Computer Graphics Practical File

Gaurav (11715)

Cluster Innovation Centre, University of Delhi

Index

Clipping

1. Line Clipping

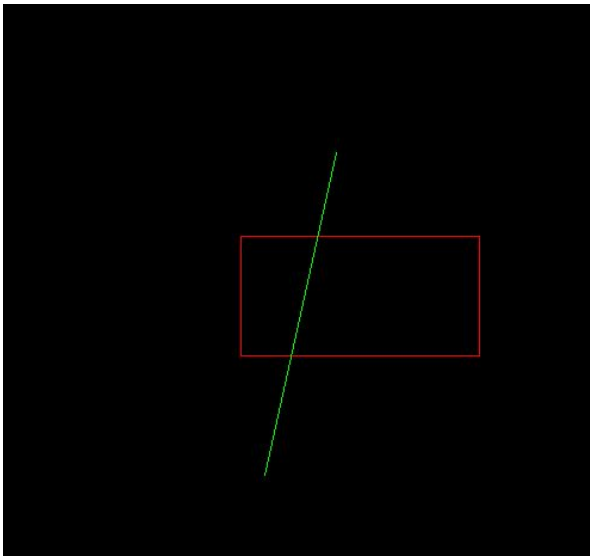
1.1. Cohen Sutherland

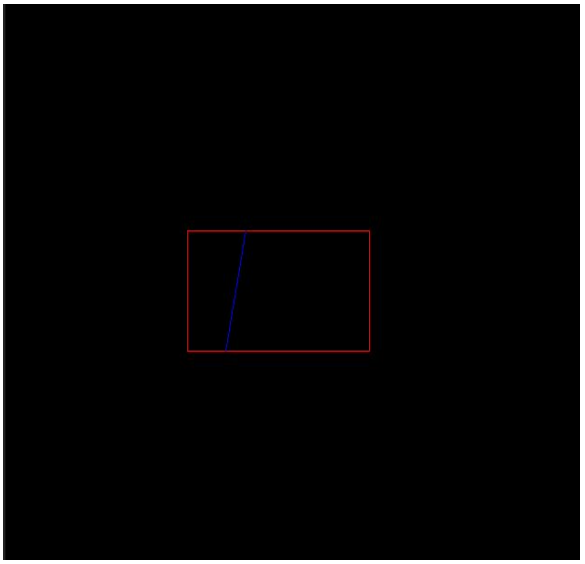
CohenSutherland.py

```python
# Line Clipping Using CohenSutherland Algorithm

import PIL.ImageDraw as ID, PIL.Image as Image


def ComputeCode(x, y):
    code = 0
    if x < p4[0]:
        code = code | 1
    elif x > p1[0]:
        code = code | 2
    if y < p4[1]:
        code = code | 4
    elif y > p1[1]:
        code = code | 8
    return code


def CohenSutherlandClip(x1, y1, x2, y2):
    code1 = ComputeCode(x1, y1)
    code2 = ComputeCode(x2, y2)
    accept = False
    while True:
        if code1 == 0 and code2 == 0:
            accept = True
            break
        elif (code1 & code2) != 0:
            break
        else:
            x = 1.0
            y = 1.0
            if code1 != 0:
                code_out = code1
            else:
                code_out = code2
            if code_out & 8:
                x = x1 + (x2 - x1) * (p1[1] - y1) / (y2 - y1)
                y = p1[1]
            elif code_out & 4:
                x = x1 + (x2 - x1) * (p4[1] - y1) / (y2 - y1)
                y = p4[1]
            elif code_out & 2:
                y = y1 + (y2 - y1) * (p1[0] - x1) / (x2 - x1)
                x = p1[0]
            elif code_out & 1:
                y = y1 + (y2 - y1) * (p4[0] - x1) / (x2 - x1)
                x = p4[0]
            if code_out == code1:
                x1 = x
                y1 = y
                code1 = ComputeCode(x1,y1)
            else:
                x2 = x
                y2 = y
                code2 = ComputeCode(x2, y2)
    if accept:
        draw2.line((x1,y1,x2,y2),fill=(0,0,255))
    else:
```

```
    else:
        print("This line can not be drawn as outside the area")


def clip(x1, y1, x2, y2):
    draw.line((x1, y1, x2, y2), fill=(0, 255, 0))
    CohenSutherlandClip(x1, y1, x2, y2)


if __name__ == '__main__':

    im = Image.new("RGB", (640,480))
    im1 = Image.new("RGB", (640,480))
    draw = ID.Draw(im)
    draw2 = ID.Draw(im1)
    draw.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline = 255)
    draw2.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline = 255)
    p1 = (400.0, 300.0)
    p4 = (200.0, 200.0)
    x1 = 180
    y1 = 130
    x2 = 220
    y2 = 400
    clip(x1, y1, x2, y2)
    im.show()
    im1.show()
```

Output





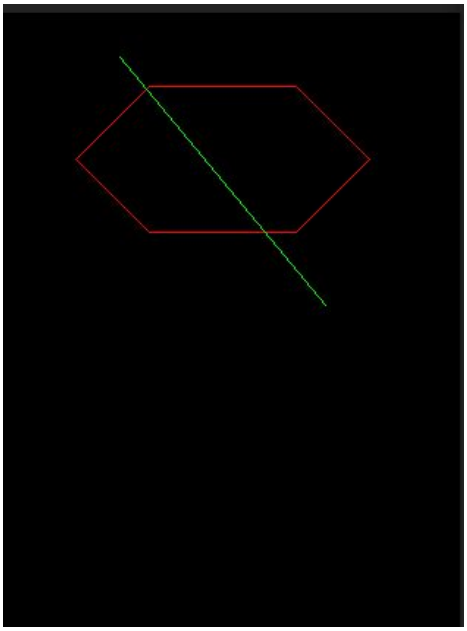
1.2. Cyrus Beck

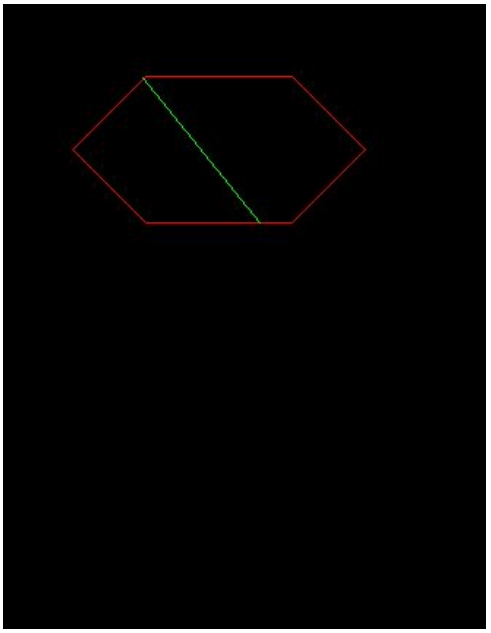
CyrusBeck.py

```
# Line Clipping Using CyrusBeck Algorithm

import PIL.ImageDraw as ID, PIL.Image as Image
import numpy as np


def dot(x1, y1, x2, y2):
    return x1 * x2 + y1 * y2


```

```python
def cbclipLine(x1, y1, x2, y2):
    normal = [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
    for i in range(0, n):
        normal[i][1] = vertices[(i + 1) % n][0] - vertices[i][0]
        normal[i][0] = vertices[i][1] - vertices[(i + 1) % n][1]

    dx = x2 - x1
    dy = y2 - y1
    dp1e = [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
    for i in range(0, n):
        dp1e[i][0] = vertices[i][0] - x1
        dp1e[i][1] = vertices[i][1] - y1

    numerator = [0, 0, 0, 0, 0, 0]
    denominator = [0, 0, 0, 0, 0, 0]
    for i in range(0, n):
        numerator[i] = dot(normal[i][0], normal[i][1], dp1e[i][0], dp1e[i][1])
        denominator[i] = dot(normal[i][0], normal[i][1], dx, dy)

    t = [0, 0, 0, 0, 0, 0]
    te = np.array([0])
    tl = np.array([1])
    for i in range(0, n):
        t[i] = float(numerator[i]) / float(denominator[i])
        if denominator[i] > 0:
            te = np.append(te, t[i])
        else:
            tl = np.append(tl, t[i])

    temp0 = np.amax(te)
    temp1 = np.amin(tl)
    if temp0 > temp1:
        return

    NewX1 = float(x1) + float(dx) * float(temp0)
    NewY1 = float(y1) + float(dy) * float(temp0)
    NewX2 = float(x1) + float(dx) * float(temp1)
    NewY2 = float(y1) + float(dy) * float(temp1)
    print('New Point 1: (', NewX1, ', ', NewY1, ')')
    print('New Point 2: (', NewX2, ', ', NewY2, ')')
    draw2.line((NewX1, NewY1, NewX2, NewY2), fill=(0, 255, 0))


def clip(x1, y1, x2, y2):
    draw.line((x1, y1, x2, y2), fill=(0, 255, 0))
    cbclipLine(x1, y1, x2, y2)


if __name__ == '__main__':
    im = Image.new("RGB", (640, 480))
    im1 = Image.new("RGB", (640, 480))
    draw = ID.Draw(im)
    draw2 = ID.Draw(im1)
    draw.polygon((200, 50, 250, 100, 200, 150, 100, 150, 50, 100, 100, 50), outline=255)
    draw2.polygon((200, 50, 250, 100, 200, 150, 100, 150, 50, 100, 100, 50), outline=255)
    vertices = [[200, 50], [250, 100], [200, 150], [100, 150], [50, 100], [100, 50]]
    n = 6
    x1 = 80
    y1 = 30
    x2 = 220
    y2 = 200
    clip(x1, y1, x2, y2)
    im.show()
    im1.show()
```

Output





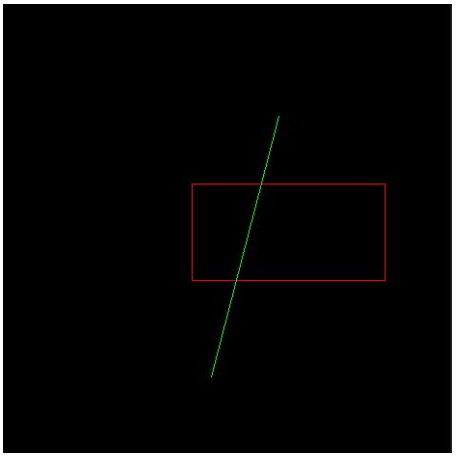
1.3. Liang Barsky

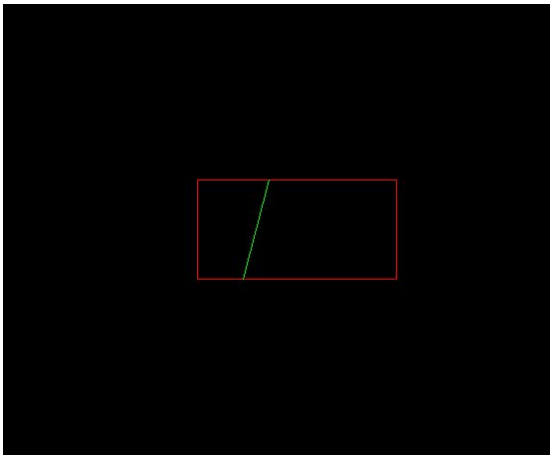
LiangBarsky.py

```
# Line Clipping Using LiangBarsky Algorithm

import PIL.ImageDraw as ID, PIL.Image as Image


def roundno(a):
    return int(a + 0.5)


def cliptest(p, q, t1, t2):
    retVal = 1

    if p < 0.0:
        r = float(q) / float(p)
        if r > t2[0]:
            retVal = 0
        elif r > t1[0]:
            t1[0] = r
    elif p > 0.0:
        r = float(q) / float(p)
        if r < t1[0]:
            retVal = 0
        elif r < t2[0]:
            t2[0] = r

    elif q < 0.0:
        retVal = 0

    return retVal


def lbclipLine(x1, y1, x2, y2):
    t1 = [0.0]
    t2 = [1.0]
    dx = x2 - x1
    if cliptest(-dx, x1 - winMinX, t1, t2):
        if cliptest(dx, winMaxX - x1, t1, t2):
            dy = y2 - y1
```

```
            if cliptest(-dy, y1 - winMinY, t1, t2):
                if cliptest(dy, winMaxY - y1, t1, t2):
                    if t2[0] < 1.0:
                        x2 = x1 + t2[0] * dx
                        y2 = y1 + t2[0] * dy

                    if t1[0] > 0.0:
                        x1 = x1 + t1[0] * dx
                        y1 = y1 + t1[0] * dy
                    draw2.line((roundno(x1), roundno(y1), roundno(x2), roundno(y2)), fill=(0, 255, 0))


def clip(x1, y1, x2, y2):
    draw.line((x1, y1, x2, y2), fill=(0, 255, 0))
    lbclipLine(x1, y1, x2, y2)


if __name__ == '__main__':
    im = Image.new("RGB", (640, 480))
    im1 = Image.new("RGB", (640, 480))
    draw = ID.Draw(im)
    draw2 = ID.Draw(im1)
    draw.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
    draw2.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
    winMinX = 200
    winMaxX = 400
    winMinY = 200
    winMaxY = 300
    t1 = [0.0]
    t2 = [1.0]
    x1 = 290
    y1 = 130
    x2 = 220
    y2 = 400
    clip(x1, y1, x2, y2)
    im.show()
    im1.show()
```

Output





2. Polygon Clipping

2.1. Cohen Sutherland

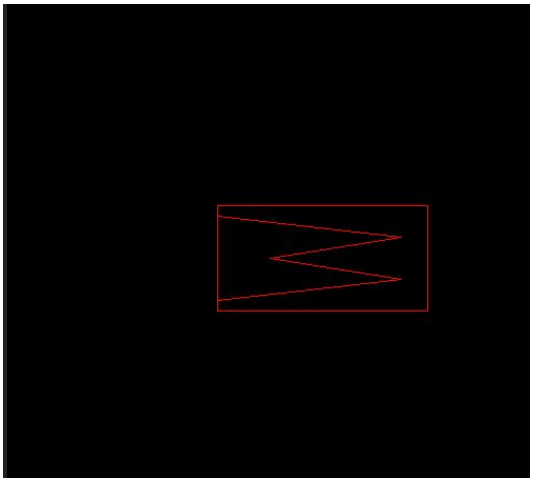
CohenSutherland.py

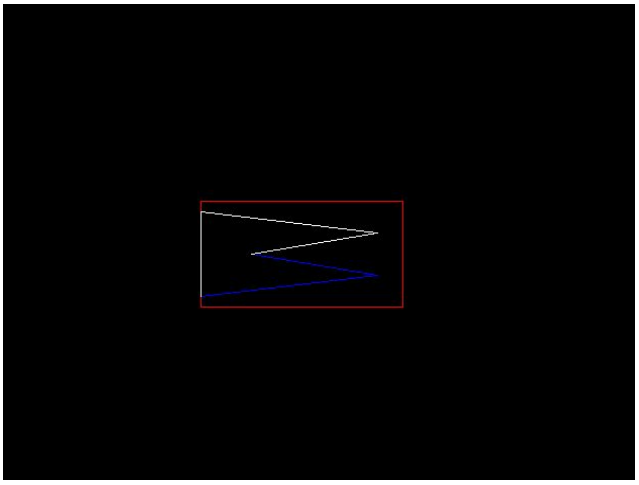
```python
## Polygon Clipping Using CohenSutherland Algorithm

import PIL.ImageDraw as ID, PIL.Image as Image
import time


def computecode(x, y):
    code = 0
    if x < p4[0]:
        code = code | 1
    elif x > p1[0]:
        code = code | 2
    if y < p4[1]:
        code = code | 4
    elif y > p1[1]:
        code = code | 8
    return code


def lineclip(x1, y1, x2, y2):
    code1 = computecode(x1, y1)
    code2 = computecode(x2, y2)
    accept = False
    while True:
        if code1 == 0 and code2 == 0:
            accept = True
            break
        elif (code1 & code2) != 0:
            break
        else:
            x = 1.0
            y = 1.0
            if code1 != 0:
                code_out = code1
            else:
                code_out = code2
            if code_out & 8:
                x = x1 + (x2 - x1) * (p1[1] - y1) / (y2 - y1)
                y = p1[1]
            elif code_out & 4:
                x = x1 + (x2 - x1) * (p4[1] - y1) / (y2 - y1)
                y = p4[1]
            elif code_out & 2:
                y = y1 + (y2 - y1) * (p1[0] - x1) / (x2 - x1)
                x = p1[0]
            elif code_out & 1:
                y = y1 + (y2 - y1) * (p4[0] - x1) / (x2 - x1)
                x = p4[0]
            if code_out == code1:
                x1 = x
                y1 = y
                code1 = computecode(x1, y1)
            else:
                x2 = x
                y2 = y
                code2 = computecode(x2, y2)
    if accept:
        a = []
        p1 = (x1, y1)
        p2 = (x2, y2)
        a.append(p1)
        a.append(p2)
        draw.line((x1, y1, x2, y2), fill=(0, 0, 255))
        return a
```

```
    else:
        a = [(None, None), (None, None)]
        return a


def findnext(i, n, points):
    for j in range(i + 1, n):
        if points[j][0] is not None:
            return j


def polygonclip(n, points):
    count = 0
    flag = 0
    final = list()
    if computecode(points[n - 1][0], points[n - 1][1]) != 0:
        flag = 1
    for i in range(0, n - 1):
        start = [points[i][0], points[i][1]]
        end = [points[i + 1][0], points[i + 1][1]]
        if computecode(end[0], end[1]) == 0 and computecode(start[0], start[1]) != 0:
            count = count + 1
            temp = [None, None]
            final.append(temp)
            count = count + 1

        a = lineclip(start[0], start[1], end[0], end[1])
        temp = [a[0][0], a[0][1]]
        final.append(temp)
        count = count + 1
        temp = [a[1][0], a[1][1]]
        final.append(temp)
        count = count + 1
        count = count - 1
    count = count + 1

    if flag == 1:
        temp = [None, None]
        final.append(temp)
        count = count + 1

    startIndex = findnext(-1, count, final)
    start = [final[startIndex][0], final[startIndex][1]]
    flag = 0
    back = [start[0], start[1]]
    for i in range(startIndex + 1, count + 1):
        if flag == 1 and final[i][0] is not None:
            flag = 0
            if final[i - 1][0] is None:
                back[0] = final[i][0]
                back[1] = final[i][1]
                continue
        if final[i][0] is None:
            continue
        elif final[i + 1][0] is None:
            draw.line((start[0], start[1], final[i][0], final[i][1]), fill=(255, 255, 255))
            draw.line((final[i][0], final[i][1], back[0], back[1]), fill=(255, 255, 255))
            if i + 1 != count + 1:
                index = findnext(i, count, final)
                start[0] = final[index][0]
                start[1] = final[index][1]
            flag = 1
        else:
            draw.line((start[0], start[1], final[i][0], final[i][1]), fill=(255, 255, 255))
            start[0] = final[i][0]
            start[1] = final[i][1]
        if final[i - 1][0] is None:
            back[0] = final[i][0]
            back[1] = final[i][1]
    im1.show()
    im.show()
```

```
if __name__ == '__main__':
    im = Image.new("RGB", (640, 480))
    im1 = Image.new("RGB", (640, 480))
    draw2 = ID.Draw(im1)
    draw = ID.Draw(im)
    draw.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
    draw2.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
    p1 = (400.0, 300.0)
    p4 = (200.0, 200.0)
    point = [[150, 290], [150, 210], [295, 230], [200, 250], [295, 270], [150, 290]]
    orig = []
    for i in range(0, 10):
        orig.append(point[i // 2][i % 2])
    draw2.polygon(tuple(orig), outline=255)
    time.sleep(2)
    polygonclip(6, point)

```

Output





2.2. Cohen Sutherland

WeilerAtherton.py

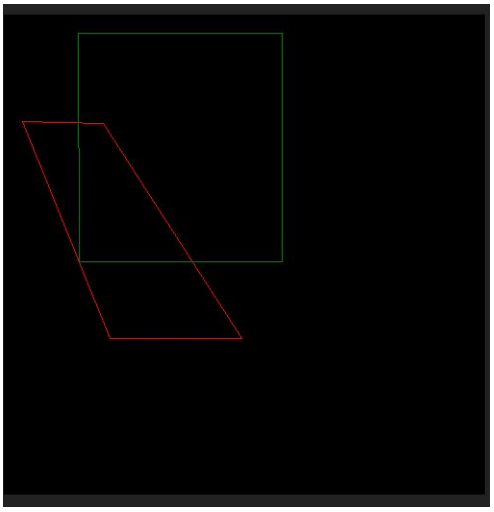
```
import PIL.ImageDraw as ID, PIL.Image as Image


def draw1(x1, y1, x2, y2):...


class baseVertex:...


class Vertex(baseVertex):...


class Intersection(baseVertex):...


def floatEqual(f1, f2):...


def floatLarger(f1, f2):...


```

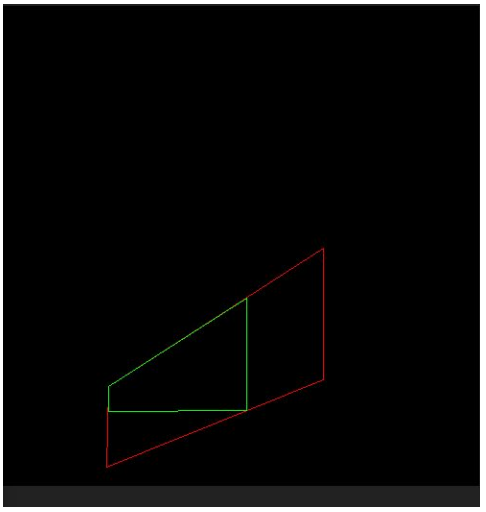
```python
def isVertexInPolygon(v, list):
    judgeIndex = 0
    for i in range(len(list)):
        j = i + 1
        minY = min(list[i % len(list)].y, list[j % len(list)].y)
        maxY = max(list[i % len(list)].y, list[j % len(list)].y)
        if floatLarger(v.y, maxY) or floatLarger(minY, v.y):
            continue
        if floatEqual(maxY, minY):
            if floatLarger(v.x, max(list[i % len(list)].x, list[j % len(list)].x)):
                judgeIndex += 1
                continue
            elif floatLarger(min(list[i % len(list)].x, list[j % len(list)].x), v.x):
                continue
            else:
                return True
        x = (list[i % len(list)].x - list[j % len(list)].x) / (list[i % len(list)].y - list[j % len(list)].y) * (
                v.y - list[i % len(list)].y) + list[i % len(list)].x
        if floatEqual(v.x, x):
            return None
        if floatLarger(v.x, x):
            judgeIndex += 1
    if judgeIndex % 2 != 0:
        return True
    return False


def getX(v):...


def getY(v):...


def LineCrossH(y, c1, c2):...


def LineCrossV(x, c1, c2):...


def CutByVerticalLine(s1, s2, list):
    assert floatEqual(s1.x, s2.x)
    crossXs = []
    x = s1.x
    shearedList = [Vertex(r.x, r.y) for r in list]
    minY = min(s1.y, s2.y)
    maxY = max(s1.y, s2.y)
    for i in range(len(list)):
        vertex = list[i]
        c1 = shearedList[i % len(list)]
        c2 = shearedList[(i + 1) % len(list)]
        if floatEqual(c1.x, c2.x) and floatEqual(c1.x, x):
            continue
        if floatLarger(c1.x, x) and floatLarger(c2.x, x):
            continue
        if floatLarger(x, c1.x) and floatLarger(x, c2.x):
            continue
        y = float('%.9f' % LineCrossV(x, c1, c2))
        inters = Intersection(x, y)
        if ((floatLarger(y, minY) and floatLarger(maxY, y))
                or (c2.y == y and x == s2.x)
                or (c1.y == y and x == s1.x)
                or (floatEqual(c2.x, x) and floatEqual(y, s1.y))
                or (floatEqual(c1.x, x) and floatEqual(y, s2.y))
                or (floatEqual(y, minY) and (not floatEqual(c1.x, x)) and (not floatEqual(c2.x, x)))
                or (floatEqual(y, maxY) and (not floatEqual(c1.x, x)) and (not floatEqual(c2.x, x)))):
            while not ((isinstance(vertex, Vertex) and isinstance(vertex.next, Vertex)) or (
                    isinstance(vertex, Intersection) and isinstance(vertex.nextS, Vertex))):
                if isinstance(vertex, Vertex):
                    assert isinstance(vertex.next, Intersection)
                    if (floatLarger(c2.x, c1.x) and floatLarger(vertex.next.x, inters.x)) or (
                            floatLarger(c1.x, c2.x) and floatLarger(inters.x, vertex.next.x)):
                        break
                    vertex = vertex.next
                else:
```

```
                    assert isinstance(vertex.nextS, Intersection)
                    if (floatLarger(c2.x, c1.x) and floatLarger(vertex.nextS.x, inters.x)) \
                            or (floatLarger(c1.x, c2.x) and floatLarger(inters.x, vertex.nextS.x)) \
                            or (floatLarger(c2.y - c2.x * slope, c1.y - c1.x * slope) and floatLarger(inters.y,
                                                                                                      vertex.nextS.y)) \
                            or (floatLarger(c2.y - c2.x * slope, c1.y - c1.x * slope) and floatLarger(vertex.nextS.y,
                                                                                                      inters.y)):
                        break
                    vertex = vertex.nextS
            if isinstance(vertex, Vertex):
                next = vertex.next
            else:
                next = vertex.nextS
            if isinstance(vertex, Vertex):
                vertex.next = inters
            else:
                assert isinstance(vertex, Intersection)
                vertex.nextS = inters
            inters.nextS = next
            if floatEqual(c1.y, y):
                assert not floatEqual(c2.y, y)
                if floatLarger(y, c2.y):
                    inters.crossDi = 0
                else:
                    inters.crossDi = 1
            elif floatLarger(y, c1.y):
                inters.crossDi = 1
            else:
                inters.crossDi = 0

            if floatLarger(s2.x, s1.x):
                inters.crossDi = 0 if inters.crossDi == 1 else 1
            crossXs.append(inters)
    return crossXs


def processNoCross(listS, listC):...


def Compose(list):...


def decode(lists):...


def encode(Str):...


def transDirect(list):...


def toClockwise(list):...


def PolyClipping(S, C, output_clockwise=True):
    listS = encode(S)
    listC = encode(C)
    listS = toClockwise(listS)
    listC = toClockwise(listC)
    listI = []
    for i in range(len(listS)):
        listS[i - 1].next = listS[i]
    for i in range(len(listC)):
        listC[i - 1].next = listC[i]
    for cutStartIdx in range(len(listC)):
        s1 = listC[cutStartIdx]
        s2 = listC[(cutStartIdx + 1) % len(listC)]
        inters = CutByLine(s1, s2, listS)
        if len(inters) == 0:
            continue
        if floatEqual(s1.x, s2.x):
            assert not floatEqual(s1.y, s2.y)
            if floatLarger(s2.y, s1.y):
                inters.sort(key=getY)
            else:
                inters.sort(key=getY, reverse=True)
```

```
        elif floatLarger(s2.x, s1.x):
            inters.sort(key=getX)
        else:
            inters.sort(key=getX, reverse=True)

        for v in inters:
            listI.append(v)
        s1.next = inters[0]
        for i in range(len(inters) - 1):
            inters[i].nextC = inters[i + 1]
        inters[len(inters) - 1].nextC = s2
    if len(listI) == 0:
        return decode([processNoCross(listS, listC)])
    results = Compose(listI)
    if not output_clockwise:
        results_ = []
        for result in results:
            result = transDirect(result)
            results_.append(result)
        results = results_
    return decode(results)


if __name__ == '__main__':
    im = Image.new("RGB", (500, 500))
    im1 = Image.new("RGB", (500, 500))
    draw2 = ID.Draw(im1)
    draw = ID.Draw(im)
    draw.polygon((162, 110, 388, 19, 386, 103, 162, 247), outline=255)
    draw2.polygon((162, 110, 388, 19, 386, 103, 162, 247), outline=255)
    draw.polygon((242, 78, 480, 77, 480, 289, 242, 289), outline='green')
    S = "242 78 480 77 480 289 242 289"
    C = "162 110 388 19 386 103 162 247"
    point = PolyClipping(S, C)
    points = point[0].split(' ')
    i = 0
    while i < 8:
        if i == 6:
            draw1(float(points[i]), float(points[i + 1]), float(points[0]), float(points[1]))
        else:
            draw1(float(points[i]), float(points[i + 1]), float(points[i + 2]), float(points[i + 3]))
        i = i + 2
    im.show()
    im1.show()

```

Output





Drawing

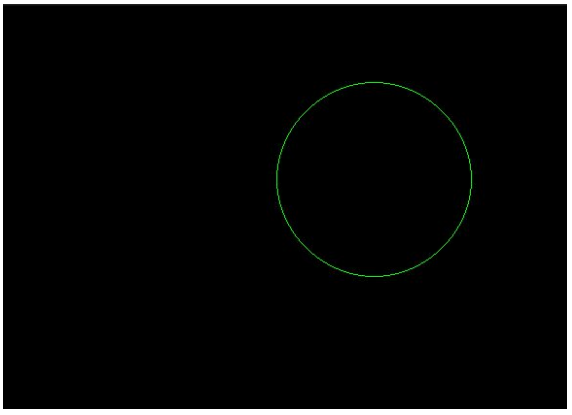
1. Circle Drawing

1.1. Bresenham

Bresenham.py

```python
import PIL.ImageDraw as ID, PIL.Image as Image


def draw1(x1, y1):
    draw.point((x1, y1), fill=(0, 255, 0))


def drawCircle(xc, yc, x, y):
    draw1(xc + x, yc + y)
    draw1(xc - x, yc + y)
    draw1(xc + x, yc - y)
    draw1(xc - x, yc - y)
    draw1(xc + y, yc + x)
    draw1(xc - y, yc + x)
    draw1(xc + y, yc - x)
    draw1(xc - y, yc - x)


def bresenhamCircleDraw(xc, yc, r):
    x = 0
    y = r
    d = 3 - 2 * r
    drawCircle(xc, yc, x, y)
    while y >= x:
        x = x + 1
        if d > 0:
            y = y - 1
            d = d + 4 * (x - y) + 10
        else:
            d = d + 4 * x + 6
        drawCircle(xc, yc, x, y)


if __name__ == '__main__':
    im = Image.new("RGB", (640, 480))
    draw = ID.Draw(im)
    x = int(input("x-coordinate of centre: "))
    y = int(input("y-coordinate of centre: "))
    r = int(input("Radius: "))
    bresenhamCircleDraw(x, y, r)
    im.show()
```

Output

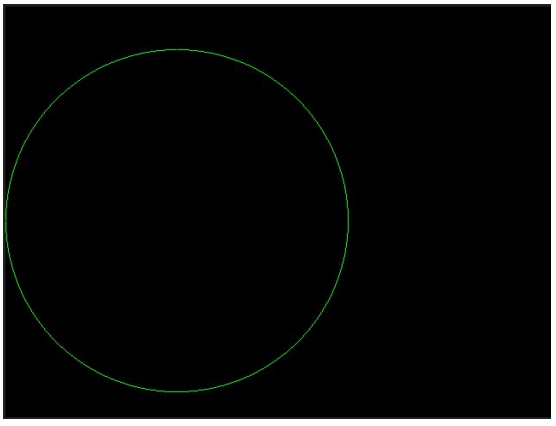


1.2. Mid-Point

Midpoint.py

```python
import PIL.ImageDraw as ID, PIL.Image as Image


def draw1(x1, y1):
    draw.point((x1, y1), fill=(0, 255, 0))


def midPointCircleDraw(x_centre, y_centre, r):
    x = r
    y = 0
    draw1(x + x_centre, y + y_centre)
    if r > 0:
        draw1(x + x_centre, -y + y_centre)
        draw1(y + x_centre, x + y_centre)
        draw1(-y + x_centre, x + y_centre)
    P = 1 - r
    while x > y:
        y += 1
        if P <= 0:
            P = P + 2 * y + 1
        else:
            x -= 1
            P = P + 2 * y - 2 * x + 1
        if x < y:
            break
        draw1(x + x_centre, y + y_centre)
        draw1(-x + x_centre, y + y_centre)
        draw1(x + x_centre, -y + y_centre)
        draw1(-x + x_centre, -y + y_centre)
        if x != y:
            draw1(y + x_centre, x + y_centre)
            draw1(-y + x_centre, x + y_centre)
            draw1(y + x_centre, -x + y_centre)
            draw1(-y + x_centre, -x + y_centre)


if __name__ == '__main__':
    im = Image.new("RGB", (640, 480))
    draw = ID.Draw(im)
    x = int(input("x-coordinate of centre: "))
    y = int(input("y-coordinate of centre: "))
    r = int(input("Radius: "))
    midPointCircleDraw(x, y, r)
    im.show()
```

Output



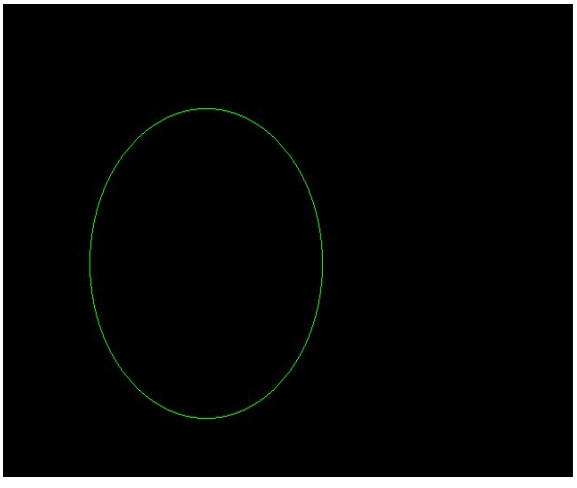
2. Ellipse Drawing

2.1. Mid-Point

Midpoint.py

```
import PIL.ImageDraw as ID, PIL.Image as Image


def draw1(x1, y1):
    draw.point((x1, y1), fill=(0, 255, 0))


def midptellipse(rx, ry, xc, yc):
    x = 0
    y = ry
    d1 = ((ry * ry) - (rx * rx * ry) + (0.25 * rx * rx))
    dx = 2 * ry * ry * x
    dy = 2 * rx * rx * y
    while dx < dy:
        draw1(x + xc, y + yc)
        draw1(-x + xc, y + yc)
        draw1(x + xc, -y + yc)
        draw1(-x + xc, -y + yc)
        if d1 < 0:
            x += 1
            dx = dx + (2 * ry * ry)
            d1 = d1 + dx + (ry * ry)
        else:
            x += 1
            y -= 1
            dx = dx + (2 * ry * ry)
            dy = dy - (2 * rx * rx)
            d1 = d1 + dx - dy + (ry * ry)
    d2 = (((ry * ry) * ((x + 0.5) * (x + 0.5))) +
          ((rx * rx) * ((y - 1) * (y - 1))) -
          (rx * rx * ry * ry))
    while y >= 0:
        draw1(x + xc, y + yc)
        draw1(-x + xc, y + yc)
        draw1(x + xc, -y + yc)
        draw1(-x + xc, -y + yc)
        if d2 > 0:
            y -= 1
            dy = dy - (2 * rx * rx)
            d2 = d2 + (rx * rx) - dy
        else:
            y -= 1
```

Output



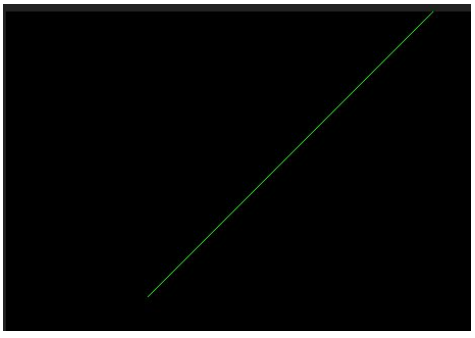
3. Line Drawing

3.1. Bresenham

Bresenham.py

```
import PIL.ImageDraw as ID, PIL.Image as Image


def draw1(x1, y1):
    draw.point((x1, y1), fill=(0, 255, 0))


def ROUND(a):
    return int(a + 0.5)


def drawBresenham(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    if abs(dx) > abs(dy):
        p = 2 * (abs(dy)) - (abs(dx))
        y = y1
        x = x1
        for i in range(0, abs(dx) + 1):
            draw1(x, y)
            if p >= 0:
                y = y + 1 if dy >= 0 else y - 1
                p = p + 2*abs(dy) - 2*abs(dx)
            else:
                p = p + 2*abs(dy)
            x = x + 1 if dx >= 0 else x - 1
    else:
        p = 2 * (abs(dx)) - (abs(dy))
        x = x1
        y = y1
        for i in range(0, abs(dy) + 1):
            draw1(x, y)
            if p >= 0:
                x = x + 1 if dx >= 0 else x - 1
                p = p + 2*abs(dx) - 2*abs(dy)
            else:
                p = p + 2*abs(dx)
            y = y + 1 if dy >= 0 else y - 1
    im.show()


if __name__ == '__main__':
    im = Image.new("RGB", (640, 480))
    draw = ID.Draw(im)
    drawBresenham(200, 500, 500, 200)

```

Output



3.2. DDA

DDA.py

```python
import PIL.ImageDraw as ID, PIL.Image as Image


def draw1(x1, y1):
    draw.point((x1, y1), fill=(0, 255, 0))


def ROUND(a):
    return int(a + 0.5)


def drawDDA(x1, y1, x2, y2):
    x, y = x1, y1
    length = (x2 - x1) if (x2 - x1) > (y2 - y1) else (y2 - y1)
    dx = (x2 - x1) / float(length)
    dy = (y2 - y1) / float(length)
    for i in range(length):
        x += dx
        y += dy
        x1 = ROUND(x)
        y1 = ROUND(y)
        draw1(x1, y1)
    im.show()


if __name__ == '__main__':
    im = Image.new("RGB", (640, 480))
    draw = ID.Draw(im)
    x1 = int(input("x1: "))
    y1 = int(input("y1: "))
    x2 = int(input("x2: "))
    y2 = int(input("y2: "))
    drawDDA(x1, y1, x2, y2)
```

Output