Rendering

- In order to keep a faster and easier interface for drawing pixels, I used a bitmap, represented by a matrix of unsigned char for keeping track the colors of the pixels on the screen. At every display rendering callback, I only render the buffer, instead of each shape.
- In order to draw a pixel, you can set the color of the corresponding element in the bitmap.

Color Filling Algorithms

Flood Filling

- The problem with flood filling is that its recursive nature is prone to stack overflow error. In order to solve this problem, I simulate the algorithm's DFS strategy with a stack:

```
static void floodFillColoring(int x, int y)
   {
       RGBColor borderColor(0, 0, 0);
       stack<pair<int, int>> S;
       S.push({x, y});


       // while stack is not empty
       while (!S.empty()) {

           // pop from the top
           pair<int, int> top = S.top();
           S.pop();
           x = top.first, y = top.second;

           if (bitMap[x][y] == fillingColor || bitMap[x][y] == borderColor) {
               continue;
           }

           bitMap[x][y] = fillingColor;

           int dx[] = {1, 0, -1, 0};
           int dy[] = {0, 1, 0, -1};

           // Starts filling adjacent pixels
           for (int i = 0; i < 4; ++i) {
               int newX = x + dx[i],
                   newY = y + dy[i];
               if (1 <= newX && newX <= WIDTH && 1 <= newY && newY <=
                   HEIGHT && bitMap[newX][newY] != fillingColor && bitMap[newX][newY] != borderColor) {
                   S.push({newX, newY});
               }
           }
       }
   }
```

Scan Line Filling

Scan Line is highly dependent on the properties of the shape, so each shape gets its own implementation

Circle + Ellipse

Iterate a point (x, y) along the circumference from 0 to 180 degree. At each step, draw a line from the (xt - x, yt - y) to (xt + x, xt + y) with xt, yt being the coordinates of the center.

Rectangle

Iterate a point (x, y) along the left side of the rectangle. At each step draw a line from the (x, y) to (x2, y), with x2 being the maximum x coordinate of the rectangle.

Polygon

An arbitrary polygon is more complicated to fill using Scan Line Algorithm. In this implementation, I am using the following additional data structures: - Edge bucket, each bucket represents an edge: - xMin: list of x-coordinates of point with minimum y-coordinate for each edge - yMax: list of y-coordinates of point with maximum y-coordinate for each edge - slopeInv: list of inverse of slope for each edge - Edge Table: the list of possible y-coordinates of the scan lines for this polygon, each entry

has edge buckets in which the y-coordinate of the lowest point in that edge has the sample value of the entry - Active Edges List: The list of edges which are being considered for the current scan line being considered

Here is the main coloring function, with additional comments for instruction of what is going on:

```
void scanLineColoring(RGBColor fillingColor)
    {
        list<int> activeList;
        vector<double> xIntercept(edgeCount);
        for (int i = 0; i < edgeCount; ++i) {
            xIntercept[i] = double(xMin[i]);
        }
        for (int scanY = minY; scanY < maxY; ++scanY) {
            // add new edge to active edges list
            for (int edge : edgeTable[normalize(scanY)]) {
                /* insert the new edge in a way that keeps the active list sorted
                in an ascending order of the xIntercept value*/
                insert(xIntercept, activeList, edge, xMin[edge]);
            }


            auto cur = activeList.begin(), prev = activeList.begin();
            ++cur;
            // parity bit
            int parity = 0;
            // iterate through each pair of adjacent points
            for (; cur != activeList.end(); ++cur, ++prev) {
                if (xIntercept[*cur] == xIntercept[*prev]) {
                    if (slopeInv[*cur] * slopeInv[*prev] < 0) {
                        /* if the two edges have the same intercept (this is their intersection)
                        and their slope have different signs */
                        continue;
                    }
                }
                // update the parity bit
                parity = (parity + 1) % 2;
                // if the parity bit is even
                if (parity == 0) {
                    // draw
                    for (int x = xIntercept[*prev]; x != xIntercept[*cur]; ++x) {
                        plot(x, scanY, fillingColor);
                    }
                }
            }

            // cerase edges which is finished with its drawing
            cur = activeList.begin();
            while (cur != activeList.end()) {
                if (yMax[*cur] == scanY) {
                    cur = activeList.erase(cur);
                }
                else {
                    ++cur;
                }
            }

            // update the x-intercepts of the points
            for (cur = activeList.begin(); cur != activeList.end(); ++cur) {
                xIntercept[*cur] += slopeInv[*cur];
            }
        }
    }
```