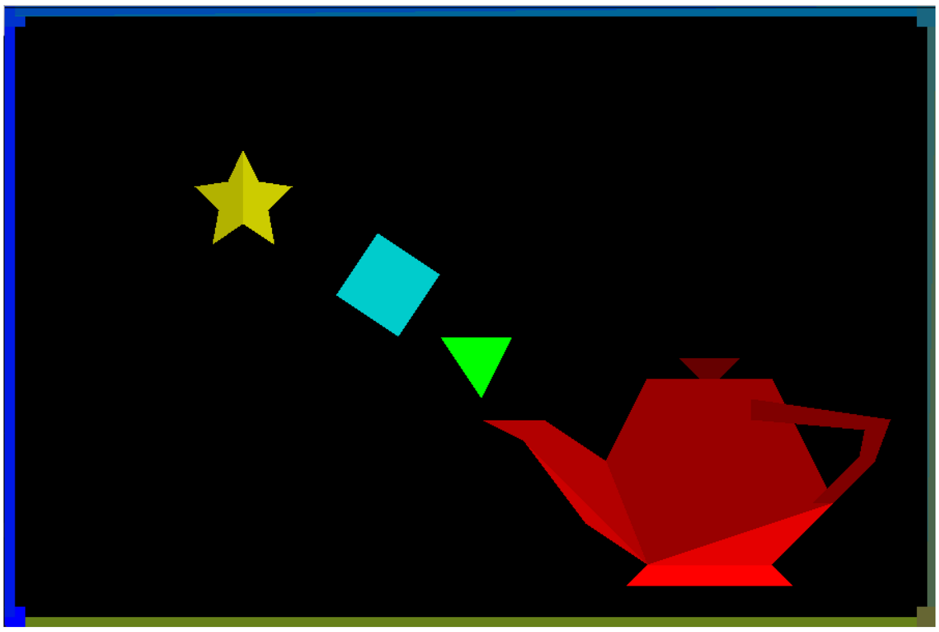
Alberto Scicali Follow
BS/MS Comp Sci. student. Commonly referred to as a Bear. Figuring it all out.
Nov 9, 2016 · 7 min read

Computer Graphics: Scan Line Polygon Fill Algorithm



All hail tea pot

This post serves as my first introduction into the "blogosphere" as well, hopefully, the first of many write-ups about computing and technology in general. So with that out of the way lets get on with it.

Today I will be discussing the Scan Line Polygon Fill (SLPF) algorithm, and then showing my implementation of the algorithm in C++. The purpose of the SLPF algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure.

. . .

Basic Idea:

The basic idea is to collect all of the edges (except horizontal edges) that compose the polygon, fill in the figure scan line by scan line using the edges as starting and stopping points.

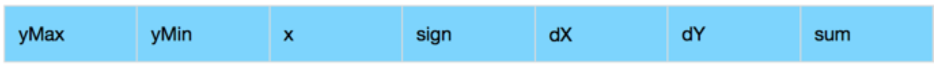
. . .

Main Components:

To successfully fill in a polygon three main components will be used: **Edge Buckets**, an **Edge Table** and an **Active List**. These components will contain an edge's information, hold all of the edges that compose the figure and maintain the current edges being used to fill in the polygon, respectively.

Edge Buckets

The edge bucket is a structure that holds information about the polygon's edges. The edge bucket looks different pending on the implementation of the algorithm, in my case it looks like this:



yMax	yMin	x	sign	dX	dY	sum

Edge Bucket representation

Here is a breakdown of what each member represent in an edge bucket:

- **yMax:** Maximum Y position of the edge
- **yMin:** Minimum Y position of the edge
- **x:** The current x position along the scan line, initially starting at the same point as the **yMin** of the edge
- **sign:** The sign of the edge's slope (either -1 or 1)
- **dX:** The absolute delta x (difference) between the edge's vertex points
- **dY:** The absolute delta y (difference) between the edge's vertex points

- **sum:** Initiated to zero. Used as the scan lines are being filled to x to the next position

Edge Table (ET)

The ET is a list that contains all of the edges that make up the figure.

Important ET notes:

- When creating edges, the vertices of the edge need to be ordered from left to right
- The edges are maintained in increasing yMin order
- The edges are removed from the ET once the Active List is done processing them
- The algorithm is done filling the polygon once all of the edges are removed from the ET

Active List (AL)

The AL contains the edges that are being processed/used to fill the polygon. Every edge in the AL has a pairing buddy edge, because when filling a scan line, pixels are filled starting from one edge until the buddy edge is encountered.

Important AL notes:

- Edges are pushed into the AL from the Edge Table once an edge's yMin is equal to the current scan line being processed
- Edges will always be in the AL in pairs
- Edges in the AL are maintained in increasing x order.
- The AL will be re-sorted after every pass

. . .

Steps:

Now that we have the main components covered, lets go over the SLPF algorithm in more detailed steps.

~General assumptions~

- The user has access to a method that can set the color of individual pixels. You will see that I simply call the setPixel() method whenever I fill in a pixel.
- The vertices of the given shape are listed in order around the circumference of the polygon
- This one is important, otherwise the polygons will not be drawn properly

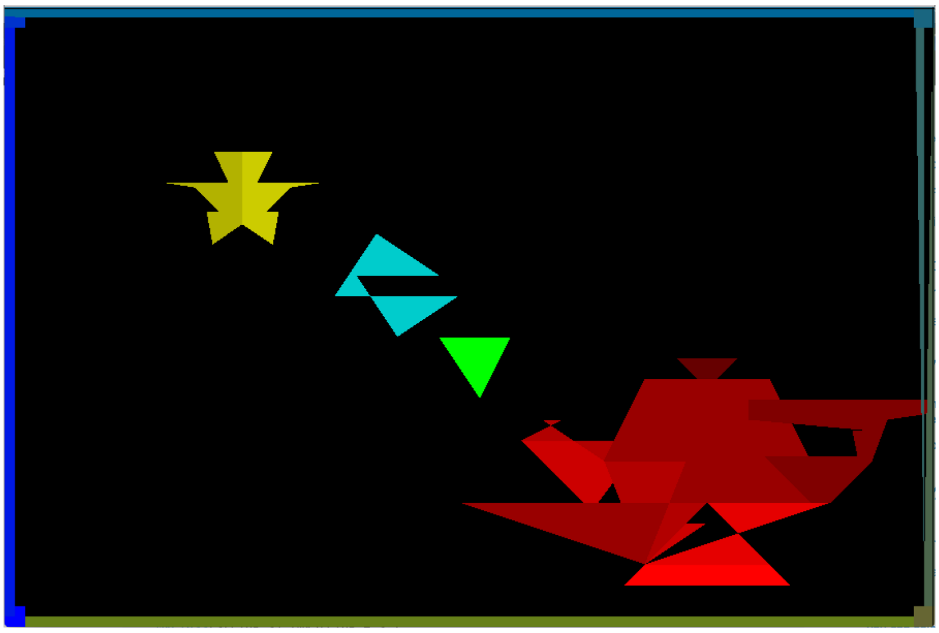
Here are the steps:

```
1. Create ET
    1. Process the vertices list in pairs, start with
[numOfVertices-1] and [0].
    2. For each vertex pair, create an edge bucket
2. Sort ET by yMin
3. Process the ET
    1. Start on the scan line equal to theyMin of the first
edge in the ET
    2. While the ET contains edges
        1. Check if any edges in the AL need to be removes
(when yMax == current scan line)
            1. If an edge is removed from the AL, remove
the associated the Edge Bucket from the Edge Table.
        2. If any edges have a yMin == current scan line,
add them to the AL
        3. Sort the edges in AL by X
        4. Fill in the scan line between pairs of edges in
AL
        5. Increment current scan line
        6. Increment all the X's in the AL edges based on
their slope
            1. If the edge's slope is vertical, the
bucket's x member is NOT incremented.
```

(Why doesn't Medium support nested lists??)

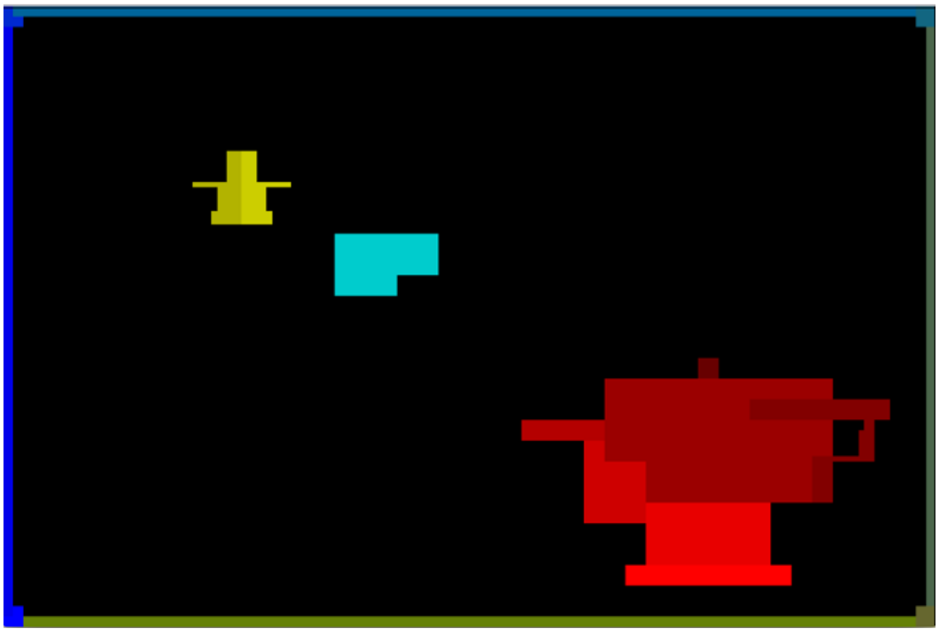
Final Important Notes:

- When creating the edge buckets if the vertices of the edge are not in left to right order you might get the incorrect sign for the bucket. If the edge buckets have the wrong sign this will happen when you try to fill them:


Something is not right here…

To maintain only integers in the Edge Bucket, the slope is not actually calculated but is instead maintained as three distinct members: sign, dX, dY.

- This allows the Edge Buckets' x member to be incremented discreetly without worrying about what happens when encountering a real number in a point
- If the slope is not interpreted correctly, this may happen:



No x increment == pixelated image?

. . .

Implementation:

Now that the details are covered, here is the implementation of the SLPF in C++.

> *Due to a request from my professor the C++ implementation of the code has been taken down and replaced with pseudo code (There is a strong concern that students will copy the code for projects). If you are interested in my C++ implementation feel free to message me for it.*

Main Program (Pseudo code)

```
/*
    Creates edge buckets from the given edges
    @param n    Number of vertices
    @param x[]  array of x points
    @param y[]  array of y points

    @return     List of edge buckets
 */
createEdges(n, x[], y[]) {
    instantiate a new edge table


   loop through x[] & y[] pairs {
        if the edge's slope is NOT undefined (verticle) {
            create bucket with edge
            add bucket to edge table
        }
    }
}


/*
    Given the edge table of the polygon, fill the polygons

    @param edgeTable The polygon's edge table
representation
 */
processEdgeTable (edgeTable) {
    while (edge table is NOT empty) {


// Remove edges from the active list if y == ymax
        if (active list is NOT empty) {
            for (iterate through all buckets in the active
list) {
                if (current bucket's ymax == current
scanline) {
                    remove bucket from active list
                    remove bucket from edge table
                }
            }
        }


// Add edge from edge table to active list if y == ymin
        for (iterate through the bucket in the edge table)
{
            if (bucket's ymin == scanline) {
                add bucket to active list
            }
        }


// Sort active list by x position and slope
        sortTheActiveList();


// Fill the polygon pixel
        for (iterate through the active list) {
            for (from vertex1.x to vertex2.x of the bucket)
{
                setPixelColor()
            }
        }
// Increment X variables of buckets based on the slope
        for (all buckets in the active list) {
            if (bucketsdX != 0) {
                bucket's sum += bucket's dX

       while (bucket's sum >= bucket's dY) {
            increment or decrement bucket's X depending on
sign of   bucket's slope
            edge's sum -= dY
                }
            }
        }
    }
}

///
// Draw a filled polygon in the Canvas C.
//
// The polygon has n distinct vertices.  The coordinates of
the vertices
// making up the polygon are stored in the x and y arrays.
The ith
// vertex will have coordinate (x[i],y[i]).
//
// @param n - number of vertices
// @param x - x coordinates
// @param y - y coordinates
///
drawPolygon(n, x[], y[]) {
    // Create edge table
    finalEdgeTable = createEdges()


// Sort edges by minY
    sort(finalEdgeTable)


processEdgeTable(finalEdgeTable)
}
```

Header file with structs

```
/*
    Bucket struct to hold edge information
 */
struct Bucket {
    int yMax;
    int yMin;
    int x;
    int sign;
    int dX;
    int dY;
    double sum;
};

/*
    Vertex struct to hold x and y values
 */
struct Vertex {
    int x;
    int y;
};
```

. . .

That all it takes to implement the SLPF algorithm! I hope this post is constructive and that you enjoyed reading through it. If there are any mistakes and you want to leave comment, feel free to do so or send me a message.

Cheers!