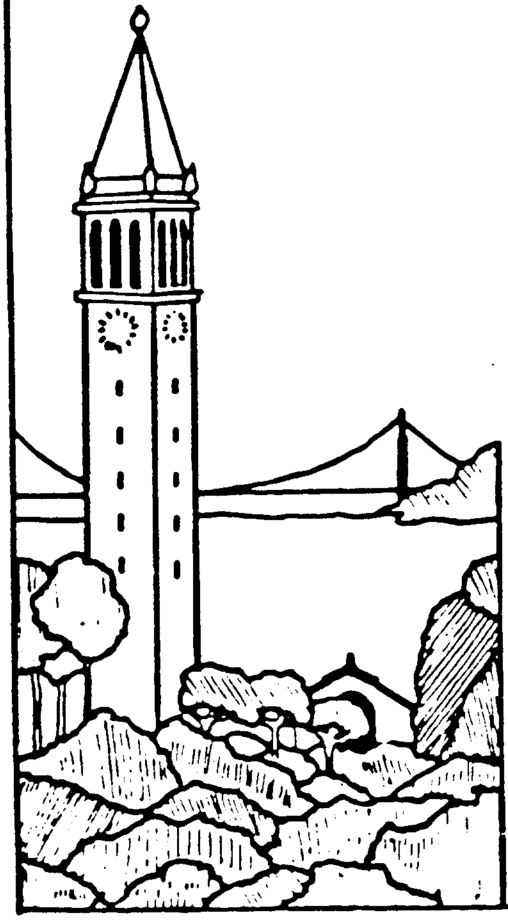
# Symmetric Interpolation of Triangular and Quadrilateral Patches Between Cubic Boundaries

*Leon A. Shirman*



Report No. UCB/CSD 87/319

December 1986

Computer Science Division (EECS)
University of California
Berkeley, California 94720

# SYMMETRIC INTERPOLATION OF TRIANGULAR AND QUADRILATERAL PATCHES BETWEEN CUBIC BOUNDARIES

*Leon A. Shirman*

*Master's Project Report*
*Under Direction of*
*Professor Carlo H. Séquin*

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
December 1986

*ABSTRACT*

In her Master's Project, Lucia Longhi [Longhi '85] has implemented an exploratory program, called *uci*, for interpolating triangular Gregory patches between cubic boundaries. This work is a continuation of the research in the area of smooth surface interpolation.

First, a symmetric version of the polygon tessellating program, *ugtess* [Gigus '85], has been developed. It splits faces of a polygon into bilaterally symmetric convex parts. Second, *uci* has been extended to include quadrilateral Gregory patches; this reduces the occurrence of asymmetric creases in the interpolating surfaces. Third, several new approaches are discussed to represent the surface with Bézier and Gregory patches and to determine the corresponding control vertices. Fourth, a method for subdivision of triangular Gregory patches into Bézier patches is described.

# 1. INTRODUCTION

The importance of representing surfaces in a computer has been widely recognized in Computer Aided Design and Manufacturing (CAD/CAM). There are basically two fields in Computer Aided Design: surface modeling and solids modeling. Surface modeling (sometimes also called free-form surface modeling) tries to represent curved surfaces that occur in the body of a car or in a human face. Various mathematical expressions for surface shape design have been proposed by Coons, Bézier, Gregory, and others ([Coons '64], [Barnhill '74], [Gregory '74]). In solid modeling, three-dimensional objects are represented as rigid solids. In such a system, complicated shapes are constructed from several primitive elements such as simple polyhedra, spheres, and cylinders. In this paper, we will be dealing with surface modeling only.

A widespread method for constructing smooth surfaces in CAD/CAM interpolates surface patches in such a way that neighbor patches meet smoothly. Smoothness can be defined in terms of differentiability, i.e. partial derivatives at boundary points must be identical. Alternatively, one can often be satisfied with *geometric continuity*, where only the shape of the adjoining patches is considered, but not their parametrization. A very important problem in designing a surface is to ensure smoothness along boundary curves.

There are various attempts to solve this problem. G. Farin [Farin '82] uses a triangular mesh to specify an object's topology, builds cubic curves between vertices, constructs quartic Bézier triangular patches, and then subdivides each patch into three subpatches and adjusts them to meet smoothly along their boundaries.

Another approach by H. Chiyokura and F. Kimura [Chiyokura et al '83] uses quadrilateral Gregory patches between cubic boundaries. In their approach, patch information is derived exclusively from boundary curves.

Farin's method uses Bézier patches, which are *polynomial.* Thus, many well-known algorithms for subdivision, intersections, etc. can be used. Chiyokura's method, on the contrary, uses *non-polynomial* Gregory patches. However, their method is *local,* i.e. a patch can be changed without the necessity to recompute other patches in such a way, that the geometric continuity between the patches will be preserved. This is very useful for interactive design. Farin's method does not have such flexibility.

In her Master's Project, L. Longhi [Longhi '85] designed an interactive system called *uci*. For a given polyhedral object, each face is first triangulated. Then normals are computed at each vertex as the average of the normals of all faces that meet at this vertex, weighted by the angle between the two edges by which a face is connected to that vertex. Cubic curves are then created by some heuristic method that gives pleasing results. The control points for triangular Gregory patches are constructed from the geometric continuity constraints across the boundaries. The patches are then evaluated at several points and triangular nets are constructed for visual representation. The program also makes it possible to construct straight edges, borders, and flat faces.

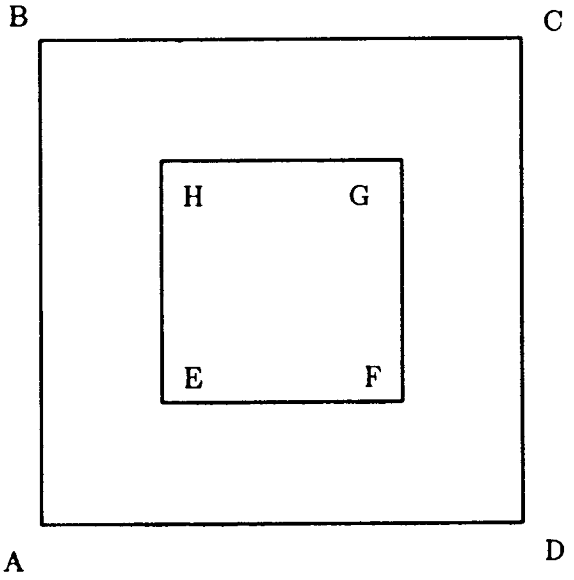


**Figure 1.1.** *A polygon with a hole*

One of the disadvantages of the above system is that it does not preserve the symmetry of an object because the triangulation of each face occurs in a rather arbitrary manner. For example, the face in Fig. 1.1 is decomposed into triangles in a manner, shown in Fig. 1.2. Clearly,

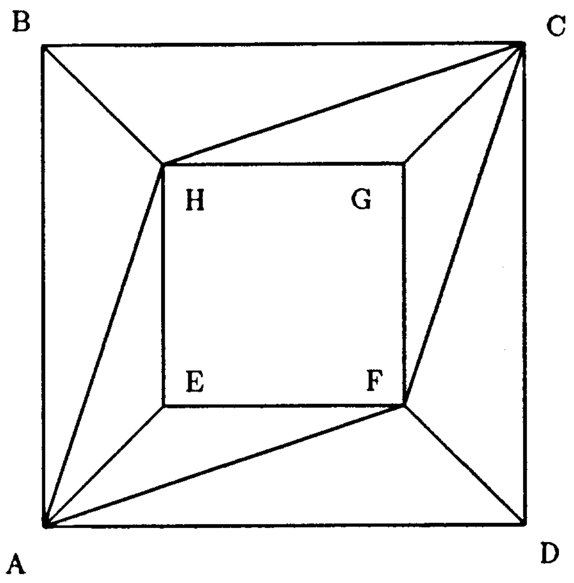


**Figure 1.2.** *The polygon of Fig. 1.1, decomposed with old* ugtess.

vertices $A$ and $C$ are different from $B$ and $D$, and they shouldn't be. As a result, undesired creases are produced in the final rendering of the object. To remedy this situation, we have designed a *symmetric version* of the program *ugtess*, which splits a polygon into symmetric bilateral convex parts (see Section 3). To further enhance the flexibility of the *uci* system, we have also introduced quadrilateral Gregory patches. In Section 4, we describe various methods of joining two Gregory patches with first order geometric continuity across their common border.

As mentioned above, it is often desirable to use Bézier patches rather than Gregory patches. In Section 5, we describe our efforts to subdivide Gregory patches into Bézier patches and thus combine the best of the two approaches by Farin and Chiyokura. Implementation aspects and new features of *uci* are described in Section 6 and in the Appendixes.

## 2. SOME BASIC MATHEMATICS

### 2.1. Bézier Patches

**Quadrilateral (Tensorproduct) Patches.** A biparametric vector-valued tensorproduct

Bézier patch of degree $(m,n)$, defined over the unit square, is given by:

$$b(u,v) = \sum_{j=0}^{n} \sum_{i=0}^{m} b_{i,j}\, B_i^m(u)\, B_j^n(v), \quad u,v \in [0,1].$$

where $B_j^l(t)$ are *Bernstein polynomials:*

$$B_j^l(t) = \frac{l!}{j!(l-j)!}\, t^j\, (1-t)^{l-j}, \quad j=0,\dots,l, \quad t \in [0,1].$$

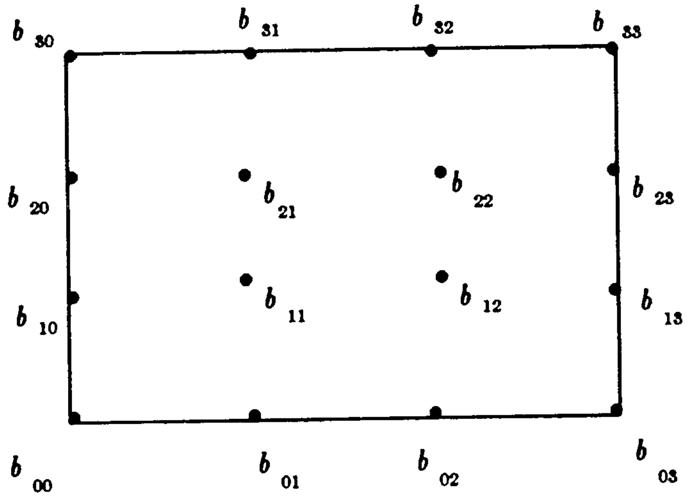


**Figure 2.1.** *A bicubic Bézier patch.*

An example of a bicubic patch $(m=n=3)$ is shown in Fig. 2.1. Thus, a Bézier patch is completely determined by the finite number of *control points* $b_{i,k}$.

**Triangular Patches.** Bézier patches can also be defined over triangles (Fig. 2.2). Let T be a triangle in space with vertices $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$. A point **P** in the plane of the triangle can be uniquely represented as follows:

$$\mathbf{P} = u\ \mathbf{T}_1 + v\ \mathbf{T}_2 + w\ \mathbf{T}_3\ , \quad u + v + w = 1.$$

**P** is said to have *barycentric coordinates* $(u,v,w)$ *with respect to* T. The interior of the triangle is characterized by the additional restrictions $u,v,w \geq 0$.

The following equation represents a Bézier patch of degree $n$ over a triangle:

$$b(u,v,w) = \sum_{\substack{i+j+k = n \\ i,j,k \geq 0}} b_{i,j,k}\, B_{i,j,k}^n(u,v,w), \quad u,v,w \geq 0, \quad u+v+w = 1.$$

Here

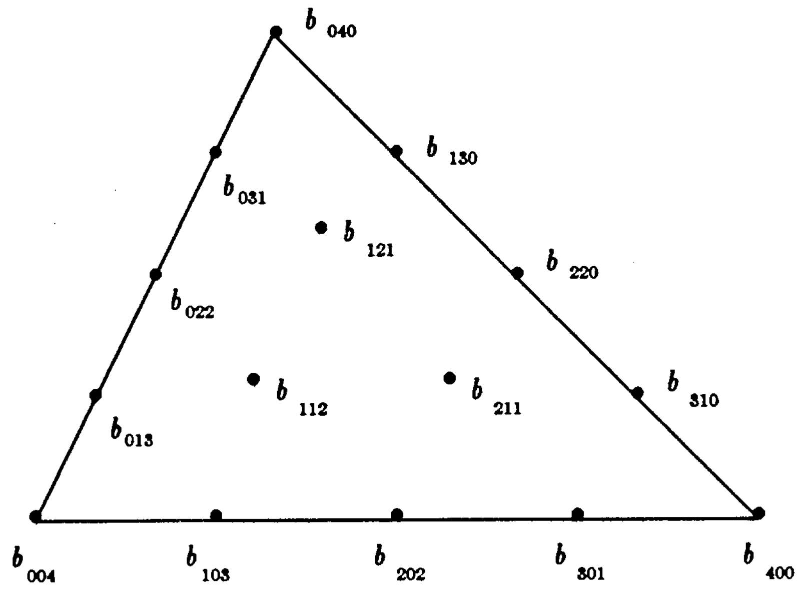


**Figure 2.2.** *A quartic triangular Bézier patch.*

$$B^n_{i,j,k}(u,v,w) = \frac{n!}{i!j!k!} u^i v^j w^k, \quad i+j+k = n, \; i,j,k \geq 0.$$

are *bivariate Berstein polynomials.* They are indeed bivariate, because one variable is dependent on the other two: $w = 1 - u - v$.

**Degree Elevation.** As one can see, equations of boundary curves are described by Berstein polynomials in one variable. Every polynomial (not necessarily univariate) of degree $n$ can be expressed in terms of Berstein basis polynomials of degree $n+1$:

$$\sum_{i=0}^{n} b_i \, B^n_i(t) = \sum_{i=0}^{n+1} b_i^* \, B^{n+1}_i(t).$$

New coefficients $b_i^*$ can be obtained as follows [Farin '82]:

$$b_i^* = \frac{n+1-i}{n+1} b_i + \frac{i}{n+1} b_{i-1}, \quad i = 0,...,n+1.$$

We assume $b_{-1} = b_{n+1} = 0$.

## 2.2. Gregory Patches

**Quadrilateral Patches.** A Gregory patch, like a Bézier patch, is defined by its *control* *points* $P_{i,j,k}$, $i,j = 0,...,n$, $k = 0,1$. In the case of a quadrilateral Gregory patch (Fig. 2.3), the equations are as follows [Chiyokura '86]:

$$S(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)\, B_j^3(v)\, Q_{i,j}(u,v), \quad u,v \in [0,1].$$

$$Q_{1,1}(u,v) = \frac{uP_{1,1,0} + vP_{1,1,1}}{u + v},$$

$$Q_{1,2}(u,v) = \frac{uP_{1,2,0} + (1-v)P_{1,2,1}}{u + 1-v},$$

$$Q_{2,1}(u,v) = \frac{(1-u)P_{2,1,0} + vP_{2,1,1}}{1-u + v},$$

$$Q_{2,2}(u,v) = \frac{(1-u)P_{2,2,0} + (1-v)P_{2,2,1}}{1-u + 1-v},$$

$$Q_{i,j}(u,v) = P_{i,j,0} = P_{i,j,1} \quad \text{otherwise.}$$

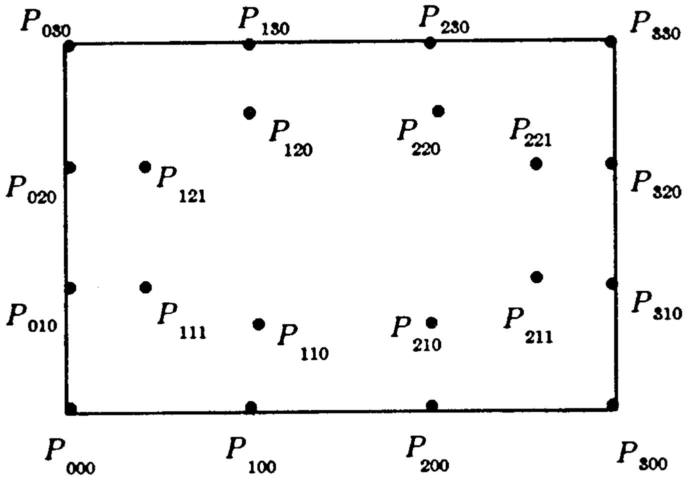


**Figure 2.3.** *A quadrilateral Gregory patch of degree 3.*

For conceptual elegance, a Gregory patch can also be expressed in equivalent recursive form [Chiyokura '83]:

$$S(u,v) = (1-u+uE)^3\, (1-v+vF)^3\, Q_{0,0}(u,v),$$

where $E$ and $F$ are *shift operators:*

$$E\, Q_{i,j} = Q_{i+1,j}, \quad F\, Q_{i,j} = Q_{i,j+1}$$

and $Q_{i,j}$ are as above.

When the eight inside control points satisfy the equations $P_{i,j,0} = P_{i,j,1}$, then the Gregory patch degenerates into a bicubic Be'zier patch. Thus, a Be'zier patch is a special case of a more general Gregory patch.

**Triangular Patches.** Working again in barycentric coordinates, a triangular Gregory patch (Fig. 2.4) of degree 4 is determined by 18 control points:

$$S(u,v,w) = \sum_{\substack{i+j+k=n \\ i,j,k \geq 0}} B^n_{i,j,k}(u,v,w)\, Q_{i,j,k}(u,v,w), \quad u,v,w \geq 0, \quad u+v+w = 1.$$

or

$$S(u,v,w) = (uE + vF + wG)^4\, Q_{0,0,0}(u,v,w), \quad u,v,w \geq 0, \quad u+v+w = 1.$$

where $E$, $F$, and $G$ are again *shift operators:*

$$E\, Q_{i,j,k} = Q_{i+1,j,k}, \quad F\, Q_{i,j,k} = Q_{i,j+1,k}, \quad G\, Q_{i,j,k} = Q_{i,j,k+1};$$

$$Q_{112}(u,v,w) = \frac{(1-u)vP_{112,u} + u(1-v)P_{112,v}}{(1-u)v + u(1-v)},$$

$$Q_{121}(u,v,w) = \frac{(1-u)wP_{121,u} + u(1-w)P_{121,w}}{(1-u)w + u(1-w)},$$

$$Q_{211}(u,v,w) = \frac{(1-v)wP_{211,v} + v(1-w)P_{211,w}}{(1-v)w + v(1-w)},$$

$$Q_{i,j,k}(u,v,w) = P_{i,j,k} \quad \text{otherwise.}$$

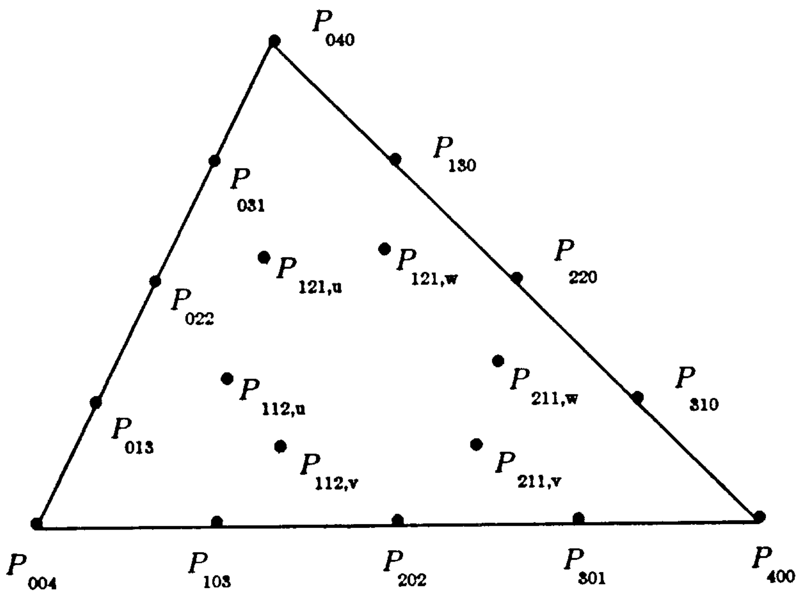


**Figure 2.4.** *A quartic triangular Gregory patch.*

Again, if

$$P_{112,u} = P_{112,v},\ P_{121,u} = P_{121,w},\ P_{211,v} = P_{211,w},$$

the triangular Gregory patch becomes a triangular Bézier patch. The Gregory patch has twice as many interior control points as the Bézier patch. When each pair of corresponding Gregory patch control points coincides, the patch becomes a Bézier patch.

Finally, we will point out *the convex hull property* of both Bézier and Gregory patches: all points on the patch are contained within the convex hull of their control points. This property is useful for rough interference checks [Faux et al '79].

# 3. SYMMETRIC CONVEX DECOMPOSITION ALGORITHM

## 3.1. Overall Description of the Algorithm

As mentioned above, *uci* does not exploit symmetries of the object. The reason for this is that the convex decomposition module, *ugtess*, does not necessarily produce a symmetric decomposition even though the input polygon (face) has a symmetry axis. As an example, consider the polygon in Fig. 3.1, which *ugtess* partitions as shown in Fig. 3.2. Clearly, this decomposition is not symmetric. Moreover, edge *HK* is redundant, i.e. it can be removed without destroying the convexity of the pieces. Our new version of the convex decomposition module does not produce such irregularities. Fig. 3.3 shows the same polygon, decomposed with the new version of *ugtess*.

The new algorithm is also based on the monotone convex decomposition [Gigus '85], utilized in *ugtess*. It proceeds in several stages:

1. Find a symmetry axis of the input polygon. If no symmetry axis exists, go to step 3.

2. Cut the polygon in half along the symmetry axis, relinking contours and keeping pointers to the symmetric parts (edges, vertices).

3. If the polygon was divided (i.e. it has a symmetry axis), perform old version of *ugtess* on one "half" polygon with removing redundant edges (see above). Otherwise (the polygon has no symmetry axis), perform old *ugtess* on the whole polygon, remove redundant edges and exit.

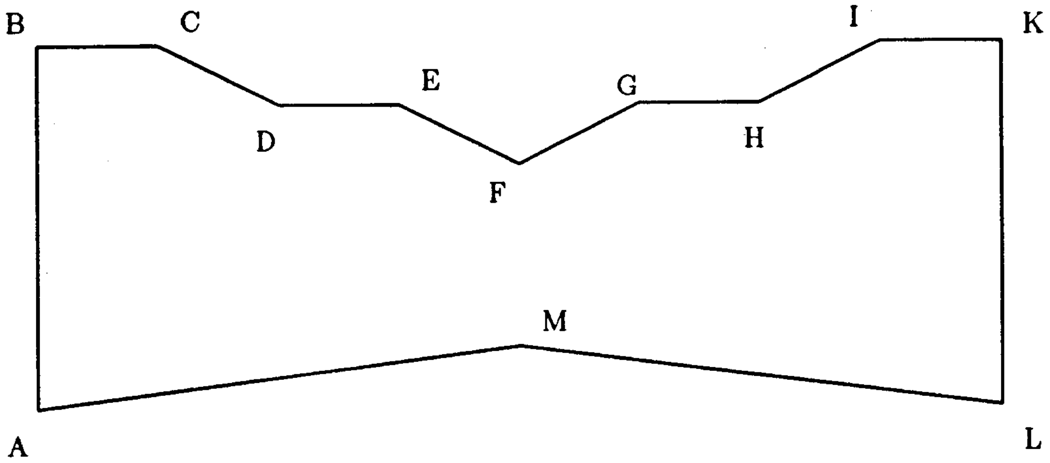


**Figure 3.1.** *A non-convex test polygon.*

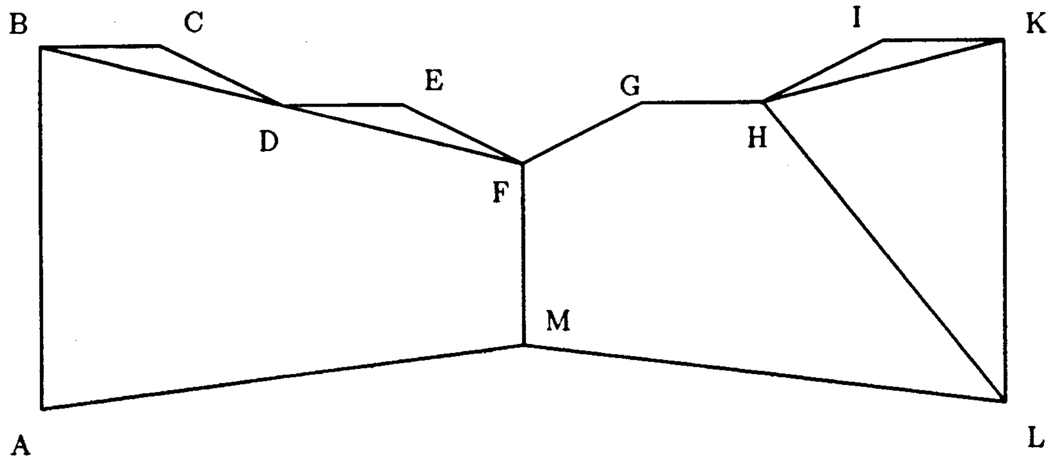


**Figure 3.2.** *The polygon of Fig. 3.1, decomposed with old* ugtess.

4. Create symmetric images of the new edges in the other "half" polygon.

5. Join the two tessellated "half" polygons, relinking contours. Remove redundant pieces of the symmetry axis and possibly new vertices (see section 3.4).

The new *ugtess* program, as well as the old one, has a *triangulate* option. If it is set, the original polygon is tessellated into triangles rather then convex polygons. To tessellate the polygon into triangles, the algorithm above is used, except that in step 3 the "half" polygon is divided into triangles, and no new edges are removed.

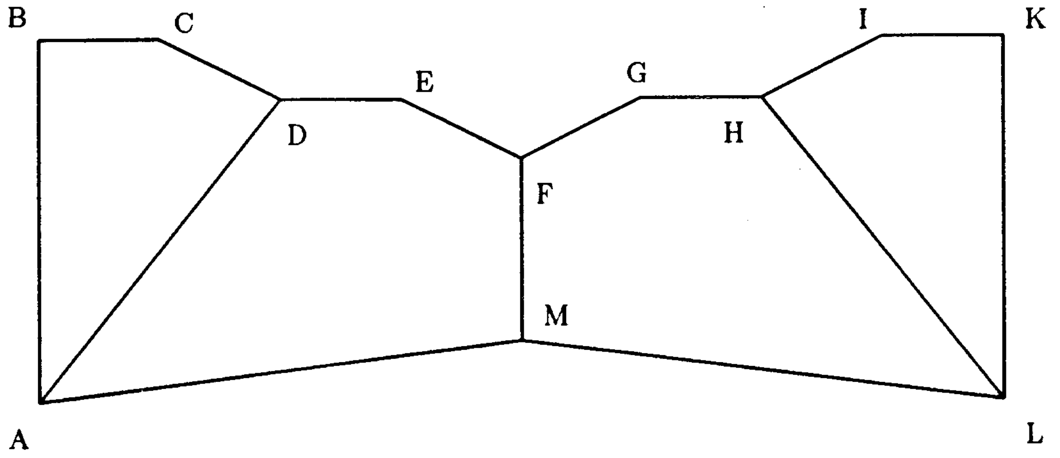


**Figure 3.3.** *The polygon of Fig. 3.2, decomposed with improved* ugtess.

The option to tessellate the polygon exclusively into triangular and convex quadrilateral polygons will be added in the near future.

### 3.2. Finding a Symmetry Axis

Locating a symmetry axis is complicated by the fact that our polygon doesn't have to be simply connected, i.e it could have holes inside (allowed by UNIGRAFIX). Suppose an outer contour of the polygon has $n$ edges. Then the polygon can have at most $n$ symmetry axes. Examples when $n$ is even and odd are shown in Fig. 3.4. Thus we have to check at most $n$ axes: when $n$ is even, $n/2$ axes go from one vertex to the opposite one, and $n/2$ axes go from the middle of one edge to the middle of the opposite one. When $n$ is odd, all $n$ axes go from a vertex to the middle of the opposite edge.

To minimize unnecessary computation, we first compute the *centers* of the outer contour and of each hole. Obviously, the center of the outer contour must lie on the proposed symmetry axis. Centers of contours, which do not lie on the axis, must have a symmetric image on the other side of the axis. This is checked directly by computing the symmetric image of a center and comparing it to the centers of other contours not on the axis. Clearly, symmetric images of centers must be centers themselves. In Fig. 3.5 small circles represent centers of contours.

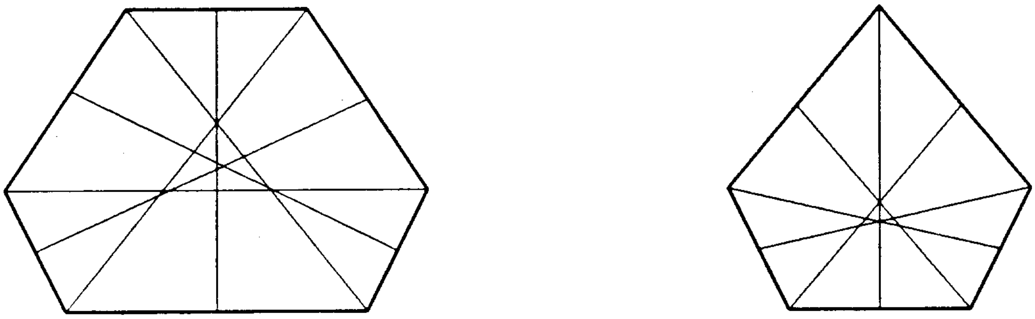


**Figure 3.4.**

*Possible symmetry axes of polygons with even and odd number of edges*

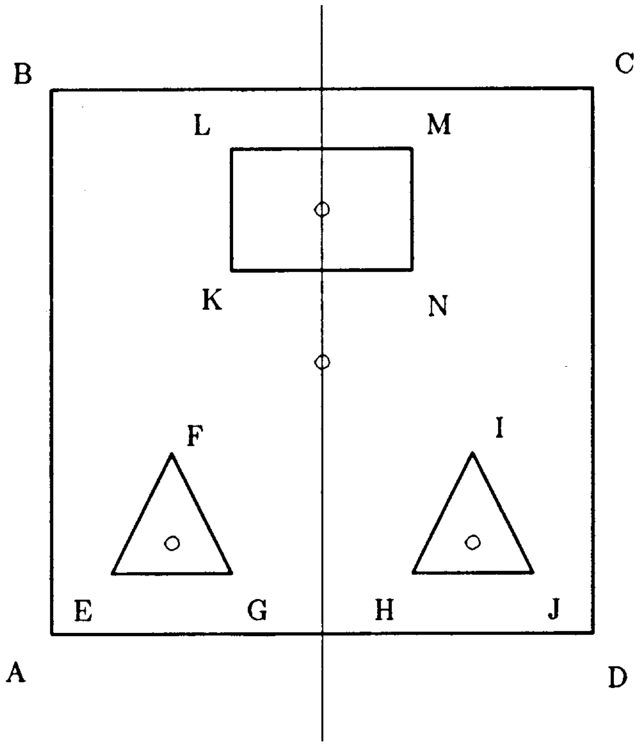


**Figure 3.5.**

*Centers of contours must be symmetric with respect*

*to the symmetric axis. They are represented with small circles.*

Now, we have to check if vertices themselves are symmetric with respect to the proposed axis. This can be done efficiently due to the fact that vertices in each contour appear in clockwise (outer contour) or counterclockwise (holes) order (UNIGRAFIX format). Therefore, we know which vertices are supposed to be symmetric images of each other. We check whether a line segment, joining them, is orthogonal to the symmetry axis, and is split in half by the symmetry axis. This test works for both outer and inner contours. For the case of an inner contour, we have to locate the symmetric image of the first vertex before the above-mentioned test can be applied.

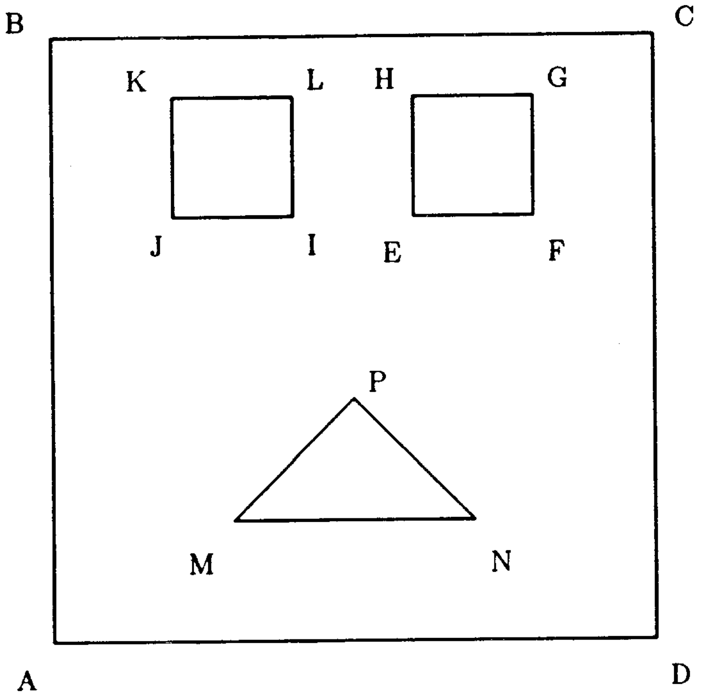


**Figure 3.6.** *A test polygon with several holes*

The UNIGRAFIX definition of the face shown in Fig. 3.6 might be written as:

f    ( A B C D ) ( E F G H ) ( J I L K ) ( M N P );

In this example, we would test the following line segments to be orthogonal and divided in half by the proposed symmetry axis: *DA, BC, MN, EI, FJ, GK, HL.*

If the polygon passes this test, we have to relink the contour to produce the two "half" polygons. To do this, we have to sort centers of holes which lie on the symmetry axis. If the axis splits an edge of a hole or of the outer contour, a *new vertex* will be introduced. Fig. 3.7 shows one resulting "half" polygon. Arrows indicate the order in which edges are linked.

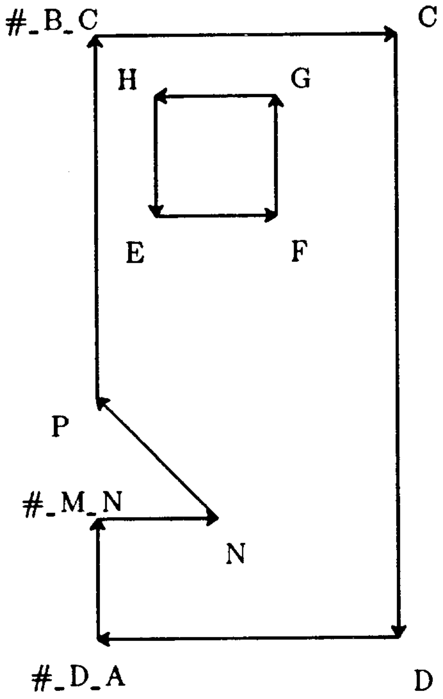


**Figure 3.7.** *The "half" polygon, generated by the symmetric cut.*

In the process of dividing the polygon we have introduced *new vertices* on the symmetry axis. For the face in Fig. 3.6, the new vertices are *#_B_C*, *#_D_A*, *#_M_N*. The program assigns names to the new vertices in the form #_*vert*1_*vert*2, if a new vertex lies on the edge between *vert*1 and *vert*2.

This part of the algorithm can be briefly described in pseudo-computer language as follows:

```
for each possible symmetry axis do
        check centers of contours;
        if centers are symmetric then
                check vertices;
                if vertices are symmetric then
                        exit: symmetric axis found;
                else loop;
        else loop;
end;
```

### 3.3. Removing Redundant Edges

This is an unsophisticated, but very important part of the algorithm. For each new edge, i.e. an edge that was added by old *ugtess,* we check if we can discard without destroying the convexity property of a subpolygon. This is done by checking an angle between incoming and outgoing edges, closest to the new edge, at both its endpoints. The edge shown in Fig. 3.8 cannot be removed, because the angle at vertex *A* would not be convex. We can easily find the proper incoming and outgoing edges, using the fact that each contour is a linked list of its edges, and each new edge is actually present twice (once in each contour that contains it).

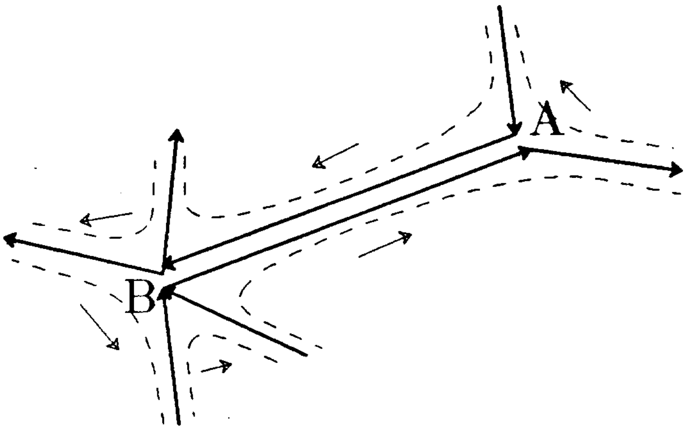


**Figure 3.8.**

*The angle at vertex* A *would not be convex, if edge* AB *is removed.*

*The dashed lines with arrows represent the contours.*

The resulting decomposition is ***minimal*** in the sense that no new edge can be removed without destroying the convexity property. However, it is ***not minimal*** in an absolute sense, since it does not necessarily produce a decomposition of a polygon into the ***smallest possible*** number of convex pieces. We feel that breaking a polygon into smallest possible number of pieces is not essential, and would complicate the program. Significantly, *uci* can only handle triangles and quadrilaterals anyway; tessellating a polygon into the smallest possible number of convex pieces is likely to produce polygons with many sides.

### 3.4. Joining the Two Halves Together

After the tessellation of one of the two symmetric halves has been completed, the same tessellation is carried out on the other half, too. Since we have symmetric pointers for vertices and edges, creating edges that are symmetric images of new edges is not difficult. Relinking the contour is also easy. Fig. 3.9 shows a tessellated symmetrical polygon after these operations.

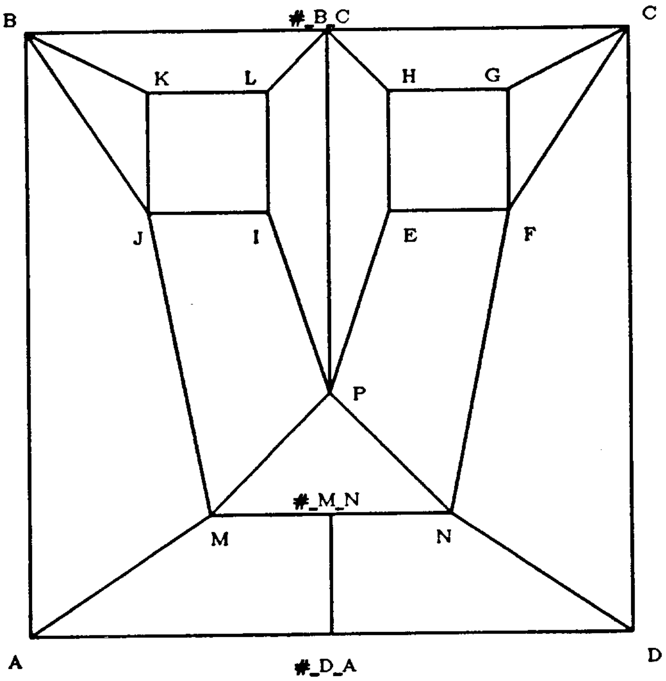


**Figure 3.9.**

*The polygon of Fig. 3.6 with the relinked contour before the removal of the unneeded parts of the symmetry axis and the new vertices.*

As we can see, edges ($\#\_B\_C \;\; P$) and ($\#\_M\_N \;\; \#\_D\_A$) are redundant and can be removed. The procedure for that has been described above. Also, new vertices $\#\_B\_C$ and $\#\_M\_N$ are redundant and can be removed. We only have to reset a few pointers. However, the new vertex $\#\_D\_A$ cannot be removed, because even after removing redundant pieces of the symmetry axis, there are new edges that use this vertex as an endpoint: ($H \;\; \#\_B\_C$) and ($\#\_B\_C \;\; L$).

We realize that adding extra vertices is an undesired feature of the algorithm. However, extra vertices will not present any difficulty for UNIGRAFIX programs, dealing with straight lines and polyhedra, that might use *ugtess'* output as their input. As for other programs, such as *uci*, one can locate a new vertex by the special name, assigned by *ugtess* (see above). In Section 5.1 we will discuss how *uci* deals with it. Simplicity was an important consideration in selecting the described approach.

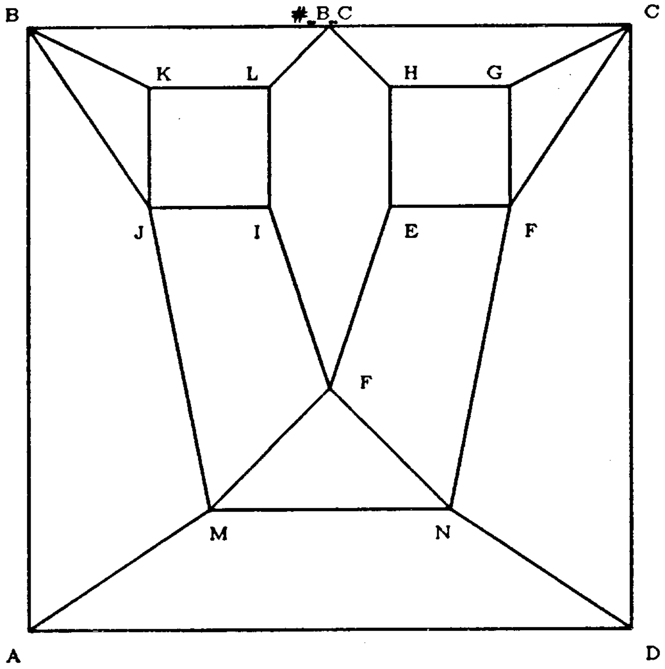


**Figure 3.10.**

*Final decomposition of the polygon.*

Fig. 3.10 shows the final output, produced by improved *ugtess*. Appendix 2 has additional examples.

## 4. GEOMETRIC CONTINUITY ACROSS CUBIC BOUNDARIES

Constructed patches must meet "smoothly" across joint boundaries. In the course of this work, "smoothness" will imply *geometric continuity of degree 1* ($G^1$). Two adjacent patches are said to be $G^1$ continuous, if the surface normal direction at the boundary is continuous.

Since the patches we are dealing with are defined in terms of their control points, we have to find the conditions on the control points of the two patches on either side of the shared boundary that guarantee $G^1$ continuity. In our application all the boundaries are cubic Be'zier curves.

## 4.1. Methods Involving Control Points on Both Sides of the Boundary

### 4.1.1. *General Formulation*

Let us now consider in its full generality the problem of joining two patches smoothly across a cubic boundary. Suppose that two patches, $\Phi$ and $\Psi$, meet along the boundary $\Gamma$ (Fig. 4.1). Then, for any point $v$ on $\Gamma$, we can find $D\Phi(v)$, a cross-boundary derivative of $\Phi$ at $\Gamma(v)$, and $D\Psi(v)$, a cross-bound derivative of $\Psi$ at the same point. Let $D\Gamma(v)$ be the tangent vector at $\Gamma(v)$. Then the necessary and sufficient conditions for patch continuity is coplanarity of all three vectors:

$$\det\,(D\Phi(v), D\Psi(v), D\Gamma(v)) = 0,\ \ v \in [0,1].$$

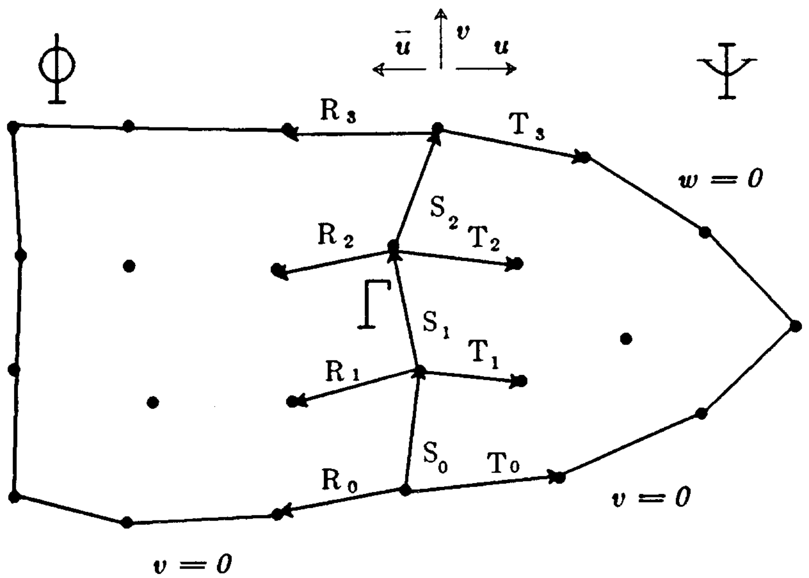


**Figure 4.1.** *Two patches, joining across the common boundary.*

Patches $\Phi$ and $\Psi$ can either be bicubic quadrilateral or quartic triangular patches. Consider the case when the quadrilateral and the triangular patch join across the common cubic boundary. For the quadrilateral patch, $D\Phi(v)$ is just the usual partial derivative $\Phi_u(0,v)$. For the triangular patch, $D\Psi(v)$ is the *radial* derivative [Farin '82]:

$$D\Psi(v) = (1-v)\,(\Psi_u - \Psi_v) + v\,(\Psi_w - \Psi_v).$$

Both bicubic quadrilateral and quartic triangular patches have two internal control points along each edge, and, therefore, the same procedure for finding these control points can be used.

The common cubic boundary of the two patches is used as a cubic for the evaluation of the bicubic quadrilateral patch, and has to be degree-elevated to quartic for the evaluation of the quartic triangular patch (see Section 2.1).

$D\Phi(v)$ and $D\Psi(v)$ are polynomials of third degree in $v$, while $D\Gamma(v)$ is a polynomial of second degree. They can be expressed in terms of Berstein polynomials and control points:

$$D\Gamma(v) = 3\sum_{i=0}^{2} \mathbf{S}_i\, B_i^2(v), \quad D\Phi(v) = 3\sum_{i=0}^{3} \mathbf{R}_i\, B_i^3(v), \quad D\Psi(v) = 3\sum_{i=0}^{3} \mathbf{T}_i\, B_i^3(v).$$

The above expressions for cross-boundary derivatives are valid for bicubic quadrilateral patches only; for quartic triangular patches, the coefficient 3 is changed to 4 [Farin '82].

If we evaluate the determinant equation for $v = 0$ and for $v = 1$, that is, for the endpoints of the boundary curve, we can see that the vectors $\mathbf{R}_0$, $\mathbf{T}_0$, $\mathbf{S}_0$ must be coplanar; analogously, vectors $\mathbf{R}_3$, $\mathbf{T}_3$, $\mathbf{S}_2$ must also lie in the same plane. These conditions are called *endpoint conditions* for the boundary curve or *vertex planarity conditions* for the two neighbor patches. Vertex normals $\mathbf{N}_0$ and $\mathbf{N}_n$ (this notation will become clear later) are orthogonal to the respective endpoint planes.

The above determinant equation is a polynomial of degree 8 in $v$, which yields 9 equations for the coefficients of the various degrees of $v$. Two of these equations represent endpoint conditions. The remaining 7 equations for the 4 unknown vectors $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{T}_1$, $\mathbf{T}_2$ (or 12 unknown scalars) proved to be so complicated, that even MACSYMA couldn't handle them [Longhi '85].

The fact that there arė 7 constraints across each boundary (and, therefore, a total of 5 degrees of freedom) has several important implications. First, one cannot, generally speaking, pick the control points in one patch freely and hope to determine suitable control points on the neighboring patch, because that would use 6 degree of freedom, while actually there are only 5. There is always one more constraint than degrees of freedom for whatever patch degree is used: each time the degree is raised by one, the order of the above determinant equation increases by three, but the extra control point also gives three additional degrees of freedom. Second, it is not possible to fit quartic triangular and/or bicubic quadrilateral Bézier patches into already defined cubic boundaries. For each quartic triangular Bézier patch, there are 3 inside control points, or 9 degrees of freedom; however, each boundary brings in on the average $7/2 = 3.5$ constraints, which gives a deficit of 1.5 unfulfilled constraints for each triangle, and, analogously, of 2 unfulfilled constraints for each quadrilateral.

This lack of degrees of freedom forces us to use Gregory patches rather than Bézier patches, if the boundary curves are predetermined. Another possibility would be to use the control points of the boundary curves to provide the required additional degrees of freedom. However, this leads to a highly complicated system of equations that could be quite difficult to solve, and it destroys the property of locality.

### 4.1.2. *Formulation with Vector Equations*

Rather then trying to solve the above determinant equation, one could premultiply each derivative by an unknown polynomial, and then equate the resulting linear combination to 0:

$$\alpha(v)\, D\Phi(v) + \mu(v)\, D\Psi(v) + \lambda(v)\, D\Gamma(v) = 0.$$

Since $D\Phi(v)$ and $D\Psi(v)$ are of degree 3, while $D\Gamma(v)$ is of degree 2, it is clear that

$$\mathbf{deg}\ (\alpha(v)) = \mathbf{deg}\ (\mu(v)) = \mathbf{deg}\ (\lambda(v)) - 1.$$

In this case, a system of *vector* equations, i.e. three identical scalar systems, one for each coordinate, will have to be solved. This is simpler than the case for the determinant equation, which mixes the components along different axes.

### 4.1.3. *Farin's Method*

Farin [Farin '83] and Chiyokura [Chiyokura '83], indeed, take the latter approach. Farin premultiplies $D\Phi$ and $D\Psi$ by a constant, and $D\Gamma$ by a linear polynomial. Since one of the coefficients can always be set to 1, this leaves 3 coefficients to be determined. However, planarity conditions on either end of the boundary give 4 constraints. This means that the original control points of the cubic boundaries have to satisfy an additional constraint. Farin formulates this as a constraint on the areas of the triangles $\mathbf{R}_0\mathbf{S}_0$, $\mathbf{S}_0\mathbf{T}_0$, $\mathbf{R}_3\mathbf{S}_2$ and $\mathbf{S}_2\mathbf{T}_3$ in Fig. 4.1. See also Section 5.1.

Thus, Farin's method allows the computation of interior control points on either side of the boundary, provided that the boundary control points satisfy the area ratio constraint [Farin '82]. Unfortunately, this approach is not flexible enough for our purposes, because we would like to choose the cubic boundaries in an unconstrained manner.

### 4.1.4. *Multiplication by the Higher Order Polynomials*

Alternatively, we can premultiply $D\Phi$ and $D\Psi$ by linear, and $D\Gamma$ by quadratic polynomials. Now there are 7 coefficients of polynomials, one of which can be set to 1, and 4 constraints at either end. This leaves 2 coefficients to be chosen freely. Our vector polynomial is of degree 4, so one can expect 5 equations for the various degrees of $v$. Two of these equations, however, are fulfilled automatically at the ends of the boundary, leaving just three vector equations. Thus, if $\mathbf{R}_1$ is set, and the 2 extra coefficients are chosen (note that again there are 5 degrees of freedom), then $\mathbf{R}_2$, $\mathbf{T}_1$ and $\mathbf{T}_2$ can be determined. Thus one can find all four control points with just one computational pass across each boundary. However, it's not clear how to choose $\mathbf{R}_1$ to ensure a "symmetric" choice of control points.

We can go even further and premultiply the derivatives by two quadratic and one cubic polynomial, giving a total of 10 coefficients. One of them again can be set to 1, 4 will be determined from planarity conditions, leaving a free choice of the remaining 5 coefficients. Now there are three more free constants than in the approach described in the previous paragraph. However, there is a very important advantage: our polynomial is now of degree 5, two conditions at the ends are satisfied, and so there are *four* vector equations for *four* points $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{T}_1$, $\mathbf{T}_2$! No guesswork is needed here in terms of control points and there is some hope for a "symmetric" solution, as soon as the 5 coefficients, representing all the degrees of freedom that exist, are picked.

We have succeeded in obtaining a closed-form solution in terms of the unknown coefficients. However, the resulting equations are too cumbersome for practical use, and it is not yet clear how to choose these unknown coefficients.

## 4.2. Methods, Involving Control Points on One Side of the Boundary

All the approaches, described above, resulted in equations, which involved both pairs of interior control points on either side of the boundary. The last one permits in principle an independent calculation of the pair of interior control points on one side of the boundary in one computational pass. In the following, we discuss methods that rely solely on the boundary information to find the pair of interior control points on either side of the boundary independently. Thus, to determine all four control points, actually *two passes* along each boundary are required. However, these methods have the advantage to be completely *local*, i.e. interior control points depend on boundary information only and not on the interior control points of the neighboring patch.

### 4.2.1. *Chiyokura's Method*

In his method, Chiyokura [Chiyokura '83], [Chiyokura '86] interpolates the cross-boundary tangent. He assumes the cross-boundary tangent to be the difference of the vectors $\mathbf{T}_0$ and $\mathbf{R}_0$ at one end of the boundary, and the difference of the vectors $\mathbf{T}_3$ and $\mathbf{R}_3$ at the other end (see Fig. 4.1). Although the cross-boundary tangent can be quadratic in general (see Section 4.1.1), Chiyokura assumes it to be linearly interpolated between the cross-boundary tangent at the ends of the boundary, which allows to compute the interior control points easily. See Appendix 3 for Chiyokura's equations. This procedure is performed separately for the two neighbor patches. The resulting interior control points guarantee $G^1$ continuity between the two patches.

Chiyokura's method provides an easy way of computing interior control points of the Gregory patch, and is currently used in our implementation.

### 4.2.2. *Intrinsic Normal Approach*

It seems natural to describe $G^1$ continuity as orthogonality of the intrinsic normal of the boundary curve $\mathbf{N}(v)$ and the cross-boundary derivative $D\Phi(v)$. The ***intrinsic normal*** of the curve $\Gamma(v)$ is defined as a normalized derivative of the unit tangent vector of $\Gamma(v)$:

$$\mathbf{N}(v) = \frac{D\mathbf{T}(v)}{|D\mathbf{T}(v)|}, \text{ where } \mathbf{T}(v) = \frac{D\Gamma(v)}{|D\Gamma(v)|}.$$

MACSYMA showed us that the equation

$$(\, \mathbf{N}(v), D\Phi(v)\,) = 0$$

is a polynomial of degree 7, giving 8 scalar equations for 6 coordinates of $\mathbf{R}_1$ and $\mathbf{R}_2$. However, the intrinsic normal $\mathbf{N}(v)$ may not coincide with the precomputed vertex normal at the ends of the boundary. Two of the above 8 equations result in constraints on the cubics themselves, which may not be acceptable.

### 4.2.3. *Prescribed Normal Method*

Rather than trying to describe the behavior of the normal with the intrinsic curve normal, we can "prescribe" the normal to the boundary. Prescribed normal $\mathbf{N}(v)$ of degree $n$ can be expressed as the Be'zier polynomial

$$\mathbf{N}(v) = \sum_{i=0}^{n} \mathbf{N}_i \, B_i^n(v)$$

and has to satisfy the following conditions:

1. $\mathbf{N}_0$ and $\mathbf{N}_n$ must be equal to the precomputed vertex normals at the beginning and the

end of the boundary, respectively;

2. $\mathbf{N}(v)$ has to be orthogonal to the boundary tangent:

$$(\, \mathbf{N}(v), D\Gamma(v)\,) = 0, \qquad v \in [0,1];$$

3. $\mathbf{N}(v)$ has to be orthogonal to the cross-boundary derivative:

$$(\, \mathbf{N}(v), D\Phi(v)\,) = 0, \qquad v \in [0,1].$$

Since the unnormalized normal vector can be expressed as a cross product between the boundary tangent $D\Gamma(v)$ and the cross-boundary derivative $D\Phi(v)$, the degree of $\mathbf{N}(v)$ can be at most 5, disregarding the normalizing denominator.

Suppose the degree of the prescribed normal is $n$. Then the equation

$$(\, \mathbf{N}(v), D\Gamma(v)\,) = 0$$

will be a Be'zier polynomial of degree $n + 2$, and we will have $n + 3$ equations. Two of these equations, however, will reflect boundary conditions at the endpoints, so, in fact, there will be only $n + 1$ constraints. All degrees of freedom are contained in the vectors $\mathbf{N}_0$ through $\mathbf{N}_n$; however, $\mathbf{N}_0$ and $\mathbf{N}_n$ are known, which leaves $3\,(n - 1)$ degrees of freedom. To match the number of constraints with the number of degrees of freedom, we set

$$n + 1 = 3\,(n - 1),$$

that is, $n = 2$. Thus, the prescribed normal of degree 2 is determined by $\mathbf{N}_0, \mathbf{N}_2, \mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2$. Indeed, substituting the Bernstein expressions for $\mathbf{N}(v)$ and $D\Gamma(v)$, we have:

$$(\, (1-v)^2\, \mathbf{N}_0 + 2\,(1-v)v\ \mathbf{N}_1 + v^2\, \mathbf{N}_2,\ \ (1-v)^2\, \mathbf{S}_0 + 2\,(1-v)v\ \mathbf{S}_1 + v^2\, \mathbf{S}_2\,) = 0.$$

Expanding the scalar product, and equating the coefficients of $(1-v)^{2-i}v^i$, $i = 0,1,2$ to 0, we get the following five equations:

$$(\, \mathbf{N}_0, \mathbf{S}_0\,) = 0,$$
$$2\,(\, \mathbf{N}_0, \mathbf{S}_1\,) + 2\,(\, \mathbf{N}_1, \mathbf{S}_0\,) = 0,$$
$$(\, \mathbf{N}_0, \mathbf{S}_2\,) + 4\,(\, \mathbf{N}_1, \mathbf{S}_1\,) + (\, \mathbf{N}_2, \mathbf{S}_0\,) = 0,$$
$$2\,(\, \mathbf{N}_1, \mathbf{S}_2\,) + 2\,(\, \mathbf{N}_2, \mathbf{S}_1\,) = 0,$$
$$(\, \mathbf{N}_2, \mathbf{S}_2\,) = 0.$$

The first and the last equations are the endpoint conditions and are trivially satisfied by the construction of the vertex normals. The remaining three equations form a system, from which $\mathbf{N}_1$ can be determined. In fact,

$$\mathbf{N}_1 = \frac{(\mathbf{N}_0,\mathbf{S}_2)(\mathbf{S}_0\times\mathbf{S}_2) - 4(\mathbf{N}_0,\mathbf{S}_1)(\mathbf{S}_1\times\mathbf{S}_2) - 4(\mathbf{N}_2,\mathbf{S}_1)(\mathbf{S}_0\times\mathbf{S}_1) + (\mathbf{N}_2,\mathbf{S}_0)(\mathbf{S}_0\times\mathbf{S}_2)}{4\,(\,\mathbf{S}_0,\,\mathbf{S}_1,\,\mathbf{S}_2\,)}.$$

The denominator is a mixed product of $\mathbf{S}_0$, $\mathbf{S}_1$ and $\mathbf{S}_2$. It is equal to zero, if the cubic boundary is planar. It turns out that, in this case, the normal will be *cubic* in the plane of the curve, and *linear* in any plane, orthogonal to it! We have to escape to the higher degree because, in the plane, a control point of a curve has only two unknown coordinates, while the number of equations remains the same.

Once we agree that the degree of $\mathbf{N}(v)$ is 2, then the equation

$$(\,\mathbf{N}(v),\, D\Phi(v)\,) = 0$$

is of degree 5, giving 6 constraints, two of which again reflect the endpoint conditions. So there are 4 constraints for 6 unknowns $\mathbf{R}_1$ and $\mathbf{R}_2$. However, if we assume that the cross-boundary derivative is *quadratic*, then, unless the normal is planar, the "middle" point $\mathbf{R}_M$ is determined uniquely! The expression for $\mathbf{R}_M$ is analogous to the expression for the "middle" normal vector $\mathbf{N}_1$:

$$\mathbf{R}_M = \frac{(\mathbf{R}_0,\mathbf{N}_2)(\mathbf{N}_0\times\mathbf{N}_2) - 4(\mathbf{R}_0,\mathbf{N}_1)(\mathbf{N}_1\times\mathbf{N}_2) - 4(\mathbf{R}_3,\mathbf{N}_1)(\mathbf{N}_0\times\mathbf{N}_1) + (\mathbf{R}_3,\mathbf{N}_0)(\mathbf{N}_0\times\mathbf{N}_2)}{4\,(\,\mathbf{N}_0,\,\mathbf{N}_1,\,\mathbf{N}_2\,)}.$$

The real control points $\mathbf{R}_1$ and $\mathbf{R}_2$ are easily obtained by the degree elevation (see Section 2.1).

The case of the planar normal, again, has to be treated separately, because the determinant of the linear system used for general curves is identically zero.

The fact that there are different cases that have to be treated separately is an obvious disadvantage of the method. The normal, generally speaking, cannot be assumed to be of degree 2, due to the special case of the planar curve.

### 4.2.4. *Formulation with Surface Shape Parameters*

What if we choose a normal vector $\mathbf{N}(v)$ of higher degree? First, the choice of the normal is no longer unique: for a normal of degree $n$, $2n - 4$ degrees of freedom will somehow have to be disposed of. Suppose, however, that we have defined the normal of degree 3. Then the scalar product equation will produce a polynomial of degree 6, giving 5 constraints for the 6 unknowns of the 2 interior control points of the patch.

The fact that there are 2 degrees of freedom for the normal of degree 3 can be exploited to give the user more flexibility in the design of the surface. We believe that these two degrees of freedom can be described as the *warp* from some default position of the two intermediate normal control vectors of the boundary. The degree of freedom, remaining for each pair of the patch control points could be viewed as the *bulge* of the surface near the boundary. This parameter would describe how far from the boundary the interior control points will lie on the average. Note that as a result of the Prescribed Normal Method, there are only four degrees of freedom (parameters), rather than five: the two warp parameters of the normal and one bulge parameter for each of the two patches, bordering the boundary.

A practical design system should choose these parameters with some good default values. The user should then be given the possibility to alter any of these parameters to introduce the desired shape modifications in the surface.

At present, the formulation of $G^1$ continuity with the surface parameters is an open research issue. We are currently investigating different ways of arriving at the best default values for the above parameters, as well as the mathematical expression of the surface in terms of these parameters.

## 5. SUBDIVISION OF THE SURFACE INTO BEZIER PATCHES

The desirability of being able to subdivide the whole surface area into polynomial patches has been already mentioned. The methods described in Section 4 to join patches with $G^1$ continuity readily lead to non-polynomial Gregory patches. An analogous approach aimed at subdividing the surface into Bézier patches proved to be futile, due to the fact that a Bézier patch has only half as many interior cóntrol points as a Gregory patch, and not all the constraints across each cubic boundary can be satisfied simultaneously (see Section 4.1.1).

As a remedy to this, we propose to create a number of Be'zier patches in the place where a single Gregory patch is ordinarily constructed. This is almost a subdivision of one patch into several, except that the original patch is actually never defined. Nevertheless, we will call this operation "subdivision", so that we don't have to invent a new word for it. A very useful and important property of the proposed subdivision is the fact that it is completely *local*, that is, all control points of a patch are determined solely from the control points of the cubics that border this patch.

## 5.1. Triangular Patch Subdivision

In the simplest case, we subdivide a triangular patch into three Be'zier patches. This subdivision employs Farin's method, which results in vector equations, and the system of resulting equations actually splits into three separate subsystems.

Suppose now that we have a triangular face and wish to construct a polynomial patch. We construct three Be'zier patches and ensure geometric continuity ($G^1$) between them. Our patches are shown in Fig. 5.1. $B_i$, $C_i$, $D_i$, $i = 1,2$ are control points on the cubic curves, which are given as far as this subvision task is concerned. $S_i$, $P_i$, $i = 1,2$ are the control points of the cubic boundaries between the subpatches and are yet to be determined, as well as the control points inside each subpatch.

Assume for the moment that $L_{13}$, $M_{13}$, $L_{12}$, $K_{12}$, $K_{23}$, $M_{23}$ are all known. These points ensure smoothness across the "real" external boundaries and can be determined without much difficulty using Chiyokura's method (see section 4.2.1), once the points $S_1$, $S_2$, and $S_3$ are known.

Clearly, $S_1$ must lie in the plane of $T_1B_1C_1$, and analogously, $S_2$ in the plane of $T_2C_2D_2$, $S_3$ in the plane of $T_3B_2D_1$, and, finally, $Z$ in the plane of $P_1P_2P_3$. Thus, there are 3 degrees of freedom

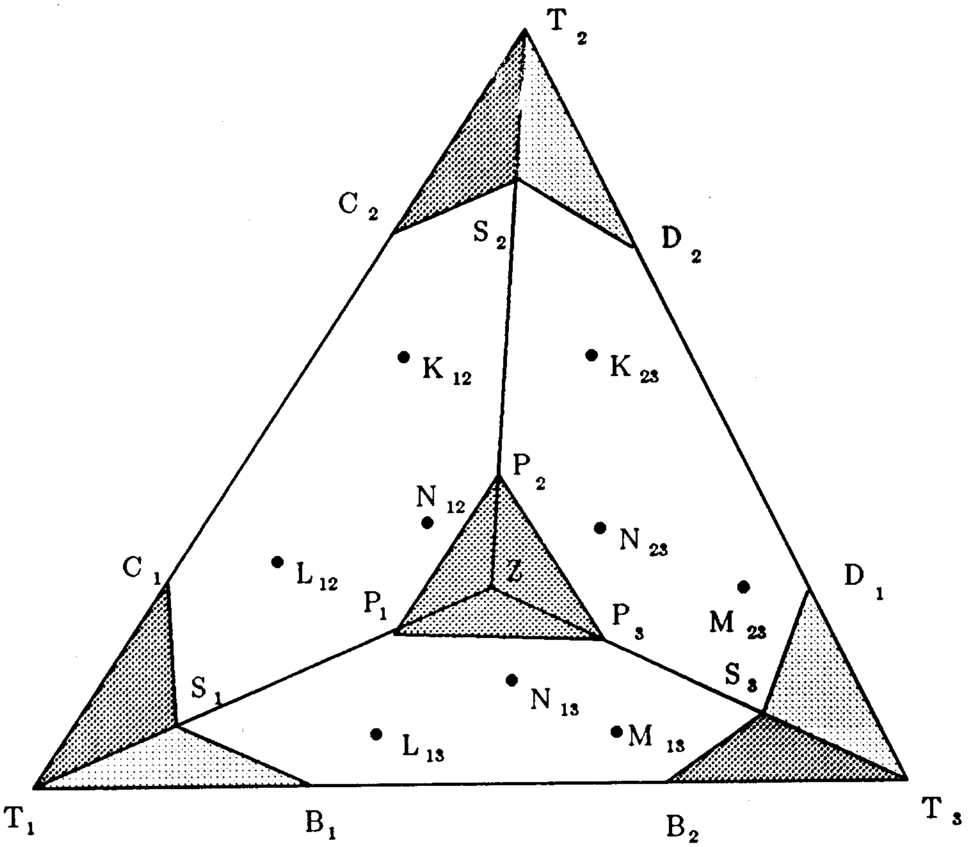


**Figure 5.1.** *Triangular patch subdivision.*

each for points $P_1$, $P_2$, $P_3$, $N_{12}$, $N_{13}$, $N_{23}$ and 2 degrees of freedom each for points $S_1$, $S_2$, $S_3$, $Z$. This gives a total of 26 degrees of freedom.

We use Farin's [Farin '83] method of ensuring $G^1$ continuity between the subpatches. For subpatches $T_1ZT_3$ and $T_1ZT_2$ to be continuous, it is necessary that the following triangle areas match:

$$\frac{\textbf{area }(T_1S_1C_1)}{\textbf{area }(T_1S_1B_1)} = \frac{\textbf{area }(P_1ZP_2)}{\textbf{area }(P_1ZP_3)}.$$

Furthermore, to guarantee $G^1$ continuity across the internal boundary, the additional two vector (six scalar) equations must be satisfied, involving points $T_1$, $S_1$, $P_1$, $Z$, $L_{12}$, $L_{13}$, $N_{12}$, $N_{13}$. Therefore, there are 7 constraints across each boundary (as expected! see Section 4.1.1), which gives a total of 21 for the whole subdivision. So there are 5 extra degrees of freedom left for the whole triangle.

We make use of these extra degrees of freedom as follows. First, we use 2 degrees of freedom and choose $Z$ to be in the *center of gravity* of the triangle $P_1P_2P_3$, that is,

$$\mathbf{Z} = \frac{1}{3}\mathbf{P}_1 + \frac{1}{3}\mathbf{P}_2 + \frac{1}{3}\mathbf{P}_3.$$

Therefore, the areas of triangles $P_1ZP_2$, $P_1ZP_3$, and $P_2ZP_3$ are *all equal.*

Now, to satisfy above area ratio condition, areas $T_1S_1C_1$ and $T_1S_1B_1$ must be equal, and analogously for the other two interior boundaries. It's sufficient that $S_1$ lies *anywhere on the median* of the triangle $T_1C_1B_1$, which goes through $T_1$. It seems natural to pick $S_1$ again in the center of gravity of $T_1C_1B_1$ (the center of gravity of a triangle is at the intersection of its medians). In making this decision, we have discarded two more degrees of freedom for each $S_i$ and at the same time satisfied the area ratio constraint for each interior boundary. Now, the number of the remaining degrees of freedom matches exactly the number of constraints (18).

The choice of the center of gravity of a triangle emerges from the following consideration. Suppose the overall triangle is equilateral. Then $Z$ would be in the center of gravity of the (big) triangle, and $S_i$ and $P_i$ would divide lines joining each vertex with $Z$ into 3 equal parts. Cubics control points would also divide each triangle side into three equal parts. But this situation corresponds to our initial choice of picking $S_i$ and $Z$ in the centers of gravity of the corresponding triangles.

There is another way to express the area ratio condition, which we actually used in solving the remaining 6 vector equations. $C_1$, $D_2$, $B_2$ and $P_i$ can be expressed in barycentric coordinates, using the corresponding points on the other side of the respective interior boundaries:

Boundary $(T_1Z)$: $\mathbf{C}_1 = \alpha_1\,\mathbf{T}_1 + \alpha_2\,\mathbf{S}_1 + \alpha\,\mathbf{B}_1, \quad \alpha_1 + \alpha_2 + \alpha = 1,$

$\mathbf{P}_2 = \alpha_3\,\mathbf{P}_1 + \alpha_4\,\mathbf{Z} + \alpha\,\mathbf{P}_3, \quad \alpha_3 + \alpha_4 + \alpha = 1,$

Boundary $(T_2Z)$: $\mathbf{D}_2 = \beta_1\,\mathbf{T}_2 + \beta_2\,\mathbf{S}_2 + \beta\,\mathbf{C}_2, \quad \beta_1 + \beta_2 + \beta = 1,$

$\mathbf{P}_3 = \beta_3\,\mathbf{P}_2 + \beta_4\,\mathbf{Z} + \beta\,\mathbf{P}_1, \quad \beta_3 + \beta_4 + \beta = 1,$

Boundary $(T_3Z)$: $\mathbf{B}_2 = \gamma_1\,\mathbf{T}_3 + \gamma_2\,\mathbf{S}_3 + \gamma\,\mathbf{D}_1, \quad \gamma_1 + \gamma_2 + \gamma = 1,$

$\mathbf{P}_1 = \gamma_3\,\mathbf{P}_3 + \gamma_4\,\mathbf{Z} + \gamma\,\mathbf{P}_2, \quad \gamma_3 + \gamma_4 + \gamma = 1.$

Area ratio constraints for all three boundaries are equivalent to equations

$$\alpha_1 + \alpha_2 = \alpha_3 + \alpha_4,$$
$$\beta_1 + \beta_2 = \beta_3 + \beta_4,$$
$$\gamma_1 + \gamma_2 = \gamma_3 + \gamma_4.$$

When we choose $S_1$, $S_2$, $S_3$, $Z$ in the center of gravity of the corresponding triangle, we actually set

$$\alpha_1 = \alpha_3 = \alpha = \beta_1 = \beta_3 = \beta = \gamma_1 = \gamma_3 = \gamma = -1$$

and

$$\alpha_2 = \alpha_4 = \beta_2 = \beta_4 = \gamma_2 = \gamma_4 = 3.$$

The parameter $\alpha$ can be thought of as the negative ratio of the areas of the triangles $T_1S_1C_1$ and $T_1S_1B_1$, or, which is the same, as the negative ratio of the areas of $P_1ZP_2$ and $P_1ZP_3$. The parameters $\beta$ and $\gamma$ have the analogous geometric interpretation.

The six vector equations [Farin '83] that we need to solve are listed below:

$$\mathbf{L}_{12} = \frac{2}{3}(\alpha_1\,\mathbf{S}_1 + \alpha_2\,\mathbf{P}_1) + \frac{1}{3}(\alpha_3\,\mathbf{T}_1 + \alpha_4\,\mathbf{S}_1) + \alpha\,\mathbf{L}_{13}$$
$$\mathbf{N}_{12} = \frac{1}{3}(\alpha_1\,\mathbf{P}_1 + \alpha_2\,\mathbf{Z}) + \frac{2}{3}(\alpha_3\,\mathbf{S}_1 + \alpha_4\,\mathbf{P}_1) + \alpha\,\mathbf{N}_{13}$$

$$\mathbf{K}_{23} = \frac{2}{3}(\beta_1\,\mathbf{S}_2 + \beta_2\,\mathbf{P}_2) + \frac{1}{3}(\beta_3\,\mathbf{T}_2 + \beta_4\,\mathbf{S}_2) + \beta\,\mathbf{K}_{12}$$
$$\mathbf{N}_{23} = \frac{1}{3}(\beta_1\,\mathbf{P}_2 + \beta_2\,\mathbf{Z}) + \frac{2}{3}(\beta_3\,\mathbf{S}_2 + \beta_4\,\mathbf{P}_2) + \beta\,\mathbf{N}_{12}$$

$$\mathbf{M}_{13} = \frac{2}{3}(\gamma_1\,\mathbf{S}_3 + \gamma_2\,\mathbf{P}_3) + \frac{1}{3}(\gamma_3\,\mathbf{T}_3 + \gamma_4\,\mathbf{S}_3) + \gamma\,\mathbf{M}_{23}$$
$$\mathbf{N}_{13} = \frac{1}{3}(\gamma_1\,\mathbf{P}_3 + \gamma_2\,\mathbf{Z}) + \frac{2}{3}(\gamma_3\,\mathbf{S}_3 + \gamma_4\,\mathbf{P}_3) + \gamma\,\mathbf{N}_{23}$$

With this, points $P_1$, $P_2$, $P_3$ can be determined directly from just one equation each. The remaining 3 equations form a simple linear system, from which $N_{12}$, $N_{13}$, $N_{23}$ can be found. Solutions to these equations, when $S_i$ and $Z$ are picked in the centers of gravity of the corresponding triangles, are listed in Appendix 4. We also have solutions for general $\alpha$'s, $\beta$'s, and $\gamma$'s, derived with MACSYMA's help, but they are too cumbersome to be listed here. Of course, each vector equation is actually 3 scalar equations, one for each coordinate.

## 5.2. Subdivision of the Patch over an Arbitrary Face

Let us now generalize the problem of the surface subdivision to the case of an arbitrary convex face with $n$ edges. We would like to represent the surface over this face as a union of $n$ Bézier patches, such that every pair of adjoining patches is $G^1$ continuous.

We use the notation and the approach of the previous section. Again, we assume that the control points $L^j_{i,i+1}$, $i = 1,...,n$, $j = 1,2$ (Fig 5.2) are known. Our task is, then, to determine points $P_i$, $S_i$, $N_{i,i+1}$, $i = 1,...,n$, and $Z$. There are 3 degrees of freedom each for the points $P_i$ and $N_{i,i+1}$, and two degrees of freedom each for the points $S_i$ and $Z$. Therefore, there is a total of $8n + 2$ degrees of freedom. On the other hand, there are 7 constraints across each interior boundary. Furthermore, a *planarity constraint* at $Z$ must be satisfied: all the points $P_i$ (and, of course, $Z$) must lie in the same plane. This introduces $n - 3$ more constraints. Thus, there is a total of $8n - 3$ constraints. We can see that there are always 5 degrees of freedom more than there are constraints. Therefore, in theory, *a surface over an arbitrary convex face can be represented as a union of geometrically continuous triangular Bézier patches of degree 4.*

However, it turns out that the planarity condition complicates the system of equations to such an extent, that so far we were not able to find a close-form solution. For the triangular case, there is no planarity condition (three points are always on the same plane!), which results in the system of vector equations. The planarity constraint, on the other hand, mixes all the coordinates, so that the equations can no longer be broken into three separate subsystems. The area ratio constraint also becomes more complicated. Even for the quadrilateral case it is not, in general, possible to pick the center point $Z$, so that all the areas of triangles $P_iZP_{i+1}$ can be made equal.

Thus, for $n > 3$, an alternative way of subdivision must be found. Since the planarity constraint is the main obstacle in obtaining an easy subdivision, it should be avoided. In fact, no

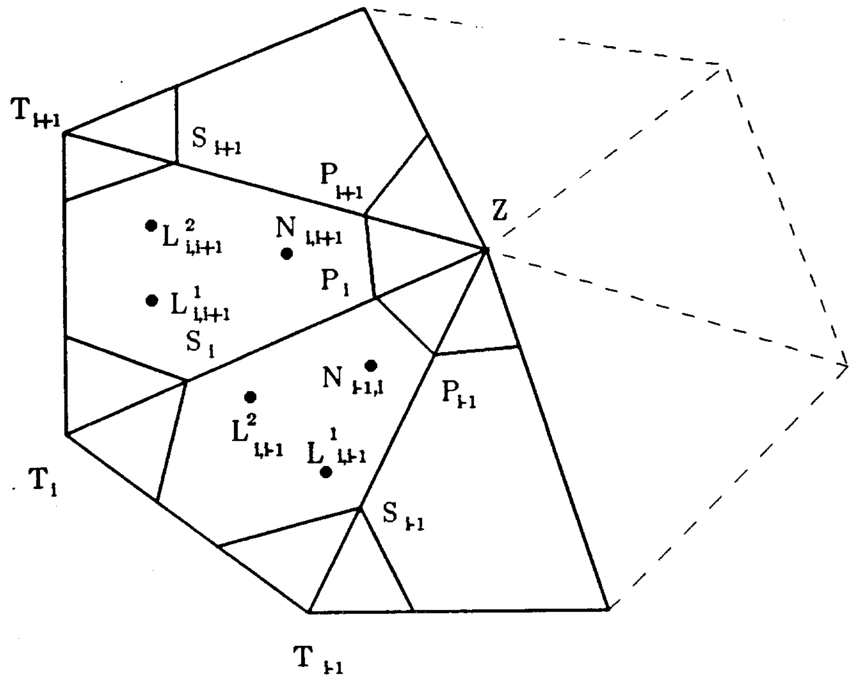


**Figure 5.2.** *Subdivision of the surface over an arbitrary face.*

more than *three* interior boundaries should meet at the same point. Keeping this in mind, we were able to subdivide an arbitrary quadrilateral patch.

## 5.3. Quadrilateral Patch Subdivision

We have arrived at the two subdivision schemes for the quadrilateral case (Fig. 5.3). Subdivision "*a*" requires less computation time, while the subdivision "*b*" treats all the vertices and sides of the quadrilateral in the same way. Due to the simplicity of the method, it does not require much computation time, either. We believe that both these subdivision schemes are acceptable.

The approach of the triangular patch subdivision is used here again to obtain all the necessary control points on the interior boundaries and the subpatches. Again, we assume that the control points of the subpatches, closest to the "real" boundary, are known. The control points of the interior boundaries, adjacent to the vertices of the face, can be picked freely, as long as the

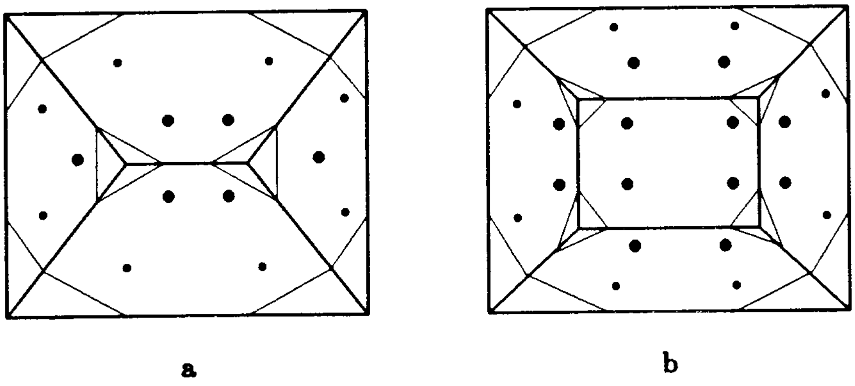


**Figure 5.3.** *The two ways of the quadrilateral patch subdivision.*

planarity condition at the corresponding vertex is satisfied. The second control points of these boundaries will be determined by Farin's formulas (see Section 5.1). The two control points in one (subdivision *a*) or in four (subdivision *b*) innermost boundaries can be chosen freely. The central points of the triangles, formed by the control points of the interior boundaries are then chosen in such a way, that the area ratio constraints are satisfied. Then the remaining control points of the subpatches (shown bold in Fig. 5.3) can be found from two (subdivision *a*) or four (subdivision *b*) separate systems of three vector equations.

A reasonable choice should be made to pick the control points in the interior boundaries. As we did with the triangular subdivision, we consider the case of the *perfect square* and make our selections based on this special case.

Our implementation uses the "symmetric" scheme *b*. It turns out that in this case, like in the triangular subdivision, the central points of the triangles should be chosen in their centers of gravity.

## 6. IMPLEMENTATION ASPECTS

### 6.1. *uci* Overview

A detailed description of *uci* can be found in [Longhi '85]. Therefore, we will only briefly describe the procedure for the smoothing of a polyhedral object.

**Tessellation of the Polyhedral Object.** The current implementation of *uci* works only with triangular and quadrilateral faces. The module *ugtess* (see section 3) can be used to triangulate the object or split it into convex pieces. In the latter case, faces with more than four edges could be produced. These faces will then have to be tessellated manually in the source file. The option to *ugtess* to tessellate the polygon into triangular and quadrilateral faces *only* will be introduced shortly.

The symmetric version of *ugtess*, as mentioned in Section 3.4, can produce new vertices in the middle of the original edges. In this case, *uci* finds the two vertices, between which the new vertex is located, computes vertex normals there and constructs the Be´zier curve between them. The curve is then evaluated for the parameter value of 0.5 and the result is assigned to the new vertex as its corrected coordinates. This correction takes place for all the new vertices before the usual procedure for curved edges computations is invoked. This approach is reasonable, because the surface will pass through the "corrected" new vertex, as it would, if the new vertex were not there.

**Vertex Normals**. A normal is assigned to each vertex of the polyhedral object. It is computed as a weighed average of all normals of the faces, meeting at this vertex. The weight factor is the angle between the two edges, that use this vertex. The normal computation based on this weighting method is *tessellation-independent.*

**Curved Edges**. Cubic Be´zier curves are then constructed. The Be´zier points are found by projecting the original edge onto the tangent plane of the vertex (this plane is perpendicular to the vertex normal). The distance between the vertex and the Be´zier point is equal to the one third of the length of the edge. This does not always give satisfactory results, and other methods are being studied.

Alternatively, the user can specify the Be´zier points for the curve, joining two vertices, in the source file. This gives the designer more flexibility in changing the shape of the resulting smooth object.

Sometimes it is desirable to have a sharp edge or a straight line in the surface. For this reason, *uci* has four different types od edges: curved border, straight border, curved edge, and straight edge. All these edge types can be specified in the source file. $G^1$ continuity is only enforced across borders.

**Patch Construction.** Once the cubic (or straight) boundaries (or edges) are defined, the patches are constructed to represent a smooth surface, unless the face is flagged to be flat. Quadrilateral faces are represented with quadrilateral Gregory patches, or with five quadrilateral Bézier patches, while triangular faces are represented with triangular Gregory patches, or with three triangular Bézier patches. By default, Gregory patches are constructed; the **-b** option can be used to obtain the Bézier representation. For each patch a quadrilateral or triangular net is then constructed for the visual representation.

## 6.2. New Features of *uci*

**Gregory and Bézier Patches.** As mentioned above, the object can be represented as a union of Gregory or Bézier patches. If the Bézier option is specified, each triangular face is represented with three triangular patches, while the quadrilateral patch is represented with five quadrilateral patches, as shown in Fig. 5.3b. The labelings of the Gregory patches are shown in Fig. 6.1. Bézier patch labelings are shown in Fig. 6.2.

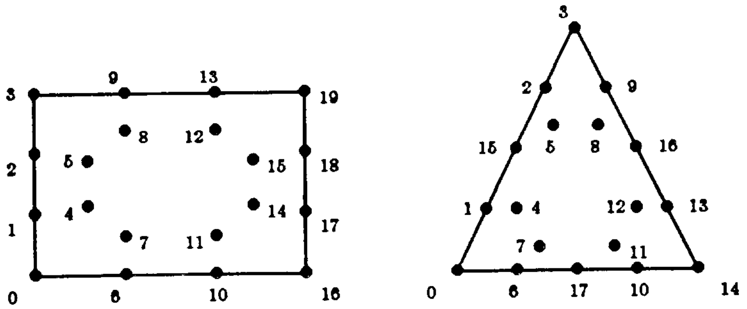


**Figure 6.1.** *Gregory patch labelings.*

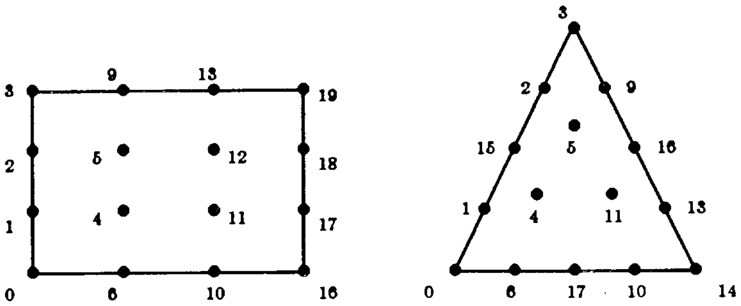


**Figure 6.2.** *Bézier patch labelings.*

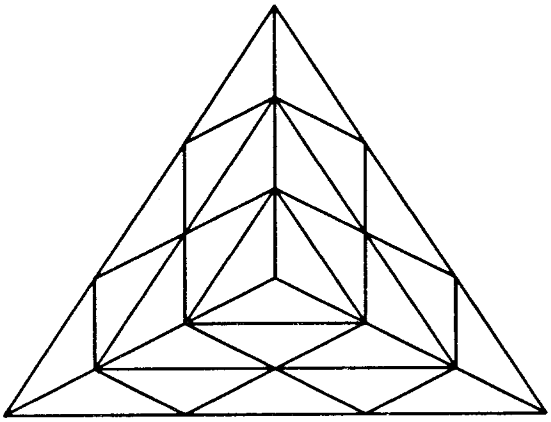


**Figure 6.3.** *The "pyramid effect".*

In our first implementation, we constructed the triangular net, used for the visual representation, on each of the subtriangles separately. However, the fact that the central angle in these subtriangles is almost always obtuse, created the illusion of an extra vertex, like a tip of a pyramid (Fig. 6.3). To get rid of this unpleasant effect, we construct the net in the parameter space of the big triangle, and compute the coordinates of the relevant point in the parameter space of the proper subtriangle for evaluation.

**Parameter space mappings.** Suppose that some point $P$ has the barycentric coordinates $(r,s,t)$ in the parameter space of the big triangle and the barycentric coordinates $(u,v,w)$ in the parameter space of the subtriangle it lies in (Fig. 6.4a). Then it easily follows that $P$ has the following representations in the parameter space of the subtriangle it is in:

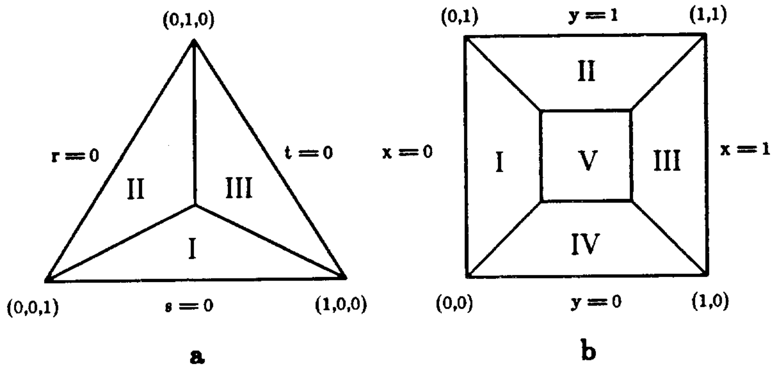


**Figure 6.4.** *Parameter space regions.*

I. $u = r - s,\ v = 3s,\ w = t - s,\quad [s < r,\ s < t];$

II. $u = 3r,\ v = s - r,\ w = t - r,\quad [r < s,\ r < t];$

III. $u = r - t,\ v = s - t,\ w = 3t,\quad [t < r,\ t < s].$

Analogously, for the quadrilateral subdivision, suppose that some point $P$ has the coordinates $(x,y)$ in the parameter space of the big quadrilateral. Some mathematics shows that $P$ can represented as follows in the parameter space $(u,v)$ of the subquadrilateral it is in (Fig. 6.4b):

I. $u = 3x,\ v = \dfrac{y - x}{1 - 2x},\quad [x < y,\ x + y < 1,\ x < \dfrac{1}{3}];$

II. $u = \dfrac{x + y - 1}{2y - 1},\ v = 3y - 2,\quad [x < y,\ x + y > 1,\ y > \dfrac{2}{3}];$

III. $u = 3x - 2,\ v = \dfrac{x + y - 1}{2x - 1},\quad [x > y,\ x + y > 1,\ x > \dfrac{2}{3}];$

IV. $u = \dfrac{x - y}{1 - 2y},\ v = 3y,\quad [x > y,\ x + y < 1,\ y < \dfrac{1}{3}];$

V. $u = 3x - 1,\ v = 3y - 1,\quad [\dfrac{1}{3} < x, y < \dfrac{2}{3}].$

**Copy Function.** A very useful **copy** function has been implemented. The data structure has been changed to hold not only the current object, but its ***backup copy*** as well. Once the object is read from the source file, the backup structure and the current structure both have the same object. *uci* knows when the original structure has been changed; if the new command is issued, which requires the original object structure, it will be copied from the backup structure into the current structure *automatically*. A good example of this is using the **patch** command after the **curvedg** command. Previously, the user had to read in the whole source file again.

The **copy** command itself could be used to copy the last-modified structure into the backup structure, or vise versa. The latter operation (**-r** option) is very useful for restoring the original structure after an undesired change.

## ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Carlo H. Séquin, without whose enthusiastic support this work would not have been possible.

Lucia Longhi helped me to understand the code and the general design of the previous *uci* version.

I would also like to thank Prof. Brian A. Barsky, who has agreed to review this report.

## Appendix 1

## UNIGRAFIX Manual Page for *ugtess*

### NAME

ugtess — tessellate faces into convex polygons or triangles

### SYNOPSIS

**ugtess** [ -t ] [ -s ] < oldobject > newobject

### DESCRIPTION

*ugtess* is a filter that *tessellates* the faces of an arbitrary unigrafix object into convex polygons.

–t The faces are *triangulated*.

–s Each face is tessellated *symmetrically* with respect to a symmetric axis if the face has one. New vertices may be introduced on the symmetry axis; the new vertex between vertices *v1* and *v2* will be assigned the name #_*v1*_*v2*.

### EXAMPLE

```
cat $(LIB)/dodeca illumF | ugshrink -H -f 0.2 | ugtess | ugplot -ed -7 2 -5 -sa -dw -sy 3 -sx 3
cat $(LIB)/dodeca illumF | ugshrink -H -f 0.2 | ugtess -t | ugplot -ed -7 2 -5 -sa -dw -sy 3 -sx 3
```

### FILES

~ug/bin/ugtess

### SEE ALSO

ugext (UG), ugfrac (UG), ugfreq (UG)

### BUGS

It is not recommended that vertex names in the object begin with #_, if you intend to use **–s** option.

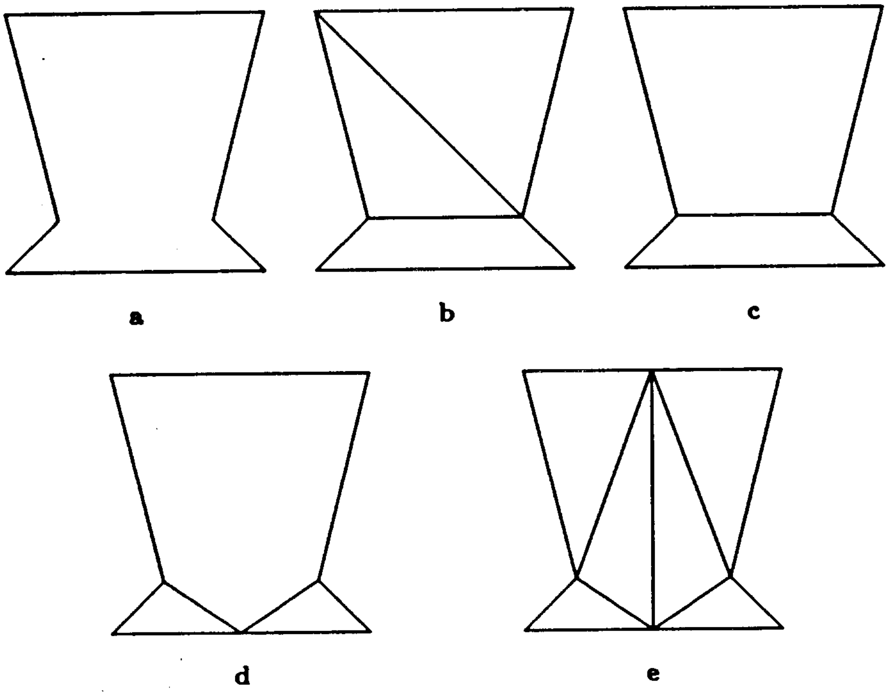
### AUTHORS

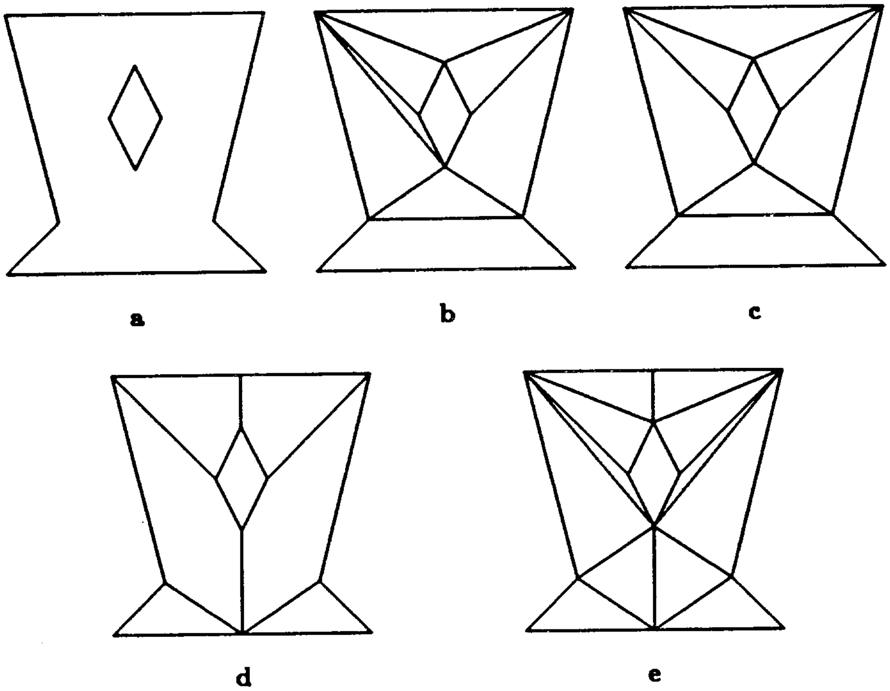
Ziv Gigus, Lucia Longhi, Leon Shirman

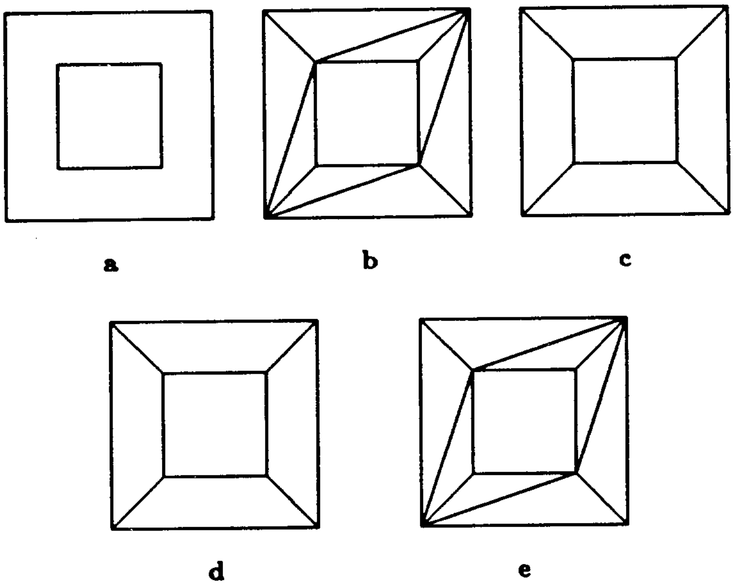
## Appendix 2

### Examples of Tessellations, Produced by *ugtess*

Figures **a** represent the original polygon; **b** is the same polygon, decomposed with old *ugtess*; **c** is the polygon with redundant edges removed; finally, **d** is the final symmetrically tessellated polygon and **e** is the triangulated polygon. Note that sometimes just removing redundant edges produces better results!




a
b
c
d
e


a
b
c
d
e

## Appendix 3

### Chiyokura's Equations

Please refer to Fig. 4.1. Vectors $\mathbf{P}_0$ and $\mathbf{P}_3$ are unit vectors and are defined as follows:

$$\mathbf{P}_0 = \frac{\mathbf{T}_0 - \mathbf{R}_0}{|\mathbf{T}_0 - \mathbf{R}_0|}$$

$$\mathbf{P}_3 = \frac{\mathbf{T}_3 - \mathbf{R}_3}{|\mathbf{T}_3 - \mathbf{R}_3|}$$

Vectors $\mathbf{P}_1$ and $\mathbf{P}_2$ are linearly interpolated between $\mathbf{P}_0$ and $\mathbf{P}_3$:

$$\mathbf{P}_1 = \frac{2}{3}\mathbf{P}_0 + \frac{1}{3}\mathbf{P}_3, \quad \mathbf{P}_2 = \frac{1}{3}\mathbf{P}_0 + \frac{2}{3}\mathbf{P}_3.$$

Then

$$\mathbf{T}_1 = \frac{(k_1 - k_0)}{3}\mathbf{P}_0 + k_0\,\mathbf{P}_1 + \frac{2}{3}h_0\,\mathbf{S}_1 + \frac{h_1}{3}\mathbf{S}_0$$

$$\mathbf{T}_2 = k_1\,\mathbf{P}_2 - \frac{(k_1 - k_0)}{3}\mathbf{P}_3 + \frac{h_0}{3}\mathbf{S}_2 + \frac{2}{3}h_1\,\mathbf{S}_1$$

where the coefficients $k_0$, $k_1$, $h_0$, $h_1$ can be found from the following equations:

$$\mathbf{T}_0 = k_0\,\mathbf{P}_0 + h_0\,\mathbf{S}_0, \quad \mathbf{T}_3 = k_1\,\mathbf{P}_3 + h_1\,\mathbf{S}_2.$$

## Appendix 4

### Solutions to the Triangular Patch Subdivision Equations (Center of Gravity Case)

The following expressions are valid only if $S_1$, $S_2$, $S_3$, $Z$ are centers of gravity of corresponding triangles, i.e.

$$\mathbf{S}_1 = \frac{1}{3}\mathbf{T}_1 + \frac{1}{3}\mathbf{C}_1 + \frac{1}{3}\mathbf{B}_1$$

$$\mathbf{S}_2 = \frac{1}{3}\mathbf{T}_2 + \frac{1}{3}\mathbf{C}_2 + \frac{1}{3}\mathbf{D}_2$$

$$\mathbf{S}_3 = \frac{1}{3}\mathbf{T}_3 + \frac{1}{3}\mathbf{B}_2 + \frac{1}{3}\mathbf{D}_1$$

$$\mathbf{Z} = \frac{1}{3}\mathbf{P}_1 + \frac{1}{3}\mathbf{P}_2 + \frac{1}{3}\mathbf{P}_3$$

Then

$$\mathbf{P}_1 = \frac{1}{6}(\mathbf{T}_1 - \mathbf{S}_1 + 3\,\mathbf{L}_{13} + 3\,\mathbf{L}_{12})$$

$$\mathbf{P}_2 = \frac{1}{6}(\mathbf{T}_2 - \mathbf{S}_2 + 3\,\mathbf{K}_{23} + 3\,\mathbf{K}_{12})$$

$$\mathbf{P}_3 = \frac{1}{6}(\mathbf{T}_3 - \mathbf{S}_3 + 3\,\mathbf{M}_{23} + 3\,\mathbf{M}_{13})$$

$$\mathbf{N}_{12} = \frac{1}{3}(-\mathbf{S}_1 - \mathbf{S}_2 + \mathbf{S}_3 + 3\,\mathbf{P}_1 + 3\,\mathbf{P}_2 - 2\,\mathbf{P}_3)$$

$$\mathbf{N}_{13} = \frac{1}{3}(-\mathbf{S}_1 + \mathbf{S}_2 - \mathbf{S}_3 + 3\,\mathbf{P}_1 - 2\,\mathbf{P}_2 + 3\,\mathbf{P}_3)$$

$$\mathbf{N}_{23} = \frac{1}{3}(\mathbf{S}_1 - \mathbf{S}_2 - \mathbf{S}_3 - 2\,\mathbf{P}_1 + 3\,\mathbf{P}_2 + 3\,\mathbf{P}_3)$$

## Appendix 5

## Uci Images

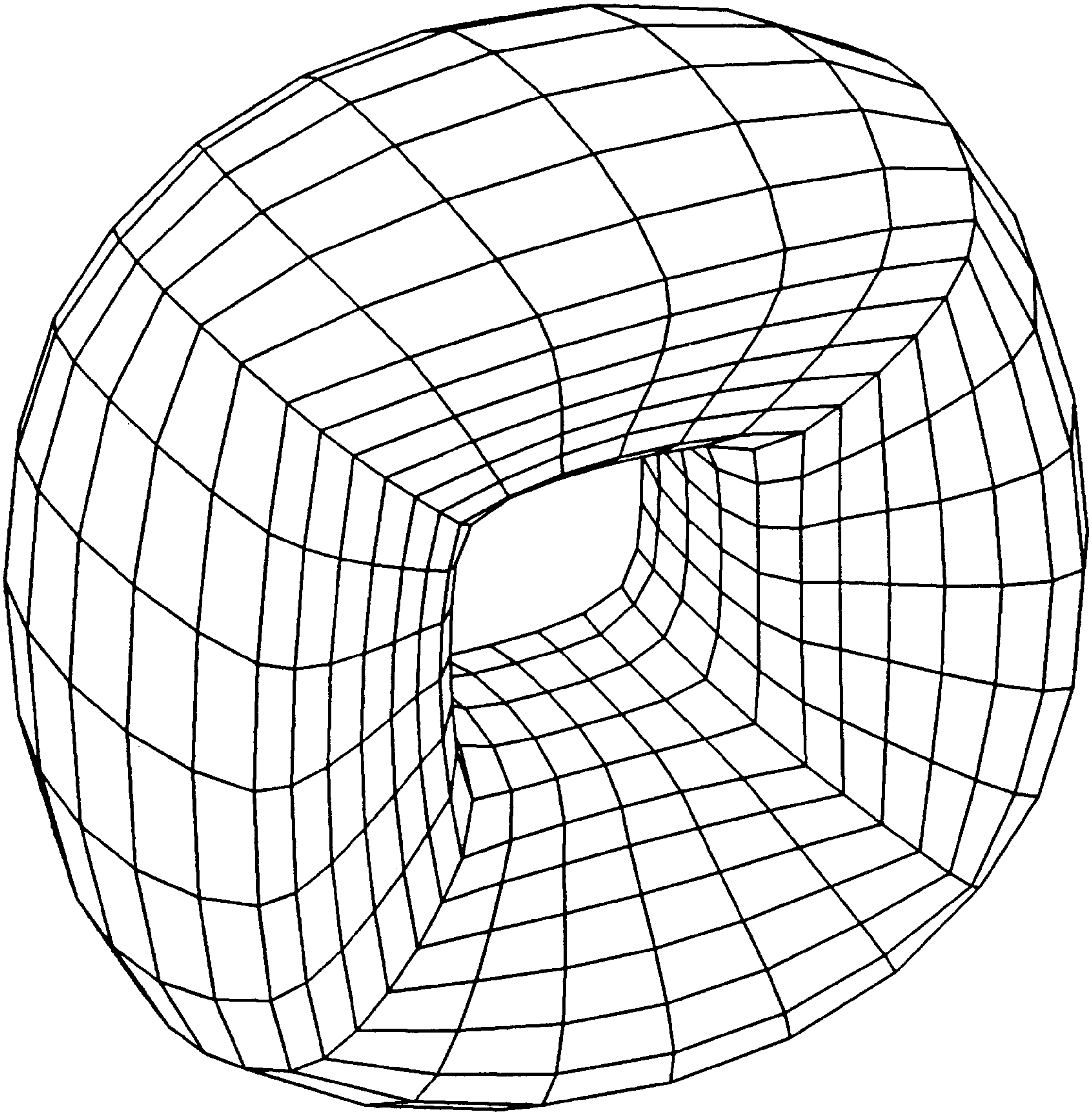


**Figure 1.** *A smooth object, represented with quadrilateral patches.*

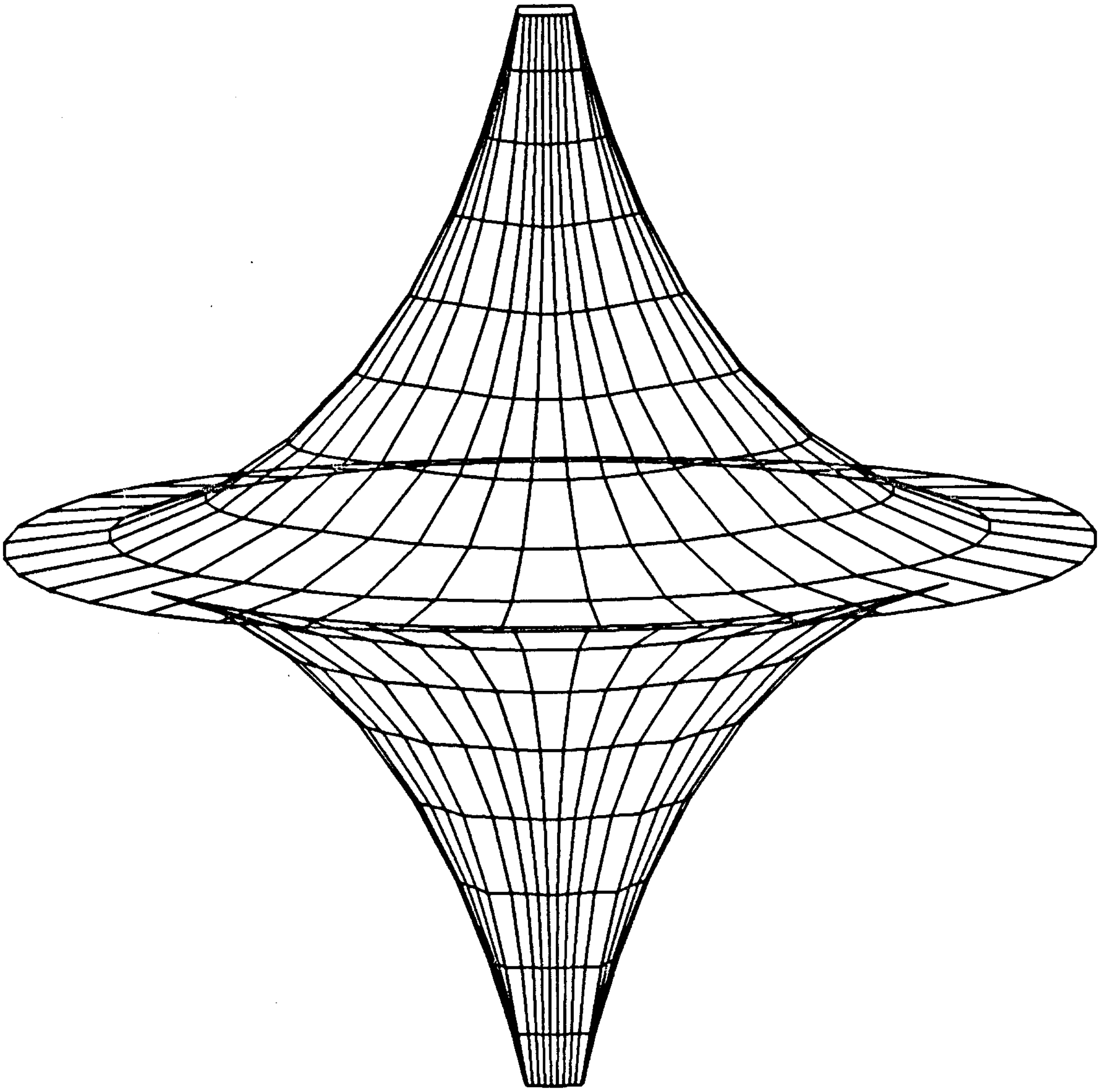


**Figure 2.** *An object with sharp edges.*

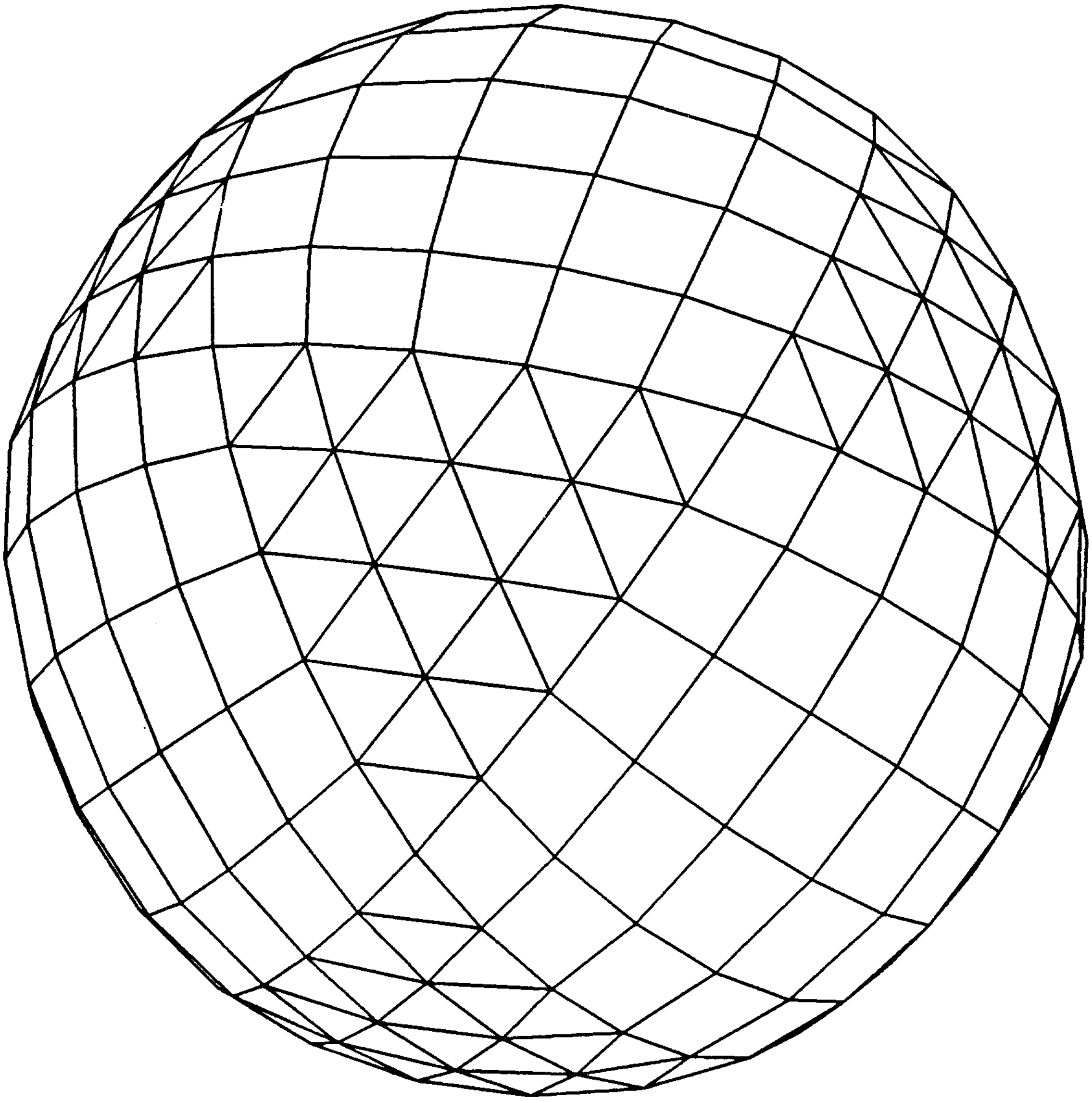


**Figure 3.** *An object, represented with quadrilateral and triangular patches.*

# REFERENCES

[Barnhill et al '74] R. Barnhill and R. Riesenfeld eds., *Computer Aided Geometric Design,* Academic Press, New York, 1974.

[Barnhill et al '83] R. Barnhill and W. Böhm eds., *Surfaces in Computer Aided Geometric Design,* North Holland, Amsterdam, 1983

[Bartels et al '86] R. Bartels, J. Beatty, and B. Barsky, *An Introduction to the Use of Splines in Computer Graphics,* Morgan & Kaufmann Publishers, Inc., Los Altos, California, 1987.

[Bogen et al '77] R. Bogen, J. Golden, M. Genesereth, and A. Doohovsky, *MACSYMA Reference Manual,* Massachusetts Institute of Technology, 1977.

[Chiyokura et al '83] H. Chiyokura and F. Kimura, *Design of Solids with Free-Form Surfaces,* Computer Graphics, Vol. 17, No. 3, pp. 289-298, 1983.

[Chiyokura '86] H. Chiyokura, *Localized Surface Interpolation Method for Irregular Meshes,* to appear in Computer Graphics, 1986.

[Coons '64] S. Coons, *Surfaces for Computer Aided Design,* Design Division, Mechanical Engineering Department, M.I.T., Cambridge, Massachusetts, 1964.

[Farin '82] G. Farin, *A Construction for Visual Continuity of Polynomial Surface Patches,* Computer Graphics and Image Processing, Vol. 20, pp. 272-282, 1982.

[Farin '83] G. Farin, *Smooth Interpolation to Scattered 3D Data,* in [Barnhill et al '83], 1983.

[Farin '86] G. Farin, *Triangular Bezier-Bernstein Patches,* Computer Aided Geometric Design, Vol. 3, No. 2, North Holland, 1986.

[Faux et al '79] I. Faux and M. Pratt, *Computational Geometry for Design and Manufacture,* Ellis Horwood Ltd, 1979

[Filip '85] D. Filip, *Practical Considerations for Triangular Patch Surfaces,* Master's Thesis, University of California, Berkeley, California, 1985.

[Foley et al '82] J. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics,* Addison-Wesley, 1982.

[Gigus '85] Z. Gigus, *Binary Space Partitioning for Previewing UNIGRAFIX Scenes,* Tech. Report No. UCB/CSD 86/280, Computer Science Division, University of California, Berkeley, California, 1985.

[Gregory '74] J. Gregory, *Smooth Interpolation Without Twist Constraints,* in [Barnhill et al '74], 1974.

[Kahmann '83] J. Kahmann, *Continuity of Curvature Between Adjacent Bézier Patches,* in [Barnhill et al '83], 1983.

[Longhi '85] L. Longhi, *Interpolating Patches Between Cubic Boundaries,* Master's Project Report, Computer Science Division, University of California, Berkeley, California, 1985.

[Séquin et al '84] C. Séquin, M. Segal and P. Wensley, *UNIGRAFIX 2.0 User's Manual and Tutorial,* Tech. Report No. UCB/CSD 83/161, Computer Science Division, University of California, Berkeley, California, 1984.

[Séquin '85a] C. Séquin, *The Berkeley UNIGRAFIX Tools, Version 2.5,* Tech. Report No. UCB/CSD 86/281, Computer Science Division, University of California, Berkeley, California, 1985.

[Séquin '85b] C. Séquin, *More... Creative Geometric Modeling,* Tech. Report No. UCB/CSD 86/278, Computer Science Division, University of California, Berkeley, California, 1985.

[Siegel '85] H. Siegel, *Jessie: An Interactive Editor for UNIGRAFIX,* Tech. Report No. UCB/CSD 86/279, Computer Science Division, University of California, Berkeley, California, 1985.