8.1 GENERALIZED POLYGONS

The Merge/Split algorithm has been extended to allow rendering of 2D and 3D polygons that have curved sides. Normal definitions of polygons require that the area be bounded by 3 or more straight line segments. The term generalized polygon is used to refer to a flat plane region bounded by any number of edges where the edges may be either straight or curved.

As long as there is a technique to scan convert a curve in scanline increments, the curve may be an edge of a polygon filled by Merge/Split (or AEL).

Similar surfaces can be rendered more directly by several approaches. Such surfaces are often tensor product surfaces where the boundary curves are NURBS, Hermite polynomials, or Bezier curves. Such a surface is viewed as a rectangle in uv parameter space which is deformed into the 3D surface. Surfaces may also be “trimmed” by a set of curves [26]. Such curves are often converted to Bezier representation. Algorithms have also been presented to directly scan convert various types of surfaces [27, 28, 29, 30].

The surfaces rendered by the modified Merge/Split algorithm differ from tensor product surfaces in that they are “flat” surfaces bounded by curves of various types.
8.2 TYPES OF CURVED SIDES

A simple approach to scan converting a curved edge of a generalized polygon would be to convert the curve to a polyline with whatever resolution might be necessary. The obvious disadvantage of this technique is that a polygon with a very large number of sides would be created. Each of the edges would be bound by vertices and each of these vertices would need to be classified and used as a stopping point.

Various techniques have been developed to convert curved lines to polylines. Example curves are circles and ellipses [31] and generalized conic sections [32]. These algorithms are designed to generate as few line segments as possible given an error tolerance to the curve.

There has been much work done on the problem of directly rasterizing various types of curves. Adaptive forward differencing has been used to render parametric cubic spline curves [33, 34], cubic curves with integer arithmetic [27], NURBS using integer arithmetic [35, 36], and trimmed NURBS [37]. In the work of Klassen [34], Bezier curves are rasterized after they have been converted to monotone segments.

The implementation presented in this work is designed to handle curved edges that are expressed in Bezier form. Bezier curves are a natural form of representation of the sides of a polygon since two of the four control points (for cubic curves) are also vertices of the polygon. The edge data structure is modified to allow representation of the additional control points.
In addition, there are many techniques available to convert various types of curves to Bezier form [38]. The Bezier form is often used even if conversion from another form is required because [39]

1. Bezier curves have the convex hull property (the curve is contained within the convex hull of the control points)
2. subdividing the curve at a local minimum or maximum is easy
3. it is easy to raise the degree of a Bezier curve to match that of other segments

In addition, efficient methods have been developed to find local minima and maxima and the Berenstain basis is well suited for polynomial root finders used to find relative extrema [40].

Properties (2) and (3) above have also been demonstrated for other types of curves. For example, B-splines can be subdivided [41] and have their degree raised [42].

Other types of curves can be rasterized. A technique exists [23] to scan convert ellipses with a midpoint algorithm similar to that used for straight lines in this work. General types of curves which are not parametric can also be rasterized [43].

Another class of curves that can be rasterized directly includes nu splines [44] and tau splines [45]. Nu splines are piecewise approximations to polynomials under tension. Techniques have been developed to convert nu splines to Bezier representation [46]. Tau splines can also be converted to Bezier representation [47]. Thus, even curves under tension can be used as sides of polygons.
8.3 MONOTONE SEGMENTS

In order to rasterize a generalized polygon with curved edges, the curves must be broken into monotone segments. Then the vertices can be classified in the same manner as straight edges.

Klassen [34] lists several reasons the curve segments should be broken into monotone segments. One of the primary reasons for his work is that loops and cusps are properly handled. Simple polygons will not display such behavior, so this reason is not important for polygon edges.

In addition monotone edges are easier to clip. This consideration is not important in this work but might be so for a full practical implementation.

There are two reasons cited that are important in polygon filling. The first is that the intervals are better behaved numerically. The second is that splitting allows the curves, since they are monotone, to be rasterized in scanline increments.

The primary reason to break Bezier curves into monotone segments in a vertex classification scheme is that the vertices can be classified as if they were straight edges. The only points that need to be considered are the end points of the edge.

One possible complication is that curve segments could intersect. Since Merge/Split and AEL assume simple polygons, this complication is assumed to not be present.

Figures 8.1, 8.2, and 8.3 illustrate the classification of vertices that bound curved edges. Figure 8.1 is a polygon with each edge a curve segment. The end points of segments (polygon vertices) are represented by filled circles. In Figure 8.2, unfilled circles represent vertices added when
curved edges are broken into monotone segments. In Figure 8.3, the curved segments are replaced by straight lines and the vertices are classified accordingly.

Most of the vertices introduced to create monotone segments are classified as CONTINUE vertices. However, one introduced vertex is classified STOP and one is classified SPLIT. Note also that one of the original vertices is classified as a MERGE vertex.

The generalized polygon represented in these figures could also be a closed Bezier curve which could be filled as a polygon by using this approach.

![Figure 8.1 Generalized polygon](https://via.placeholder.com/150)

*Figure 8.1 Generalized polygon*

*Filled circles are vertices*
Figure 8.2 Generalized polygon with monotone segments
Open circles are vertices added to create monotone segments.

Figure 8.3 Vertices are classified as if monotone edges were straight
There are, however, some situations where treating a curved edge as a straight edge connecting the two end points can result in incorrect vertex classification. Vertex v in Figure 8.4 illustrates the problem. Dotted lines indicate the relationships that would be used if the edges are assumed to be straight lines. Figure 8.5 illustrates the proper approach to vertex classification for a curved edge. The curved edge is treated as if it were the straight edge that is the tangent to the curve at the vertex being classified as indicated by the dotted line.

![Figure 8.4 Incorrect vertex classification](image1)

**Figure 8.4 Incorrect vertex classification**

![Figure 8.5 Vertex classification using tangent lines](image2)

**Figure 8.5 Vertex classification using tangent lines**
In order to break a Bezier curve (or other type) into monotone segments, it is first necessary to identify local minima and maxima in both x and y coordinates. Since the curves are parametric, the algorithm will be identical for either coordinate. Each will generate a sequence of 0, 1, or 2 relative extrema including the ends of the interval. Combining the results from each coordinate yields a sequence of 0, 1, 2, 3, or 4 points where the curve should be split so that all segments are monotone.

One simple approach would be to evaluate the Bezier curve in increments, but this method would be essentially the same as conversion to polylines that would result in adding many vertices to the polygon. Depending on curvature and desired resolution, more than one hundred vertices could be added for each curved edge. Another option is to use a general root finder on the derivatives. The derivative can be easily found by various methods [48, 49]. The root finder can be based on binary subdivision [50] or a more specialized subdivision approach [26].

The approach used in this work is based on a technique developed by Whitted [51,52] and used in [34]. In this approach the curve is converted into a power series in t for each of the x and y parametric representations. The derivative is taken, which for a cubic curve yields a quadratic equation. A point of inflection is also easily found if present. Relative extrema are detected by comparing the values of the derivative at the end points and at any points of inflection. At most one square root is necessary, and it is needed only if it is known that it will yield a real solution.

The values of the parameter for the extrema are ordered in a list of 0 to 4 points where the curve should be split, yielding 1 to 5 intervals which will be monotone edges of the polygon.
After the splitting points are determined, the Bezier curve needs to be split in such a way that the resulting curves are Bezier curves parametric in the interval 0 to 1 so that they will be acceptable polygon edges. New control points must be created so this condition is met.

The end points of the generated segments will be points on the curve which correspond to the \( t \) values at the splitting points. For example, for an original edge \( B(t) \) with splitting points \( t_1 \), \( t_2 \), and \( t_3 \), the new curves will be:

\[
B_a (t_a : 0 \leq t_a \leq 1) = B (t: 0 \leq t \leq t_1)
\]

\[
B_b (t_b : 0 \leq t_b \leq 1) = B (t: t_1 \leq t \leq t_2)
\]

\[
B_c (t_c : 0 \leq t_c \leq 1) = B (t: t_2 \leq t \leq t_3)
\]

\[
B_d (t_d : 0 \leq t_d \leq 1) = B (t: t_3 \leq t \leq 1)
\]

Figure 8.6 Curve with 3 extrema
For Bezier curves the subdivision process has been described by Lane [52] and Farin [49, 53]. It is also possible to subdivide the curves by representing them using a power series basis and limits on parameters to represent the different segments [51, 28]. Farin’s approach has advantages for adaptive forward difference methods, however, since it increases the smallest step size with each subdivision.

The method applied in the modified Merge/Split algorithm follows that of Farin as originally proposed by deCasteljau and is numerically stable. The approach is to take a Bezier curve defined over [0,1] and split it into two curves defined over [0,c] and [c,1] in such a way as to have two curves each defined over [0,1]. A parameter, s, is introduced over the interval [0,c] and the curve C is constructed from the original curve B such that C is part of B. By requiring identical derivatives at key points on C and B, the subdivision formula

$$C_j = B_{k+1}^j(c)$$

is derived and is easily calculated in the process of evaluating B at point c using the deCasteljau algorithm.

### 8.4 MODIFICATIONS TO MERGE/SPLIT

If the polyline approach is used to represent a curved edge, the data structures and algorithms are unchanged. The conversion to a polyline from some form of input curve can be viewed as a preprocessing step.
If the curve representation is to be used for rasterization, additional information must be stored as part of the edge data structure. The edge data structure should be created at input rather than waiting for the edge to be created during the filling process as is done for straight edges. For example, if the Bezier form is used, the input will consist of four control points that define the edge. The control points must be stored until they are needed. An obvious approach is to input the control points during vertex input and create the edge data structure during this phase of the algorithm.

Since the edges are associated with a beginning and ending vertex in the context of filling, it is reasonable to associate the edge data structure created at input with the vertex data structure. The vertex data structure will have two additional fields, PredEdge and SuccEdge, both of which are pointers to edge data structures. These pointers will be NULL for a new straight edge and can be checked during the filling process. Note that the field EdgeType is not used for curved edges since there will be no horizontal edges except in the case of degenerate curved edges.

The edge data structure will need to have a field to indicate the edge type so that the proper rasterization algorithm is used. This field could indicate a straight edge and differentiate between various types of curves that might be implemented. In addition, proper values such as control points and intermediate values used for rasterization are stored in the edge data structure.

For the Bezier representation, an edge structure contains the coordinates of the control points, current parameter value, current step size, values needed for forward difference calculations, and a flag to indicate that the edge has been completely rasterized.

The input format for the algorithm must also be modified. For Bezier segments, the middle control points must be input as well as the bounding vertices. If two adjacent edges are Bezier
segments, the common end point is a control point for both edges. It is, therefore, necessary to code the input data to indicate if a value represents the end point of a straight edge or a control point of a Bezier segment. The common end point of two adjacent segments should be so indicated to avoid repetition.

One additional complication is that the vertex that is (arbitrarily) input first might also be a control point for the last edge if the last edge is a curve.

If B-spline curves are used as polygon edges, the control points must be input. The actual vertices could be input in addition or they could be calculated from the control points. The control points overlap in the sense that one control point is involved in the specification of more than one curve segment. If there are consecutive edges, control points could be linked or repeated. Repetition results in an easier implementation but would be more difficult to manage if interactive curve modification is allowed.

The first consecutive B-spline curve segment is represented by four control points, with additional consecutive segments represented by fewer control points. The case where the first edge input and the last edge input are consecutive B-splines is more problematic than with Bezier curves since the first and second edges will need values not appearing in the input data stream until the last segment unless control points are repeated.

After the data is input, the curves must be broken into monotone segments before edges are created. This process creates new vertices of the polygon to bound the monotone edges. Once all vertex end points bounding monotone segments are stored, the process of vertex classification is done exactly as in the case of a polygon with all straight edges.
The only other modification to the algorithm is ensuring that the curve of a given type is properly rasterized.