

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 计科1班

学 号 20201202075

姓 名 myp

指导教师 崔振超

2023年4月3日

一、 实验目的

利用直线、圆与填充算法绘制图形。图形自拟，图形的复杂性与分值成正比。

要求：

作用算法必须包括：中点直线，中点圆，Bresenham直线，Bresenham圆与填充算法。

代码中包括尽量多的注释。

不能抄袭。

二、 设计方案

一、

利用breseham画圆算法，画出多啦A梦的脑袋、眼睛、脸等，利用中点划线算法，画出胡子、鼻子等，在改良breseham画圆算法画出微笑，最后画出通过

种子填充算法进行上色

二、

采用Bresenham画线算法，输入参数为起点坐标和终点坐标。注意，此算法斜率k满足0<=k<=1,且x1<x2。返回值为直线段各点的坐标。使用时运用对称即可得到任意斜率的直线。

采用中点画圆算法，通过重新推导书中的算法（书中算法圆心为坐标原点，且仅为1/8圆），利用对称等方法，设计出可以指定圆心和半径的圆。输入参数为圆心坐标和半径，输出为圆的各点坐标。

三、 详细代码

1、中点画线算法M文件如下

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]

2.Bresehamline画线算法M文件如下

def GenericBresenhamLine(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

3、中点画圆算法M文件如下

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]

4、Bresehamcircle画圆算法M文件如下

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

5、种子填充的M文件如下

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

6、画多啦A梦的的M文件如下

if \_\_name\_\_ == '\_\_main\_\_':

image = np.zeros((1000, 1000, 3), np.uint8)

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

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

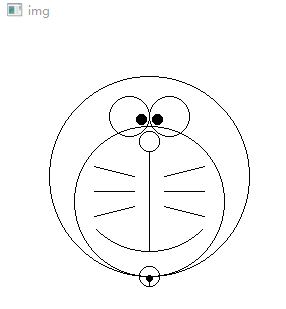
Boundaryfill(93, 159) # 填充右眼球

Boundaryfill(252, 151) # 填充铃铛的的孔

cv.imshow('img', image)

cv.waitKey(0)

四、实验结果



五、实验体会

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：书中给出的画线算法对斜率有限制、画圆算法如何指定圆心以及如何根据1/8圆绘制整个圆等。

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