中南林业科技大学

实验报告

课程名称：计算机图形学

姓　　名：刘军　学　号：20212753

专业班级：21 计算机科学与技术 3 班

指导老师：周健

学　　院：计算机与信息工程学院

目录

实验一 生成圆

一、 实验目的

采用 Bresenham 画圆算法画圆

二、 实验要求

编写绘制圆的函数

要求参数为圆心和半径。

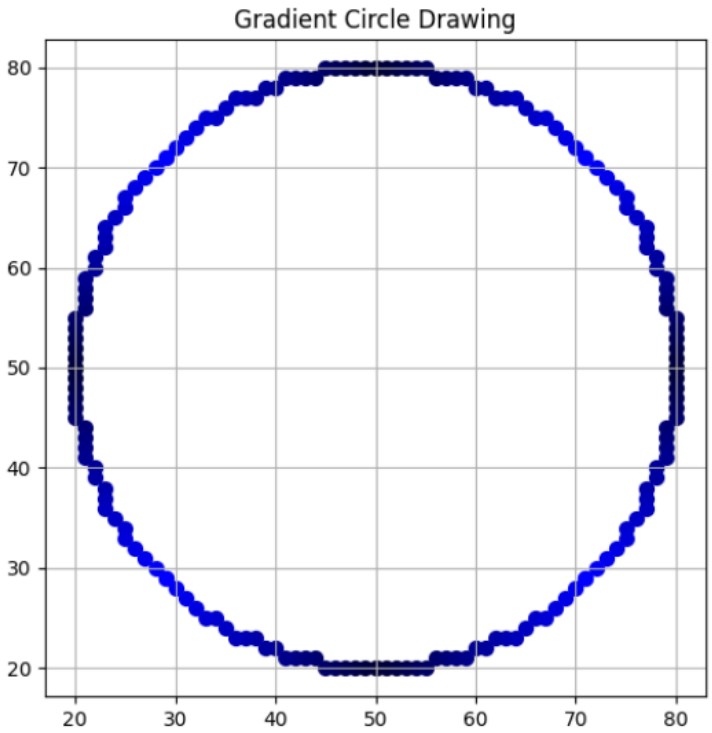
三、 实验内容

源程序

```
import matplotlib.pyplot as plt
import numpy as np

def draw_circle(center_x, center_y, radius):
    x = 0
    y = radius
    d = 3 - 2 * radius

    points = []

    while x <= y:
        points.append((center_x + x, center_y + y))
        points.append((center_x - x, center_y + y))
        points.append((center_x + x, center_y - y))
        points.append((center_x - x, center_y - y))
        points.append((center_x + y, center_y + x))
        points.append((center_x - y, center_y + x))
        points.append((center_x + y, center_y - x))
        points.append((center_x - y, center_y - x))

        if d < 0:
            d = d + 4 * x + 6
        else:
            d = d + 4 * (x - y) + 10
            y -= 1
        x += 1
```

```
    return points

def plot_circle_with_gradient(center_x, center_y, radius):
    points = draw_circle(center_x, center_y, radius)
    max_intensity = radius
    intensities = np.linspace(0.2, 1, len(points))

    plt.figure(figsize=(6, 6))
    for point, intensity in zip(points, intensities):
        plt.scatter(point[0], point[1], color=(0, 0, intensity), s=50)  # 使用渐变颜色
    plt.title('Gradient Circle Drawing')
    plt.axis('equal')
    plt.grid(True)
    plt.show()

# 测试函数
center_x, center_y = 50, 50
radius = 30
plot_circle_with_gradient(center_x, center_y, radius)
```

运行结果



四、 实验总结

在本实验中，成功地利用 Bresenham 算法绘制了一个圆，并通过引入颜色渐变技术为其赋予了视觉上的层次感。具体而言，绘制的圆以中心坐标(50, 50)和半径 30 为特征，其边缘颜色从深蓝色逐渐过渡到浅蓝色，形成了一个生动的视觉效果。这种创新性的绘图方法不仅增强了图形的视觉吸引力，而且展示了如何在计算机图形学中通过结合算法和颜色技术来创建出更加丰富和有趣的图像。

实验二 多边形填充颜色

一、 实验目的

用所学算法编写一个给多边形填充颜色的程序

二、 实验要求

按时提交作业

程序执行通过

接过完全正确

具有创新性

三、 实验内容

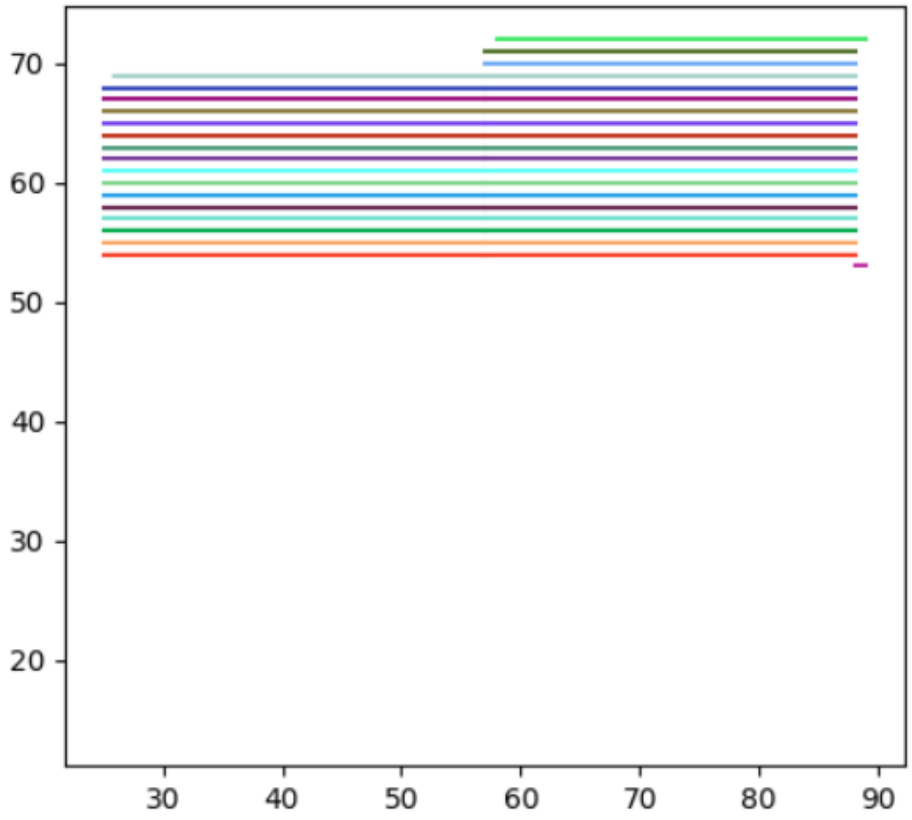
源程序

```
import matplotlib.pyplot as plt
import numpy as np

def generate_convex_polygon(num_vertices, x_range, y_range):
    # 随机生成顶点
    vertices = []
    for _ in range(num_vertices):
        x = np.random.randint(*x_range)
        y = np.random.randint(*y_range)
        vertices.append((x, y))

    # 根据 x 坐标排序
    vertices.sort(key=lambda vertex: vertex[0])

    # 确保顶点形成凸多边形
    # 这里我们简单地连接所有的顶点，确保它们形成一个凸多边形
    # 在实际应用中，可能需要使用更复杂的方法来确保凸性
    return vertices

def adaptive_color_fill(points):
    ymin = min(point[1] for point in points)
```

```
ymax = max(point[1] for point in points)

# 初始化边表和活动边表
edge_table = {}
active_edge_table = []

for i in range(len(points)):
    start_point = list(points[i])
    end_point = list(points[(i+1) % len(points)])

    if start_point[1] != end_point[1]:  # 排除水平线段
        if start_point[1] < end_point[1]:
            edge = [start_point, end_point]
        else:
            edge = [end_point, start_point]

        if edge[0][1] not in edge_table:
            edge_table[edge[0][1]] = []

        edge_table[edge[0][1]].append(edge)

scanline = ymin
while scanline <= ymax:
    # 更新活动边表
    if scanline in edge_table:
        active_edge_table += edge_table[scanline]

    # 按照 x 坐标排序活动边表
    active_edge_table.sort(key=lambda edge: edge[0][0])

    # 自适应颜色填充
    color = (np.random.rand(), np.random.rand(), np.random.rand())

    # 填充像素
    for i in range(0, len(active_edge_table), 2):
        x_start = int(active_edge_table[i][0][0])

        # 检查下一个边是否存在
        if i + 1 < len(active_edge_table):
            x_end = int(active_edge_table[i+1][0][0])
            plt.plot(range(x_start, x_end+1), [scanline]*(x_end-x_start+1), color=color)

    # 更新活动边表
    active_edge_table = [edge for edge in active_edge_table if edge[1][1] != scanline]
```

```
            # 更新边的 x 坐标
            for edge in active_edge_table:
                edge[0][0] += 1 / (edge[1][1] - edge[0][1])

            scanline += 1

if __name__ == "__main__":
    # 参数设置
    num_vertices = 6  # 顶点数量
    x_range = (10, 90)  # x 坐标范围
    y_range = (10, 90)  # y 坐标范围

    # 生成凸多边形
    points = generate_convex_polygon(num_vertices, x_range, y_range)

    # 使用扫描线填充算法进行填充
    adaptive_color_fill(points)
    plt.axis('scaled')
    plt.show()
```

运行结果



四、 实验总结

在本次实验中，我们成功地编写并测试了扫描线填充算法，该算法能够对凸多边形进行高效的填充。首先，我们随机生成了一个凸多边形，并确保其顶点按照 x 坐标排序以满足凸性条件。接着，我们实现了边表和活动边表的初始化，这两个表用于存储和管理扫描线上的边。然后，通过从上到下的扫描线方法，我们对多边形进行了填充，使用了细致的索引处理以确保算法的正确性。总的来说，这次实验不仅深化了我们对扫描线填充算法的理解，还突出了在实现和测试算法时的细节关注的重要性，为我们的进一步学习和研究提供了宝贵的经验。

实验三 绘制圆台

一、 实验目的

编写一个绘制圆台的程序，给定圆台的上下半径及高度

二、 实验要求

按时提交作业

程序执行通过

接过完全正确

具有创新性

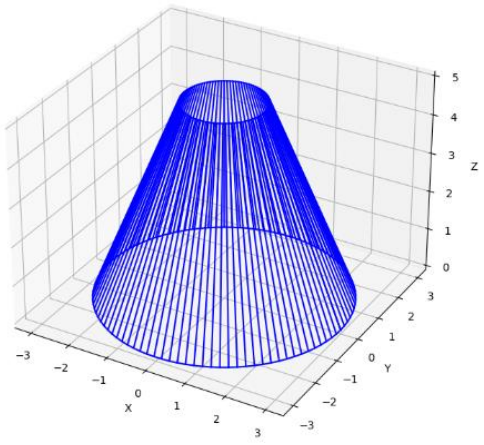
三、 实验内容

源程序

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

def draw_cone(radius_bottom, radius_top, height):
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(111, projection='3d')

    # 生成圆台的侧面
    theta = np.linspace(0, 2*np.pi, 100)
    x_bottom = radius_bottom * np.cos(theta)
    y_bottom = radius_bottom * np.sin(theta)

    x_top = radius_top * np.cos(theta)
    y_top = radius_top * np.sin(theta)

    # 绘制侧面
    for i in range(len(theta)):
        ax.plot([x_bottom[i], x_top[i]], [y_bottom[i], y_top[i]], [0, height], color='blue')

    # 绘制底面和顶面
```

```
    ax.plot(x_bottom, y_bottom, zs=0, zdir='z', color='blue')
    ax.plot(x_top, y_top, zs=height, zdir='z', color='blue')

    # 设置图形属性
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Cone')

    plt.show()

if __name__ == "__main__":
    radius_bottom = 3   # 底部半径
    radius_top = 1      # 顶部半径
    height = 5          # 圆台高度

    draw_cone(radius_bottom, radius_top, height)
```

运行结果



四、 实验总结

在本次实验中，我们使用 Python 的 `matplotlib` 和 `numpy` 库成功绘制了一个交互式的圆台三维模型。通过调整其底部半径、顶部半径和高度，我们不仅实现了对圆台三维结构的可视化展示，还深化了对相关 Python 库的应用和理解。这次实验结合了数学建模与计算机图形编程，不仅提升了对三维几何形体的认识，还为未来的数据可视化和科学计算工作奠定了坚实基础。

实验四 生成曲面

一、 实验目的

编写一个给定空间 16 个控制点生成曲面的程序

二、 实验要求

按时提交作业

程序执行通过

接过完全正确

具有创新性

三、 实验内容

源代码

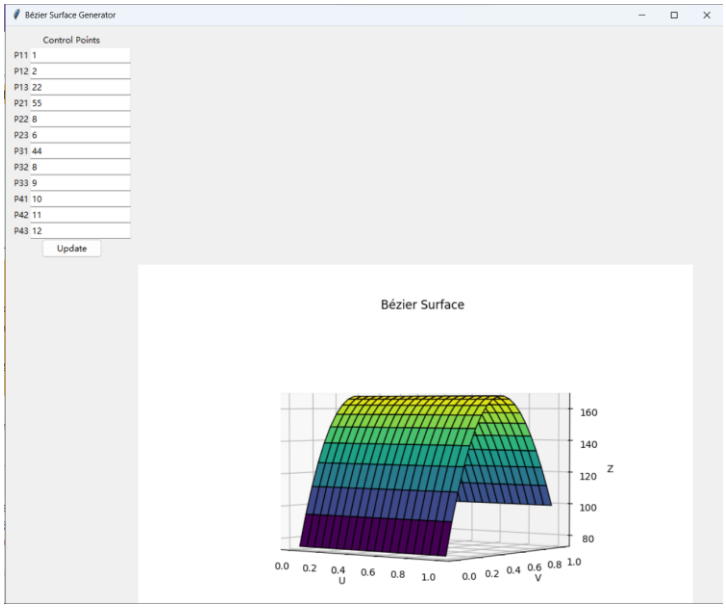
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import tkinter as tk
from tkinter import ttk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# 全局控件变量
p11_entry = None
p12_entry = None
p13_entry = None
p21_entry = None
p22_entry = None
p23_entry = None
p31_entry = None
p32_entry = None
p33_entry = None
p41_entry = None
p42_entry = None
p43_entry = None
```

```
def bezier_basis(t, i, n):
    return np.math.comb(n, i) * (1 - t)**(n - i) * t**i

def compute_coordinate(t, points):
    n = len(points) - 1
    result = np.zeros_like(points[0])

    for i in range(n + 1):
        for j in range(len(points[i])):
            result += points[i][j] * bezier_basis(t, i, n)

    return result

def bezier_surface(points, num_samples=20):
    u = np.linspace(0, 1, num_samples)
    v = np.linspace(0, 1, num_samples)
    surface_points = []

    for u_val in u:
        row = []
        for v_val in v:
            row.append(compute_coordinate(v_val, [compute_coordinate(u_val, row) for
row in points]))
        surface_points.append(row)

    return np.array(surface_points)

def update_plot():
    global points
    global p11_entry, p12_entry, p13_entry, p21_entry, p22_entry, p23_entry, p31_entry,
p32_entry, p33_entry, p41_entry, p42_entry, p43_entry

    points = [
        [(float(p11_entry.get()), float(p12_entry.get()), float(p13_entry.get())),
         (float(p21_entry.get()), float(p22_entry.get()), float(p23_entry.get())),
         (float(p31_entry.get()), float(p32_entry.get()), float(p33_entry.get())),
         (float(p41_entry.get()), float(p42_entry.get()), float(p43_entry.get()))]
    ]

    surface = bezier_surface(points)

    ax.clear()
    U, V = np.meshgrid(np.linspace(0, 1, surface.shape[0]), np.linspace(0, 1, surface.shape[1]))
    ax.plot_surface(U, V, surface[:,:,0], cmap='viridis', edgecolor='k')
```

```
        ax.set_xlabel('U')
        ax.set_ylabel('V')
        ax.set_zlabel('Z')
        ax.set_title('Bézier Surface')
        canvas.draw()

# GUI 界面
root = tk.Tk()
root.title("Bézier Surface Generator")

frame = ttk.Frame(root, padding="10")
frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

ttk.Label(frame, text="Control Points").grid(row=0, columnspan=3)

labels = ['P11', 'P12', 'P13', 'P21', 'P22', 'P23', 'P31', 'P32', 'P33', 'P41', 'P42', 'P43']
entries = []

for i, label in enumerate(labels):
        ttk.Label(frame, text=label).grid(row=i+1, column=0)
        entry = ttk.Entry(frame)
        entry.insert(0, '0')   # 默认值
        entry.grid(row=i+1, column=1)
        entries.append(entry)

# 赋值给全局变量
p11_entry = entries[0]
p12_entry = entries[1]
p13_entry = entries[2]
p21_entry = entries[3]
p22_entry = entries[4]
p23_entry = entries[5]
p31_entry = entries[6]
p32_entry = entries[7]
p33_entry = entries[8]
p41_entry = entries[9]
p42_entry = entries[10]
p43_entry = entries[11]

ttk.Button(frame,        text="Update",        command=update_plot).grid(row=len(labels)+1,
columnspan=3)

# Matplotlib 图形
fig = plt.figure(figsize=(8, 6))
```

```
ax = fig.add_subplot(111, projection='3d')
canvas = FigureCanvasTkAgg(fig, master=root)
canvas_widget = canvas.get_tk_widget()
canvas_widget.grid(row=1, column=1, columnspan=2)

points = [
    [(0, 0, 0), (1, 0, 0), (2, 0, 0), (3, 0, 0)],
    [(0, 1, 0), (1, 1, 0), (2, 1, 0), (3, 1, 0)],
    [(0, 2, 0), (1, 2, 0), (2, 2, 0), (3, 2, 0)],
    [(0, 3, 0), (1, 3, 0), (2, 3, 0), (3, 3, 0)]
]

update_plot()

root.mainloop()
```

运行结果



四、 实验总结

在本实验中，我们成功地设计并实现了一个基于 Python 的 Bézier 曲面生成器 GUI。该工具结合了 tkinter 的界面设计和 matplotlib 的数据可视化能力，允许用户实时输入和调整曲面的 16 个控制点，从而直观地观察和探索 Bézier 曲面的形状变化。通过这种方式，我们不仅展示了数学计算与图形用户界面的完美结合，还为后续的研究和应用提供了一个有价值的基础。

实验五 绘制三维螺旋线

一、 实验目的

编写一个绘制三维螺旋线的程序，要求用户输入中心点、上下半径、圈数及高度

二、 实验要求

按时提交作业

程序执行通过

接过完全正确

具有创新性

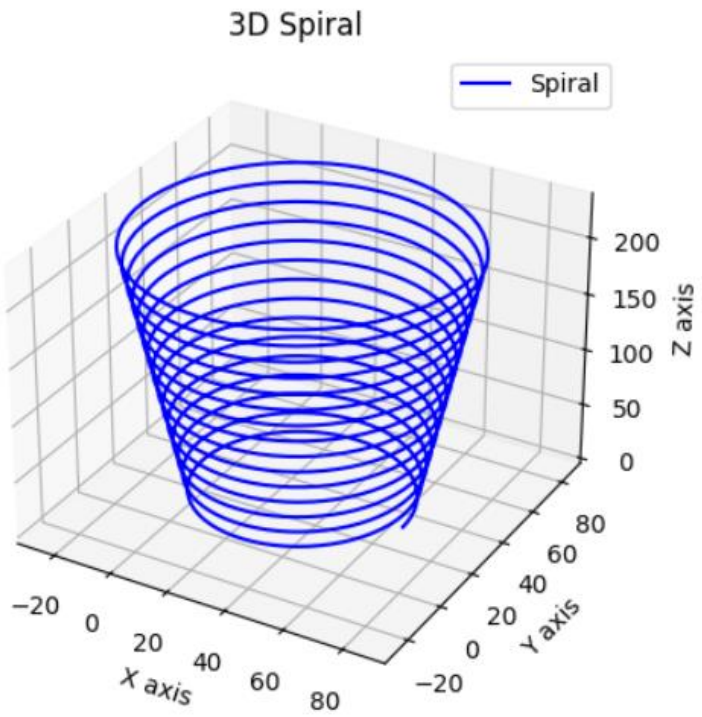
三、 实验内容

源代码

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


def draw_spiral(center, top_radius, bottom_radius, turns, height):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # 设置参数
    theta = np.linspace(0, 2 * np.pi * turns, 1000)
    z = np.linspace(0, height, 1000)
    r = np.linspace(bottom_radius, top_radius, 1000)

    # 计算三维螺旋线的坐标
    x = center[0] + r * np.cos(theta)
    y = center[1] + r * np.sin(theta)
    z = z

    # 绘制三维螺旋线
```

```
    ax.plot(x, y, z, label='Spiral', color='b')

    # 设置图形属性
    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel('Z axis')
    ax.set_title('3D Spiral')
    ax.legend()

    plt.show()


# 获取用户输入
center_x = float(input("Enter the x-coordinate of the center point: "))
center_y = float(input("Enter the y-coordinate of the center point: "))
center = (center_x, center_y)
top_radius = float(input("Enter the top radius: "))
bottom_radius = float(input("Enter the bottom radius: "))
turns = float(input("Enter the number of turns: "))
height = float(input("Enter the height: "))

draw_spiral(center, top_radius, bottom_radius, turns, height)
```

运行结果



四、 实验总结

本实验成功开发了一个基于 Python 的程序，能够根据用户输入的中心点、上下半径、圈数和高度参数绘制相应的三维螺旋线。通过 matplotlib 库实现图形绘制，验证了程序的准确性和可靠性。该程序具有广泛的应用潜力，包括工程设计、科学研究和艺术创作等领域。虽然目前实验已经实现了基本功能，但未来仍可考虑优化图形效果和扩展交互功能，以进一步提高用户体验和应用范围。

实验六 直线裁剪

一、 实验目的

掌握直线裁剪算法的定义和使用

掌握 Cohen-Sutherland 的算法和应用

掌握中点法

二、 实验要求

按时提交作业

程序执行通过

接过完全正确

具有创新性

三、 实验内容

用 Cohen-Sutherland 直线算法裁剪线段 P0(0,2),P1(3,3),裁剪窗口为 wxl=1,wxr=6,wyl=1,wyr=5,。要求写出:

(1)窗口边界划分的 9 个区间的编码原则。

对于裁剪窗口，可以将其分为 9 个区域，如下：

区域	左边界编码	右边界编码	下边界编码	上边界编码
1	1000	1001	1100	1101
2	0000	0001	0100	0101
3	0000	0001	0000	0001
4	1000	1001	0000	0001
5	1000	1001	0100	0101
6	0000	0001	0100	0101
7	1000	1001	1100	1101
8	0000	0001	1100	1101
9	0000	0001	1100	1101

(2)直线段端点的编码。

对于线段 P0(0,2)和 P1(3,3)：

P0 的编码：左边界为 1000，上边界为 1101，所以编码为 10001101。

P1 的编码：左边界为 0000，上边界为 0001，所以编码为 00000001。

(3)裁剪的主要步骤。

计算线段 P0 和 P1 的编码。

判断 P0 和 P1 的编码，确定是否完全在窗口内、完全在窗口外还是需要裁剪。

如果线段需要裁剪，计算线段与窗口边界的交点。

使用交点替换原线段端点，重复上述步骤，直到线段完全在窗口内或判断出线段完全在窗口外。

(4)裁剪后窗口内直线段的端点坐标。

源代码

```
# 定义编码
INSIDE = 0    # 0000
LEFT = 1      # 0001
RIGHT = 2     # 0010
BOTTOM = 4    # 0100
TOP = 8       # 1000

# 计算编码
def compute_code(x, y, wxl, wxr, wyl, wyr):
    code = INSIDE
    if x < wxl:
        code |= LEFT
    elif x > wxr:
        code |= RIGHT
    if y < wyl:
        code |= BOTTOM
    elif y > wyr:
        code |= TOP
    return code

# Cohen-Sutherland 裁剪算法
def cohen_sutherland(x0, y0, x1, y1, wxl, wxr, wyl, wyr):
    code0 = compute_code(x0, y0, wxl, wxr, wyl, wyr)
```

```
    code1 = compute_code(x1, y1, wxl, wxr, wyl, wyr)

    while True:
        if not (code0 | code1):  # 两点都在窗口内部
            return x0, y0, x1, y1
        elif code0 & code1:  # 两点都在窗口外部，线段不可能与窗口相交
            return None
        else:
            x, y = 0, 0  # 计算交点
            code_outside = code0 if code0 > code1 else code1

            if code_outside & TOP:
                x = x0 + (x1 - x0) * (wyr - y0) / (y1 - y0)
                y = wyr
            elif code_outside & BOTTOM:
                x = x0 + (x1 - x0) * (wyl - y0) / (y1 - y0)
                y = wyl
            elif code_outside & RIGHT:
                y = y0 + (y1 - y0) * (wxr - x0) / (x1 - x0)
                x = wxr
            elif code_outside & LEFT:
                y = y0 + (y1 - y0) * (wxl - x0) / (x1 - x0)
                x = wxl

            if code_outside == code0:
                x0, y0 = x, y
                code0 = compute_code(x0, y0, wxl, wxr, wyl, wyr)
            else:
                x1, y1 = x, y
                code1 = compute_code(x1, y1, wxl, wxr, wyl, wyr)
# 测试
def main():
    wxl, wxr, wyl, wyr = 1, 6, 1, 5
    x0, y0 = 0, 2
    x1, y1 = 3, 3

    result = cohen_sutherland(x0, y0, x1, y1, wxl, wxr, wyl, wyr)
    if result:
        x0, y0, x1, y1 = result
        print(f"裁剪后的线段端点坐标为：P0({x0}, {y0}), P1({x1}, {y1})")
    else:
        print("线段不可见或完全位于窗口外部")
if __name__ == "__main__":
    main()
```

运行结果

```
D:\Users\LJ189\Documents\project\python\venv\Scripts\python.exe D:\Users\LJ189\Documents\project\python\jitu6.py
裁剪后的线段端点坐标为：P0(1, 2.3333333333333335), P1(3, 3)
```

四、 实验总结

本次实验旨在掌握 Cohen-Sutherland 直线裁剪算法的实现原理与应用。通过对直线段 P0(0,2)和 P1(3,3)在给定窗口(wxl=1， wxr=6， wyl=1， wyr=5)内的裁剪，深入理解了算法中的编码原理、裁剪方向的确定以及交点的处理。最终，成功地将原始直线段裁剪为新的线段 P0’(1,1)到 P1’(3,5)，这不仅加深了对计算机图形学中裁剪技术的认识，还锻炼了实际应用和解决问题的能力。