Laboratory work 6

1 Objectives

This laboratory highlights the Bresenham algorithms used for rendering some of the graphical primitives on a computer display. This paper begins by presenting some generic information about the algorithms and then exemplifies them for line and circle rasterization.

2 Theoretical background

The Bresenham algorithm for drawing lines onto a bi-dimensional space (like a computer display) is a fundamental method used in computer graphics discipline. The algorithm's efficiency makes it one of the most required methods for drawing continuous lines, circles or other graphical primitives. This process is called rasterization.

Each line, circle or other graphical primitives will be plotted pixel by pixel. Each pixel is described by a fixed (x, y) position in the bi-dimensional XOY space. The algorithm approximates the real line by computing each one of the line's pixel's position. Since the pixels are the smallest addressable screen elements in a display device, the algorithm approximation is good enough to "trick" the human eyes and to get the illusion of a real line. Figure 1a and Figure 1b shows the real line and the approximated line drawn over the pixel grid.

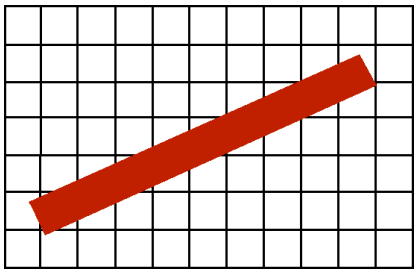

Figure 1a. Real line

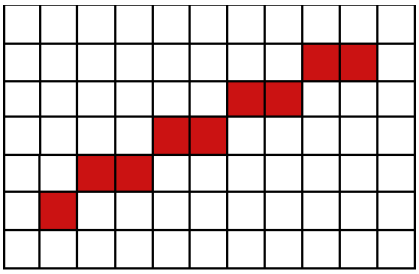

Figure 1b. Approximated line

Before moving on, it is worth to mention that both the line (1) and circle (2) can be mathematically described using the following equations:

$$y = m \cdot x + c \quad (1)$$

$$\text{with } m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$(x-a)^2 + (y-b)^2 = R^2 \quad (2)$$

where:

- m is the line's slope;
- (x_1, x_2), (y_1, y_2) are the two endpoints of the line segment;
- (a, b) represents the coordinates of the circle's center;

2.1 Bresenham's algorithm for line

For simplicity we will take into account a line segment with the slope from 0 to 1. Suppose the two endpoints of the line are $A(x_1, y_1)$ and $B(x_2, y_2)$. At this point we have to choose an initial point to start the algorithm. We can choose this point ($P(x_i, y_i)$) to be either A or B. Based on the starting position, we have eight possible choices to draw the next pixel of the line. This is due to the fact that each pixel is surrounded by 8 adjacent pixels.

Our example will consider only the case where we have two choice alternatives for the next pixel position (in other words this example will work only for the first octant of the trigonometric circle). For example, for the current point P we have the following drawing possibilities: $T(x_{i+1}, y_i)$ or $S(x_{i+1}, y_{i+1})$.

The decision criterion (Figure 2) for Bresenham's algorithm is based on the distance between the current point, P, and the real line segment. So the closest point (*T* or *S*) to the line segment will be chosen.

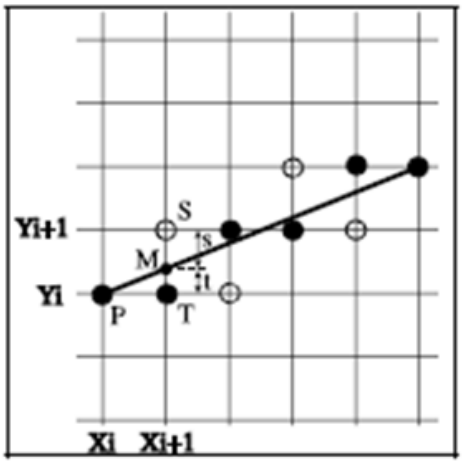

Figure 2. Decision criterion to choose the next line pixel that will plot on the screen display

The following paragraphs will describe the general steps of the Bresenham's algorithm in natural language rather than a programmatically one, because it is easier to understand.

a. Let us assume that we have to draw a line segment with the endpoints represented by $A(x_1, y_1)$ and $B(x_2, y_2)$. We translate the line segment with $(-x_1, -y_1)$ to place it on the XOY system origin.

b. Let $dx = x_2 - x_1, dy = y_2 - y_1$. The line that needs to be drawn can be described as $y = \frac{dx}{dy} x$.

c. In this step we intend to compute the next line pixel, using the criterion mentioned above. From Figure 2, we can deduce that the closest point to the real line value is $T(x_{i+1}, y_i)$. Based on this observation we could say that $M(x_{i+1}, x_{i+1} \cdot \frac{dy}{dx})$ (3).

In other words $\begin{cases} t = y_m - y_i \\ s = y_{i+1} - y_m \end{cases} \Rightarrow t - s = 2 \cdot y_m - 2 \cdot y_i - 1$ (4).

Taking into account (3) and (4) we obtain $dx \cdot (t-s) = 2 \cdot dy \cdot x_{i+1} - 2 \cdot dx \cdot y_i - dx$. If the x coordinates of the line segment endpoints are in $x_1 < x_2$ relationship, then the sign of $t-s$ will coincide with the sign of $dx \cdot (t-s)$.

d. We can obtain the following recurrence relationship: $d_{i+1} = d_i + 2 \cdot dy - 2 \cdot dx \cdot (y_i - y_{i+1})$ if we consider that $dx \cdot (t-s) = d_{i+1}$.

The initial value of $d_i = 2 \cdot dy - dx$ is obtained for $x_0 = 0$ and $y_0 = 0$. We can conclude that:

- If $d_i \geq 0 \Rightarrow (t-s) \geq 0$ and the closest point to the real line segment is $S(x_{i+1}, y_{i+1})$. Based on this observations we find that d_i value can be computed as $d_{i+1} = d_i + 2(dy - dx)$.
- If $d_i < 0 \Rightarrow (t-s) < 0$ and the closest point to the real line segment is $T(x_{i+1}, y_i)$. Then the recurrence formula to compute d_i is $d_{i+1} = d_i + 2dy$.

The pseudo code for the Bresenham algorithm is described in the following paragraph, and it is based on the mathematical observations mentioned above.

```
//draw a line in the first octant
Algorithm Bresenham_line()
{
   //Initialize increments
   dx = abs(x2-x1);
   dy = abs(y2-y1);
   d = 2*dy-dx;
   inc1 = 2*dy;
   inc2 = 2*(dy-dx);

   //Set the starting point, end point and current point
   startX = x1;
   startY = y1;
   endX   = x2;
   endY   = y2;
   currentX = x1;
   currentY = y1;


   //Draw each pixel of the line
   while (currentX < endX) {
```

```
   //Draw the current pixel
   DrawPixel(currentX, currentY);
   increment currentX;

      if (d < 0) then {
         increment d using inc1;
      }
      else {
         increment currentY;
         increment d using inc2;
      }
   }
}
```

2.2 Bresenham's algorithm for circle

Let's say we want to scan-convert a circle centered at $(0,0)$ with an integer radius R (Figure 3). We'll see that the ideas we previously used for line scan-conversion can also be used for this task. First of all, notice that the interior of the circle is characterized by the inequality $D(x,y): x^2+y^2-R^2<0$.

We'll use $D(x,y)$ to derive our decision variable.

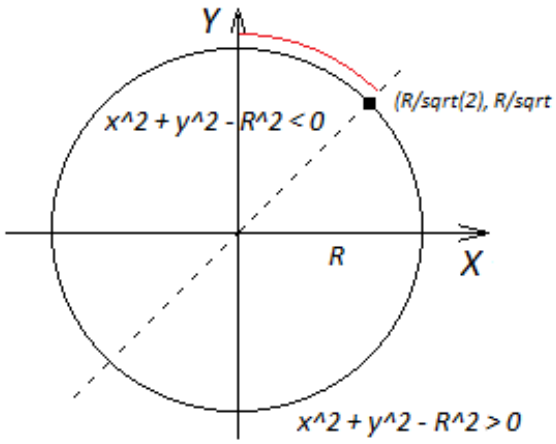

Figure 3. A circle and the description of its interior and exterior as two quadratic inequalities

Following the same approach as for the line segment representation, we'll first present the Bresenham's algorithm for the circle in natural language, describing for each step the general ideas behind it.

a. First, let's think how to plot pixels close to the 1/8 of the circle marked red in Figure 3. The range of the x coordinate for such pixels is from 0 to $R\sqrt{2}$. We'll go over vertical scanlines through the centers of the pixels and, for each such scanline, compute the pixel on that line

which is the closest to the scanline-circle intersection point (black dots in Figure 4). All such pixels will be plotted by our procedure.

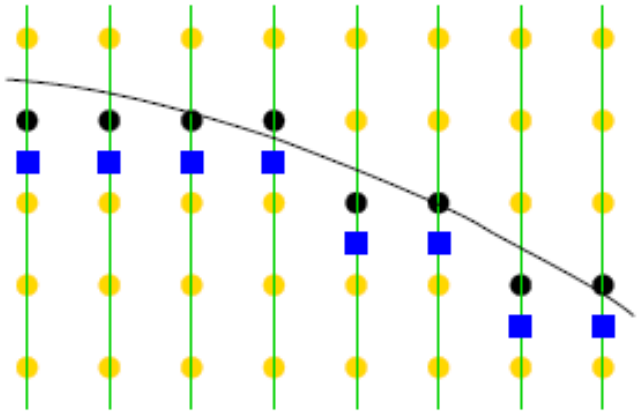

Figure 4. A circle and the description of its interior and exterior as two quadratic inequalities

b. Notice that each time we move to the next scanline, the y-coordinate of the plotted point either stays the same or decreases by 1 (the slope of the circle is between -1 and 0). To decide what needs to be done, we'll use the decision variable, which will be the value of $D(x, y)$ evaluated at the blue square (e.g. the midpoint between the plotted pixel and the pixel immediately below).

c. The first pixel plotted is $(0, R)$ and therefore the initial value of the decision variable should be

$$D(0, R-0.5)=(R-0.5)^2-R^2=0.25-R$$

The y variable, holding the second coordinates of the plotted pixels, will be initialized to *R*. Let's think now at what happens after a point (x, y) is plotted. First, we'll pretend that we need to move the plotted point to the right (no change in y) and check if this keeps the decision variable negative (we don't want any blue squares outside the circle). If (x, y) is the last plotted point, the decision variable is $D(x, y-0.5)$. After we move to the right, it becomes $D(x+1, y-0.5)$. Simple arithmetic shows that it increases by $D(x+1, y-0.5)$ - $D(x, y-0.5) = 2x+1$. If this increase makes it positive, we'd better move down by 1 pixel. This puts the blue square at $(x+1, y-1.5)$ and means that we need to increase the decision value by the previous $2x+1$ plus $D(x+1, y-1.5)-D(x+1, y-0.5)=2-2y$.

d. Clearly, to make the decision variable integer, we need to scale it by a factor of 4. Eight-way symmetry is used to go from 1/8-the of the circle to the full circle.

The pseudo code for the Bresenham's algorithm for circle is described below, based on the earlier made observations.

```
Algorithm Bresenham_circle ()
{
    currentY = R;
    d = 1/4 – R;

    //Go only one eighth of a circle
    for currentX = 0 to ceil(R/sqrt(2)) do {

        plot_points(currentX,currentY);
        increment the decision variable by 2*currentX + 1;

        if (d > 0) then
        {
             increment the decision variable by 2 – 2*currentY;
             decrement currentY;
        }
    }
}
```

You can find the **plot_points** function definition below:

```
Function plot_points (x, y)
{
    DrawPixel (x,y);
    DrawPixel (x,–y);
    DrawPixel (–x,y);
    DrawPixel (–x,–y);
    DrawPixel (y,x);
    DrawPixel (–y,x);
    DrawPixel (y,–x);
    DrawPixel (–y,–x);
}
```

3 Geometry rendering using SDL's API

The SDL 2.0 library provides a hardware accelerated rendering API for basic shapes such as rectangles, lines or points.

In order to use hardware accelerated rendering, we have to create a new **SDL_Renderer** for our SDL window.

```
//The window renderer
SDL_Renderer* renderer = NULL;
...
//Create renderer for window
renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
```

To select a rendering color and to clear the window we can use **SDL_SetRenderDrawColor** and **SDL_RenderClear**.

```
//Clear screen
SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF);
SDL_RenderClear(renderer);
```

Whenever a primitive is drawn, its color is the currently selected rendering color. To render points (pixels) we can use **SDL_RenderDrawPoint**.

```
//Draw current point
SDL_RenderDrawPoint(renderer, tmpCurrentX, tmpCurrentY);
```

4 Assignment

- Explore the implementation of Bresenham's algorithm for drawing lines provided in the laboratory's resources folder.
- Extend Bresenham's algorithm implementation for drawing lines to work in all octants of the trigonometric circle. The algorithm presented in this document, as well as the sample code cover only the first octant. Note: Be careful when implementing Bresenham's algorithm. The SDL system of coordinates is different from the Cartesian system of coordinates.
- Implement the function to draw circles using Bresenham's algorithm in the provided sample code.