

Computer Graphics Practical File

Gaurav (11715)

Cluster Innovation Centre, University of Delhi

Index

Clipping	1
Line Clipping	1
Cohen Sutherland	1
Cyrus Beck	2
Liang Barsky	4
Polygon Clipping	6
Cohen Sutherland	6
Cohen Sutherland	8
Drawing	12
Circle Drawing	12
Bresenham	12
Mid-Point	13
Ellipse Drawing	14
Mid-Point	14
Line Drawing	15
Bresenham	15
DDA	15

Clipping

1. Line Clipping

1.1. Cohen Sutherland

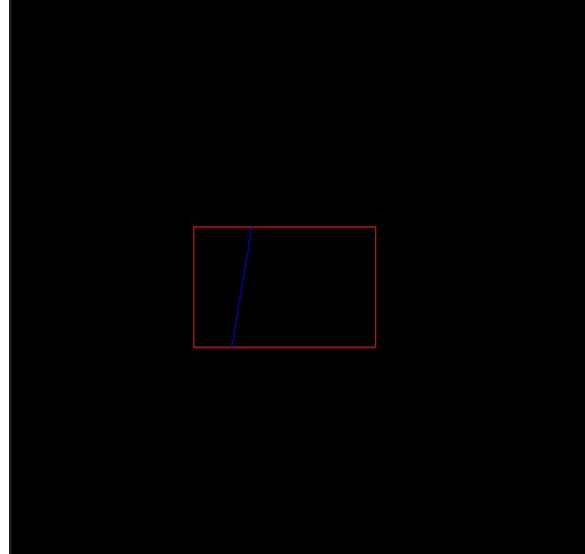
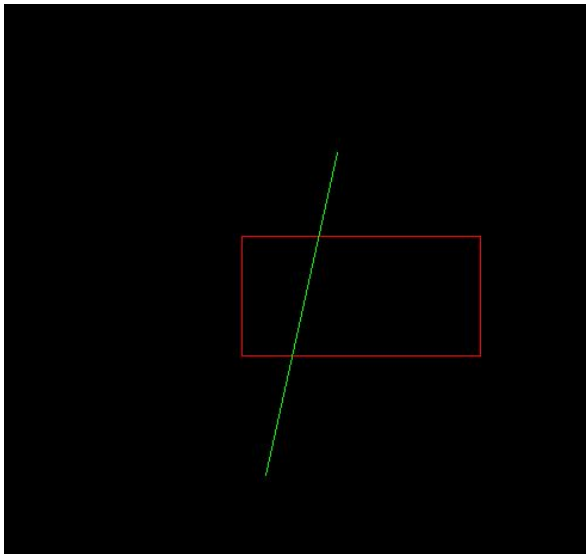
```
CohenSutherland.py
1 # Line Clipping Using CohenSutherland Algorithm
2
3 import PIL.ImageDraw as ID, PIL.Image as Image
4
5
6 def ComputeCode(x, y):
7     code = 0
8     if x < p4[0]:
9         code = code | 1
10    elif x > p1[0]:
11        code = code | 2
12    if y < p4[1]:
13        code = code | 4
14    elif y > p1[1]:
15        code = code | 8
16    return code
17
18
19 def CohenSutherlandClip(x1, y1, x2, y2):
20     code1 = ComputeCode(x1, y1)
21     code2 = ComputeCode(x2, y2)
22     accept = False
23     while True:
24         if code1 == 0 and code2 == 0:
25             accept = True
26             break
27         elif (code1 & code2) != 0:
28             break
29         else:
30             x = 1.0
31             y = 1.0
32             if code1 != 0:
33                 code_out = code1
34             else:
35                 code_out = code2
36             if code_out & 8:
37                 x = x1 + (x2 - x1) * (p1[1] - y1) / (y2 - y1)
38                 y = p1[1]
39             elif code_out & 4:
40                 x = x1 + (x2 - x1) * (p4[1] - y1) / (y2 - y1)
41                 y = p4[1]
42             elif code_out & 2:
43                 y = y1 + (y2 - y1) * (p1[0] - x1) / (x2 - x1)
44                 x = p1[0]
45             elif code_out & 1:
46                 y = y1 + (y2 - y1) * (p4[0] - x1) / (x2 - x1)
47                 x = p4[0]
48             if code_out == code1:
49                 x1 = x
50                 y1 = y
51                 code1 = ComputeCode(x1, y1)
52             else:
53                 x2 = x
54                 y2 = y
55                 code2 = ComputeCode(x2, y2)
56     if accept:
57         draw2.line((x1, y1, x2, y2), fill=(0, 0, 255))
58     else:
```

```

58     else:
59         print("This line can not be drawn as outside the area")
60
61
62     def clip(x1, y1, x2, y2):
63         draw.line((x1, y1, x2, y2), fill=(0, 255, 0))
64         CohenSutherlandClip(x1, y1, x2, y2)
65
66
67     if __name__ == '__main__':
68
69         im = Image.new("RGB", (640, 480))
70         im1 = Image.new("RGB", (640, 480))
71         draw = ID.Draw(im)
72         draw2 = ID.Draw(im1)
73         draw.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
74         draw2.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
75         p1 = (400.0, 300.0)
76         p4 = (200.0, 200.0)
77         x1 = 180
78         y1 = 130
79         x2 = 220
80         y2 = 400
81         clip(x1, y1, x2, y2)
82         im.show()
83         im1.show()

```

Output



1.2. Cyrus Beck

```

CyrusBeck.py
1  # Line Clipping Using CyrusBeck Algorithm
2
3  import PIL.ImageDraw as ID, PIL.Image as Image
4  import numpy as np
5
6
7  def dot(x1, y1, x2, y2):
8      return x1 * x2 + y1 * y2
9
10

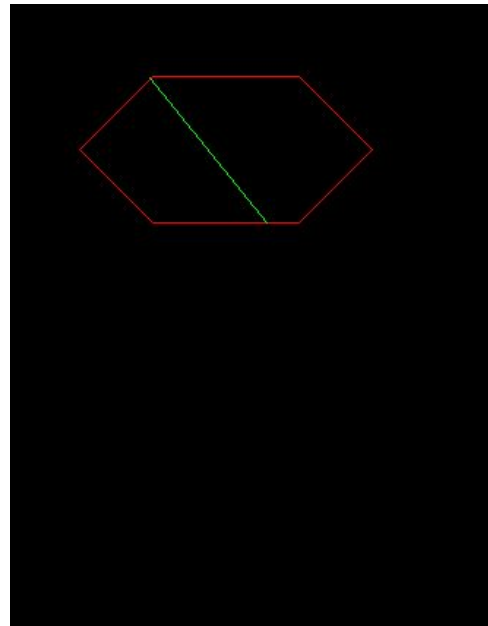
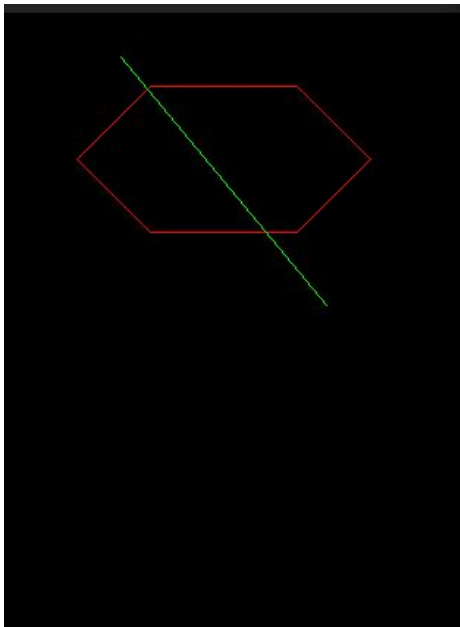
```

```

11 def cbclipLine(x1, y1, x2, y2):
12     normal = [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
13     for i in range(0, n):
14         normal[i][1] = vertices[(i + 1) % n][0] - vertices[i][0]
15         normal[i][0] = vertices[i][1] - vertices[(i + 1) % n][1]
16
17     dx = x2 - x1
18     dy = y2 - y1
19     dple = [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
20     for i in range(0, n):
21         dple[i][0] = vertices[i][0] - x1
22         dple[i][1] = vertices[i][1] - y1
23
24     numerator = [0, 0, 0, 0, 0, 0]
25     denominator = [0, 0, 0, 0, 0, 0]
26     for i in range(0, n):
27         numerator[i] = dot(normal[i][0], normal[i][1], dple[i][0], dple[i][1])
28         denominator[i] = dot(normal[i][0], normal[i][1], dx, dy)
29
30     t = [0, 0, 0, 0, 0, 0]
31     te = np.array([0])
32     tl = np.array([1])
33     for i in range(0, n):
34         t[i] = float(numerator[i]) / float(denominator[i])
35         if denominator[i] > 0:
36             te = np.append(te, t[i])
37         else:
38             tl = np.append(tl, t[i])
39
40     temp0 = np.amax(te)
41     temp1 = np.amin(tl)
42     if temp0 > temp1:
43         return
44
45     NewX1 = float(x1) + float(dx) * float(temp0)
46     NewY1 = float(y1) + float(dy) * float(temp0)
47     NewX2 = float(x1) + float(dx) * float(temp1)
48     NewY2 = float(y1) + float(dy) * float(temp1)
49     print('New Point 1: (' + str(NewX1) + ', ' + str(NewY1) + ')')
50     print('New Point 2: (' + str(NewX2) + ', ' + str(NewY2) + ')')
51     draw2.line((NewX1, NewY1, NewX2, NewY2), fill=(0, 255, 0))
52
53
54 def clip(x1, y1, x2, y2):
55     draw.line((x1, y1, x2, y2), fill=(0, 255, 0))
56     cbclipLine(x1, y1, x2, y2)
57
58
59 if __name__ == '__main__':
60     im = Image.new("RGB", (640, 480))
61     im1 = Image.new("RGB", (640, 480))
62     draw = ID.Draw(im)
63     draw2 = ID.Draw(im1)
64     draw.polygon((200, 50, 250, 100, 200, 150, 100, 150, 50, 100, 100, 50), outline=255)
65     draw2.polygon((200, 50, 250, 100, 200, 150, 100, 150, 50, 100, 100, 50), outline=255)
66     vertices = [[200, 50], [250, 100], [200, 150], [100, 150], [50, 100], [100, 50]]
67     n = 6
68     x1 = 80
69     y1 = 30
70     x2 = 220
71     y2 = 200
72     clip(x1, y1, x2, y2)
73     im.show()
74     im1.show()

```

Output



1.3. Liang Barsky

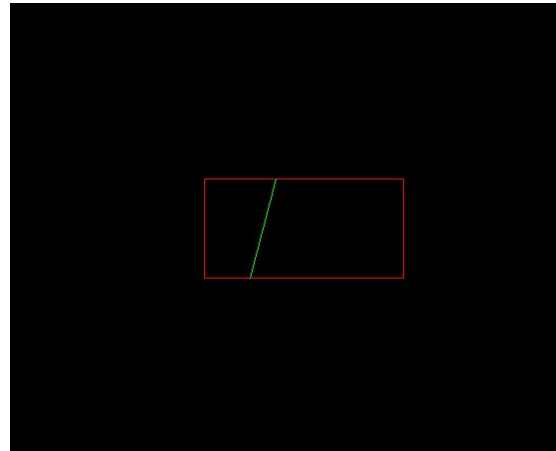
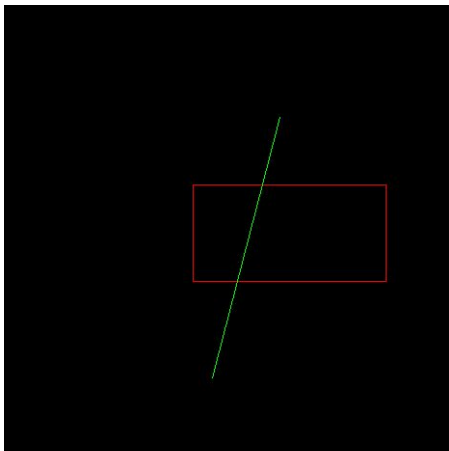
```
LiangBarsky.py ×
1  # Line Clipping Using LiangBarsky Algorithm
2
3  import PIL.ImageDraw as ID, PIL.Image as Image
4
5
6  def roundno(a):
7      return int(a + 0.5)
8
9
10 def cliptest(p, q, t1, t2):
11     retVal = 1
12
13     if p < 0.0:
14         r = float(q) / float(p)
15         if r > t2[0]:
16             retVal = 0
17         elif r > t1[0]:
18             t1[0] = r
19     elif p > 0.0:
20         r = float(q) / float(p)
21         if r < t1[0]:
22             retVal = 0
23         elif r < t2[0]:
24             t2[0] = r
25
26     elif q < 0.0:
27         retVal = 0
28
29     return retVal
30
31
32 def lbclipLine(x1, y1, x2, y2):
33     t1 = [0.0]
34     t2 = [1.0]
35     dx = x2 - x1
36     if cliptest(-dx, x1 - winMinX, t1, t2):
37         if cliptest(dx, winMaxX - x1, t1, t2):
38             dy = y2 - y1
```

```

39         if cliptest(-dy, y1 - winMinY, t1, t2):
40             if cliptest(dy, winMaxY - y1, t1, t2):
41                 if t2[0] < 1.0:
42                     x2 = x1 + t2[0] * dx
43                     y2 = y1 + t2[0] * dy
44
45                 if t1[0] > 0.0:
46                     x1 = x1 + t1[0] * dx
47                     y1 = y1 + t1[0] * dy
48             draw2.line((roundno(x1), roundno(y1), roundno(x2), roundno(y2)), fill=(0, 255, 0))
49
50
51 def clip(x1, y1, x2, y2):
52     draw.line((x1, y1, x2, y2), fill=(0, 255, 0))
53     lbclipLine(x1, y1, x2, y2)
54
55
56 if __name__ == '__main__':
57     im = Image.new("RGB", (640, 480))
58     im1 = Image.new("RGB", (640, 480))
59     draw = ID.Draw(im)
60     draw2 = ID.Draw(im1)
61     draw.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
62     draw2.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
63     winMinX = 200
64     winMaxX = 400
65     winMinY = 200
66     winMaxY = 300
67     t1 = [0.0]
68     t2 = [1.0]
69     x1 = 290
70     y1 = 130
71     x2 = 220
72     y2 = 400
73     clip(x1, y1, x2, y2)
74     im.show()
75     im1.show()

```

Output



2. Polygon Clipping

2.1. Cohen Sutherland

```
CohenSutherland.py
1  ## Polygon Clipping Using CohenSutherland Algorithm
2
3  import PIL.ImageDraw as ID, PIL.Image as Image
4  import time
5
6
7  def computecode(x, y):
8      code = 0
9      if x < p4[0]:
10         code = code | 1
11     elif x > p1[0]:
12         code = code | 2
13     if y < p4[1]:
14         code = code | 4
15     elif y > p1[1]:
16         code = code | 8
17     return code
18
19
20 def lineclip(x1, y1, x2, y2):
21     code1 = computecode(x1, y1)
22     code2 = computecode(x2, y2)
23     accept = False
24     while True:
25         if code1 == 0 and code2 == 0:
26             accept = True
27             break
28         elif (code1 & code2) != 0:
29             break
30         else:
31             x = 1.0
32             y = 1.0
33             if code1 != 0:
34                 code_out = code1
35             else:
36                 code_out = code2
37             if code_out & 8:
38                 x = x1 + (x2 - x1) * (p1[1] - y1) / (y2 - y1)
39                 y = p1[1]
40             elif code_out & 4:
41                 x = x1 + (x2 - x1) * (p4[1] - y1) / (y2 - y1)
42                 y = p4[1]
43             elif code_out & 2:
44                 y = y1 + (y2 - y1) * (p1[0] - x1) / (x2 - x1)
45                 x = p1[0]
46             elif code_out & 1:
47                 y = y1 + (y2 - y1) * (p4[0] - x1) / (x2 - x1)
48                 x = p4[0]
49             if code_out == code1:
50                 x1 = x
51                 y1 = y
52                 code1 = computecode(x1, y1)
53             else:
54                 x2 = x
55                 y2 = y
56                 code2 = computecode(x2, y2)
57     if accept:
58         a = []
59         p1 = (x1, y1)
60         p2 = (x2, y2)
61         a.append(p1)
62         a.append(p2)
63         draw.line((x1, y1, x2, y2), fill=(0, 0, 255))
64     return a
```

```

65     else:
66         a = [(None, None), (None, None)]
67         return a
68
69
70     def findnext(i, n, points):
71         for j in range(i + 1, n):
72             if points[j][0] is not None:
73                 return j
74
75
76     def polygonclip(n, points):
77         count = 0
78         flag = 0
79         final = list()
80         if computeCode(points[n - 1][0], points[n - 1][1]) != 0:
81             flag = 1
82         for i in range(0, n - 1):
83             start = [points[i][0], points[i][1]]
84             end = [points[i + 1][0], points[i + 1][1]]
85             if computeCode(end[0], end[1]) == 0 and computeCode(start[0], start[1]) != 0:
86                 count = count + 1
87                 temp = [None, None]
88                 final.append(temp)
89                 count = count + 1
90
91             a = lineclip(start[0], start[1], end[0], end[1])
92             temp = [a[0][0], a[0][1]]
93             final.append(temp)
94             count = count + 1
95             temp = [a[1][0], a[1][1]]
96             final.append(temp)
97             count = count + 1
98             count = count - 1
99         count = count + 1
100
101         if flag == 1:
102             temp = [None, None]
103             final.append(temp)
104             count = count + 1
105
106         startIndex = findnext(-1, count, final)
107         start = [final[startIndex][0], final[startIndex][1]]
108         flag = 0
109         back = [start[0], start[1]]
110         for i in range(startIndex + 1, count + 1):
111             if flag == 1 and final[i][0] is not None:
112                 flag = 0
113                 if final[i - 1][0] is None:
114                     back[0] = final[i][0]
115                     back[1] = final[i][1]
116                     continue
117             if final[i][0] is None:
118                 continue
119             elif final[i + 1][0] is None:
120                 draw.line((start[0], start[1], final[i][0], final[i][1]), fill=(255, 255, 255))
121                 draw.line((final[i][0], final[i][1], back[0], back[1]), fill=(255, 255, 255))
122                 if i + 1 != count + 1:
123                     index = findnext(i, count, final)
124                     start[0] = final[index][0]
125                     start[1] = final[index][1]
126                 flag = 1
127             else:
128                 draw.line((start[0], start[1], final[i][0], final[i][1]), fill=(255, 255, 255))
129                 start[0] = final[i][0]
130                 start[1] = final[i][1]
131                 if final[i - 1][0] is None:
132                     back[0] = final[i][0]
133                     back[1] = final[i][1]
134         im1.show()
135         im.show()
136
137

```

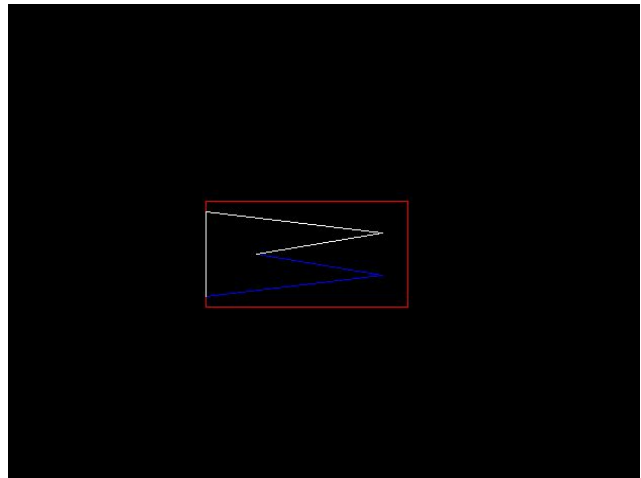
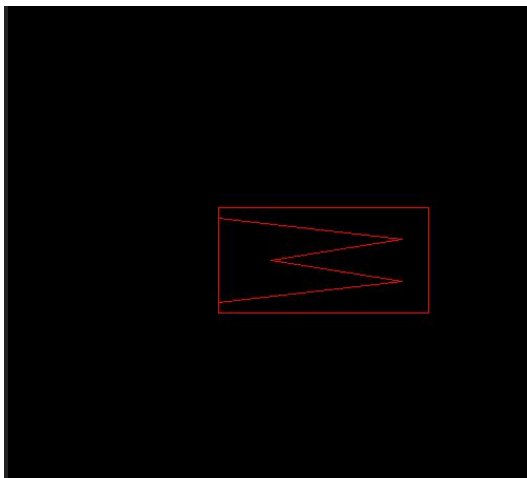


```

138 if __name__ == '__main__':
139     im = Image.new("RGB", (640, 480))
140     im1 = Image.new("RGB", (640, 480))
141     draw2 = ID.Draw(im1)
142     draw = ID.Draw(im)
143     draw.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
144     draw2.polygon((200, 200, 400, 200, 400, 300, 200, 300), outline=255)
145     p1 = (400.0, 300.0)
146     p4 = (200.0, 200.0)
147     point = [[150, 290], [150, 210], [295, 230], [200, 250], [295, 270], [150, 290]]
148     orig = []
149     for i in range(0, 10):
150         orig.append(point[i // 2][i % 2])
151     draw2.polygon(tuple(orig), outline=255)
152     time.sleep(2)
153     polygonclip(6, point)
154

```

Output



2.2. Cohen Sutherland

```

WeilerAtherton.py
1  import PIL.ImageDraw as ID, PIL.Image as Image
2
3
4  def draw1(x1, y1, x2, y2):...
5
6
7
8  class baseVertex:...
9
10
11
12
13
14  class Vertex(baseVertex):...
15
16
17
18
19
20  class Intersection(baseVertex):...
21
22
23
24
25
26
27
28
29  def floatEqual(f1, f2):...
30
31
32
33
34
35
36
37  def floatLarger(f1, f2):...
38
39
40
41
42
43
44
45

```

```

46 def isVertexInPolygon(v, list):
47     judgeIndex = 0
48     for i in range(len(list)):
49         j = i + 1
50         minY = min(list[i % len(list)].y, list[j % len(list)].y)
51         maxY = max(list[i % len(list)].y, list[j % len(list)].y)
52         if floatLarger(v.y, maxY) or floatLarger(minY, v.y):
53             continue
54         if floatEqual(maxY, minY):
55             if floatLarger(v.x, max(list[i % len(list)].x, list[j % len(list)].x)):
56                 judgeIndex += 1
57             continue
58         elif floatLarger(min(list[i % len(list)].x, list[j % len(list)].x), v.x):
59             continue
60         else:
61             return True
62         x = (list[i % len(list)].x - list[j % len(list)].x) / (list[i % len(list)].y - list[j % len(list)].y) * (
63             v.y - list[i % len(list)].y) + list[i % len(list)].x
64         if floatEqual(v.x, x):
65             return None
66         if floatLarger(v.x, x):
67             judgeIndex += 1
68     if judgeIndex % 2 != 0:
69         return True
70     return False
71
72
73 def getX(v):...
74
75
76
77 def getY(v):...
78
79
80
81 def LineCrossH(y, c1, c2):...
82
83
84
85 def LineCrossV(x, c1, c2):...
86
87
88
89 def CutByVerticalLine(s1, s2, list):
90     assert floatEqual(s1.x, s2.x)
91     crossXs = []
92     x = s1.x
93     shearedList = [Vertex(r.x, r.y) for r in list]
94     minY = min(s1.y, s2.y)
95     maxY = max(s1.y, s2.y)
96     for i in range(len(list)):
97         vertex = list[i]
98         c1 = shearedList[i % len(list)]
99         c2 = shearedList[(i + 1) % len(list)]
100         if floatEqual(c1.x, c2.x) and floatEqual(c1.x, x):
101             continue
102         if floatLarger(c1.x, x) and floatLarger(c2.x, x):
103             continue
104         if floatLarger(x, c1.x) and floatLarger(x, c2.x):
105             continue
106         y = float('%9f' % LineCrossV(x, c1, c2))
107         inters = Intersection(x, y)
108         if ((floatLarger(y, minY) and floatLarger(maxY, y))
109             or (c2.y == y and x == s2.x)
110             or (c1.y == y and x == s1.x)
111             or (floatEqual(c2.x, x) and floatEqual(y, s1.y))
112             or (floatEqual(c1.x, x) and floatEqual(y, s2.y))
113             or (floatEqual(y, minY) and (not floatEqual(c1.x, x)) and (not floatEqual(c2.x, x)))
114             or (floatEqual(y, maxY) and (not floatEqual(c1.x, x)) and (not floatEqual(c2.x, x)))):
115             while not ((isinstance(vertex, Vertex) and isinstance(vertex.next, Vertex)) or (
116                 isinstance(vertex, Intersection) and isinstance(vertex.nextS, Vertex))):
117                 if isinstance(vertex, Vertex):
118                     assert isinstance(vertex.next, Intersection)
119                     if (floatLarger(c2.x, c1.x) and floatLarger(vertex.next.x, inters.x)) or (
120                         floatLarger(c1.x, c2.x) and floatLarger(inters.x, vertex.next.x)):
121                         break
122                 vertex = vertex.next
123         else:

```

```

197         assert isinstance(vertex.nextS, Intersection)
198         if (floatLarger(c2.x, c1.x) and floatLarger(vertex.nextS.x, inters.x)) \
199             or (floatLarger(c1.x, c2.x) and floatLarger(inters.x, vertex.nextS.x)) \
200             or (floatLarger(c2.y - c2.x * slope, c1.y - c1.x * slope) and floatLarger(inters.y,
201                                                                                       vertex.nextS.y)) \
202             or (floatLarger(c2.y - c2.x * slope, c1.y - c1.x * slope) and floatLarger(vertex.nextS.y,
203                                                                                       inters.y)):
204             break
205         vertex = vertex.nextS
206     if isinstance(vertex, Vertex):
207         next = vertex.next
208     else:
209         next = vertex.nextS
210     if isinstance(vertex, Vertex):
211         vertex.next = inters
212     else:
213         assert isinstance(vertex, Intersection)
214         vertex.nextS = inters
215     inters.nextS = next
216     if floatEqual(c1.y, y):
217         assert not floatEqual(c2.y, y)
218         if floatLarger(y, c2.y):
219             inters.crossDi = 0
220         else:
221             inters.crossDi = 1
222     elif floatLarger(y, c1.y):
223         inters.crossDi = 1
224     else:
225         inters.crossDi = 0
226
227     if floatLarger(s2.x, s1.x):
228         inters.crossDi = 0 if inters.crossDi == 1 else 1
229     crossXs.append(inters)
230 return crossXs
231
232
233 def processNoCross(listS, listC):...
241
242
243 def Compose(list):...
274
275
276 def decode(list):...
285
286
287 def encode(Str):...
297
298
299 def transDirect(list):...
304
305
306 def toClockwise(list):...
326
327
328 def PolyClipping(S, C, output_clockwise=True):
329     listS = encode(S)
330     listC = encode(C)
331     listS = toClockwise(listS)
332     listC = toClockwise(listC)
333     listI = []
334     for i in range(len(listS)):
335         listS[i - 1].next = listS[i]
336     for i in range(len(listC)):
337         listC[i - 1].next = listC[i]
338     for cutStartIdx in range(len(listC)):
339         s1 = listC[cutStartIdx]
340         s2 = listC[(cutStartIdx + 1) % len(listC)]
341         inters = CutByLine(s1, s2, listS)
342         if len(inters) == 0:
343             continue
344         if floatEqual(s1.x, s2.x):
345             assert not floatEqual(s1.y, s2.y)
346             if floatLarger(s2.y, s1.y):
347                 inters.sort(key=getY)
348             else:
349                 inters.sort(key=getY, reverse=True)

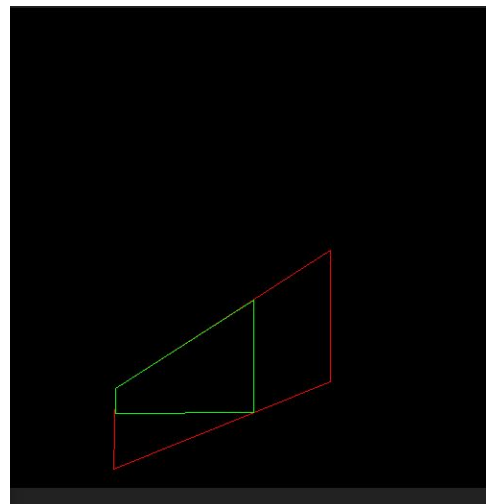
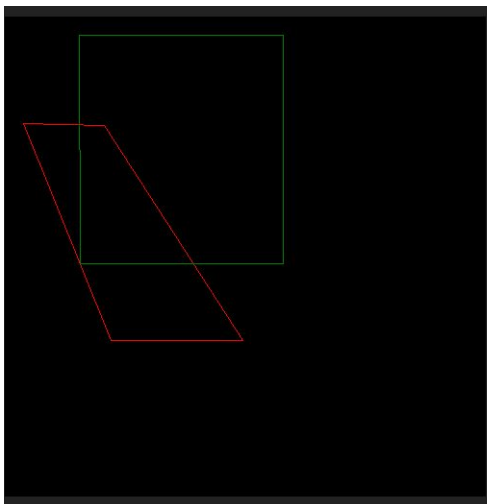
```

```

350         elif floatLarger(s2.x, s1.x):
351             inters.sort(key=getX)
352         else:
353             inters.sort(key=getX, reverse=True)
354
355         for v in inters:
356             listI.append(v)
357         s1.next = inters[0]
358         for i in range(len(inters) - 1):
359             inters[i].nextC = inters[i + 1]
360         inters[len(inters) - 1].nextC = s2
361     if len(listI) == 0:
362         return decode([processNoCross(listS, listC)])
363     results = Compose(listI)
364     if not output_clockwise:
365         results_ = []
366         for result in results:
367             result = transDirect(result)
368             results_.append(result)
369         results = results_
370     return decode(results)
371
372
373 if __name__ == '__main__':
374     im = Image.new("RGB", (500, 500))
375     im1 = Image.new("RGB", (500, 500))
376     draw2 = ID.Draw(im1)
377     draw = ID.Draw(im)
378     draw.polygon((162, 110, 388, 19, 386, 103, 162, 247), outline=255)
379     draw2.polygon((162, 110, 388, 19, 386, 103, 162, 247), outline=255)
380     draw.polygon((242, 78, 480, 77, 480, 289, 242, 289), outline='green')
381     S = "242 78 480 77 480 289 242 289"
382     C = "162 110 388 19 386 103 162 247"
383     point = PolyClipping(S, C)
384     points = point[0].split(' ')
385     i = 0
386     while i < 8:
387         if i == 6:
388             draw1(float(points[i]), float(points[i + 1]), float(points[0]), float(points[1]))
389         else:
390             draw1(float(points[i]), float(points[i + 1]), float(points[i + 2]), float(points[i + 3]))
391         i = i + 2
392     im.show()
393     im1.show()
394

```

Output



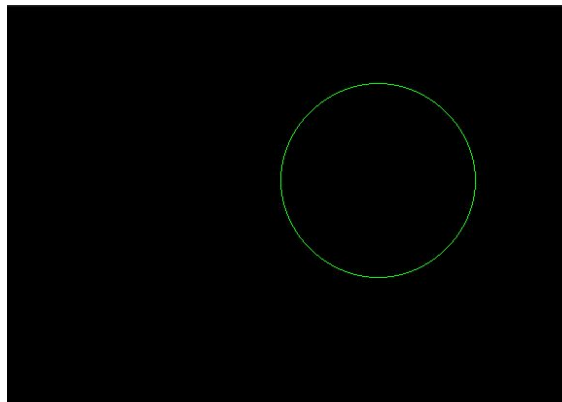
Drawing

1. Circle Drawing

1.1. Bresenham

```
Bresenham.py
1 import PIL.ImageDraw as ID, PIL.Image as Image
2
3
4 def draw1(x1, y1):
5     draw.point((x1, y1), fill=(0, 255, 0))
6
7
8 def drawCircle(xc, yc, x, y):
9     draw1(xc + x, yc + y)
10    draw1(xc - x, yc + y)
11    draw1(xc + x, yc - y)
12    draw1(xc - x, yc - y)
13    draw1(xc + y, yc + x)
14    draw1(xc - y, yc + x)
15    draw1(xc + y, yc - x)
16    draw1(xc - y, yc - x)
17
18
19 def bresenhamCircleDraw(xc, yc, r):
20     x = 0
21     y = r
22     d = 3 - 2 * r
23     drawCircle(xc, yc, x, y)
24     while y >= x:
25         x = x + 1
26         if d > 0:
27             y = y - 1
28             d = d + 4 * (x - y) + 10
29         else:
30             d = d + 4 * x + 6
31             drawCircle(xc, yc, x, y)
32
33
34 if __name__ == '__main__':
35     im = Image.new("RGB", (640, 480))
36     draw = ID.Draw(im)
37     x = int(input("x-coordinate of centre: "))
38     y = int(input("y-coordinate of centre: "))
39     r = int(input("Radius: "))
40     bresenhamCircleDraw(x, y, r)
41     im.show()
```

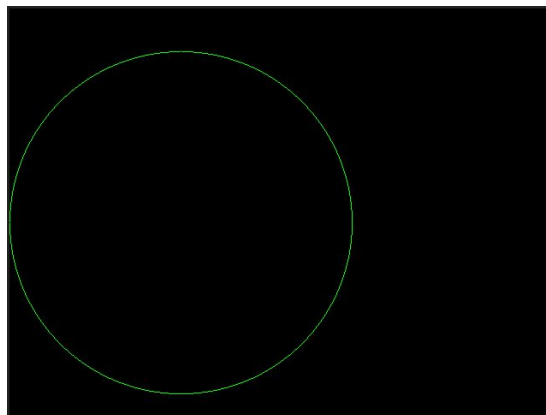
Output



1.2. Mid-Point

```
Midpoint.py
1 import PIL.ImageDraw as ID, PIL.Image as Image
2
3
4 def draw1(x1, y1):
5     draw.point((x1, y1), fill=(0, 255, 0))
6
7
8 def midPointCircleDraw(x_centre, y_centre, r):
9     x = r
10    y = 0
11    draw1(x + x_centre, y + y_centre)
12    if r > 0:
13        draw1(x + x_centre, -y + y_centre)
14        draw1(y + x_centre, x + y_centre)
15        draw1(-y + x_centre, x + y_centre)
16    P = 1 - r
17    while x > y:
18        y += 1
19        if P <= 0:
20            P = P + 2 * y + 1
21        else:
22            x -= 1
23            P = P + 2 * y - 2 * x + 1
24        if x < y:
25            break
26        draw1(x + x_centre, y + y_centre)
27        draw1(-x + x_centre, y + y_centre)
28        draw1(x + x_centre, -y + y_centre)
29        draw1(-x + x_centre, -y + y_centre)
30        if x != y:
31            draw1(y + x_centre, x + y_centre)
32            draw1(-y + x_centre, x + y_centre)
33            draw1(y + x_centre, -x + y_centre)
34            draw1(-y + x_centre, -x + y_centre)
35
36
37 if __name__ == '__main__':
38     im = Image.new("RGB", (640, 480))
39     draw = ID.Draw(im)
40     x = int(input("x-coordinate of centre: "))
41     y = int(input("y-coordinate of centre: "))
42     r = int(input("Radius: "))
43     midPointCircleDraw(x, y, r)
44     im.show()
```

Output

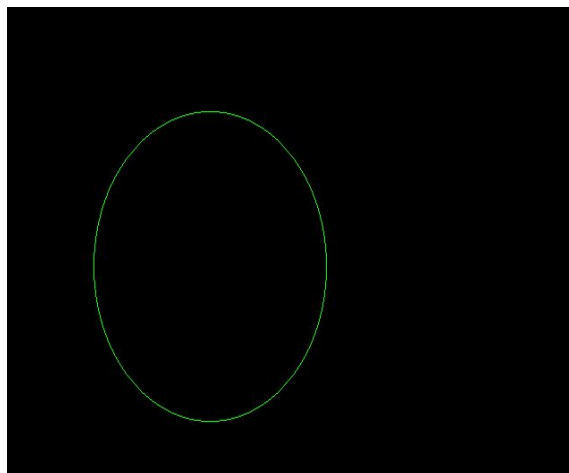


2. Ellipse Drawing

2.1. Mid-Point

```
Midpoint.py
1 import PIL.ImageDraw as ID, PIL.Image as Image
2
3
4 def draw1(x1, y1):
5     draw.point((x1, y1), fill=(0, 255, 0))
6
7
8 def midptellipse(rx, ry, xc, yc):
9     x = 0
10    y = ry
11    d1 = ((ry * ry) - (rx * rx * ry) + (0.25 * rx * rx))
12    dx = 2 * ry * ry * x
13    dy = 2 * rx * rx * y
14    while dx < dy:
15        draw1(x + xc, y + yc)
16        draw1(-x + xc, y + yc)
17        draw1(x + xc, -y + yc)
18        draw1(-x + xc, -y + yc)
19        if d1 < 0:
20            x += 1
21            dx = dx + (2 * ry * ry)
22            d1 = d1 + dx + (ry * ry)
23        else:
24            x += 1
25            y -= 1
26            dx = dx + (2 * ry * ry)
27            dy = dy - (2 * rx * rx)
28            d1 = d1 + dx - dy + (ry * ry)
29    d2 = (((ry * ry) * ((x + 0.5) * (x + 0.5))) +
30          ((rx * rx) * ((y - 1) * (y - 1))) -
31          (rx * rx * ry * ry))
32    while y >= 0:
33        draw1(x + xc, y + yc)
34        draw1(-x + xc, y + yc)
35        draw1(x + xc, -y + yc)
36        draw1(-x + xc, -y + yc)
37        if d2 > 0:
38            y -= 1
39            dy = dy - (2 * rx * rx)
40            d2 = d2 + (rx * rx) - dy
41        else:
42            y -= 1
```

Output

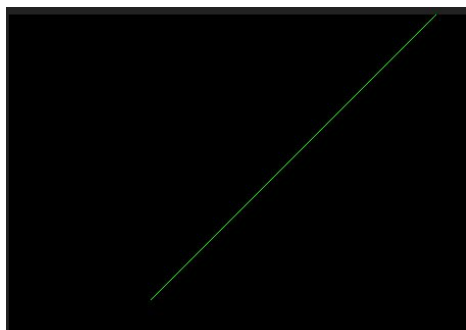


3. Line Drawing

3.1. Bresenham

```
Bresenham.py x
1 import PIL.ImageDraw as ID, PIL.Image as Image
2
3
4 def draw1(x1, y1):
5     draw.point((x1, y1), fill=(0, 255, 0))
6
7
8 def ROUND(a):
9     return int(a + 0.5)
10
11
12 def drawBresenham(x1, y1, x2, y2):
13     dx = x2 - x1
14     dy = y2 - y1
15     if abs(dx) > abs(dy):
16         p = 2 * (abs(dy)) - (abs(dx))
17         y = y1
18         x = x1
19         for i in range(0, abs(dx) + 1):
20             draw1(x, y)
21             if p >= 0:
22                 y = y + 1 if dy >= 0 else y - 1
23                 p = p + 2*abs(dy) - 2*abs(dx)
24             else:
25                 p = p + 2*abs(dy)
26                 x = x + 1 if dx >= 0 else x - 1
27     else:
28         p = 2 * (abs(dx)) - (abs(dy))
29         x = x1
30         y = y1
31         for i in range(0, abs(dy) + 1):
32             draw1(x, y)
33             if p >= 0:
34                 x = x + 1 if dx >= 0 else x - 1
35                 p = p + 2*abs(dx) - 2*abs(dy)
36             else:
37                 p = p + 2*abs(dx)
38                 y = y + 1 if dy >= 0 else y - 1
39     im.show()
40
41
42 if __name__ == '__main__':
43     im = Image.new("RGB", (640, 480))
44     draw = ID.Draw(im)
45     drawBresenham(200, 500, 500, 200)
46
```

Output



3.2. DDA

```
DDA.py
1 import PIL.ImageDraw as ID, PIL.Image as Image
2
3
4 def draw1(x1, y1):
5     draw.point((x1, y1), fill=(0, 255, 0))
6
7
8 def ROUND(a):
9     return int(a + 0.5)
10
11
12 def drawDDA(x1, y1, x2, y2):
13     x, y = x1, y1
14     length = (x2 - x1) if (x2 - x1) > (y2 - y1) else (y2 - y1)
15     dx = (x2 - x1) / float(length)
16     dy = (y2 - y1) / float(length)
17     for i in range(length):
18         x += dx
19         y += dy
20         x1 = ROUND(x)
21         y1 = ROUND(y)
22         draw1(x1, y1)
23     im.show()
24
25
26 if __name__ == '__main__':
27     im = Image.new("RGB", (640, 480))
28     draw = ID.Draw(im)
29     x1 = int(input("x1: "))
30     y1 = int(input("y1: "))
31     x2 = int(input("x2: "))
32     y2 = int(input("y2: "))
33     drawDDA(x1, y1, x2, y2)
```

Output

