# COMP 557 Extra Notes

**These notes are for your interest only. You are NOT responsible for the material.**

## Filling Polygons

In lecture 1, we looked at how to draw a line. Here are some extra notes on how to draw a polygon, in particular, to fill the region bounded by a polygon.

Let's assume the vertices of the polygon are pixels (integer coordinates). We allow the polygon to be *non-convex* . For simplicity, we also assume the edges do not intersect (other than at the vertex where two adjacent edges meet).

A set is convex if the line segment joining any two points lies entirely within the set. Define a 2D convex polygon by an ordered set of M vertices, with edges joining the vertices such that the $i$th edge joins the $i$th vertex to the $i + 1$st vertex and the Mth edge joins the Mth vertex to the first vertex. The algorithm below does not assume the vertices have integer coordinates, but let's just assume this, i.e.

```
int  E[M][2];
```

A principled approach to filling such a polygon is as follows: for each image row, we find the intersection of the row (called a *scan line*) with each edge of the polygon, then ensure that the edges are sorted appropriately, then fill in "spans" between adjacent pairs of edges in the list.

Given the polygon, create an *edge table (ET)* as follows. For each row y in the image, create a list of edges that have exactly one vertex at this row and such that the other vertex is at a higher row, such that "higher" means larger y. (We ignore horizontal edges. We'll see why shortly.) Each edge in the ET is represented as a triplet: $(x, 1/m, y_{upper})$, where x is the image column of the vertex in row y, $1/m$ is the inverse of the slope of that edge, and $y_{upper}$ is the y value of the other vertex. Sort the edges within each list (row of the ET) : sort by x first and by 1/m second. i.e. If two edges have the same x coordinate, then sort by 1/m. (We'll see why shortly.)

Once the ET has been built, we fill the polygon. We proceed row by row. For each row, we compute a sorted list of edges that intersect the row, called the active edge list (AEL). For each edge in the AEL, we keep track of $\{x, 1/m, y_{upper}\}$, except that now x is a float.

```
initialize the AEL to be empty
for y := 1 to N_rows {
    insert non-horizontal edges from row y of ET into AEL,
        respecting the sorting rule.
    remove edges from AEL  for which y = y_upper
    fill pixels between adjacent pairs of edges in AEL
    update the x value of each edge in AEL by adding  1/m;
  }
```

To see why we add $1/m$, we note that $m = \Delta y/\Delta x$. But $\Delta y = 1$ as we go from one row to the next. Hence $\Delta x = 1/m$. Now we can see why we sorted the edges by $1/m$ in the case that two edges have the same $x$ value. The reason is that when we go to the next row, we want to be sure that the edges are ordered in terms of increasing $x$ values witin the row, so we can easily figure out which pixels should be filled.

[You might be wondering about the condition that edges are non-intersecting, and where this condition appears in the algorithm. It appears at the `fill pixels between adjacent pairs of edges` step. We examine the first and second edges (and third and fourth, and fifth and sixth, etc) of the list and fill pixels between the x values of each of these edge pairs. If edges can intersect, then we would need to reorder the edges in the list, prior to filling. ]

When implementing this algorithm, we would need to have a policy for whether we fill points that lie on an edge. We may have to change the algorithm slightly to ensure this policy is correctly implemented. For example, suppose we wished to fill pixels that lie on edges. This is a fine policy, but note that we would need to treat certain vertex pixels in a special way: if two edges both terminate at this pixel (a "roof" vertex), then the algorithm would delete these edges before the fill step. Similarly, horizontal edges would need to be treated more carefully. (As it stands, the algorithm may or may not fill a horizontal edge. It depends on whether the edges that are attached to this horizontal edge are above or below.)

Finally, observe that it would be useless for the algorithm to insert horizontal edges because $y_{lower} = y_{upper}$ for such edges and so the algorithm would remove the horizontal edge right away.

## Example

Consider a polygon defined by the four vertices (0,5), (5,5), (10,1), (6,7). Show that the pixels that are filled are the following:
Solution: the edge table contains of two lists:

$$\begin{aligned} row\ 5 &: (0,3,7) \\ row\ 1 &: (10, \frac{-5}{4}, 5) \rightarrow (10, -\frac{2}{3}, 7) \end{aligned}$$

The active edge list consists of the following (after the insertion/deletion steps):

$$\begin{aligned} y =\ 7 &: \\ y =\ 6 &: (3,3,7) \rightarrow (6\ \frac{2}{3}, -\frac{2}{3}, 7) \\ y =\ 5 &: (0,3,7) \rightarrow (7\ \frac{1}{3}, -\frac{2}{3}, 7) \\ y =\ 4 &: (6\ \frac{1}{4}, \frac{-5}{4}, 5) \rightarrow (8, -\frac{2}{3}, 7) \\ y =\ 3 &: (7\ \frac{1}{2}, \frac{-5}{4}, 5) \rightarrow (8\ \frac{2}{3}, -\frac{2}{3}, 7) \\ y =\ 2 &: (8\ \frac{3}{4}, \frac{-5}{4}, 5) \rightarrow (9\ \frac{1}{3}, -\frac{2}{3}, 7) \\ y =\ 1 &: (10, \frac{-5}{4}, 5) \rightarrow= (10, -\frac{2}{3}, 7) \end{aligned}$$

The pixels that would be filled by the algorithm (assuming that we fill pixels that lie on the edges) are the following.

$$\begin{aligned}
y = \ & 7 \quad : \\
y = \ & 6 \quad : \quad (3,6),(4,6),\ldots,(6,6) \\
y = \ & 5 \quad : \quad (0,5),(1,5),(2,5),\ldots(7,5) \\
y = \ & 4 \quad : \quad (7,4),(8,4) \\
y = \ & 3 \quad : \quad (8,3) \\
y = \ & 2 \quad : \quad (9,2) \\
y = \ & 1 \quad : \quad (10,1)
\end{aligned}$$