

Polygon Scan Conversion

1 Triangles

Fast triangle scan-conversion is based on linearity of the edges. If an intersection of a scanline and an edge is known, then to get the intersection of that edge with the next scanline below, some constant (depending on the edge's slope) needs to be added to the x-coordinate of the current intersection (see Figure 1).

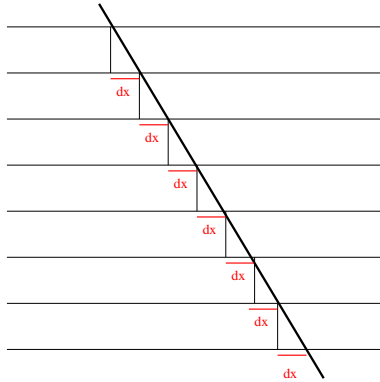


Figure 1: Intersections of a line with horizontal scanlines.

This leads to the algorithm in Figure 2. For simplicity, let's assume that the triangle's vertices have integer coordinates. The algorithm works by walking through the scanlines intersecting the triangle starting from the uppermost one until the lowermost is reached. Each such scanline intersects the triangle's boundary at two points (unless degeneracy like horizontal edge in the triangle appears – see next page). The x-coordinates of the intersection points are updated at each step by adding a constant (one over the edge's slope).

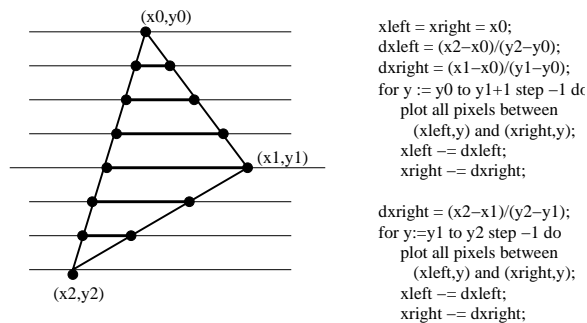


Figure 2: Triangle scan-conversion.

Clearly, things change slightly if the vertex (x_1, y_1) is on the left of the edge

joining the other two vertices. In this case, the increment that is applied to the intersection point on the left needs to be changed (Figure 3). Also, care must be taken of horizontal edges.

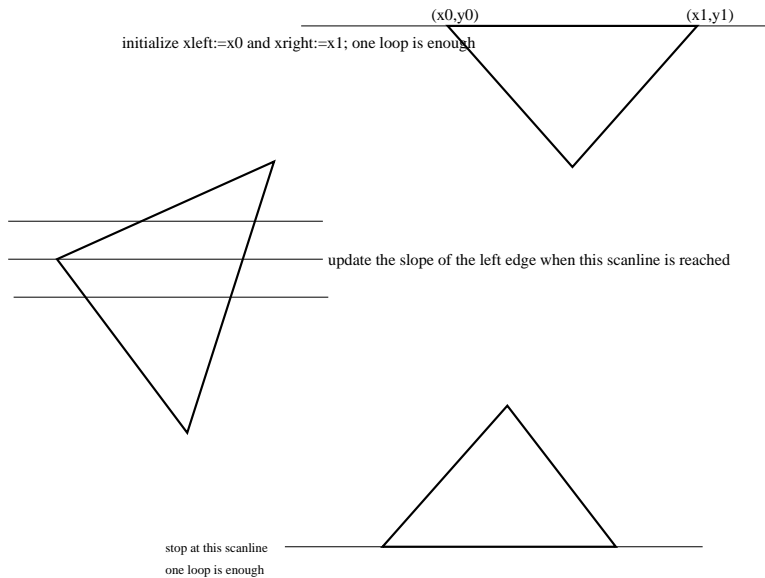


Figure 3: Triangle ZOO: each triangle type has to be treated differently.

2 Linear interpolation while scan-conversion

A similar idea as that of adding a slope-related constant to update the intersection point applies to interpolated quantities. Linear functions change by a constant along any vector. Thus, in particular, if we precompute the change along horizontal and vertical unit vectors (Figure 4), we can use them to compute the value at a neighboring pixel from the available value at the current one.

3 Polygons

Our algorithm for triangle scan conversion can be generalized to the case of a general polygon. Here is how.

Just as before, we'll march through the scanlines starting from the one through the uppermost vertex (let's say they all have integer coordinates) until we reach the one through the lowermost one. We won't have just two intersection points now: there may be more. We will keep track of all of them and use them to find pixels inside the polygon. Notice that the intersected edges (we call them active edges) and the order of intersections does not change if our

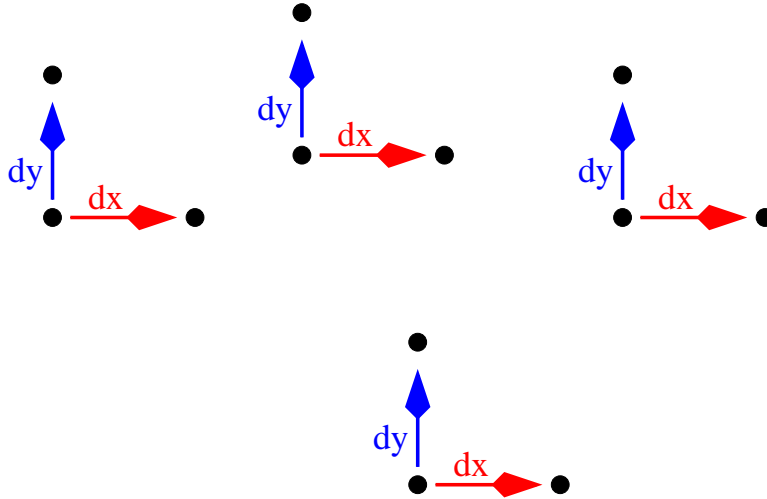


Figure 4: The amount of change of a linear function along the blue and red vectors is the same everywhere.

scanline doesn't pass a vertex. Thus, we will maintain the list of active edges, changing it only when a vertex is passed (See Figure 5). If, for each scanline, the pixels between the intersections with the first and the second, the third and the fourth, the fifth and the sixth.... edges are plotted, we'll end up with a rasterized image of the polygon.

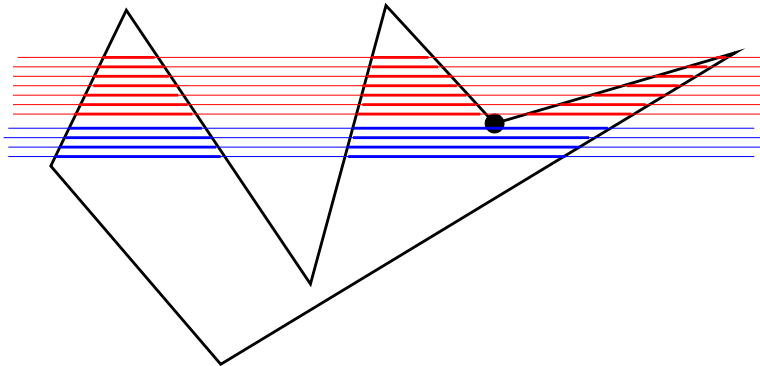


Figure 5: Active edges for all red scanlines are the same. After the scanline passes the big-black-dot-vertex, the active edge list changes (in this case, the two meeting at the dot have to be removed). Notice that pairing up the intersections starting from the left and filling pixels between intersection in each pair leads to correctly scan-converted polygon.