# SCAN CONVERTING POLYGONS

JONATHAN R. SENNING

## 1. OVERVIEW

The algorithm described here can be used to scan convert a nonconvex polygon. Holes within the polygon are not allowed and none of the polygon's edges must cross any other edges of the polygon. The polygon is specified by a vertex list, with the vertices given in a counterclockwise order around the polygon.

The basic approach is to generate a list of edges for the polygon and then make necessary modifications so that when moving from left to right along a scan line an edge will indicate a change from the polygon's interior to exterior or vice-versa.

## 2. TRANSFORM THE VERTEX LIST TO AN EDGE LIST

The first step is to generate an edge list from the vertex list. As this is done, some non-horizontal edges may be shortened so that no more than one edge intersects any position in the raster.

Let $\{v_1, v_2, \ldots, v_n\}$ denote the list of vertices, where $v_i = (x_i, y_i)$. Let $\{e_1, e_2, \ldots, e_n\}$ denote the list of edges, where the edge $e_i$ is constructed from the vertices $v_i$ and $v_{i+1}$ with the understanding that $v_{n+1} = v_1$. This gives

$$\begin{aligned} e_i &= \{(x_i, y_i), (x_{i+1}, y_{i+1})\}, \qquad i = 1, 2, \ldots, n-1 \\ e_n &= \{(x_n, y_n), (x_1, y_1)\} \end{aligned}$$

Note that vertex $v_i$ is common to both edges $e_{i-1}$ and $e_i$.

It is important that there be an even number of edges intersecting each scan line. On scan lines which do not contain any of the polygon's vertices this will be true. At the location of some vertices, however, there may be an odd number of edges and so at least one edge must be shortened to change this. If

$$dx_i = (x_{i+1} - x_i), \qquad dy_i = (y_{i+1} - y_i)$$

then

(1) If $dy_i \neq 0$ **and** $dy_{i+1} \neq 0$ (neither edge is horizontal) then
   - If $dy_i$ and $dy_{i+1}$ have different signs then **do not shorten** either edge.
   - If $dy_i$ and $dy_{i+1}$ have same sign then **shorten** either $e_{i-1}$ or $e_i$.

(2) If $dy_i = 0$ **or** $dy_{i+1} = 0$ (at least one edge is horizontal) then
   - If $(dx_{i+1})(dy_i) \leq (dx_i)(dy_{i+1})$ then **do not shorten** either edge.
   - If $(dx_{i+1})(dy_i) > (dx_i)(dy_{i+1})$ then **shorten** the non-horizontal edge.

This logic can be coded in the following manner:

---

*Date*: Written in 1993. Revised 2001, 2006.

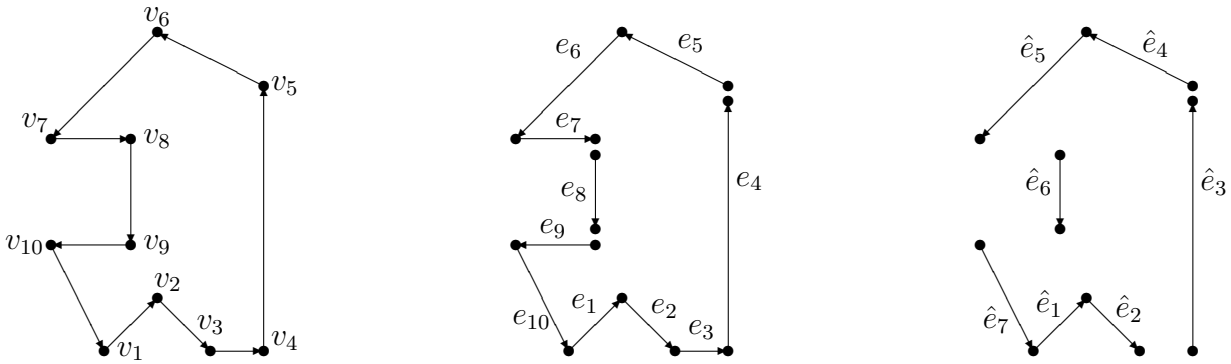
$k \leftarrow (dy_i)(dy_{i+1})$
if $k < 0$ then
    do not shorten either edge
else if $k > 0$ then
    shorten either edge
else
    if $(dx_i)(dy_{y+1}) \leq (dx_{i+1})(dy_i)$ then
        do not shorten either edge
    else
        shorten non-horizontal edge
    endif
endif

Once the edge list with shortened edges has been generated, all horizontal edges are removed. This creates a new list of edges, denoted $\{\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_m\}$ where $m \leq n$ since the number of non-horizontal edges is less than or equal to the number of edges on the polygon. The circumflex above the $e_i$ denotes that these are the edges in the edge set after the horizontal edges have been removed.

The mapping from vertex list to shortened edge list to shortened edge list with no horizontal edges can be diagrammed as

$$\{v_1, v_2, \ldots, v_n\} \rightarrow \{e_1, e_2, \ldots, e_n\} \rightarrow \{\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_m\}.$$

Figure 1 shows the steps of edge shortening and horizontal edge removal.




Original Edge Set  Shortened Edge Set  Final Edge Set

FIGURE 1. Vertex List to Edge List Conversion

## 3. INCREMENTAL CALCULATION OF EDGE POINTS

The computation of the point of intersection between a scan line and an edge is made somewhat more efficient than might be expected by the property of **coherence**. If the point of intersection of the edge and a given scan line is known, computing the edge's point of intersection on an adjacent scan line is straightforward.

The points $(x_i, y_i)$ and $(x_{i+1}, y_{i+1})$ are both on an edge with slope $m$ provide that

$$y_{i+1} - y_i = m(x_{i+1} - x_i)$$

and between adjacent scan lines $y_{i+1} - y_i = 1$ so we find

$$x_{i+1} = x_i + \frac{1}{m}.$$

This means that if $x_i$ is the point where the edge intersects the $i^{th}$ scan line, then the point at which that same edge intersects the $i + 1^{st}$ scan line is at $x_{i+1} = x_i + 1/m$.

Since $m$ and $1/m$ are rational numbers, floating point data must be used for $x$ in order to compute $x_{i+1}$ correctly from $x_i$. Alternatively, a Bresenham-style algorithm could be used to avoid the use of floating point numbers.

## 4. Edge Table Construction

The next step is to associate each scan line with the edges it intersects. This is done by indexing the edges by the smallest $y$-value on each edge.

Let $y_i$ be the minimum $y$-value on edge $\hat{e}_i$; this will of course be the $y$-value at one of the edge's endpoints. Figure 2 shows what the $y_i$ values are are for the example polygon.

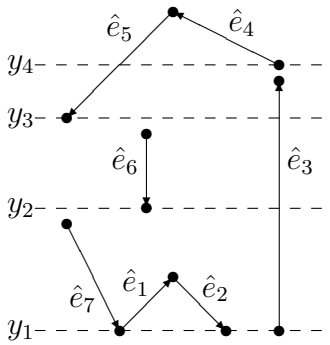



FIGURE 2. How minimum $y$ values are used to construct edge table

Each edge $\hat{e}_i$ can be associated with one of $y_1$, $y_2$, $y_3$ and $y_4$ since these comprise the set of minimum $y$ values for all of the edges. This association results in the following logical edge table:

$$y_1 : \{\hat{e}_1, \hat{e}_2, \hat{e}_3, \hat{e}_7\},$$
$$y_2 : \{\hat{e}_6\},$$
$$y_3 : \{\hat{e}_5\},$$
$$y_4 : \{\hat{e}_4\}.$$

Assume that the frame buffer is updated from bottom to top, scan line by scan line. As each scan line is processed, the edge table is checked to determine if any edges start on the scan line. The necessary information about each edge must also be included in the edge table. This includes the information necessary to compute the $x$ value of the intersection between a given scan line and each edge and the $y$ value of the scan line at which the edge ends.

As discussed above, each edge has a minimum $y$ value, identified $y_{min}$. Call the corresponding $x$ value for that endpoint $x_{min}$ even though the $x_{min}$ value may be greater than the $x$ value of the other endpoint.

Only the minimum amount of data necessary to determine all points on the edge should be stored in the edge table. This can be done by storing $x_{min}$, $y_{min}$, $1/m$, and $y_{max}$. Note that $y_{min}$ is stored implicitly as the index into the table, while the other three values need to be stored explicitly. Although the algorithm being described does not use it directly, the following pseudo-code shows that the information stored for an edge in the edge table is sufficient to compute all the points on the edge.

```
x ← x_min                    (x is a floating point variable)
y ← y_min                    (y is an integer variable)
m_inv ← 1/m                  (m is the slope and is a floating point variable)
while ( y ≤ y_max ) do
    set_pixel( round( x ), y )
    x ← x + m_inv
    y ← y + 1
end while
```

The data structure for the edge table can be an array with one entry for each scan line where the $i^{th}$ array element is a pointer to a linked list containing all edges whose $y_{min}$ value is $y_i$. If no edge begins on a given scan line, the corresponding entry in the edge table should be NIL. The edge table data structure is shown in Figure 3. All the edges with $y_{min} = y_i$ should be stored in order of increasing $x_{min}$.

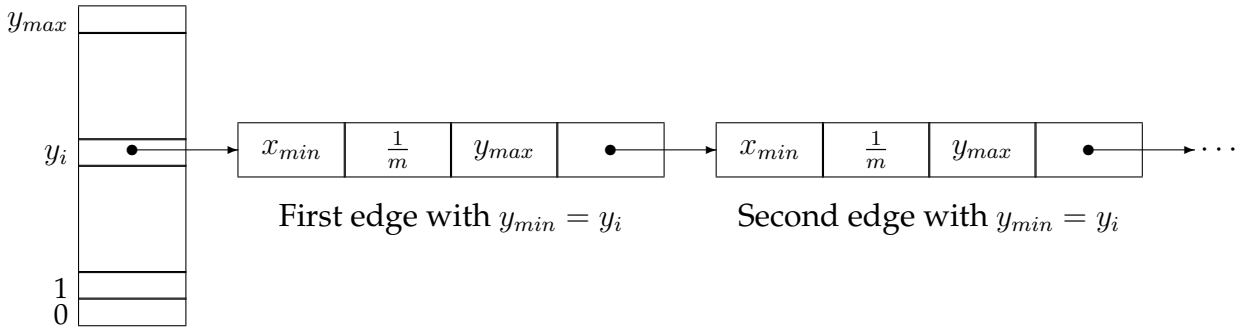



FIGURE 3. Edge Table Data Structure

## 5. ACTIVE EDGE LIST (AEL)

Once the edge table has been constructed, the process of updating the frame buffer can begin. This is done by starting at the bottom scan line, $y = 0$, and working up. The first step in processing each new scan line is to look in the edge table to determine if any edges start on the scan line. If so, then the edge specification must be merged into another data structure, the Active Edge List (AEL), sometimes called an Active Edge Table.

The AEL is a dynamic structure that contains an entry for each edge that intersects the scan line currently being processed. Thus, before the processing of the scan line begins, a check is made of the edge table to see if any new edges need to be added to the AEL (i.e., an edge starts on the current scan line). Then the AEL is examined to which portions

of the scan line should be filled. Finally, the data in the AEL is updated so that it will be correct for the next scan line and any edges for which $y_{max}$ has been reached are removed.

The AEL is easily implemented as a linked list whose elements have the same fields as the elements of the edge table; see Figure 4. The value of $x_{current}$ is initially set to $x_{min}$ and incremented by $1/m$ after each scan line is processed.

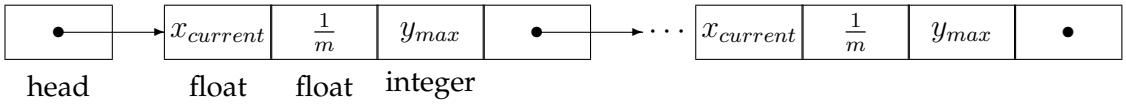



FIGURE 4. Active Edge List Data Structure

When merging edges from the edge table into the AEL, the edges should be inserted in order of increasing $x_{current}$ values. This ensures that the interior of the polygon is filled correctly. If duplicate $x_{current}$ values are encountered either one may appear first, assuming that after updating the $x_{current}$ values the AEL is reordered in increasing $x_{current}$ values.

## 6. SUMMARY OF ALGORITHM

The algorithm described here can be summarized as follows.

construct edge list with shortened edges
remove horizontal edges from edge list
create edge_table using the shortened edge descriptions
initialize AEL to an empty list
for $y$ = 0 to number of scan lines $-1$ do
    if edge_table[$y$] $\neq$ NIL then
        merge edges pointed to by $y$ in edge_table into AEL
    plot pixels along scan line between pairs of edges
    purge edges from AEL for which $y_{max} = y$
    update each $x_{current}$ value in AEL
    reorder AEL (sort on $x_{current}$ values)
end for

*E-mail address*: jonathan.senning@gordon.edu

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE, GORDON COLLEGE, 255 GRAPEVINE ROAD, WENHAM MA, 01984