Volume 6, Issue 11, November 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

RJ - ASHI Algorithm: A New polygon/Line Clipping Algorithm for 2D Space

Raja Singh, Ashish Kumar
Indian Institute of Technology, BHU, Varanasi,
Uttar Pradesh, India

Abstract— Most of the line clipping algorithms like Cohen-Sutherland and Liang-Barsky algorithm involves a lot of calculations. This paper proposes a new line clipping algorithm for 2D space against a rectangular clipping window which is more efficient according to the data we calculated. The possible extended algorithm for 3D space is also presented. The algorithm proposed for the 2D space is compared against traditional line clipping algorithms. The algorithm was tested for a set of random line segments and polygons and the results showed that this algorithm performs better than the traditional 2D line clipping algorithm in terms of time and space complexity in cases where computation time exceeds scan time required for single scan.

Keywords— Computer Graphics, Line Clipping, 2D geometry, 3D geometry.

I. INTRODUCTION

In computer graphics, it is very important to clip an area or a volume of interest which is to be displayed on the computer monitor. This region of interest is normally a rectangle or a general polygon in two-dimension and known as the clipping window. When it comes to three-dimensional clipping, a volume is used to extract a part from a three-dimensional scene. Generally the lines are clipped by this clipping volume and it is a polyhedron. Cuboids are widely used as the clipping volume. Three-dimensional clipping is one of the most essential processes in medical applications, video games, computer aided design and many other applications. Sometimes it is necessary to discard the data that is not contained in the visible region to avoid the overflow of the internal registers of the display device. Furthermore, lesser memory consumption can be achieved from loading only a certain part of a scene to the memory by clipping unnecessary parts. Therefore, improving the efficiency of clipping algorithms has a great impact on efficiency of the overall graphics system. Cohen-Sutherland line clipping algorithm [1], Liang- Barsky line clipping algorithm [2], Cyrus-Beck line clipping algorithm [3] and Nicholl-Lee-Nicholl line clipping algorithm [4] are few of the traditional line clipping algorithms. The Cohen-Sutherland and the Liang-Barsky algorithms can be extended to three-dimensional clipping. Nicholl-Lee-Nicholl algorithm performs fewer comparisons and divisions making it faster than others [1]. However, it is difficult to extend for three-dimensional clipping. The Cohen- Sutherland algorithm is one of the simplest and most widely used clipping algorithms in computer graphics. This algorithm works very fast in situations like when the line segment is completely inside or outside of the clipping window. When the line segment is not completely inside or outside, the algorithm becomes inefficient due to repeated calculations [1]. This algorithm can be easily extended to three–dimensional clipping, but the space is needed to divide into 27 mutually exclusive volumes, when the clipping volume is a cube or a cuboid. Also each volume is assigned a region code [11]. Throughout the clipping process those region codes should be stored in the memory. So in terms of space and time complexity Cohen – Sutherland algorithm is inefficient for complex problems. According to Hearn and Baker [1] all most all the 2D, 3D line clipping algorithms involve three steps: -

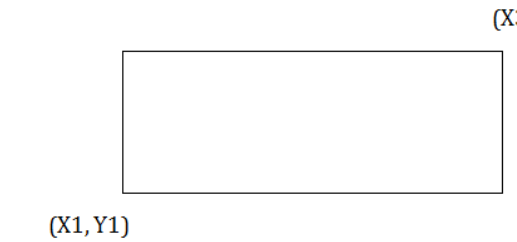
1. for a given line segment check whether it lies completely inside the clipping volume.
2. If not check whether it lies completely outside the clipping volume.
3. Otherwise perform intersection calculations with one or more clipping planes.

These three steps lead the algorithm to perform many calculations. In the proposed 2D space line clipping algorithm, the traditional three step procedure has not been used.

II. METHODOLOGY

This section presents the proposed line clipping algorithm. For line clipping, a rectangular clipping window is considered. Following conventions have been used to label the rectangular window.

Let lower left coordinate of the clipping window be (x1, y1) and upper right corner coordinates be (x3, y3).



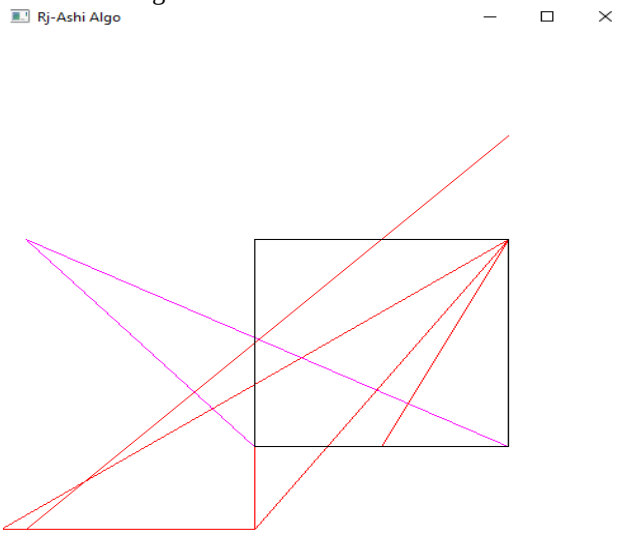
Clipping window

Proposed algorithm –

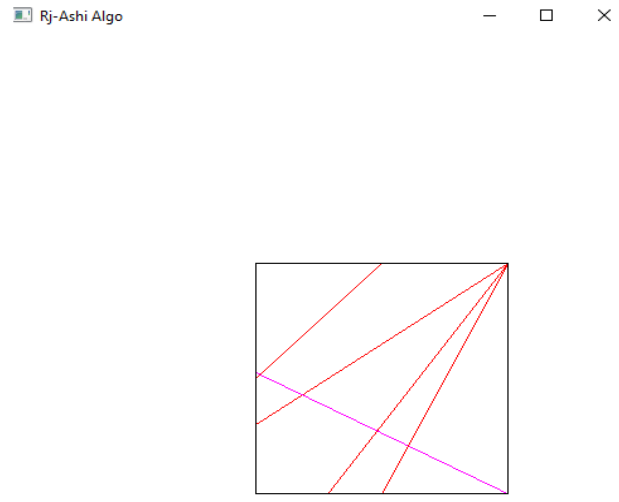
This algorithm uses two scans to clip the region instead of using just using single scan like traditional algorithms and hence is very useful in complex cases where computation time exceeds the time required for single scan this is a two-step algorithm.

First step - Draw the lines and polygons as if no clipping window is there in a single scan along with the clipping window, that means the region that shows clipping window along with drawn polygons and lines as shown in figure

Second step - Then on the second scan as we draw the pixel we check whether the pixel is inside or outside the clipping region if it is outside then we color it with the given background color else we do nothing . It just uses the simple fact that instead of using a single scan we can use two scans as single scan for clipping thus reducing the computations and result will be as shown in figure 2.



Unclipped area drawn during first scan



Clipped area draw during second scan

Implementation of the proposed algorithm -

Following work has been done in C++ using OpenGL Graphics Library.

First draw whatever polygon you want to draw using regular methods in OpenGL.

Ex- code to draw the shown figure-a has been given below


```
x1=200, y1=100, x2=200, y2=300, x3=400, y3=300, x4=400, y4=100 ;

glColor3f (3.0, 0.0, 0.0);
glBegin (GL_LINE_LOOP);
        glVertex2f (20, 20);
        glVertex2f (400, 400);
glEnd ();

glColor3f (5.0, 0.0, 0.0);          //changing color
glBegin (GL_LINE_LOOP);     // draw the polygon
        glVertex2f (200, 20);
        glVertex2f (x2, y2);
        glVertex2f (x3, y3);
        glVertex2f (0, 20);
glEnd ();

glColor3f (3.0, 0.0, 2.0);
glBegin (GL_LINE_LOOP);     // draw the polygon
        glVertex2f (x1, y1);
        glVertex2f (20, y2);
        glVertex2f (20, y3);
        glVertex2f (x4, y4);
glEnd ();

glColor3f (3.0, 0.0, 0.0);
glBegin (GL_LINE_LOOP);     // draw the polygon
        glVertex2f (x1, y1);
        glVertex2f (x2, 20);
        glVertex2f (x3, y3);
        glVertex2f (300, y4);
glEnd ();

glColor3f (0.0, 0.0, 0.0);
glBegin (GL_LINE_LOOP);     // draw the polygon
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        glVertex2f(x3, y3);
        glVertex2f(x4, y4);
glEnd ();
```

Now we call a function scan for scanning the drawn figure-a for the second time.

The function changes the color of the pixel to the background color (white in our case) if pixel is outside the clipping window else doesn't touch it.

(x1 , y1) and (x3, y3) are the coordinates of the lower left and upper right corner of rectangular clipping window and 500 is the window size in our case.

```
void scan(float x1,float y1,float x3,float y3)
{
        for (int i=0;i<500;i++)
       {
                    for(int j=0;j<500;j++)
                   {
                            if(x1<=i && x3>=i && y1<=j && y3>=j ){}// checking whether its inside
                                                                    //rectangular window or not
                            else
                            drawpixel (i, j);   //function to draw a pixel at a given coordinate i,j
                   }
       }
}
```

Complexity of algorithm –

It is an algorithm where time taken to draw the background i.e. polygons etc. is taken in to account hence its complexity depends on the polygon, line drawing algorithm used in the program/code where as other part is a constant time algorithm, i.e. time taken to scan pixel by pixel given window and checking whether it is inside or outside clipping region.

O (algo) =O (draw polygons/lines) + O (1)

{For scanning second time and checking whether pixel is inside or not}

For 3D space -

This algorithm can be easily extended for 3D spaces for instance if we have a cuboidal clipping window in a 3D space then the method would be same first draw polygons and lines in 3D space then during second scan we would check whether the pixel are inside or outside the clipping window for cuboidal volume this would be pretty easy as can easily check whether a pixel (i, j, k) is inside the clipping window or not , if we are given coordinates of cuboid or length breadth and height of cuboid, complexity would remain the same as in 2d space.

III. RESULTS AND DISCUSSION

The proposed algorithm was tested for random test cases of line segments. The test results indicated that it performs well in complex cases. In order to validate the algorithm, it was compared against the Cohen-Sutherland 2D line clipping algorithm.

The following hardware and software were used for testing.

Computer: Intel(R) Pentium(R) CPU B940 @ 2.00 GHz; 2.

00 GHz, 1.85 GB usable RAM, 64 bit operating system

IDE Details: code blocks C++, OpenGL graphics library;

We used the clock() function in both cases to calculate the number of clock cycles it takes to process and display the result using OpenGL graphics library and the system had same load in both cases -

Number of random line segments	Cohen-Sutherland (time in clock cycle)	Rj-ashi algorithm (proposes algo) (time in clock cycle)
10	35	25
100	47	30
1000	54	40
10000	62	43
100000	73	45
1000000	198	50
10000000	1598	55

Results shows that the second scan of proposed algorithm took only 4-5 clock cycles whereas computation time in Cohen - Sutherland is comparably quite large in complex cases hence giving our algorithm overall better performance. Our algorithm is comparable when the number of line segment is less in number or we can say the case is not complex but in complex cases our algorithm is better than the Cohen - Sutherland algorithm i.e. in realistic scenario's where we want the clipping to be done on complex cases then our algorithm will be more useful and will give better performance.

IV. CONCLUSIONS

According to the test results, the proposed 2D line clipping algorithm is faster than the traditional algorithms where computational time exceeds the time taken for a single scan and it takes almost comparable time in simple cases. Therefore, this algorithm can be successfully used in 2D applications where line clipping is involved. This algorithm can further be extended to clip 3D regions in a polyhedron volume and polygonal - region in case of 2D regions.

REFERENCES –

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

[2] Wenjun Huang, ―The Line Clipping Algorithm Basing on Affine Transformation‖, Intelligent Information Management, 2010, 2,380-385, Published Online June 2010 (http://www.SciRP.org/journal/iim)

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

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

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

[7] S.R. Kodituwakku, K.R. Wijeweera, M.A.P. Chamikara. An efficient algorithm for line clipping in computer graphics programming; will appear in Ceylon Journal of Science (Physical Sciences), 2012.

[8] You-Dong Liang, Brian A. Barsky, Mel Slater. Some Improvements to a Parametric Line Clipping Algorithm. pp. 2.

[9] Patrick-Gilles Maillot. Model Clipping Triangle Strips and Quad Meshes, Sun Microsystems, Inc.

[10] Fuhua Cheng, Yue-Kwo Yen. A Parallel Line Clipping Algorithm and its Implementation.

[11] J.D. Foley and A. van Dam, Fundamentals of Interactive Computer Graphics, Addison Wesley, Reading, Mass, 1982. pp. 227.

ABOUT AUTHORS

Raja Singh is an undergraduate current pursuing degree in B.Tech in computer science and engineering in IIT (B.H.U) Varanasi (India). His research interests include computer graphics, image processing, computer vision, and artificial intelligence. Email id--raja.singh.cse14@itbhu.ac.in

Ashish Kumar is an undergraduate current pursuing degree in I.D.D in computer science and engineering in IIT (B.H.U) Varanasi (India). His research interests include computer graphics, image processing, computer vision, and artificial intelligence. Email id --ashish.kumar.cse14@itbhu.ac.in

