Volume 2, Issue 5, May 2012 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

An Efficient Line Clipping Algorithm for 3D Space

R. Kodituwakku
Dept. of Statistics & Computer Science
Faculty of Science
University of Peradeniya

K. R. Wijeweera
Dept. of Statistics & Computer Science
Faculty of Science
University of Peradeniya

M. A. P. Chamikara
Post Graduate Institute of Science
Faculty of Science
University of Peradeniya
pathumchamikara@gmail.com

Abstract— **This paper proposes a new line clipping algorithm for 3D space against a cuboid which is not generated based on Cohen-Sutherland or Liang-Barsky line clipping algorithms. The proposed algorithm is based on a newly proposed simple theory developed using basic mathematical concepts. All most all the 3D line clipping algorithms involve three steps to check whether a line segment lies completely inside the clipping volume or lies completely outside the clipping volume or intersection calculations when it is not completely inside or outside. The proposed algorithm does not follow these steps. The algorithm was tested for a large number of random line segments and the results showed that the new 3D space line clipping algorithm performs better than the Cohen-Sutherland 3D line clipping algorithm in terms of time and space.**

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 [2]. 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 [10]. Furthermore, lesser memory consumption can be achieved from loading only a certain part of a scene to the memory by clipping unnecessary parts [9]. 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 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 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 3D space line clipping algorithm, the traditional three step procedure has not been used. A new efficient 2D space line clipping algorithm was introduced in an earlier stage of this research [7]. The Pseudo code of the 2D space line clipping algorithm is as follows.

```
Begin
//Calculating m and c
m = (y[1]-y[0])/(x[1]-x[0]); //Gradient of the line segment
c= (x[0]*y[1]-x[1]*y[0])/(x[0]-x[1]);//Y-intercept of the line segment
For i=0 to i=1 //For each end point of the line segment
```

```
 If x[i] < minx //End point is in the -ve side of x=minx line
         //Calculating the intersection point with x=minx line
         x[i] = minx;
         y[i] = m*minx + c;
 ElseIf x[i] > maxx //End point is in the +ve side of x=maxx line
         //Calculating the intersection point with x=maxx line
         x[i] = maxx;
         y[i] = m*maxx + c;
 EndIf

 If y[i] < miny //End point is in the -ve side of y=miny line
         //Calculating the intersection point with y=miny line
         x[i] = (miny-c)/m;
         y[i] = miny;
 ElseIf y[i] > maxy //End point is in the +ve side of y=maxy line
         //Calculating the intersection point with y=maxy line
         x[i] = (maxy-c)/m;
         y[i] = maxy;
 EndIf
EndFor

// Initial line is completely outside
If (x[0]-x[1]<1) AND (x[1]-x[0]<1) Then //x-coordinates are equal
         //Do nothing
Else //x-coordinates are not equal
         //Save the line with end points (x[0],y[0]),(x[1],y[1])
EndIf
End
```

Where, the end points of the line segment are A=(x[0], y[0]) and B=(x[1], y[1]), and conventions depicted in Figure 1 have been used to label the rectangular clipping window.

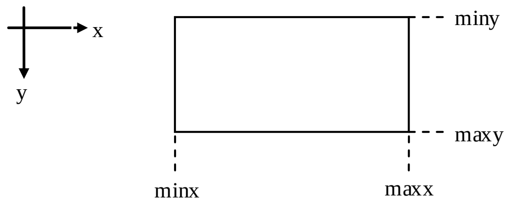
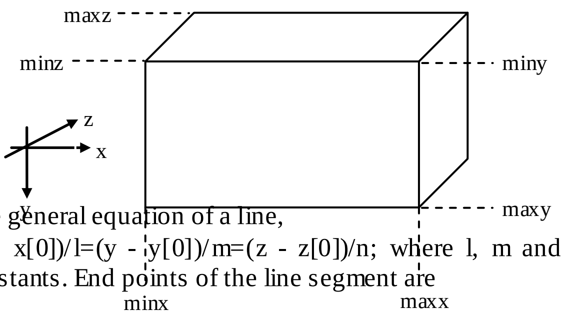


Fig.1. 2D Clipping window

In this paper, a three – dimensional extension of that algorithm is proposed. It is an extension of the above 2D clipping algorithm.

II. METHODOLOGY

This section presents the proposed line clipping algorithm and analyses its performance. For line clipping, a cuboid clipping volume is considered. Figure 2 depicts the conventions that have been used to label the volume.



The general equation of a line,
(x - x[0])/l=(y - y[0])/m=(z - z[0])/n; where l, m and n are constants. End points of the line segment are

Fig.2. Clipping volume

A=(x[0], y[0], z[0]) and B=(x[1], y[1], z[1]).

1) Mathematical background of the proposed algorithm

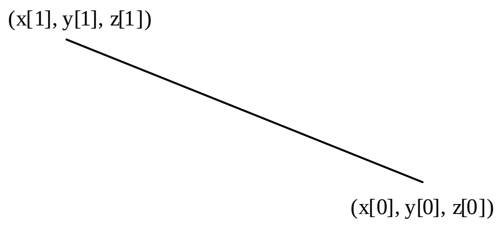


Fig.3. A straight line

Equation of the line shown in Figure 3 is
(x - x[0])/l=(y - y[0])/m=(z - z[0])/n; where l, m and n are constants.
By substituting, (x[1], y[1], z[1]), the equation becomes
(x[1] - x[0])/l=(y[1] - y[0])/m=(z[1] - z[0])/n;
From (x[1] - x[0])/l = (y[1] - y[0])/m
=> (x[1] - x[0])/(y[1] - y[0]) = l/m = a;
From (y[1] - y[0])/m = (z[1] - z[0])/n
=> (y[1] - y[0])/(z[1] - z[0]) = m/n = b;
Therefore, ab = (l/m)(m/n) = l/n;

<u>Now consider the intersection of the line and the x = p plane depicted in Figure 4.</u>

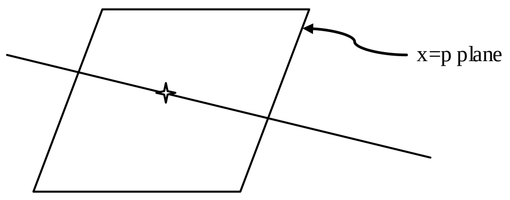


Fig.4. A straight line intersects with x=p plane

Consider, (x - x[0])/l = (y - y[0])/m = (z - z[0])/n;
From (y - y[0])/m = (x - x[0])/l => (y - y[0])/m = (p - x[0])/l
=> y = (p - x[0])(m/l) + y[0];
From (x - x[0])/l = (z - z[0])/n => (z - z[0])/n = (p - x[0])/l
=> z = (p - x[0])(n/l) + z[0];
Therefore, the point of intersection is
{p, (p - x[0])/a + y[0], (p - x[0])/(ab) + z[0]}

<u>Similarly consider the intersection of the line and the y = q plane shown in Figure 5.</u>

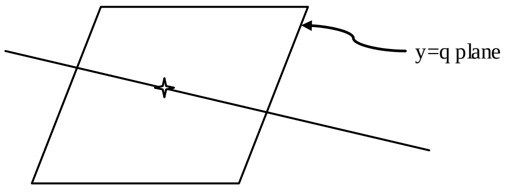


Fig.5. A straight line intersects with y=q plane

Consider, (x-x[0])/l=(y-y[0])/m=(z-z[0])/n;
From (x - x[0])/l = (y - y[0])/m => (x - x[0])/l = (q - y[0])/m
=> x = (q - y[0])(l/m) + x[0];

From (z - z[0])/n = (y - y[0])/m => (z - z[0])/n = (q - y[0])/m => z = (q - y[0])(n/m) + z[0];
Therefore, the point of intersection is
{a(q - y[0]) + x[0], q, (q - y[0])/b + z[0]}

Finally, consider the intersection of the line and the z = r plane shown in Figure 6.

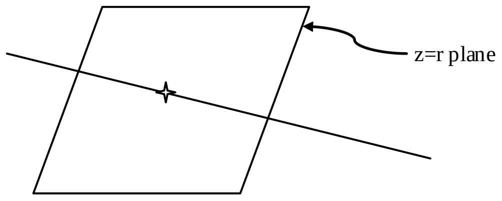


Fig.6. A straight line intersects with x=p plane

Consider (x - x[0])/l = (y - y[0])/m = (z - z[0])/n;
From (x - x[0])/l = (z - z[0])/n => (x - x[0])/l = (r - z[0])/n => x = (r - z[0])(l/n) + x[0];
From (y - y[0])/m = (z - z[0])/n => (y - y[0])/m = (r - z[0])/n => y = (r - z[0])(m/n) + y[0];
Therefore, the point of intersection is
{ab(r - z[0]) + x[0], b(r - z[0]) + y[0], r}

2) Pseudo code of the proposed algorithm

All the symbols used in the following pseudo code are shown in Figure 2 or provided in section 2.1.

```
Begin
//Calculating a and b
a = (x[1] - x[0])/(y[1] - y[0]);
b = (y[1] - y[0])/(z[1] - z[0]);
For i=0 to i=1 //For each end point of the line segment
 If x[i] < minx Then //End point is in -ve side of x=minx plane
        //Calculating the intersection point with x=minx plane
        y[i] = (minx - x[0])/a + y[0];
        z[i] = (minx - x[0])/(a*b) + z[0];
        x[i] = minx;
 ElseIf x[i] > maxx Then//End point is in +ve side of x=maxx plane
        //Calculating the intersection point with x=maxx plane
        y[i] = (maxx - x[0])/a + y[0];
        z[i] = (maxx - x[0])/(a*b) + z[0];
        x[i] = maxx;
 EndIf
 If y[i] <miny Then //End point is in -ve side of y=miny plane
        //Calculating the intersection point with y=miny plane
        x[i] = a*(miny - y[0]) + x[0];
        z[i] = (miny - y[0])/b + z[0];
        y[i] = miny;
 ElseIf y[i] > maxy Then //End point is in +ve side of y=maxy plane
        //Calculating the intersection point with y=maxy plane
        x[i] = a*(maxy - y[0]) + x[0];
        z[i] = (maxy - y[0])/b + z[0];
        y[i] = maxy;
 EndIf
 If z[i] < minz Then // End point is in -ve side of z=minz plane
        //Calculating the intersection point with z=minz plane
        x[i] = a*b*(minz - z[0]) + x[0];
        y[i] = b*(minz - z[0]) + y[0];
        z[i] = minz;
 ElseIf z[i] > maxz Then//End point is in +ve side of z=maxz plane
        //Calculating the intersection point with z=maxz plane
        x[i] = a*b*(maxz - z[0]) + x[0];
        y[i] = b*(maxz - z[0]) + y[0];
        z[i] = maxz;
 EndIf
EndFor

 // Initial line is completely outside
If (x[0] - x[1]) < 1 AND (x[1] - x[0]) < 1 Then //x-coords are equal
            // Do nothing
Else        //x-coords are not equal
   // Save the line with end points (x[0], y[0], z[0]), (x[1], y[1], z[1])
EndIf
End
```

3) Implementation of the proposed algorithm

The proposed algorithm has been developed using C++ programming language. In order to improve the understand ability of the algorithm, the source code is presented below.

```
void clipMY3D(double x[],double y[],double z[],double minx,double miny,double minz,double maxx,double maxy,double maxz)
{
int i;
double a,b; // Two constants depending on the line

if((x[0]!=x[1]) && (y[0]!=y[1]) && (z[0]!=z[1])) // Line is not parallel to xy, yz and zx planes
{
a=(x[1]-x[0])/(y[1]-y[0]);
b=(y[1]-y[0])/(z[1]-z[0]);

for(i=0; i<2; i++)
{
        if(x[i] < minx)
        {
                y[i] = (minx - x[0])/a + y[0];
                z[i] = (minx - x[0])/(a*b) + z[0];
                x[i] = minx;
        }
        else if(x[i] > maxx)
        {
                y[i] = (maxx - x[0])/a + y[0];
                z[i] = (maxx - x[0])/(a*b) + z[0];
                x[i] = maxx;
        }
        if(y[i] < miny)
        {
                x[i] = a*(miny - y[0]) + x[0];
                z[i] = (miny - y[0])/b + z[0];
                y[i] = miny;
        }
        else if(y[i] > maxy)
        {
                x[i] = a*(maxy - y[0]) + x[0];
                z[i] = (maxy - y[0])/b + z[0];
                y[i] = maxy;
        }
        if(z[i] < minz)
        {
                x[i] = a*b*(minz - z[0]) + x[0];
                y[i] = b*(minz - z[0]) + y[0];
                z[i] = minz;
        }
        else if(z[i] > maxz)
        {
                x[i] = a*b*(maxz - z[0]) + x[0];
                y[i] = b*(maxz - z[0]) + y[0];
                z[i] = maxz;
        }
}

if((x[0] - x[1] < 1) && (x[1] - x[0] < 1)) // Initial line is completely
outside
{
        // Do nothing
}
else
{
        cout<< x[0] << "," << y[0] << "," << z[0] << "     ;
        " << x[1] << "," << y[1] << "," << z[1] << endl;
}

}
else if(z[0] == z[1]) // Line is parallel to xy plane
{
        clipMY1(x, y, z, minx, miny, maxx, maxy);
}
else if(x[0] != x[1]) // Line is parallel to xz plane
{
        clipMY2(x, y, z, minz, minx, maxz, maxx);
}
else if(y[0] != y[1]) //Line is parallel to yz plane
{
        clipMY3(x, y, z, miny, minz, maxy, maxz);
}
else // Line is parallel to z-axis
{
        // initial line is completely outside
        if((x[0] <= minx) || (x[0] >= maxx) || (y[0] <= miny) ||
(y[0] >= maxy))
        {
                // do nothing
        }
        else
        {
                for(i=0; i<2; i++)
                {
                        if(z[i] < minz)
                        {
                                z[i] = minz;
                        }
                        else if(z[i] > maxz)
                        {
                                z[i] = maxz;
                        }
                }
                // initial line is completely outside
                if((z[0] - z[1] < 1) && (z[1] - z[0] < 1))
                {
                        // do nothing
                }
                // draw the clipped line
                else
                {
                        Cout << x[0] << "," << y[0] << "," << z[0] << "     ;
                        " << x[1] << "," << y[1] << "," << z[1] << endl;
                }
        }
}

}
```

NOTE: clipMY1, clipMY2, clipMY3 are functions that have been designed to clip lines in 2D space[7]. Once the line segment is parallel to one of the principle planes it can be considered as a 2D clipping problem.

4) Analysis of the algorithm

This section analyzes the algorithm for some possible situations.

Case 1: Line is completely inside as shown in Figure 7

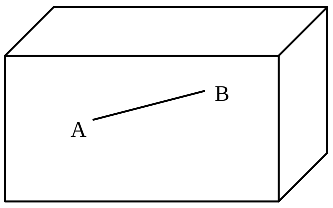


Fig.7.Line is completely inside

x[A] != x[B]$\rightarrow$true
y[A] != y[B]$\rightarrow$true
z[A] != z[B]$\rightarrow$true
Consider point A,
x[A] < minx$\rightarrow$false
x[A] > maxx$\rightarrow$ false
y[A] < miny$\rightarrow$false
y[A] > maxy$\rightarrow$ false
z[A] < minz$\rightarrow$ false
z[A] > maxz$\rightarrow$false
Therefore, the initial position of A is not changed.
Consider point B,
x[B] < minx$\rightarrow$false
x[B] > maxx$\rightarrow$ false
y[B] < miny$\rightarrow$false
y[B] > maxy$\rightarrow$ false
z[B] < minz$\rightarrow$false
z[B] > maxz$\rightarrow$false
Therefore, the initial position of B is not changed.
(x[A] - x[B] < 1) && (x[B] - x[A]) $\rightarrow$ false
Therefore, the line with the end points A and B is drawn.

Case 2: Line is completely outside as depicted in Figure 8

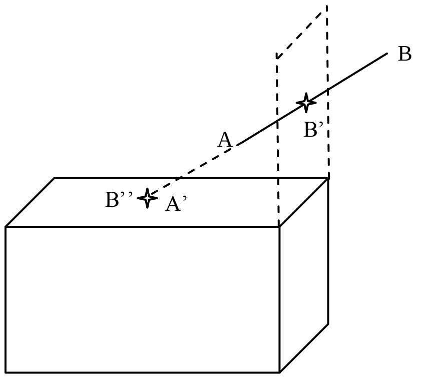


Fig.8. Line is completely outside

x[A] != x[B]→true
y[A] != y[B]→true
z[A] != z[B]→true
Consider point A,
x[A] < minx→false
x[A] > maxx→false
y[A] < miny→true
So, A→A'
z[A'] < minz→false
z[A'] > maxz→false
Consider point B,
x[B] < minx→ false
x[B] > maxx→true
So, B→B'
y[B'] < miny→true
So, B'→B''
z[B''] < minz→false
z[B''] > maxz→false
(x[A'] - x[B''] <1) && (x[B''] - x[A']) → true
Therefore, the line is ignored.

Case 3: Line intersects the clipping window as shown in Figure 9

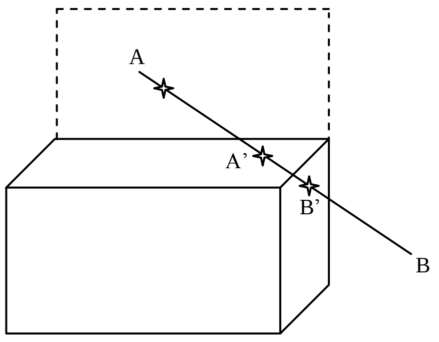


Fig.9. Line is intersecting the boundaries

x[A] != x[B]→true
y[A] != y[B]→true
z[A] != z[B]→true
Consider point A,
x[A] < minx→false
x[A] > maxx→ false
y[A] < miny→true
So, A→A'
z[A'] < minz→false
z[A'] > maxz→false
Consider point B,
x[B] < minx→ false
x[B] > maxx→true
So, B→B'
y[B'] < miny→false
y[B'] > maxy→false
z[B'] < minz→ false
z[B'] > maxz→ false
(x[A'] - x[B'] < 1) && (x[B'] - x[A']) → false
Therefore, the line with the end points A' and B' is drawn.

Case 4: Line partially inside the clipping window as depicted in Figure 10

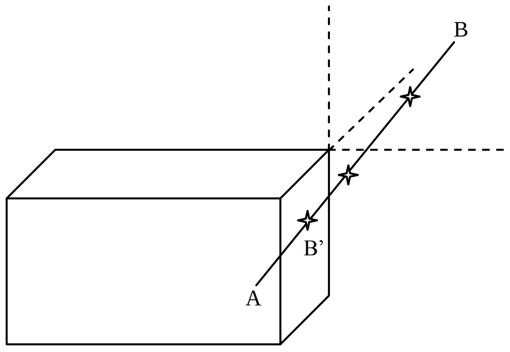


Fig.10. Line is partially inside the clipping window

x[A] != x[B]→true
y[A] != y[B]→true
z[A] != z[B]→true
Consider point A,
x[A] < minx→false
x[A] > maxx→ false
y[A] < miny→false
y[A] > maxy→ false
y[A] < minz→ false
y[A] < maxz→ false
Therefore, the initial position of A is not changed.
Consider point B,
x[B] < minx→ false
x[B] > maxx→true
Therefore, B→B'
y[B'] < miny→false
y[B'] < maxy→false
z[B'] < minz→ false
z[B'] < maxz→ false
(x[A] - x[B'] < 1) && (x[B'] - x[A]) → false
Therefore, the line with the end points A and B' is drawn.

III. RESULTS AND DISCUSSION

The proposed algorithm was tested for all the possible test cases of line segments. The test results indicated that it performs well in all possible situations. In order to validate the algorithm, it was compared against the Cohen-Sutherland 3D space line clipping algorithm. The following hardware and software were used for testing.

Computer: Intel(R) Pentium(R) Dual CPU; E2180 @ 2.00 GHz; 2.00 GHz, 0.98 GB RAM IDE Details: Turbo C++; Version 3.0; Copyright(c) 1990, 1992 by Borland International, Inc. Method [2]:
The clipping volume (cuboid) with values minx = miny = minz = 100 and maxx = maxy = maxz = 300 was used for clipping. Random points were generated in the range 0 - 399 by using the randomize() function. These random points were considered as end points to generate random lines. Number of clock cycles taken by each algorithm to clip 1000000 random lines were counted using the clock() function. The results are shown in Table 1.

TABLE I
NUMBER OF CLOCK CYCLES COMPARISON

Step	Cohen-Sutherland algorithm	The proposed algorithm
1	3596	3403
2	3497	3271
3	3946	3431
4	3867	3980
5	3489	3097
6	4014	3790
7	3906	3767
8	3638	3737
9	4018	3914
10	3839	3684

The results prove that the new 3D space line clipping algorithm is faster in 8 steps out of 10 steps than the Cohen-Sutherland algorithm. Performance in our algorithm is poor when the line segment is completely outside because a number of intersection calculations are needed to remove such lines which are completely outside of the clipping window. Therefore, in a situation where many randomly generated lines are completely outside of the clipping window, the performance of the newly proposed algorithm is slightly lower than Cohen-Sutherland algorithm.

IV. CONCLUSIONS

According to the test results, the proposed 3D space line clipping algorithm is faster than the 3D Cohen–Sutherland algorithm. Therefore, this algorithm can be successfully used in 3D applications where line clipping involved. This algorithm can further be extended to clip lines within a polyhedron volume.

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.

S. R. Kodituwakku is an associate professor at the Department of Statistics and Computer Science, University of Peradeniya, Sri Lanka. His research interests include database systems, distributed computing, Role based Access Control systems, and software engineering.

K. R. Wijeweera is an undergraduate, following a computer science special degree in University of Peradeniya, Sri Lanka. His research interests include computer graphics, image processing, computer vision, and artificial intelligence.

M. A. Pathum Chamikara is working as a research assistant at the Post Graduate Institute of Science (PGIS), University of Peradeniya, Sri Lanka. He received his BSc (Special) degree in Computer Science, University of Peradeniya, Sri Lanka (2010). His research interests include Crime analysis, GIS (Geographic Information Systems), image processing, computer vision and artificial intelligence.