Quadratic Bezier Triangles As Drawing Primitives

J. Bruijns, Philips Research Laboratories*

ABSTRACT

We propose to use quadratic Bezier triangles as additional drawing primitives: quadratic Bezier triangles require much less model data for faithful representation of curved surfaces than planar triangles. Therefore, they require less storage and/or transmission capacity. Furthermore, they allow automatic level-of-detail. Finally, they result in considerable savings in model-view transformations and lighting calculations. We present two algorithms for rendering these triangles, each of which can be easily incorporated in hardware render systems currently used for planar triangles.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation - display algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Surface representations.

Additional Key Words and Phrases: 3D graphics rendering, graphics pipeline.

1 INTRODUCTION

Current 3D graphics ICs accelerate the rasterization of polygonal objects. However, accurate rendering of curved objects (especially the silhouettes) requires a large number of small triangles. Hence, a lot of computation is spent at the front end of the rendering pipeline (for example, transforming the vertices from world to view (eye) coordinates, performing lighting calculations, clipping the triangles, etc.). This front end is currently a bottleneck in the graphics pipeline. Rasterizer processors are idle for a substantial part of their time (depending on the relative speed of the rasterizer processor with regard to the geometry processor or CPU).

NURBS surfaces require much fewer vertices for modeling curved objects, reducing the number of transformations considerably. But hardware acceleration of the rendering process is not so easy. Therefore, if they are used during modeling, they are often transformed to polygonal meshes by the authoring system for incorporation in applications to be run on low end graphics systems. Direct rendering of NURBS surfaces will be discussed in further detail in section 1.1.

Quadratic Bezier triangles (see [3] for an introduction) also give the opportunity to model curved objects with fewer vertices. They do not have the same modeling power as NURBS (but they could be seen as a means to reduce the amount of data in planar triangle models). Moreover, we will show that they are much simpler to render. Indeed, it is relatively easy to incorporate the render algorithms, presented in section 3, in current 3D graphics hardware pipelines.

* Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
email: bruijns@natlab.research.philips.com

1998 Workshop on Graphics Hardware Lisbon Portugal

Cubic and higher-order Bezier triangles have more modeling power than quadratic Bezier triangles but also require more computational power which makes them less suitable for low end graphics systems.

The remainder of this paper is organized as follow. First, previous work will be briefly discussed. In section 2, some properties of quadratic Bezier triangles will be summarized. In section 3, methods for rendering quadratic Bezier triangles will be discussed. In section 4, the results achieved will be reported. Finally, section 5 will close with conclusions.

1.1 Previous work

Previous work on accurate and fast rendering of curved objects was mainly[1] based on the use of quadrangular parametric surfaces (e.g. NURBS). Quadrangular parametric surfaces are very powerful for modeling curved objects. They are the preferred geometric entities in computer-aided geometric modeling. Rendering methods for these surfaces can be classified in three main categories: scan line rendering, parameter rendering and tessellation.

1.1.1 Scan line rendering

Scan line rendering methods compute and render the pixels of the intersection of the scan line plane and the object to be rendered. The edge and silhouette curves are intersected to form scan line segments. Depth, surface normal and so on are linearly interpolated between the end points of these segments [1, 16].

These methods require a lot of processing power because the intersection points have to be calculated numerically. A lot of case analysis is required to prevent the algorithms failing in special cases. Therefore, these methods are less amenable to hardware acceleration.

1.1.2 Parameter rendering

Parameter rendering of a curve requires computations of the pixel position, the global normal vector and the texture vector for a set of parameter values. From $t = 0$ to $t = 1$ a sequence of parameter values has to be generated such that at least each pixel along the curve is visited. Each of these variables (pixel position and so on) is given by an nth (e.g. 3rd) degree polynomial in t between 0 and 1. Therefore, parameter rendering can be implemented by means of an adaptive forward difference algorithm [9, 10, 14]. This algorithm can be easily implemented in dedicated hardware.

Parameter rendering of a surface requires parameter rendering of a set of curves of this surface such that no pixel gaps occur between two successive curves. The next curve can also be found by means of an adaptive forward difference algorithm. But rendering surfaces is inefficient because many pixels will be evaluated more than once. Indeed, many curves have to overlap on screen to avoid pixel gaps.

1.1.3 Tessellation

Tessellation approximates a quadrangular parametric surface by means of a polygonal mesh. Forsey et. al. [4] proposed recursive subdivision with crack prevention. But, trimming curves are not

[1] In [5, 6, 8] direct rendering of (a CSG of) quadrics is described. But, these systems are for the time being too expensive and/or too slow for low end graphic systems. Besides, assigning continuously varying attributes like color and texture is not so easy for a set of connected implicit surfaces (quadrics).

dealt with.

Alyn Rockwood et al. [13] proposed the following algorithm. The surface and its trimming curves are first transformed to quadrangular Bezier patches and Bezier curves. Per patch and per trimming curve a maximum step size in u, v (surface) and t (trimming curve) direction is computed on the basis of a user tolerance in screen coordinates. A patch is divided into "uv-monotone" regions. Each region is tessellated in quads and triangles (along the trimming curve) on the basis of the maximum step sizes. The surface is evaluated at vertices of these quads and triangles. The resulting quads and triangles are rendered.

This method is incorporated in the OpenGL Utility Library [11]. However, because of some of the iterative algorithms involved, the method is not very suitable for complete implementation in 3D graphics ICs intended for consumer market products.

Abi-Ezzi et al. [1] accelerated and improved this algorithm by preprocessing the view-independent parts (up to and including the generation of the regions) but the resulting algorithm is still intended for high end graphics stations (CAD/CAM applications).

Subodh Kumar et al. [7] described an algorithm, suitable for the interactive display of large NURBS models on high end graphic systems. This algorithm overcomes some of the problems with the method of Alyn Rockwood et al. [13]. But, this algorithm is for the time being not yet suitable for low end graphic systems.

1.2 What is new?

We propose to use quadratic Bezier triangle models for the description of curved surfaces. Compared with planar triangle models this reduces the amount of data considerable (see Table 3 in section 4.4). Compared with NURBS surfaces this simplifies the rendering algorithms because there is no need for trimming curves and there is no need to subdivide the quadrangular Bezier patches in "uv-monotone" regions.

We propose a tessellation algorithm which selects the number of segments for each Bezier triangle border on the basis of the length (also done in [1, 7, 13]) and/or the curvature of the projected border (discussed in further detail in section 3.3.6). In this way cracks between connected Bezier triangles are avoided. Three tessellation grids are proposed. These three are chosen because they make it possible to compute the vertices of the generated planar triangles by means of second order forward differences. The first two tessellation grids are based on a regular subdivision of the parameter domain [12]. The third one is based on equidistant subdivision of one of the parameters. Each of the resulting grid lines is subdivided in a fixed number of segments. Given the number of segments, the rendering algorithm selects the tessellation grid which results in the fewest number of generated planar triangles (see appendix A). The three tessellation grids and their corresponding algorithms are discussed in further detail in section 3.1, 3.3.2 and 3.3.3.

2 QUADRATIC BEZIER TRIANGLES

A quadratic Bezier triangle is defined by the following mapping from the parameter domain (u, v, w) to the $\boldsymbol{x}$ range (see Figure 1):

$$\boldsymbol{x}(u,v,w) = u^2\boldsymbol{x}_{200} + v^2\boldsymbol{x}_{020} + w^2\boldsymbol{x}_{002} + 2uv\boldsymbol{x}_{110} + 2vw\boldsymbol{x}_{011} + 2wu\boldsymbol{x}_{101} \quad (1)$$

$$\text{with } u+v+w = 1 \text{ and } 0 \le u, v, w \le 1 \quad (2)$$

$\boldsymbol{x}$ is a vector with position, normal, intensity, texture coordinates and other attributes. So, equation (1) gives not only the position,

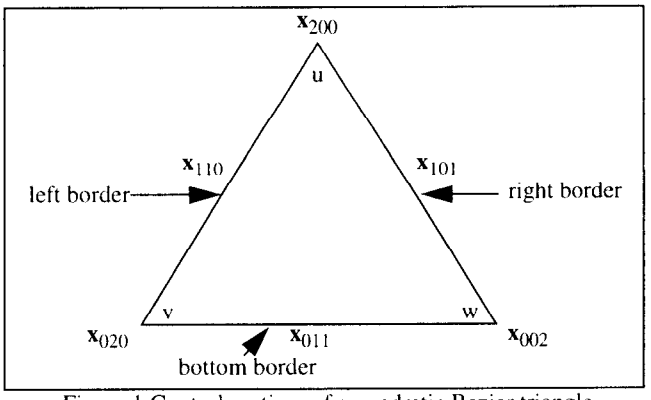

Figure 1 Control vertices of a quadratic Bezier triangle

but also the normal or the intensity, the texture and any other attributes as function of the parameters u, v and w. All attributes are accounted for with the same interpolation schema.

The corner control vertices $\{\boldsymbol{x}_{200}, \boldsymbol{x}_{020}, \boldsymbol{x}_{002}\}$ lie on the surface patch. The control vertices $\{\boldsymbol{x}_{110}, \boldsymbol{x}_{011}, \boldsymbol{x}_{101}\}$ (which will be called middle edge control vertices from now on) do not lie on the surface patch except in the case of linear edge curves.

The left border is defined by $\{\boldsymbol{x}_{200}, \boldsymbol{x}_{110}, \boldsymbol{x}_{020}\}$, the bottom border by $\{\boldsymbol{x}_{020}, \boldsymbol{x}_{011}, \boldsymbol{x}_{002}\}$ and the right border by $\{\boldsymbol{x}_{002}, \boldsymbol{x}_{101}, \boldsymbol{x}_{200}\}$.

The parameter u is 1 at $\boldsymbol{x}_{200}$ and 0 at the bottom border. The parameter v is 1 at $\boldsymbol{x}_{020}$ and 0 at the right border. The parameter w is 1 at $\boldsymbol{x}_{002}$ and 0 at the left border.

We observe that

1. the restriction on the parameters u, v and w forces $\boldsymbol{x}(u, v, w)$ to be a surface patch with only two free parameters.
2. the borders of a quadratic Bezier triangle are quadratic curves. These curves can be written in Bezier form by using the corresponding control vertices.
3. a curve for which one of the parameters u, v or w is constant is a quadratic curve.
4. two connected Bezier triangles share the control vertices of their common border. So, not only the position but also the normal vector and texture coordinates are C_0 continuous. However, a C_0 continuous normal does not imply that the shape as defined by the positions is C_1 continuous.

3 RENDERING

Quadratic Bezier triangles (see [12] for evaluation algorithms for arbitrary Bernstein-Bezier forms) can be rendered by means of tessellation after the model-view and projection transformation but before the perspective division (the rational forms due to the perspective division cannot be handled by forward differences). Each Bezier triangle is approximated by means of one or more planar triangles which are rasterized as usual. In the latter case, these triangles have been obtained by means of tessellation. Two tessellation methods are investigated: fixed subdivision and variable subdivision.

3.1 Fixed subdivision

Usually, an object consists of several connected quadratic Bezier triangles. To keep the graphics pipeline simple, each Bezier triangle should be rendered independently of the other Bezier triangles. To avoid cracks, the common boundary should be approximated by the same polyline for each connected Bezier triangle. A very

simple way of doing this is by using the same fixed subdivision for each Bezier triangle.

Fixed subdivision is based on an equidistant subdivision in the parameter domain (u, v, w) (see Figure 2). The subdivision in the

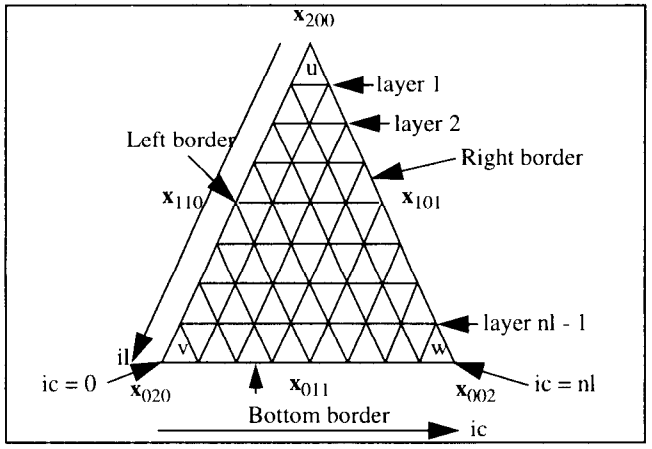

Figure 2 Fixed subdivision

u direction results in nl layers (indexed by il). The grid points along a layer are indexed by ic from 0 (at the left border) up to and including il (at the right border). The quadratic Bezier triangle is approximated by a set of planar triangles with their vertices defined on the grid of the equidistant parameter subdivision. The parameter values at this grid are:

$$u(il, ic) = (nl - il)/nl \tag{3}$$

$$v(il, ic) = (il - ic)/nl \tag{4}$$

$$w(il, ic) = ic/nl \tag{5}$$

The values of the vertices at this grid follow by substitution of equations (3), (4) and (5) in equation (1).

3.2 Fast fixed subdivision

Fast fixed subdivision (which will be called FFS below) accelerates fixed subdivision by repeatedly applying forward differences to remove the multiplications required in the direct application of equation (1).

Because the parameter u is constant for layer il, the corresponding curve on the quadratic Bezier triangle is a quadratic curve. Therefore, the vertices $\{\boldsymbol{x}(il, ic)\}$ on layer il can be computed by means of second-order forward differences in the ic direction:

$$\boldsymbol{x}(il, ic) = \boldsymbol{x}(il, ic - 1) + \boldsymbol{dicx}(il, ic - 1) \tag{6}$$

$$\boldsymbol{dicx}(il, ic) = \boldsymbol{dicx}(il, ic - 1) + \boldsymbol{d2icx} \tag{7}$$

$$\boldsymbol{d2icx} = 2(\boldsymbol{x}_{020} - 2\boldsymbol{x}_{011} + \boldsymbol{x}_{002})/nl^2 \tag{8}$$

The second-order forward difference is independent not only of ic (as expected) but also of il.

The initial values of these difference equations in ic are quadratic ($\boldsymbol{x}$) and linear ($\boldsymbol{dicx}$) polynomials in il respectively. Therefore, these initial values can be computed by means of second- (first-) order forward differences in the il direction. The initial values of the difference equations in il depend only on the control vertices and the number of layers nl. Therefore, they can be computed during initialization.

Some remarks:

1. Except for the initialization, the resulting algorithm consists of additions only.
2. Speed-up (at the expense of more adders and multipliers) can be realized because all computations can be done in parallel for all the elements of the vectors.
3. Applying second-order differences results in a cumulative error. Experiments have shown that this cumulative error is less than 10^{-6} for 32 layers, using single float precision on an Onyx (tm). The consequences for a fixed-precision implementation have not yet been investigated.

3.3 Variable subdivision

FFS is a very fast, but also very rigid method for subdividing quadratic Bezier triangles. Each Bezier triangle of an object is subdivided into the same way in the same number of triangles. It is possible to adjust this number of triangles on the basis of an object's screen size. This would lead to good results if all the Bezier triangles in the object had the same size and curvature after view projection. However, the Bezier triangles that are smaller or flatter than average are covered by more triangles than needed; the Bezier triangles that are larger or more curved than average are covered by fewer triangles than needed. Therefore, methods for variable subdivision have been devised and investigated.

The main problem with variable subdivision is the possibility of cracks. Cracks at the borders can be avoided if, and only if, the number of segments for a border depends only on this border, and not on any other property of the quadratic Bezier triangle involved. Given a Bezier triangle, the first step is to compute the number of segments for each border (which will be called border number). This will be discussed in further detail in section 3.3.6.

The next step is to select a number of layers for the quadratic Bezier triangle on the basis of these border numbers. Each layer is then subdivided into a number of segments. The resulting grid is used to generate triangles for the inner area of the Bezier triangle. This inner area consists of those grid points which are not on the outer borders. Finally, the gap between the borders and the inner area is filled by triangles between the varying number of vertices on the outer borders and the vertices on the inner area borders (this will be discussed further in section 3.3.4).

3.3.1 The requirements

The selection of the number of layers, the number of segments for each layer and the gap filling method should be based on the following requirements:

1. The quality of the rendered image of the quadratic Bezier triangle should at least be the same as the image generated by means of fixed subdivision.

   This requirement can be met by using the maximum of the three border numbers as the number of layers. If a smaller number than this maximum is used, the quality of the image may be insufficient in the neighborhood of the border(s) with more segments. Indeed, using a smaller number may lead to anomalies.

2. It should be possible to compute the vertices by means of forward differences.

   Direct computation of the vertices requires 5 floating point additions and 12 floating point multiplications per coordinate. Forward differences requires 2 floating point additions per coordinate and a small number of floating point operations for initialization.

This requirement can be met by using either the layer number as the number of segments per layer (resulting in the same grid as for FFS; but in the case of variable subdivision only the inner points are used) or a constant number of segments per layer (follows from the formula for the first and second-order forward differences). The first case gives the linear[2] maximum variable subdivision (abbreviated to LMVS below)). The second case gives the constant maximum variable subdivision (abbreviated to CMVS below).

3. The number of generated triangles should be minimal (setup for rasterization of a triangle requires about 15 floating point operations per coordinate [15].

   This requirement determines the selection between FFS, LMVS and CMVS. If the border numbers are all equal, FFS is applied because this method is faster and more economical than LMVS and CMVS. If the border numbers are not equal, the selection between LMVS and CMVS is based on the number of generated triangles to avoid costly setups. The number of planar triangles required for LMVS and CMVS respectively can be estimated on the basis of the border numbers during run-time (see appendix A). So, the selection between FFS, LMVS and CMVS can be done by rendering system itself.

   If CMVS is chosen, the number of segments per layer (and thus the number of triangles per layer) should be at least equal to the number of segments of the last layer (the bottom border number). Otherwise, the quality of the image may be insufficient in the neighborhood of this border. Given this relation between the number of segments per layer and the bottom border number, the bottom border number should be minimal. Otherwise more planar triangles will be generated than needed. Therefore, if the bottom border number is greater than the two other border numbers, a cyclic permutation should be applied to the labels of the control vertices of the quadratic Bezier triangle so that the control vertices of the border with the lowest border number become x_{020} and x_{002}. To facilitate automatic choosing between LMVS and CMVS, this cyclic permutation is always applied if the bottom border number is not the minimum border number.

3.3.2 LMVS

LMVS uses the maximum of the three border numbers as the number of layers and FFS for the inner area. An example is given in Figure 3. The number of segments for the right border is 8. The number of segments for the left and bottom borders is 3. The inner area is indicated by the bold solid lines. The thin lines represent the FFS grid (the bold solid lines of the inner area hide most of the thin lines of the FFS grid). The grid for the vertices on the borders (the filled dots on the bold border lines in Figure 3) may differ from this FFS grid.

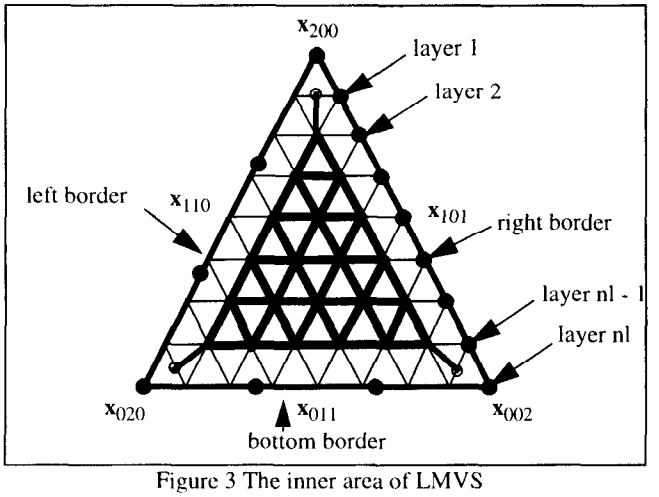

Figure 3 The inner area of LMVS

When the inner area has been triangulated, the gaps between the outer borders and the inner area borders have to be filled (see section 3.3.4) with triangles between the outer border vertices and the vertices of the corresponding border of the inner area (see section 3.3.5 for an exception). The distances between the corners of the quadratic Bezier triangle and the corresponding inner area corners are rather large in relation to the distances between the outer borders and the inner area borders. This may result in slender triangles amidst normal triangles. Therefore, three vertices halfway between the corner vertices of the Bezier triangle and the corresponding corner vertices of the inner area are computed and included in the inner area borders for gap filling. This is indicated by the open dots at the end of the three dangling edges in Figure 3.

If the number of layers is less than four, the inner area is empty. In this case the quadratic Bezier triangle is subdivided by an inside point (for example the centre point of the Bezier triangle) into three triangles. These triangles are filled by means of triangle fans between the vertices of the borders and this inside point.

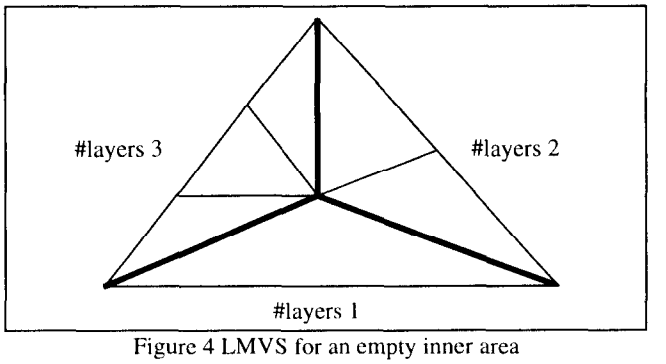

Figure 4 LMVS for an empty inner area

3.3.3 CMVS

CMVS generates for each layer, except the lowest, a constant number of segments. This constant number of segments per layer is equal to the number of segments of the bottom border with a minimum of three. Indeed, there should be at least one segment for the inner area, one for the left gap and one for the right gap. An example is given in Figure 5. There are six layers in the long vertical direction. Each layer has been subdivided into four segments, the two inner segments of which are used for the inner area. This inner

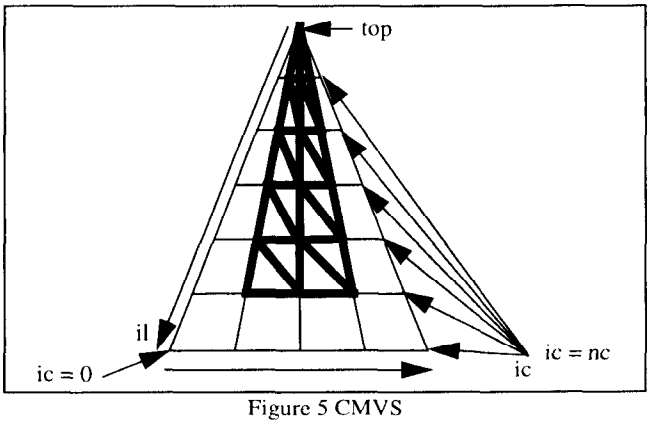

Figure 5 CMVS

[2]Called linear because in this method, as in fixed subdivision, each layer has one grid point more than the previous layer.

area is triangulated as illustrated in Figure 5 (indicated by bold lines). The triangles between the top of the quadratic Bezier triangle and the two inner segments of the first layer are included in the generation of the inner area. The gaps between the three outer borders and the extended inner area are filled in the same way as for LMVS (see section 3.3.4).

The parameter values at the grid points are:

$$u(il, ic) = (nl - il)/nl \tag{9}$$

$$v(il, ic) = (il/nl)((nc - ic)/nc) \tag{10}$$

$$w(il, ic) = (il/nl)(ic/nc) \tag{11}$$

The vertices along the layers follow from substitution of equations (9), (10) and (11) in equation (1). Because the parameter u is constant for layer il, the corresponding curve on the quadratic Bezier triangle is a quadratic curve. Therefore, the vertices $\{\boldsymbol{x}(il, ic)\}$ on layer il can be computed by means of second-order forward differences in the ic direction:

$$\boldsymbol{x}(il, ic) = \boldsymbol{x}(il, ic - 1) + \boldsymbol{dicx}(il, ic - 1) \tag{12}$$

$$\boldsymbol{dicx}(il, ic) = \boldsymbol{dicx}(il, ic - 1) + \boldsymbol{d2icx}(il) \tag{13}$$

$$\boldsymbol{d2icx}(il) = 2il^2(\boldsymbol{x}_{020} - 2\boldsymbol{x}_{011} + \boldsymbol{x}_{002})/(nc^2nl^2) \tag{14}$$

As expected, the second-order forward difference is independent of ic. Unlike with FFS, the second-order forward difference depends on il.

The initial values of these difference equations in ic are quadratic polynomials in il. Therefore, these initial values can be computed by means of second-order forward differences in the il direction. The initial values of the difference equations in il depend only on the control vertices, the number of layers nl and the number of segments per layer nc. Therefore, they can be computed during initialization.

3.3.4 Gap filling

The gaps between the outer borders and the inner area borders are filled by triangles between the outer border vertices and the vertices of the corresponding border of the inner area. Three heuristic methods are devised, implemented and tested. Gap filling method 1 (see Figure 6) connects the vertices of the smaller (in number of vertices) border with the centre vertices of the larger border (either the outer border or the inner area border) by means of a triangle strip and connects the outer vertices of the larger border with the first and last vertex, respectively, of the smaller border by means of triangle fans.

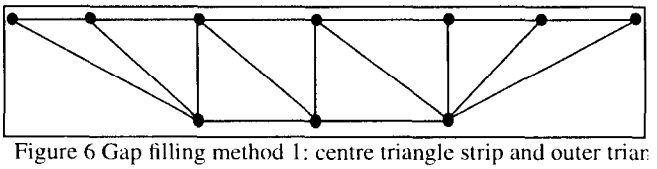

Figure 6 Gap filling method 1: centre triangle strip and outer trian

Gap filling method 2 (see Figure 7) generates one centre triangle fan and two triangle strips at both sides. If the centre triangle fan is

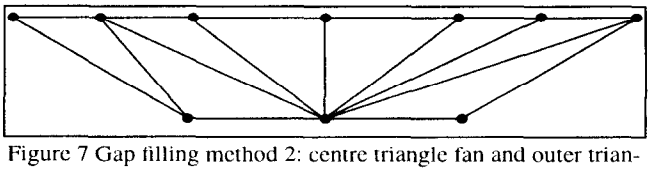

Figure 7 Gap filling method 2: centre triangle fan and outer trian-

empty (both borders have the same number of vertices), a single triangle strip is generated instead of two triangle strips. Method 2 yields a better image if the number and/or the position of vertices differ considerably.

Gap filling method 3 (see Figure 8) replaces the two triangles of the quadrangles of the triangle strip(s) by a triangle fan. Method 3

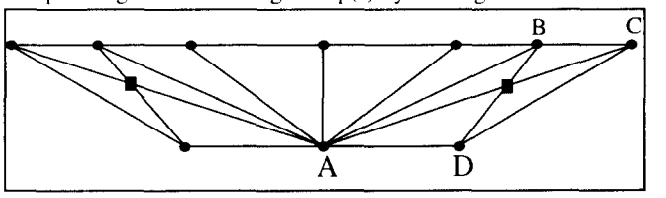

Figure 8 Gap filling method 3: additional vertices

yields a better image in the case of sharp highlights. If for example the two points B and D of the short diagonal of quadrangle ABCD of Figure 7 are highlighted and the two points A and C of the long diagonal are not, and if this quadrangle is only triangulated by the long diagonal, this diagonal will not have highlights while the other edges of the two triangles will be partly highlighted. In this case a gap will show a series of dark spots in the highlighted area. The additional vertex solves this problem because this vertex will also be highlighted. Instead of the intersection point of the two diagonals the centre point of the four vertices is computed. This is faster and better if the quadrangle is degenerated to a triangle.

Remark:

The vertices for subdivision of the outer borders can also be computed by means of second-order forward differences.

3.3.5 Optimization

Gap filling can be (partly) avoided, always in the case of CMVS and sometimes in the case of LMVS. The bottom gap can be avoided in the case of CMVS if the number of segments of the bottom border is greater than two. An example is given in the left picture of Figure 9. The bottom layer is processed in the same way as

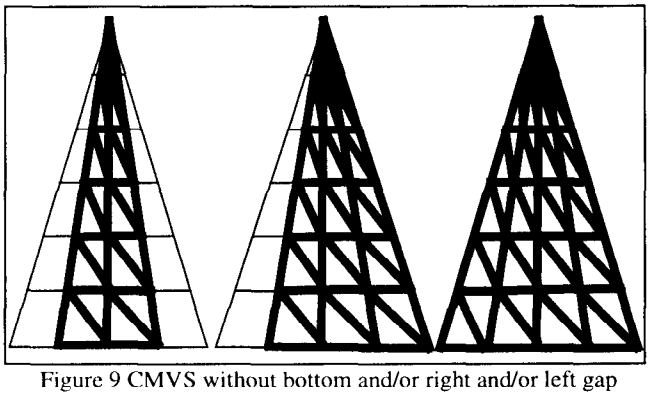

Figure 9 CMVS without bottom and/or right and/or left gap

the other layers. In this case the bottom border grid coincides with the CMVS grid for the lowest layer.

The left (right) gap can be avoided in the case of CMVS if the left (right) border number is equal to the number of layers. Because the number of layers is equal to the maximum border number, and because the bottom border number is equal to the minimum border number, there is at least one gap which can be avoided. An example is given in the middle picture of Figure 9. The right gap is processed in the same way as the inner area. In this case the right border grid coincides with the CMVS grid on the right border.

In the case of CMVS filling all the gaps can be avoided if both the left and the right border numbers are equal and the bottom border number is greater than three. An example is given in the right picture of Figure 9. The left, bottom and right gaps are processed in

the same way as the inner area. In this case all the border grids coincide with the CMVS grid on the borders.

In the case of LMVS filling the left and right gap can be avoided if both the left and right border numbers are equal (and thus equal to the number of layers). An example is given in Figure 10. The left

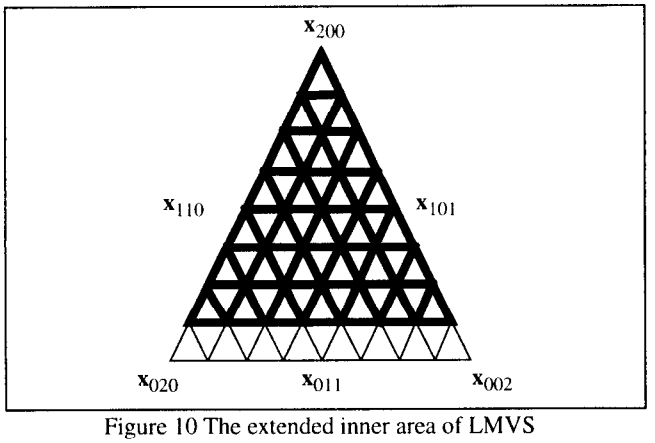

Figure 10 The extended inner area of LMVS

and right gaps are processed in the same way as the inner area. The triangulation of the bottom gap is not shown. In this case the left and right border grids coincide with the FFS grid on these borders. Moreover, the end points at the dangling edges (see section 3.3.2 and/or Figure 3) are not needed.

3.3.6 Subdivision criteria for borders

As already explained in section 3.3, cracks at the borders can be avoided if, and only if, these borders are always subdivided into the same number of segments. If quadratic Bezier triangles have to be rendered independent of each other, the border numbers have to be computed only on the basis of the properties (the vertices) of the border itself, and not on the basis of any other property of the Bezier triangle involved.

The selection of the subdivision criteria is based on the following requirements:

1. The criteria should be based on screen properties to get viewpoint dependent tessellation.
2. The user should have simple intuitive control of the resulting subdivision.
3. The required computations should be as simple (to be able to include them on chip in the future) and as fast (they have to be repeated twice for each frame for each border) as possible.

These requirements resulted in two user-adjustable size controls (see Figure 11):

1. The maximum number of pixels per border segment (abbreviated to MAXLEN below).
2. The maximum number of pixels per difference vector of two succeeding border segments (abbreviated to MAXD2S below).

MAXLEN makes it possible to subdivide straight borders to obtain better highlights, if lighting calculations are applied to the vertices of the generated triangles. MAXD2S gives the opportunity to control the approximation of curved borders. So, applying these criteria for the tessellation of quadratic Bezier triangles gives the opportunity to avoid the polyline like silhouette curves.

Precisely computing the minimum number of segments for a given MAXLEN requires an iterative process. The border has to be rendered for an increasing number of segments until the number of pixels for each segment is less than or equal to MAXLEN. Therefore, the required number of segments is computed by dividing an estimated border length by MAXLEN. This border length is computed by adding the lengths of the line segments of the control polygon (chord approximation). This approximation is exact for straight borders.

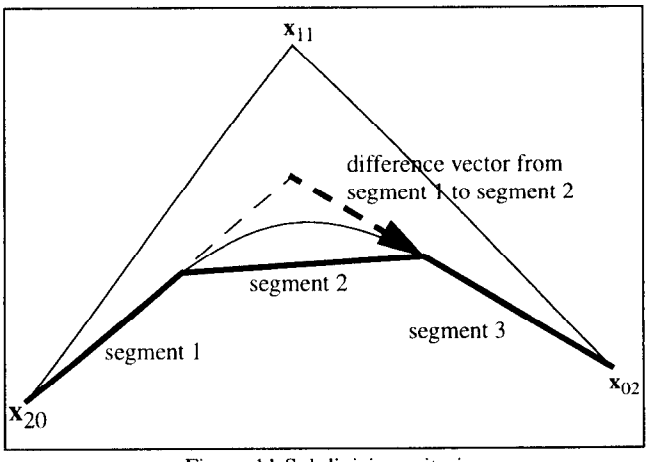

Figure 11 Subdivision criteria

The length of the difference vector is equal to length of the second-order forward difference of the three control vertices divided by the square of the number of segments (see equation (8)):

$$\|d2x\| = 2\|x_{20} - 2x_{11} + x_{02}\|/n^2 \leq \text{MAXD2S} \qquad (15)$$

Because a border curve of a quadratic Bezier triangle is a quadratic curve, this difference vector is constant over this border. Therefore, the number of segments is given by:

$$n = \sqrt{\frac{2\|x_{20} - 2x_{11} + x_{02}\|}{\text{MAXD2S}}} \qquad (16)$$

Because the maximum number of segments is bounded by a small number (typically 32), this square root can be computed by means of a look-up table (typically between 1 and 1024).

The resulting number of segments is the maximum of the number of segments needed to ensure that the first requirement is fulfilled, and of the number of segments needed to ensure that the second requirement is fulfilled.

Application of these size controls requires the computation of the length of the border and of the difference vector. The absolute ($L1$) norm, the Euclidean ($L2$) norm or the maximum ($L\infty$) norm can be used to compute the length. The maximum norm requires the least computations, the Euclidean norm the most computations. The maximum norm results in the smallest number of segments, the absolute norm in the largest number of segments.

4 EXPERIMENTS AND RESULTS

To test the image quality and savings obtainable with quadratic Bezier triangles, a number of Bezier triangle models were generated. Five curved surfaces (discussed further in section 4.1) were used. These curved surfaces were approximated in model space by quadratic Bezier triangles (not to be mixed up with viewpoint dependent tessellation as described in [1, 7, 13]). Notice that this approximation gives not only the position but also the normal and the texture coordinates of the control vertices of the quadratic Bezier triangles. The number of Bezier triangles was chosen by means of visual inspection (the goal was not to investigate optimal approximation of curved surfaces but to generate planar and Bezier triangle models of the same objects with the same method to com-

pare the number of generated triangles). This number was increased until the resulting image no longer changed. Fixed subdivision was applied to generate this image. The number of layers was chosen so that a higher number of layers did not change this image. Notice that generation of quadratic Bezier triangle models is supposed to be part of the authoring software.

The quadratic Bezier triangle models were rendered for various viewing parameters (discussed further in section 4.2) and various variable subdivision parameter combinations (discussed further in section 4.3). Notice that variable subdivision gives the opportunity to tessellate a quadratic Bezier triangle on the basis of its screen properties (see section 3.3.6).

4.1 Geometry

The following five curved surfaces were used:

1. Sphere (abbreviated to sp).

   The centre was at (0, 0, 0). The radius was 1. First, each hemisphere was approximated by four Bezier triangles. Next, each Bezier triangle was three times recursively subdivided into four Bezier triangles (the middle edge control vertices became corner control vertices). This resulted in a total of 512 quadratic Bezier triangles. The corner control vertices of all the Bezier triangles lay on the sphere surface. The middle edge control vertices were chosen so that the corresponding surface points lay on the sphere surface.

2. Rational cylinder (abbreviated to rc).

   The rational cylinder consisted of four bicubic Bezier parts. The x y coordinates of a part were defined by a rational Bezier curve of degree 3. This curve was derived from a quadratic rational Bezier curve by means of degree elevation [3]. This curve was an exact representation of a quarter of a circle. These circle parts were swept in the z direction to generate a cylinder surface. The rational cylinder was approximated by 88 quadratic Bezier triangles.

3. OpenGl Bezier surface (abbreviated to bz).

   The OpenGl Bezier surface was the example on page 336 of [11]. It was a single quadratic Bezier triangle. The OpenGl Bezier surface was approximated by 29 Bezier triangles.

4. OpenGl NURBS surface (abbreviated to nu).

   The OpenGl NURBS surface was the example on page 344 of [11]. It was a simple NURBS patch. The OpenGl NURBS surface was approximated by 68 quadratic Bezier triangles.

5. Utah teapot (abbreviated to tp).

   The Utah teapot consisted of 6 separate parts (data not included). Each part consisted of a number of connected quadrangular Bezier patches. The total number of quadrangular Bezier patches was 32. The teapot was approximated by 442 quadratic Bezier triangles.

The NURBS surfaces were approximated by quadratic Bezier triangles by means of a recursive subdivision algorithm. The subdivision stopped when the edge curves of the Bezier triangles were within a user-controllable tolerance of the NURBS surface. Common edges of the Bezier patches were approximated by a set of connected common edges of quadratic Bezier triangles to avoid cracks in the Bezier triangle model. The tolerance was set so that a smaller tolerance did not change the resulting image generated by fixed subdivision. Notice that approximation of NURBS by means of quadratic Bezier triangles is not the issue of this paper but using quadratic Bezier triangles as drawing primitives.

4.2 Viewing parameters

Two sets of viewing parameters were used. The first set of viewing parameters was used to compare fixed and variable subdivision with regard to the attainable image quality and the number of generated planar triangles (see section 4.1 for the object abbreviations):

object	fov	azimuth	elevation
sp	40	0	90
rc	40	0	90
bz	22	90	0
nu	50	0	40
tp	44	0	40

Table 1 First set of viewing parameters

The chosen fovs (field of views in degrees) yielded almost maximum coverage of the drawing area. The chosen azimuths and elevations (also in degrees) resulted in a good view of the structure of the objects.

The second set of viewing parameters was used to show the effect of variable subdivision on the triangulation for different screen sizes. For each object the fov was varied from 50 to 150 in steps of 25. The azimuth and elevation were the same as for the first set of viewing parameters.

4.3 Variable subdivision parameters

The variable subdivision parameters were the subdivision criteria for the borders (see section 3.3.6) and the gap filling method (see section 3.3.4). The parameter combinations used are:

Combi	MAXLEN	MAXD2S	Gap filling method
1	1280	2	3
2	1280	2	2
3	1280	3	3
4	1280	3	2

Table 2 Parameter combinations used for variable subdivision

4.4 Number of triangles

The number of quadratic Bezier triangles and the number of generated planar triangles are shown in table 3. The column labelled "Bezier" contains the number of quadratic Bezier triangles. The column labelled "FFS" contains the number of triangles generated by means of FFS. The column labelled "FFS view" contains the number of triangles generated by means of variable subdivision for the first set of viewing parameters given in table 1 (the same set of viewing parameters as used for FFS). The first row for an object contains the number of triangles generated for combi 1, the second row for combi 2 and so on (see Table 2).

4.5 Pictures

Figure 13 (see appendix B) contains a wire-frame picture of a sphere approximated by 512 quadratic Bezier triangles rendered by means of fixed subdivision with 4 layers per quadratic Bezier triangle. Figure 14 and 15 contain wire-frame pictures of the same Bezier triangle model rendered by means of variable subdivision. Notice that the number of planar triangles per Bezier triangles

object	Bezier	FFS	Variable subdivision					
			FFS view	050	075	100	125	150
sp	512	8192	10248	6504	3104	1824	1112	560
			6672	4880	2816	1824	1112	560
			4192	4072	1880	1248	672	512
			3712	3328	1880	1248	672	512
rc	88	1408	2252	1756	1316	984	352	244
			2016	1580	1188	880	352	244
			1632	1368	984	408	296	224
			1488	1240	880	408	296	224
bz	29	464	963	470	209	161	102	38
			739	364	189	145	96	38
			660	292	167	102	70	36
			522	242	151	96	70	36
nu	68	1088	3568	3568	2350	1608	910	406
			2636	2636	1726	1202	690	376
			2442	2442	1648	972	478	244
			1816	1816	1238	732	432	244
tp	442	28288	18932	16630	10660	6772	4280	2132
			14600	12882	8222	5422	3520	1878
			12688	11327	7170	4386	2592	1354
			9978	8811	5648	3650	2258	1254

Table 3 The number of quadratic Bezier triangles and the number of planar triangles generated

increases when the borders are more curved. Note also, that the number of planar triangles can be controlled by means of the criterion MAXD2S (see section 3.3.6) and the gap filling method (see section 3.3.4).

Figure 16, 17 and 18 contain shaded pictures of the Utah teapot. The lighting calculation (no high lights) was only applied to the vertices of the quadratic Bezier triangle. The intensity of the vertices of the generated planar triangles was computed in the same way as their position. Notice that variable subdivision give pictures as least as good as fixed subdivision. Figure 19 and 20 contain two shaded pictures with high lights. Notice that the high lights are better in the semi-Phong picture (lighting calculation for each vertex of the generated planar triangles; in this case the normals of the generated vertices are computed in the same way as their positions). Figure 21, 22, 23 and 24 contain shaded pictures for decreasing field of view. Notice that the smaller pictures are acceptable although the number of generated planar triangles decreases (see Table 3).

5 CONCLUSIONS

The following conclusions can be drawn from the results, the pictures and the experiences gathered during the creation and rendering of the approximations:

1. It is possible to generate quality images of curved surfaces on the basis of approximation by means of quadratic Bezier triangles (see figure 16 and following figures).
2. The advantages of using quadratic Bezier triangles instead of planar triangles for curved surfaces are:
   (a) Much less model data (much less storage and/or transmission).
   (b) Much fewer vertices for the geometry transformations.
   (c) One can interchange the efficiency of the illumination calculations for the quality by either doing illumination calculations on the vertices of the Bezier triangles or on the vertices of the generated planar triangles. (see figures 19 and 20, respectively).
3. When variable subdivision is compared with fixed subdivision, variable subdivision yields the same or better quality images, especially at the silhouette curve (compare figure 16 with figures 17 and 18).
4. The criterion for estimating the required number of segments is very suitable for decreasing the level-of-detail in favor of higher frame rates.
5. Variable subdivision automatically generates fewer triangles for smaller (in screen space) objects while the image quality remains acceptable (see table 3 and figure 21 and following figures).
6. Variable subdivision gives the opportunity to adjust the number of triangles to the curvature of the borders (see figures 14 and 15).
7. Fixed and variable subdivision can be easily incorporated in render systems currently used for planar triangles. The subdivision should take place after geometry transformation and before rasterization. The position with regard to the lighting calculations will depend on the quality required for highlights. Furthermore, the forward difference calculations are very similar to the interpolation calculations of the rasterizer.

   Notice that the described algorithms cannot directly be implemented in hardware. Additional work is necessary for a hardware implementation:
   (a) What should and can be done in parallel?
   (b) What are the required number of bits?
   (c) How much intermediate storage is required?
   (d) How are compute intensive functions (like square root) approximated?

REFERENCES

1. Salim S. Abi-Ezzi and Srikanth Subramaniam "Fast Dynamic Tessellation of Trimmed NURBS Surfaces". Eurographics'94 Conference Proceedings, Vol. 13, No. 3, pp. C-107-C-126,

September 1994.

2. James F. Blinn "A scan line algorithm for displaying parametrically defined surfaces". Computer Graphics (SIGGRAPH'78 Conference Proceedings), Vol. 12, No. 3, August 1978, pp. 27-27.
3. Gerald Farin "Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide". Academic Press, Inc., 1988.
4. David R. Forsey and R. Victor Klassen "An Adaptive Subdivision Algorithm for Crack Prevention in the Display of Parametric Surfaces". Proceedings of Graphics Interface '90, Halifax, Nova Scotia, May 1990.
5. Sylvain Karpf, Christophe Chaillou, Eric Nyiri, Michel Meriaux "Real-time Display of Quadric Objects in the I.M.O.G.E.N.E. Machine". ACM Symposium on Solid Modelling, Austin, 1991.
6. G. Kedem and J.L. Ellis "The Raycasting Machine". Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers ICCD'84, pp. 533-538, October 1984.
7. Subodh Kumar, Dinesh Manocha and Anselmo Lastra "Interactive Display of Large NURBS Models". IEEE Transactions on Visualization and Computer Graphics, Vol. 2, No.4, pp. 323-336, December 1996.
8. H. Laporte, E. Nyiri, M. Froumentin and C. Chaillou "A graphics system based on quadrics". Computers and Graphics, Vol. 19, No. 2, March/April 1995, pp. 251-260.
9. Sheue-Ling Lien, Michael Shantz and Vaughan Pratt "Adaptive Forward Differencing for Rendering Curves and Surfaces". Computer Graphics (SIGGRAPH'87 Conference Proceedings), Vol. 21, No. 4, July 1987, pp. 111-118.
10. Sheue-Ling Lien, Michael Shantz and Robert Rocchetti "Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing". Computer Graphics (SIGGRAPH'89 Conference Proceedings), Vol. 23, No. 3, July 1989, pp. 157-166.
11. Jackie Neider, Tom Davis, Mason Woo "OpenGL Programming Guide. The Official Guide to Learning OpenGl, Release 1". SiliconGraphics Computer Systems, 1993.
12. Jorg Peters "Evaluation and Approximate Evaluation of the Multivariate Bernstein-Bezier Form on a Regularly Partioned Simplex". ACM Transactions on Mathematical Software, Vol. 20, No. 4, Pages 460-480, December 1994.
13. Alyn Rockwood, Kurt Heaton and Tom Davis "Real-Time Rendering of Trimmed Surfaces". Computer Graphics, Vol. 23, No. 3, pp. 107-116, July 1989.
14. Michael Shantz and Sheue-Ling Lien "Shading Bicubic Patches". Computer Graphics (SIGGRAPH'87 Conference Proceedings), Vol. 21, No. 4, July 1987, pp. 189-196.
15. Gerrit A. Slavenburg, Selliah Rathnam, and Henk Dijkstra "The Trimedia TM-1 PCI VLIW Mediaprocessor". Hot Chips 8, Stanford California, August 19-20, 1996.
16. Turner Whitted "A scan line algorithm for computer display of curved surfaces". Computer Graphics (SIGGRAPH'78 Conference Proceedings), Vol. 12, No. 3, August 1978, pp. 26.

A The number of generated triangles

As already explained in section 3.3.6, the number of generated triangles is used to choose between LMVS and CMVS. The number of generated triangles is the sum of the number of generated triangles for the inner area, for the bottom gap, for the left gap and for the right gap.

A.1 LMVS

The number of generated triangles for variable subdivision depends on the relation between the border numbers (nl_L is left border number, nl_R is right border number and nl_B is bottom border number) because of the optimization described in section 3.3.5. This results in the following cases for LMVS (always the number of layers $nl = max(nl_L, nl_R)$ and $nl_B \le min(nl_L, nl_R)$):

1. $nl_L = nl_R$:

$$nt = nl(nl-1) + 3nl_B \tag{17}$$

2. $nl_R < nl$:

$$nt = nl^2 + 4 + 3(nl_B + nl_R) \tag{18}$$

3. $nl_L < nl$:

$$nt = nl^2 + 4 + 3(nl_B + nl_L) \tag{19}$$

A.2 CMVS

For CMVS the following cases have to be distinguished:

1. $nl_B < 3$ and $nl_L = nl_R$:

$$nt = 6nl + 3nl_B - 6 \tag{20}$$

2. $nl_B < 3$ and $nl_R < nl$:

$$nt = 5nl + 3(nl_B + nl_R) - 4 \tag{21}$$

3. $nl_B < 3$ and $nl_L < nl$:

$$nt = 5nl + 3(nl_B + nl_L) - 4 \tag{22}$$

4. $nl_B \ge 3$ and $nl_L = nl_R$:

$$nt = nl_B \times (2nl - 1) \tag{23}$$

5. $nl_B \ge 3$ and $nl_R < nl$:

$$nt = nl_B(2nl - 1) - nl + 3nl_R + 1 \tag{24}$$

6. $nl_B \ge 3$ and $nl_L < nl$:

$$nt = nl_B(2nl - 1) - nl + 3nl_L + 1 \tag{25}$$

A.3 Selection between LMVS and CMVS.

A simple way to compare LMVS with CMVS is to compute the difference between the number of generated triangles: $dlcnt$. Given the various conditions the following cases can be distinguished:

1. $nl_B < 3$ and $nl_L = nl_R$:

$$dlcnt = nl^2 - 7nl + 6 \tag{26}$$

2. $nl_B < 3$ and $nl_R < nl$ or $nl_L < nl$:

$$dlcnt = nl^2 - 5nl + 8 \tag{27}$$

3. $nl_B \ge 3$ and $nl_L = nl_R$:

$$dlcnt = nl^2 - nl(2nl_B + 1) + 4nl_B \tag{28}$$

4. $nl_B \ge 3$ and $nl_R < nl$ or $nl_L < nl$:

$$dlcnt = nl^2 - nl(2nl_B - 1) + 4nl_B + 3 \tag{29}$$

The selection criterion is now:

If $dlcnt < 0$ LMVS else CMVS. (30)

Strict application of this criterion requires a lot of case distinctions and a lot of computations. Fortunately, case 1 gives only a small negative $dlcnt$ for nl equal to 4 and 5. Case 2 gives always a positive $dlcnt$. Therefore, case 1and 2 can be combined:

If $nl_B < 3$ CMVS. (31)

For case 3 and 4, the selection criterion (30) results in

If $2nl_B > nl(nl-1)/(nl-2)$ LMVS else CMVS. (32)

respectively

If $2nl_B > (nl^2 + nl + 3)/(nl-2)$ LMVS else CMVS. (33)

To avoid the computations of (32) and (33) both criteria are be approximated by

If $nl < 2nl_B - 1$ LMVS else CMVS. (34)

Criterion (31) together with criterion (34) give a simple selection test which results in a minimum number of generated triangles in almost all cases. Besides, if application of these criteria results in selection of the wrong subdivision method, the resulting picture is still correct and the number of generated triangles is not much more.

There is one exception for the application of these criteria. CMVS should not be used if the halfway edge (the halfway edge is the edge between the middle edge control vertices of the left and right border) is greater than the bottom border. An example is given in Figure 12. Use of CMVS leads in this case to poor approximations

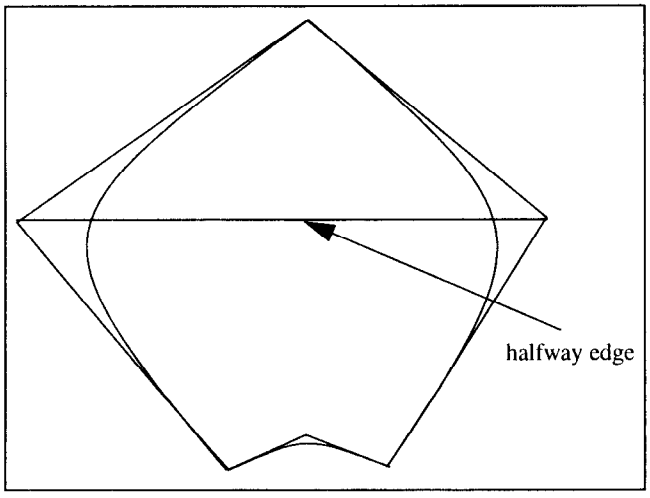

Figure 12 A greater halfway edge

in the middle area.

The halfway edge is in the same way as the borders transformed to a number of segments (see section 3.3.6). But, because only two vertices are available, the difference vector is not used to compute the number of segments.

B Pictures

Figure 13 Fixed subdivision

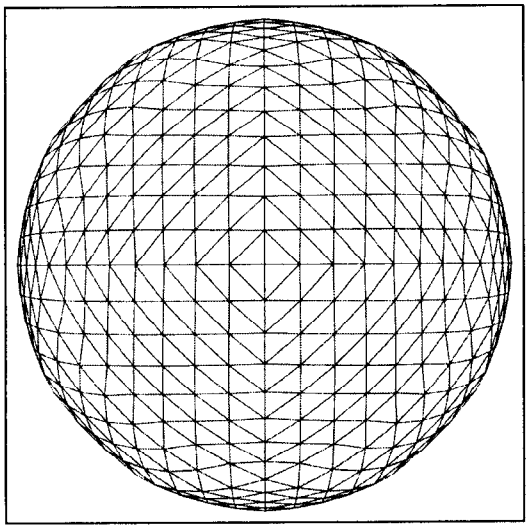

Figure 14 Variable subdivision (combi 1)

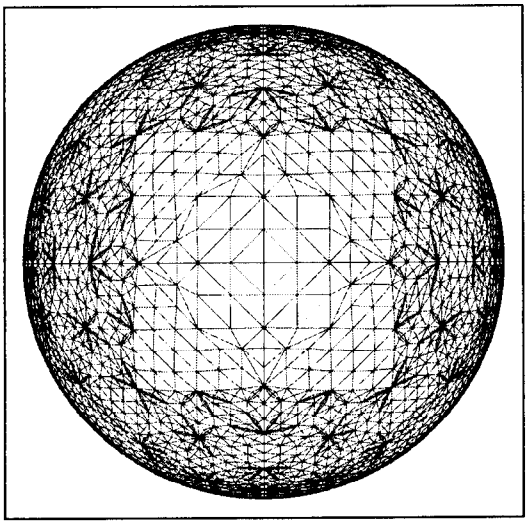

Figure 15 Variable subdivision (combi 4)