# INTERPOLATING PATCHES BETWEEN CUBIC BOUNDARIES

*Lucia Longhi*

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

*ABSTRACT*

We describe the theoretical frame for a method of creating and describing rounded objects of arbitrary topology in CAD, and its implementation for UNIGRAFIX, a polygon-based modeler developed at UC Berkeley that generates black-and-white, smooth-shaded images on several output devices. The mathematical foundation for building triangular patches interpolating cubic edges and blending with geometric continuity is given, and various approaches are discussed. To represent curvature information, we extended the UNIGRAFIX language to **UniCubix**, and we implemented **uci**, an interactive shell that interprets a **UniCubix** description and converts it into UNIGRAFIX wireframes or polyhedral nets that approximate curved patches. **Uci** also provides a prototype of a global smoothing operation, that takes a polyhedral object of arbitrary topology and creates the **UniCubix** representation of a smooth object interpolating the input vertices.

*Support for this work was provided in part by the Semiconductor Research Cooperative*

## 1. INTRODUCTION

A widespread method for constructing smooth free-form surfaces in Computer Aided Design and Manufacture (CAD/CAM) consists of interpolating surface patches to a skeleton of polynomial curves, in such a way that neighbor patches meet with coincident tangent planes along their boundaries [Barnhill et al '74; Barnhill et al '83]. Those polynomial curves can be specified by the designer as important profiles of the object, or generated by the design system from polyhedral grid lines that merely determine the topology of the object.

The geometrical basis for one such system was presented by G. Farin [Farin '82b]. In his approach, the designer inputs points belonging to the surface of the object, a tangent plane on each point, and a triangulated mesh to specify its topology. Farin's algorithm then joins the specified vertices with cubic curves tangent to the given planes at their endpoints, builds quartic Bezier triangular patches (see Section 2) interpolating those curves, subdivides each patch into 3 quartic subpatches, and adjusts subpatches to meet smoothly across their boundaries.

Another approach by H. Chiyokura and F. Kimura [Chiyokura et al '83], implemented in the MODIF solid modeler at Tokyo University, allows the designer to specify cubic curves bounding regions of 3 to 6 sides. The system subdivides these into quadrilateral subregions by constructing auxiliary cubic curves. Gregory patches (see Section 2.2) are used to interpolate those curves. They are computed independently of their neighbors, since each patch is built to match tangent plane information that is derived exclusively from the cubic boundary curve.

At the cost of more restricted input freedom and less locality, Farin's method produces polynomial patches that can be fed into a wealth of algorithms (for subdivision, rendering, hidden-features elimination, intersection calculations, etc) that are based on polynomial representation. Chiyokura's method permits the design of a wider variety of shapes and its locality makes it suitable for interactive design, but it produces non-polynomial surfaces. In both methods patches meet with continuity of tangent planes.

Our research is an attempt to combine the best of both approaches, aiming at a method that would allow a net of arbitrary cubic curves to be specified, and would build individual interpolating polynomial patches based exclusively on this boundary information.

We concentrated our studies on triangular patches because triangular meshes can describe objects of arbitrary topology. An arbitrary polyhedral sketch of the object may thus have to be preprocessed. There are many algorithms that triangulate arbitrary polygons [Cavendish '74, Garey *et_al* '78, Lewis *et_al* '79].

Bézier patches are more constrained than Gregory patches. We analyzed under what conditions quartic Bézier patches can produce the required tangent plane continuity along the cubic boundaries. Both Farin and Chiyokura start by exploring the general conditions for

geometric continuity between adjacent patches, but then make some special assumptions about the behavior of the cross-boundary derivative in order to keep the computation manageable. One goal of our research was to go as far as possible with making as few as possible extra assumptions about the cross-boundary derivative. The results of our exploration in that area are discussed in Section 3.

Another goal of this project was to implement a prototype of an interactive module within the context of UNIGRAFIX for the design of interpolating free-form surfaces. Since UNIGRAFIX describes only polyhedral objects, we extended that language in two ways: one to permit the designer to specify which faces or edges should be curved or linear; another, called **UniCubix** , to unambiguously describe curved objects by including the specification of all its Bézier points. **UniCubix** is described in Section 4 and Appendix 3. The interactive module is **ucl** described in Section 4 and Appendix 4.

## 2. PRELIMINARIES

### 2.1. Triangular Bernstein-Bézier Patch

This section surveys relevant known results on Bézier triangles. We will use the *non-parametric* formulation, that uses a triangle as the domain for polynomials defined over it. For proofs or more details refer to [Farin '80; Farin '82b; Böhm et al '84; Filip '85].

Let T be a triangle in space with vertices $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$. A point $\mathbf{P}$ in the plane of the triangle can be uniquely represented by [Barnhill '77]

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

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

Any polynomial $p(u,v,w)$ of degree $n$ has a unique representation in terms of the basis of *bivariate Bernstein polynomials* of degree $n$

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

i.e.

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

An equivalent form without dependent variables is:

$$B^n_{i,j}(u,v) = \frac{n!}{i!j!(n-i-j)!}\, u^i v^j (1-u-v)^{n-i-j}\ ,\quad i,j \geq 0\ .$$

$$p(u,v) = \sum_{\substack{i,j \geq 0 \\ i+j \leq n}} b_{i,j} B^n_{i,j}(u,v) , \quad u+v \leq 1 , \quad u,v \geq 0 ,$$

The coefficients $b_{i,j}$, called *Bézier ordinates* possess a geometric interpretation: Figure 2.1 shows how they relate to the parameters $i,j,k = n-i-j$ for a patch of degree $n = 3$. The Bézier ordinates are plotted using the the points $(u,v) = (\frac{i}{n}, \frac{j}{n})$ for abscissae over the triangle in the ( $u$ , $v$ ) plane. The ordinates determine a polyhedral net of triangular faces, called *control polyhedron*, which models the shape of the surface patch.

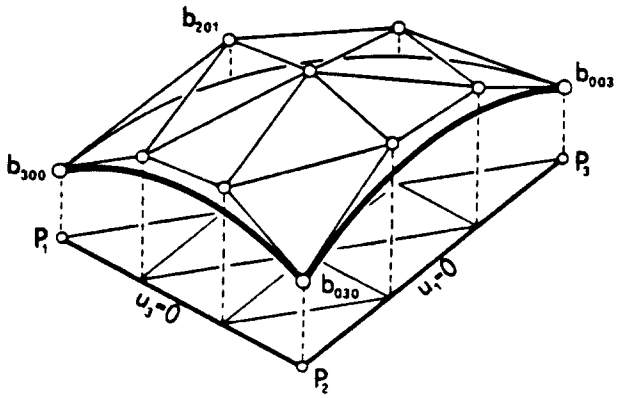



**Figure 2.1** *Cubic Triangular Bernstein-Bézier Patch*

**Degree Elevation:** Every polynomial of degree $n$ can be expressed in terms of the Bernstein-Bézier basis polynomials of degree $n+1$:

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

The $b^*_{i,j,k}$ are given by [de Casteljau '59]

$$b^*_{i,j,k} = \frac{1}{n+1} (i\, b_{i-1,j,k} + j\, b_{i,j-1,k} + k\, b_{i,j,k-1}) .$$

**Evaluation:** De Casteljau [de Casteljau '59] derived the following recursive formula to evaluate $p(\mathbf{u})$ for any point $\mathbf{u}$ with barycentric coordinates $(u,v,w)$. Defining

$$b^0_{i,j,k}(\mathbf{u}) = b_{i,j,k},$$

$$b^l_{i,j,k}(\mathbf{u}) = u\, b^{l-1}_{i+1,j,k}(\mathbf{u}) + v\, b^{l-1}_{i,j+1,k}(\mathbf{u}) + w\, b^{l-1}_{i,j,k-1}(\mathbf{u}) , \quad i+j+k = n-l ,$$

then

$$p(u,v,w) = b^{n}_{0,0,0}(\mathbf{u}) .$$

**Derivatives:** Let $\mathbf{u}(t) = (1-t)\,\mathbf{u}_1 + t\,\mathbf{u}_2$, $t$ real, be a straight line through $\mathbf{u}_1$ and $\mathbf{u}_2$, two points in the $u,v,w$ barycentric parameter plane and let $\Delta\mathbf{u} = \mathbf{u}_2 - \mathbf{u}_1$.

By differentiating $p(\mathbf{u}(t))$ with respect to $t$ and evaluating at $t = 0$ we obtain the directional derivative of $p$ at $\mathbf{u}_1$ in the direction $\Delta\mathbf{u}$. The $r^{th}$ directional derivative is given by

$$D^{r}_{\Delta\mathbf{u}}(\mathbf{u}1) = \frac{n!}{(n-r)!} \sum_{\substack{i+j+k=n-r \\ i,j,k \geq 0}} b^{r}_{i,j,k}(\Delta\mathbf{u})\, B^{n-r}_{i,j,k}(\mathbf{u}1)$$

where the $b^{r}_{i,j,k}()$ are the recursively defined coefficients for de Casteljau's evaluation.

In the particular case of $t$ moving on an isoparametric line, say $t = u$ and $v$ constant, we have $\Delta\mathbf{u} = (1,0,-1)$ and $b^{l}_{i,j,k}(\Delta\mathbf{u}) = b^{l-1}_{i+1,j,k}(\Delta\mathbf{u}) - b^{l-1}_{i,j,k+1}(\Delta\mathbf{u})$. This observation will be used frequently when searching for conditions of $G^1$ continuity across patch boundaries in Section 3.

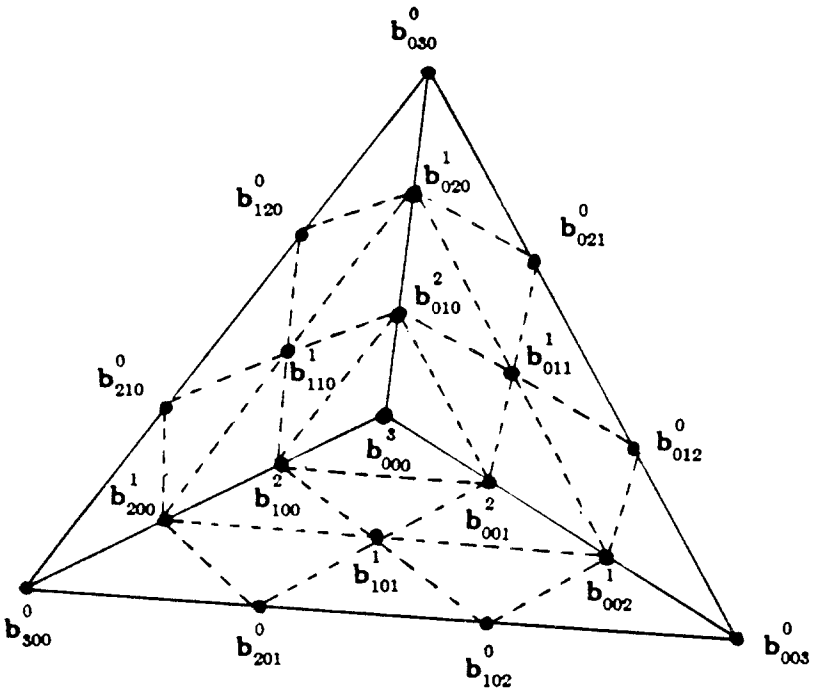



**Figure 2.2** *Bézier ordinates of a subdivided cubic patch*

**Subdivision:** This is a frequently used process in Computer Aided Geometric Design, in which a patch $p$ is split into disjoint subpatches whose union represent the original one. Given $\mathbf{u}$ inside

$T$, one can subdivide $T$ into three subtriangles, and these can be regarded as new separate domains for $p$. The coefficients of the subpatches in their Berstein-Bézier expression are those generated in the de Casteljau recursive evaluation of $p(\mathbf{u}) = b^n_{0,0,0}(\mathbf{u})$. They are (see Figure 2.2)

$$\{b^l_{i,j,0}(\mathbf{u})\}\ ,\ \{b^l_{i,0,k}(\mathbf{u})\}\ ,\ \{b^l_{0,j,k}(\mathbf{u})\}\ ,\quad l = 0,1,\ldots,n\ .$$

### 2.2. Triangular Gregory Patch

Rectangular Gregory patches were introduced by Chiyokura and Kimura, [Chiyokura et al '83] as modified bicubic Bézier patches. We will present here the triangular version for quartic boundaries, following their method. We use the fourth degree to gain more flexibility in the manipulation of 6 interior control points (as opposed to 3 in the cubic case) when searching for conditions of cross-boundary continuity (Section 2.3). Cubic boundaries can anyhow be expressed as quartic curves by the degree elevation process reviewed in Section 2.1.

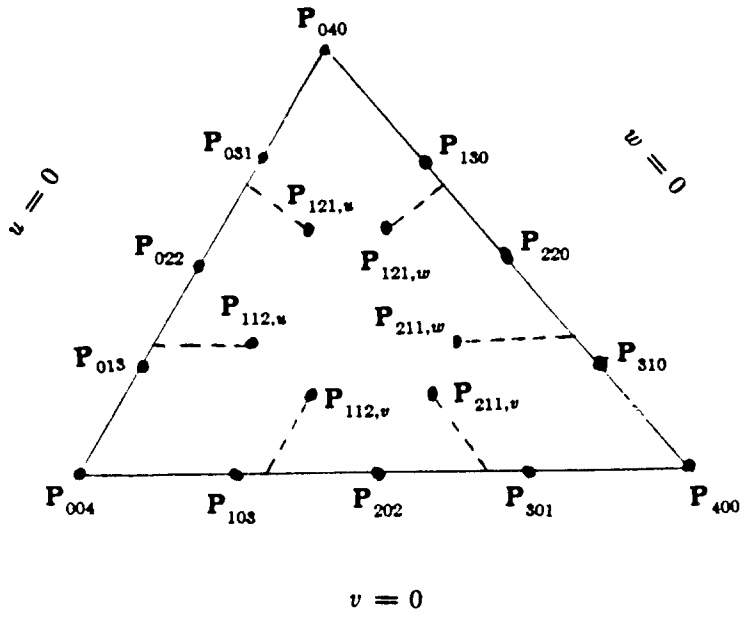



**Figure 2.3** *Triangular Gregory Patch*

Working again in barycentric coordinates, a triangular Gregory patch is determined by 18 control points (see Figure 2.3) and defined by

$$g(u,v,w) = (\ u\ E + v\ F + w\ G)^4\ P_{000}(u,v,w)\ ,\quad u,v,w \geq 0\ ,\ u + v + w = 1\ ,$$

where

$E$, $F$, $G$ are the shift operators $E\ P_{i\,j\,k} = P_{i+1\,j\,k}$ , $F\ P_{i\,j\,k} = P_{i\,j+1\,k}$ , $G\ P_{i\,j\,k} = P_{i\,j\,k+1}$ ;

$P_{ij*}(u,v,w) = P_{ij*}$ for $i = 0$ or $j = 0$ or $k = 0$, (control points at the boundary of the patch); and

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

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

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

One can think of a Gregory patch as a quartic Bézier patch whose three interior control points move along three "small" line segments. $P_{112}$ moves on a straight line segment between $P_{112,u}$ and $P_{112,v}$ according to the parameters $u$ and $v$. Similarly for $P_{121}$ and $P_{211}$ . When computing a point $g(u,v,w)$ on the patch, the closer it is to a border (for example $u = 0$), the bigger are the weights of the two control points close to that side ($P_{121,u}$ and $P_{112,u}$).

In particular, if

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

all $P_{ij*}(u,v,w)$ are constant and the Gregory patch reduces to a triangular quartic Bezïer patch.

Like Bézier patches, the Gregory patch has the *convex hull property* : all points on the patch are contained in the smallest convex polyhedron containing all its control points). This is useful in design for rough interference checks and to determine planar regions within the surface [Faux et al '79].

## 3. GEOMETRIC CONTINUITY ACROSS CUBIC BOUNDARIES

### 3.1. Matching Cross-boundary Tangents on a Single Edge

The problem of describing a notion of smoothness between patches is still open (some work in that direction can be found in [DeRose '85]). However, it is generally agreed that the minimal necessary (and often sufficient) condition is continuity of surface normal direction. This is referred to as *geometric continuity of degree 1* and denoted by $G^1$. Since the patches we are dealing with are defined in terms of control points, the key issue is to find the conditions for control points on either side of a curved edge that guarantee that the two patches meet at the edge with $G^1$ continuity. If they both use the given curve for a boundary, then tangent continuity in the

direction of the boundary is automatically guaranteed. If they are also equally parametrized from the point of view of both patches, then even the derivative *along* the curve will be the same for both patches. Thus we only have to find conditions for the *cross-boundary* derivative, or at least for the behavior of the cross-boundary tangent (from now on referred to as CBT).

To deal with this derivative in full generality turns out to be rather difficult. So most workers in the field make some special assumption about it to make the computations manageable. We first set out to see whether we could avoid making any arbitrary assumptions about the CBT and then use the extra freedom to achieve blending with Bézier patches rather than Gregory patches between arbitrary cubic boundary curves.

### 3.1.1. *Some Basic Insights*

First we formulate the $G^1$ continuity constraints in most generality. Let $\Phi$ and $\Psi$ be two surface patches with a common boundary curve $\Gamma(v)$, and let $[D\Gamma](v)$ denote its tangent vector. Let $[D_1\Phi](v)$ denote a cross-boundary derivative of $\Phi$ at $\Gamma(v)$, i.e., $[D_1\Phi](v)$ lies in the tangent plane of $\Phi$ at $\Gamma(v)$ and $[D_1\Phi](v)$ is not collinear with $[D\Gamma](v)$. Analogously, we define a cross-boundary derivative $[D_2\Psi](v)$. The necessary and sufficient condition for $G_1$ continuity is coplanarity of all three vectors [Farin '82a]:

$$\det\left( [D_1\Phi](v)\,,\, [D_2\Psi](v)\,,\, [D\Gamma](v) \right) = 0\,.$$

In the following we concentrate on quartic Bézier patches whose boundary curves are cubic. Our basic construction is the same as in [Farin '82b].

It is interesting to note that the same construction can be applied to quartic triangular patches and to bicubic quadrilateral patches [Farin '82b], thus our results are useful for neighboring patches of any combinations of these two kinds of patches (quartic triangle - quartic triangle, bicubic rectangle - bicubic rectangle or quartic triangle - bicubic rectangle).

For the computation of the cross-boundary derivatives in triangles we will use the "radial" directional derivative [Farin '82a]

$$[D\Phi](v) = (1-v)\left( [D_u\Phi](v) - [D_v\Phi](v) \right) + v\left([D_w\Phi](v) - [D_v\Phi](v)\right).$$

The formulas for derivatives of Bézier patches (see Section 2.1) can then be expressed in terms of vectors joining control points. With the nomenclature in Figure 3.1,

$$[D\Phi](v) = 4\sum_{i=0}^{3}\mathbf{T}_i\,B_i^3(v)\,,\quad [D\Psi](v) = 4\sum_{i=0}^{3}\mathbf{R}_i\,B_i^3(v)\,,\quad [D_v\Gamma](v) = 3\sum_{i=0}^{2}\mathbf{S}_i\,B_i^2(v)\,,$$

and the above determinant is

$$\det\left[\, 4\sum_{i=0}^{3}\mathbf{R}_iB_i^3(v)\,,\, 4\sum_{i=0}^{3}\mathbf{T}_iB_i^3(v)\,,\, 3\sum_{i=1}^{3}\mathbf{S}_iB_i^2(v)\,\right] = 0.$$

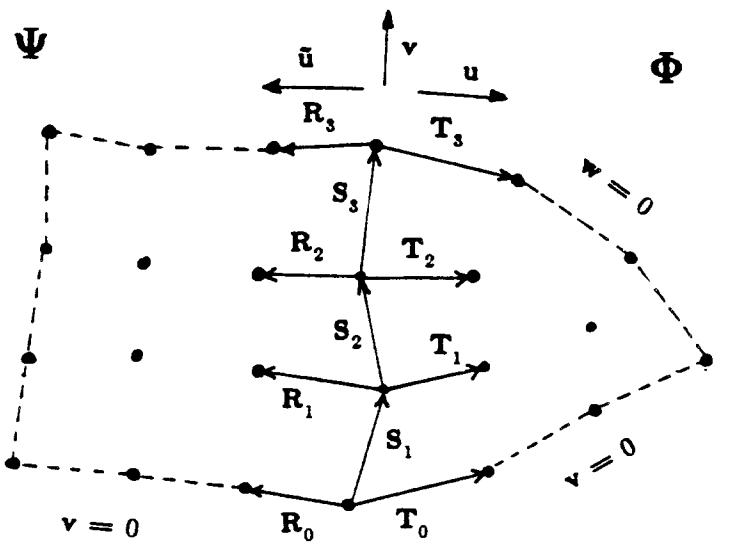



**Figure 3.1** *Nomenclature for neighbor patches*

which is an 8-degree polynomial in $v$ equalized to 0, (the two CBT derivatives are of degree 3, and the derivative along the curve is quadratic). Thus by sorting out the terms of the same order with the aid of the symbolic manipulator MACSYMA [Bogen *et_al* '77; Fateman '82], we obtain 9 separate equations relating the vectors in Figure 3.1 (see Appendix 1).

With proper substitution the first and last equations are the requirement of coplanarity of all the first Bézier segments emerging from a vertex. For the other 7 constraints we have not found equally simple geometrical interpretations.

To achieve locality, we have explored different approaches. First we used auxiliary patches in the style of [Chiyokura *et_al* '83] for the purpose of specifying the CBT behavior. An auxiliary patch will be determined on one side of a cubic edge, exclusively based on information contained in the boundary. "Real" patches will then be matched to these auxiliary patches with equal tangent planes on the boundary curve. Thus the real patches will meet with $G^1$ continuity. Our second approach is to define the behavior of the CBT with the binormal of the cubic boundary curve.

In this situation there are only six degrees of freedom per edge for the two interior Bézier points in the real patch. So the seventh constraint is some restriction that must be obeyed when selecting the auxiliary patch on the edge.

It should be pointed out here, that this approach can be generalized to higher orders. If the order of both patches is incremented by one, the order of all product terms from the determinants increases by 3. But at the same time we have to determine the coordinates of an additional Bézier point. This still leaves us with one extra constraint.

The ennumerations of the degrees of freedom suggests that some relationship must be fulfilled in the auxiliary patch that specifies the CBT (i.e. the "$(3n+1)^{th}$ constraint", where n is

the number of Bézier points to be determined in one of the patches along this boundary). We don't yet understand what the nature of this limitation is in the most general case.

**3.1.2.** *Previous Attempts.*

Dealing with this determinant formulation leads to unmanageably large equations that not even MACSYMA can solve - it simply runs out of memory. To simplify the problem, different people have used different ways to formulate the problem of matching the CBTs and thereby obtaine $G^1$ continuity.

1) Farin [Farin '82], for instance, specifies the coincidence of the tangent planes at the boundary by introducing some simple coefficient functions and setting the following weighted sum to zero:

$$\mu\,[D_1\Phi](v)+\alpha\,[D_2\Psi](v)+\{\,(1-v)\,\lambda_0 + v\,\lambda_1\,\}[D\Gamma](v) = 0.$$

After setting $\alpha = 1$, there remains a single linear weighting function and a simple constant to relate the three relevant partial derivatives. Unfortunately this simplification leads to some coupling conditions that put an additional constraint onto the choice of Bézier points for the boundary curves, so these can not be chosen completely freely. The ratio of the areas of the triangles must match:

$$\frac{\mathbf{area}(\mathbf{R}_0\,\mathbf{S}_1)}{\mathbf{area}(\mathbf{T}_0\,\mathbf{S}_1)} = \frac{\mathbf{area}(\mathbf{R}_3\,\mathbf{S}_3)}{\mathbf{area}(\mathbf{T}_3\,\mathbf{S}_3)}$$

where $\mathbf{area}(\mathbf{A}\,\mathbf{S})$ is the area of the triangle with 2 sides equal to $\mathbf{A}$ and $\mathbf{S}$.

2) Chiyokura [Chiyokura *et_al* '83], on the other hand, demands that the CBT for the auxiliary patch $A$ be only quadratic, and then introduces two linear weighting functions $k(v)$ and $h(v)$ to represent the $G^1$ condition:

$$[D_u A](v) = k(v)\,[D_u\Psi](v) + h(v)\,[D_v\Psi](v)\,.$$

The coefficients of these functions are determined from the planarity conditions at the endpoint vertices of the cubic curve. The two real patches are then matched individually against such standard auxiliary patches.

This formulation allows completely free choice of the boundary curve and always finds two Bézier points.

**3.1.3.** *Our Various Experiments Starting from the Determinant Formulation*

We started from the most general formulation of the CBT continuity condition expressed with the determinants in Section 3.1.1. We tried to make some clever guesses about how we should restrict the general CBT behavior in order to make the system solvable. Following the concept of an auxiliary patch, we tried to formulate the constraint on the CBT behavior in terms of a restriction on the interior Bézier points of the real patch. These were some cases we tried:

1) Parallelogram case

We tried a CBT that varies as a cubic with zero derivatives at both ends. We can achieve this by setting the vectors $\mathbf{T}_1 = \mathbf{T}_0$, $\mathbf{T}_2 = \mathbf{T}_3$ in the auxiliary patches. This condition on $\mathbf{T}_1$, $\mathbf{T}_2$ did nothing to simplify the equations.

When this substitution is made for the real patch too, i.e. also $\mathbf{R}_1 = \mathbf{R}_0$ and $\mathbf{R}_2 = \mathbf{R}_3$, in the 9 equations arising from determinant formulation of $G^1$ continuity across the edge, then one of the equations becomes an extra restriction on the Bézier points defining the boundary curves:

$$det(\mathbf{R}_3\ \mathbf{S}_2\ \mathbf{T}_0) + det(\mathbf{R}_0\ \mathbf{S}_2\ \mathbf{T}_3) = 0\ .$$

However, applied symmetrically to all edges coming together in a vertex, this is equivalent to saying that the quadrilateral determined by an original vertex, the first Bézier points on the two boundary curves, and the nearest in-face Bézier point is a parallelogram in all cases. Thus the two Gregory points automatically merge and we get a real Bézier triangle.

The above extra constraint may be difficult to satisfy and would destroy the locality of the definitions of the curved edges. Furthermore, even if the resulting constraints on the curved edges turn out to be easy to fulfill, Chiyokura has shown that this approach may not be very desirable since it leads to very flat areas around the original vertices.

2) Lower degree for CBT

Following Chiyokura's approach, we can demand that the CBT function of the given patch be one order lower than possible, i.e. the same order as the along-the-boundary derivative. For our special case this implies a quadratic function only, whereas in general a quartic triangular patch or a bicubic patch could give a cubic CBT function. If we start with this assumption, then all the product terms in the determinants have one degree less, and thus there is one constraint less. This leaves 6 constraints for the determination of two Bézier points. We have verified that if we plug the condition $\mathbf{T}_0 - 3\ \mathbf{T}_1 + 3\ \mathbf{T}_2 - \mathbf{T}_3 = 0$ into our system of equations, indeed one of the equations disappears. Thus with a free choice of one Bézier point in the auxiliary patch ($\mathbf{T}_1$ or $\mathbf{T}_2$), we get exactly six constraints for the two Bézier points in the real patch. This implies that limiting the degree of the CBT is a reasonable way to standardize the behavior of the CBT.

3) Linear interpolation

We can also make the additional assumption made by Chiyokura, that the derivatives in the auxiliary patch are linearly interpolated along the edge. This is equivalent to setting $\mathbf{T}_1 = \frac{2}{3}\mathbf{T}_0 + \frac{1}{3}\mathbf{T}_3$ and $\mathbf{T}_2 = \frac{2}{3}\mathbf{T}_3 + \frac{1}{3}\mathbf{T}_0$. The corresponding equations are given in Appendix 2.

This is the approach actually used in our first prototype implementation (see Section 4). The Bézier points or the real patch are then determined by the following formulas [Chiyokura *et_al* '83]:

$$\mathbf{R}_1 = \frac{(k_1 - k_0)}{3}\mathbf{T}_0 + k_0\,\mathbf{T}_1 + \frac{2}{3}h_0\,\mathbf{S}_2 + \frac{h_1}{3}\mathbf{S}_1$$

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

where

$$\mathbf{R}_0 = k_0\,\mathbf{T}_0 + h_0\mathbf{S}_1$$

$$\mathbf{R}_3 = k_1\,\mathbf{T}_3 + h_1\mathbf{S}_3$$

With that, the auxiliary patch and the CBT are entirely determined.

In this process we have used up 6 degrees of freedom, one more than we originally had. However it turns out that two constraints are automatically fulfilled by our choice, and we have thus only 5 constraints left. Thus we could carry one degree of freedom with the setting of the two interior Bézier points, which might be exploited later when we need to make the conditions from several boundary curves compatible. Unfortunately, even the remaining 5 constraints are hard to interpret since they involve rather strange linear combinations of all possible determinants. They are too complicated to be used directly in a closed form solution.

It should be pointed out, that a second order derivative in the CBT of the auxiliary patch does not imply that the CBT of the patch on the other side is also quadratic. (There would then be more constraints than remaining degrees of freedom.)

4) A weaker parametrization

We could require only that $\mathbf{T}_0 - a\,\mathbf{T}_1 + b\,\mathbf{T}_2 - \mathbf{T}_3 = 0$. This is inspired from the above constraint for a 2nd degree CBT. Geometrically this means that the two vectors $\mathbf{T}_1$ and $\mathbf{T}_2$ together must span a plane that contains the difference vector $\mathbf{T}_0 - \mathbf{T}_3$. If the latter is zero, then the vectors $\mathbf{T}_1$ and $\mathbf{T}_2$ are completely free. This condition again leads to large equations hard to

interpret and whether we can use this to formulate some useful constraints is an open question.

5) CBT = binormal of the boundary curve

A natural choice for the tangent plane to the surface along the boundary curve is the one that has as its normal the *normal* vector of the curve. If $\Gamma(v)$ parametrizes a boundary curve, its *tangent* vector $\mathbf{T}(v)$ is $\dfrac{D_v\Gamma(v)}{|\ D_v\Gamma(v)\ |}$ and its *normal* $\mathbf{N}(v)$ is $\dfrac{D_v\mathbf{T}(v)}{|\ D_v\mathbf{T}(v)\ |}$.

In these terms, the formulation for the $G^1$ condition can be expressed as a vanishing scalar product:

$$< \mathbf{N}(v)\ ,\ \sum_{i=0}^{3} \mathbf{R}_i\ B_i^3(v) > = 0$$

Note that it is enough to use the numerator of $\mathbf{N}(v)$ in the scalar product.

MACSYMA showed us that the above equation is a 7th degree polynomial in $v$, which equated to 0 produces 8 equations with the 6 coordinates of $\mathbf{R}_1$ and $\mathbf{R}_2$ for unknowns. Two of the equations reflect constraints on the boundary curves meeting at the vertices. Solving the remaining 6 proved again to be an overwhelming task for MACSYMA. The solution would provide a closed-form expression for the 2 interior control points in terms of boundary information.

It is questionable whether this approach is useful in practice. The binormal could swing around wildly if the curve itself is S-shaped. A smooth interpolation as implied by Chiyokura's approach seems more appropriate.

### 3.1.4. *Our Experiments with Coefficient Functions and Vectors*

Since the use of the determinants leads to such unwieldy equations, we have also looked at the formulation of the $G^1$ continuity with the use of the derivative vectors together with some simple coefficient functions. To keep as much generality as possible, we used two linear functions and one quadratic function (in front of the along-boundary derivative vector):

$$k(v)\ 4\ \sum_{i=0}^{3}\mathbf{T}_i\ B_i^3(v) + h(v)\ 4\ \sum_{i=0}^{3}\mathbf{R}_i\ B_i^3(v) + a(v)\ 3\ \sum_{i=1}^{3}\mathbf{S}_i\ B_i^2(v) = 0$$

with $k(v)$ and $h(v)$ linear and $a(v)$ quadratic, expressed in the Bernstein basis: $k(v) = k_0\,(1-v) + k_1\,v\ ;\ \ h(v) = h_0\,(1-v) + h_1\,v\ ;\ \ a(v) = a_0\,(1-v)^2 + 2\,a_1\,v\,(1-v) + a_2\,v^2$. This gives us more coefficients and does not constrain possible solutions as much as the functions chosen by Farin or by Chiyokura.

To start with, there are seven coefficients. The planarity constraint at either end of the boundary gives additional constraints and reduces the number of variables to only 3.

Expressing $\mathbf{R}_1$ and $\mathbf{R}_2$ with the use of the planarity equations and substituting for them in the other 5 equations, results in a constraint system of manageable complexity. The corresponding equations are shown in Appendix 2. It essentially shows that if the 3 coefficients of $a(v)$ are chosen and $\mathbf{T}_1$ is set, $\mathbf{T}_2$ is then determined. Thus it looks like the formulation with the coefficient functions has left us with six degrees of freedom. The question is whether given $\mathbf{T}_1$ and $\mathbf{T}_2$, we can also solve for the 3 coefficients of $a(v)$. This would allow us to determine $\mathbf{R}_1$ and $\mathbf{R}_2$ completely. If not, what is the implied restriction on the $\mathbf{T}_1$, $\mathbf{T}_2$ vectors ?

Furthermore, we need to know whether there are other good auxiliary patch definitions, and perhaps even some that lead to a more or less automatic fulfillment of the constraints on the Bézier points that arise when more than one boundary curve are considered simultaneously, and the Bézier point derived across either one should be the same.

## 4. A SYSTEM FOR THE DESIGN OF SMOOTH INTERPOLATING SURFACES

With an understanding of the constraints implied by Bézier patches and by the $G^1$ continuity on the cubic boundaries and with the mechanisms developed above, one can construct a system that provides smooth surfaces of arbitrary topology through a set of points.

This section describes the implementation for the basis of such a system in the framework of UNIGRAFIX [Séquin *et_al* '83]. The main concepts and operational modules are described in the following.

### 4.1. Conceptual View of an Interactive Design System

The ideal setting for design would consist of an interactive editor in the style of H.B. Siegel's *Jessie* [Siegel '86], with which the user can manipulate the basic elements of a picture (vertices, faces, etc) as well as the control points that define curved edges and patches. The designer works with a mouse and keyboard to specify the cubic boundaries, which the system then uses to generate curved patches by one of the methods discussed in Section 3.

Sometimes control points are not a very intuitive representation of the geometry of curves or surfaces, and the task of specifying them for every edge to be curved is a rather tedious task. Thus the system has to provide some automation, to produce an initial smooth shape that can be then corrected interactively by manipulation of its control points.

The minimal information that must be provided to the system is a polyhedral approximation of the object to specify its topology. The system can then provide local rounding operations, e.g. for individual edges as in the MODIF modeler [Chiyokura *et_al* '84]. Another approach is to

provide global rounding of the whole object, except for specifically marked features. We have implemented the latter approach.

### 4.2. Global Smoothing Procedures for Polyhedral Objects

The starting point is a rough polyhedral approximation to the object consisting of vertices to be interpolated, edges connecting them and faces to determine its topology. From this input the system derives the first smooth approximation by producing a set of cubic curves automatically, one for each edge in the polyhedral model.

We will discuss next various options of how such initial smoothing can be achieved as well as our first simple implementation in the program called **ucl** .

**Triangulation of the Polyhedral Object:** Our first implementation works exclusively with triangular patches. Thus, the polyhedral object must be preprocessed to triangulate its faces. The UNIGRAFIX module **ugtess** is a stand-alone program that preforms tessellations of faces of UNIGRAFIX objects into convex or triangular pieces [Séquin '86]. This module is not included in **ucl** .

**Defining the Vertex Normals:** To build the cubic curves meeting smoothly at the given vertices, a tangent plane, or equivalently a vertex normal, has to be assigned to each vertex.

Our system computes each vertex normal as the weighted average between all the normals of all faces using this point. The weighting factor is the angle between the two edges of each face using that vertex.

Alternatively, the area of the face could be taken into account so that a set of eight points defining a rectilinear brick would result more or less in the ellipsoid that results from scaling the sphere circumscribing a cube when the same scaling operation is applied that transforms the cube into the brick.

**Defining the Curved Edges:** With the vertex normals and thus the tangent planes at all vertices defined, the original straight edges of the polyhedral mesh are then converted into curved cubic edges by computing their Bézier points. These points are found by projecting an original edge onto the vertex tangent plane and using a suitable fraction of the original edge length as the distance between the vertex and the Bézier point. The corresponding factor will determine the bulge of the curve and the corresponding roundness of the resulting surface.

Again, one can experiment with more or less complicated functions involving edge length, angle between edge and tangent plane, area of the adjacent faces, and others to find the combination that best produces a smooth interpolating surface of the kind a designer would draw through the given points.

**Building the Patches and Rendering:**

*Gregory Patches:* Next, **ucl** computes the interior control points of a Gregory patch per curved face (see Section 3.1.3-3). If we simply want a rounded object rendered with smooth shading, then Gregory patches are appropriate. Since the current renderer of UNIGRAFIX only deals with planar faces, curved patches have to be approximated by polyhedrons. There are no subdivision algorithms for Gregory patches that we are aware of, thus rendering can only be achieved by evaluation of selected points on the surface and constructing approximating facets between them. The user has the choice of how fine the approximation will be by specifying the number of linear segments that will represent each cubic border. A net of triangular facets is then generated based on those linear segments.

*Bézier Patches:* We described already the advantages of using polynomial Bézier patches instead of Gregory patches. To achieve this conceptually, one can subdivide the original faces in the polyhedral approximation of the object as much as necessary to achieve enough degrees of freedom to build Bézier patches on each subdomain within the face. This approach would give the designer complete control over all cubic edges. Alternatively, one can constrain somewhat the cubic edges or the tangent planes at the vertices in such a way that each original triangular face in the polyhedron can be the domain for a Bézier patch. If this is feasible, then one version of the system could automatically fulfill these constraints in creating the cubic edges and create such a minimal representation of an interpolating surface.

If Bézier patches were used, the number of consecutive subdivisions of each patch (see Section 2.1) defines how fine a polyhedral approximating net the system constructs. Adaptive subdivision can take care of surfaces with widely varying curvatures [Filip '85]. A polyhedral net constructed in this way reflects better the geometrical characteristics of a surface, since large facets reveal areas of little curvature and small facets areas of more curvature.

### 4.3. Selective Smoothing

Since many objects are not overall smooth, our system has to give the possibility of specifying edges that will remain straight, or that will show a discontinuity of tangent planes, or faces that will remain flat after the global smoothing process. In the first place, this calls for an extension of the UNIGRAFIX polyhedral description language to capture the intent of the designer. Secondly, we need a language to describe the resulting curved shapes unambiguously. The latter extension of UNIGRAFIX is called **UniCubix** (see section 4.4).

**Treatment of edges:** In the internal data structure, we need to distinguish between **edges** , which are visible as geometric features in the final object, and **borders** , which are termination lines for joining patches together. Both edges and borders can be either straight or

curved.

A straight edge or border gets Bézier points on the edge, at a distance of 1/3 and 2/3 of the length of the edge from any of its endpoints.

Since the interior control points of a patch are derived entirely from the control points on the boundary, the Bézier points on edges have to preview the shape of the patch. For instance: a curved border ( **bl**, **bc** ) that comes to meet with a sharp edge ( **el**, **ec** ) at a vertex gets its closest-to-the-vertex Bézier point to be out of the vertex tangent plane, to create a discontinuity in tangent planes at the sharp edge (see Figure 4.1). We chose for the location of a Bézier point in such a circumstance to be the middle point between its vertex and the other Bézier point on the edge. Note that this way of breaking the tangent plane continuity doesn't introduce extraneous concavities.

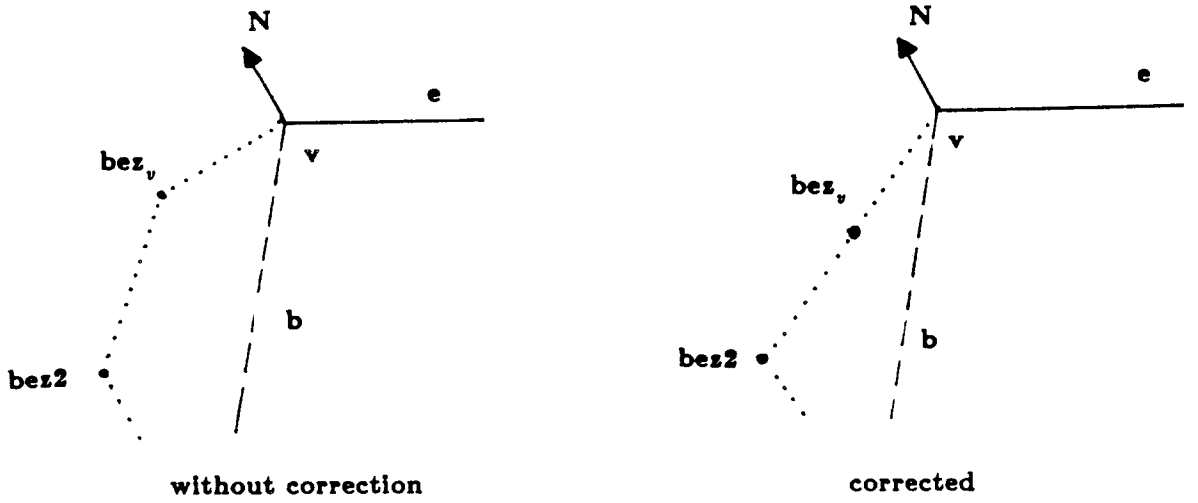



**Figure 4.1** - *Correction on edges neighboring a sharp edge*

A curve bounding a flat face gets Bézier points following the above guidelines; they are subsequently replaced by their orthogonal projection onto the plane of the face.

**Treatment of flat faces:** Faces marked to stay flat are flagged internally to avoid the computation of its interior control points, since they don't need to be subdivided.

Flatness implies that all their edges (which might be curved) must have their Bézier points lie in the plane of the face.

### 4.4. Description of Smooth Objects

Whether the curved object is created by hand with an interactive editor or by an automated process, we need a format to capture and store the result with its complete curvature information. For that purpose we extended the UNIGRAFIX descriptive language into **UniCubix** . We searched for a format that would make minimal changes in the syntax of UNIGRAFIX to keep coherence

between the two formats. Thus, UNIGRAFIX statements can be read as part of **UniCubix** and used in the same manner to build the internal data structure.

A brief summary of the basic UNIGRAFIX statements follows. For a detailed description see [Séquin *et_al* '83b].

| | | |
|---|---|---|
| vertex: | **v** | *ID x y,z;* |
| wire: | **w** | *[ID] (v1 v2 ... vn) (...) [colorID];* |
| polyhedral face: | **f** | *[ID] (v1 v2 ... vn) (...) [colorID];* |
| definition: | **def** | *defID;* |
| | | *non-def-commands* |
| | **end;** | |
| instance: | **i** | *[ID] (defID [transformations]);* |
| array: | **a** | *[ID] ( defID [transformations]) size [transformations];* |
| light: | **l** | *[ID] ( intensity [x y z [h]];* |
| color: | **c** | *colorId intensity [hue [saturation [translucency]]];* |
| include file: | **include** | *filename [transformations];* |
| comment: | **{** | *anything but unmatched* { }, {*nesting is OK*}} |

The added statements in **UniCubix** follow:

*Edge Specifications*

| | | |
|---|---|---|
| curved border: | **bc** | *[ID] (v1 v2* $b1_x\ b1_y\ b1_z\ b2_x\ b2_y\ b2_z$ *);* |
| straight border: | **bl** | *[ID] (v1 v2);* |
| curved edge: | **ec** | *[ID] (v1 v2* $b1_x\ b1_y\ b1_z\ b2_x\ b2_y\ b2_z$ *);* |
| straight edge: | **el** | *[ID] (v1 v2);* |

These four statements *add* curvature information. Even though an edge can be described with any of these statements prior to being implicitly specified in a face, a reasonable description would not include edges that don't belong to faces.

For the moment we work with Bézier curves on the edges, but the edge/border statements could in principle contain other, more complicated construction rules. This is one of the reasons why curvature information is given in explicit edge statements and is not integrated into the face statement. Another reason is to avoid restating the construction specifications more than once for each edge shared by adjacent patches. The third reason is the above mentioned intent to introduce minimal changes to UNIGRAFIX and to make **UniCubix** a direct extension.

- *Patch Specification*

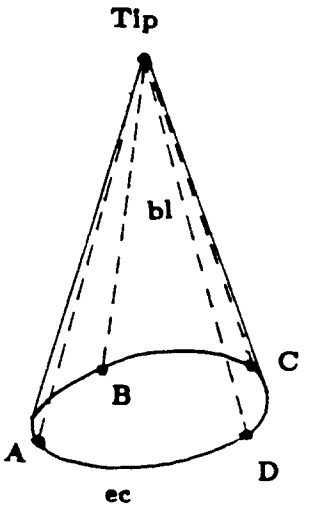
*curved patch:* **p** *[ID] (v1 v2 ... vn);*

Since a patch is fully determined by its contour and the standardize behavior of the cross-boundary derivative, there is no need for an extra statement. Still we make an explicit distinction between flat faces and potentially curved patches to save unnecessary processing: the face statement ( **f ...** ) will be treated like a polygon, no interior control points will be computed for it, and it will not be subdivided. It may have curved edges, but these (and their control points) must lie in the plane of the face; the keyword **p** , on the other hand, indicates that a curved patch is expected, for which interior control points and subdivision are necessary.

Here is an example of a cone in **UniCubix** format:

```
v    Tip    0 0 1;
v    A      0 1 -1;
v    B      1 0 -1;
v    C      0 -1 -1;
v    D      -1 0 -1;
f    Bot    (A B C D);   {flat base}
p    a      (Tip B A);
p    b      (Tip C B);
p    c      (Tip D C);
p    d      (Tip A D);

bl   (Tip A);
bl   (Tip B);
bl   (Tip C);
bl   (Tip D);
ec   (A  B   0.3 1 -1    1 0.3 -1);
ec   (B  D   1 -0.3 -1   0.3 -1 -1);
ec   (C  D   -0.3 -1 -1  -1 -0.3 -1);
ec   (D  A   -1 0.3 -1   -0.3 1 -1);
```




### 4.5. Interactive Framework for Previewing the Object during Design

We implemented **ucl** (for UniCubix Interactive) as a basis for the modeling system combining the above ideas.

So far, we have used it mostly for debugging algorithms and to gain a better understanding of how the smoothing operations work on the geometry of objects. It was built upon **ugl** (for UNIGRAFIX Interactive, [Gal '86]), which provided the I/O subroutines, the basic data structures, and the viewing and transformation options.

To gain a detailed understanding of all steps of the process, we included several facilities to view objects during various stages of their transformation from a polyhedral UNIGRAFIX object to a curved **UniCubix** shape. For illustrations see Appendix 5. It is also possible to view the control elements such as:

- vertex normals
- cubic curves with their Bézier points
- control points of patches.

Each step in the process, can be written out in UNIGRAFIX format.

The **ucl** manual page for UNIGRAFIX can be found in Appendix 4.

## 5. IMPLEMENTATION

### 5.1. Data Structure

The data structure for **ucl** is basically the same as for the UNIGRAFIX 2 system:

- a VERTEX structure contains its coordinates, and pointers to edges meeting at that vertex, and faces using the vertex. The addition for **ucl** is a VECT field containing the normalized vertex normal, computed as soon as a smoothing command is used. Vertex normals are computed for all vertices in the same way, even if straight edges or flat faces meet at the vertex. When the time to use them comes (in computing interior control points for instance) the program decides whether the stored vertex normal or the appropriate face normal should be used.

- an EDGE structure points to its endpoint VERTEX structures and to the list (FLIST) of faces using it. For **ucl** three VECT pointers were added to point to the three quartic Bézier points that result from the degree elevation of the cubic curves. These rather than the 2 control points of the cubic curves were stored to avoid recomputation of the degree elevated points at several points in the program.

- a FACE structure points to its contour list (in the present version of **ucl** this list is assumed to have one element, since only triangular faces are acceptable). The added fields are: an array of 18 pointers to VECT structures, to hold the control points of a curved patch (actually allocated only for curved patches), and an array of 3 pointers to its VERTEX corners, to avoid frequent search through CONTOUR and ELIST structures for the corners of the triangles and to keep a necessary and coherent orientation when building the array of 18 control points.

The only other changes were new flag definitions to mark the four types of edges (STREDGE, CUREDGE, STRBORDER, CURBORDER), and the two types of faces (PLANAR and CURPATCH), and the new commands ( STREDGECMD, CUREDGECMD, STRBORDERCMD, CURBORDERCMD, CURPATCHCMD, FLATFACECMD) in **ucl** .

### 5.2. Modularization

The program was functionally modularized according to the commands present in the interface. Subroutines concerning each command were grouped in a separate file, and basic subroutines used by several commands were stored in the file **basic.c** .

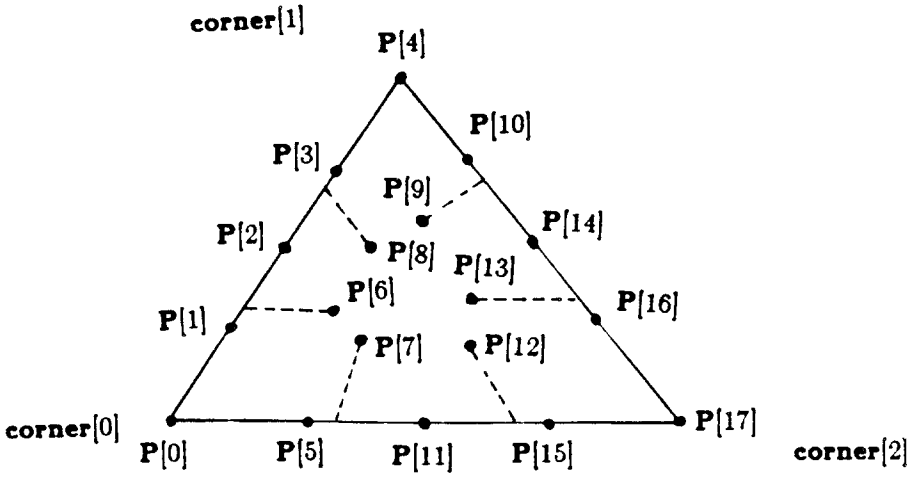



**Figure 5.1** *Labeling of a Gregory patch*

### 5.3. Hints for Future Program Maintainers

The labeling of a quartic Gregory patch is shown in Figure 5.1.

## 6. DISCUSSION

**UniCubix** has proved to be a satisfactory format to describe curved objects defined by cubic curves. The first implementation of **ucl** is a useful prototype that demonstrates that the conceptual view of such a system is appropriate. It also serves well as a framework to study variations of the smoothing algorithm.

### 6.1. Present limitations of ucl

- The constraint that faces have to be triangulated, usually breaks symmetries of the object, sometimes in a less than pleasant way, since the algorithm supported by **ugtess** occasionally produces long and skinny triangles.

- The locality of the patch specifications produces discontinuity of tangent planes when a flat face and a curved patch meet: the flat face if forced to have CBT on its plane, but the patch is assigned the standardized CBT irrespective of the behavior of its neighbor, producing unwanted creases. The handling of mixed vertices, where different types of edges and borders meet, needs to be reviewed.

- Currently the data structure supports only one object internally. Thus after an object is read and modified by any of the commands, the original object is partially lost. For instance, if the user requests to see the cubic curves bounding the patches, wires are created to approximate them and faces are deleted, impeding any further construction of patches. In this case, the user has to clear the current scene and read the original object again from its ASCII description.

A better framework would keep an initial "core" object, to which information useful in several steps of the smoothing process would be added (like vertex normals, and control points). Temporary objects for display of intermediate steps are then built in a "scratch-pad" structure (e.g. the wires representing the vertex normals, the cubic curves wire representation, the wire net of the control points for patches, or the polyhedral triangular net rendered finally for the display module). The "core" object would be represented in **UniCubix** when stored in a file, and the "scratch-pad" object would only require UNIGRAFIX statements for its description.

- There is a conflict between the current construction of objects and the present Gouraud [Gouraud '71] shader in UNIGRAFIX: the latter displays no sharp features since illuminations are averaged across all edges (even sharp ones). One way around the problem could be to duplicate sharp edges, its end vertices and the edges of the two faces meeting at the sharp edge that contain the endpoints of the sharp edge. This would create slits in the surface (marked as borders by the shader) and would change the topology of the object. A better solution would be to improve the Gouraud shader, to account for sharp features.

- No concern has been given to color. This would be useful to view different steps of the design of the object simultaneously (like vertex normals superimposed in a different color onto the net of control points).

Long term reforms should adapt **ucl** to H.B.Siegel's *Jessie* style of interaction, to avoid cumbersome manual specifications in ASCII files. New options to create contour lines on curved patches may then prove useful for quick feedback about the resulting shapes.

In spite of the listed disadvantages, **ucl** is the foundation of a needed addition to the UNIGRAFIX system that would bring it closer to become a useful CAD modeler. In our research it has proved a good tool to understand the geometrical relations among control points, patch surfaces, tangent planes of the objects we attempted to model.

## 7. ACKNOWLEDGEMENTS

This work would not have been possible without the enthusiastic, dedicated and inspiring support of Prof. Carlo H. Séquin. His penetrating ideas and enlightening suggestions made our collaboration a continuous source of excitement, and an unforgettable experience.

My husband, Filip , encouraged me throughout with more than enough care, patience and love.

I am also thankful to Prof. Larry Rowe, who generously agreed to review this report in the face of many other commitments, and to Dan Filip for innumerable discussions that helped clarify many ideas and for pointing to the right literature. Nachshon Gal provided me with his **ugi** code which made the development of **uci** much easier. Ziv Gigus initiated me into UNIGRAFIX.

Support for this work was provided in part by the Semiconductor Research Corporation.

## Appendix 1

## $G^1$ Continuity Conditions across the Common Boundary of Neighbor Quartic Triangular or Bicubic Rectangular Bézier Patches

The following equations are the coefficients of the terms of degree 0 through 8 of the 3 by 3 determinant in Section 3.1.1. Refer to Figure 3.1 for nomenclature.

$$\det ( R_0 S_1 T_0 ) = 0$$

$$2 \det ( R_0 S_2 T_0 ) + 3 \det ( R_0 S_1 T_1 ) + 3 \det ( R_1 S_1 T_0 ) = 0$$

$$\det ( R_0 S_3 T_0 ) + 3 \det ( R_0 S_1 T_2 ) + 3 \det ( R_2 S_1 T_0 ) +$$
$$6 \det ( R_0 S_2 T_1 ) + 6 \det ( R_1 S_2 T_0 ) + 9 \det ( R_1 S_1 T_1 ) = 0$$

$$\det ( R_0 S_1 T_3 ) + \det ( R_3 S_1 T_0 ) + 3 \det ( R_0 S_3 T_1 ) + 3 \det ( R_1 S_3 T_0 ) +$$
$$6 \det ( R_0 S_2 T_2 ) + 6 \det ( R_2 S_2 T_0 ) + 9 \det ( R_1 S_1 T_2 ) + 9 \det ( R_2 S_1 T_1 ) +$$
$$18 \det ( R_1 S_2 T_1 ) = 0$$

$$2 \det ( R_0 S_2 T_3 ) + 2 \det ( R_3 S_2 T_0 ) + 3 \det ( R_1 S_1 T_3 ) + 3 \det ( R_3 S_1 T_1 ) +$$
$$3 \det ( R_0 S_3 T_2 ) + 3 \det ( R_2 S_3 T_0 ) + 9 \det ( R_1 S_3 T_1 ) + 9 \det ( R_2 S_1 T_2 ) +$$
$$18 \det ( R_1 S_2 T_2 ) + 18 \det ( R_2 S_2 T_1 ) = 0$$

$$\det ( R_0 S_3 T_3 ) + \det ( R_3 S_3 T_0 ) + 3 \det ( R_2 S_1 T_3 ) + 3 \det ( R_3 S_1 T_2 ) +$$
$$6 \det ( R_1 S_2 T_3 ) + 6 \det ( R_3 S_2 T_1 ) + 9 \det ( R_2 S_3 T_1 ) + 9 \det ( R_1 S_3 T_2 ) +$$
$$18 \det ( R_2 S_2 T_2 ) = 0$$

$$\det ( R_3 S_1 T_3 ) + 3 \det ( R_1 S_3 T_3 ) + 3 \det ( R_3 S_3 T_1 ) +$$
$$6 \det ( R_2 S_2 T_3 ) + 6 \det ( R_3 S_2 T_2 ) + 9 \det ( R_2 S_3 T_2 ) = 0$$

$$2 \det ( R_3 S_2 T_3 ) + 3 \det ( R_2 S_3 T_3 ) + 3 \det ( R_3 S_3 T_2 ) = 0$$

$$\det ( R_3 S_3 T_3 ) = 0$$

## Appendix 2

## $G^1$ Continuity Conditions across the Common Boundary of Neighbor Quartic Triangular or Bicubic Rectangular Bézier Patches when the Cross-Boundary Derivative is Linear

The following equations are the coefficients of the terms of degree 0 through 6 of the 3 by 3 determinant in Section 3.1.1, when the CBT is linearly interpolated between vertices (see Section 3.1.3-3). Refer to Figure 3.1 for nomenclature.

$$\det ( R_0 S_1 T_0 ) = 0$$

$$2 \det ( R_0 S_2 T_0 ) + \det ( R_0 S_1 T_3 ) + 3 \det ( R_1 S_1 T_0 ) = 0$$

$$\det ( R_0 S_3 T_0 ) + 3 \det ( R_1 S_1 T_3 ) + 3 \det ( R_2 S_1 T_0 ) +$$

$$2 \det ( R_0 S_2 T_3 ) + 6 \det ( R_1 S_2 T_0 ) = 0$$

$$\det ( R_0 S_3 T_3 ) + \det ( R_3 S_1 T_0 ) + 3 \det ( R_1 S_3 T_0 ) + 3 \det ( R_2 S_1 T_3 ) +$$

$$6 \det ( R_1 S_2 T_3 ) + 6 \det ( R_2 S_2 T_0 ) = 0$$

$$\det ( R_3 S_1 T_3 ) + 3 \det ( R_1 S_3 T_3 ) + 3 \det ( R_2 S_3 T_0 ) +$$

$$2 \det ( R_3 S_2 T_0 ) + 6 \det ( R_2 S_2 T_3 ) = 0$$

$$2 \det ( R_3 S_2 T_3 ) + 3 \det ( R_2 S_3 T_3 ) + \det ( R_3 S_3 T_0 ) = 0$$

$$\det ( R_3 S_3 T_3 ) = 0$$

## Appendix 3

### 1. UniCubix Language Summary

UniCubix consists of UNIGRAFIX statements plus curvature information.

UNIGRAFIX Statements:

| | | |
|---|---|---|
| vertex: | **v** | *ID x y z;* |
| wire: | **w** | *[ID] (v1 v2 ... vn) (...) [colorID];* |
| polyhedral face: | **f** | *[ID] (v1 v2 ... vn) (...) [colorID];* |
| definition: | **def** | *defID;*<br>*non-def-commands*<br>**end;** |
| instance: | **i** | *[ID] (defID [transformations]);* |
| array: | **a** | *[ID] ( defID [transformations]) size [transformations];* |
| light: | **l** | *[ID] ( intensity [x y z [h]];* |
| color: | **c** | *colorId intensity [hue [saturation [translucency]]];* |
| include file: | **include** | *filename [transformations];* |
| comment: | { | *anything but unmatched* { }, {*nesting is OK*} |

Added UniCubix Statements:

| | | |
|---|---|---|
| curved border: | **bc** | *[ID] (v1 v2* $b1_x\ b1_y\ b1_z\ b2_x\ b2_y\ b2_z$ ); |
| straight border: | **bl** | *[ID] (v1 v2);* |
| curved edge: | **ec** | *[ID] (v1 v2* $b1_x\ b1_y\ b1_z\ b2_x\ b2_y\ b2_z$ ); |
| straight edge: | **el** | *[ID] (v1 v2);* |
| curved patch: | **p** | *[ID] (v1 v2 ... vn);* |

**Patches** meet at **borders** with continuity of tangent planes, thus borders are invisible seams. They can be straight or curved.

**Edges** are visible and sharp. There is discontinuities of tangent planes for patches meeting at an edge. They can be straight or curved.

Curved borders and curved edges carry the specification of their two Bézier points in the same order as the order of the end vertices, i.e. the above **bc** or **ec** statements specify a cubic curve with Bézier points { $v1\ (b1_x\ b1_y\ b1_z)\ (b2_x\ b2_y\ b2_z)\ v2$ }. Bézier points of straight edges and borders are computed automatically by the system when needed.

## 2. "Smooth" Command Input Format

The command **smooth** in **ucl** is designed to automatically replace edges by cubic curves and polyhedral faces by Gregory patches in the manner described in Section 4.2. Thus a UNIGRAFIX file read in by **smooth** computes and stores control points of edges and faces and produces an object whose description in ASCII format is in UniCubix format: all face statements **f** are transformed into patch statements **p** , all edges are specified as **bc** with their control points.

However, sometimes a designer wants to curve some of the faces, but wants others to remain flat. Also, specifying control points for curved edges is not easy. To aid the user in this task, the **smooth** command takes the following auxiliary statements:

curved edge: **ec** *[ID] (v1 v2);*

curved border: **bc** *[ID] (v1 v2);*

straight edge: **el** *[ID] (v1 v2);*

straight border: **bl** *[ID] (v1 v2);*

flat face: **F** *[ID] (v1 v2 ... vn);*

completes the control points automatically, and flags flat faces internally to avoid subdivision.

By default, edges are replaced by curved borders **bc** , unless explicitly specified to be of a different type. If information about Bézier points is explicitly given, it is used as is, rather than recomputing new points.

All faces **f** are converted into patches **p** except for specified flat faces **F** .

**Note:**

**f** represents a flat face in UniCubix (and in UNIGRAFIX ),

**f** results in a curved patch through the **smooth** operation,

**F** results in a flat face through the **smooth** operation.

Appendix 4

**UNIGRAFIX Manual Page for uci**

## NAME

uci - interactive UniCubix environment

## SYNOPSIS

**uci** [ arguments, options ]

## DESCRIPTION

*Uci* is an editing, viewing, and designing tool for UniCubix scenes. It provides most of the ugi capabilities for scenes with curved edges and curved surface patches.
The term *current scene* in this manual refers to all UniCubix objects that were either read in or created by one of the modifiers since the beginning of the session or since the last **clear** command.
The following options can be used:

**-fi** *filename*
Use file *filename* to find uci commands. Each command with its arguments must appear on a single line. This enables setting initial parameters, or even running a whole session as a batch job. Lines beginning with a '#' are ignored.

**-e** Echo the commands from the command-file. Comment lines are echoed too.

After processing the commands from the input file, uci will prompt you with:

*uc>*

You may now enter commands. Command names are single words, which may be abbreviated to any unique prefix The commands can be divided into four logical groups: I/O, Display, Modify, and Misc. The following sections describe each command.
Uci records the session in a log file called *uci.log* in the current directory.

## I/O commands

• **read** *filename1* [xform-options1] *filename2* [xform-options2] ...

**Read** is used to read in scenes from files, optionally transforming them, and adding them to the *current scene*. Each set of transformations applies only to the filename preceding it; i.e., transformations do not accumulate. See description of *xform* for details of the options. Since everything becomes part of the same scene, name conflicts may occur between two objects of the same type and same name from different files. Uci handles those conflicts correctly in most cases. When writing such a scene to a file use the **newlabels** command first.
If the *filename* is missing, standard input is read. Type your UniCubix statements in, and end input by typing **q**.
Similarly, the special filename **<<** tells uci to read the next lines as UniCubix statements, until the **q** statement. This is useful in writing uci command files (following the spirit of csh scripts).

EXAMPLE: **read** cube -rx 20 -tx 3 cube -rx -20 -tx -3

• **smooth** *filename1* [options1] *filename2* [options2] ...

Reads in a scene from a file in UNIGRAFIX, UniCubix or the special format described below and replaces edges by cubic curves and faces by curved patches unless specified otherwise.

The special commands understood by **smooth** that are not legal in UNIGRAFIX or UniCubix are:

| | | |
|---|---|---|
| bc | [ID] (v1 v2); | {curved border; default} |
| ec | [ID] (v1 v2); | {curved edge} |
| F | [ID] (v1 v2 ... vn); | {flat face} |

**Smooth** computes unspecified control points of curved borders or edges. All faces **f** are converted into patches **p** , except for specified flat faces **F** . Edges are replaced by curved borders **bc** , unless specified otherwise.

Note:

**f** represents a flat face in UniCubix or UNIGRAFIX,

**f** results in a curved patch after the **smooth** command,

**F** results in a flat face after the **smooth** command.

**-b** *factor*
Change the bulge of edges (and thus patches) by *factor*. Default is 1.0, *factor* = 0 produces flat patches, *factor* outside of [0,1] may produce loops and self intersections.

**remaining options**
as in **read** .

EXAMPLE: **smooth** cube -b 0.5 -ry 5 -tx 20 -sy 2

• **write** [options] *filename*

Writes the current scene to the named file in UNIGRAFIX or UniCubix format, depending on whether there is any curvature information available. If *filename* is not specified, standard output is used.

**-h** Writes the scene in a hierarchical format. This is the default option. This is not allowed if a modifier like **curvedg** or **patch -p** was called.

**-f** Writes a *flattened* scene (no hierarchy).

**-C** Writes a *compact* scene: vertices are written only if actually used by a face or a wire.

**-l** Writes illumination sources to *filename*.il .

**-v** Writes viewing parameters to *filename*.vp .

**-d** Prints the names of loaded definitions.

**-ae** Attach plane equations to top level faces.

**-al** Attach illumination values to top level faces.

**-s** Writes all previous commands to a uci command file. This file can then be used with the **-fl** option to uci to reproduce the current session. (In effect, the logfile is copied to the specified filename, with a time stamp).

Only one output format (either scene description or set of commands) can be written to *filename*, so only the last one specified will have effect. (To write out the scene both as a scene file and as a uci command file, use the **write** command twice).

EXAMPLE: **write** -i -h sceneOut

• **newlabels** [options]

Gives new sequential labels to scene objects. Used mainly to convert very long names to short ones, and to avoid naming conflicts before writing a scene to a file.

**-h** Keep hierarchical structure. Objects in definitions retain their original labels, while top-level objects get new labels. This is the default option. As in **write**, **-h** is not allowed if a modifier like **curvedg** or **pa -p** was called.

**-f** Scene is flattened, and hierarchy is lost. All objects get new labels.

• **clear** [options]

Clears the current scene, and allows you to start a new scene.

**-s** Clear only scene description.

**-t** Like **-s** but only *top level* objects are cleared. All loaded definitions remain, and can be referenced and instantiated by future **read** commands.

**-l** Clear only illumination sources.

**-v** Clear only viewing parameters..

If no options were specified, the default is to clear everything. Your verification is then requested.

## Display commands

• **view** [options]

Sets the *viewing parameters* for the current scene. Once set, the parameters remain in effect for the rest of the session, unless reset by another **view** command. They can be temporarily overridden by **display** options (see below) for a certain scene display. If you change from a perspective view (**-ep**) to an orthogonal view (**-ed**) or vice-versa, all parameters that are not relevant to the current view are saved but ignored.
All **display** options can be specified. An additional option is:

**-p** Prints the current viewing parameters. (Note that the set of viewing parameters is updated only *after* the **view** command, so **-p** will not reflect any new parameters set in the current **view**).

EXAMPLE: **view** -v -sa -ab -ed 3 2 -7 -vr 22

• **display** [options]

Displays the current scene on the requested device. All *ugdisp* options can be used, except for **-fi** for input file, since input is the current scene. Viewing parameters are added to the parameters that were set by **view** (see above), and override them in case of conflict. They take effect only for this one display. See *Ugdisp (UG)* man page for option details.

EXAMPLE: **display** -sg -st

• **Illuminate** [options]

Modifies the set of illum sources, and the illumination of the current scene. Two shading models can be used: uniform shading for each face, or smooth shading of the whole scene (using *Gouraud* shading). In addition, *fog* options can be specified to *fade* to a white or black background.

Options to modify the set of illum sources:

**-a** *id intensity* [x y z]
Add an illumination source. The new source may be *directional* or *ambient* in which case the direction vector is not specified. *intensity* should be in the range (0,1].

**-r** *id* Remove the specified source.

**-p** Print the list of current illum sources.

Options to modify illumination of the scene:

**-l** Illuminate the scene with uniform face shading.

**-g** Illuminate the scene with Gouraud shading.

**-fw x y z** *radius1 radius2*
Fade against white background in the interval *radius1-radius2* from the *fog point* **x y z**.

**-fb x y z** *radius1 radius2*

Fade against black background in the interval *radius1-radius2* from the *fog point* **x y z**.

The two shading models can coexist in the data structure since *uniform* data is kept in faces, and *smooth* data is kept in vertices. Thus, once any shading model was calculated (either implicitly by a **display** command, or explicitly by an **illuminate** command) it remains and can be used. To update any model after an illum source was added/removed, **-i** or **-g** should be called.

EXAMPLE: **illum** -a moon 0.4 -3 10 -15 -p -i

## Modify commands

- **xform** [options]

Transforms the whole scene. (with optional transformation of illum sources). Hierarchical structure is retained, and if the scene is written with the **-h** option (see **write**) these transformations are appended to the end of the xform-lists of top-level instances and arrays.

**-tx, -ty, -tz** *amount*
Translate scene by *amount* in the specified direction.

**-rx, -ry, -rz** *angle*
Rotate scene around specified axis by *angle*.
(positive degrees cause counter-clockwise rotation when viewed in direction of positive axis).

**-sx, -sy, -sz** *factor*
Scale the scene by *factor* in the appropriate dimension.

**-sa** *factor*
Scale the scene by *factor* in all three dimensions.

**-mx, -my, -mz**
Mirror specified coordinates about the origin.

**-ma** Mirror all coordinates about the origin.

**-M3** *9x9 matrix*
Use *one to nine numbers* as transformation matrix.

**-M4** *4x4 matrix*
Use *one to sixteen numbers* as transformation matrix.

**-xl** Transform coordinates of light sources as well.

**-px** Print the list of specified transformations.

**-pm** Print the total transformation matrix.

EXAMPLE: **xform** -sa .7 -ry 23 -tx 10 -M3 1 0 2 -1 9 1 -xl -pm

- **smooth** [options]

See **I/O** commands.

- **bulge** *factor*

Change the bulge of cubic curves by the given factor.

EXAMPLE: **bulge** 1.2

- **patch** [options]

Computes and stores control points of Gregory patches for non-planar faces.

**-p** Creates a wire representation of Gregory control points linked to their edges.

EXAMPLE: **patch** -s 4 -p

- **normals** [options]

To each vertex it adds a wire representation of a vertex normal that is the average of normals of all faces meeting at the vertex, each weighted by the angle of the face at the vertex.

**-l** *number*
Length of wires representing the normals. Default: l=1.0.

**-c** Clear previously displayed normals.

EXAMPLE: **normals** -l .7

- **curvedg** [options]

Approximates the cubic curves in the current scene with piecewise linear wires trains. Faces are deleted.

**-s** *number*
Number of linear segments to represent each edge. Default: s=10

**-p** Add representation of Bezier points of edges linked to their vertices.

EXAMPLE: **curvedg** -s 4 -p

- **net** [options]

Create a planar face representation of non-planar patches in UNIGRAFIX format.

**-s** *number*
Number of segments each edge is subdivided into for rendering. Default: s=5.

EXAMPLE: **net -s 4**

## Misc commands

- **help** *command-name*

*Command-name* is a unique substring of one of the commands. Description of the command and its options is printed. Without any argument it prints the menu of commands.

EXAMPLE: **help** help

- **quit**

Ends the session.

- ! *string*

C-shell escape. *String* contains a csh command.

## FILES

~ug/bin/uci
~ug/bin/ugdisp

**SEE ALSO**

ugi (UG), ugdisp (UG)

**DIAGNOSTICS**

Checks input files for syntax errors. Allows duplicate names.

**BUGS**

Yet to be reported.

**AUTHORS**

Lucia Longhi, Nachshon Gal

**Appendix 5**

**Ucl Images**

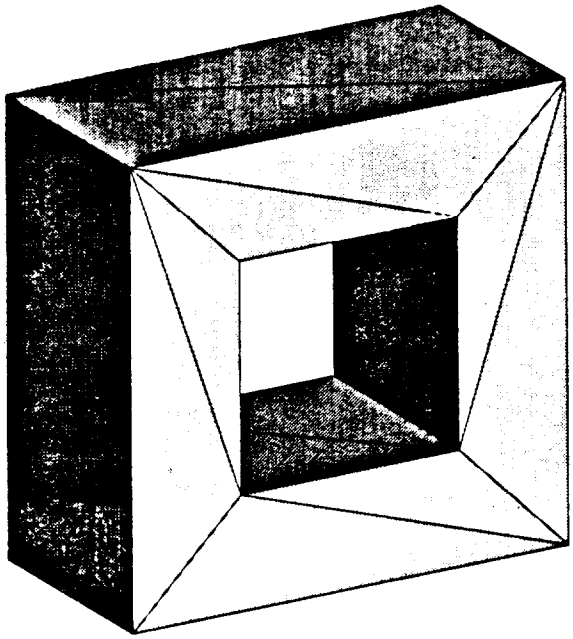


**Figure 1** *Input: Triangulated Polyhedron*

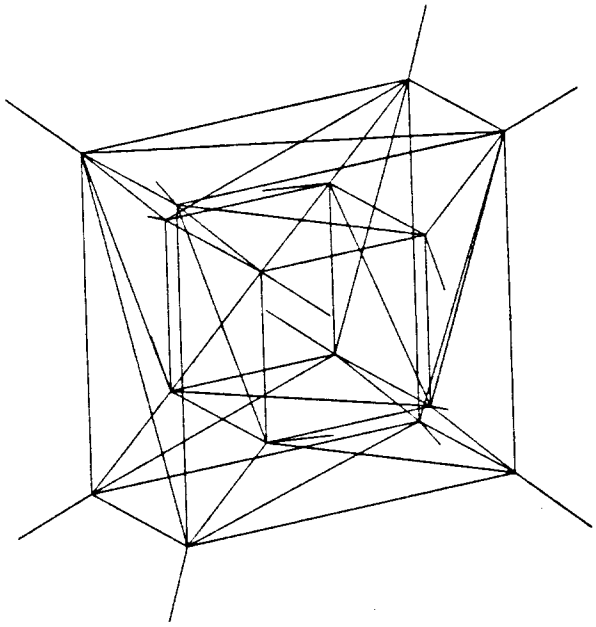


**Figure 2** *Vertex Normals*

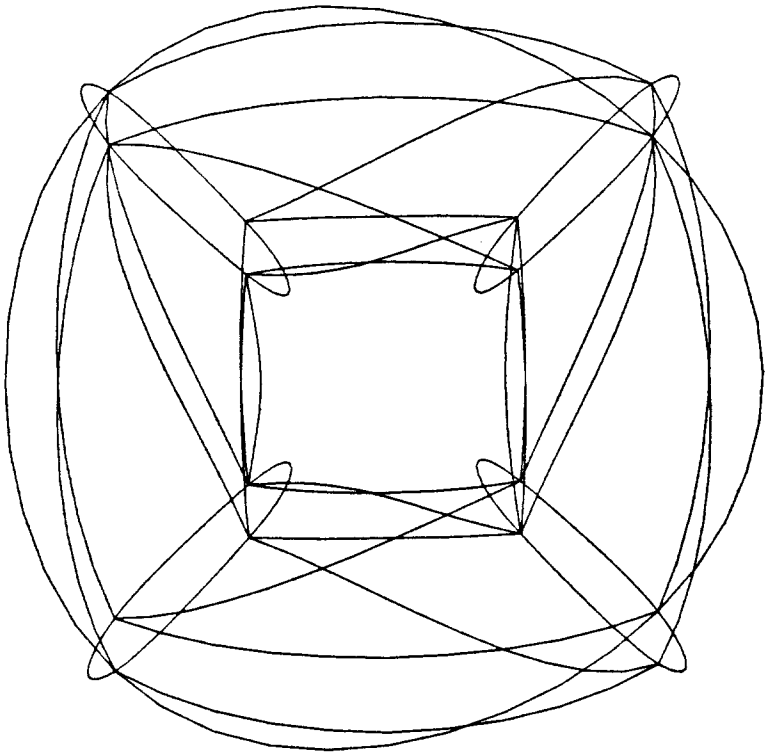


**Figure 3** *Cubic Edges*

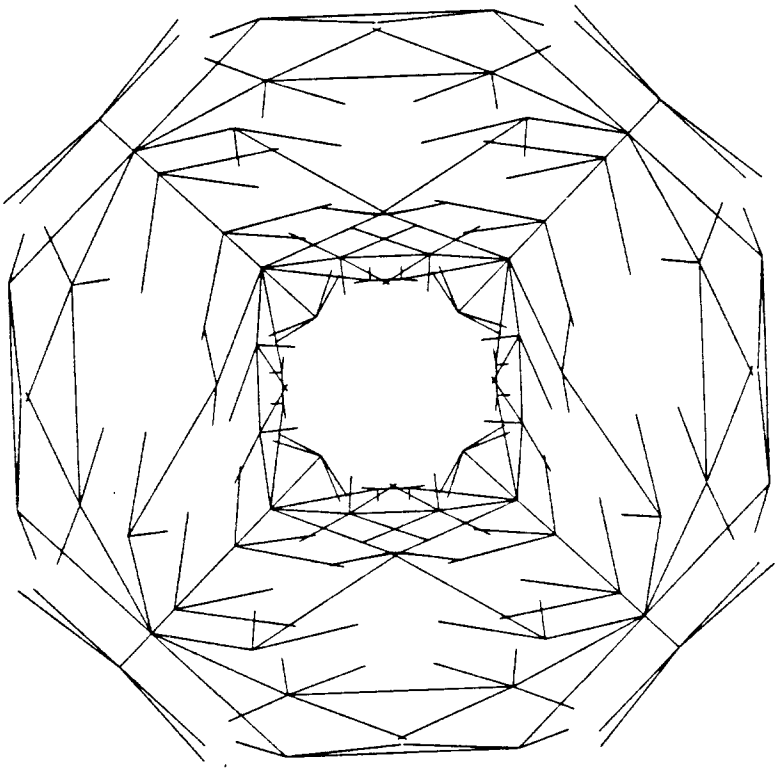


**Figure 4** *Wire Net Linking Gregory Control Points to Edges*

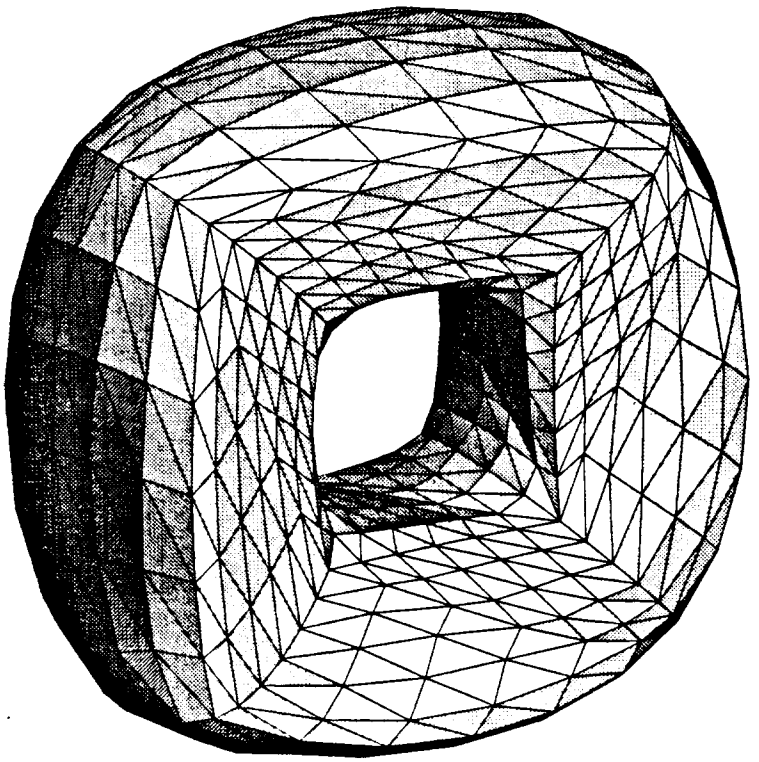


**Figure 5** *Approximated Polyhedral Net: 6-segments per Edge*

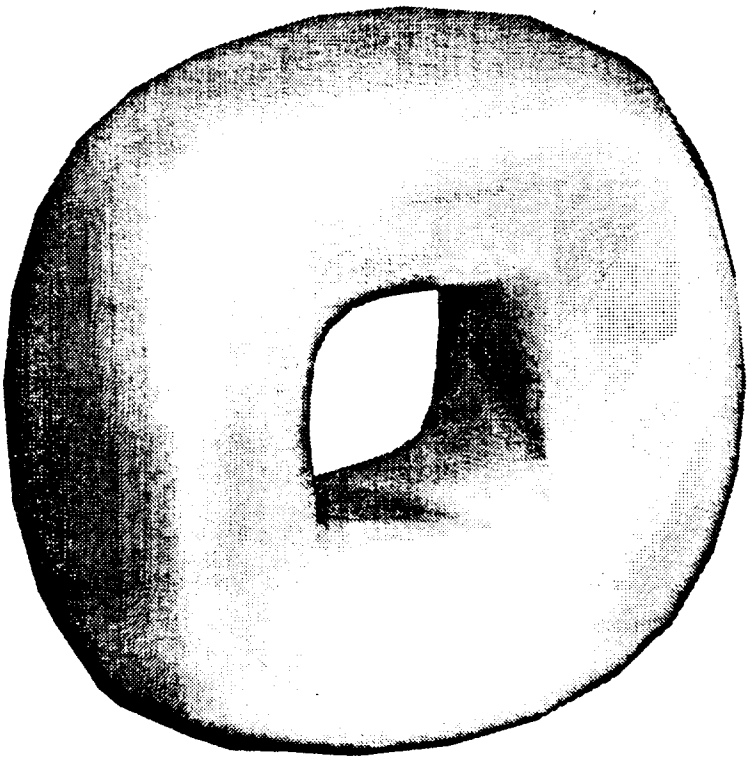


**Figure 6** *Gouraud Shaded Display: Default Bulge = 1.0*

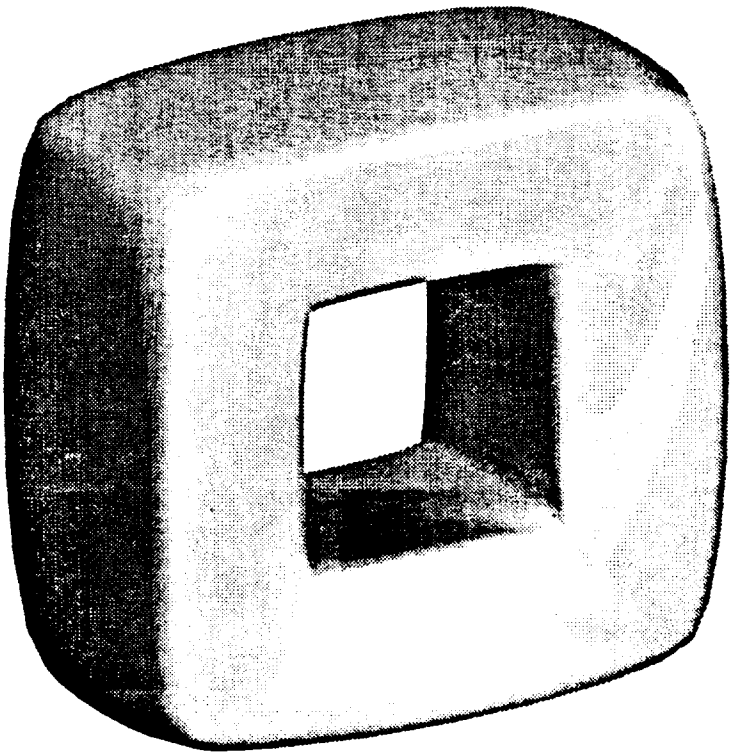


**Figure 7** *Bulge = 0.2*

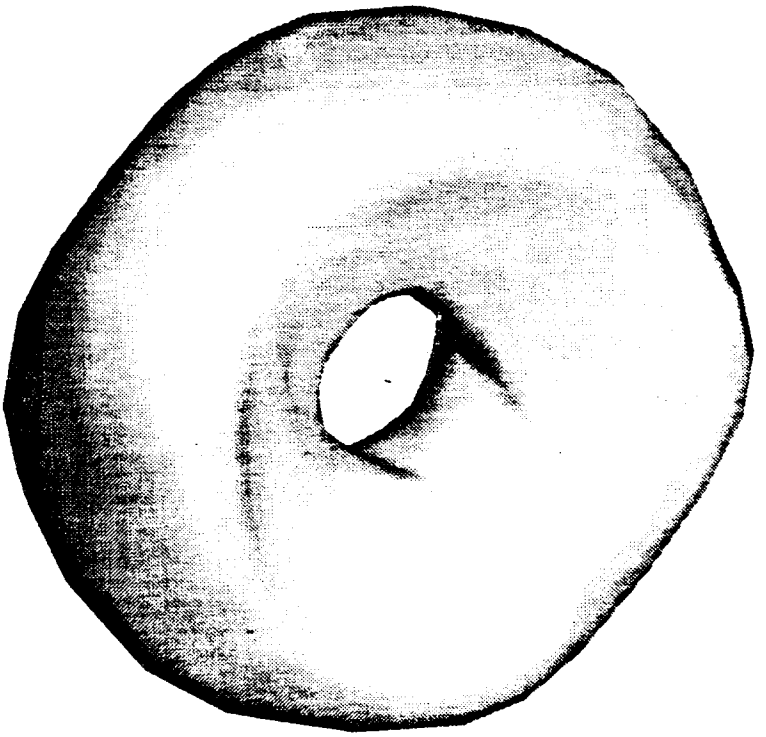


**Figure 8** *Bulge = 1.5*

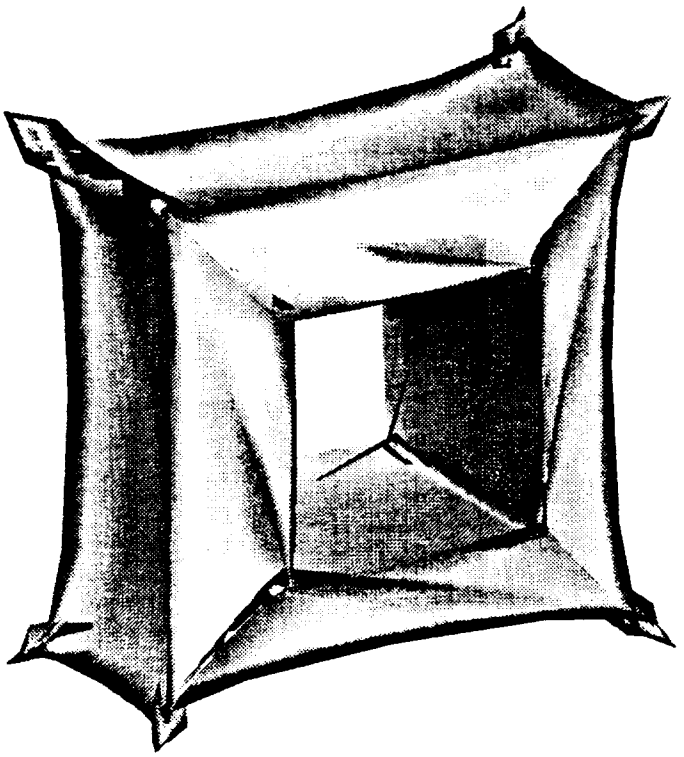


**Figure 9** *Bulge = -0.5 : Loops and Self-intersections*

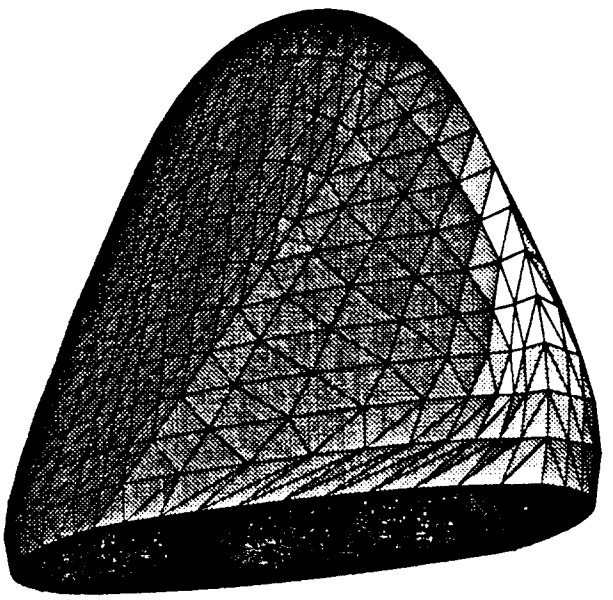


**Figure 10** *Object with flat faces, sharp edges and straight borders*

**References**


[Barnhill *et_al* '74] Robert C. Barnhill and Richard R. Riesenfeld eds., *Computer Aided Geometric Design* , Academic Press, New York (1974).

[Barnhill *et_al* '83] Robert C. Barnhill and Wolfgang Böhm eds., *Surfaces in CAGD* , North Holland, Amsterdam (1983).

[Bartels *et_al* '83] Richard H. Bartels, John C. Beatty, and Brian A. Barsky, *An Introduction to the Use of Splines in Computer Graphics,* Tech. Report No. UCB/CSD 83/136, Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, California, USA, August, 1983. Also Tech. Report No. CS-83-9, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[Bogen *et_al* '77] Richard Bogen, Jeffery Golden, Michael Genesereth, and Alexander Doohovskoy, *MACSYMA Reference Manual (version 9)* , Massachusetts Institute of Technology, 1977.

[Böehm '82] Wolfgang Böehm, "On Cubics: A Survey" *Computer Graphics and Image Processing,* Vol. 19, 1982, pp. 201-226.

[Böehm *et_al* '84] Wolfgang Böehm, Gerald Farin, and Jürgen Kahmann, "A Survey of Curve and Surface Methods in CAGD," *Computer Aided Geometric Design* , Vol. 1, 1984, pp. 61-74.

[de Casteljau '59] P. de Casteljau, *Courbes et Surfaces à Pôles,* A. A. André Citroen, Paris.

[Cavendish '74] J.C. Cavendish, "Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method," Int. J. Num. Mech. Engng., 1974, Vol 8., pp 679-696.

[Chiyokura *et_al* '83] Hiroaki Chiyokura and Fumihiko Kimura, "Design of Solids with Free-form Surfaces," ACM-SIGGRAPH *Computer Graphics* , Vol. 17, No. 3, pp. 289-298.

[Chiyokura *et_al* '84] H. Chiyokura and F. Kimura, "A New Surface Interpolation Method for Irregular Curve Models", *Computer Graphics Forum* , Vol. 3, pp. 209-218.

[Coons '64] Steven A. Coons, "Surfaces for Computer Aided Design," Design Division, Mechanical Engineering Department, M.I.T., Cambridge, Massachusetts.

[DeRose '85] Tony D. DeRose, *Geometric Continuity: A Parametrization Independent Measure of Continuity for Computer Aided Geometric Design,* Ph.D. Dissertation, University of California, Berkeley, August 1985.

[Fateman '82] Richard J. Fateman, *Addendum to the MACSYMA Reference Manual for the VAX,* Computer Science Division, University of California, Berkeley (1982).

[Farin '79] Gerald Farin, *Subsplines über Dreiecken* , Ph.D. Dissertation, Technische Universität Braunschweig, Federal Republic of Germany, December 1979.

[Farin '82a] Gerald Farin, "A Construction for Visual $C^1$ Continuity of Polynomial Surface Patches," *Computer Graphics and Image Processing,* Vol. 20, 1982, pp. 272-282.

[Farin '82b] Gerald Farin, "Smooth Interpolation to Scattered 3D Data," in [Barnhill *et_al* '83],

pp. 43-63.

[Farin '82c] Gerald Farin, "Designing $C^1$ Surfaces Consisting of Triangular Cubic Patches," *Computer-Aided Design,* Vol. 14, No. 5, September, 1982, pp. 253-256.

[Faux *et_al* '79] Ivor D. Faux and Michael J. Pratt, *Computational Geometry for Design and Manufacture,* Ellis Horwood Ltd. (1979).

[Filip '85] Daniel J. Filip, *Practical Considerations for Triangular Patch Surfaces* , M.S. Thesis, University of California, Berkeley, California, December 1985.

[Filip '86] Daniel J. Filip, "Adaptive Subdivision for Bézier Triangles," to appear in *Computer Aided Design* , 1986.

[Gal '86] Nachshon Gal, *UGI: An Interactive Environment for UNIGRAFIX,* to appear as UCB/CS Technical Report, Electrical Engineering and Computer Science Dept, University of California, Berkeley, California, January 1986.

[Garey *et_al* '78] M.R. Garey, D.S. Johnson, F.P. Preparata and R.E. Tarjan, "Triangulating a Simple Polygon," Information Processing Letters, Vol. 7, No. 4, June 1978.

[Goldman '83] Ronald N. Goldman, "Subdivision Algorithms for Bézier Triangles," *Computer-Aided Design,* Vol. 15, No. 3, May 1983, pp. 159-166.

[Gouraud '71] H. Gouraud, "Continuous Shading of Curved Surfaces," IEEE Transactions on Computers , C-24(8), August 1976, pp. 891-898.

[Gregory '74] John A.Gregory, "Smooth Interpolation Without Twist Constraints," in [Barnhill *et_all* '74], pp. 71-87.

[Hobby '85] John D. Hobby, "Smooth, Easy to Compute Interpolating Splines," Tech. Report STAN/CS/85/1047, Computer Science Dept., Stanford University, Palo Alto, California, January 1985.

[Kahmann '83] Jüergen Kahmann, "Continuity of Curvature Between Adjacent Bézier Patches," pp. 65-75 in [Barnhill *et_al* '83], pp. 66-75.

[Lewis *et_al* '78] B.A. Lewis, J.S. Robinson, "Triangulation of Planar Regions with Applications," Computer Journal, 1979, Vol. 21, No. 4, pp 324-332.

[Sabin '76] Malcolm A. Sabin, *The Use of Piecewise Forms for the Numerical Representation of Shape,* Ph.D. Thesis, Hungarian Academy of Sciences, Budapest (1976).

[Séquin *et_al* '83a] Carlo H. Séquin and P.S. Strauss, "UNIGRAFIX," *IEEE 1983 Proceedings of the 20th Design Automation Conference,"* pp 374-381.

[Séquin *et_al* '83] Carlo H. Séquin, Mark Segal and Paul Wensley, "UNIGRAFIX 2.0 User's Manual and Tutorial," Tech. Report No. UCB/CSD 83/161, Computer Science Division, Electrical Engineering and Computer Sciences Department,University of California, Berkeley, California, December 1984.

[Séquin '86] Carlo H. Séquin ed., *More Creative Geometric Modeling* , to appear as UCB/CS Technical Report, Computer Science Division, Electrical Engineering and Computer Science, University of California, Berkeley, California, 1986.

[Siegel '86] H.B. Siegel, *Jessie: An Interactive Editor for Unigrafix* , to appear as UCB/CS Technical Report, Computer Science Division, Electrical Engineering and Computer Science, University of California, Berkeley, California, January 1986.

[Várady '84] T. Várady and M. J. Pratt, "Design Techniques for the Definition of Solid Objects with Free-form Geometry," *Computer Aided Geometric Design* , Vol. 1, pp. 207-225.