COMPUTER GRAPHICS ALGORITHMS

A REPORT BY

NJERU OLIVER NJIRU

ADMISSION NUMBER – 663565

CONTENTS

LINE DRAWING ALGORITHMS

According to (Wikipedia, 2022), a line drawing algorithm approximates a line segment on discrete graphical media, such as pixel-based displays and printers. A list of line drawing algorithms are:

- Naïve algorithm
- Digital differential analyzer (graphics algorithm) that is similar to naïve line-drawing algorithm but with minor variations.
- Bresenham's line algorithm is optimized to use only additions (i.e. no divisions or multiplications) and also avoids floating-point computations.
- Xiaolin Wu's line algorithm can perform spatial anti-aliasing and appears to be "ropey" from brightness varying along the length of the line which may be greatly reduced by pre-compensating the pixel values for the target display's gamma curve.
- Gupta-Sproull algorithm

Naïve line-drawing algorithm

It has the equation:

dx = x2-x1

dy = y2-y1

for x from x1 to x2 do

    y = y1 + dy x (x-x1) / dx

    plot(x, y)

This algorithm will work okay given dx>=dy(i.e., the slope is less than or equal to 1), but if dx<dy(i.e., the slope is greater than 1), the line becomes quite sparse with many gaps, and in the limiting case of dx=0, a division by zero exception will occur.

Disadvantages

It is inefficient due to use of floating point calculations

Bresenham's Line Generation Algorithm

We will explain this with examples drawn from (GeeksforGeeks, 2022). Given the coordinates for two points, A(x1, y1) and B(x2, y2), find all the intermediate points required for drawing line AB on the computer screen of pixels noting that every pixel has integer coordinates.

Example

Input: A(0,0), B(4,4)

Output: (0,0), (1,1), (2,2), (3,3), (4,4)

Input: A(0,0), B(4,2)

Output: (0,0), (1,0), (2,1), (3,1), (4,2)

Constraints

Lines are drawn from left to right

x1<x2 and y1<y2

Slope of the line is between 0 and 1, meaning we draw the line from the lower left to upper right.

To solve this, we can implement a naïve approach which works but is slow. The sample code from this naïve approach is:

```
void naiveDrawLine(x1, x2, y1, y2){

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

    for(x=x1;x<=x2;x++){

        //Assuming that the round function finds closest integer to a given float

        y=round(mx+c);

        print(x,y);

    }
```

}

The idea of Bresenham's algorithm is to avoid floating point multiplication and addition to compute mx+c, and then compute the round value of (mx+c) in every step and we also move across the x-axis in unit intervals. We always increase x by 1, and we choose about next y, whether we need to go to y+1 or remain on y. In other words, from any position (Xk, Yk) we need to choose between (Xk + 1, Yk) and (Xk + 1, Yk + 1). We would like to pick the y value (among Yk + 1 and Yk) corresponding to a point that is closer to the original line.We need a decision parameter to decide whether to pick Yk + 1 or Yk as the next point. The idea is to keep track of slope error from the previous increment to y. If the slope error becomes greater than 0.5, we know that the line has moved upwards one pixel and that we must increment our y coordinate and readjust the error to represent the distance from the top of the new pixel – which is done by subtracting one from the error.

The code is:

```
// Modifying the naive way to use a parameter to decide next y.

void withDecisionParameter(x1, x2, y1, y2)

{

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

        slope_error = [Some Initial Value];

        for (x = x1, y = y1; x = 0.5) {

        y++;

        slope_error -= 1.0;

        }

}
```

How to avoid floating point arithmetic

The above algorithm still includes floating point arithmetic and to avoid floating point arithmetic, consider

the value below value m.

$m = (y2 – y1)/(x2 – x1)$

We multiply both sides by $(x2 – x1)$

We also change slope_error to slope_error * $(x2 – x1)$. To avoid comparison with 0.5, we further change

it to slope_error * $(x2 – x1) * 2$.

Also, it is generally preferred to compare with 0 than 1.

The C++ code is:

```
// Modifying the above algorithm to avoid floating point arithmetic and use comparison with 0.

void bresenham(x1, x2, y1, y2)
```

```
{

        m_new = 2 * (y2 - y1) slope_error_new =

        [Some Initial Value] for (x = x1, y = y1; x = 0)

        {

                y++;

                slope_error_new -= 2 * (x2 - x1);

        }

}
```

The initial value of slope_error_new is 2*(y2 – y1) – (x2 – x1). The implementation of the above algorithm is shown below and only works if the slope of the line is less than 1:

```
// Assuming:

// 1) Line is drawn from left to right.

// 2) x1 < x2 and y1 < y2
```

```
// 3) Slope of the line is between 0 and 1.

//    We draw a line from lower left to upper

//    right.

#include <bits/stdc++.h>

using namespace std;




// function for line generation

void bresenham(int x1, int y1, int x2, int y2)

{

   int m_new = 2 * (y2 - y1);

   int slope_error_new = m_new - (x2 - x1);

   for (int x = x1, y = y1; x <= x2; x++) {

      cout << "(" << x << "," << y << ")\n";
```

```
        // Add slope to increment angle formed

        slope_error_new += m_new;




        // Slope error reached limit, time to

        // increment y and update slope error.

        if (slope_error_new >= 0) {

            y++;

            slope_error_new -= 2 * (x2 - x1);

        }

    }

}
```

```
// driver code

int main()

{

   int x1 = 3, y1 = 2, x2 = 15, y2 = 5;



   // Function call

   bresenham(x1, y1, x2, y2);

   return 0;

}
```

The Output will be:

```
(3,2)

(4,3)

(5,3)
```

(6,3)

(7,3)

(8,4)

(9,4)

(10,4)

(11,4)

(12,5)

(13,5)

(14,5)

(15,5)

The time complexity is O(x2-x1)

The Space complexity is O(1)

To implement any kind of slope:

```
#include <bits/stdc++.h>

using namespace std;



void plotPixel(int x1, int y1, int x2, int y2, int dx,

        int dy, int decide)

{

   // pk is initial decision making parameter

   // Note:x1&y1,x2&y2, dx&dy values are interchanged

   // and passed in plotPixel function so

   // it can handle both cases when m>1 & m<1

   int pk = 2 * dy - dx;

   for (int i = 0; i <= dx; i++) {
```

```
cout << x1 << "," << y1 << endl;

// checking either to decrement or increment the

// value if we have to plot from (0,100) to (100,0)

x1 < x2 ? x1++ : x1--;

if (pk < 0) {

   // decision value will decide to plot

   // either  x1 or y1 in x's position

   if (decide == 0) {

      // putpixel(x1, y1, RED);

      pk = pk + 2 * dy;

   }

   else {

      //(y1,x1) is passed in xt
```

```
        // putpixel(y1, x1, YELLOW);

        pk = pk + 2 * dy;

    }

}

else {

    y1 < y2 ? y1++ : y1--;

    if (decide == 0) {

        // putpixel(x1, y1, RED);

    }

    else {

        //  putpixel(y1, x1, YELLOW);

    }
```

```
            pk = pk + 2 * dy - 2 * dx;

        }

    }

}



// Driver code

int main()

{

    int x1 = 100, y1 = 110, x2 = 125, y2 = 120, dx, dy, pk;



    dx = abs(x2 - x1);

    dy = abs(y2 - y1);
```

```
// If slope is less than one

if (dx > dy) {

   // passing argument as 0 to plot(x,y)

   plotPixel(x1, y1, x2, y2, dx, dy, 0);

}

// if slope is greater than or equal to 1

else {

   // passing argument as 1 to plot (y,x)

   plotPixel(y1, x1, y2, x2, dy, dx, 1);

}
```

```
}
```

with the output being

```
100,110

101,110

102,111

103,111

104,112

105,112

106,112

107,113

108,113

109,114

110,114
```

111,114

112,115

113,115

114,116

115,116

116,116

117,117

118,117

119,118

120,118

121,118

122,119

123,119

124,120

125,120

According to (JavatPoint, n.d.), the advantages of Bresenham's Line Algorithm are:

- It involves only integer arithmetic, so it is simple.
- It avoids the generation of duplicate points.
- It can be implemented using hardware because it does not use multiplication and division.
- It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

Bresenham's Line Algorithm's disadvantages according to (JavatPoint, n.d.) are:

- This algorithm is meant for basic line drawing only Initializing is not a part of Bresenham's line algorithm, so to draw smooth lines, you should want to look into a different algorithm.

DIGITAL DIFFERENTIAL ANALYZER

A digital differential analyzer algorithm is the simple line generation algorithm with 5 steps according to

(tutorialspoint, n.d.) shown below:

Step 1: Get the input of two end points (X0,Y0) and (X1,Y1).

Step 2: Calculate the difference between two end points.

```
dx = X1 - X0

dy = Y1 - Y0
```

Step 3: Based on the calculated difference in step-2, you need to identify the number of steps to put pixel.

If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))

Steps = absolute(dx);
```

else

Steps = absolute(dy);

Step 4: Calculate the increment in x coordinate and y coordinate.

Xincrement = dx / (float) steps;

Yincrement = dy / (float) steps;

Step 5: Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

for(int v=0; v < Steps; v++)

{

x = x + Xincrement;

y = y + Yincrement;

```
        putpixel(Round(x), Round(y));

        }
```

I will now end with the differences between the Digital Differential Algorithm and Bresenham's Line Algorithm in the table below according to (JavatPoint, n.d.).

DDA	Bresenham's Algorithm
Uses floating point, i.e., Real Arithmetic	Uses fixed point, i.e., Integer Arithmetic
Uses multiplication and division in its operation	Uses only subtraction and adding in its operation
Is slower than Bresenham's Line Algorithm in drawing a line because it uses real arithmetic (Floating point operation)	Is faster than DDA because it involves only addition & subtraction in its calculation and uses only integer arithmetic
Is not accurate and efficient as Bresenham's Line Algorithm	Is more accurate and Efficient than DDA
Can draw a circle and curves but are not accurate as Bresenham's Line Algorithm	Can draw a circle and curves with more accuracy than DDA

LINE CLIPPING ALGORITHM

According to (Encyclopedia, 2022), line clipping is the process of removing lines or portions of lines outside an area of interest, the parts that exceed say a view port or view volume. There are three algorithms for line clipping, notably being Cohen-Sutherland, Midpoint Subdivision Line Clipping Algorithm and Liang-

Barsky according to (javatpoint, n.d.). A line clipping method consists of various parts, tests conducted on a given line segment to find out whether it lies outside the view port, intersection calculations carried out with one or more clipping boundaries determining which portion of the line is inside or outside the viewport.

Cohen Sutherland Line Clipping Algorithm

This algorithm detects whether lines lie inside or outside the screen by placing them in the following categories:

- Visible – for when a line's endpoints line within the window
- Not Visible – for when a line lies outside the window.
    - Let A (x1,y2) and B (x2,y2) are endpoints of line.
    - xmin,xmax are coordinates of the window.
    - ymin,ymax are also coordinates of the window.

- x1>xmax
- x2>xmax
- y1>ymax
- y2>ymax
- x1<xmin
- x2<xmin
- y1<ymin
- y2<ymin

- Clipping Case – for when the line is neither visible or invisible

Advantages of Cohen Sutherland Line Clipping

- It calculates end-points very quickly and rejects and accepts lines quickly.
- It can clip pictures much large than screen size.

Algorithm of Cohen Sutherland Line Clipping:

Step1 - Calculate positions of both endpoints of the line

Step2 - Perform OR operation on both of these end-points

Step3 - If the OR operation gives 0000

Then

line is considered to be visible

else

Perform AND operation on both endpoints

If And ≠ 0000

then the line is invisible

else

And=0000

Line is considered the clipped case.

Step4 - If a line is clipped case, find an intersection with boundaries of the window

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

(a) If bit 1 is "1" line intersects with left boundary of rectangle window

y3=y1+m(x-X1)

where X = Xwmin

where Xwminis the minimum value of X co-ordinate of window

(b) If bit 2 is "1" line intersect with right boundary

y3=y1+m(X-X1)

where X = Xwmax

where X more is maximum value of X co-ordinate of the window

(c) If bit 3 is "1" line intersects with bottom boundary

X3=X1+(y-y1)/m

where y = ywmin

ywmin is the minimum value of Y co-ordinate of the window

(d) If bit 4 is "1" line intersects with the top boundary

X3=X1+(y-y1)/m

where y = ywmax

ywmax is the maximum value of Y co-ordinate of the window

Program to perform Line Clipping using Cohen Sutherland Algorithm:

```
#include <iostream.h>

#include <conio.h>

#include <graphics.h>

#include <dos.h>

class data

{

   int gd, gmode, x, y, xmin,ymin,ymax,xmax;

   int a1,a2;

   float x1, y1,x2,y2,x3,y3;
```

```
    int xs, ys, xe, ye;

    float maxx,maxy;

    public:

        void getdata ();

        void find ();

        void clip ();

        void display (float, float,float,float);

        void checkonof (int);

        void showbit (int);

};

void data :: getdata ()

{

    cout<<"Enter the minimum and maximum coordinate of window (x, y) ";
```

```
        cin >>xmin>>ymin>>xmax>>ymax;

        cout<<"Enter the end points of the line to be clipped";

        cin >>xs>>ys>>xe>>ye;

        display (xs, ys, xe,ye);

}

void data :: display (float, xs, float, ys,float xe, float ye)

{

   int gd=DETECT;

   initgraph (&gd,&gmode, "");

   maxx=getmaxx();

   maxy=getmaxy();

   line (maxx/2,0,maxx/2,maxy);

   line (0, maxy/2,maxx,maxy/2);
```

```
    rectangle (maxx/2+xmin,maxy/2-ymax,maxx/2+xmax,maxy/2-ymin);

    line (maxx/2+xs,maxy/2-ys,maxx/2+xe,maxy/2-ye);

    getch();

}

void data :: find ()

{

    a1=0;

    a2=0;

    if ((ys-ymax)>0)

            a1+=8;

    if ((ymin-ys)>0)

       a1+=4;

    if ((xs-xmax)>0)
```

```
    a1+=2;

      if ((xmin-xs)>0)

    a1+=1;

if ((ye-ymax)>0)

   a2+=8;

     if ((ymin-ye)>0)

        a2+=4;

    if ((xe-xmax)>0)

         a2+=2;

    if ((xmin-xe)>0)

          a2+=1;

   cout<<"\nThe area code of Ist point is ";

          showbit (a1);
```

```
        getch ();

        cout <<"\nThe area code of 2nd point is ";

        showbit (a2);

        getch ();

}

void data :: showbit (int n)

{

       int i,k, and;

       for (i=3;i>=0;i--)

       {

            and =1<<i;

      k = n?

      k ==0?cout<<"0": cout<<"1\"";
```

```
        }

}

void data ::clip()

{

        int j=a1&a2;

        if (j==0)

        {

            cout<<"\nLine is perfect candidate for clipping";

            if (a1==0)

      {

                else

           {

                checkonof(a1);
```

```
      x2=x1;y2=y1;

 }

 if (a2=0)

 {

     x3=xe; y3=ye;

 }

else

{

      checkonof (a2);

      x3=x1; y3=y1;

 }

 xs=x2; ys=y2;xe=x3;ye=y3;

 cout << endl;
```

```
        display (xs,ys,xe,ye);

        cout<<"Line after clipping";

        getch ()

      }

    else if ((a1==0) && (a2=0))

    {

        cout <<"\n Line is in the visible region";

        getch ();

    }

}

void data :: checkonof (int i)

{

    int j, k,l,m;
```

```
  1=i&1;

  x1=0;y1=0;

   if (1==1)

  {

        x1=xmin;

        y1=ys+ ((x1-xs)/ (xe-xs))*(ye-ys);

  }

  j=i&8;

if (j>0)

{

        y1=ymax;

  x1=xs+(y1-ys)/(ye-ys))*(xe-xs);

 }
```

```
k=i & 4;

if (k==1)

{

      y1=ymin;

      x1=xs+((y1-ys)/(ye-ys))*(xe-xs);

}

m= i&2;

 if (m==1)

 {

       x1=xmax;

       y1=ys+ ((x1-xs)/ (xe-xs))*(ye-ys);

  }

  main ()
```

```
{

    data s;

    clrscr();

    s.getdata();

    s.find();

    getch();

    closegraph ();

    return ();

}
```

ANTI-ALIASING

Antialiasing is a technique used in computer graphics to remove the aliasing effect which is the appearance of jagged edges or jaggies in a rasterized image which is an image rendered using pixels

according to (GeeksforGeeks, Computer Graphics | Antialiasing - GeeksforGeeks, 2022). Cause of antialiasing is Undersampling that results in loss of information of the picture and occurs when sampling is done at a frequency lower than Nyquist sampling frequency and to avoid this loss, we need to have our sampling frequency at least twice that of highest frequency occurring in the object.

Nyquist Sampling Frequency: $f_s=2*f_{max}$

This can also be stated as that our sampling interval should be no larger than half the cycle interval. This maximum required the sampling interval is called Nyquist sampling interval Δxs:

Δxs = Δxcycle/2

Where Δxcycle=1/fmax

Methods of Antialiasing

- Using high-resolution display

- When you use a high-resolution display, the jaggies become so small that they become indistinguishable by the human eye thereby making the jagged edges get blurred out and edges appear smooth.
- A practical application is the retina displays in Apple devices where OLED displays have high pixel density due to which jaggies formed are so small that they are blurred and indistinguishable by our eyes.

- Post filtering (Supersampling)
  - Here, we are increasing the sampling resolution by treating the screen as if it's made of a much more fine grid, due to which the effective pixel size is reduced but the screen resolution remains the same. Now, intensity from each subpixel is calculated and average intensity of the pixel is found from the average of intensities of subpixels, thus we do sampling at higher resolution and display the image at lower resolution or resolution of

the screen, hence this technique is called supersampling. This method is also known as post filtration as this procedure is done after generating the rasterized image.

- A practical application is in gaming, SSAA (Supersample Antialiasing) or FSAA (full-scene antialiasing) is used to create best image quality. It is often called the pure AA and hence is very slow and has a very high computational cost. This technique was widely used in early days when better AA techniques were not available. Different modes of SSAA available are: 2X, 4X, 8X, etc. denoting that sampling is done x times (more than) the current resolution.

- A better style of AA is MSAA (multisampling Antialiasing) which is a faster and approximate style of supersampling AA.It has lesser computational cost. Better and sophisticated supersampling techniques are developed by graphics card companies like CSAA by NVIDIA and CFAA by AMD.

- Pre-filtering (Area Sampling)

    - In area sampling, pixel intensities are calculated proportional to areas of overlap of each pixel with objects to be displayed. Here pixel color is computed based on the overlap of scene's objects with a pixel area.For example: Suppose, a line passes through two pixels. The pixel covering bigger portion(90%) of line displays 90% intensity while less area(10%) covering pixel displays 10-15% intensity. If pixel area overlaps with different color areas, then the final pixel color is taken as an average of colors of the overlap area. This method is also known as pre-filtering as this procedure is done BEFORE generating the rasterized image. It's done using some graphics primitive algorithms.

- Pixel phasing

- It's a technique to remove aliasing where pixel positions are shifted to nearly approximate positions near object geometry. Some systems allow the size of individual pixels to be adjusted for distributing intensities which is helpful in pixel phasing.

With all these explained, I rest my case on the report of computer graphics algorithms.

REFERENCES

Encyclopedia, W. T. (2022, July 21). *Line Clipping - Wikipedia*. Retrieved from Wikipedia The Free Encyclopedia: https://en.wikipedia.org/wiki/Line_clipping#:~:text=There%20are%20two%20common%20algorithms,the%20view%20area%20or%20volume.

GeeksforGeeks. (2022, November 15). *Bresenham's Line Generation Algorithm - GeeksforGeeks*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/

GeeksforGeeks. (2022, July 03). *Computer Graphics | Antialiasing - GeeksforGeeks*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/computer-graphics-antialiasing/

javatpoint. (n.d.). *Computer Graphics | Line Clipping - javatpoint*. Retrieved from javatpoint: https://www.javatpoint.com/computer-graphics-line-clipping

JavatPoint. (n.d.). *Computer Graphics Bresenham's Line Algorithm - javapoint*. Retrieved from JavatPoint: https://www.javatpoint.com/computer-graphics-bresenhams-line-algorithm

tutorialspoint. (n.d.). *Line Generation Algorithm*. Retrieved from tutorialspoint simply easy learning: https://www.tutorialspoint.com/computer_graphics/line_generation_algorithm.htm#:~:text=Step%201%20%E2%88%92%20Get%20the%20input,difference%20between%20two%20end%20points.&text=Step%203%20%E2%88%92%20Based%20on%20the,coordinate%3B%20otherwise%20in%20y%20co

Wikipedia. (2022, October 18). *Line drawing algorithm*. Retrieved from Wikipedia The Free Encyclopedia:

https://en.wikipedia.org/wiki/Line_drawing_algorithm