C^1 Quadratic Interpolation of Meshes

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: March 2, 1999
Last Modified: November 19, 2008

Contents

1 Introduction

This document describes the algorithm developed by Cendes and Wong in [1] for interpolating arbitrary point sets whose elements are $(x_i, y_i, f(x_i, y_i), f_x(x_i, y_i), f_y(x_i, y_i))$ where $0 \leq i < N$. The spatial points (x_i, y_i) are arbitrarily spaced. At each spatial point the user specifies the function f and its two partial derivatives f_x and f_y. The outline of the algorithm is as follows:

1. *Triangulation.* The spatial points must be triangulated. I use a Delaunay triangulation. The code is a modification of code written by Watson and uses an algorithm described in [2].
2. *Subdivison.* Each triangle is subdivided into six triangles. The subdivision requires knowledge of the inscribed centers of the triangle and its three adjacent triangles.
3. *Bezier net construction.* Each subtriangle is further partitioned into four triangles. This subdivision is affine and the partition is used to build a quadratic function via the Bezier triangle method described in Chapter 18 of Farin's book [3]. The quadratics are of course C^1 functions, but additionally the interpolation is C^1 at any interface with other triangles, whether they are part of the current subdivision or part of the subdivision of an adjacent triangle.

The implementation takes an arbitrary point set, applies the steps listed above, and has a method which takes as input a new spatial location (x, y) and produces as output the interpolated function $f(x, y)$ and its two partial derivatives $f_x(x, y)$ and $f_y(x, y)$.

This algorithm is attractive in that it involves quadratic functions rather than cubic functions which are used in many standard schemes, so computation time is reduced. (To be fair in comparing, one should also look at the costs for initial setup of the data structures.) If continuous second-derivative information is required, such as in computing surface curvatures, then the quadratic algorithm can not be used.

The interpolation has local control. If the function or derivative values are modified at a single data point, then the affine subdivision of the triangles sharing the data point does not change, but the function values at the additional control points must be recalculated. If the spatial component of a single data point is modified, then the affine subdivisions of the triangles sharing the data point change. These change are propagated to any immediately adjacent triangles of those which share the data point, but no further.

Finally, note that the Cendes-Wong algorithm essentially is a mesh smoothing algorithm where the mesh is a triangulation of the graph of the function $f(x, y)$ from which the data points are sampled. The algorithm does not immediately apply to the case of a general triangulated mesh (unless that mesh is the graph of a continuous piecewise linear function in some rotated coordinate system). I provide a generalization of the Cendes-Wong algorithm to general meshes, but where the vertices of the mesh satisfy a *regularity* condition. This condition is required to preclude vertices where the topology of the mesh has particularly bad features. My algorithm simply terminates if such a vertex occurs, but it should not be difficult to modify the code so that the mesh is subdivided locally at the offending vertex. Any new triangles added to the mesh must be preprocessed (subdivision, Bezier net construction) and any adjacent old triangles affected by these changes must be reprocessed.

Consequences of my algorithm need to be investigated. For example, it would be desirable for the interpolated surface to be convex whenever the underlying mesh is convex. Experiments seem to indicate this is the case, but I have not proved it. As another example, if the angle between two adjacent triangles is extremely small, it is possible for the interpolated subsurfaces from the two triangles to intersect. This is also undesirable behavior, but probably the algorithm is not easily modified to prevent this. A related problem is when two

mesh triangles, while separated by a large distance traveled along the mesh, may be extremely close in space (the mesh has doubled back on itself).

2 Mathematical Preliminaries

2.1 Barycentric Coordinates

Let a triangle have vertices $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$. Any point $\mathbf{p}$ can be written as a *barycentric combination* of the triangle vertices, $\mathbf{p} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$, where $u + v + w = 1$. The coefficients (u, v, w) are the *barycentric coordinates* of $\mathbf{p}$ with respect to the triangle. This definition is independent of dimension; the vectors can be in $\mathbb{R}^2$ or $\mathbb{R}^3$ (or in any higher dimensional space for that matter).

Constructing u, v, and w is a matter of solving a system of linear equations. Subtracting $\mathbf{c}$ from the equation for $\mathbf{p}$, using $w = 1 - u - v$, and grouping terms leads to $\mathbf{p} - \mathbf{c} = u(\mathbf{a} - \mathbf{c}) + v(\mathbf{b} - \mathbf{c})$. The two vectors $\mathbf{a} - \mathbf{c}$ and $\mathbf{b} - \mathbf{c}$ are linearly independent since they are two edges of the same triangle. The linear independence also guarantees that there is a unique solution for u and v. Dot these vectors with the equation for $\mathbf{p} - \mathbf{c}$ to set up the system

$$\begin{bmatrix} (\mathbf{a}-\mathbf{c})\cdot(\mathbf{a}-\mathbf{c}) & (\mathbf{a}-\mathbf{c})\cdot(\mathbf{b}-\mathbf{c}) \\ (\mathbf{a}-\mathbf{c})\cdot(\mathbf{b}-\mathbf{c}) & (\mathbf{b}-\mathbf{c})\cdot(\mathbf{b}-\mathbf{c}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} (\mathbf{a}-\mathbf{c})\cdot(\mathbf{p}-\mathbf{c}) \\ (\mathbf{b}-\mathbf{c})\cdot(\mathbf{p}-\mathbf{c}) \end{bmatrix}$$

The two equations are of the form $m_{00}u + m_{01}v = r_0$ and $m_{01}u + m_{11}v = r_1$. The solution is $u = (m_{11}r_0 - m_{01}r_1)/\Delta$ and $v = (m_{00}r_1 - m_{01}r_0)/\Delta$ where $\Delta = m_{00}m_{11} - m_{01}^2$.

The solution may also be stated geometrically as

$$u = \frac{\text{area}(\mathbf{p}, \mathbf{b}, \mathbf{c})}{A}, v = \frac{\text{area}(\mathbf{a}, \mathbf{p}, \mathbf{c})}{A}, w = \frac{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{p})}{A},$$

where $A = \text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c})$ is the area of the original triangle.

2.2 Inscribed Centers

Let a triangle have vertices $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$. Let the inscribed center of the triangle be $\mathbf{p} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$ for barycentric coordinates (u, v, w). The triangle formed by $\mathbf{p}$, $\mathbf{b}$, and $\mathbf{c}$ has base length $|\mathbf{b}-\mathbf{c}|$ and height given by the radius r of the inscribed circle. Thus, $\text{area}(\mathbf{p}, \mathbf{b}, \mathbf{c}) = |\mathbf{b} - \mathbf{c}|r/2$. Similarly, $\text{area}(\mathbf{a}, \mathbf{p}, \mathbf{c}) = |\mathbf{a} - \mathbf{c}|r/2$ and $\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{p}) = |\mathbf{a}-\mathbf{b}|r/2$. The total area is the sum of these three values, $A = (|\mathbf{b}-\mathbf{c}|+|\mathbf{a}-\mathbf{c}|+|\mathbf{a}-\mathbf{b}|)r/2$. The barycentric coordinates of the inscribed center are therefore

$$u = \frac{|\mathbf{b}-\mathbf{c}|}{|\mathbf{b}-\mathbf{c}| + |\mathbf{a}-\mathbf{c}| + |\mathbf{a}-\mathbf{b}|}, v = \frac{|\mathbf{a}-\mathbf{c}|}{|\mathbf{b}-\mathbf{c}| + |\mathbf{a}-\mathbf{c}| + |\mathbf{a}-\mathbf{b}|}, w = \frac{|\mathbf{a}-\mathbf{b}|}{|\mathbf{b}-\mathbf{c}| + |\mathbf{a}-\mathbf{c}| + |\mathbf{a}-\mathbf{b}|}$$

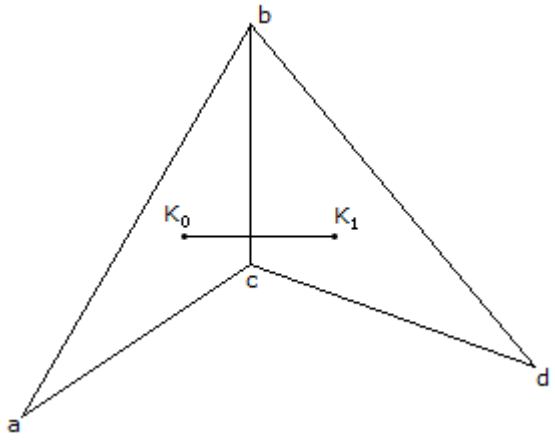
These are just ratios of the lengths of the triangle sides to the triangle perimeter.

One of the properties of the inscribed center is that each line from a vertex to the center bisects the angle corresponding to that vertex. This property may be used to prove the following result which is needed in the subdivision algorithm:

The line segment connecting the inscribed centers of two adjacent triangles must intersect the common edge of the triangles at an interior point.

If two adjacent triangles form a convex quadrilateral, then clearly the line segment connecting the inscribed centers has the desired property. If the triangles do not form a convex quadrilateral, as is shown in Figure 2.1, some work must be done to prove the result.

Figure 2.1 Adjacent triangles. Does the intersection of the line connecting inscribed centers intersect the interior of the common edge?



The inscribed centers are $\mathbf{K}_0$ and $\mathbf{K}_1$. Set up the intersection equations as

$$(1-s)\mathbf{K}_0 + s\mathbf{K}_1 = (1-t)\mathbf{c} + t\mathbf{b}.$$

Note that $\mathbf{K}_0$ and $\mathbf{K}_1$ lie on different sides of common edge $\langle \mathbf{b}, \mathbf{c} \rangle$, so the line segment connecting the centers must intersect the line containing the common edge, implying $0 < s < 1$. The geometry of the setting also implies that the intersection must occur on the $\mathbf{c}$-side of $\mathbf{b}$, which implies $t < 1$. If we can additionally prove that $t > 0$, then the line segment connecting the inscribed centers must intersect the interior of the common triangle edge.

Subtracting $\mathbf{c}$, rearranging terms, and dotting with $\mathbf{b} - \mathbf{c}$ yields

$$\begin{aligned} t|\mathbf{b}-\mathbf{c}|^2 &= (1-s)[(\mathbf{K}_0 - \mathbf{c}) \cdot (\mathbf{b} - \mathbf{c})] + s[(\mathbf{K}_1 - \mathbf{c}) \cdot (\mathbf{b} - \mathbf{c})] \\ &= (1-s)[|\mathbf{K}_0 - \mathbf{c}||\mathbf{b} - \mathbf{c}| \cos(\theta_0/2) + s[|\mathbf{K}_1 - \mathbf{c}||\mathbf{b} - \mathbf{c}| \cos(\theta_1/2) \end{aligned}$$

where θ_0 is the angle formed by edges $\mathbf{a} - \mathbf{c}$ and $\mathbf{b} - \mathbf{c}$ and θ_1 is the angle formed by edges $\mathbf{d} - \mathbf{c}$ and $\mathbf{b} - \mathbf{c}$. The half-angles in the formula occur because of the bisection property mentioned earlier. Since $0 < \theta_i < \pi$ for interior angles in a triangle, it follows that $0 < \theta_i/2 < \pi/2$ and $\cos(\theta_i/2) > 0$. The convex combination in the above formula is therefore positive, which implies that $t > 0$.

2.3 Bezier Triangles

Define a *multiindex on three indices* as $I = (i_0, i_1, i_2)$ where $0 \le i_j \le |I|$ and $|I| = i_0 + i_1 + i_2$. Define $E_0 = (1,0,0)$, $E_1 = (0,1,0)$, and $E_2 = (0,0,1)$. Given a triangular array of points $\mathbf{b}_I \in \mathbb{R}^3$ where $|I| = n$, and given a barycentric coordinate $\mathbf{u} = (u_0, u_1, u_2)$, recursively define

$$\mathbf{b}_I^0(\mathbf{u}) = \mathbf{b}_I$$

and

$$\mathbf{b}_J^r(\mathbf{u}) = \sum_{k=0}^{2} u_k \mathbf{b}_{J+E_k}^{r-1}(\mathbf{u})$$

where $1 \le r \le n$ and $|J| = n - r$. The point $\mathbf{b}^n(\mathbf{u}) := \mathbf{b}_0^n(\mathbf{u})$ is a point on the *Bezier triangle* determined by the original array. The iterative algorithm is called the *de Casteljau algorithm.*

When $n = 1$, this states that the point on the Bezier triangle is just the barycentric combination of the three vertices $\mathbf{b}_{(n,0,0)}$, $\mathbf{b}_{(0,n,0)}$, and $\mathbf{b}_{(0,0,n)}$. The interpolation algorithm is concerned with the case $n = 2$. The triangle array is organized as

$$\begin{array}{ccc} & \mathbf{b}_{(0,0,2)} & \\ \mathbf{b}_{(1,0,1)} & & \mathbf{b}_{(0,1,1)} \\ \mathbf{b}_{(2,0,0)} & \mathbf{b}_{(1,1,0)} & \mathbf{b}_{(0,2,0)} \end{array}$$

I will use (u, v, w) as the components of the barycentric coordinate. For $r = 1$,

$$\begin{aligned} \mathbf{b}_{(1,0,0)}^1 &= u\mathbf{b}_{(2,0,0)} + v\mathbf{b}_{(1,1,0)} + w\mathbf{b}_{(1,0,1)} \\ \mathbf{b}_{(0,1,0)}^1 &= u\mathbf{b}_{(1,1,0)} + v\mathbf{b}_{(0,2,0)} + w\mathbf{b}_{(0,1,1)} \\ \mathbf{b}_{(0,0,1)}^1 &= u\mathbf{b}_{(1,0,1)} + v\mathbf{b}_{(0,1,1)} + w\mathbf{b}_{(0,0,2)} \end{aligned}$$

For $r = 2$,

$$\begin{aligned} \mathbf{b}_{(0,0,0)}^2 &= u\mathbf{b}_{(1,0,0)}^1 + v\mathbf{b}_{(0,1,0)}^1 + w\mathbf{b}_{(0,0,1)}^1 \\ &= \begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} \mathbf{b}_{(2,0,0)} & \mathbf{b}_{(1,1,0)} & \mathbf{b}_{(1,0,1)} \\ \mathbf{b}_{(1,1,0)} & \mathbf{b}_{(0,2,0)} & \mathbf{b}_{(0,1,1)} \\ \mathbf{b}_{(1,0,1)} & \mathbf{b}_{(0,1,1)} & \mathbf{b}_{(0,0,2)} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \end{aligned}$$

so the triangular Bezier patch is a quadratic function. This formula is a nice generalization of tensor products for rectangular grids.

2.4 Derivatives

Given a surface vector $\mathbf{x}(\mathbf{u})$ where $\mathbf{u} = (u_0, u_1, u_2)$ are barycentric coordinates ($u_0 + u_1 + u_2 = 1$) and a barycentric direction $\mathbf{d} = (d_0, d_1, d_2)$ with $d_0 + d_1 + d_2 = 0$, the derivative in the given direction is the tangent vector

$$D_{\mathbf{d}}\mathbf{x}(\mathbf{u}) = \sum_{i=0}^{2} d_i \mathbf{x}_{u_i}$$

where $\mathbf{x}_{u_i}$ denotes the partial derivative of $\mathbf{x}$ with respect to barycentric component u_i. The second-order direction derivative is

$$D^2_{\mathbf{d}}\mathbf{x}(\mathbf{u}) = \begin{bmatrix} d_0 & d_1 & d_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{u_0u_0} & \mathbf{x}_{u_0u_1} & \mathbf{x}_{u_0u_2} \\ \mathbf{x}_{u_1u_0} & \mathbf{x}_{u_1u_1} & \mathbf{x}_{u_1u_2} \\ \mathbf{x}_{u_2u_0} & \mathbf{x}_{u_2u_1} & \mathbf{x}_{u_2u_2} \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

A general formulation can be made by using Bernstein polynomials,

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

where $i + j + k = n$. The r^{th}-order directional derivative is

$$D^r_{\mathbf{d}}\mathbf{x}(\mathbf{u}) = \sum_{|I|=r} \partial^I \mathbf{x}(\mathbf{u}) B^r_I(\mathbf{d})$$

where $I = (i_0, i_1, i_2)$ and $\partial^I \mathbf{x} = \partial^{|I|}\mathbf{x}/\partial u_0^{i_0} \partial u_1^{i_1} \partial u_2^{i_2}$. For a Bezier triangle, the r^{th}-order directional derivative is given in terms of de Casteljau iterates and Bernstein polynomials:

$$D^r_{\mathbf{d}}\mathbf{b}^n(\mathbf{u}) = \frac{n!}{(n-r)!} \sum_{|I|=r} \mathbf{b}^{n-r}_I(\mathbf{u}) B^r_I(\mathbf{d})$$

For the quadratic case $n = 2$, the first and second directional derivatives of $b^2(u, v, w)$ are

$$D^1_{(d,e,f)}\mathbf{b}^2(u,v,w) = 2 \begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} \mathbf{b}_{(2,0,0)} & \mathbf{b}_{(1,1,0)} & \mathbf{b}_{(1,0,1)} \\ \mathbf{b}_{(1,1,0)} & \mathbf{b}_{(0,2,0)} & \mathbf{b}_{(0,1,1)} \\ \mathbf{b}_{(1,0,1)} & \mathbf{b}_{(0,1,1)} & \mathbf{b}_{(0,0,2)} \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix}$$

and

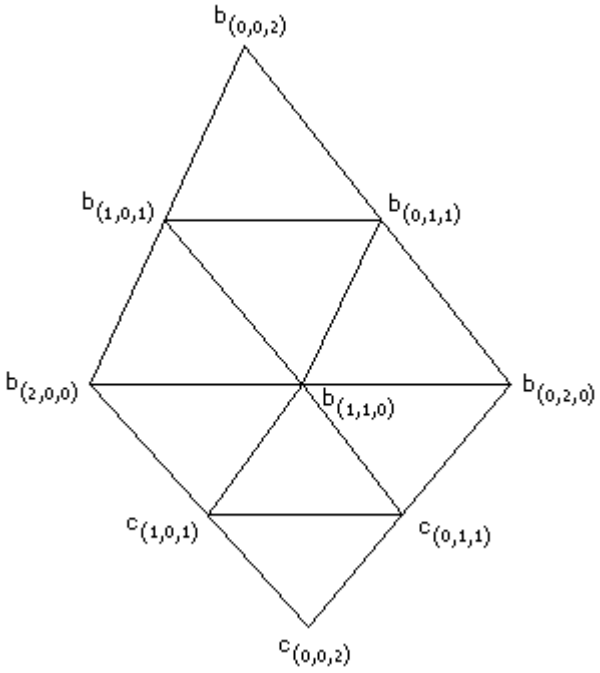
$$D^2_{(d,e,f)}\mathbf{b}^2(u,v,w) = 2 \begin{bmatrix} d & e & f \end{bmatrix} \begin{bmatrix} \mathbf{b}_{(2,0,0)} & \mathbf{b}_{(1,1,0)} & \mathbf{b}_{(1,0,1)} \\ \mathbf{b}_{(1,1,0)} & \mathbf{b}_{(0,2,0)} & \mathbf{b}_{(0,1,1)} \\ \mathbf{b}_{(1,0,1)} & \mathbf{b}_{(0,1,1)} & \mathbf{b}_{(0,0,2)} \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix}$$

Note that the second derivative is constant with respect to u, v, and w, as expected for a quadratic function.

2.5 Derivative Continuity

Farin provides a comprehensive development of derivative continuity on the common boundary between two adjacent triangular patches. The main result is that derivatives up through order s of $\mathbf{b}^n$ depend only on the $s + 1$ rows of control points "parallel" to the boundary in question. I only discuss the cases relevant to the quadratic interpolation, $s = 0$ and $s = 1$. Figure 2.2 illustrates two adjacent triangular patches ($n = 2$).

Figure 2.2 Adjacent Bezier triangle patches.



The patches define two functions $\mathbf{b}^2(u,v,w)$ and $\mathbf{c}^2(u,v,w)$. Continuity of the functions is guaranteed if

$$\mathbf{b}_{(2,0,0)} = \mathbf{c}_{(2,0,0)}, \quad \mathbf{b}_{(1,1,0)} = \mathbf{c}_{(1,1,0)}, \quad \mathbf{b}_{(0,2,0)} = \mathbf{c}_{(0,2,0)}$$

Continuity of the derivatives is guaranteed if

$$\begin{aligned} \mathbf{c}_{(1,0,1)} &= u\mathbf{b}_{(1,0,1)} + v\mathbf{b}_{(2,0,0)} + w\mathbf{b}_{(1,1,0)}, \\ \mathbf{c}_{(0,1,1)} &= u\mathbf{b}_{(0,1,1)} + v\mathbf{b}_{(1,1,0)} + w\mathbf{b}_{(0,2,0)}. \end{aligned}$$

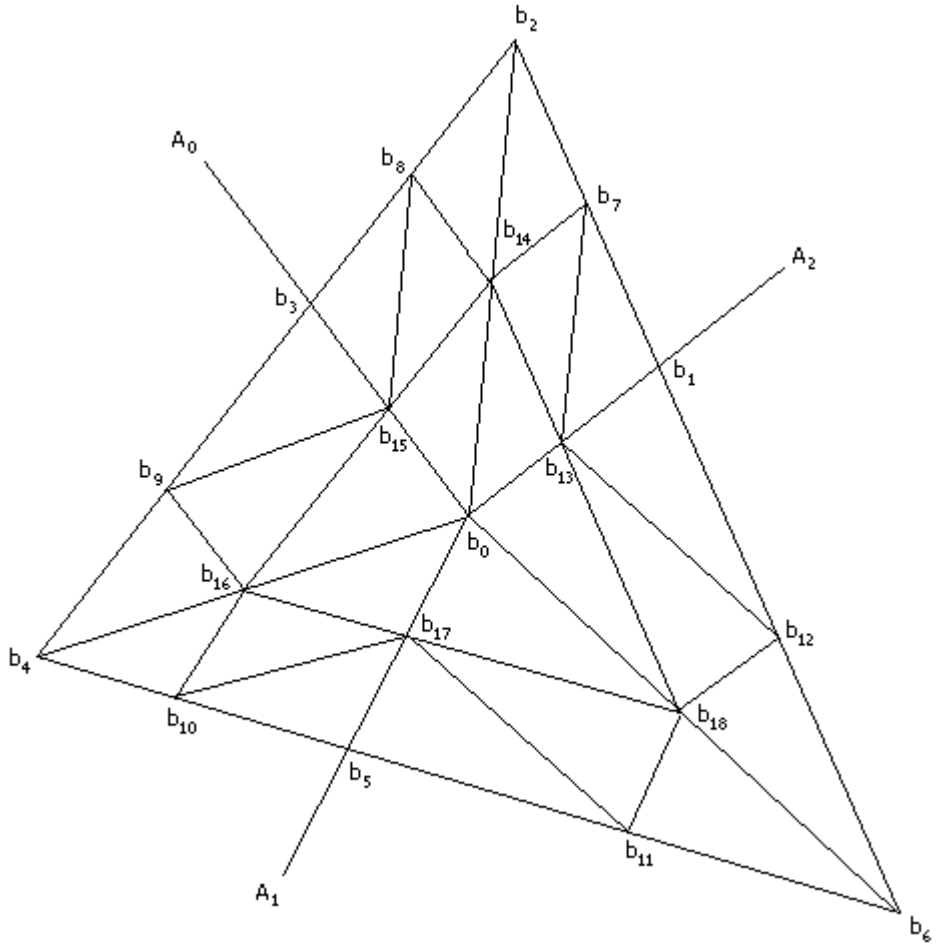
Each pair of shaded triangles in the figure are coplanar. Moreover, the two pairs have the same barycentric coordinates. I will refer to these two continuity conditions as *coplanarity* and *coaffinity*. Note that coaffinity implies coplanarity.

3 The Algorithm for Graphs of $f(x,y)$

This section is a description of the Cendes-Wong algorithm. The input is a set of points of the form $(x_i, y_i, f(x_i,y_i), f_x(x_i,y_i), f_y(x_i,y_i))$ for $0 \le i < N$. The output is a globally C^1 quadratic interpolating function which takes as input spatial points (x,y) and produces as output function values $f(x,y)$ and derivatives $f_x(x,y)$ and $f_y(x,y)$. Indirectly the algorithm also produces a triangulation of the data points.

A Delaunay triangulation is applied to the spatial components of the data points. The idea is to subdivide the triangles and fit the subtriangles as quadratic Bezier triangles so that derivative continuity is achieved on each shared triangle edge. The Cendes-Wong paper provides a construction which stresses the coplanarity condition for derivative continuity. The coaffinity condition is a consequence of the affine subdivision of the planar triangles. Consider one of the triangles, shown in Figure 3.1.

Figure 3.1 Control points in triangle subdivision.



The points $\mathbf{b}_2$, $\mathbf{b}_4$, and $\mathbf{b}_6$ are the vertices of the triangle (spatial components in the xy-plane). The point $\mathbf{b}_0$ is the inscribed center. The points $\mathbf{A}_i$ are the inscribed centers for the adjacent triangles. The points $\mathbf{b}_1$, $\mathbf{b}_3$, and $\mathbf{b}_5$ are the intersections of the triangle edges with the line segments connecting the inscribed center with those of its adjacent triangles. In the case that the triangle does not have an adjacent triangle for one of its edges (the edge is on the boundary of the mesh), then I use the midpoint of the edge in lieu of

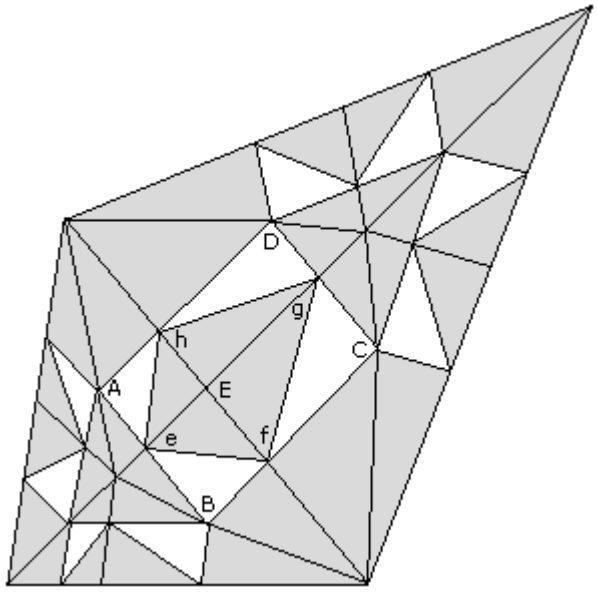
an intersection. The spatial relationships for the subdivision points are as follows.

$$\mathbf{b}_0 = \delta_0\mathbf{b}_2 + \delta_1\mathbf{b}_4 + \delta_2\mathbf{b}_6,\ \ \delta_0+\delta_1+\delta_2 = 1, \qquad \mathbf{b}_3 = \alpha_0\mathbf{b}_2 + \alpha_1\mathbf{b}_4,\ \ \alpha_0+\alpha_1 = 1,$$
$$\mathbf{b}_5 = \beta_1\mathbf{b}_4 + \beta_2\mathbf{b}_6,\ \ \beta_1+\beta_2 = 1, \qquad \mathbf{b}_1 = \gamma_0\mathbf{V}_2 + \gamma_2\mathbf{V}_6,\ \ \gamma_0+\gamma_2 = 1,$$
$$\mathbf{b}_8 = (\mathbf{b}_2+\mathbf{b}_3)/2, \qquad \mathbf{b}_9 = (\mathbf{b}_4+\mathbf{b}_3)/2,$$
$$\mathbf{b}_{10} = (\mathbf{b}_4+\mathbf{b}_5)/2, \qquad \mathbf{b}_{11} = (\mathbf{b}_6+\mathbf{b}_5)/2,$$
$$\mathbf{b}_7 = (\mathbf{b}_2+\mathbf{b}_1)/2, \qquad \mathbf{b}_{12} = (\mathbf{b}_6+\mathbf{b}_1)/2,$$
$$\mathbf{b}_{14} = (\mathbf{b}_2+\mathbf{b}_0)/2, \qquad \mathbf{b}_{16} = (\mathbf{b}_4+\mathbf{b}_0)/2,$$
$$\mathbf{b}_{18} = (\mathbf{b}_6+\mathbf{b}_0)/2, \qquad \mathbf{b}_{15} = \alpha_0\mathbf{b}_{14} + \alpha_1\mathbf{b}_{16},$$
$$\mathbf{b}_{17} = \beta_1\mathbf{b}_{16} + \beta_2\mathbf{b}_{18}, \qquad \mathbf{b}_{13} = \gamma_0\mathbf{b}_{14} + \gamma_2\mathbf{b}_{18}$$

The 3D mesh points are denoted $(\mathbf{b}_i, \phi_i)$. The indices are convenient for identifying the six Bezier control points for each of the 6 subdivision triangles. If i is the index for a triangle, $1 \le i \le 6$, then the indices of the control points for that triangle are 0, $12+i$, $13+i$ mod 6, i, $6+i$, and $1+i$ mod 6.

The goal now is to specify functions and derivatives at the three vertices and to choose function values at the remaining sixteen so that the coplanarity and coaffinity conditions are satisfied in the Bezier triangle constuction. Figure 3.2 has diagrams from from the Cendes and Wong paper.

Figure 3.2 A combination of two diagrams from the Cendes and Wong paper.



The shaded regions must be coplanar for derivative continuity to occur. The shaded quadrilateral straddling the interface of two triangles must be planar. The result that shows this is

Let A, B, C, and D be any four points in $\mathbb{R}^3$. Let e, f, g, and h be points along the line segments AB, BC, CD, and DA, respectively. If

$$\frac{\text{Length}(Ae)}{\text{Length}(AB)} = \frac{\text{Length}(Dg)}{\text{Length}(DC)} = \rho_1 \text{ and } \frac{\text{Length}(Bf)}{\text{Length}(BC)} = \frac{\text{Length}(Ah)}{\text{Length}(AD)} = \rho_2,$$

then the four points e, f, g, and h are coplanar.

The proof involves showing $eg = (\rho_1/\rho_2)ef + ((1-\rho_1)/\rho_2)eh$, in which case eg, ef, and eh are linearly dependent vectors and must be coplanar. The quadrilateral $ABCD$ is constructed so that the desired length ratios hold, and the result applies.

Now for the construction of the function values at the control points. Let ϕ_i denote the function values at the 19 control points, $0 \le i \le 18$. The vertex values ϕ_2, ϕ_4, and ϕ_6 are already specified. The derivative values at the vertices are also specified, call them $\nabla\phi_i$, $i = 2, 4, 6$.

To satisfy coplanarity at vertex $\mathbf{V}_0$:

$$\begin{aligned}
\phi_7 &= \phi_2 + \nabla\phi_2 \cdot (\mathbf{b}_7 - \mathbf{b}_2) \\
\phi_8 &= \phi_2 + \nabla\phi_2 \cdot (\mathbf{b}_8 - \mathbf{b}_2) \\
\phi_{14} &= \phi_2 + \nabla\phi_2 \cdot (\mathbf{b}_{14} - \mathbf{b}_2)
\end{aligned}$$

To satisfy coplanarity at vertex $\mathbf{V}_1$:

$$\begin{aligned}
\phi_9 &= \phi_4 + \nabla\phi_4 \cdot (\mathbf{b}_9 - \mathbf{b}_4) \\
\phi_{10} &= \phi_4 + \nabla\phi_4 \cdot (\mathbf{b}_{10} - \mathbf{b}_4) \\
\phi_{16} &= \phi_4 + \nabla\phi_4 \cdot (\mathbf{b}_{16} - \mathbf{b}_4)
\end{aligned}$$

To satisfy coplanarity at vertex $\mathbf{V}_2$:

$$\begin{aligned}
\phi_{11} &= \phi_6 + \nabla\phi_6 \cdot (\mathbf{b}_{11} - \mathbf{b}_6) \\
\phi_{12} &= \phi_6 + \nabla\phi_6 \cdot (\mathbf{b}_{12} - \mathbf{b}_6) \\
\phi_{18} &= \phi_6 + \nabla\phi_6 \cdot (\mathbf{b}_{18} - \mathbf{b}_6)
\end{aligned}$$

To satisfy coplanarity of the quadrilaterals containing $\mathbf{E}_0$, $\mathbf{E}_1$, and $\mathbf{E}_2$:

$$\begin{aligned}
\phi_3 &= \alpha_0\phi_8 + \alpha_1\phi_9 \\
\phi_5 &= \beta_1\phi_{10} + \beta_2\phi_{11} \\
\phi_1 &= \gamma_0\phi_7 + \gamma_2\phi_{12}
\end{aligned}$$

To satisfy coplanarity of the large triangle containing $\mathbf{C}$:

$$\begin{aligned}
\phi_{15} &= \alpha_0\phi_{14} + \alpha_1\phi_{16} \\
\phi_{17} &= \beta_1\phi_{16} + \beta_2\phi_{18} \\
\phi_{13} &= \gamma_0\phi_{14} + \gamma_2\phi_{18} \\
\phi_0 &= \delta_0\phi_{14} + \delta_1\phi_{16} + \delta_2\phi_{18}
\end{aligned}$$

Verifying coaffinity in the spatial components is straightforward. The triangle vertices are related by $\mathbf{a}_0 = u\mathbf{b}_0 + v\mathbf{b}_3 + w\mathbf{b}_5$. The midpoints are $\mathbf{b}_1 = (\mathbf{b}_0 + \mathbf{b}_3)/2$, $\mathbf{b}_2 = (\mathbf{b}_0 + \mathbf{b}_5)/2$, $\mathbf{b}_4 = (\mathbf{b}_3 + \mathbf{b}_5)/2$, $\mathbf{a}_1 = (\mathbf{a}_0 + \mathbf{b}_3)/2$, and $\mathbf{a}_2 = (\mathbf{a}_0 + \mathbf{b}_5)/2$. Consider

$$\begin{aligned}
\mathbf{a}_1 &= (\mathbf{a}_0 + \mathbf{b}_3)/2 \\
&= (u/2)\mathbf{b}_0 + ((v+1)/2)\mathbf{b}_3 + (w/2)\mathbf{b}_5 \\
&= (u/2)\mathbf{b}_0 + ((v+u+v+w)/2)\mathbf{b}_3 + (w/2)\mathbf{b}_5 \\
&= u(\mathbf{b}_0 + \mathbf{b}_3)/2 + v\mathbf{b}_3 + w(\mathbf{b}_3 + \mathbf{b}_5)/2 \\
&= u\mathbf{b}_1 + v\mathbf{b}_3 + w\mathbf{b}_4
\end{aligned}$$

Similarly,

$$\begin{aligned}
\mathbf{a}_2 &= (\mathbf{a}_0 + \mathbf{b}_5)/2 \\
&= (u/2)\mathbf{b}_0 + (v/2)\mathbf{b}_3 + ((w+1)/2)\mathbf{b}_5 \\
&= (u/2)\mathbf{b}_0 + (v/2)\mathbf{b}_3 + ((w+u+v+w)/2)\mathbf{b}_5 \\
&= u(\mathbf{b}_0 + \mathbf{b}_5)/2 + v(\mathbf{b}_3 + \mathbf{b}_5)/2 + w\mathbf{b}_5 \\
&= u\mathbf{b}_2 + v\mathbf{b}_4 + w\mathbf{b}_5
\end{aligned}$$

Therefore, the midpoint subdivision satisfies the coaffinity conditions. We need to verify that the function values assigned to the control points also satisfy the coaffinity conditions. This turns out to be a consequence of the midpoint subdivision and the coplanarity of certain triangles in the Bezier net.

For example, let $\mathbf{b}_1 = u\mathbf{b}_3 + v\mathbf{b}_0 + w\mathbf{b}_2$ for some barycentric coordinates (u, v, w). The midpoint subdivision guarantees that $\mathbf{b}_7 = u\mathbf{b}_8 + v\mathbf{b}_{14} + w\mathbf{b}_2$. The plane at the vertex $\mathbf{b}_2$ is of the form $\phi = K + \mathbf{N} \cdot \mathbf{b}$. Therefore,

$$\begin{aligned}
\phi_7 - u\phi_8 - v\phi_{14} - w\phi_2 &= (K + \mathbf{N} \cdot \mathbf{b}_7) - u(K + \mathbf{N} \cdot \mathbf{b}_8) - v(K + \mathbf{N} \cdot \mathbf{b}_{14}) - w(K + \mathbf{N} \cdot \mathbf{b}_2) \\
&= K(1 - u - v - w) + \mathbf{N} \cdot (\mathbf{b}_7 - u\mathbf{b}_8 - v\mathbf{b}_{14} - w\mathbf{b}_2) \\
&= K(0) + \mathbf{N} \cdot \mathbf{0} \\
&= 0
\end{aligned}$$

so $\phi_7 = u\phi_8 + v\phi_{14} + w\phi_2$. The midpoint subdivision also guarantees that $\mathbf{b}_{13} = u\mathbf{b}_{15} + v\mathbf{b}_0 + w\mathbf{b}_{14}$. The plane containing control points $\mathbf{b}_i$ for $i = 0$ and $13 \leq i \leq 18$ is also of the form $\phi = K + \mathbf{N} \cdot \mathbf{b}$. A similar argument shows that $\phi_{13} = u\phi_{15} + v\phi_0 + w\phi_{14}$. Thus, the two subtriangles satisfy the coaffinity condition. The same argument holds for any pair of subtriangles, both within a single triangle and across a triangle boundary.

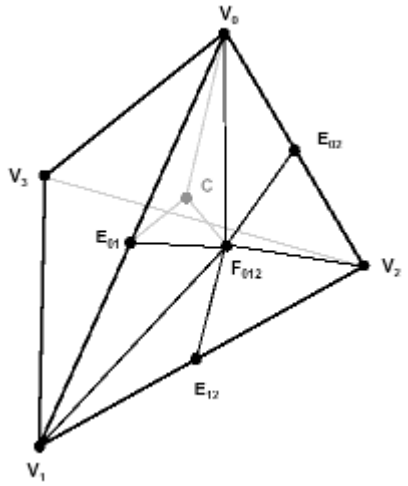
4 The Algorithm for Graphs of $f(x, y, z)$

The Cendes-Wong algorithm may be extended to the interpolation of graphs of functions of three variables.

It is assumed that the 3D points have been tetrahedralized. Consider a tetrahedron whose vertices are V_i, $0 \leq i \leq 3$. Let C be the center of the inscribed sphere for the tetrahedron. Let F_{ijk} be a point on the face with vertices V_i, V_j, and V_k. If the face is shared with another tetrahedron, let F_{ijk} be the intersection of that face and the line connecting the inscribed centers of the tetrahedra sharing that face. If the face is not shared, let F_{ijk} be the average of the vertices for that face. Let E_{ij} be the midpoint of the edge joining vertices V_i and V_j where $i < j$. Such a consistent choice, when multiple tetrahedra share the same edge, allows the construction of the Bezier net without having to analyze neighbor relationships. However, if only two tetrahedra share the same edge, then any interior edge point suffices. I will take advantage of this later when I discuss the extension of the Cendes-Wong algorithm to general triangular meshes, the extension relying on an embedding of the problem as a 3D graph in 4D.

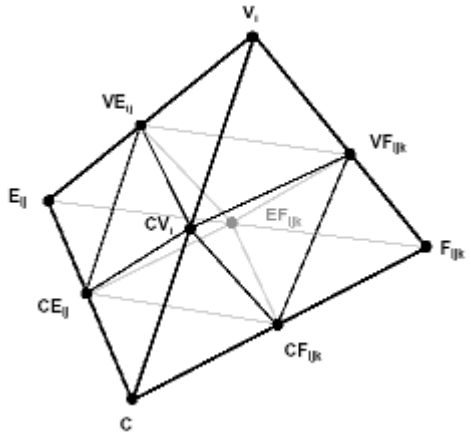
The tetrahedron can be subdivided into 24 smaller tetrahedrons, each having four vertices consisting of C, a vertex V_i, an edge point E_{ij}, and a face point F_{ijk}. Figure 4.1 shows a typical tetrahedron and one subtetrahedron.

Figure 4.1 Subdivision of a tetrahedron.



Each face of a subtetrahedron is partitioned into four Bezier triangles by selecting the midpoints of each of the edges. Figure 4.2 shows the labeling of those midpoints.

Figure 4.2 Subdivision of a subtetrahedron.



The equations relating all the labeled points are given below.

$$
\begin{aligned}
C &= c_0V_0 + c_1V_1 + c_2V_2 + c_3V_3;\ c_i \geq 0,\ c_0 + c_1 + c_2 + c_3 = 1 && \text{1 equation} \\
F_{ijk} &= f_0V_i + f_1V_j + f_2V_k;\ f_i \geq 0,\ f_0 + f_1 + f_2 = 1 && \text{4 equations} \\
E_{ij} &= (V_i + V_j)/2 && \text{6 equations} \\
VE_{ij} &= (V_i + E_{ij})/2 && \text{12 equations} \\
VF_{ijk} &= (V_i + F_{ijk})/2 && \text{12 equations} \\
CV_i &= (C + V_i)/2 && \text{4 equations} \\
CE_{ij} &= (E_{ij} + C)/2 && \text{6 equations} \\
CF_{ijk} &= (C + F_{ijk})/2 && \text{4 equations} \\
EF_{ijk} &= (E_{ij} + F_{ijk})/2 && \text{12 equations}
\end{aligned}
$$

There are a total of 61 equations relating the points. The same number of equations relates the sample values at those points. (Compare to the 19 values in the 2D case.)

It is also assumed that at each vertex a function value and gradient have been specified. At vertex V_i let the function value be ϕ_{V_i} and let the gradient vector be $D\phi_{V_i}$. The remaining sample values will be denoted ϕ_P where P is one of the points mentioned earlier. To guarantee derivative continuity at a vertex, the Bezier net about that vertex must be covolumetric. This is possible by choosing

$$
\begin{aligned}
\phi_{VE_{ij}} &= \phi_{V_i} + D\phi_{V_i} \cdot (VE_{ij} - V_i) \\
\phi_{VF_{ijk}} &= \phi_{V_i} + D\phi_{V_i} \cdot (VF_{ijk} - V_i) \\
\phi_{CV_i} &= \phi_{V_i} + D\phi_{V_i} \cdot (C - V_i)
\end{aligned}
$$

From these equations and the barycentric relationships, it follows that

$$\begin{aligned}\phi_{F_{ijk}} &= f_0\phi_{VF_{ijk}} + f_1\phi_{VF_{jki}} + f_2\phi_{VF_{kij}} \\ \phi_C &= c_0\phi_{CV_0} + c_1\phi_{CV_1} + c_2\phi_{CV_2} + c_3\phi_{CV_3} \\ \phi_{CF_{ijk}} &= f_0\phi_{CV_i} + f_1\phi_{CV_j} + f_2\phi_{CV_k}\end{aligned}$$

To guarantee derivative continuity along the edges, we need

$$\phi_{E_{ij}} = (\phi_{VE_{ij}} + 0.5\phi_{VE_{ji}})/2$$

The 1/2 coefficients occur because E_{ij} was chosen as the midpoint between two vertices, so it is also the midpoint between VE_{ij} and VE_{ji}. Also note that $\phi_{E_{ij}}$ must depend only on function values at V_i and V_j since it is possible for arbitrarily many tetrahedrons to share an edge. Any dependence of an edge value on a particular tetrahedron would invalidate the construction.

It is also necessary that the subtetrahedrons containing the edge between VE_{ij} and VE_{ji} are covolumetric. Define

$$D\phi_{ij} = \frac{\phi_{VE_{ij}} - \phi_{VE_{ji}}}{|VE_{ij} - VE_{ji}|^2}\,(VE_{ij} - VE_{ji})$$

The remaining sample values are determined by

$$\begin{aligned}\phi_{CE_{ij}} &= \phi_{E_{ij}} + D\phi_{ij}\cdot(C - E_{ij}) \\ \phi_{EF_{ijk}} &= \phi_{E_{ij}} + D\phi_{ij}\cdot(EF_{ijk} - E_{ij})\end{aligned}$$

The argument of coaffinity is the same as in the 2D case: covolumetricity and midpoint subdivision imply coaffinity. However, this is more challenging to visualize since the covolumetricity refers to two 3-dimensional tetrahedrons belonging to the same 3-dimensional hyperplane which lives in 4D.

5 The Algorithm for Surfaces

This section is a description of an extension of the Cendes-Wong algorithm to triangular meshes which are not necessarily the graph of a function. The input is a triangular mesh in $\mathbb{R}^3$. The output is a globally C^1 quadratic interpolating function which takes as input a mesh triangle and a barycentric coordinate for that triangle and produces as output a surface point (x, y, z) and the surface normal $\mathbf{N}(x, y, z)$ at that point.

The idea is to apply the 3D algorithm to each spatial component. That is, the functions to interpolate are $X = X(x, y, z)$, $Y = Y(x, y, z)$, and $Z = Z(x, y, z)$ where the mesh points provide the samples for X, Y, and Z. The triangles in the mesh are considered as boundary faces of a mesh of tetrahedra. The tetrahedra can be interpolated using the 3D algorithm. The interpolation restricted to those faces which belong to the triangular mesh is C^1 and quadratic. While it would be burdensome to produce a tetrahedral mesh, it is not necessary. Since the interpolation on a tetrahedral face only depends on the Bezier nodes living on that face, we can build the interpolation function directly from the triangular mesh.

Because the triangles are treated as boundary faces in the tetrahedral mesh, the subdivision of a triangle does not require building the inscribed centers. It is sufficient to compute the center of mass for the subdivision,

which saves some computation time. However, such a subdivision does not take into account the geometry of neighboring triangles. My implementation still uses subdivision based on inscribed centers so as to let the neighbor geometry affect the interpolation. In the Cendes-Wong algorithm, the subdivision required computing the intersection of a triangle edge with the line segment connecting the inscribed centers of the adjacent triangle. Since we are now working with the meshes in $\mathbb{R}^3$, the line segment will not intersect the edge. Instead we find the intersection of the triangle edge with a plane containing the two inscribed centers and the average normal of the triangles sharing that edge.

Another aspect of the generalization is the selection of vertex information. In the case of graphs, the user needed to specify the function and its derivatives at each vertex. In the case of general meshes, the user needs to specify the functions X, Y, and Z and their derivatives at the sample points. Rather than require the user to select the derivatives, I select a normal vector at each vertex (which acts as derivative information) by analyzing all the triangles which meet at that vertex.

5.1 Selecting Normal Vectors at Vertices

Some of the additional control points a triangle will be generated by projecting triangle points onto three specified planes at the triangle vertices. The normal vectors at the vertices must be chosen to guarantee that the projections exist. Let T_1 through T_n be a list of triangles in the mesh which share the common vertex $\mathbf{V}$. For all projections to exist, it is necessary that the selected normal $\mathbf{N}$ form an angle of at most 90 degrees with all triangle normals $\mathbf{N}_i$, $1 \le i \le n$.

This is not always possible. The surface at a vertex can be ruffled enough to prevent this. For example, let the common vertex be $\mathbf{0} = (0,0,0)$. Define the six points $\mathbf{V}_0 = (0,-1,0)$, $\mathbf{V}_1 = (0,0,-1)$, $\mathbf{V}_2 = (-1,0,0)$, $\mathbf{V}_3 = (0,1,0)$, $\mathbf{V}_4 = (-1,1,-1)$, and $\mathbf{V}_5 = (1,0,0)$. Six triangles forming a mesh and sharing $\mathbf{V}_0$ are $T_i = \langle \mathbf{0}, \mathbf{V}_i \mathbf{V}_{i+1} \rangle$ for $0 \le i \le 5$ and where the indices are computed modulo 6. The triangle normals are $\mathbf{N}_0 = (-1,0,0)$, $\mathbf{N}_1 = (0,-1,0)$, $\mathbf{N}_2 = (0,0,1)$, $\mathbf{N}_3 = (1,0,-1)/\sqrt{2}$, $\mathbf{N}_4 = (0,1,1)/\sqrt{2}$, and $\mathbf{N}_5 = (0,0,1)$. If there were a vector $\mathbf{N} = (x,y,z)$ such that $\mathbf{N} \cdot \mathbf{N}_i \ge 0$ for all i, then $-x \ge 0$, $-y \ge 0$, $z \ge 0$, $x - z \ge 0$, $y + z \ge 0$, and $z \ge 0$. A few algebraic steps will show that the only solution is $x = y = z = 0$.

If the normal vectors of the triangles containing common vertex are plotted as points on the unit sphere, then the existence of a normal satisfying the acuteness condition reduces to showing that the minimum angle cone containing the normal points has angle no larger than 180 degrees. In this case, a normal satisfying the condition is the cone axis. The following algorithm will construct the cone axis for the minimal cone if its angle is smaller than 180 degrees. If the angle is 180 degrees or larger, then the algorithm will detect this and terminate. Vertices for which this normal exists are defined to be *regular* vertices of the mesh.

The minimal cone, assuming angle smaller than 180 degrees, must contain on its boundary those two normal vectors whose angle between them is largest amongst all normal vectors. However, it is possible that the cone angle is greater than the largest angle between normals. For example, the largest angle for pairs of the three normals $(1,0,0)$, $(0,1-\epsilon,\epsilon(2-\epsilon))$, and $(0,1-\epsilon,-\epsilon(2-\epsilon))$, for $\epsilon > 0$ and small, is 90 degrees. However, the cone has angle slightly larger than 90 degrees.

The algorithm begins by computing the two normal vectors with largest angle between them, label the vectors $\mathbf{N}_1$ and $\mathbf{N}_2$. If angle is not smaller than 180 degrees, the algorithm terminates and the vertex is flagged as not regular. If angle is smaller than 180 degrees, the cone axis is initialized to be the unitized average of the two normal vectors, $\mathbf{A} = (\mathbf{N}_1 + \mathbf{N}_2)/|\mathbf{N}_1 + \mathbf{N}_2|$. The cone angle is tracked via its dot product, $d = \mathbf{A} \cdot \mathbf{N}_1$.

The remaining vertices are tested one at a time to see if their inclusion forces the cone angle to increase. Let $\mathbf{N}_3$ be the test vector. If $\mathbf{N}_3 \cdot \mathbf{A} < d$, then the test vector is outside the current cone. The cone must be expanded to include the test vector. The new cone axis must be equiangular with $\mathbf{N}_i$ for $i = 1, 2, 3$. Let the new axis be $\mathbf{A} = \sum_{i=1}^{3} u_i \mathbf{N}_i$. To be equiangular, we need $\mathbf{A} \cdot \mathbf{N}_i = \lambda$, a constant for all i. Define $d_{ij} = \mathbf{N}_i \cdot \mathbf{N}_j$. We may solve the system of three equations

$$\begin{bmatrix} 1 & d_{12} & d_{13} \\ d_{12} & 1 & d_{23} \\ d_{13} & d_{23} & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

and then unitize the resulting vector to obtain $\mathbf{A}$. The solution to the system is

$$\begin{aligned} u_1 &= (1 - d_{12} - d_{13}) + d_{23}(d_{12} + d_{13} - d_{23}) \\ u_2 &= (1 - d_{12} - d_{23}) + d_{13}(d_{12} + d_{23} - d_{13}) \\ u_3 &= (1 - d_{13} - d_{23}) + d_{12}(d_{13} + d_{23} - d_{12}) \end{aligned}$$

If the new cone angle forms an obtuse angle with either of the boundary normals $\mathbf{N}_1$ or $\mathbf{N}_2$, then the algorithm terminates and the vertex is flagged as not regular. Otherwise, after all normals have been tested, the vector $\mathbf{A}$ is the normal which forms acute angles with all the triangle normals and the vertex is flagged as regular.

5.2 Subdividing the Triangles

The indexing of control points is the same as in the Cendes-Wong algorithm. The three vertices are labeled as $\mathbf{p}_2$, $\mathbf{p}_4$, and $\mathbf{p}_6$. The inscribed center $\mathbf{p}_0$ is computed for the triangle. Keep in mind that the triangle lives in $\mathbb{R}^3$, so the inscribed center is on that triangle and not in the xy-plane. Three subtriangles are formed by connecting the inscribed center to the three vertices. The edge points $\mathbf{p}_1$, $\mathbf{p}_3$, and $\mathbf{p}_5$ are constructed in a slightly different manner than in the graph case. Let $\mathbf{A}_0$ be the inscribed center of the triangle sharing edge $\langle \mathbf{p}_2, \mathbf{p}_4 \rangle$ and let $\mathbf{N}_0$ be its normal. Let $\mathbf{N}$ be the normal of the current triangle. The point $\mathbf{p}_3$ is the intersection of the edge $\langle \mathbf{p}_2, \mathbf{p}_4 \rangle$ with the plane containing points $\mathbf{p}_0$ and $\mathbf{A}_0$ and containing the "average" normal $\mathbf{N} + \mathbf{N}_0$. This plane has equation

$$(\mathbf{N} + \mathbf{N}_0) \times (\mathbf{p}_0 - \mathbf{A}_0) \cdot (\mathbf{x} - \mathbf{p}_0) = 0$$

If the point of intersection is $\mathbf{p}_3 = \mathbf{p}_2 + t(\mathbf{p}_4 - \mathbf{p}_2)$, then

$$t = \frac{(\mathbf{N} + \mathbf{N}_0) \times (\mathbf{p}_0 - \mathbf{A}_0) \cdot (\mathbf{p}_0 - \mathbf{p}_2)}{(\mathbf{N} + \mathbf{N}_0) \times (\mathbf{p}_0 - \mathbf{A}_0) \cdot (\mathbf{p}_4 - \mathbf{p}_2)}$$

Similar constructions apply to the other two edges. Intuitively these intersections always exist. Given two triangles meeting at an edge, you can rigidly move one to the plane and rotate the other into the same plane using the edge as rotation axis. In the plane we already proved that the intersections always exist. While this intersection was between two lines, it can also be viewed as the intersection of the plane containing the inscribed centers and the normal of this plane. Reversing the transformations cannot undo the intersection.

The remaining subdivision uses midpoint selection, just as in the graph case.

5.3 Selecting Control Points

At a vertex point $\mathbf{v}$, the user must specify the new positions X, Y, Z, and their derivatives. A Bezier node $\mathbf{b}$ which is immediately adjacent to $\mathbf{v}$ is modified as

$$\boldsymbol{\phi}_v = \mathbf{v} + D(\mathbf{b} - \mathbf{v})$$

where D is a 3×3 derivative matrix. The first row of D is the user-specified derivative ∇X, the second row is ∇Y, and the third row is ∇Z. If instead an "average" normal $\mathbf{N}$ is desired (a more intuitive choice about how a vertex is smoothed), then set $D = \mathbf{N}\mathbf{N}^{\mathrm{T}}$. Each $\boldsymbol{\phi}_v$ is therefore a projection of $\mathbf{v}$ onto the plane containing $\mathbf{b}$ and having normal $\mathbf{N}$. Note that such selection is a *convenience*. It is possible to select D at each vertex regardless of topology at that vertex, and the construction still produces a C^1 quadratic interpolation.

Given a triangle in the mesh with vertices $\mathbf{b}_2$, $\mathbf{b}_4$, $\mathbf{b}_6$, and computed normals $\mathbf{N}_2$, $\mathbf{N}_4$, $\mathbf{N}_6$, the other Bezier nodes are computed just as in the 2D graph case. The α_i, β_i, γ_i, and δ_i are the proportions from the subdivision, as before. The nodes are

$$\begin{aligned}
\boldsymbol{\phi}_2 &= \mathbf{b}_2 \\
\boldsymbol{\phi}_4 &= \mathbf{b}_4 \\
\boldsymbol{\phi}_6 &= \mathbf{b}_6 \\
\boldsymbol{\phi}_7 &= \mathbf{b}_2 + \mathbf{N}_2 \cdot (\mathbf{b}_7 - \mathbf{b}_2) \\
\boldsymbol{\phi}_8 &= \mathbf{b}_2 + \mathbf{N}_2 \cdot (\mathbf{b}_8 - \mathbf{b}_2) \\
\boldsymbol{\phi}_{14} &= \mathbf{b}_2 + \mathbf{N}_2 \cdot (\mathbf{b}_{14} - \mathbf{b}_2) \\
\boldsymbol{\phi}_9 &= \mathbf{b}_4 + \mathbf{N}_4 \cdot (\mathbf{b}_9 - \mathbf{b}_4) \\
\boldsymbol{\phi}_{10} &= \mathbf{b}_4 + \mathbf{N}_4 \cdot (\mathbf{b}_{10} - \mathbf{b}_4) \\
\boldsymbol{\phi}_{16} &= \mathbf{b}_4 + \mathbf{N}_4 \cdot (\mathbf{b}_{16} - \mathbf{b}_4) \\
\boldsymbol{\phi}_{11} &= \mathbf{b}_6 + \mathbf{N}_6 \cdot (\mathbf{b}_{11} - \mathbf{b}_6) \\
\boldsymbol{\phi}_{12} &= \mathbf{b}_6 + \mathbf{N}_6 \cdot (\mathbf{b}_{12} - \mathbf{b}_6) \\
\boldsymbol{\phi}_{18} &= \mathbf{b}_6 + \mathbf{N}_6 \cdot (\mathbf{b}_{18} - \mathbf{b}_6) \\
\boldsymbol{\phi}_3 &= \alpha_0 \boldsymbol{\phi}_2 + \alpha_1 \boldsymbol{\phi}_4 \\
\boldsymbol{\phi}_5 &= \beta_1 \boldsymbol{\phi}_4 + \beta_2 \boldsymbol{\phi}_6 \\
\boldsymbol{\phi}_1 &= \gamma_0 \boldsymbol{\phi}_2 + \gamma_2 \boldsymbol{\phi}_6 \\
\boldsymbol{\phi}_{15} &= \alpha_0 \boldsymbol{\phi}_{14} + \alpha_1 \boldsymbol{\phi}_{16} \\
\boldsymbol{\phi}_{17} &= \beta_1 \boldsymbol{\phi}_{16} + \beta_2 \boldsymbol{\phi}_{18} \\
\boldsymbol{\phi}_{13} &= \gamma_0 \boldsymbol{\phi}_{14} + \gamma_2 \boldsymbol{\phi}_{18} \\
\boldsymbol{\phi}_0 &= \delta_0 \boldsymbol{\phi}_{14} + \delta_1 \boldsymbol{\phi}_{16} + \delta_2 \boldsymbol{\phi}_{18}
\end{aligned}$$

6 Implementation Notes

Just a few notes on the implementations.

1. I am using some simple data structures to store the vertices, edges, and faces. Probably I should change these to ones that are compatible with OpenGL and other graphics libraries.

2. In the graph case, each triangle maintains an array of 19 doubles which represent the function values at the control points. Two triangles sharing a common edge will maintain duplicates of the control values on that common edge. Probably I should let the edge data structures maintain their own control points to reduce memory usage. This is even more important in the general mesh case since the array is now of 19 points, each point consisting of 3 doubles.

3. In the graph case, any triangle with a mesh boundary edge (an edge not shared by any other triangle) will cause problems when you try to compute the intersection of that edge with a line connecting the inscribed centers of the adjacent triangles. I handle this in the graph case by attaching "dummy triangles" to those boundary edges. The intersections are designed to be the midpoints of those edges. The user of the code need not worry about these issues; they are handled automatically.

   In the general mesh case, the triangle data structure has a list of indices to adjacent triangles. If an adjacent triangle does not exist, then the index must be set to a negative number. This lets the method for computing intersections know that the midpoint of the edge should be used instead.

4. Currently, if a vertex is not regular, the method that tries to compute a normal vector at that vertex fails. I have a flag in the mesh interpolation class which indicates success or failure in building the normals. This flag should be tested before attempting to evaluate the interpolation method. The sample code shows how to use the flag.

   Any choice of normal will provide a C^1 fit, so it is not necessary to terminate the algorithm. However, it would be nice to interpolate in a geometrically intuitive way.

7 References

1. Zoltan J. Cendes and Steven H. Wong, C^1 *quadratic interpolation over arbitrary point sets*, IEEE Computer Graphics & Applications, pp. 8-16, November 1987

2. Dave F. Watson, *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*, The Computer Journal, 24(2), pp. 167-172, 1981.

3. Gerald Farin, *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*, Academic Press, Inc., San Diego, CA, 1990.