GENERALIZATION OF LIANG-BARSKY LINE CLIPPING ALGORITHM

Project report submitted in partial fulfilment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

By

SACHIN GUPTA
(161211)

UNDER THE SUPERVISION OF

Prof. KARANJEET SINGH

(HOD, MATHEMATICS DEPARTMENT)

to

Department of Computer Science & Engineering and Information Technology

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT, SOLAN - 173234

HIMACHAL PRADESH

DECLARATION

I hereby declare that the work presented in this report entitled **“ Generalization of Liang-Barsky Line Clipping Algorithm”** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2019 to May 2020 under the supervision of **Prof. Dr. Karanjeet Singh** (HOD, Mathematics Department).
The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Sachin Gupta.

(Candidate Signature)
Sachin Gupta
Roll. No.- 161211

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Karan

(Supervisor Signature)
Prof. Dr. Karanjeet Singh
(HOD, Mathematics Department)

ACKNOWLEDGEMENT

We express our deepest gratitude for having an opportunity to continue our project work with the Department of Mathematics and Computer Science Engineering at Jaypee University of Information Technology, Waknaghat, Solan, H. P. We gratefully acknowledge to **Prof. Dr. Satya Prakash Ghrera,** HOD, CSE for his valuable support and suggestions. We express our deepest gratitude and special thanks to our supervisor **Prof. Karanjeet Singh** for his keen interest in our project, took time out to hear, guide and keep us on the correct path, giving necessary advice and guidance to carry out our project work. His prompt inspirations, timely suggestions with kindness, enthusiasm and dynamism had been solely and mainly responsible for our deep interest in our project works.

TABLE OF CONTENT

LIST OF FIGURES

ABSTRACT

In this project, I have derived a new efficient algorithm with the help of “Liang-Barsky” line clipping algorithm which can clip any general “Single parameter dependent” curve and it is much better than cohen-sutherland line clipping algorithm as well as Liang-Barsky line clipping algorithm in terms of cost and efficiency. Because if the object, which is to be clipped, is being rendered line by line then at each stage we need to calculate point of intersection between clipping window and higher order single parametrised curves. Which will involve lots of calculations while finding the intersection points so to avoid that, with the help of idea of liang-barsky algorithm first, we will be parametrizing the curve and then will restrict it into the clipping window by specific boundary conditions. Same concept we will be using for clipping the 3-D object against 3-D clipping window and from now we are naming it as clipping volume.

CHAPTER-1

INTRODUCTION

1.1 Introduction

In computer graphics, clipping is a method to disable or enable some portion of the text, images or objects after rendering within a desired region. Mathematically we can easily understand the concept of clipping using terminology of constructive geometry. When we render any image then it only includes pixels which lies in the intersection between the clip region and the scene model. All the lines and surfaces which were outside of the view volume should be rejected.

To improve performance and efficiency of rendering generally clipping regions are specified. A well-chosen clip allows to minimize calculations, related to pixels that the user cannot see, to the renderer to save time and energy. Pixels that are outside the clip region will not be drawn. More informally, pixels that will not be drawn are called “clipped."

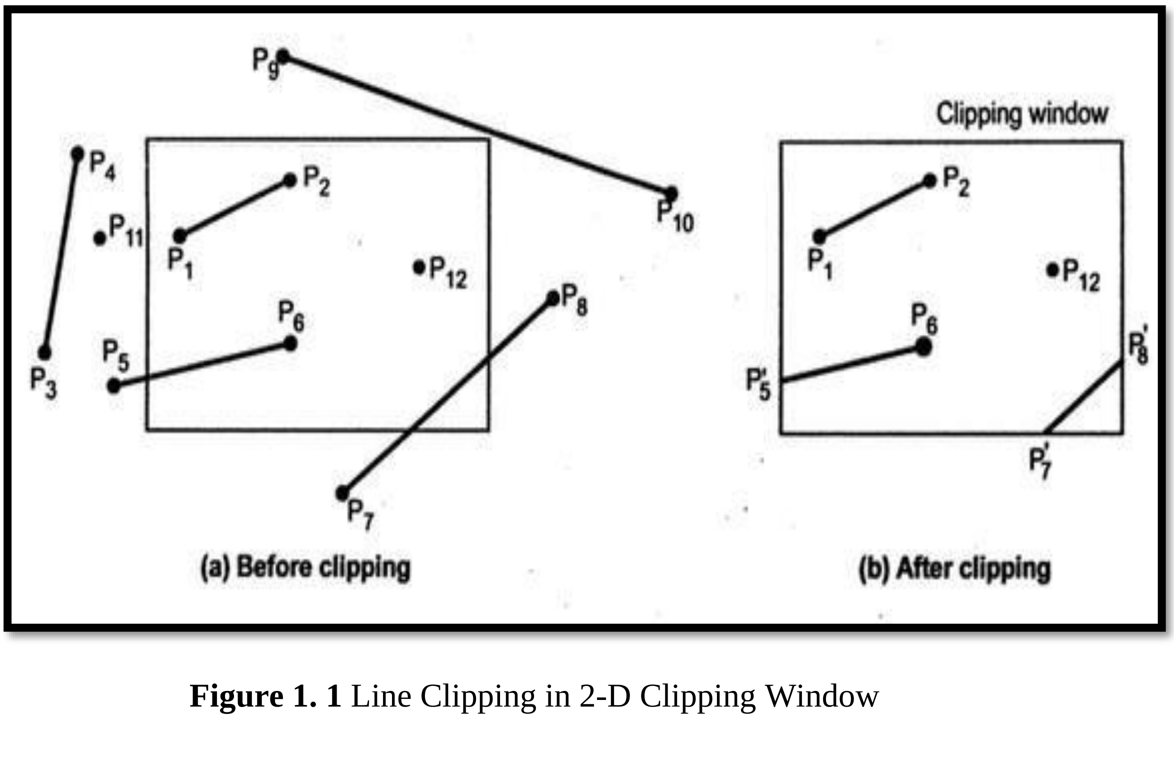


Figure 1. 1 Line Clipping in 2-D Clipping Window

In the viewing pipeline, we need only those portion of the picture which are within the clipping window (it may be 2-D or 3-D). for 2-D objects we use the term “clipping window” and for 3-D we may use “clipping volume”. Clipping algorithms are to be applied in world coordinate so that using viewing pipeline we project only desired portion on device coordinate. Alternatively, the world coordinate part mapped to device coordinate first, or normalized device coordinate, then clipped w.r.t. the boundaries of viewport.

2-D Viewing Pipeline: -

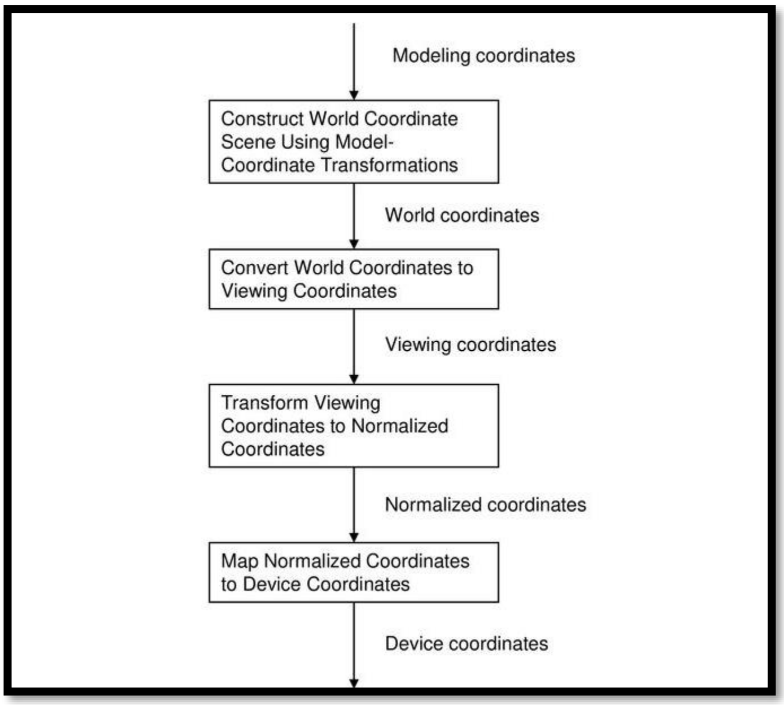

Modeling coordinates
Construct World Coordinate Scene Using Model-Coordinate Transformations
World coordinates
Convert World Coordinates to Viewing Coordinates
Viewing coordinates
Transform Viewing Coordinates to Normalized Coordinates
Normalized coordinates
Map Normalized Coordinates to Device Coordinates
Device coordinates

Figure 1. 2 Flow Chart of 2-D Viewing Pipelining

- Model / Object Coordinates: -

  This is the coordinate system where individual objects are designed which are to be used to make world coordinates.

- World Coordinates: -

  This is the coordinate system where individual objects are integrated to make a proper scene.

- Viewing Coordinates: -

  From world coordinate we move the scene into the coordinate of camera called viewer's coordinate.

- Normalized Coordinates: -

  As we know when we take any picture or video from camera then it should be platform independent so for this purpose we use normalized coordinates system.

- Device Coordinates: -

  After normalized coordinates we map the picture or video to specific device coordinates.

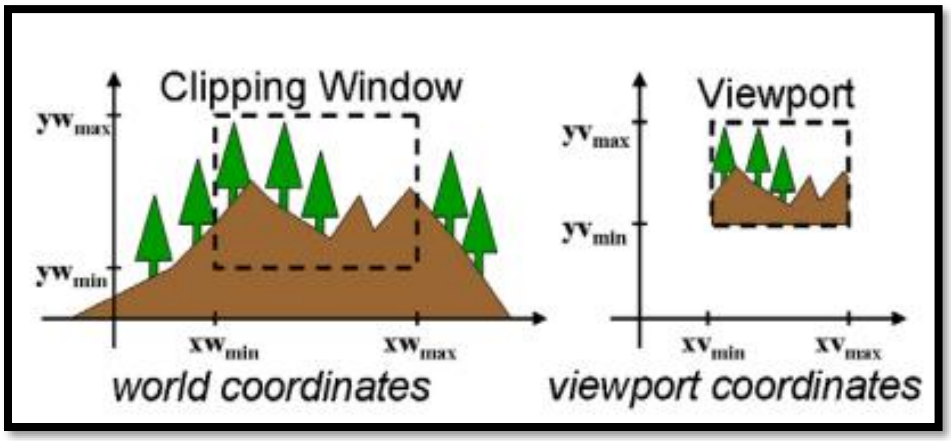


Figure 1. 3 Viewport Coordinates

Type of Clipping: -

- Line Clipping: -

  Already described.

- Polygon Clipping: -

  Here we retain the portion of polygon which lies inside the clipping window against which we want to clip.

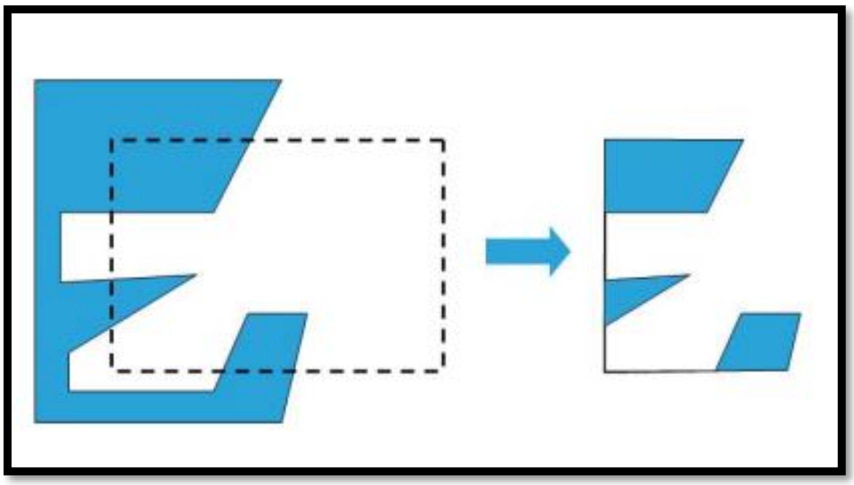


Figure 1. 4 Polygon Clipping

- Text Clipping: -

  Here we retain the portion of text which lies inside the clipping window against which we want to clip.

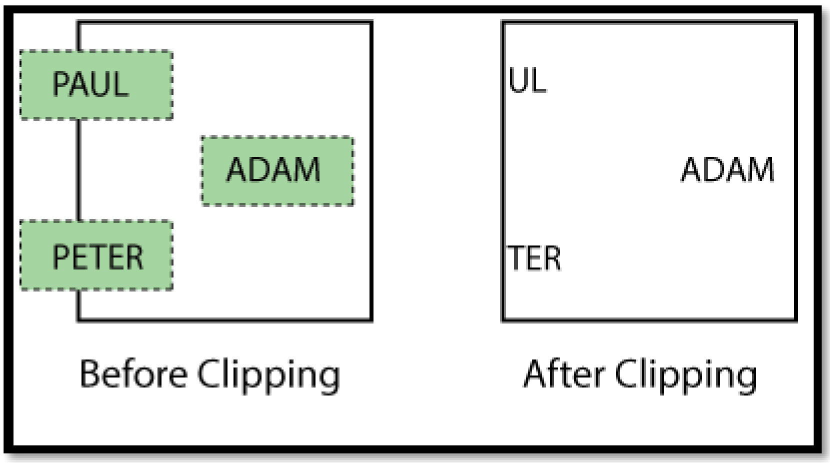


Figure 1.5 Text Clipping

1.2 Problem Statement

In this project, we are trying to derive a new algorithm by modifying Liang-Barsky's line clipping algorithm into a general (single-parameterized) curve clipping algorithm.

As we know there are many clipping algorithms such as Liang-Barsky line clipping algorithm, Cohen-Sutherland line clipping algorithm, Nicholl-Lee-Nic-holl Line Clipping algorithm, Sutherland-Hodgeman polygon Clipping algorithm and many more. But they all involve lot of calculation as compare to our new derived algorithm if we go for any arbitrary 2-D shaped curve. Once we derive this, we will be trying the same for 3-D Clipping window.

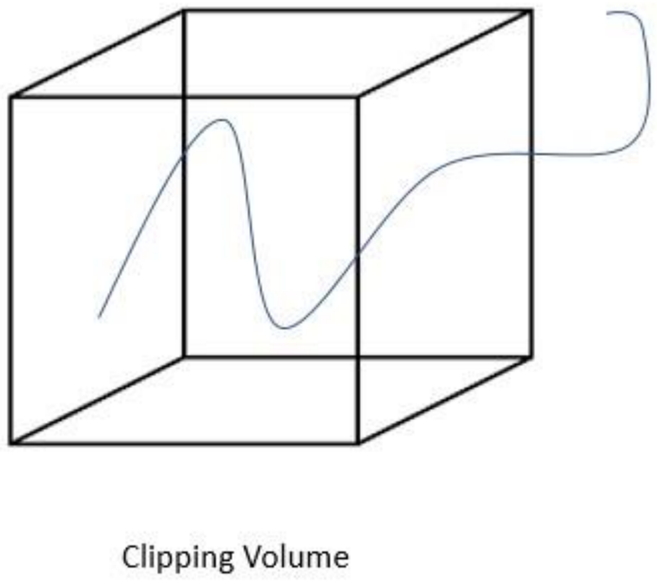


Figure 1.6 General Curve in Clipping Volume

1.3 Objectives

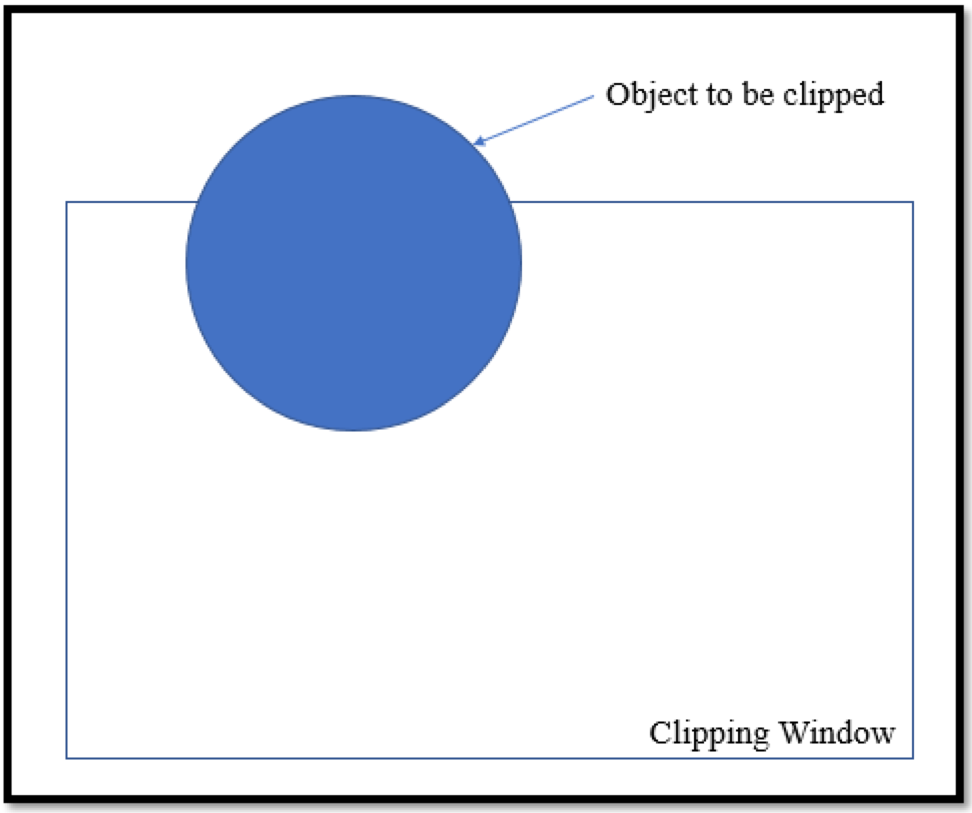


Figure 1.7 Circle inside the clipping volume

Now in the above picture we need to clip the circle against this rectangular clipping window. it could be done using line clipping algorithm by rendering the above window line by line. As you can see by doing so lots of calculation will be involved because circle is 2nd order curve. So here we are trying to derive and algorithm which can clip such an arbitrary and more complex shapes in minimum calculations and less efforts.

Once we derive this, we will be trying the same against 3-D Clipping volume.

1.4 Methodology

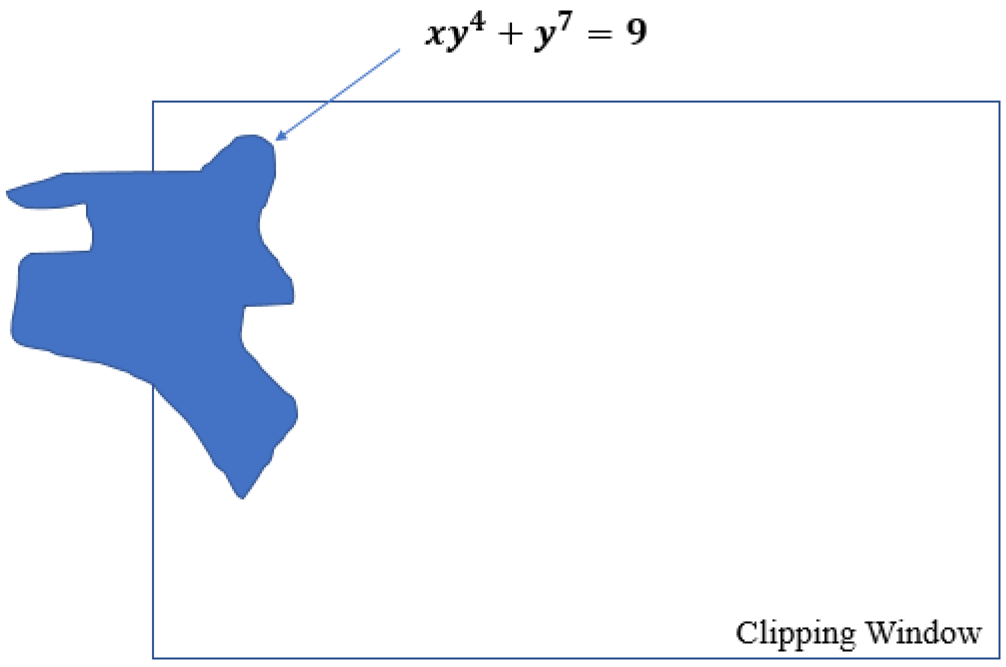


Figure 1. 8 Higher order Curve in Clipping Volume

For clipping above shape from clipping window if we use line clipping algorithm it will involve lots of calculation. So, to avoid that we need to parametrize the above curve and restrict the parametrized coordinate within the clipping window. By doing so we get the extreme values of the parameter and by using these parameters we get the required coordinates to clip the shape easily.

Using this method, we can clip off many more complex shapes with less efforts and minimum cost.

CHAPTER-2

LITERATURE SURVEY

2.1 Point Clipping Algorithm:

Let there be a point having coordinates P (a, b). Now we want to clip it w.r.t. the given clipping window. We must accept (enable) only those points which will lie inside the clipping window else reject (disable) all other points.

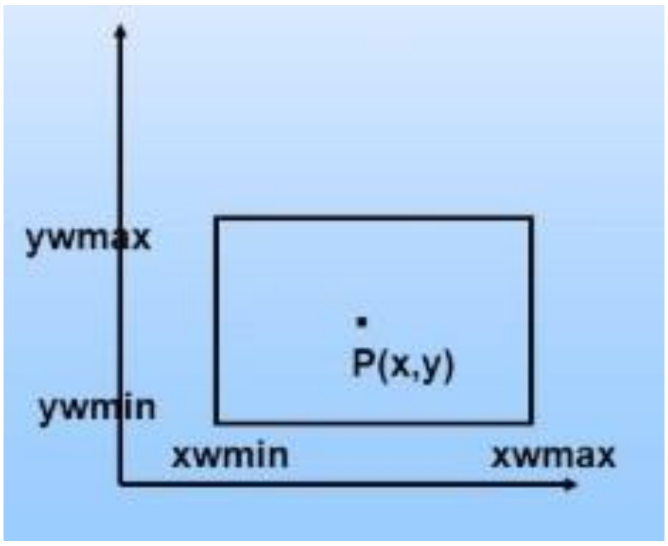


Figure 2.1 2-D clipping window

Algorithm:

Point P (a, b) will be accepted if,

$\boldsymbol{xw_{min}} \leq \boldsymbol{a} \leq \boldsymbol{xw_{max}}$ && $\boldsymbol{yw_{min}} \leq \boldsymbol{b} \leq \boldsymbol{yw_{max}}$

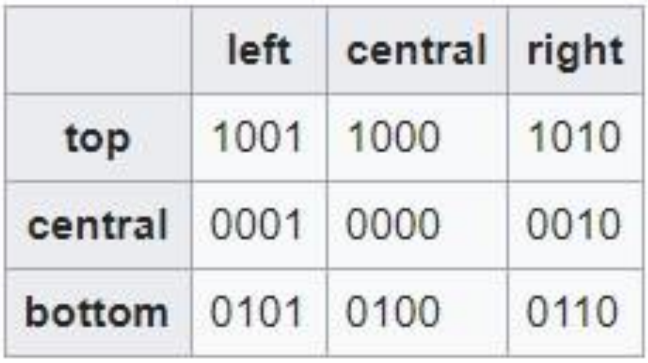
Both the inequalities should hold simultaneously.

Otherwise point will be disabled.

Same concept we will be using in our modifying algorithm and for 3-D point against 3-D clipping volume as well.

2.2 Cohen Sutherland Line Clipping Algorithm:

This is one of the line clipping algorithm where we divide 2-Dimensional space into 9 different regions as shown in picture below. Region of interest or clipping window will always be shown in the centre.



	left	central	right
top	1001	1000	1010
central	0001	0000	0010
bottom	0101	0100	0110

Figure 2.2 Divided region around clipping window

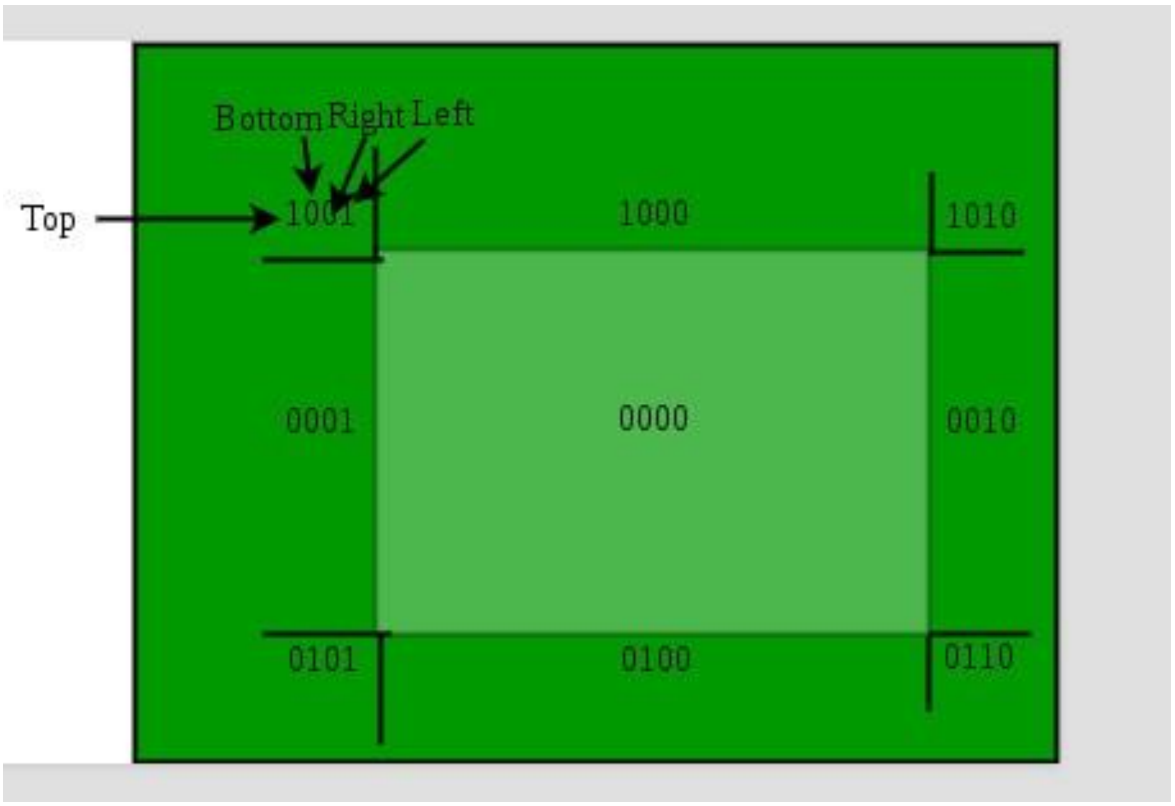


Figure 2.3 Bit-Code for divided region

We have created 9 regions. 1 inside and 8 outside region.

Now we have given the line which we need to clip against the given rectangular clipping window. We will assign both the end of the given line by their bit-code.

(xmin, xmax, ymin and ymax).

If x < xmin, bit number 1 will be set.
If x > xmax, bit number 2 will be set.
If y < ymin, bit number 3 will be set.
If y > ymax, bit number 4 will be set.

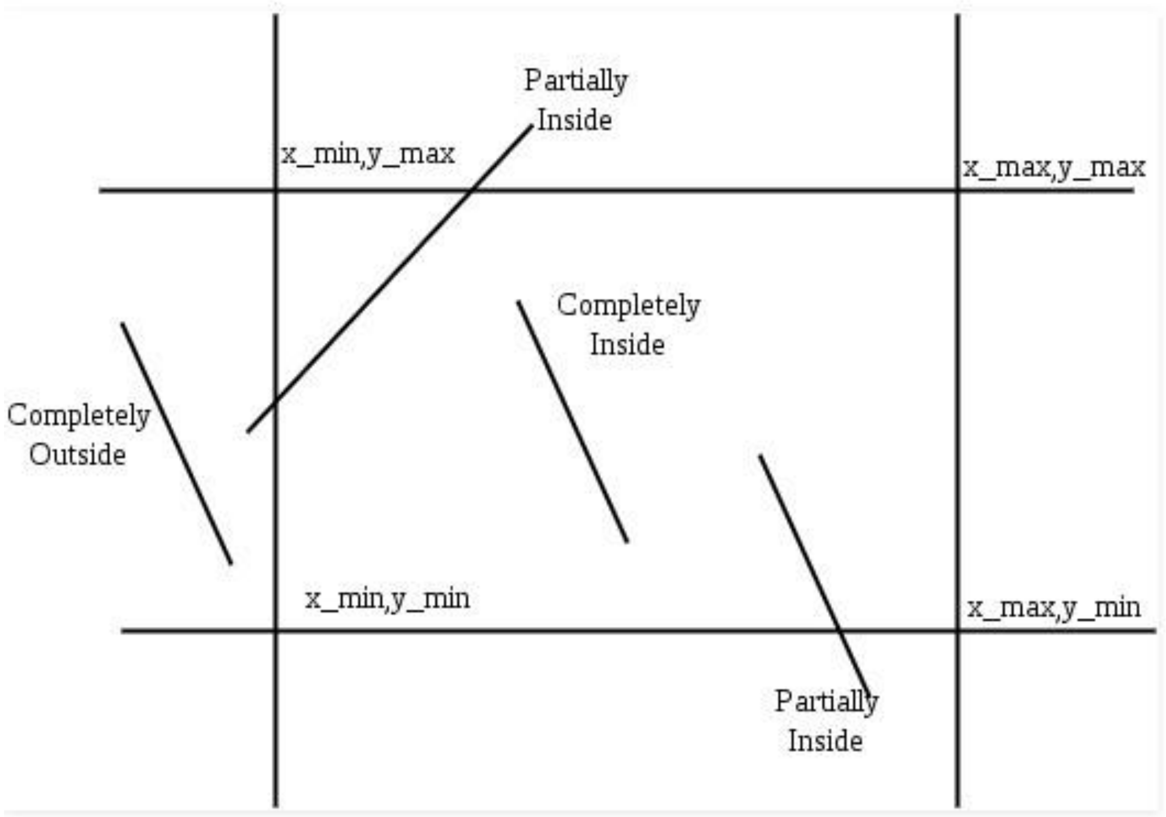


Figure 2.4 Cohen-Sutherland test cases

Total 3 possibilities can be listed –

1. Perform Bitwise OR operation from the both end points of line. If it is equals to 0000 it means both the ends were completely inside of the clipping window. Complete line will be retained.
2. Perform Bitwise AND operation from the both end points of line. If it is not equals to 0000 it means line is completely outside of the clipping window. Complete line will be rejected.
3. Else line will be partially inside and partially outside of the clipping window in this case we need to find out the intersection point by solving line equation with the equation of the clipping window simultaneously and retain the required line.

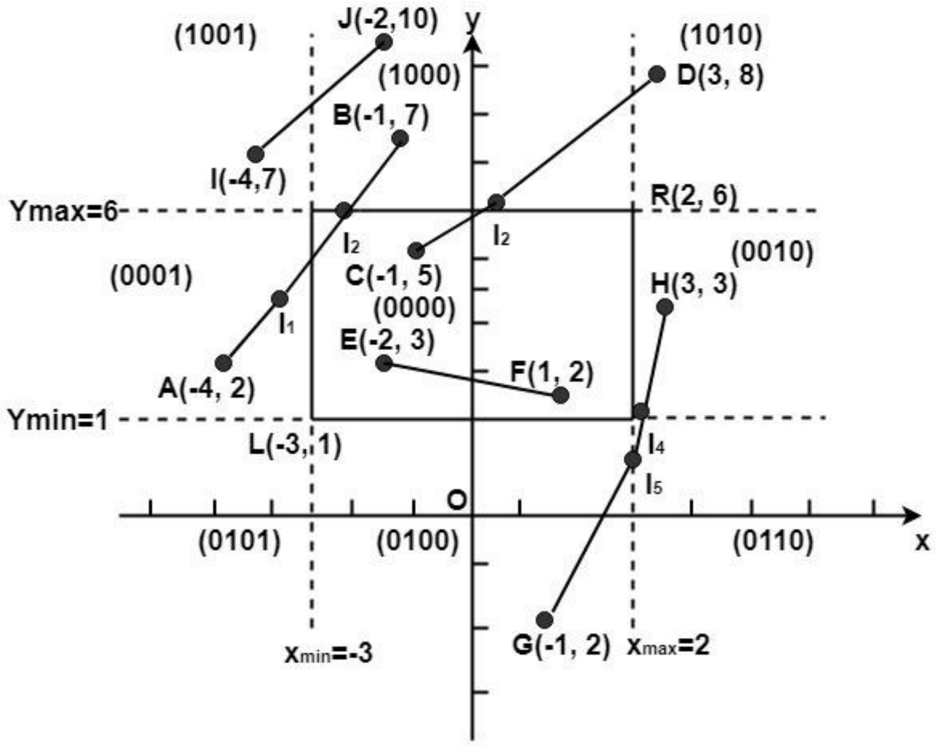


Figure 2.5 Cohen-Sutherland Examples

2.3 Liang-Barsky Line Clipping Algorithm:

This is one of the more efficient algorithm than Cohen Sutherland line clipping algorithm and many others. Here we have used the concept of parametrization. Firstly, we parametrized the line and restricted the parametrized point within the clipping window and solve the inequality to find out which portion of the line we need to enable and which portion we need to disable. This algorithm can be extended against 3-D clipping volume as well.

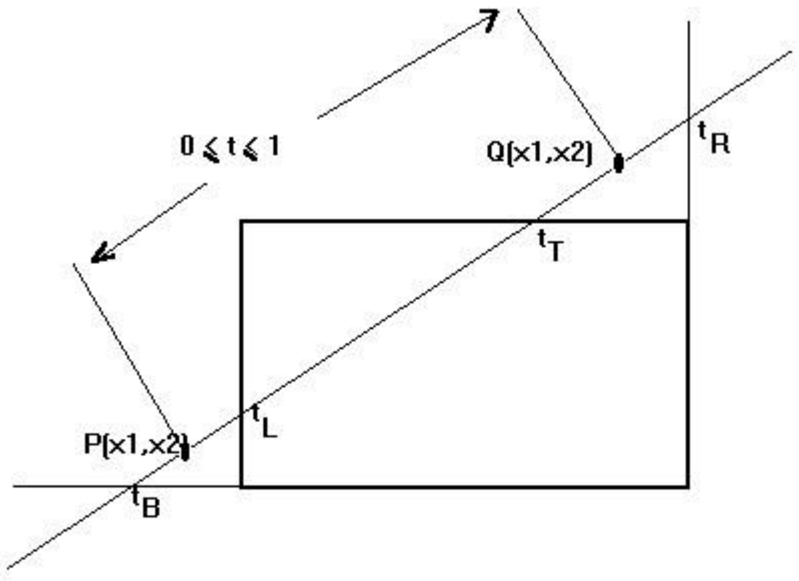


Figure 2.6 Parametrized 2-D line

As we can see from the picture that we want to clip the line having both the end points are (x1, y1) & (x2, y2). If we parametrize this line by assuming parameter t=0 at P (x1, y1) and t=1 at Q (x2, y2). Then,

X=x1+(x2-x1) *t

Y=y1+(y2-y1) *t

Algorithm:

- Set t1=0 and t2=1
- XWmin ≤ x1+(x2-x1) *t ≤ XWmax
- YWmin ≤ y1+(y2-y1) *t ≤YWmax
- Rewrite the above inequalities in the form $tP_k \leq q_k$

$-t\Delta x \leq x1 - XWmin$ …. (1)

$t\Delta x \leq XWmax - X1$…. (2)

$t\Delta y \leq YWmax - y1$…. (3)

$-t\Delta y \leq y1 - YWmin$…. (4)

From (1),

p1=$-\Delta x$, q1= $x1 - XWmin$

From (2),

p2=Δx, q2= $XWmax - X1$

From (3),

p4=Δy, q4= $YWmax - y1$

From (4),

p3=$-\Delta y$, q3= $y1 - YWmin$

So, $tP_k \leq q_k$ where k=1, 2, 3, 4

Case 1) if $P_k = 0$ then line will go parallel to coordinate axis. •

q_k >0 line will be inside the clipping window

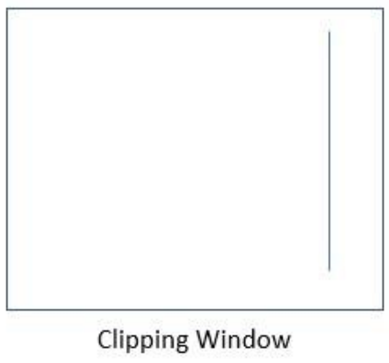


Figure 2.7 Test Case-1

- q_k <0 line will be outside the clipping window

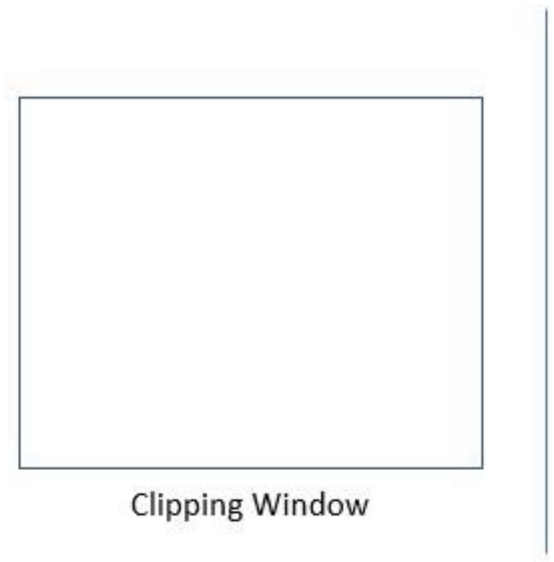


Figure 2.8 Test Case-2

Case 2) if $P_k < 0$, t1=max (0, qk/pk) and t2=1

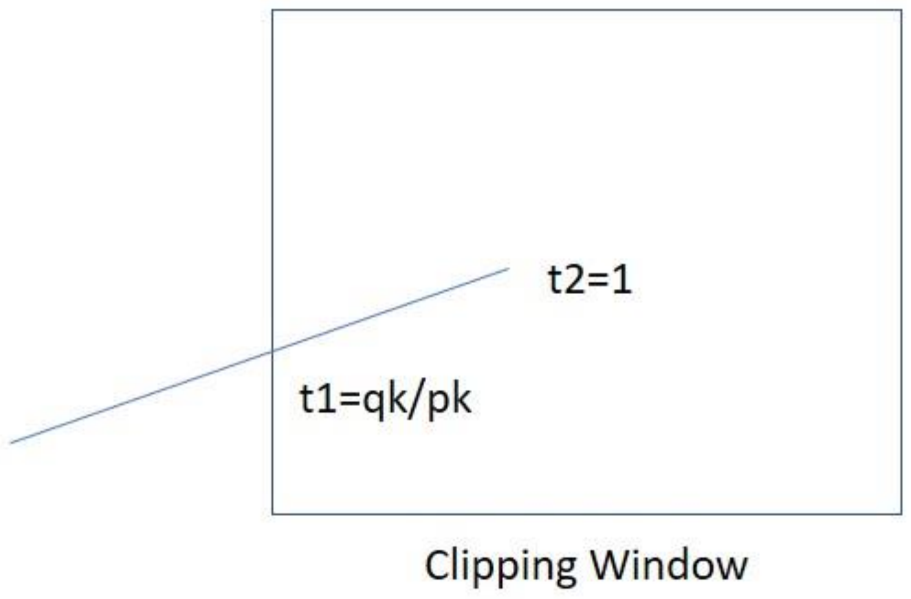


Figure 2.9 Test Case-3

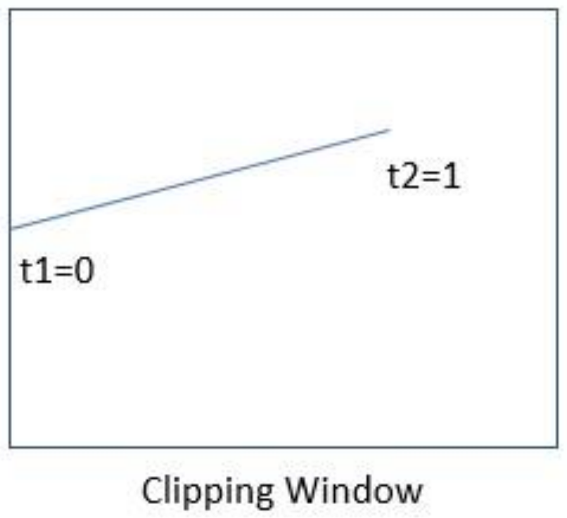


Figure 2.10 Test Case-4

Case 3) if $P_k > 0$, t2=min (1, qk/pk) and t1=0

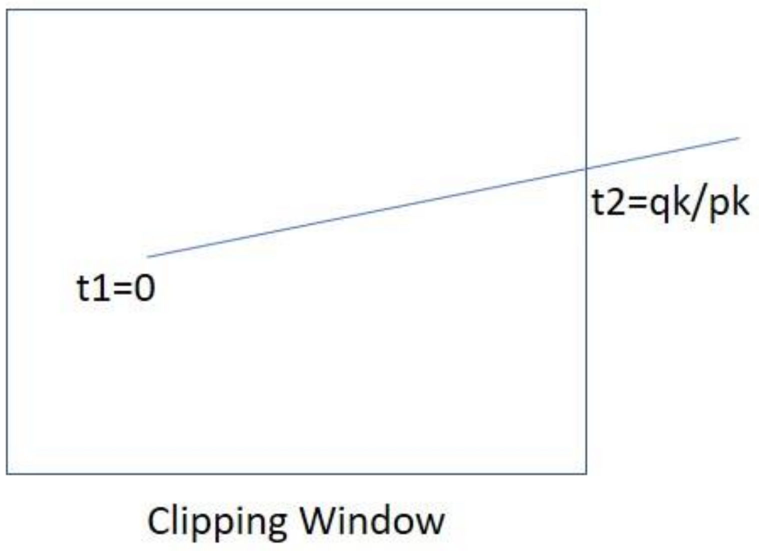


Figure 2.11 Test Case-5

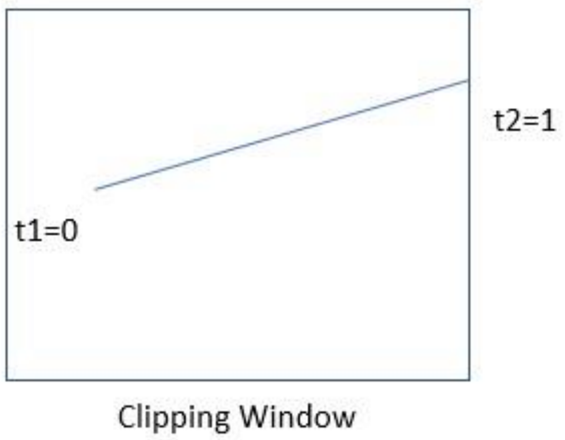


Figure 2.12 Test Case-6

2.4 Line Clipping Algorithm for 3D Space

This paper has proposed another line cutting calculation for 3D space against a cuboid which isn't produced dependent on Cohen-Sutherland or Liang-Barsky line cutting calculations. The proposed calculation depends on a recently proposed basic hypothesis created utilizing essential numerical ideas. All most all the 3D line cut-out calculations include three stages to check whether a line section lies totally inside the cut-out volume or lies totally outside the cut-out volume or convergence estimations when it isn't totally inside or outside. The proposed calculation doesn't pursue these means. The calculation was tried for an enormous number of irregular line fragments and the outcomes demonstrated that the new 3D space line cutting calculation performs superior to the Cohen-Sutherland 3D line section calculation as far as reality.

This paper says that in 3-D space, using liang-barsky clipping algorithm we can’t clip the line but using the same concept we will clip not only the line but any general 2-D curve.

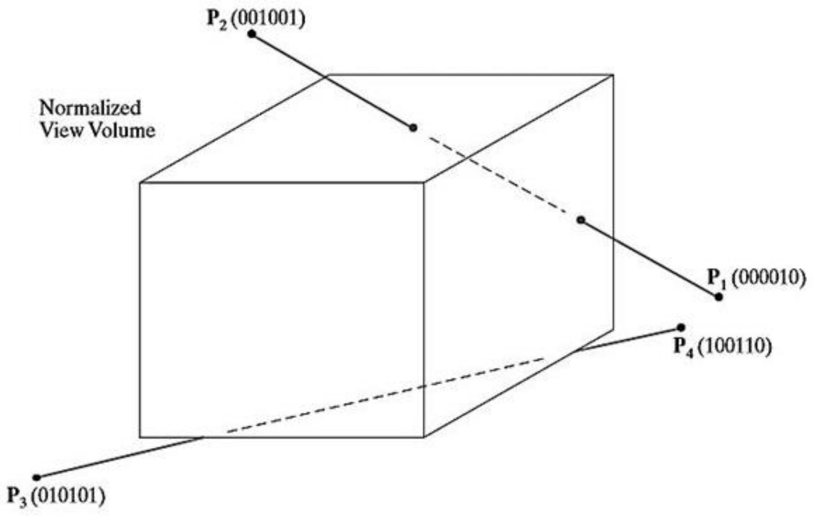


Figure 2.13 Clipping of 3-D Line

2.5 Comparison among Various Line Clipping Algorithm

In this paper, I have read the comparison among various well-known clipping algorithm line Nicholl Lee Nicholl clipping algorithm and all mentioned above.

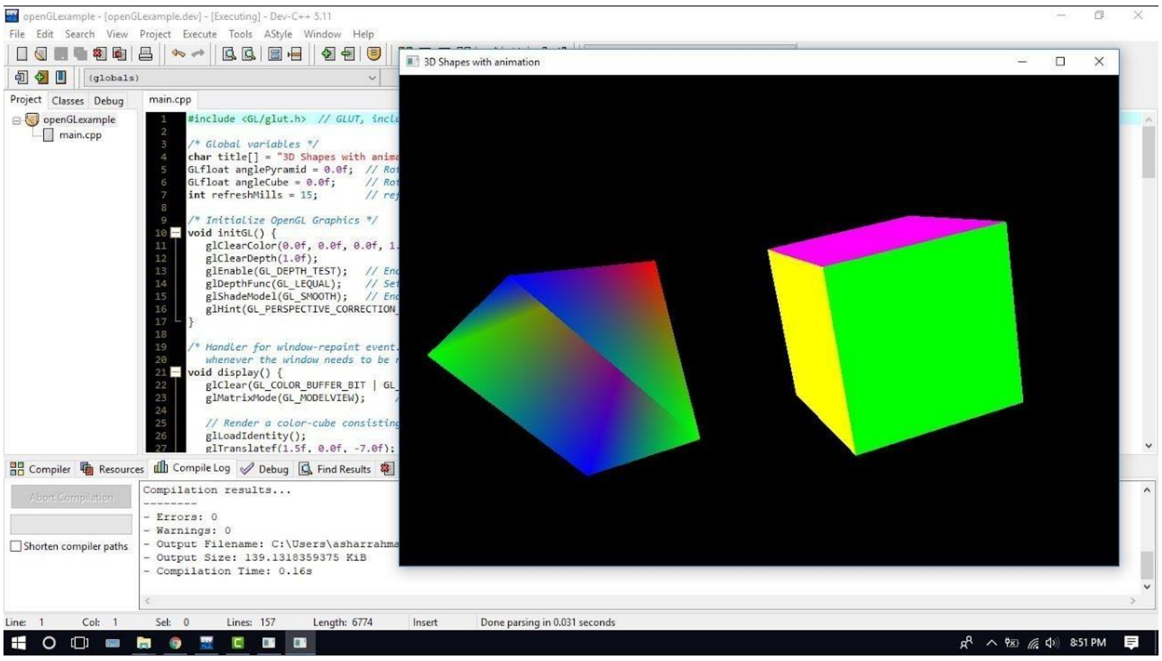
Here we have seen the comparison of various Line clipping algorithms based on its working principles. One method for improving the efficiency of a clipping algorithm is to minimize the repetition of algorithm. Here region codes are being used to identify the position of line. Another algorithm minimizes intersection calculations. To achieve this goal an efficient clipping algorithm is presented here. One is on the basis of testing x-y plane to reduce intersection calculation. Our algorithm with reducing these calculations can avoid the repetition of these algorithms.

CHAPTER-3

System Development

3.1 OpenGL: Open Graphics Library(OpenGL) is an API (Application Programming Interface) which is cross platform and cross language. It's used for rendering 2-D or 3-D pictures (Vector Graphics).

OpenGL is an advancing API. New forms of the OpenGL details are consistently discharged by the Khronos Group, every one of which stretches out the API to help different new highlights. The subtleties of every variant are chosen by agreement between the Group's individuals, including illustrations card producers, working framework planners, and general innovation organizations, for example, Mozilla and Google.

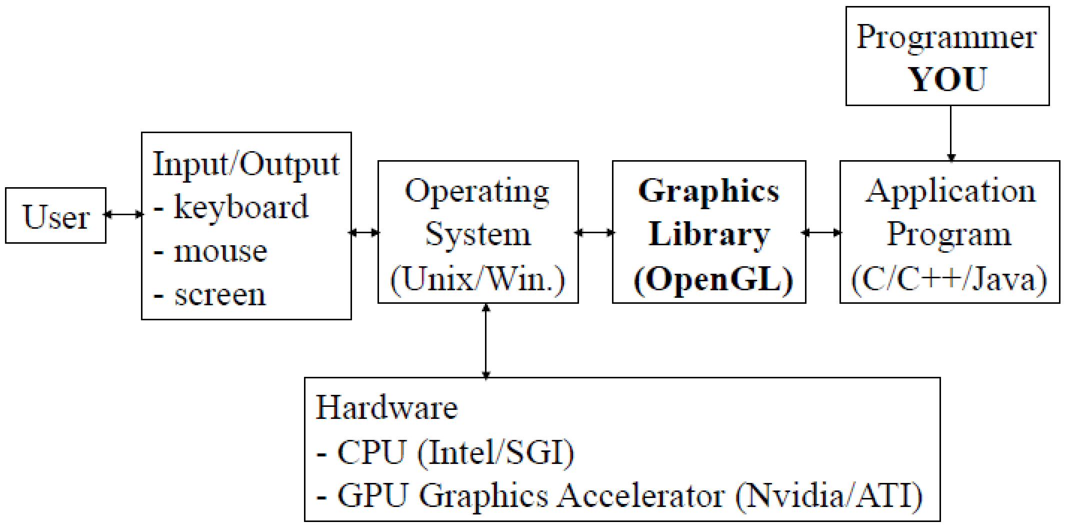

openGLexample - [openGLexample.dev] - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug
openGLexample
main.cpp
3D Shapes with animation
Compiler Resources Compile Log Debug Find Results
Abort Compilation
Shorten compiler paths
Line: 1 Col: 1 Sel: 0 Lines: 157 Length: 6774 Insert Done parsing in 0.031 seconds
8:51 PM

```
#include <GL/glut.h>  // GLUT, inclu

/* Global variables */
char title[] = "3D Shapes with anima
GLfloat anglePyramid = 0.0f;  // Rot
GLfloat angleCube = 0.0f;     // Rot
int refreshMills = 15;        // ref

/* Initialize OpenGL Graphics */
void initGL() {
   glClearColor(0.0f, 0.0f, 0.0f, 1.
   glClearDepth(1.0f);
   glEnable(GL_DEPTH_TEST);   // Ena
   glDepthFunc(GL_LEQUAL);    // Set
   glShadeModel(GL_SMOOTH);   // Ena
   glHint(GL_PERSPECTIVE_CORRECTION_
}

/* Handler for window-repaint event.
   whenever the window needs to be r
void display() {
   glClear(GL_COLOR_BUFFER_BIT | GL_
   glMatrixMode(GL_MODELVIEW);     /

   // Render a color-cube consisting
   glLoadIdentity();
   glTranslatef(1.5f, 0.0f, -7.0f);
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\asharrahma
- Output Size: 139.1318359375 KiB
- Compilation Time: 0.16s
```

Fig 3.1 OpenGL

3.2 API for 2D/3D Graphics:


Programmer
YOU
User
Input/Output
- keyboard
- mouse
- screen
Operating System (Unix/Win.)
Graphics Library (OpenGL)
Application Program (C/C++/Java)
Hardware
- CPU (Intel/SGI)
- GPU Graphics Accelerator (Nvidia/ATI)

CHAPTER-4

Algorithms

In this project, we are deriving a new algorithm so that we would be able to clip any general (single parametrized) 2-D curve against 2-D clipping window as well as against 3-D clipping volume.

4.1 Clipping circular shape against 2-D Clipping window

Suppose we want to clip the given 2-D arbitrary shape whose equation is already given,

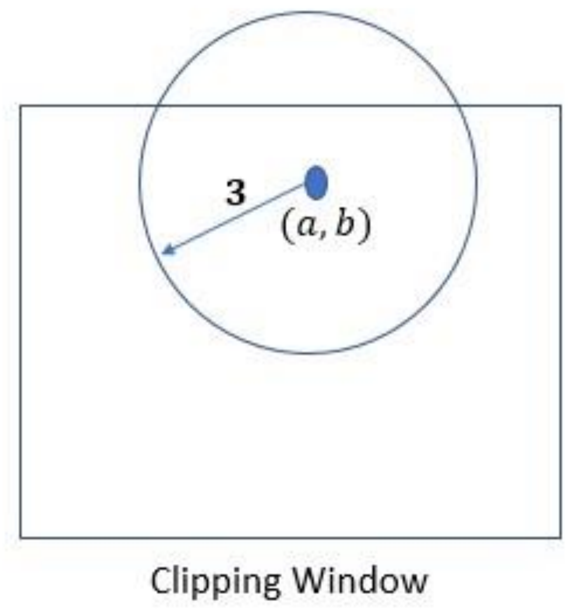


Figure 4.1 Circle before clipping

Here we want to clip the circular part which is inside the clipping window.

It's given that circle having centre: (a, b) and radius 3 unit. Equation of circle: $(x - a)^2 + (y - b)^2 = 9 \dots$ (1)

Equation of boundaries of clipping window are given as x=xmin, x=xmax, y=ymin, y=ymax

Here if you notice that if we use normal line clipping algorithm then it will involve lot of calculations so we will be doing using modified clipping algorithm.

Parametrization of circle: x=a+3cosθ

and y=b+3sinθ

Algorithm:

Solve following inequality simultaneously to find all the desired value of θ_1 and θ_2 in for which circle lies inside the clipping window.

1. XWmin$\leq$ a + 3cosθ $\leq$XWmax
2. YWmin$\leq$ b + 3sinθ $\leq$YWmax
3. 0$\leq \theta \leq 2\pi$
4. After solving this we get two values of θ called θ_1 and θ_2, and for each value we get one corresponding point on circle. Between both these points where $\theta_1 \leq \theta \leq \theta_2$ we retain the curve and rest of the curve will be removed.

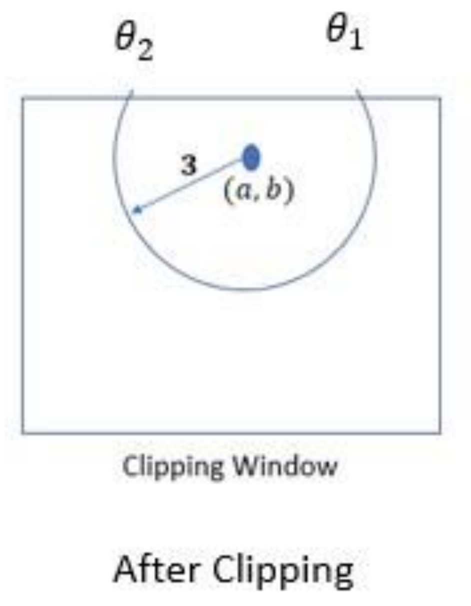


Figure 4.2 Circle after clipping

4.2 Clipping 3-D line against 3-D Clipping volume

Let's take another example where we are clipping a 3-D line against clipping volume (Cuboid).

Consider a 3-D line passing through a cuboid as shown in figure and we want to clip the portion which is inside the clipping volume.

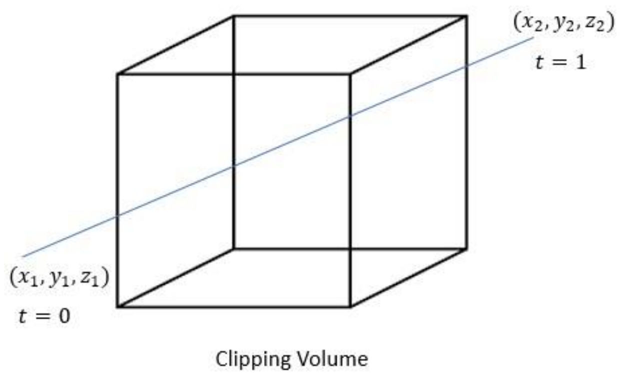


Figure 4.3 3-D line before clipping

Let's parametrize the line as,

x=x1+(x2-x1)*t

y=y1+(y2-y1)*t

z=z1+(z2-z1)*t

it's given that equation of boundaries of clipping volume are x=xmin, x=xmax, y=ymin, y=ymax, z=zmin, z=zmax.

For clipping the line inside the volume all these inequalities should hold simultaneously.

- xmin≤x1+(x2-x1) *t≤xmax
- ymin≤y1+(y2-y1) *t≤ymax
- zmin≤z1+(z2-z1) *t≤zmax

- $0 \leq t \leq 1$

After solving these inequalities, we will get the two boundary values of t1 and t2. Put these values in the parametrized equation to get the coordinates of the required line and join to obtain it.

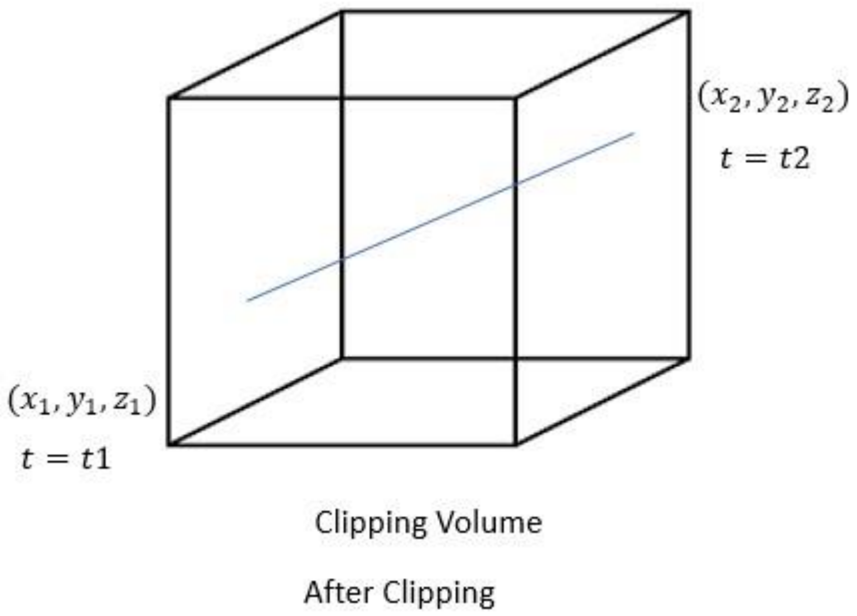


Figure 4.4 3-D line after clipping

4.3 Clipping Parabolic Shape against 2-D Clipping Window

Let's take few more examples for better understanding.

We are going to clip a parabola against a two-dimensional clipping window.

Consider a parabola $y^2 = 4ax$ is passing through a clipping window having boundary lines are x=xmin, y=ymin, x=xmax and y=ymax as shown in figure.

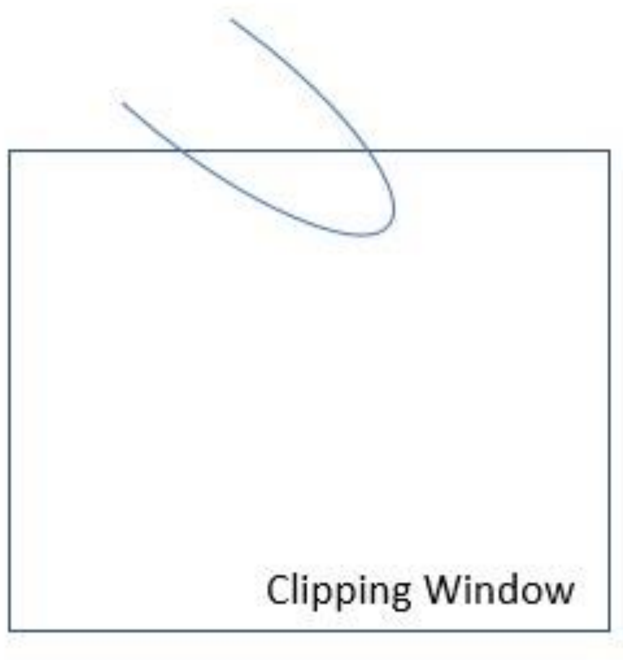


Figure 4.5 Parabola before clipping

Parametrization of parabola: $y^2 = 4ax$

x= at^2 y=2at

for clipping the parabola following inequalities holds true simultaneously.

- xmin$\leq at^2 \leq$xmax
- ymin$\leq$2at$\leq$ymax
- $0 \leq t \leq 1$

From above three inequalities we get two boundary values of t say t1 and t2. Ultimately you get the points in-between which parabola need to be retained.

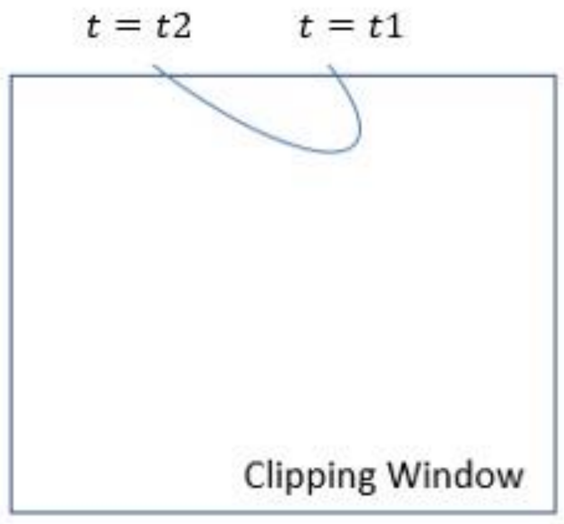


Figure 4.6 Parabola after clipping

4.4 Clipping Elliptical Shape against 2-D Clipping Window

Let’s assume an elliptical shape which we want to clip against a particular clipping window.

Consider an ellipse $\frac{x^2}{2^2} + \frac{y^2}{10^2} = 1$ and we want to clip this against the clipping window as shown in figure.

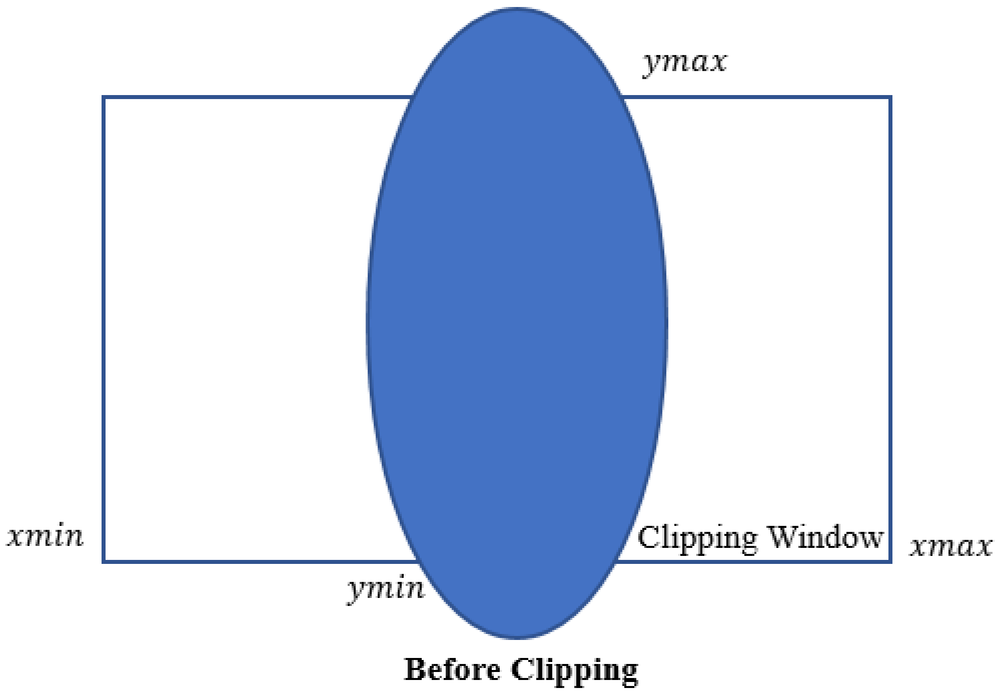


Figure 4.7 Elliptical shape before clipping

<u>Algorithm</u>:

Parametrization of Ellipse: $\frac{x^2}{2^2} + \frac{y^2}{10^2} = 1$

$$x = 2\cos u \;\; \& \; y = 10\sin u$$

For clipping the ellipse, following inequalities must hold simultaneously.

i $xmin \leq 2\cos u \leq xmax$

ii $ymin \leq 10\sin u \leq ymax$

iii $0 \leq u \leq 2\pi$

After solving these three inequalities simultaneously we get four values of u say

u_1, u_2, u_3, u_4.

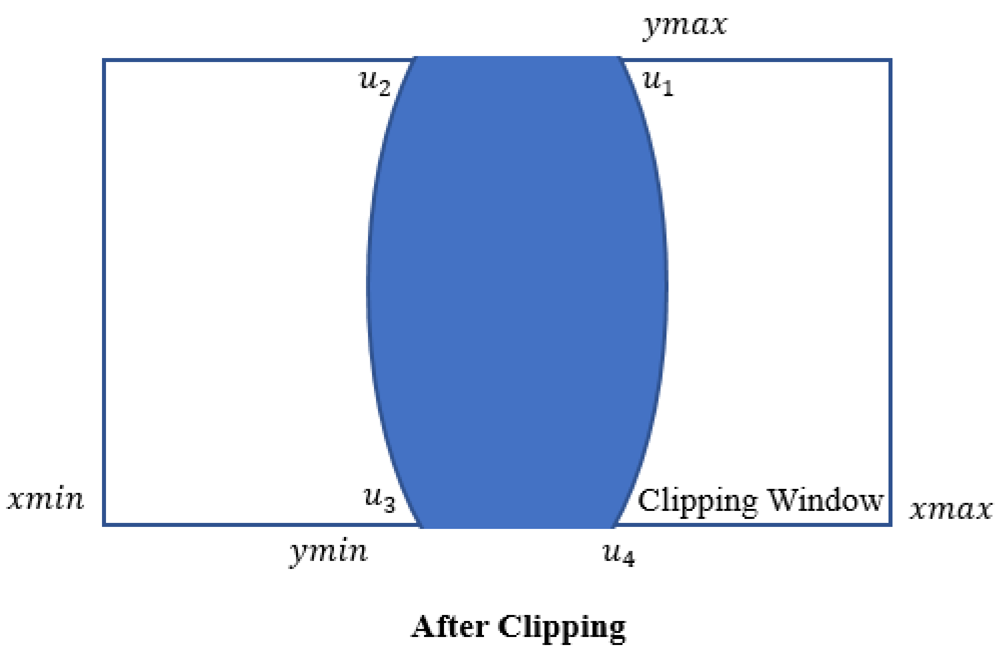


Figure 4.8 Elliptical shape after clipping

Put these four parameters in parametric equations and obtain all four coordinates and with the help of this obtain the clipped ellipse.

4.5 Clipping Spherical Shape against 3-D Clipping Volume

Let’s assume a spherical shape which we want to clip against clipping volume.

Consider a sphere of radius r having equation: $x^2 + y^2 + z^2 = r^2$

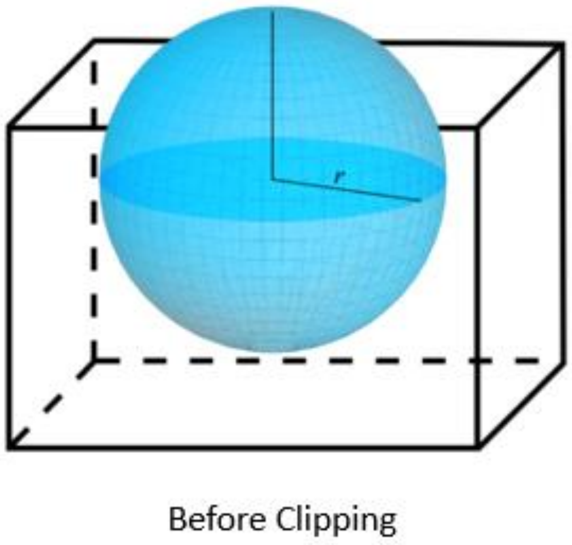


Figure 4.9 Spherical shape before clipping

Algorithm:

Let’s Parametrize the sphere as

$x = r \sin a \ cosb,$

$y = r \ sin \ a \sin b,$

$z = r \cos a.$

For clipping this shape against the given clipping volume, following inequalities must holds simultaneously.

i. $xmin \leq r \sin a \ cosb \leq xmax$

ii. $ymin \leq r \sin a \ sinb \leq ymax$

iii. $zmin \leq \cos a \leq zmax$

iv. $0 \leq b \leq 2\pi$

v. $0 \leq a \leq \pi$

By solving all the five inequalities we will get the condition for desired clipped sphere as shown below.

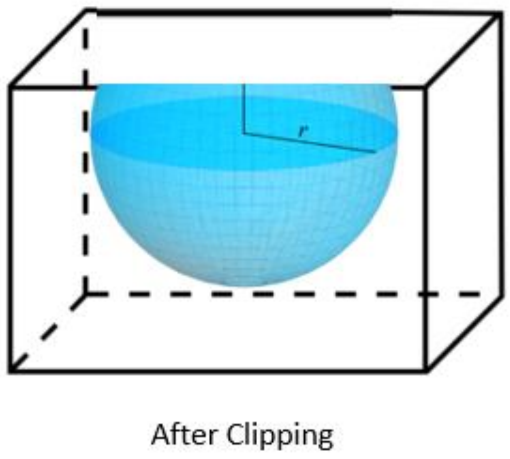
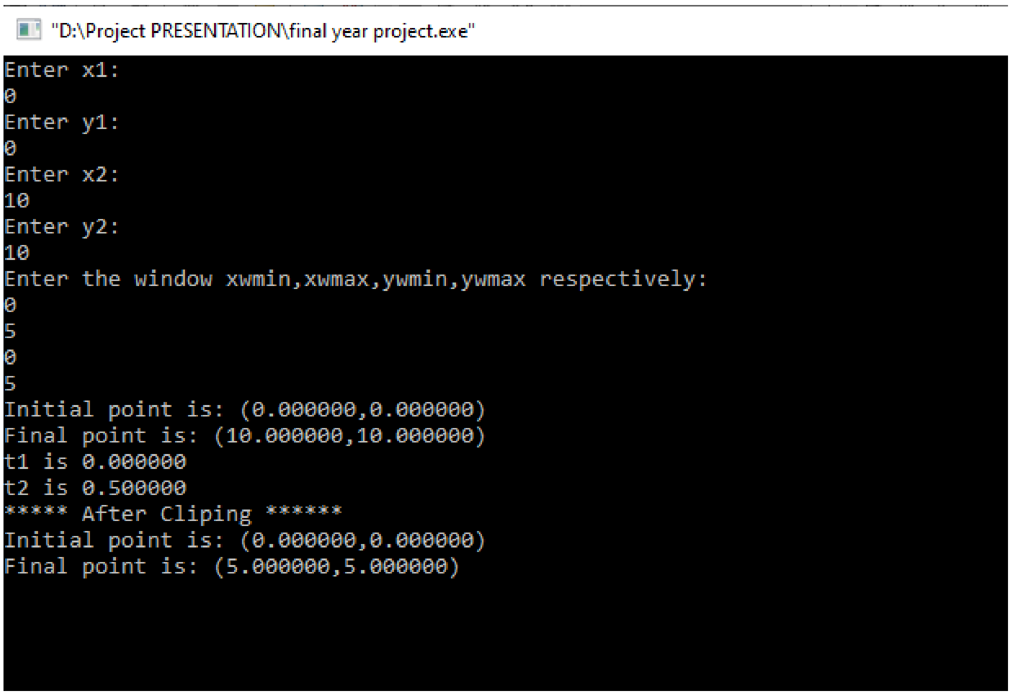


Figure 4.10 Spherical shape after clipping

CHAPTER-5

Result and Performance Analysis

5.1 Result



```
"D:\Project PRESENTATION\final year project.exe"
Enter x1:
0
Enter y1:
0
Enter x2:
10
Enter y2:
10
Enter the window xwmin,xwmax,ywmin,ywmax respectively:
0
5
0
5
Initial point is: (0.000000,0.000000)
Final point is: (10.000000,10.000000)
t1 is 0.000000
t2 is 0.500000
***** After Cliping ******
Initial point is: (0.000000,0.000000)
Final point is: (5.000000,5.000000)
```

Figure 5.1 Clipped Line against a rectangular clipping window

5.2 Conclusion

So, after finding a beautiful approach for clipping any general (single parameter dependent curve) we have reduced our calculations and performance time. When we take pictures from our camera then it converts from world coordinates to viewing coordinates but as we know sometimes we don't want full picture to be projected on device coordinates through viewing pipeline so we need to clip the original image. one simple way is to render line by line but when curves, that to be clipped, are complex then this approach is very lengthy and time taking. So, in this project we have clipped so many 2-D or 3-D curves /lines with our new approach successfully.

5.2 Future Work:

Lists of things for future work:

- To improve its performance, we will try to optimize this algorithm as much as we can
- To learn more about another graphics software where I can use this algorithm
- Implementation of this logic in various other field of graphics.

REFERENCES

[1] V. Skala, ―O (lg N) Line clipping Algorithm in E , ‖Computers and Graphics, Vol. 18, No. 4, 1994, pp. 517-527.

[2] R. A. Earnshaw et al. (eds.), New Advances in Computer Graphics © Springer-Verlag Tokyo 1989

[3] D. Hearn and M. P. Baker, ―Computer Graphics,‖ C Version, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, 1998, p. 224-248

[4] T. M. Nicholl, D. T. Lee and R. A. Nicholl, ―An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis,‖ Computers and Graphics, Vol. 21, No. 4, 1987, pp. 253-262.

[5] Wenjun Huang, ―The Line Clipping Algorithm Basing on Affine Transformation‖, Intelligent Information Management, 2010, 2,380-385, Published Online June 2010

[6] M. Cyrus and J. Beck, ―Generalized Two and Three Dimensional Clipping,‖ Computers and Graphics, Vol. 3, No. 1, 1978, pp. 23-28.

[7] C. B. Chen and F. Lu, ―Computer Graphics Basis,‖ Publishing House of Electronics Industry, Beijing, 2006, pp.167-168.

Appendix – I

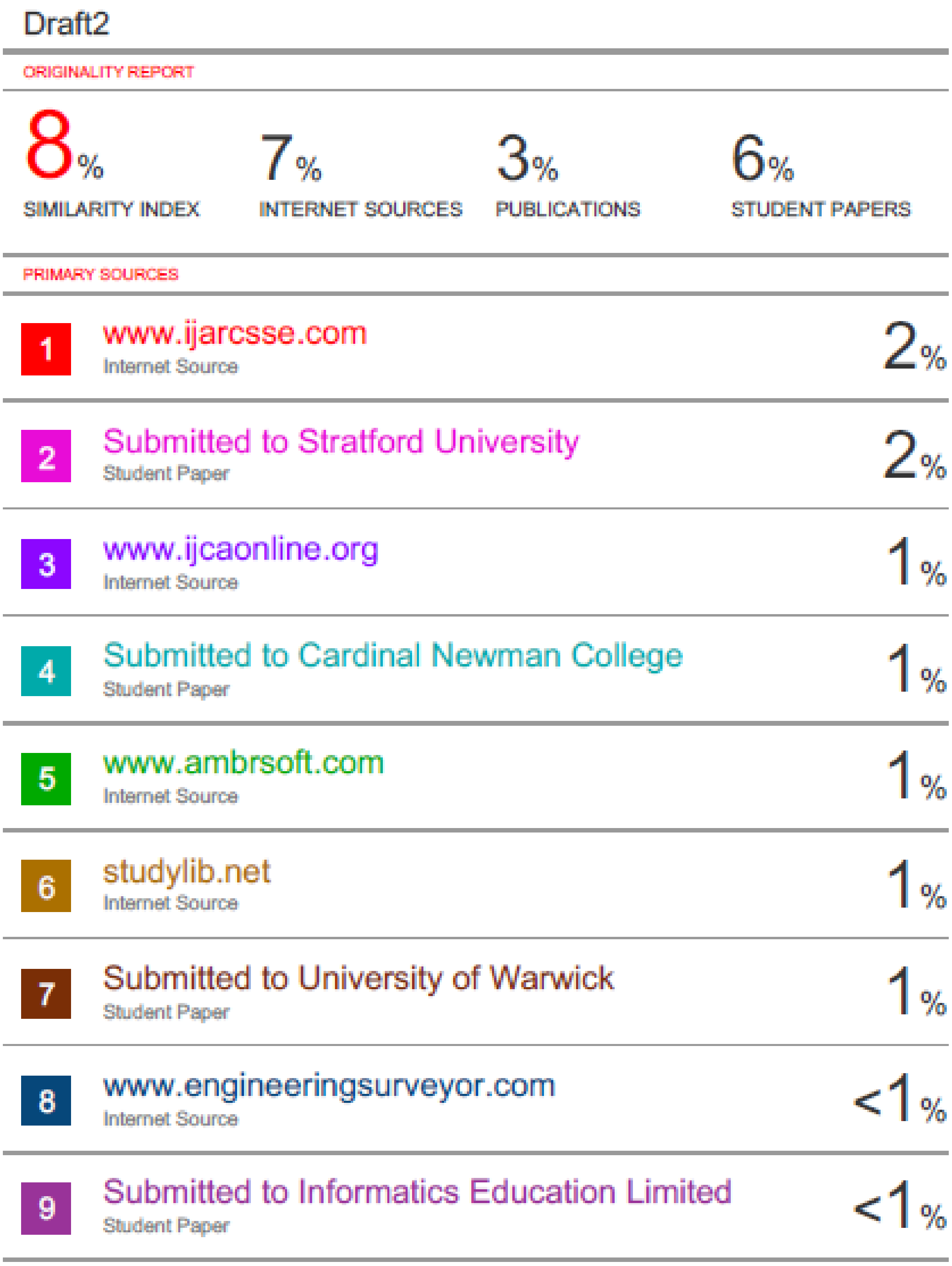
Code:

```
#include<stdio.h>
#include<conio.h>
float min(float a, float b, float c);
float max(float a, float b, float c);
float main()
{  float i, t, t1, t2,j,x1,y1,x2,y2, xwmin, xwmax, ywmin, ywmax,t3,t4, x1n,y1n,x2n,y2n
, c1=0,c2=1;
printf("Enter x1:\n");
scanf("%f",&x1);
printf("Enter y1:\n");
scanf("%f",&y1);
printf("Enter x2:\n");
scanf("%f",&x2);
printf("Enter y2:\n");
scanf("%f",&y2);
printf("Enter the window xwmin,xwmax,ywmin,ywmax respectively:\n");
scanf("%f \t %f \t %f \t %f",&xwmin,&xwmax,&ywmin,&ywmax);

printf("Initial point is: (%f,%f)\n",x1,y1);
printf("Final point is: (%f,%f)\n",x2,y2);

   for(i=1;i<=100;i++)
   {
     for(j=1;j<=100;j++)
     {
      if(x1+(i/j)*(x2-x1)==xwmin)
      t1=i/j;

      if(x1+(i/j)*(x2-x1)==xwmax )
      t2=i/j;
     }
   }

    for(i=1;i<=100;i++)
   {
     for(j=1;j<=100;j++)
     {
      if(y1+(i/j)*(y2-y1)==ywmin)
      t3=i/j;


      if(y1+(i/j)*(y2-y1)==ywmax)
```

```
       t4=i/j;
      }
   }

  t1 =  max(t1,t3, c1);
  t2 = min(t2,t4, c2);
 /*if(t3>=t1)
   {
     t1==t3;
   }
   if(t2>=t4)
   {
      t2==t4;
   }*/

   printf("t1 is %f\n",t1);
    printf("t2 is %f\n",t2);




    printf("***** After Cliping ****** \n");

    x1n=x1+(t1*(x2-x1));
    y1n=y1+(t1*(y2-y1));

    x2n=x1+(t2*(x2-x1));
    y2n=y1+(t2*(y2-y1));

    printf("Initial point is: (%f,%f)\n",x1n,y1n);
    printf("Final point is: (%f,%f)\n",x2n,y2n);
    getch();
}

float min(float a, float b, float c)
{
   float temp;
   if(a<b)
         temp=a;
      else temp=b;

      if(temp<c)
          temp=temp;
          else temp=c;

      return temp;
}
```

```
float max(float a, float b, float c)
{
   float temp;
   if(a<b)
      temp=b;
      else temp=a;
      if(temp<c)
         temp=c;
         else temp=temp;

      return temp;
}
```

Plagiarism Report



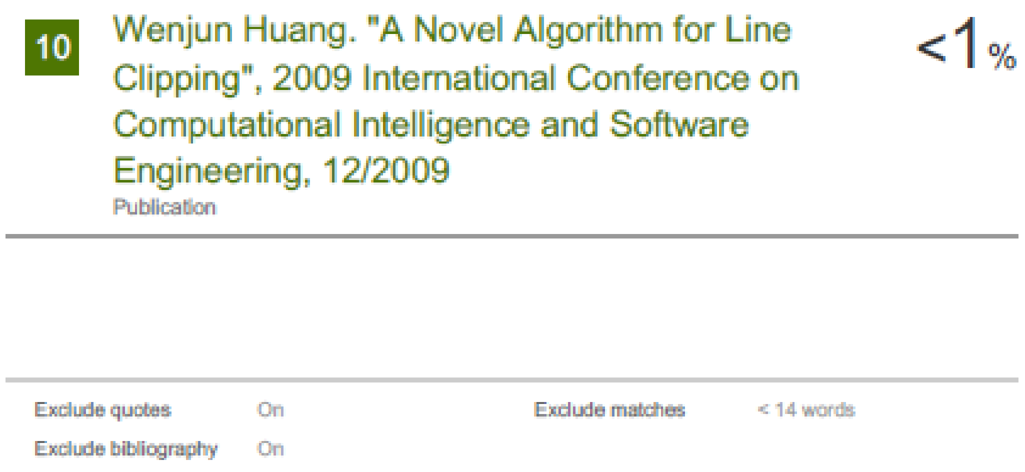
Draft2

ORIGINALITY REPORT

8%	7%	3%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ijarcsse.com Internet Source	2%
2	Submitted to Stratford University Student Paper	2%
3	www.ijcaonline.org Internet Source	1%
4	Submitted to Cardinal Newman College Student Paper	1%
5	www.ambrsoft.com Internet Source	1%
6	studylib.net Internet Source	1%
7	Submitted to University of Warwick Student Paper	1%
8	www.engineeringsurveyor.com Internet Source	<1%
9	Submitted to Informatics Education Limited Student Paper	<1%



10	Wenjun Huang. "A Novel Algorithm for Line Clipping", 2009 International Conference on Computational Intelligence and Software Engineering, 12/2009 Publication	<1%

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		