数制作可以交互的Bezier曲线生成代码。

完成三阶B样条曲线的绘制。  
设置一个函数，该函数的功能是将给定坐标点进行旋转、平移或放缩变换。利用1中的函数对你所做的简笔画进行向左平移15像素，以图像中心为中心点缩小至原始图像的0.75倍，然后以图像中心为中心旋转30度，然后输出。  
完成一种直线裁剪算法。

## 二、实验过程

### 1. Hermit曲线

def hermite\_basis(t):

h0 = 2 \* t \*\* 3 - 3 \* t \*\* 2 + 1

h1 = -2 \* t \*\* 3 + 3 \* t \*\* 2

h2 = t \*\* 3 - 2 \* t \*\* 2 + t

h3 = t \*\* 3 - t \*\* 2

return h0, h1, h2, h3

def hermite\_interpolation(t, p0, p1, m0, m1):

h0, h1, h2, h3 = hermite\_basis(t)

return h0 \* p0 + h1 \* p1 + h2 \* m0 + h3 \* m1

def Hermite(image):

# 输入端点及其切线

p0, p1 = np.array([50, 500]), np.array([500, 50])

m0, m1 = np.array([250, 0]), np.array([0, 250])

# 在0到1之间生成均匀分布的参数值t

t\_values = np.linspace(0, 1, 100)

# 计算三次Hermite差值曲线上的点

curve\_points = [hermite\_interpolation(t, p0, p1, m0, m1) for t in t\_values]

# 绘制曲线

for i in range(len(curve\_points) - 1):

pt1 = tuple(curve\_points[i].astype(int))

pt2 = tuple(curve\_points[i + 1].astype(int))

cv.line(image, pt1, pt2, (0, 0, 255), 2)

# 绘制端点

cv.circle(image, tuple(p0), 5, (0, 255, 0), -1)

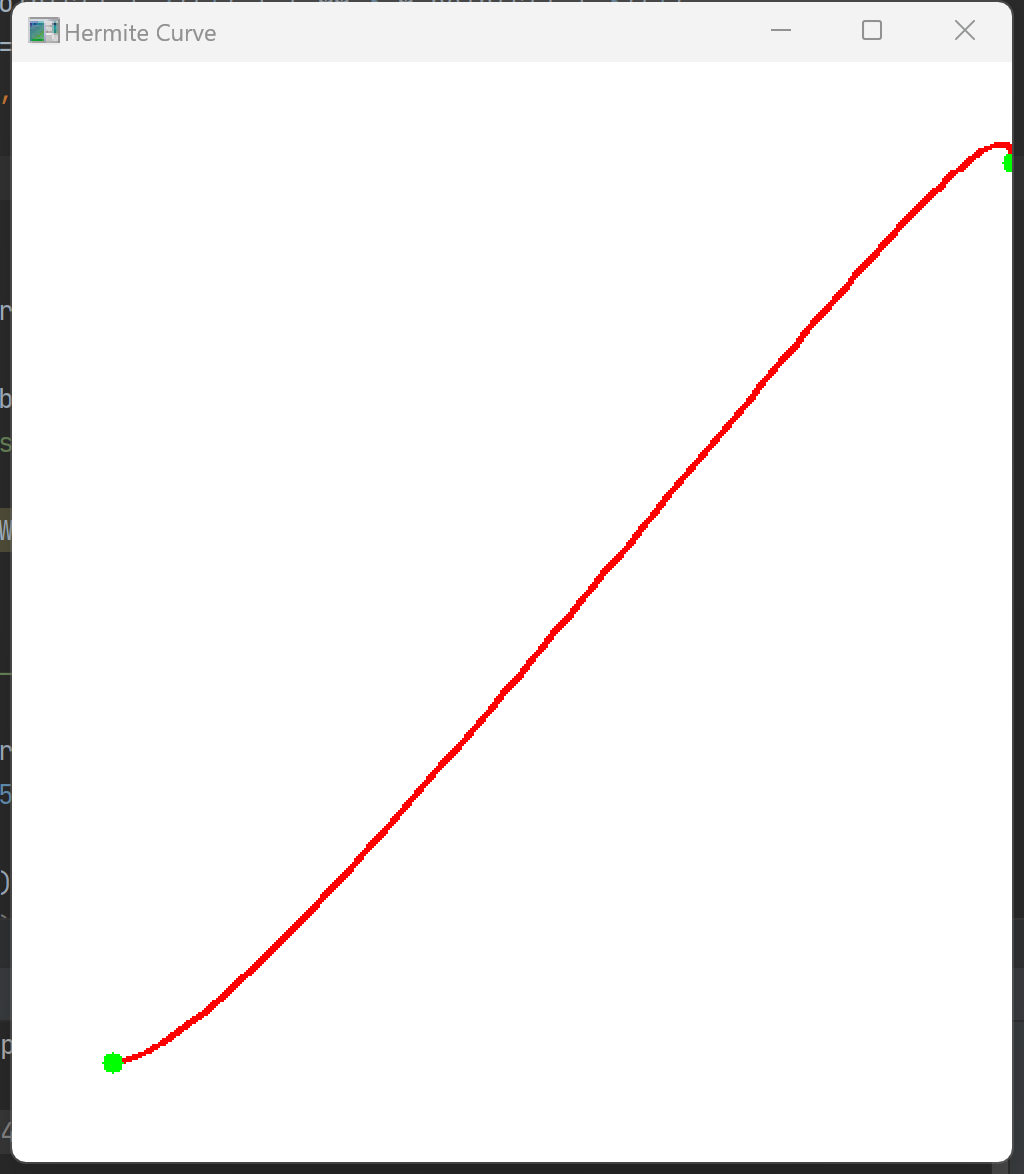
cv.circle(image, tuple(p1), 5, (0, 255, 0), -1)

# 显示图像

cv.imshow("Hermite Curve", image)

cv.waitKey(0)

cv.destroyAllWindows()



### 2. Bizier曲线

def bezier\_curve(t, p0, p1, p2, p3):

b0 = (1 - t) \*\* 3

b1 = 3 \* t \* (1 - t) \*\* 2

b2 = 3 \* t \*\* 2 \* (1 - t)

b3 = t \*\* 3

return b0 \* p0 + b1 \* p1 + b2 \* p2 + b3 \* p3

def draw\_curve(image, curve\_points, color=(0, 0, 0), thickness=2):

for i in range(len(curve\_points) - 1):

p1 = tuple(curve\_points[i].astype(int))

p2 = tuple(curve\_points[i + 1].astype(int))

cv.line(image, p1, p2, color, thickness)

def Bezier(image):

# 定义控制点

p0 = np.array([50, 450])

p1 = np.array([150, 50])

p2 = np.array([350, 50])

p3 = np.array([450, 450])

# 在0到1之间生成均匀分布的参数值t

t\_values = np.linspace(0, 1, 100)

# 计算三次Bezier曲线上的点

curve\_points = np.array([bezier\_curve(t, p0, p1, p2, p3) for t in t\_values])

# 绘制曲线

draw\_curve(image, curve\_points, color=(0, 0, 0), thickness=2)

# 绘制控制点和控制折线

control\_points = np.array([p0, p1, p2, p3])

for i in range(len(control\_points) - 1):

cv.line(image, tuple(control\_points[i]), tuple(control\_points[i + 1]), (0, 0, 255), 1)

cv.drawMarker(image, tuple(p0), (0, 0, 255), cv.MARKER\_STAR, 10, 2)

cv.drawMarker(image, tuple(p1), (0, 0, 255), cv.MARKER\_STAR, 10, 2)

cv.drawMarker(image, tuple(p2), (0, 0, 255), cv.MARKER\_STAR, 10, 2)

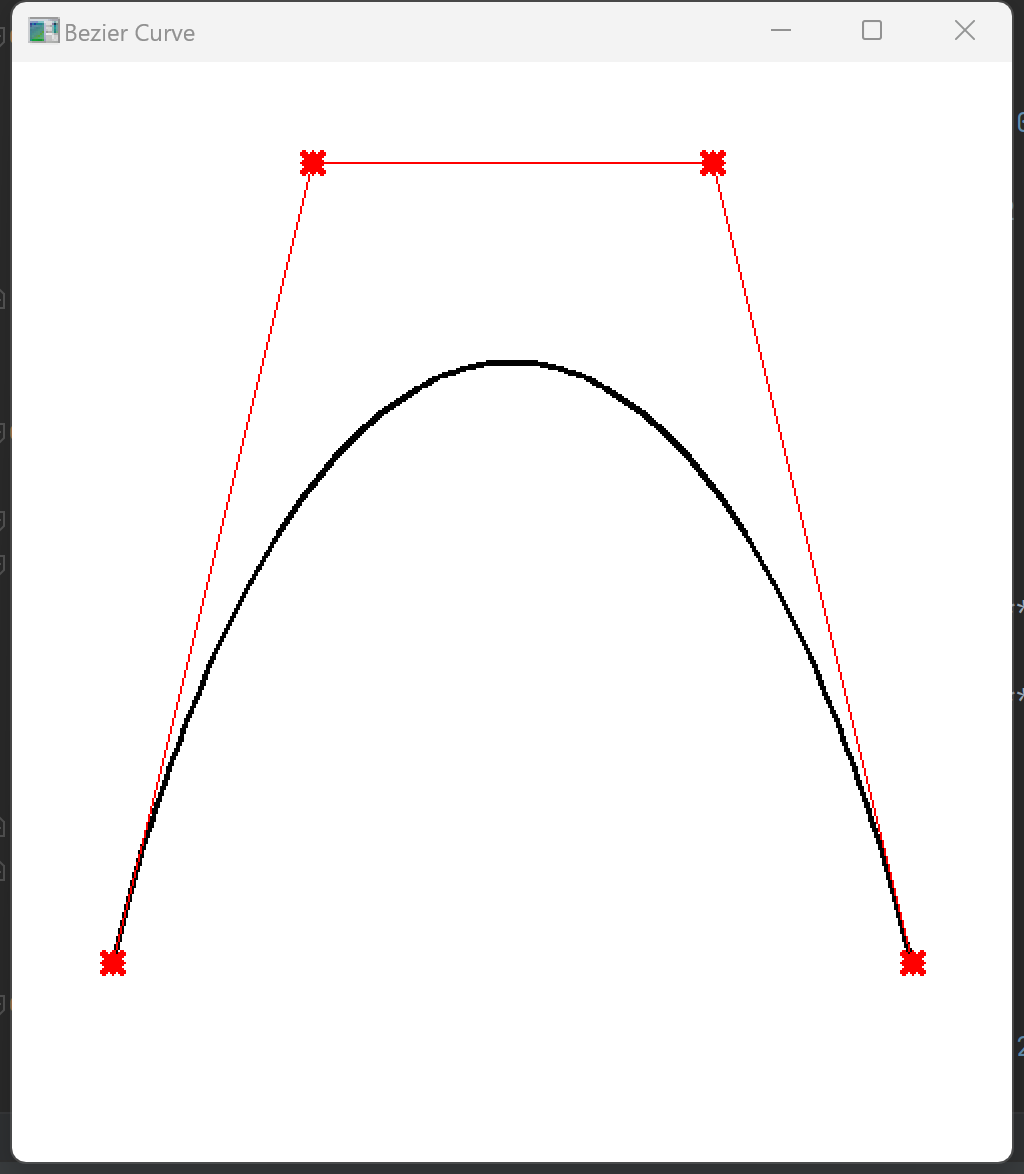
cv.drawMarker(image, tuple(p3), (0, 0, 255), cv.MARKER\_STAR, 10, 2)

# 显示图像

cv.imshow('Bezier Curve', image)

cv.waitKey(0)

cv.destroyAllWindows()



### 3.B样条曲线

def b\_spline(x):

if 0 <= np.abs(x) < 1:

return np.abs(x) \*\* 3 / 2.0 - np.abs(x) \*\* 2 + 2.0 / 3.0

elif 1 <= np.abs(x) < 2:

return -1.0 / 6.0 \* np.abs(x) \*\* 3 + np.abs(x) \*\* 2 - 2 \* np.abs(x) + 4.0 / 3.0

else:

return 0.0

def draw\_bspline(points, img):

n\_points = len(points)

for i in range(n\_points - 3):

for t in np.arange(0, 1, 0.01):

x = (1 - t) \*\* 3 \* points[i][0] + 3 \* t \* (1 - t) \*\* 2 \* points[i + 1][0] + 3 \* t \*\* 2 \* (1 - t) \* \

points[i + 2][0] + t \*\* 3 \* points[i + 3][0]

y = (1 - t) \*\* 3 \* points[i][1] + 3 \* t \* (1 - t) \*\* 2 \* points[i + 1][1] + 3 \* t \*\* 2 \* (1 - t) \* \

points[i + 2][1] + t \*\* 3 \* points[i + 3][1]

x, y = int(x), int(y)

img[y, x, :] = [0, 0, 255]

return img

def B(image):

points = np.array([[50, 200], [150, 50], [350, 350], [400, 200]], np.int32)

image = draw\_bspline(points, image)

cv.imshow("B-spline Curve", image)

cv.waitKey(0)

cv.destroyAllWindows()

图表, 折线图

描述已自动生成

### 4.变换算法

import numpy as np

import math

import cv2 as cv

def DDALine(x1, y1, x2, y2):

dx = x2 - x1

dy = y2 - y1

tmp = 0

# 斜率判断

if (abs(dx) > abs(dy)):

tmp = abs(dx)

else:

tmp = abs(dy)

# 一个等于1，一个小于1

delta\_x = float(dx / tmp)

delta\_y = float(dy / tmp)

x = x1 + 0.5

y = y1 + 0.5

for i in range(0, int(tmp + 1)):

# 绘制像素点

image[int(x), int(y)] = [0, 0, 0]

x += delta\_x

y += delta\_y

def lineMidPoint(x1, y1, x2, y2):

x = x1

y = y1

a = y1 - y2 # 计算差距

b = x2 - x1

if b >= 0:

cx = 1

else:

b = -b

cx = -1

if a <= 0:

cy = 1

else:

a = -a

cy = -1

image[x, y] = [0, 0, 0]

# 斜率绝对值 <= 1

if (-a <= b):

d = 2 \* a + b

d1 = 2 \* a

d2 = 2 \* (a + b)

while x != x2:

if (d < 0):

y += cy

d += d2

else:

d += d1

x += cx

image[x, y] = [0, 0, 0]

# 斜率绝对值 > 1

else:

d = 2 \* b + a

d1 = 2 \* b

d2 = 2 \* (a + b)

while (y != y2):

if (d < 0):

d += d1

else:

x += cx

d += d2

y += cy

image[x, y] = [0, 0, 0]

def BresenhamLine(x1, y1, x2, y2):

dx = abs(x2 - x1)

dy = abs(y2 - y1)

# 根据直线的走势方向，设置变化的单位是正是负

s1 = 1 if ((x2 - x1) > 0) else -1

s2 = 1 if ((y2 - y1) > 0) else -1

# 根据斜率的大小，交换dx和dy，可以理解为变化x轴和y轴使得斜率的绝对值为[0,1]

boolInterChange = False

if dy > dx:

tmp = dx

dx = dy

dy = tmp

boolInterChange = True

# 初始误差

e = 2 \* dy - dx

x = x1

y = y1

for i in range(0, int(dx + 1)):

image[x, y] = [0, 0, 0]

if e >= 0:

# 此时要选择横纵坐标都不同的点，根据斜率的不同，让变化小的一边变化一个单位

if boolInterChange:

x += s1

else:

y += s2

e -= 2 \* dx

# 根据斜率的不同，让变化大的方向改变一单位，保证两边的变化小于等于1单位，让直线更加均匀

if boolInterChange:

y += s2

else:

x += s1

e += 2 \* dy

def MidpointCircle(x0, y0, r):

x = x0 # 对x赋值

y = y0 + r # 对y赋值

d = 1.25 - r # 求d

image[x, y] = [0, 0, 0]

while x + y0 - x0 - y < 0: # 遍历x，y

if d < 0: # 当d小于0时

d = d + 2 \* x + 3 - 2 \* x0 # d进行迭代

x = x + 1

else:

d = d + 2 \* (x - y) + 5 + 2 \* (y0 - x0) # d进行迭代

x = x + 1 # x + 1

y = y - 1 # x + 1

image[x, y] = [0, 0, 0]

image[2 \* x0 - x, y] = [0, 0, 0]

image[x0 + y0 - y, x0 + y0 - x] = [0, 0, 0]

image[x0 - y0 + y, x0 + y0 - x] = [0, 0, 0]

image[2 \* x0 - x, 2 \* y0 - y] = [0, 0, 0]

image[x, 2 \* y0 - y] = [0, 0, 0]

image[x0 - y0 + y, y0 - x0 + x] = [0, 0, 0]

image[x0 + y0 - y, y0 - x0 + x] = [0, 0, 0]

def Bresenhamcircle(x1, y1, r):

x = 0 # 设置变换

y = r # 设置变化范围

d = 3 - 2 \* r # d的变化

while x <= y:

image[x1 + x, y1 + y] = [0, 0, 0]

image[x1 - x, y1 + y] = [0, 0, 0]

image[x1 + x, y1 - y] = [0, 0, 0]

image[x1 - x, y1 - y] = [0, 0, 0]

image[x1 + y, y1 + x] = [0, 0, 0]

image[x1 - y, y1 + x] = [0, 0, 0]

image[x1 + y, y1 - x] = [0, 0, 0]

image[x1 - y, y1 - x] = [0, 0, 0]

if d < 0: # 如果d小于0

d = d + 4 \* x + 6 # d进行迭代

else:

d = d + 4 \* (x - y) + 10 # d进行迭代

y = y - 1 # y-1

x = x + 1 # x+1

def Bresenhamcircle2(x1, y1, r):

x = 0 # 设置变换

y = r # 设置变化范围

d = 3 - 2 \* r # d的变化

while x <= y:

image[x1 + y, y1 + x] = [0, 0, 0]

image[x1 + y, y1 - x] = [0, 0, 0]

if d < 0: # 如果d小于0

d = d + 4 \* x + 6 # d进行迭代

else:

d = d + 4 \* (x - y) + 10 # d进行迭代

y = y - 1 # y-1

x = x + 1 # x+1

def Boundaryfill(x, y):

if image[x, y].all() != 0: # 判断是否被便利过和是否是要改的位置

image[x, y] = [0, 0, 0] # 更改对对应位置像素值

Boundaryfill(x, y - 1) # 向四个方向探索

Boundaryfill(x, y + 1) # 向四个方向探索

Boundaryfill(x - 1, y) # 向四个方向探索

Boundaryfill(x + 1, y) # 向四个方向探索

def translation(P, Tx, Ty):

P.append(1)

P = np.matrix(P)

T = [

[1, 0, 0],

[0, 1, 0],

[Tx, Ty, 1]

]

T = np.matrix(T)

PP = P \* T

return list(np.array(PP))[0]

def draw():

Bresenhamcircle(150, 150, 100) # 画头的轮廓

Bresenhamcircle(175, 150, 75) # 画脸的轮廓

Bresenhamcircle(90, 130, 20) # 画左眼

Bresenhamcircle(90, 170, 20) # 画右眼

Bresenhamcircle(115, 150, 10) # 画鼻子

Bresenhamcircle(93, 142, 5) # 画左眼眼球

image[98, 142] = [0, 0, 0] # 填充圆的缺失点

image[88, 142] = [0, 0, 0] # 填充圆的缺失点

image[93, 137] = [0, 0, 0] # 填充圆的缺失点

image[93, 147] = [0, 0, 0] # 填充圆的缺失点

Bresenhamcircle(93, 158, 5) # 画右眼眼球

image[98, 158] = [0, 0, 0] # 填充圆的缺失点

image[88, 158] = [0, 0, 0] # 填充圆的缺失点

image[93, 153] = [0, 0, 0] # 填充圆的缺失点

image[93, 163] = [0, 0, 0] # 填充圆的缺失点

DDALine(125, 150, 225, 150) # 画鼻子

Bresenhamcircle2(150, 150, 75) # 画微笑的嘴

DDALine(165, 95, 165, 135) # 画左边胡子直线

DDALine(165, 165, 165, 205) # 画左上胡子

DDALine(140, 95, 150, 135) # 画左下胡子

DDALine(190, 95, 180, 135) # 画右边胡子直线

DDALine(150, 165, 140, 205) # 画右上胡子

DDALine(180, 165, 190, 205) # 画右下胡子

Bresenhamcircle(250, 150, 10) # 画铃铛的外形

lineMidPoint(255, 150, 260, 150) # 画铃铛的缝

Bresenhamcircle(252, 150, 3) # 画铃铛的孔

image[252, 153] = [0, 0, 0] # 填充圆的缺失点

image[252, 147] = [0, 0, 0] # 填充圆的缺失点

image[249, 150] = [0, 0, 0] # 填充圆的缺失点

image[255, 150] = [0, 0, 0] # 填充圆的缺失点

image[254, 148] = [0, 0, 0] # 填充圆的缺失点

image[254, 152] = [0, 0, 0] # 填充圆的缺失点

image[250, 148] = [0, 0, 0] # 填充圆的缺失点

image[250, 152] = [0, 0, 0] # 填充圆的缺失点

Boundaryfill(93, 146) # 填充左眼球

Boundaryfill(93, 159) # 填充右眼球

Boundaryfill(252, 151) # 填充铃铛的的孔

# 比例变换

def scale(P, Sx, Sy, x0=0, y0=0):

P.append(1)

P = np.matrix(P)

S = [

[Sx, 0, 0],

[0, Sy, 0],

[x0 \* (1 - Sx), y0 \* (1 - Sy), 1]

]

S = np.matrix(S)

PP = P \* S

return list(np.array(PP))[0]

# 旋转变换

def rotate(P, t, x0=0, y0=0):

t = t / 180 \* math.pi

P.append(1)

P = np.matrix(P)

T = [

[math.cos(t), math.sin(t), 0],

[-math.sin(t), math.cos(t), 0],

[x0 \* (1 - math.cos(t)) + y0 \* math.sin(t), y0 \* (1 - math.cos(t)) - x0 \* math.sin(t), 1]

]

T = np.matrix(T)

PP = P \* T

return list(np.array(PP))[0]

image = np.zeros((550, 550, 3), np.uint8)

image[:] = [255, 255, 255]

draw()

newimg = np.ones((550, 550, 3), dtype=np.uint8)

newimg[:] = [255, 255, 255]

i = 0

while i < 550:

j = 0

while j < 550:

P = rotate([i, j], 30, 350, 350)

P = scale([P[0], P[1]], 0.5, 0.5, 350, 350)

# print(P)

newimg[min(549, int(P[0])), min(549, int(P[1]))] = image[i, j]

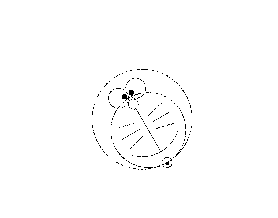
j += 1

i += 1

# 显示图像

cv.imshow('img', newimg)

cv.waitKey(0)



### 5.裁剪算法

import cv2

import numpy as np

# 创建图像背景

img = np.zeros((600, 800, 3), np.uint8)

# 定义全局变量

is\_drawing, point\_counter = False, 2

line\_point1\_x, line\_point1\_y, line\_point2\_x, line\_point2\_y = 0, 0, 0, 0

rect\_point1\_x, rect\_point1\_y, rect\_point2\_x, rect\_point2\_y = 0, 0, 0, 0

def liang\_barsky\_algorithm(x1, y1, x2, y2, xl, yb, xr, yt):

dx = x2 - x1

dy = y2 - y1

p = [1, -dx, dx, -dy, dy]

q = [1, x1 - xl, xr - x1, y1 - yb, yt - y1]

cv2.line(img, (x1, y1), (x2, y2), (0, 0, 0))

# 如果线段完全在裁剪区域外，则不进行裁剪

if (dx == 0 and (q[1] < 0 or q[2] < 0)) or (dy == 0 and (q[3] < 0 or q[4] < 0)):

return

umax, umin = 0, 1

for i in range(1, 5):

if p[i] > 0:

umin = min(umin, q[i] / p[i])

elif p[i] < 0:

umax = max(umax, q[i] / p[i])

if umax > umin:

return

x1, y1, x2, y2 = int(x1 + umax \* dx), int(y1 + umax \* dy), int(x1 + umin \* dx), int(y1 + umin \* dy)

# 显示裁剪后的线段的端点坐标

cv2.putText(img, f'(x:{x1},y:{y1})', (x1, y1), cv2.FONT\_HERSHEY\_PLAIN, 1.0, (0, 0, 255), thickness=1)

cv2.putText(img, f'(x:{x2},y:{y2})', (x2, y2), cv2.FONT\_HERSHEY\_PLAIN, 1.0, (0, 0, 255), thickness=1)

# 绘制裁剪后的线段

cv2.line(img, (x1, y1), (x2, y2), (255, 255, 255))

def draw\_func(event, x, y, flags, param):

global is\_drawing, point\_counter

global line\_point1\_x, line\_point1\_y, line\_point2\_x, line\_point2\_y

global rect\_point1\_x, rect\_point1\_y, rect\_point2\_x, rect\_point2\_y

if event == cv2.EVENT\_LBUTTONDOWN:

cv2.circle(img, (x, y), 1, (0, 255, 0), thickness=5)

if point\_counter == 2:

point\_counter -= 1

rect\_point1\_x, rect\_point1\_y = x, y

elif point\_counter == 1:

point\_counter -= 1

rect\_point2\_x, rect\_point2\_y = x, y

cv2.rectangle(img, (rect\_point1\_x, rect\_point1\_y), (rect\_point2\_x, rect\_point2\_y), (255, 0, 255))

else:

if is\_drawing:

line\_point2\_x, line\_point2\_y = x, y

cv2.line(img, (line\_point1\_x, line\_point1\_y), (line\_point2\_x, line\_point2\_y), (255, 255, 255))

is\_drawing = False

else:

line\_point1\_x, line\_point1\_y = x, y

is\_drawing = True

cv2.namedWindow('src')

cv2.setMouseCallback('src', draw\_func)

while True:

cv2.imshow('src', img)

code = cv2.waitKey(100)

# 按下'q'键退出

if code == ord('q'):

break

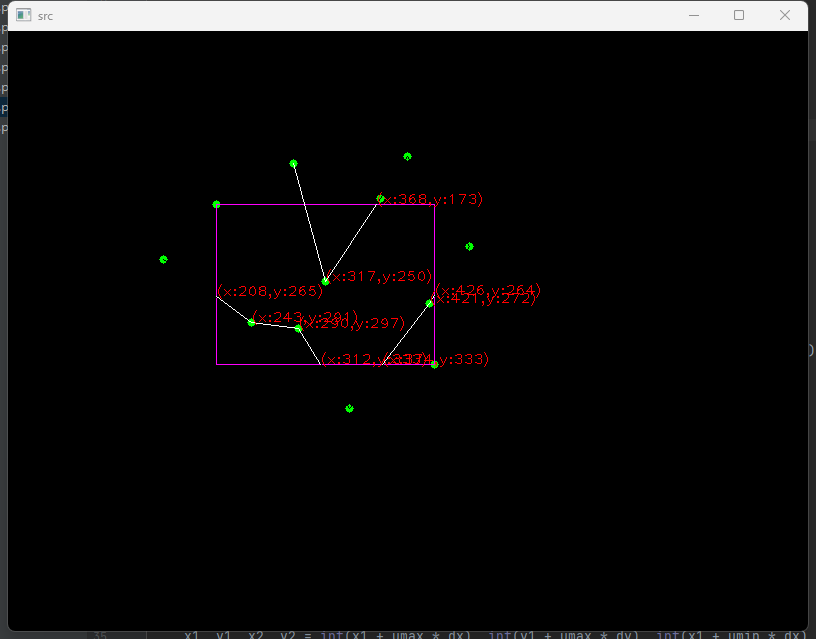
# 按下'x'键进行裁剪

elif code == ord('x'):

liang\_barsky\_algorithm(line\_point1\_x, line\_point1\_y, line\_point2\_x, line\_point2\_y, rect\_point1\_x, rect\_point1\_y,

rect\_point2\_x, rect\_point2\_y)

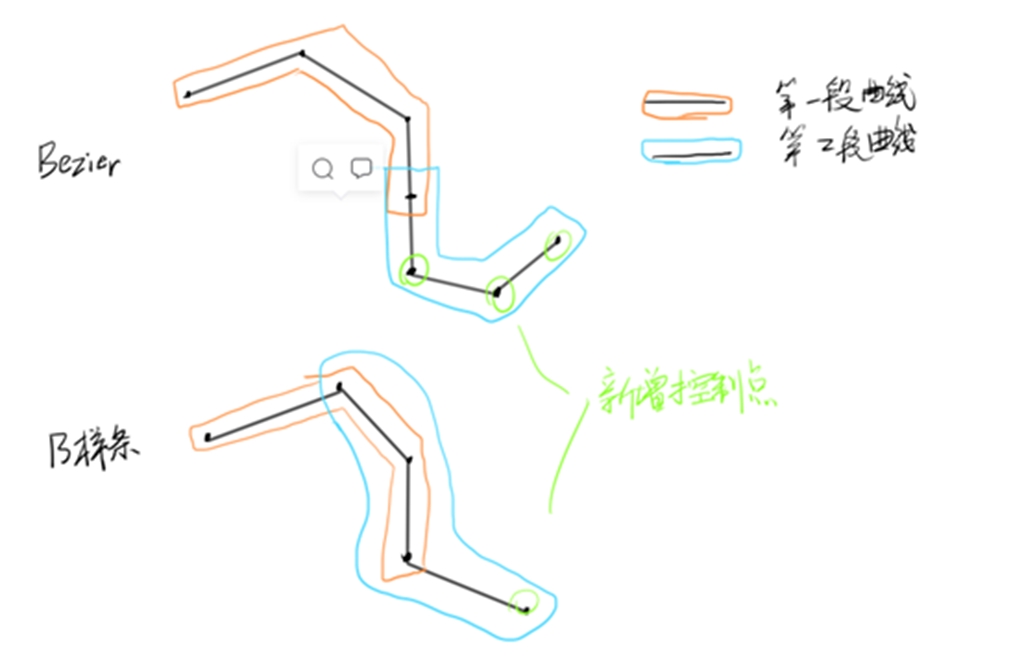
cv2.destroyAllWindows()



## 三、算法比较

### Bezier曲线和B样条曲线的异同：

三次Bezier曲线是四个控制点为一组，每增加一条曲线需要增加三个控制点，即控制点数= 3\*曲线数+1。而三次B样条曲线也是四个控制点为一组，不过每增加一条曲线只需要添加一个控制点，即控制点数=曲线数+3。三次

Bezier曲线每两段相邻曲线有一个相同的控制点，而三次B样条曲线有三个。

Bezier曲线存在不足：一是控制多边形的顶点个数决定了Bezier曲线的阶次，即n+1个顶点的控制多边形必然会产生n次Bezier曲线，并且当n较大时，控制多边形对曲线的控制将会减弱。二是Bezier不能做局部修改，即改变某一个控制点的位置对整条曲线都有影响。

B样条方法保留了Bezier方法的优点，克服其由于整体表示带来的不具备局部性质的特点，具有表示与设计自由型曲线曲面的强大功能。

四、

## 四、心得体会

本次实验，最大的体会就是遇到问题一定要多想原理，一步一步看是在那出问题了，可以手动的去算几步，比如：书中给出的矩阵乘积是直接有结果的。但是当自己推几步后，感悟会更加深刻。

遇到问题，首先自己思考，多想想算法的含义，把含义吃透，彻底理解了，就会顺畅很多