

| **TITLE**: Write a program to demonstrate the LINE CLIPPING algorithm |
| --- |

**AIM:**

**Visit Vlab and Explore it**

[**https://cse18-iiith.vlabs.ac.in/exp/clipping-line/**](https://cse18-iiith.vlabs.ac.in/exp/clipping-line/)

Write a program to demonstrate the LINE CLIPPING algorithm

1. Cohen-Sutherland-algorithm
2. Mid-Point Subdivision Line Clipping Algorithm
3. Liang-Barsky Line Clipping Algorithm

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

*Students should write appropriate CO as per syllabus*

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algorithm:**

Algorithm of midpoint subdivision Line Clipping:

Step1: Calculate the position of both endpoints of the line

Step2: Perform OR operation on both of these endpoints

Step3: If the OR operation gives 0000

then

Line is guaranteed to be visible

else

Perform AND operation on both endpoints.

If AND ≠ 0000

then the line is invisible

else

AND=6000

then the line is clipped case.

Step4: For the line to be clipped. Find midpoint

Xm=(x1+x2)/2

Ym=(y1+y2)/2

Xmis midpoint of X coordinate.

Ymis midpoint of Y coordinate.

Step5: Check each midpoint, whether it nearest to the boundary of a window or not.

Step6: If the line is totally visible or totally rejected not found then repeat step 1 to 5.

Step7: Stop algorithm.

Example: Window size is (-3, 1) to (2, 6). A line AB is given having co-ordinates of A (-4, 2) and B (-1, 7). Does this line visible. Find the visible portion of the line using midpoint subdivision?

Solution:

Step1: Fix point A (-4, 2)

**Implementation details:**

**x1= 1**

**y1= 6**

**x2= 15**

**y2= 11**

**x\_max = 12**

**y\_max = 14**

**x\_min = 4**

**y\_min = 2**

**def cohen\_sutherland(x1,y1,x2,y2,x\_max,y\_max,x\_min,y\_min):**

**a = checkCase(x1,y1,x\_max,y\_max,x\_min,y\_min)**

**b = checkCase(x2,y2,x\_max,y\_max,x\_min,y\_min)**

**if(a[0]+b[0]+a[1]+b[1]+a[2]+b[2]+a[3]+b[3]==0):**

**return("visible Line")**

**pass**

**elif(a[0]\*b[0]+a[1]\*b[1]+a[2]\*b[2]+a[3]\*b[3]==0):**

**m=(y2-y1)/(x2-x1)**

**ans=[]**

***#line clipped***

**if(a[3]+b[3]==1):**

***#left boundary of rectangle window***

**y=y1+m\*(x\_min-x1)**

**x=x\_min**

**ans.append((x,y))**

**if(y<y\_min or y>y\_max):**

**return("Invisible line case 2")**

**if(a[2]+b[2]==1):**

***#right boundary of rectangle window***

**y=y1+m\*(x\_max-x1)**

**x=x\_max**

**ans.append((x,y))**

**if(y<y\_min or y>y\_max):**

**return("Invisible line case 2")**

**if(a[1]+b[1]==1):**

***#bottom boundary of rectangle window***

**x=x1+(y\_min-y1)/m**

**y=y\_min**

**ans.append((x,y))**

**if(x<x\_min or x>x\_max):**

**return("Invisible line case 2")**

**if(a[0]+b[0]==1):**

***#top boundary of rectangle window***

**x=x1+(y\_max-y1)/m**

**y=y\_max**

**ans.append((x,y))**

**if(x<x\_min or x>x\_max):**

**return("Invisible line case 2")**

**return("Clipped line ", ans,a,b)**

**pass**

**return("Invisible line")**

**def checkCase(x,y,x\_max,y\_max,x\_min,y\_min):**

**bit=[0,0,0,0]**

**if(x<x\_min):**

**bit[3]=1**

**if(x>x\_max):**

**bit[2]=1**

**if(y<y\_min):**

**bit[1]=1**

**if(y>y\_max):**

**bit[0]=1**

**return bit**

**print(cohen\_sutherland(x1,y1,x2,y2,x\_max,y\_max,x\_min,y\_min))**

*#***TODO**

x1**=** 1

y1**=** 6

x2**=** 15

y2**=** 11

x\_max **=** 12

y\_max **=** 14

x\_min **=** 4

y\_min **=** 2

**def** findLine(**x1**,**y1**,**x2**,**y2**,**x\_max**,**y\_max**,**x\_min**,**y\_min**):

a **=** checkCase(x1,y1,x\_max,y\_max,x\_min,y\_min)

b **=** checkCase(x2,y2,x\_max,y\_max,x\_min,y\_min)

**if**(a[0]**+**b[0]**+**a[1]**+**b[1]**+**a[2]**+**b[2]**+**a[3]**+**b[3]**==**0):

**return**(1) *#****visible***

**elif**(a[0]**\***b[0]**+**a[1]**\***b[1]**+**a[2]**\***b[2]**+**a[3]**\***b[3]**==**0):

**return**(2) *#****clipped***

**return**(0) *#****invisible***

array**=**[]

**def** checkCase(**x**,**y**,**x\_max**,**y\_max**,**x\_min**,**y\_min**):

bit**=**[0,0,0,0]

**if**(x**<**x\_min):

bit[3]**=**1

**if**(x**>**x\_max):

bit[2]**=**1

**if**(y**<**y\_min):

bit[1]**=**1

**if**(y**>**y\_max):

bit[0]**=**1

**return** bit

**def** mid(**x1**,**y1**,**x2**,**y2**,**x\_max**,**y\_max**,**x\_min**,**y\_min**):

midx**=**(x1**+**x2)**//**2

midy**=**(y1**+**y2)**//**2

print((x1,y1,x2,y2))

**if**(x1**==**midx **or** x2**==**midx **or** y1**==**midy **or** y2**==**midy):

*#****end of recursion***

**return** 0

firstHalf**=**findLine(x1,y1,midx,midy,x\_max,y\_max,x\_min,y\_min)

**if**(firstHalf **==** 2 ):*#* ***if half is clipped recurse***

print((x1,y1,midx,midy,x\_max,y\_max,x\_min,y\_min))

mid(x1,y1,midx,midy,x\_max,y\_max,x\_min,y\_min)

**elif**(firstHalf **==** 1 ): *#* ***if first half is visible, return the coordinates***

Ax1**=**x1

Ay1**=**y1

Ax2**=**midx

Ay2**=**midy

array.append((Ax1,Ay1,Ax2,Ay2))

secondHalf**=**findLine(midx,midy,x2,y2,x\_max,y\_max,x\_min,y\_min)

**if**(secondHalf **==** 2 ):*#* ***if half is clipped recurse***

print((midx,midy,x2,y2,x\_max,y\_max,x\_min,y\_min))

mid(midx,midy,x2,y2,x\_max,y\_max,x\_min,y\_min)

**elif**(secondHalf **==** 1 ): *#* ***if first half is visible, return the coordinates***

Ax1**=**midx

Ay1**=**midy

Ax2**=**x2

Ay2**=**y2

array.append((Ax1,Ay1,Ax2,Ay2))

print(mid(x1,y1,x2,y2,x\_max,y\_max,x\_min,y\_min))

print(array)

print("final coordinates -- ")

print(min(tup[0] **for** tup **in** array),min(tup[1] **for** tup **in** array),max(tup[2] **for** tup **in** array),min(tup[3] **for** tup **in** array))

x1**=** 7

y1**=** 9

x2**=** 11

y2**=** 4

x\_max **=** 10.0

y\_max **=** 8.0

x\_min **=** 4.0

y\_min **=** 4.0

**def** laingBaserkey(**x1**,**y1**,**x2**,**y2**,**x\_max**,**y\_max**,**x\_min**,**y\_min**):

dx**=**x2**-**x1

dy**=**y2**-**y1

p1 **=** **-**dx

p2 **=** dx

p3 **=** **-**dy

p4 **=** dy

q1 **=** x1 **-** x\_min

q2 **=** x\_max **-** x1

q3 **=** y1 **-** y\_min

q4 **=** y\_max **-** y1

**if**(p1**==**0):

**return**("parallel line")

**if**(p3**==**0):

**return**("parallel line")

*#****If p1 < 0 and p3 < 0, or if p2 < 0 and p4 < 0, the line lies completely outside the clipping window. In this case, the algorithm terminates.***

**if**((p1 **<** 0 **and** p3 **<** 0) **or** (p2 **<** 0 **and** p4 **<** 0)):

**return**("out of window")

tin **=** max(q1 **/** p1 ,q3 **/** p3, 0)

tout **=** min(q2 **/** p2 ,q4 **/** p4, 1)

answer**=**[]

**if**(tin**>**tout):

**return**("Invisible line case 2")

x**=** x1**+**tin**\***dx

y**=** y1**+**tin**\***dy

answer.append((x,y))

x**=** x1**+**tout**\***dx

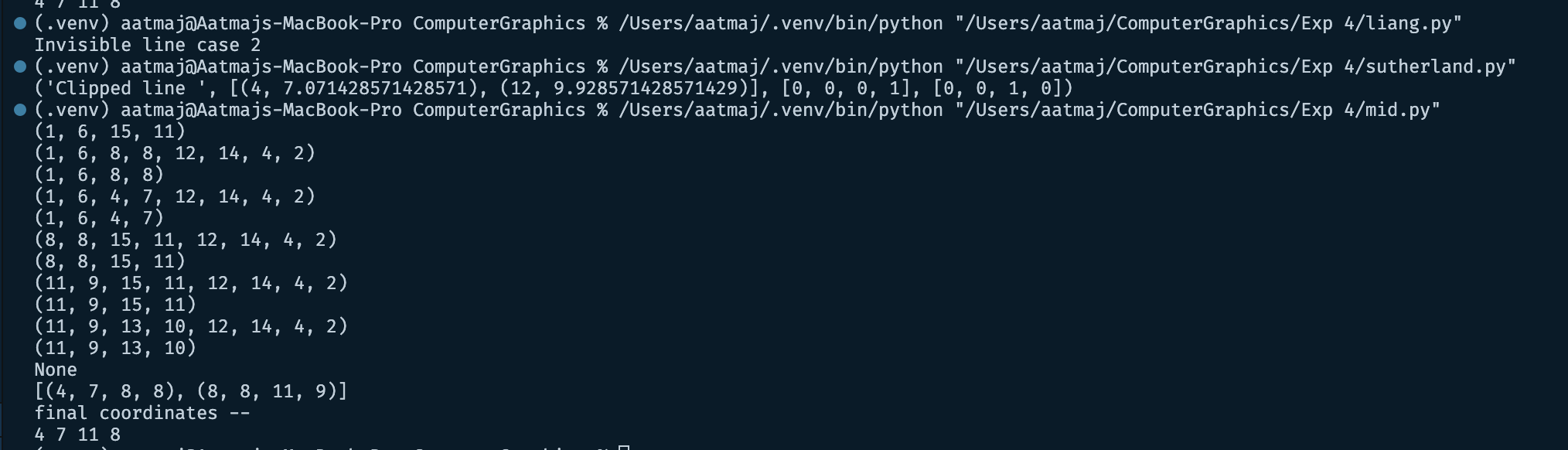
y**=** y1**+**tout**\***dy

answer.append((x,y))

**return**("Clipped",answer)

print(laingBaserkey(x1,y1,x2,y2,x\_max,y\_max,x\_min,y\_min))

**Output(s) (final edited screen shot):**



**Conclusion and discussion (Comparative ):**

Thus we have implemented Line clipping algorithms. We understood the working of cohen sutherland and midpoint line ubdivision algorithm. The cohen sutherland algorithm is a very easy and effective method of performing line clipping. The Midpoint line subdivision on the other hand is a recursive algorithm that requires lot of resources. The liang clipping algorithm is also easy to implement. It relies on the line formulas.

**Date: 30 Aug 23**

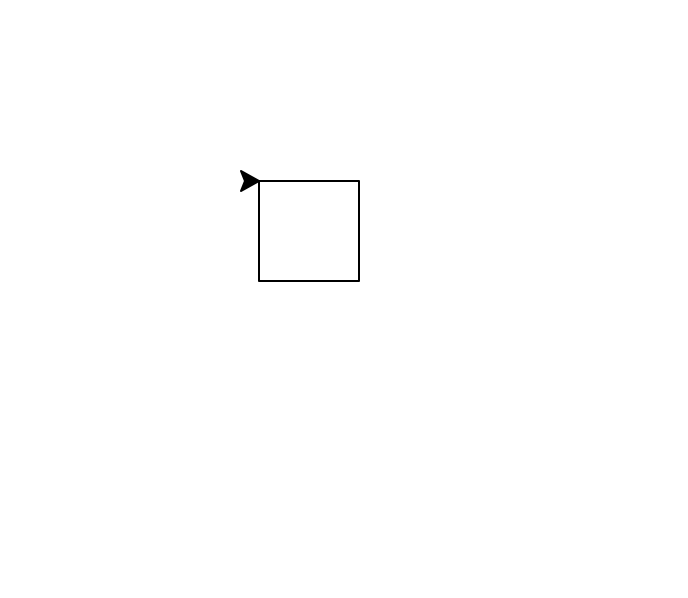
**Signature of faculty in-charge**

**Post Lab**

What is Turtle in CG, Demonstrate use of Turtle by implementing it?

Turtle is a Python library to draw graphics. After we import Turtle we can give commands like forward, backward, right, left etc. This commands will draw different shapes when we tell the turtle to move.

“Turtle” is a Python feature like a drawing board, which lets us command a turtle to draw all over it! We can use functions like turtle.forward(…) and turtle.right(…) which can move the turtle around.



*#* ***Python program to draw square***

*#* ***using Turtle Programming***

**import** turtle

skk **=** turtle.Turtle()

**for** i **in** range(4):

skk.forward(50)

skk.right(90)

turtle.done()