CHAPTER 6 – MERGE/SPLIT ALGORITHM

6.1 OVERVIEW

The Merge/Split algorithm derives its name from the use of vertices classified as MERGE to combine regions being filled and vertices classified as SPLIT to create new regions.

The algorithm treats regions of the polygon, each defined by a pair of bounding edges, as independent areas to be filled. The classification of the vertices that are encountered as event points drives the algorithm and causes the region to be modified.

When a region bounded by two edges is being filled, the actual calculation of the proper values for the pixels takes place one scan line at a time. Therefore, the edges are scan converted one step in the y direction at a time.

The approach used to scan convert the edge is incremental so that linear interpolation of z-value, color, intensity, etc. can be done. Since it is possible that the next pixel generated does not have a different y value, the rasterization algorithm may need to be called more than once to assure that the next edge pixel is on the next display line.

The standard midpoint line algorithm as presented by Foley [21] and as originally presented by Pitteway [22] and modified by van Aken [23] is used in the integer implementation of Merge/Split. More recent algorithms were investigated [24, 25] but they present no advantage for this application since they typically take advantage of symmetry considerations and use double
(or other multiple) step techniques. The symmetry considerations allow the second half of the 
line to be drawn from the first half with little additional effort. For the algorithms presented here, 
the work might be wasted if there is an intervening vertex that causes a merge or split operation. 
Additionally, two edges are being processed at a time and the two edges will not typically have 
the same end coordinate so that converting edges from the opposite end would have the edges 
unsynchronized in y value.

In this application, since the edges are traversed and rasterized in the positive y direction, only 
octants 1, 2, 3, and 4 are possible. The initial value of “d”, the decision variable, and the positive 
and negative increment values are calculated and stored with the edge. If the polygons being 
filled have additional attributes, such as z-value, surface normal, color, and intensity values, these 
values and the associated increments would be stored with the edge. The values are determined 
from the corresponding values stored as part of the vertex data structure for the vertices the define 
the edge.

When a region is being filled, the y value of the next vertex in the sorted vertex list is used as a 
stopping point for the filling process. When this y value is reached, the corresponding vertex is 
examined to see if it is the end point of one (or both) of the edges or if it is a SPLIT vertex which 
is located within the region being filled. The region inclusion check is simple since the current x 
value is stored for each edge and the x value of the vertex in question is known. Since the test is 
trivial at this point, it would be inefficient to check stopping point vertices for region inclusion for 
all possible regions as a pre-processing step.

If the stopping point is neither the end point of one of the edges nor a split vertex within the 
region, it must lie outside the region and the filling process is resumed until the next stopping 
point is reached.
The Merge/Split Algorithm and related procedures are presented below. Edges of a segment are e1 and e2, v is the vertex causing a transition, and vs is stopping point for the region. The function PredEdge builds an edge from a vertex v to its predecessor and SuccEdge builds an edge from vertex v to its successor. NextVert returns the next vertex in sorted order to serve as the next stopping point. EndVert returns the ending vertex of the edge supplied as a parameter. The fill function fills a horizontal line between at value y between two supplied x values.

Note that when implemented, recursive calls would be replaced with continuation of a region as possible (e.g. CONTINUE_PRED).

```plaintext
for each vertex v in list of START vertices
    if ( v.EdgeType != PRED_HORIZON)
        e1 = PredEdge( v )
    if ( v.EdgeType = SUCC_HORIZON )
        e2 = SuccEdge( SuccVert( v ) )
        vs = NextVert( SuccVert( v ) )
    else
        e2 = SuccEdge( v )
        vs = NextVert( v )
    fill_segment( v, e1, e2, vs )
```

**Merge/Split Algorithm**
Procedure split(e1, e2, v)

    if (e2.x < e1.x) swap(e1, e2)
    if (v.EdgeType = NO_HORIZON)
        e3 = SuccEdge(v)
        e4 = PredEdge(v)
        vs = NextVert(v)
    else /* PRED_HORIZON */
        fill(v.x, PredVert(v).x, v.y)
        e3 = SuccEdge(v)
        e4 = PredEdge(PredVert(v))
        vs = NextVert(PredVert(v))
    fill_segment(v, e1, e3, vs)
    fill_segment(v, e4, e2, vs)

Split Algorithm

Procedure merge(v, e)

    if (v.MergeEdge != NULL)
        fill_segment(v, e, v.MergeEdge, NextVert(v))
    else if (v.LeftVert != NULL)
        merge(v.LeftVert, e)
    else
        v.MergeEdge = e

Merge Algorithm
Procedure fill_segment( v, e1, e2, vs )

while (e1.y < vs.y)
    rasterize e1 until e1.y changes, rasterize e2 until e2.y changes
    fill (e1.x, e2.x, e1.y)
if (vs = EndVert( e1) or vs = PredVert( EndVert( e1 ) )
    if ( vs.EdgeType != NO_HORIZON) fill to predecessor or successor
        case ( vs.VertexType)
            STOP: return
            CONTINUE_PRED:
                if( vs.EdgeType = NO_HORIZON)
                    e1p = PredEdge( vs )
                    vsp = NextVert( vs )
                else if (vs.EdgeType = PRED_HORIZON)
                    e1p = PredEdge( PredVert (vs) )
                    vsp = NextVert( PredVert( vs ) )
                else
                    e1p = PredEdge( vs )
                    vsp = NextVert( SuccVert( vs ) )
                fill_segment( vs, e1p, e2, vsp )
            MERGE:
                if ( vs.y = EndVert(e2).y)
                    EndVert( e2 ).LeftVert = vs
                    if ( EndVert( e2 ).MergeEdge != NULL )
                        merge( vs, EndVert( e2 ).MergeEdge )
                else 
                    merge( vs, e2 )
                return
        return
    else if ( vs = EndVert( e2 ) or vs = SuccVert( EndVert( e2 ) )
        if ( vs.EdgeType != NO_HORIZON)
            fill to predecessor or successor as appropriate
            case ( vs.VertexType)
                CONTINUE_SUCC:
                    if ( vs.EdgeType = NO_HORIZON)
                        e2p = SuccEdge( vs )
                        vsp = NextVert( vs )
                    else if ( vs.EdgeType = SUCC_HORIZON)
                        e2p = SuccEdge( SuccVert( vs ) )
                        vsp = NextVert( SuccVert( vs ) )
                    else
                        e2p = SuccEdge( vs )
                        vsp = NextVert( PredVert( vs ) )
                    fill_segment( vs, e1, e2p, vsp )
            MERGE:
                merge( vs, e1 )
                return
        else if ( ( vs.VertexType = SPLIT) and ( vs between e1.x and e2.x ) )
            split( e1, e2, vs )
        else fill_segment( vs, e1, e2, NextVert( vs ) )
        Fill Segment Algorithm
6.2 REGION FILLING

Filling a region is initiated from a vertex labeled START or from a pair of edges that form a new region after a SPLIT or MERGE vertex has been encountered. Before the region is begun, the edges are possibly not initialized.

If a region is begun from a START vertex, neither edge has been created and the region is defined only by the START vertex and its successor and predecessor vertices. The situation is slightly more complicated if a horizontal edge is present.

If a region is begun because of the action taken in response to a MERGE vertex, the two edges already exist. If a SPLIT vertex causes the two regions to be begun, each will have one edge (from the region being split) and will need to create a new edge from the SPLIT vertex and its successor or predecessor.

When a CONTINUE_PRED or CONTINUE_SUCC vertex is encountered, filling of the region continues with the terminated edge replaced by a new edge generated from the ending vertex and its predecessor or successor.

A region is filled until a STOP vertex is reached. It is possible that there will be two STOP vertices for a region since there may be a horizontal edge at this point. In this case both vertices will be labeled as STOP with one being labeled PRED_HORIZON and the other as SUCC_HORIZON.
6.2.1 Beginning a Region

If a new region is begun with a START vertex, \( V \), one edge of the region, \( E_1 \), is created from \( V \) and its predecessor, \( P \), while the other bounding edge, \( E_2 \), is created from \( V \) and its successor, \( S \). If the START vertex is also part of a horizontal edge, a horizontal fill operation is done between the vertices. One of the bounding edges is created from \( V \), which is labeled PRED_HORIZON, and its successor and the other from \( V \), which is labeled SUCC_HORIZON, and its predecessor.

\[
\begin{array}{c}
\text{P} \\
\text{V} \\
E_1 \\
S, V_5 \\
E_2 \\
\text{S, V_5} \\
\text{V} \\
\text{P} \\
\end{array}
\]

Figure 6.1 Beginning a region

As discussed earlier a new region may begin as the result of a MERGE or SPLIT vertex. More detail of this process is presented in the sections describing the action taken when these vertices are encountered.

Once the new edge(s) are created, they are initialized and the filling process is begun.
6.2.2 Encountering a Vertex

When a vertex, \( V_S \), which is also \( S \) in Figure 6.1, is encountered as a stopping point and is part of the region, the region must be modified. If \( V_S \) is labeled CONTINUE, one of the edges is replaced with a new edge. If \( V_S \) is labeled MERGE two regions are combined. If \( V_S \) is labeled SPLIT two regions are created.

As mentioned in the introduction to this chapter, encountering a STOP vertex indicates the end of the region.

6.2.2.1 CONTINUE Vertex

The case of a CONTINUE vertex is simple. Figure 6.2 illustrates the situation when vertex \( V_S \) is encountered while filling the region between edge \( E_1 \) and edge \( E_2 \).

When \( V_S \) is encountered edge \( E_1 \) is unmodified and edge \( E_2 \) is replaced by a new edge determined by \( V_S \) and \( S \). In this case \( V_S \) is labeled CONTINUE_SUCC. Note that \( V_S \) is the end point of edge \( E_2 \).

One approach to continuation of the filling process would be to treat the region as a new region and to execute a new call to the fill segment routine. However, to avoid the overhead of a function call, the region is treated as continuing with a modified edge. Since the initialization flag for the new edge has not been set, the filling routine must detect this situation and initialize the edge before continuing.
If the vertex $V_S$ were associated with edge $E_1$, it would be labeled CONTINUE_PRED and similar processing would be necessary.

If $V_S$ is CONTINUE_SUCC and is also labeled PRED_HORIZON, the new edge is created from $V_S$ and $S$. The predecessor of $V_S$, vertex $P$, is then ignored as a stopping point.

If $V_S$ is CONTINUE_SUCC and is also labeled SUCC_HORIZON, the new edge is formed from $S$ and $SS$ (the successor of the successor of $V$). Point $S$ is ignored as a stopping point.
6.2.2.2 MERGE Vertex

The most difficult case arises when the stopping point encountered is a MERGE vertex. Since the regions are being filled independently, possibly on different processors, and a MERGE vertex indicates two (or more) regions are to be combined, the region that is finished first must “wait” until the other region is finished. Here “finished” means that the same MERGE vertex is encountered in the filling of the second of the regions to be combined. If the processing is being done on a parallel system, the wait must be done in such a way so as not to prohibit the processor from working in another region.

The non-terminating edge of the first region to finish is associated with the MERGE vertex through the MergeEdge pointer field in the vertex data structure. When a region reaches a MERGE vertex, the MergeEdge pointer is checked for the presence of a waiting edge.
In the Figure 6.5, V is the MERGE vertex. In (a) the leftmost region is finished and the edge from A to B is associated with vertex V. Since the edge has been initialized and partially scan converted, the present position, P₁, is part of the edge data structure. In (b) the region bounded by the edges from C to D and from F to V reaches MERGE vertex V, the edge A to B is waiting and the new region is formed by two initialized edges, A to B with present position P₁ and C to D with present position P₂.

If a horizontal edge is present at the MERGE vertex, the continuing edge of the first region to finish is associated with the leftmost MERGE vertex regardless of which one is encountered first. The leftmost vertex can be easily determined by the PRED_HORIZON and SUCC_HORIZON labels.

![Figure 6.5 Encountering a MERGE vertex](image)

6.2.2.3 SPLIT Vertex

The result of encountering a SPLIT vertex within the region can be viewed in two ways. One view is that two new regions are created, each with one edge from the previous region. Therefore, two calls to the fill segment routine should be made. These calls may be recursive in a
sequential implementation or may be handled by placing the regions in a queue of regions to be filled in a parallel implementation.

The other view is that the region is modified and one new region is created. This approach improves efficiency in that one of the regions does not need to be stored and the overhead of starting a new region is avoided. The continuing region is treated in the same manner as if the split vertex had been a CONTINUE vertex in that one of the two bounding edges is replaced while the other is not effected.

Figure 6.6 illustrates a SPLIT vertex, V, that is encountered while filling a region bounded by edges E₁ and E₂. Edge E₁ was initialized based on vertices A and B while edge E₂ was initialized based on vertices C and D. Edge E₁ has been scan converted up to the point P₁ and E₂ to the point P₂ when V is encountered. Vertex V has successor vertex S and predecessor vertex P.

The first of the new regions is bounded by the already initialized edge E₁ and a new edge, E₃, defined by vertices V and S. The second is bounded by E₂ and a new edge, E₄, defined by vertices V and P. Note that the SPLIT vertex causes two new edges to be created much like a START vertex, except that the edges are associated with different regions.

When a new edge is first used for filling it is initialized. The edge data structure contains an initialization flag so that the fact that edges E₁ and E₂ are already initialized can be determined.
If a horizontal edge is present at the SPLIT vertex, there are actually two vertices labeled SPLIT with one also labeled SUCC_HORIZON and the other PRED_HORIZON. In this case one new edges is formed from VP (vertex labeled PRED_HORIZON) and S and the other new edge is formed from VS (vertex labeled SUCC_HORIZON) and P.

6.2.2.4 Multiple Vertices

The presence of multiple vertices at the same y value complicates the algorithm somewhat. However, since the vertices are encountered in a left to right sorted order, many such cases result
in a region being started or continued only to stop again before any filling is done. The case
where both edges of a region have a CONTINUE vertex at the same y value is, therefore, trivial.

If more than one MERGE vertex occurs on the same scan line, the “middle” region will have no
continuing edges but may be finished first, last, or someone in between. In this case, the
continuing edge is associated with the leftmost MERGE vertex as in the case where the MERGE
vertex is part of a horizontal edge. However, it is not as simple to determine the leftmost
MERGE vertex as it is in the case where the only two such vertices are “connected” by the
PRED_HORIZON and SUCC_HORIZON labels.

When a “middle” region terminates, the continuing edge is associated with the leftmost MERGE
vertex. Note that this is the vertex that actually stops the filling of the region since it is
encountered first. The rightmost MERGE vertex is associated with the leftmost by placing a
pointer to the leftmost vertex in the LeftVert field of the vertex data structure.

Figure 6.8 illustrates a situation where there are 3 MERGE vertices with the same y value. When
region A terminates edge E\_1 is associated with vertex V\_1. If there is already an edge stored in the
vertex data structure, both edges are known and the new region is determined. When region B
finishes, vertex V\_2 is checked for an associated edge. If one exists it is moved to vertex V\_1 and
the LeftVert pointer of V\_2 is set to point to V\_1. If there is no edge associated with V\_2 the pointer
is set and will be used later when an edge is associated with V\_2.
Assume that the regions finish in the order C A D B. When C finishes, the LeftVert pointer of $V_3$ points to $V_2$. When A finishes edge $E_1$ is associated with $V_1$ through MergeEdge. When D finishes edge $E_2$ is associated with $V_3$. Since the LeftVert field of $V_3$ is points to $V_2$, edge $E_2$ is associated with $V_2$. When B finishes the edge associated with $V_2$ is passed to $V_1$. Since $V_1$ already has an edge associated with it, the new region is defined by $E_1$ and $E_2$.

It is also possible to have a SPLIT vertex combined with MERGE vertices as illustrated in Figure 6.9. Region B encounters $V_1$ before it encounters $V_2$. Therefore, region B is not split. However, the LeftVert field of $V_3$ will point to $V_1$ since $V_1$ and $V_3$ are end points of the edges defining region B and they both have the same y value. Region C will stop for $V_3$ and $E_2$ will be associated with $V_3$. Eventually, regardless of order, the region between $E_1$ and $E_2$ will be defined. However, since $V_1$ was the last stopping point for this region, $V_2$ will cause the region to be split into the regions D and E. When region E is being filled, $V_3$ will be encountered but will be ignored since it is a MERGE vertex that is not endpoint of either of the bounding edges $E_2$ or $E_4$. 

**Figure 6.8 Multiple MERGE vertices**
In a similar manner, arbitrary arrangements of vertices are successfully filled by this algorithm including cases where some of the MERGE and SPLIT vertices are part of a horizontal edge.