CHAPTER 5 – CLASSIFICATION OF VERTICES

The vertices of the polygon are classified as a preprocessing step to the filling algorithm. In addition to the Edge Type and Vertex type classifications, the vertices must be sorted in the order they will be encountered as stopping points in the filling algorithms.

The vertices are sorted by x coordinate within y coordinate to serve as stopping points for the algorithm. A stopping point is only a true event point if it lies within the region being filled, but this is not determined until the point is encountered.

Assuming that the polygon is to be filled in a vertical direction on the display, the stopping points are sorted by increasing y value. In order to properly handle events that occur at the same y value, points with the same y value are also sorted by increasing x value.

A polygon is represented by a sequence of exterior vertices followed by a (possibly empty) set of disjoint sequences of interior vertices. Each sequence of interior vertices forms the boundary of a hole in the polygon. The exterior vertices are listed in counterclockwise order while the interior vertices are listed in clockwise order. Boundaries may intersect at their vertices, but interior boundaries may not intersect the exterior boundary.

The vertices are classified by considering them in order around the boundaries. The proper Edge Type and Vertex Type can both be determined by the spatial relationship of a vertex to its successor and predecessor vertices. In the case of horizontal edges, the successor of the successor or the predecessor of the predecessor must also be considered.
Each vertex is represented by a data structure which includes its coordinates, pointers to its predecessor and successor vertices on the polygon boundary, edge type classification, vertex type classification, and an index to a sorted array of vertices to indicate the next vertex to be considered as a stopping point during filling. Two additional fields, LeftVert and MergeEdge, are necessary to handle the merging of multiple segments at the same y coordinate. These fields are needed only for the Merge/Split algorithm.

The vertex data structure is created as the vertices are input. All the fields except the classification and sorted fields are initialized during the input phase.

The data structure for an edge exists only when the edge is bounding a region that is currently being filled. This data structure contains pointers to the starting and ending points of the edge, current coordinates for the point on the edge at the scanline being filled, a flag to indicate if the edge has been initialized, and various fields needed to rasterize the edge. The AEL algorithm also needs a pointer to the next edge in the active edge list.

5.1 EDGE TYPE

Each vertex is classified as SUCC_HORIZON, PRED_HORIZON, or NO_HORIZON based on whether it is part of a horizontal edge or not and whether the horizontal edge extends from the first (leftmost) vertex to its successor or predecessor vertex.
5.2 VERTEX TYPE

A vertex v is either a START vertex or a SPLIT vertex if both the successor vertex and the predecessor vertex are above v. A vertex v is a START vertex if the predecessor vertex is to the left of the line that contains v and its successor vertex (Figure 5-2 (a)). A vertex v is classified as a SPLIT vertex if the predecessor vertex is to the right of the line that contains v and its successor vertex (Figure 5-2 (b)).

For the parallel implementation of the Merge/Split algorithm, the vertices that are classified as START are stored in a linked list so that independent regions may be begun simultaneously.

If both the predecessor and successor vertices are below v, the vertex is either a STOP vertex or a MERGE vertex. The vertex v is classified as a STOP vertex if the successor vertex is to the left
of the line that contains v and its predecessor vertex (Figure 5-2 (c)) or as a MERGE vertex if the successor vertex is to the right of the line that contains v and its predecessor vertex (Figure 5-2 (d)).

If either the successor vertex or the predecessor vertex is above v and the other is below v, the vertex is a CONTINUE vertex. The vertex is classified as CONTINUE_SUCC or CONTINUE_PRED depending on whether the next edge created as the region is continued is formed by the vertex and its successor or predecessor. If the successor vertex is below v and the predecessor is above v, the vertex is classified as CONTINUE_PRED (Figure 5-2 (e)). The vertex is a CONTINUE_SUCCE if the opposite case is true (Figure 5-2 (f)). In the following figure, v represents the vertex being classified, P represents its predecessor and S represents its successor.

The classification of vertices with an adjacent horizontal edge is performed in the same manner as when such an edge is not present except that the comparison is modified slightly. To determine if
the successor and predecessor vertices are above or below $v$, the successor of the successor (SS) must be used rather than $S$ if $v$ is SUCC_HORIZON or the predecessor of the predecessor (PP) must be used rather than $P$ if $v$ is PRED_HORIZON.

The comparison of a successor or predecessor to an edge must also be modified. In the case where no horizontal edge is present, a vertex is classified as a START vertex if $S$ and $P$ are above $v$ and $P$ is to the left of the line from $v$ to $S$. If the vertex being classified is SUCC_HORIZON, $P$ must be to the left of the line from $S$ to SS. The case with no horizontal edge can also be stated as requiring $S$ to be to the right of the line from $v$ to $P$. The equivalent test for the START vertex that is also PRED_HORIZON is to require $S$ to be to the right of the line from $P$ to PP.

The above comparisons for horizontal edges can be simplified from a point and line comparison to a coordinate comparison. If both points ($P$ and SS or PP and $S$) are above $v$, $v$ is a START vertex if it is SUCC_HORIZON and $v$ is to the left of $S$ or if it is PRED_HORIZON and $v$ is to the right of $P$.

The vertices of the following figure have the same classification as their counterparts in the previous figure. The successor of the successor of $v$ is represented by SS and the predecessor of the predecessor is represented by PP.
When a horizontal edge is present, both endpoints of the horizontal edge have the same Vertex Type. One of the vertices is of Edge Type SUCC_HORIZON while the other is PRED_HORIZON.

If the vertex being classified (and, obviously, its successor and predecessor) are part of a hole boundary rather than part of the exterior boundary, the classifications are performed in an identical manner. Since the vertices are entered in clockwise rather than counterclockwise order, the region being filled is on the same side of an edge bounding a hole as an edge on the exterior boundary. Thus the region being filled lies to the left of an edge if the edge is traversed from a vertex to its successor. Figure 4.2 represents most of the cases for a polygon with an interior hole.
In the Vertex Classification Algorithm, \( p v \) is the vertex used as the predecessor for classification purposes while \( s v \) is the vertex used as the successor.

5.3 STOPPING POINTS

When a region is being filled, stopping points are encountered. These points may or may not cause the region being filled to be modified. Some stopping points are associated with other regions. In order to not interrupt the filling process for such points, each stopping point would have to be linked to a region. In order for this to be done, the regions of the polygon must be determined before the filling process is begun. Much of the same work would need to be repeated in order to handle CONTINUE vertices. Therefore, this possibility is not practical. The order of the vertices is maintained by constructing a sorted vertex list using pointers in the vertex data structure.
for each vertex v
    sv = SuccVert( v )
    pv = PredVert( v )
    if ( sv.y = v.y )
        v.EdgeType = SUCC_HORIZON
        ssv = SuccVert( sv )
    else
        ssv = sv
        if ( pv.y = v.y)
            v.EdgeType = PRED_HORIZON
            ppv = PredVert( pv )
        else
            ppv = pv
            v.EdgeType = NO_HORIZON
    if ( ppv.y > v.y )
        if ( ssv.y < v.y ) v.VertexType = CONTINUE_PRED
        else
            if ((( v.EdgeType = NO_HORIZON) and (pv is to right of v-sv))
            or
            (( v.EdgeType = PRED_HORIZON) and (v.x < pv.x))
            or
            (( v.EdgeType = SUCC_HORIZON) and ( v.x > sv.x)))
                v.VertexType = SPLIT
            else
                v.VertexType = START
    else
        if ( ssv.y > v.y ) v.VertexType = CONTINUE_SUCC
        else
            if ((( v.EdgeType = NO_HORIZON) and (pv is to left of v-sv))
            or
            (( v.EdgeType = PRED_HORIZON) and (v.x > pv.x))
            or
            (( v.EdgeType = SUCC_HORIZON) and ( v.x < sv.x)))
                v.VertexType = MERGE
            else
                v.VertexType = STOP

**Vertex Classification Algorithm**