Second Year Report
and
Request for Renewal
for 3rd Year

of

Multiresolution and Explicit Methods for Vector Field Analysis and Visualization

May 15, 1997

National Aeronautics and Space Administration
Ames Research Center
NAS Systems Division, Code RNR
Moffett Field, CA 94035-1000
NRA2-36206(LMV)

Technical Contact: Gregory M. Nielson, CS&E, ASU, Tempe, AZ 85287-5406
(602) 965-2785, 962-6739, email: nielson@asu.edu

Introduction

This is a request for a second renewal (3rd year of funding) of a research project on the topic of multiresolution and explicit methods for vector field analysis and visualization. In this report, we describe the progress made on this research project during the second year and give a statement of the planned research for the third year.

Report on Research Progress During the Second Year

There are two aspects to this research project. The first is concerned with the development of techniques for computing tangent curves for use in visualizing flow fields. The second aspect of the research project is concerned with the development of multiresolution methods for curvilinear grids and their use as tools for visualization, analysis and archiving of flow data. We report on our work on the development of numerical methods for tangent curve computation first.

The basic idea of these methods is to decompose the domain into a collection of triangles (or tetrahedra) and assume linear variation of the vector field over each cell. With this assumption, the equations which define a tangent curve become a system of linear, constant coefficient ODE's which can be solved explicitly. In our report on the progress made in the first year of the project, we reported on our work in 2D. We have done the mathematical analysis on the explicit and incremental methods for 2D and written and tested thoroughly the computer programs implementing certain aspects of these algorithms. We have submitted for publication (see [1]) some of our research results in this area. We mentioned in the previous progress report our plans to extend our work in this area to 3D. This involves tetrahedral decompositions of the 3D curvilinear grids (see our submitted research in this area [2]) and the development of explicit and incremental methods for linearly varying vector fields over tetrahedra. We have completed the mathematical analysis and are in the program development stages at this moment. We include here the mathematics for the 3D case. In the situation of a 2D flow over a 2D domain, there were five (5) distinct cases determined by the eigenvalues of the coeficient matrix of the ODE defining the tangent curves of the flow. In the 3D situation there are nine (9) cases.

3D Explicit Respesentations

$$P'(t) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} P(t) + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \quad P(0) = P_0$$

Case 1) A has three real, nonzero, eigenvalues $(r_1 \neq r_2 \neq r_3 \neq 0)$

$$P(t) = E_1 e^{tr_1} + E_2 e^{tr_2} + E_3 e^{tr_3} + P_c$$

$$E_1 = \left(\frac{A - r_2 I}{r_1 - r_2}\right)\left(\frac{r_3 I - A}{r_3 - r_1}\right)(P_0 - P_c),$$

$$E_2 = \left(\frac{A - r_3 I}{r_2 - r_3}\right)\left(\frac{r_1 I - A}{r_1 - r_2}\right)(P_0 - P_c),$$

$$E_3 = \left(\frac{A - r_1 I}{r_3 - r_1}\right)\left(\frac{r_2 I - A}{r_2 - r_3}\right)(P_0 - P_c), \qquad AP_c + B = 0$$

Case 2) A has one real, nonzero and two equal, nonzero, eigenvalues $(r_1 \neq r_2 = r_3 \neq 0)$

$$P(t) = E_1 e^{tr_1} + E_2 e^{tr_2} + E_3 t e^{tr_2} + P_0$$

$$E_1 = \left(\frac{A - r_2 I}{r_2 - r_1}\right)(P_0 - P_c),$$

$$E_2 = \frac{(r_1 I - A)(r_1 I - 2r_2 I + A)}{(r_2 - r_1)^2}(P_0 - P_c),$$

$$E_3 = \frac{(A - r_1 I)(A - r_2 I)}{(r_2 - r_1)}(P_0 - P_c), \qquad AP_c + B = 0$$

Case 3) A has three equal, nonzero, eigenvalues $(r_1 = r_2 = r_3 \neq 0)$

$$P(t) = E_1 e^{tr_1} + E_2 t e^{tr_1} + E_3 t^2 e^{tr_1} + P_c$$

$$E_1 = (P_0 - P_c),$$

$$E_2 = (A - r_1 I)(P_0 - P_c),$$

$$E_3 = \frac{(A - r_1 I)^2}{2}(P_0 - P_c), \qquad AP_c + B = 0$$

Case 4) A has one real, nonzero and two complex, eigenvalues $\left(r_1 \neq 0,\ \mu \pm \lambda i,\ \lambda \neq 0\right)$

$$P(t) = E_1 e^{tr_1} + E_2 e^{\mu t}\cos(\lambda t) + E_3 e^{\mu t}\sin(\lambda t) + P_c$$

$$E_1 = \frac{(A-\mu I)^2 + \lambda^2 I}{(\mu - r_1)^2 + \lambda^2}(P_0 - P_c),$$

$$E_2 = \frac{(r_1 I - A)(r_1 I - 2\mu I + A)}{(\mu - r_1)^2 + \lambda^2}(P_0 - P),$$

$$E_3 = \frac{(A - r_1 I)\left((A-\mu I)(\mu - r_1) + \lambda^2 I\right)}{\lambda\left((\mu - r_1)^2 + \lambda^2\right)}(P_0 - P), \qquad AP_c + B = 0$$

Case 5) A has one real, nonzero and two zero eigenvalues $\left(r_1 = r_2 = 0,\ r_3 \neq 0\right)$

$$P(t) = E_1 t + E_2 t^2 + E_3\left(e^{tr_3} - tr_3 - 1\right) + P_0$$

$$E_1 = \left(AP_0 + B\right),$$

$$E_2 = \frac{AB}{2}\left(I - \frac{A}{r_3}\right),$$

$$E_3 = \left(\frac{A}{r_3}\right)^2\left(P_0 + \frac{B}{r_3}\right).$$

Case 6) A has three zero eigenvalues $\left(r_1 = r_2 = r_3 = 0\right)$

$$P(t) = E_1 t + E_2 t^2 + E_3 t^3 + P_0$$

$$E_1 = \left(AP_0 + B\right),$$

$$E_2 = \frac{A}{2}\left(AP_0 + B\right),$$

$$E_3 = \frac{A^2 B}{6}.$$

Case 7) A has two real, nonzero and one zero eigenvalues $\left(r_1 = 0,\ 0 \neq r_2 \neq r_3 \neq 0\right)$

$$P(t) = E_1 t + E_2\left(e^{tr_2} - 1\right) + E_3\left(e^{tr_3} - 1\right) + P_0$$

$$E_1 = \left(\left(I - \frac{A}{r_2}\right)\left(\frac{A - r_3 I}{r_2 - r_3}\right) + \left(I - \frac{A}{r_3}\right)\left(\frac{r_2 I - A}{r_2 - r_3}\right)\right)B,$$

$$E_2 = \frac{A}{r_2}\left(P_0 + \frac{B}{r_2}\right)\left(\frac{A - r_3 I}{r_2 - r_3}\right),$$

$$E_3 = \frac{A}{r_3}\left(P_0 + \frac{B}{r_3}\right)\left(\frac{r_2 I - A}{r_2 - r_3}\right).$$

Case 8) A has two real equal, nonzero and one zero eigenvalues $\left(r_1 = 0,\ r_2 = r_3 \neq 0\right)$

$$P(t) = E_1 t + E_2\left(e^{tr_2} - 1\right) + E_3 t e^{tr_2} + P_0$$

$$E_1 = \left(I + \frac{A}{r_2}\left(\frac{A}{r_2} - 2I\right)\right)B,$$

$$E_2 = \frac{A}{r_2}\left(2\left(P_0 + \frac{B}{r_2}\right)\left(I - \frac{A}{r_2}\right) + \frac{AP_0 + B}{r_2}\right),$$

$$E_3 = A\left(\frac{A}{r_2} - I\right)\left(P_0 + \frac{B}{r_2}\right).$$

Case 9) A has one zero and two complex, eigenvalues $\left(r_1 = 0,\ \mu \pm \lambda i,\ \lambda \neq 0\right)$

$$P(t) = E_1 t + E_2\left(e^{\mu t}\cos(\lambda t) - 1\right) + E_3 e^{\mu t}\sin(\lambda t) + P_c$$

$$E_1 = \left(I - A\left(\frac{2\mu I - A}{\mu^2 + \lambda^2}\right)\right)B,$$

$$E_2 = \left(\frac{A}{\mu^2 + \lambda^2}\right)\left((2\mu I - A)P_0 + \left(\frac{3\mu^2 I - \lambda^2 I - 2\mu A}{\mu^2 + \lambda^2}\right)B\right),$$

$$E_3 = \left(\frac{A}{\lambda\left(\mu^2 + \lambda^2\right)}\right)\left(\left(\mu A - (\mu + \lambda)(\mu - \lambda)\right)P_0 + \left(\frac{A(\mu + \lambda)(\mu - \lambda) - \mu\left(\mu^2 - 3\lambda^2\right)}{\mu^2 + \lambda^2}\right)b\right)$$

Based upon the explicit representations give above, we have developed incremental methods which compute points on the exact solution of the tangent curves. We present the formulas for these incremental methods for the nine (9) cases below. (The matrix, E, consists of columns which are the eigenvectors of matrix A.)

3D Incremental Method in Cartesian Coordinates

Case 1) A has three real, nonzero, eigenvalues $\left(r_1 \neq r_2 \neq r_3 \neq 0\right)$

If $|E| \neq 0$ then

$$P(t+\Delta t) - P_c = E\begin{bmatrix} e^{\Delta t r_1} & 0 & 0 \\ 0 & e^{\Delta t r_2} & 0 \\ 0 & 0 & e^{\Delta t r_3} \end{bmatrix} E^{-1}\left(P(t) - P_c\right)$$

If $|E| = 0$ then

$$P(t+\Delta t) = P(t) \pm \Delta t\left(P_c - P_0\right)$$

Case 2) A has one real, nonzero and two equal, nonzero, eigenvalues $\left(r_1 \neq r_2 = r_3 \neq 0\right)$

If $|E| \neq 0$ then

$$P(t+\Delta t) - P_c = E\begin{bmatrix} e^{\Delta t r_1} & 0 & 0 \\ 0 & e^{\Delta t r_2} & 0 \\ 0 & \Delta t e^{\Delta t r_2} & e^{\Delta t r_2} \end{bmatrix} E^{-1}\left(P(t) - P_c\right)$$

If $|E| = 0$ then

$$P(t+\Delta t) = P(t) \pm \Delta t\left(P_c - P_0\right)$$

Case 3) A has three equal, nonzero, eigenvalues $\left(r_1 = r_2 = r_3 \neq 0\right)$

If $|E| \neq 0$ then

$$P(t+\Delta t) - P_c = E e^{\Delta t r_1}\begin{bmatrix} 1 & 0 & 0 \\ \Delta t & 1 & 0 \\ (\Delta t)^2 & 2\Delta t & 1 \end{bmatrix} E^{-1}\left(P(t) - P_c\right)$$

If $|E| = 0$ then

$$P(t+\Delta t) = P(t) \pm \Delta t\left(P_c - P_0\right)$$

Case 4) A has one real, nonzero and two complex, eigenvalues $\left(r_1 \neq 0, \mu \pm \lambda i, \lambda \neq 0\right)$

$$P(t+\Delta t) - P_c = E\begin{bmatrix} e^{\Delta t r_1} & 0 & 0 \\ 0 & \cos(\lambda\Delta t)e^{\mu\Delta t} & -\sin(\lambda\Delta t)e^{\mu\Delta t} \\ 0 & \sin(\lambda\Delta t)e^{\mu\Delta t} & \cos(\lambda\Delta t)e^{\mu\Delta t} \end{bmatrix} E^{-1}\left(P(t) - P_c\right)$$

Case 5) A has one real, nonzero and two zero eigenvalues $\left(r_1 = r_2 = 0, r_3 \neq 0\right)$

$$P(t+\Delta t) = \left(I + A\left(\Delta t + \frac{A}{r_3^2}\left(e^{\Delta t r_3} - \Delta t r_3 - 1\right)\right)\right) P(t) +$$
$$\Delta t\left(I + A\left(\frac{\Delta t}{2}\left(I - \frac{A}{r_3}\right) + \frac{A}{\Delta t r_3^3}\left(e^{\Delta t r_3} - \Delta t r_3 - 1\right)\right)\right) B$$

Case 6) A has three zero eigenvalues $\left(r_1 = r_2 = r_3 = 0\right)$

$$P(t+\Delta t) = \left(I + \Delta t A + \frac{(\Delta t A)^2}{2}\right) P(t) + \Delta t\left(I + \frac{\Delta t A}{2} + \frac{(\Delta t A)^2}{6}\right) B$$

Case 7) A has two real, nonzero and one zero eigenvalues $\left(r_1 = 0, 0 \neq r_2 \neq r_3 \neq 0\right)$

$$P(t+\Delta t) = \left(I + \frac{A}{r_2}\left(\frac{A - r_3 I}{r_2 - r_3}\right)\left(e^{\Delta t r_2} - 1\right) + \frac{A}{r_3}\left(\frac{r_2 I - A}{r_2 - r_3}\right)\left(e^{\Delta t r_3} - 1\right)\right) P(t)$$
$$+ \Delta t\left(I + \frac{A}{\Delta t r_2^2}\left(\frac{A - r_3 I}{r_2 - r_3}\right)\left(e^{\Delta t r_2} - \Delta t r_2 - 1\right) + \frac{A}{\Delta t r_3^2}\left(\frac{r_2 I - A}{r_2 - r_3}\right)\left(e^{\Delta t r_3} - \Delta t r_3 - 1\right)\right) B$$

Case 8) A has two real equal, nonzero and one zero eigenvalues $\left(r_1 = 0,\ r_2 = r_3 \neq 0\right)$

$$P(t+\Delta t) = \left(I + A\left(\frac{A}{r_2} - I\right)\Delta t e^{\Delta t r_2} + \frac{A}{r_2}\left(2I - \frac{A}{r_2}\right)\left(e^{\Delta t r_2} - 1\right)\right) P(t)$$
$$+ \Delta t\left(I + \frac{A}{r_2}\left(\frac{A}{r_2} - I\right) e^{\Delta t r_2} + \frac{A}{\Delta t r_2^2}\left(3I - \frac{2A}{r_2}\right)\left(e^{\Delta t r_2} - 1\right) + \frac{A}{r_2}\left(\frac{A}{r_2} - 2I\right)\right) B$$

Case 9) A has one zero and two complex, eigenvalues $\left(r_1 = 0, \mu \pm \lambda i, \lambda \neq 0\right)$

$$P(t+\Delta t) = \begin{pmatrix} I + \left(\frac{A}{\mu^2+\lambda^2}\right)(2\mu I - A)e^{\mu\Delta t}(\cos(\lambda\Delta t) - 1) \\ + \left(\frac{A}{\lambda(\mu^2+\lambda^2)}\right)(\mu A - (\mu^2 - \lambda^2)I)e^{\mu\Delta t}\sin(\lambda\Delta t) \end{pmatrix} P(t)$$

$$+ \Delta t \begin{pmatrix} I + \frac{A}{\Delta t}\left(\frac{3\mu^2 I - \lambda^2 I - 2\mu A}{(\mu^2+\lambda^2)^2}\right)e^{\mu\Delta t}(\cos(\lambda\Delta t) - 1) \\ + \frac{A}{\lambda\Delta t}\left(\frac{A}{(\mu^2+\lambda^2)} - \frac{\mu(\mu^2 - 3\lambda^2)}{(\mu^2+\lambda^2)^2}I\right)e^{\mu\Delta t}\sin(\lambda\Delta t) + I + A\left(\frac{2\mu - A}{\mu^2+\lambda^2}\right) \end{pmatrix} B$$

We did not discuss this in the report on the first year's progress, but we have also developed incremental methods that compute points on the true tangent curves (not approximations) using barycentric coordinates. The reason for doing the computations in barycentric coordinates is to make it cheap and easy to determine when the curve leaves one triangle and enters a neighboring triangle. All that needs to be done is to check the sign of the barycentric coordinates. If all three are not positive, then the curve has left the triangle and which barycentric is not positive tells us which neighboring triangle the tangent curve has entered. We give the formulas for the 2D incremental methods using barycentric coordinates below. These formulas are based upon explicit, exact solutions represented in terms of barycentric coordinates which we also include here.

In the formulas below, $p(t)$ represents the tangent curve in barycentric coordinates. That is

$$p(t) = \begin{pmatrix} b_i(t) \\ b_j(t) \\ b_k(t) \end{pmatrix} \text{ and } p(t) = b_i(t)V_i + b_j(t)V_j + b_k(t)V_k$$

where V_i, V_j, V_k are the velocity vectors at the vertices of the triangle currently containing the tangent curve. The matrix V (used below) includes the information which defines the tangent curve which is similar in role to the matrix A and column vector B in the cartesian coordinate case.

2D Explicit Solutions in Barycentric Coordinates:

$$p'(t) = Vp(t), \quad p(0) = p_0$$

Case 1) V has two real, nonzero and one zero eigenvalues $0 \neq r_1 \neq r_2 \neq 0, r_3 = 0$

$$p(t) = e_1 e^{tr_1} + e_2 e^{tr_1} + p_c$$

$$e_1 = \frac{V}{r_1}\left(\frac{V - r_2 I}{r_1 - r_2}\right)p_0, \qquad e_2 = \frac{V}{r_2}\left(\frac{r_1 I - V}{r_1 - r_2}\right)p_0$$

Case 2) V has one real, nonzero and two zero eigenvalues $r_1 = 0,\ r_2 \neq 0,\ r_3 = 0$

$$p(t) = e_1 t + e_2\left(e^{tr_2} - 1\right) + p_0$$

$$e_1 = V\left(I - \frac{V}{r_2}\right)p_0 \qquad e_2 = \frac{V^2}{(r_2)^2}p_0$$

Case 3) V has three zero eigenvalues $r_1 = r_2 = r_3 = 0$

$$p(t) = e_1 t + e_2 t^2 + p_0$$

$$e_1 = V p_0 \qquad e_2 = \frac{V^2}{2}p_0$$

Case 4) A has two real equal, nonzero and one zero eigenvalues $r_1 = r_2 \neq 0, r_3 = 0$

$$P(t) = e_1 e^{tr_2} + e_2 t e^{tr_2} + p_c$$

$$e_1 = \frac{V}{r_2}\left(2I - \frac{V}{r_2}\right)p_0 \qquad e_2 = V\left(\frac{V}{r_2} - I\right)p_0$$

Case 5) A has one zero and two complex, eigenvalues $\mu \pm \lambda i,\ \lambda \neq 0,\ r_3 = 0$

$$P(t) = e_1 e^{\mu t}\cos(\lambda t) + e_2 e^{\mu t}\sin(\lambda t) + p_c$$

$$e_1 = \left(\frac{V(2\mu I - V)}{\mu^2 + \lambda^2}\right)p_0 \qquad e_2 = \left(\frac{V\left(\mu A - \mu^2 + \lambda^2\right)}{\lambda\left(\mu^2 + \lambda^2\right)}\right)p_0$$

2D Incremental Method in Barycentric Coordinates

Case 1) V has two real, nonzero and one zero eigenvalues $0 \neq r_1 \neq r_2 \neq 0,\ r_3 = 0$

$$p(t+\Delta t)=\left(I+\frac{V}{r_1}\left(\frac{V-r_2 I}{r_1-r_2}\right)\left(e^{\Delta t r_1}-1\right)+\frac{V}{r_2}\left(\frac{r_1 I-V}{r_1-r_2}\right)\left(e^{\Delta t r_2}-1\right)\right)p(t)$$

Case 2) V has one real, nonzero and two zero eigenvalues $r_1 = 0,\ r_2 \neq 0, r_3 = 0$

$$p(t+\Delta t)==\left(I+V\left(\Delta t+\frac{V}{r_2^2}\left(e^{\Delta t r_2}-\Delta t r_2-1\right)\right)\right)p(t)$$

Case 3) V has three zero eigenvalues $r_1 = r_2 = r_3 = 0$

$$p(t+\Delta t)=\left(I+\Delta t V+\frac{(\Delta t V)^2}{2}\right)p(t)$$

Case 4) V has two real equal, nonzero and one zero eigenvalues $r_1 = r_2 \neq 0,\ r_3 = 0$

$$p(t+\Delta t)=\left(I+V\left(\frac{V}{r_2}-I\right)\Delta t e^{\Delta t r_2}+\frac{V}{r_2}\left(2I-\frac{V}{r_2}\right)\left(e^{\Delta t r_2}-1\right)\right)p(t)$$

Case 5) V has one zero and two complex, eigenvalues $\mu \pm \lambda i,\ \lambda \neq 0,\ r_3 = 0$

$$p(t+\Delta t)=\left(I+\left(\frac{V(2\mu I-V)}{\mu^2+\lambda^2}\right)\left(e^{\mu\Delta t}\cos(\lambda\Delta t)-1\right)+\left(\frac{V\left(\mu V-\mu^2+\lambda^2\right)}{\lambda\left(\mu^2+\lambda^2\right)}\right)e^{\mu\Delta t}\sin(\lambda\Delta t)\right)p(t)$$

We are at this moment in the process of finishing off the development of the equations similar to the above for 3D and plan on implementing all of the 3D methods mentioned above in the near future.

We now move to a discussion on our research progress in the other topic of this research project and this is in the area of multiresolution modeling. Our work in this area can be broken down into two subareas: i) triangular grids with applications to arbitrary domains and ii) curvilinear grids. We first take up the topic of triangular grids and our current extensions to tetrahedral grids.

Earlier we reported on some new Haar type wavelets for subdivision of triangular domains as illustrated in Figure 1. The basic decomposition step is illustrated in Figure 2. The ϕ's are the characteristic functions for the subtriangles and therefore form a basis for piecewise contant functions. The ψ's are the wavelet functions which capture the detail (error) and hopefully they are mutually orthogonal so that we obtain best least squares approximations when we do the wavelet decompositions.

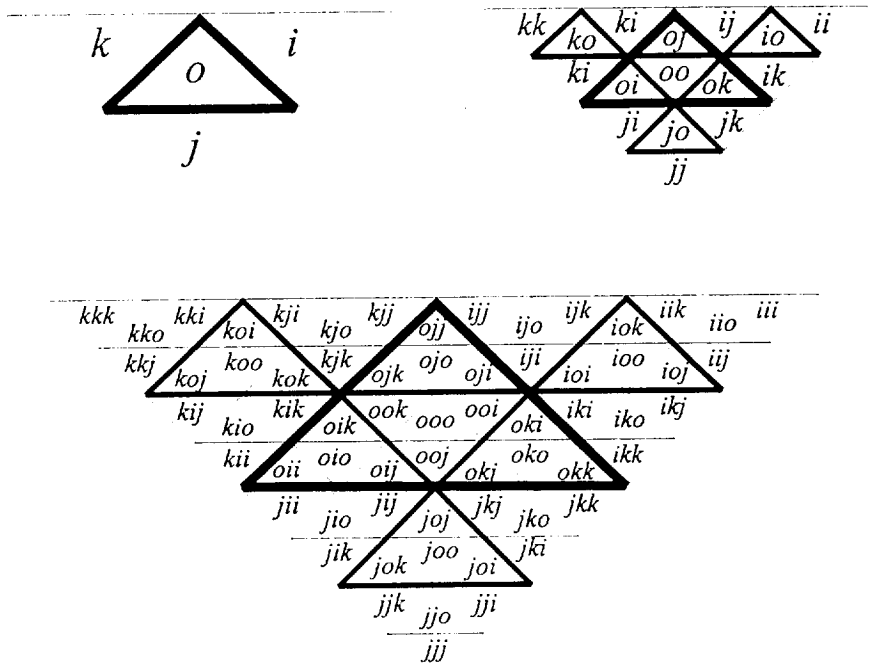


Figure 1. Nested triangular domain with labeling scheme.

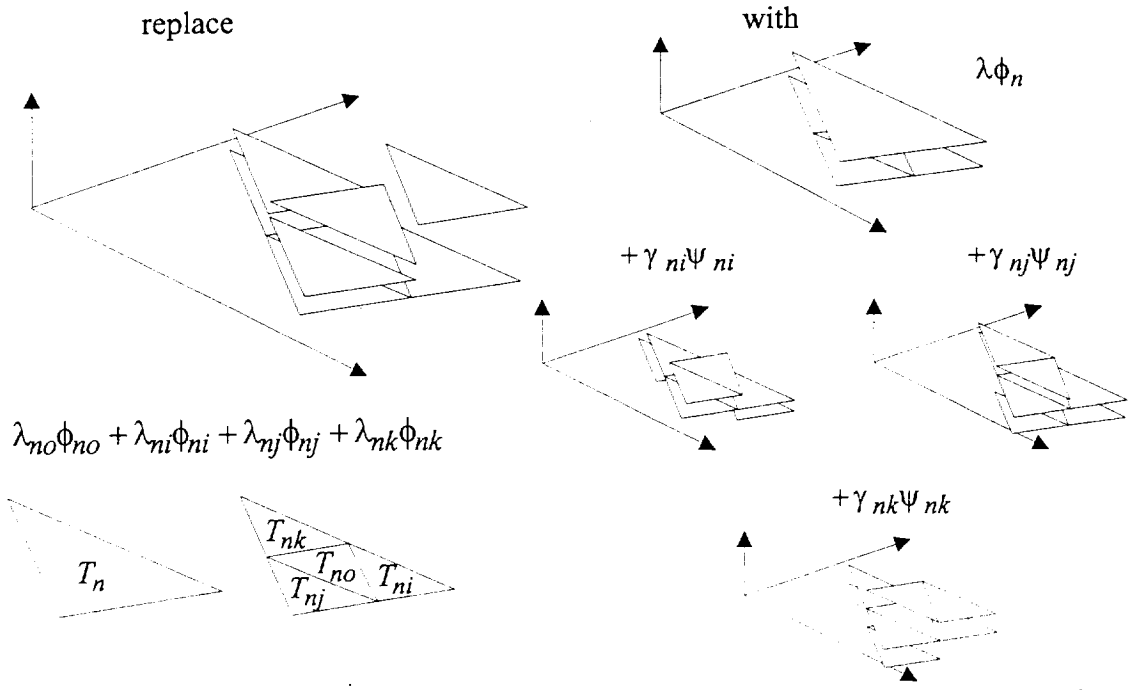
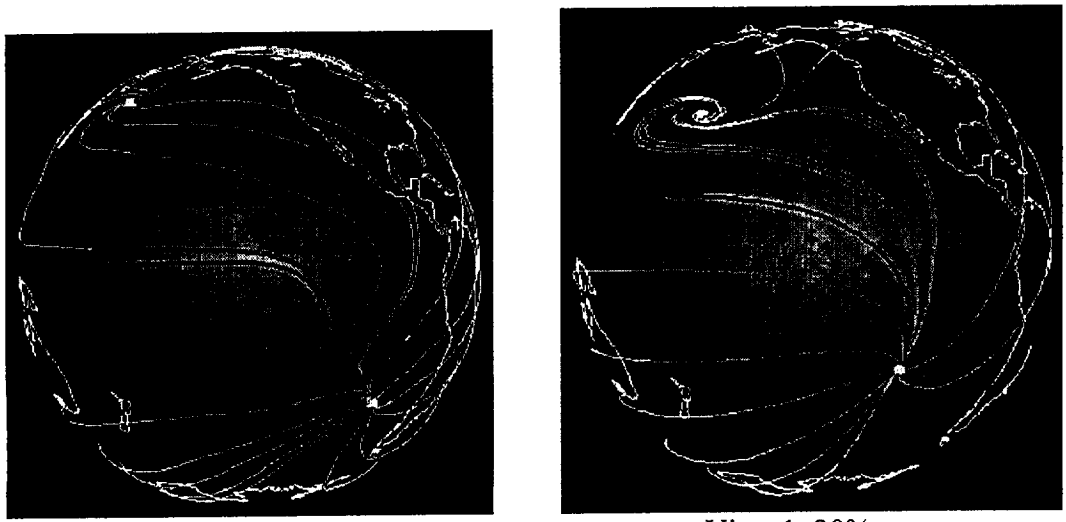


Figure 2. Basic decomposition step for Haar wavelets over nested triangular domains

Biorthogonal wavelets within this context have previously appeared. We have developed some improvements. While these new wavelets are also only biorthogonal, they have the property that when the areas do become the same, the wavelets become mutually orthogonal and this is a definite plus when it comes to ability to approximate data. A nice example of where this happens is with the standard triangular decomposition of the sphere.

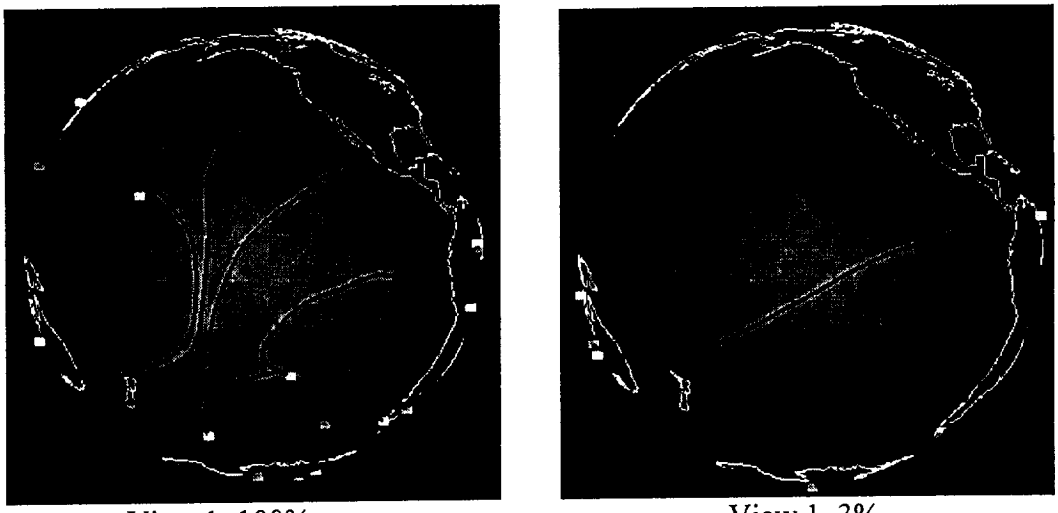
We have applied our new wavelets to data consisting of a vector field defined over the sphere. Our research on Haar wavelets for nested triangular grids and applications to fluid flow analysis over a spherical domain is reported in the submitted manuscript [4].



View 1, 1% View 1, 20%

Figure 3. Partial (nearly orthogonal) spherical wavelet reconstruction of a flow field defined over a spherical domain.

Also we have applied our new, nearly orthogonal wavelets to some "real data" provided to us by Roger Crawfis and Nelson Max of LLNL. This is simulated data resulting from a global weather model. A typical image for this work is shown in Figure 4. More details can be found in the submitted manuscript [4].



View 1, 100% View 1, 3%

Figure 4. Partial reconstruction of global weather data.

We have started our work on extending the research on wavelets defined over nested triangular grids to the case of nested tetrahedral grids. One surprise has already come up. We had thought that it would be natural to use the nested tetrahedral grids based upon the basic decompositon as shown in Figure 5. This decompostion seemed to be the 3D analog of the decompositon used for triangles shown in Figure 1. We started to work with these types of grids and then found out much to our surprise that we could not define affine invariant wavelets over these types of decompositions. We feel that the property of affine invariance is extremely important. In essence it says that the results of any analysis or compression based upon these wavelets is not affected by the way we label the triangles. Clearly, it is undesirable to have an algorithm that depends on the inner details of how it is coded. In order to maintain the property of affine invariance, we have moved to a different type of basic decomposition step. It is shown in Figure 6. Rather than 8 subtetraheda per tetrahedron, we have 17 (yes 17!) subtetrahedra per tetrahedron. Utilizing Mathemaica we have recently been able to define some affine invariant Haar wavelets based upon this type of decomposition. We are really quite exicited about these results and can't wait to implement them and see how they perform.

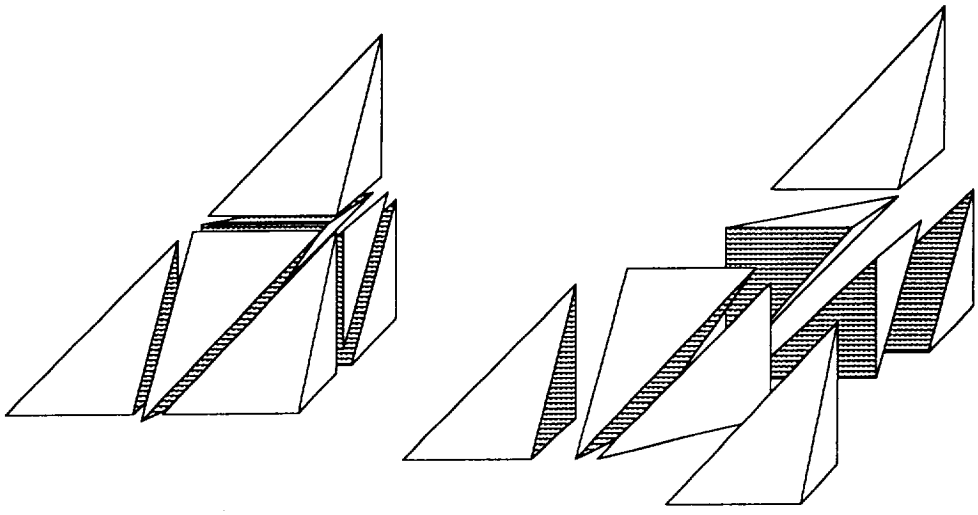


Figure 5. A standard tetrahedral subdivision which will not work for defining affine invariant wavelets.

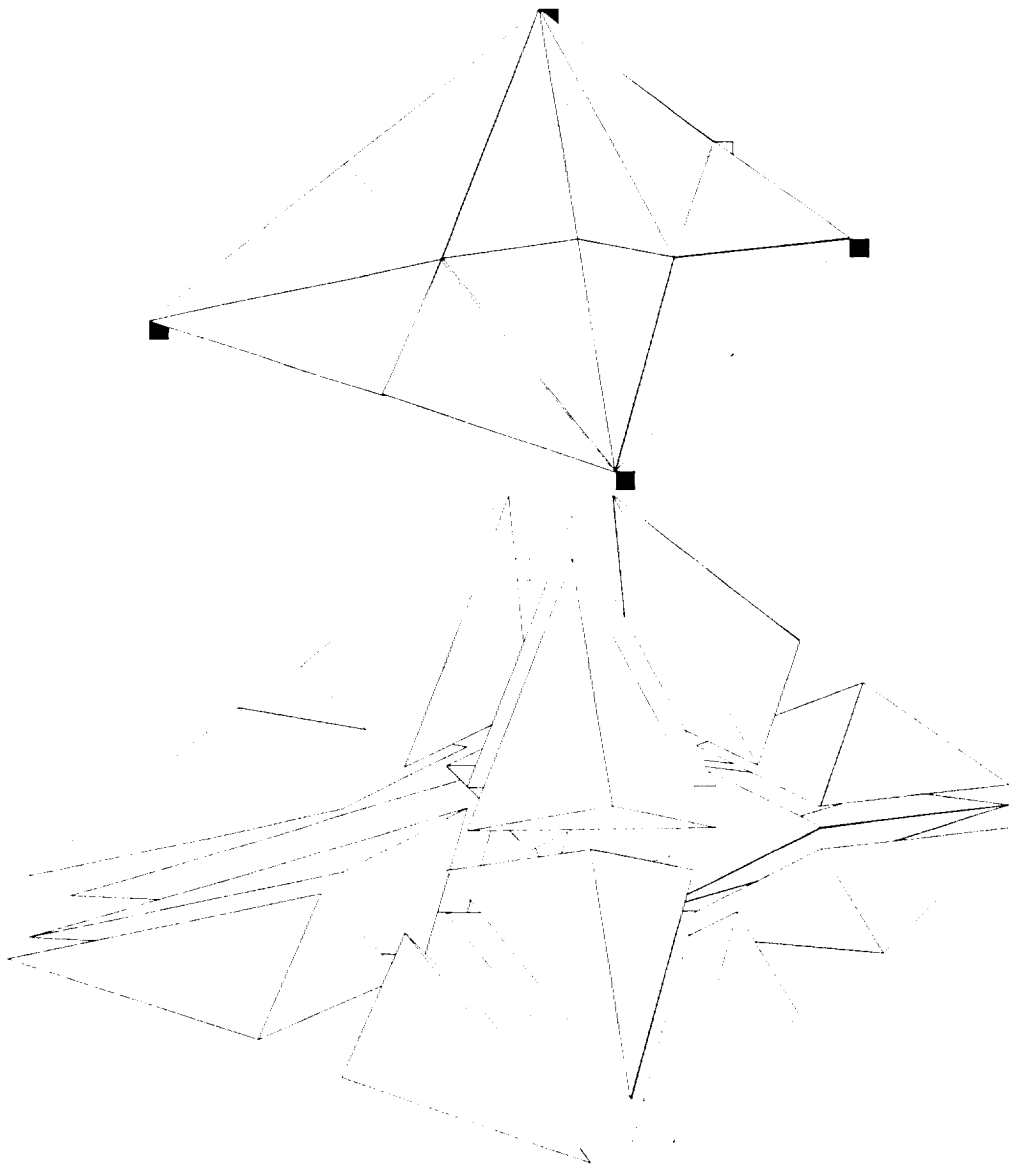


Figure 6. Decomposing a tetrahedron into 17 sub-tetrahedra. The top figure shows the edges on the faces and the bottom is an exploded view.

In order to convey some idea of our current progress on Haar wavelets defined over tetrahedrizations shown in Figure 6, we include some of our preliminary results for the basic refinement equations for these new wavelets. The free parameters below are c, a_0, a_1, a_2, b_0, b_1, b_2, and b_3. We impose the orthogonality conditions and then use Mathematica to solve for the refinement equations with the values give below.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
c	a_0	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	b_0	b_1	b_1	b_1
c	a_1	a_0	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	b_1	b_0	b_1	b_1
c	a_1	a_1	a_0	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	b_1	b_1	b_0	b_1
c	a_1	a_1	a_1	a_0	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	b_1	b_1	b_1	b_0
c	a_1	a_1	a_1	a_1	a_0	a_1	a_1	a_1	a_1	a_1	a_1	a_1	b_0	b_1	b_1	b_1
c	a_1	a_1	a_1	a_1	a_1	a_0	a_1	a_1	a_1	a_1	a_1	a_1	b_1	b_0	b_1	b_1

c	a_1	a_1	a_1	a_1	a_1	a_1	a_0	a_1	a_1	a_1	a_1	a_1	b_1	b_1	b_0	b_1
c	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_0	a_1	a_1	a_1	a_1	b_1	b_1	b_1	b_0
c	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_0	a_1	a_1	a_1	b_0	b_1	b_1	b_1
c	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_0	a_1	a_1	b_1	b_0	b_1	b_1
c	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_0	a_1	b_1	b_1	b_0	b_1
c	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_0	b_1	b_1	b_1	b_0
c	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	b_2	b_3	b_3	b_3
c	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	b_3	b_2	b_3	b_3
c	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	b_3	b_3	b_2	b_3
c	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2	b_3	b_3	b_3	b_2

Solutions:

The following constants are used in the solutions which give values for a_0, a_1, a_2, b_0, b_1, b_2, and b_3.

Constant	Symbolic Value
k_1	$\frac{-c}{16} \pm \frac{5c}{64}\sqrt{17}$
k_2	$\frac{-c}{16} \pm \frac{c}{12}\sqrt{17} \pm \frac{c}{48}\sqrt{221}$
k_3	$\frac{-c}{16} \pm \frac{13c}{192}\sqrt{17}$
k_4	$\frac{-c}{16} \pm \frac{c}{16}\sqrt{17}$

Solution 1:

a_0	-7/10c - 51/5 k_1
a_1	k_1
a_2	-21/80c - 16/5 k_1
b_0	-3/40c - 1/5 k_1
b_1	-3/40c - 1/5 k_1
b_2	211/80c + 216/5 k_1
b_3	-13/80c - 8/5 k_1

Solution 2:

a_0	-c - 15 k_4
a_1	k_4
a_2	k_4
b_0	k_4
b_1	k_4
b_2	-c - 15 k_4
b_3	k_4

Solution 3:

a_0	-21/26c -155/13 k_3
a_1	k_3
a_2	35/208c - 48/13 k_3
b_0	-5/104c + 3/13 k_3
b_1	-5/104c + 3/13 k_3
b_2	-661/208c - 648/13 k_3
b_3	11/208c + 24/13 k_3

Solution 4:

a_0	$\frac{2419c^3 + 38320c^2k_2 - 19008ck_2^2 - 101376k_2^3}{408c^2}$
a_1	$-\frac{257c^3 + 3632c^2k_2 - 1728ck_2^2 - 9216k_2^3}{408c^2}$
a_2	$\frac{60c^3 + 959c^2k_2 - 432ck_2^2 - 2034k_2^3}{17c^2}$
b_0	k_2
b_1	k_2
b_2	$-\frac{6565c^3 + 102960c^2k_2 - 46656ck_2^2 - 248832k_2^3}{136c^2}$
b_3	$\frac{223c^3 + 3632c^2k_2 - 1728ck_2^2 - 9216k_2^3}{136c^2}$

We now discuss our research in the area of multiresolution models for curvilinear grids. In the original proposal we mentioned the idea of using multiresolution models for parametric curves in this context. The idea is based upon the observation that a curvilinear grid can be viewed as a parametric map and researchers have already developed techniques for using wavelets on parametric curves. Unfortunately there is a basic problem with this approach and that lies in the fact that not only do you obtain a lower resolution approximation to the flow, but you also get a lower resolution

approximation to the boundary of the domain. This means that the domain will be "smoothed down" also. We have overcome this problem in 2D by using a knot removal technique to yield a collection of nested domains as illustrated in Figure 7. (Note that the detail of the inner boundary (not the outer, though) remains through all levels of approximation).

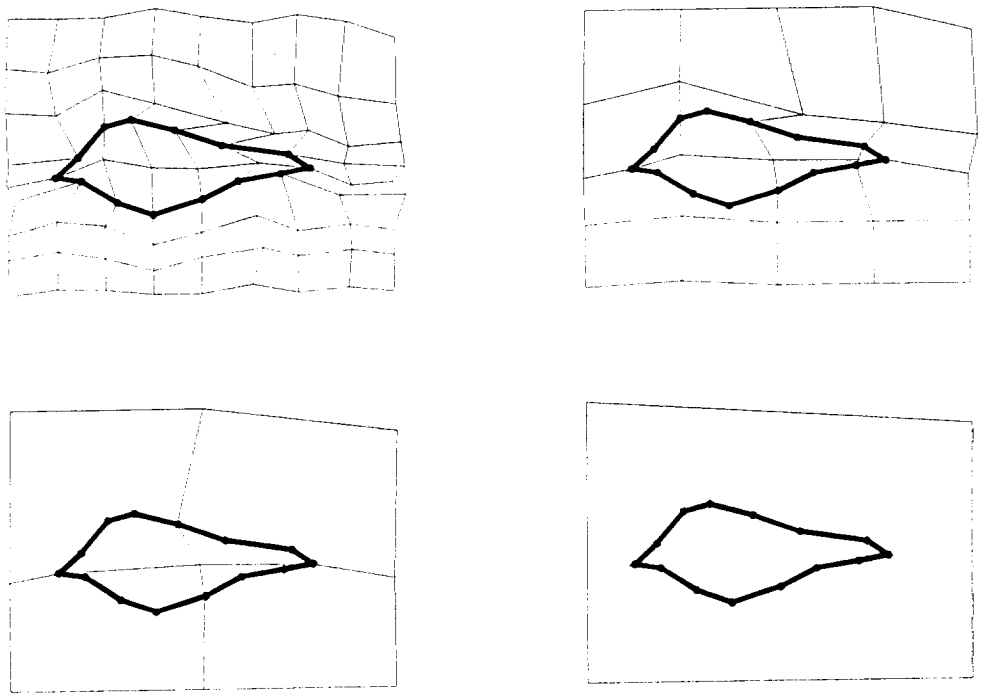


Figure 8. Decomposition by knot removal

We have developed some Haar wavelets for these types of decomposition and examples of our new algorithms are included in the submitted manuscripts [1] and [4]. We plan to write another research paper delineating more of the details of these methods, but will most likely wait to do this after our 3D work is completed. We have done most of the developmental work for the 3D case and are in the middle of doing the implementation.

As we had planned, we have extended the Haar wavelets to higher degree continuity. We have used the so called "lifting scheme." These higher order methods are more expensive to compute, but they are much more efficient. An example is shown in Figure 9.

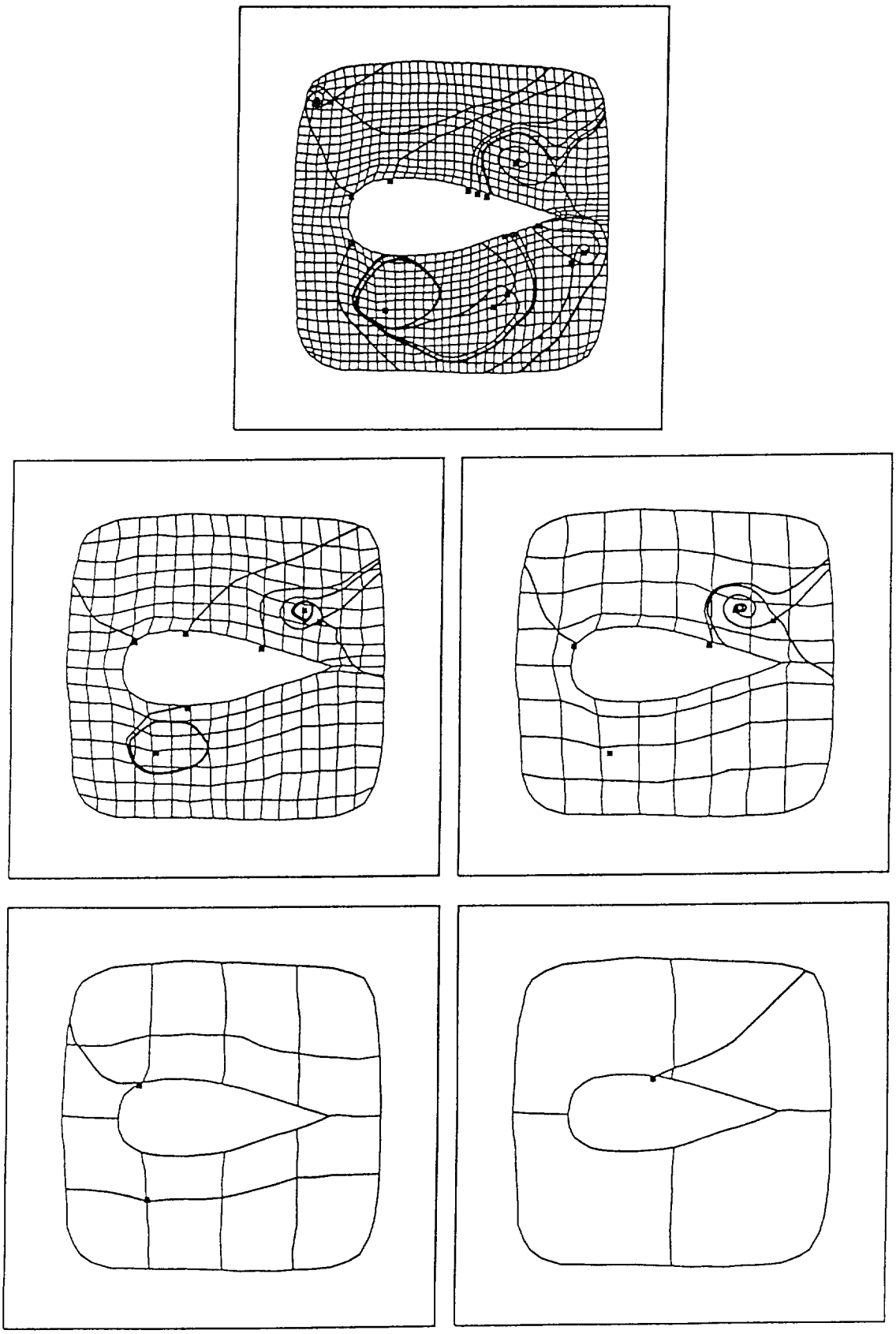


Figure 9a. Haar wavelet approximations over nested curvilinear grid.

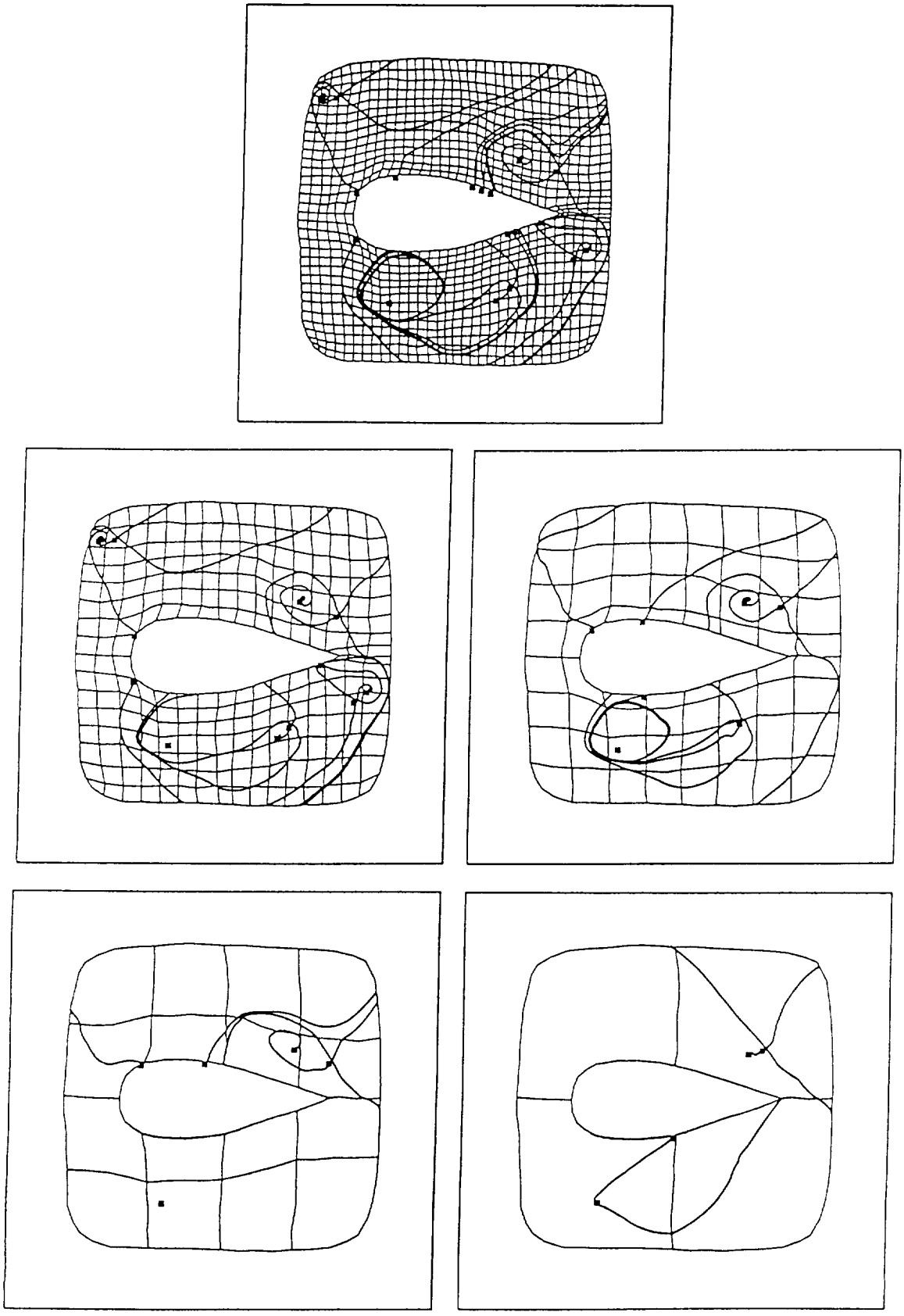


Figure 9b. Improved linear wavelet approximations over nested curvilinear grid.

Summary of 2nd Year Research

- Explicit Methods

We have developed, implemented and thoroughly tested the 2D explicit techniques based upon linear variation over triangles. We can compute a single tangent curve or compute a complete topological graph of a 2D flow over a curvilinear grid.

We have done the mathematical analysis for the extension of the explicit method to 3D (some of the equations are given earlier in this report) and we are currently developing and implementing incremental methods for 3D tetrahedra and curvilinear grids.

- Multiresolution Methods for Curvilinear Grids

We extended our piecewise constant wavelets over 2D curvilinear grids to piecewise linear wavelets. We are currently in the middle of implementing and testing these methods. Some preliminary results are shown in Figure 9 of this report.

We have developed the data structures and algorithms for extending our knot removal algorithms for curvilinear grids to 3D. This requires the tetrahedrization of general polyhedra with fixed boundary polygons. We are now in the middle of implementing these results.

- Multiresolution methods for nested triangular grids

We have used a lifting scheme to extend the 2D Haar wavelets to piecewise linear.

We have developed the basic decompositionn and refinement equations for Haar 3D wavelets over certain (see Figure 6) tetrahedral subdivisions. These new equations appear for the first time in this report.

Publications resulting from the research of this project

(All publications listed here specifically acknowledge support under this NASA grant and copies are attached)

1. G. M. Nielson, I.-H. Jung, N. Srinivasan, J. Sung and J.-B. Yoon, Tools for computing tangent curves and topological graphs for visualizing piecewise linearly varying vector fields over triangulated domains, to appear in the book, *Scientific Visualization: Overviews, Methodologies, and Techniques*, CS Press, 1997. pp. 517-558.

2. G. M. Nielson, Tools for triangulations and tetrahedrizations and construction functions defined over them, to appear in the book, Scientific Visualization: Overviews, Methodologies, and Techniques, CS Press, 1997, pp. 419-515.

3. G. M. Nielson and Junwon Sung, Interval volume tetrahedrization, submitted to Visualization '97 Conference.

4. G. M. Nielson, Il-Hong Jung and Junwon Sung, Haar wavelets over triangular domains with applications to multiresolution models for flow over a sphere, submitted to Visualization '97 Conference.

5. G. M. Nielson and Richard Franke, Computing the separating surface for segmented data, submitted to Visualization '97 Conference.

Statement of Work for Next Year

As we have previously done, we structure our discussion upon the topics of this research proposal: explicit methods, multiresolution methods for curvilinear grids, and multiresolution methods for triangular grids.

As far as explicit methods go, in the past year, we have created some algorithms for the efficient computation of 2D tangent curves based upon our ideas of explicit solutions to linearly varying vector fields. One of the types of methods we have developed are (exact) incremental methods. We have done all of the mathematical work for the extension of these methods to 3D and plan to implement and test these new 3D methods. In addition, we have been testing some 2D methods which work directly in barycentric coordinates. Barycentric coordinates allow one to discuss tangent curves (and other descriptive visualization objects) in a context that is independent of the coordinate system used for the data. This has several advantages including the ease of detecting which cell a point (on a tangent curve) lines within. We are at this moment in the middle of doing the mathematical development for the 3D situation. This is taking some effort, but it appears that everything will eventually work out. Once the equations have been obtained, we will implement the methods and compare and test them.

Points of attachment and detachment are special critical points (zero flow) on the interior boundary (car body or airplane wing for example). They are very important in the computation of topological methods. In the past, most of the literature on this topic was not so rigorous. Using the exact solutions for linearly varying vector fields, we have been able to give very precise definitions which lead to effective algorithms in the case of 2D flows. We reported on this work in [1]. We are planning to extend this work to 3D. Here, points of attachment and detachment will be replaced with edges of attachment and detachment. We are hopeful that the same rigorous characterization and resulting effective algorithms well exist in the 3D case.

Earlier, we developed some knot removal techniques which lead to a collection of nested domains for 2D curvilinear grids. These nested domains were the necessary for the multiresolution (wavelet) analysis for vector fields defined over 2D curvilinear grids. We subsequently developed, implemented and reported on this research. See [1] and [4]. We want to extend all of this to 3D. We have already done the development for the 3D nested grids and we now currently extending the Haar wavelets to this context. There have been several snags along the way, but we feel that we can overcome this obstacles and obtain the results which are comparable to our 2D work. We will implement these new 3D Haar wavelets and test them on real data. Just as we were able to use "lifting" techniques to extend the Haar wavelets to linear wavelets for the nested 2D curvilinear grids, we hope to do the same for 3D grids.

Previously, we developed some new Haar wavelets for nested triangular domains as shown in Figure 1. We have applied these new techniques to a variety of data sets including slices of 3D flow around and automobile and weather data representing flow over the earth. We have reported on these results in the submitted manuscripts [1] and [4]. We are currently involved in testing the extension of these methods to higher order, linear methods. For the next year, we want to extend all of these results to 3D. We were

originally frustrated with the basic mathematics when we first attempted to use the decomposition of Figure 5. This did not work out, but we are currently having success with the decomposition shown in Figure 6. Some of the early mathematical results are mentioned above. We will also use the basic building blocks of these techniques to develop some methods for the adaptive volume modeling of 3D volume data.

Summary of Proposal for Research in 3rd Year

- Explicit Methods

Implement and test algorithms for tetrahedra based incremental methods.

Continue development and implementation of 3D methods utilizing Barycentric coordinates.

Develop the proper definitions and characterizations of the "edges of attachment" and "edges of detachment" for 3D flows near an inner boundary.

- Multiresolution Methods for Curvilinear Grids

We have previously developed the 3D knot removal techniques for generating nested domains for 3D curvilinear grids. We are also now doing the development of the 3D Haar wavelet in this context. We plan to complete this development and implement and test these algorithms.

We propose to extend Haar wavelets for 3D curvilinear grids to piecewise linear wavelets for 3D curvilinear grids.

- Multiresolution methods for nested tetrahedral grids

Using the subdivision as shown in Figure, we plan to further develop, implement and test Haar wavelets for tetrahedral grids. We plan to extend these techniques for higher order (linear) wavelets which should have more efficient compression and approximation capabilities.

We plan to use the basic multiresolution techniques to develop some adaptive methods for modeling volume data. This would allow details to be added and removed as the user's attention is focused in different regions of the data.

Budget

We are requesting funds for the principal investigator as follows:

Gregory M. Nielson
2 summer months
(see budget sheet for amounts)

We are requesting funds for two students:

Two ASU Graduate Students
9 months half time, 3 months full time
(see budget sheet for amounts)

We are requesting funds in the operations category as explained on the budget sheet.

ESTIMATED BUDGET

PERIOD OF PERFORMANCE: January 1,1995 through December 31, 1997

							Year 1	Year 2	Year 3	Summary
I.	SALARIES & WAGES									
	A. PI: Nielson		0.00	pm/AY +	2.00	pm/Sum	$16,610	$17,607	$18,663	$52,880
	B. Co-PI:		0.00	pm/AY +	0.00	pm/Sum	$0	$0	$0	$0
	C. Co-PI:		0.00	pm/AY +	0.00	pm/Sum	$0	$0	$0	$0
	D. Co-PI:		0.00	pm/AY +	0.00	pm/Sum	$0	$0	$0	$0
	E. Postdocs	0 @	0.00	pm/CY			$0	$0	$0	$0
	F. Technicians	0 @	0.00	pm/CY			$0	$0	$0	$0
	G. Graduate Associates	2 @	9.00	pm/AY +	6.00	pm/Sum	$35,020	$36,050	$37,767	$108,837
	H. Graduate Assistants	0 @	0.00	pm/AY +	0.00	pm/Sum	$0	$0	$0	$0
	I. Undergraduates	0 @	0.00	pm/AY +	0.00	pm/Sum	$0	$0	$0	$0
	J. Secretary	0 @	0.00	pm/CY			$0	$0	$0	$0
	TOTAL						$51,630	$53,657	$56,430	$161,717
II.	FRINGE BENEFITS									
	A. Faculty at		25%				$4,153	$4,402	$4,666	$13,221
	B. Staff at		30%				$0	$0	$0	$0
	C. Students at		3%				$1,051	$1,082	$1,133	$3,266
	TOTAL						$5,204	$5,484	$5,799	$16,487
III.	TRAVEL						$0	$0	$0	$0
IV.	OPERATIONS									
	A. Consultants						$0	$0	$0	$0
	B. Publication/Page Charges						$0	$0	$0	$0
	C. Photocopy, telephone, postage, misc.						$500	$500	$500	$1,500
	D. Material & Lab Supplies/Equipment Use Fees						$1,000	$1,000	$1,000	$3,000
	E. Computer Software						$200	$0	$0	$200
	F. Lab Equipment under $500						$0	$0	$0	$0
	G. Subcontracts						$0	$0	$0	$0
	H. Participant Support Costs						$0	$0	$0	$0
	I. Tuition	0 @	$1,895	per AY	(4% escalation)		$0	$0	$0	$0
	J. Student Stipends						$0	$0	$0	$0
	K. Other						$0	$0	$0	$0
	TOTAL						$1,700	$1,500	$1,500	$4,700
V.	CAPITAL EQUIPMENT						$0	$0	$0	$0
VI.	TOTAL DIRECT COSTS						$58,534	$60,641	$63,729	$182,904
VII.	INDIRECT COSTS									$95,733
				Year 1	52.0%	MTDC	$30,438			
				Year 2	52.5%	MTDC		$31,837		
				Year 3	52.5%	MTDC			$33,458	
VIII.	TOTAL PROJECT COSTS						$88,972	$92,478	$97,187	$278,637

Chapter 21

Tools for Computing Tangent Curves and Topological Graphs for Visualizing Piecewise Linearly Varying Vector Fields over Triangulated Domains

Gregory M. Nielson, Il-Hong Jung, Nat Srinivasan, Junwon Sung, and Jong-Beum Yoon

Abstract. *We describe some methods for computing tangent curves and topological graphs for a vector field defined over a two-dimensional domain. We assume that the vector field is piecewise linearly defined over a triangulation of the domain. Piecewise explicit representations in terms of elementary transcendental functions form the basis of algorithms for determining and displaying tangent curves. Topological methods which link critical values with separating tangent curves are developed and discussed.*

21.1 Introduction

Streamlines are a well-established visualization technique for investigating a vector field. In this chapter we describe some new techniques for computing these invariant tangent curves. Conventional methods for computing tangent curves consist of using numerical methods for solving vector-valued initial valued problems. Euler's method is the simplest and Runge-Kutta-type methods are often used in practice (see [3,24,39,41,49]). The issues involved in selecting a particular algorithm usually focus on the two, often opposing, requirements of accuracy and speed. Accuracy is especially important in the computation of

tangent curves for topological methods. Erroneous results can easily occur unless special provisions are taken to control errors. Speed is particularly important for interactive methods, but accuracy should not be discounted too much in this context since the goal of the visualization is to impart meaningful and valid information. The methods described in this chapter allow for a reasonable balance to be achieved between accuracy and speed.

A streamline (or tangent) curve is a parametric curve

$$P(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

that is everywhere tangent to the vector field. If

$$V(x,y) = \begin{pmatrix} u(x,y) \\ v(x,y) \end{pmatrix}$$

represents a static vector field, then P is characterized by the equations

$$P'(t) = V(P(t)) = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \end{pmatrix} = \begin{pmatrix} u(x(t),y(t)) \\ v(x(t),y(t)) \end{pmatrix}. \tag{21.1}$$

Typically there is an entire family of solutions for Equation (21.1) and a particular solution is selected with the initial condition

$$P(0) = \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = P_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}.$$

In this chapter, we discuss methods for the special case where $V(x,y)$ is a piecewise linear function. The domain is decomposed into a collection of triangles and V has the form

$$V(x,y) = \begin{pmatrix} u(x,y) \\ v(x,y) \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y + b_1 \\ a_{21}x + a_{22}y + b_2 \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix} + B \tag{21.2}$$

over each triangle. With this assumption, the tangent curve becomes a piecewise concatenation of curve segments with each segment associated with a particular triangle. The entry point on a particular triangle provides the initial conditions for the constant coefficient ODE which characterizes the curve segment for each triangle. The exit point serves as the entry point for the next adjoining triangle domain. This basic idea is further illustrated in Figure 21.1. In this way, it is possible to completely characterize and know a tangent curve by a sequence of entry (exit) points.

There are a variety of sources for the type of data covered here. If a 2D flow (or a 3D flow assumed to be constant in one direction) is measured at a collection of scattered, planar points, then the domain points can serve as the vertices for a triangulation of the domain. The Delaunay triangulation of the convex hull would be a possibility. See Nielson [33]. We will show examples later of a flow over a spherical domain which represents wind speed and direction over the Earth. If the data is measured at scattered locations, these points can serve as the vertices of a triangulation of the sphere and the methods of this chapter can be applied. If the data is modeled, then the model can be evaluated on a triangular grid and again the methods of this chapter will apply. Curvilinear grids which normally associate

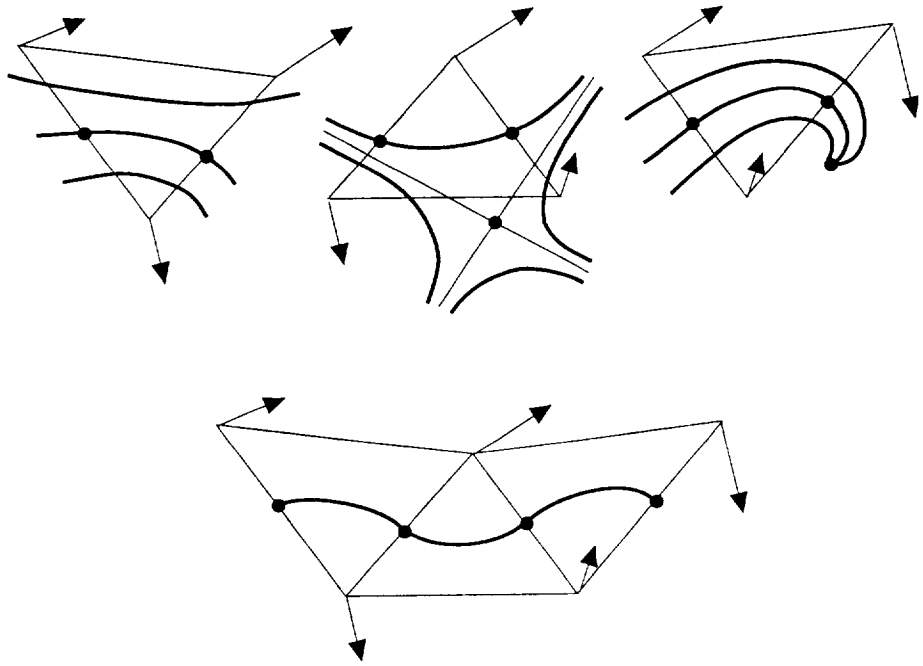


Figure 21.1: A composite tangent curve running across three triangles.

with simulated data can also be triangulated by simply inserting one diagonal or the other into the quadrilateral cells. See Figure 21.2.

The values A and B of Equation (21.2) for a particular triangle are determined from the values of the vector field at the vertices of the triangle. If the vertices of the triangle are labeled as in Figure 21.3, then the equations

$$\begin{aligned} a_{11}x_i + a_{12}y_i + b_1 &= u(x_i, y_i) \\ a_{11}x_j + a_{12}y_j + b_1 &= u(x_j, y_j) \\ a_{11}x_k + a_{12}y_k + b_1 &= u(x_k, y_k) \\ \\ a_{21}x_i + a_{22}y_i + b_2 &= v(x_i, y_i) \\ a_{21}x_j + a_{22}y_j + b_2 &= v(x_j, y_j) \\ a_{21}x_k + a_{22}y_k + b_2 &= v(x_k, y_k) \end{aligned} \tag{21.3}$$

will yield the values of A and B. It is also possible to use barycentric coordinates to find these values. The details of this will be discussed later in Section 21.4.

The remainder of the chapter is organized as follows. In Section 21.2, we discuss explicit methods which are based upon the fact that the differential equation which characterizes a tangent curve is first order with constant coefficients and so an explicit solution in terms of common transcendental functions is possible. Section 21.3 is concerned with incremental methods which produce a sequence of points on the curve (or approximations)

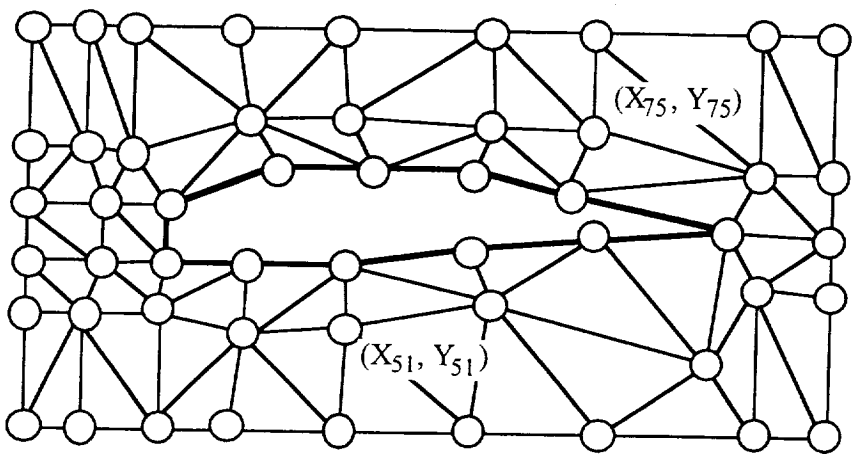


Figure 21.2: Triangulated curvilinear grid.

with the computation of each subsequent point based upon the previous point. Section 21.4 covers much of the same material of Sections 21.2 and 21.3, but within the context of barycentric coordinates rather than the conventional Cartesian coordinates. Topological methods are covered in Section 21.5.

21.2 Explicit Methods

In this section we describe what we call "explicit" methods. Because of the special nature of the ODE which characterizes a tangent curve, it is possible to obtain explicitly defined solutions and therefore, there is no need to use numerical methods to compute the values of these curves. Library routines for computing common transcendental functions can be used instead. Computing the exit point of a particular curve segment, which will serve as the initial point for the next curve segment, amounts to computing the intersection of a curve and a line segment. This is equivalent to a univariate root computation problem. We discuss two general approaches to this intersection point calculation problem. Each approach is covered in a subsequent subsection. In Section 21.2.1 we discuss the approach which is based upon a parametric representation of the tangent curve and an implicit representation of a line containing one of the edges of the triangle. The parametric curve is substituted into the implicit line equation yielding a single equation in the parameter of the tangent curve. The root of this equation yields the parameter value of the intersection point and this point is subsequently tested to determine if it is actually on the edge (a subset of the line). In Section 21.2.2, the second general approach is covered. It is based upon an implicit representation of the tangent curve and a parametric representation of the edge of the triangle. In this approach, the parametric representation of the edge is substituted into the implicit representation of the tangent curve, yielding again a single, univariate equation. The root is computed and used to evaluate the parametric representation of the edge so as to obtain the intersection point.

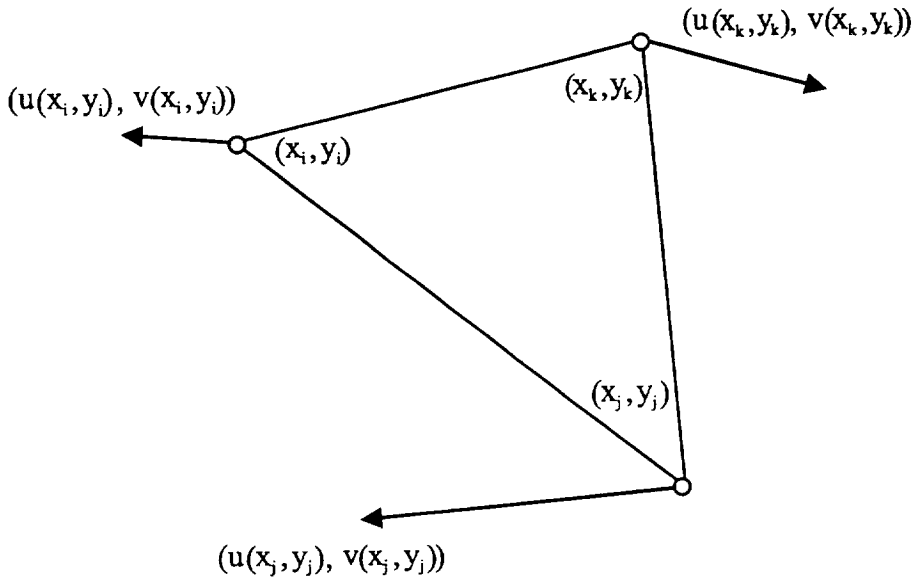


Figure 21.3: Notation and conventions used for data.

21.2.1 Parametric Tangent Curve, Implicit Edge

In this section, we discuss the general properties of a tangent curve for a linearly varying vector field over a single triangular domain. In particular we give the details of a parametric representation,

$$P(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}.$$

(Implicit representations are covered in the next section, Section 21.2.2.) The curve enters the triangle at a point $P(0) = P_0$ and then either attaches to a critical point in the triangle or exits from the triangle at a point $P(t_e) = P_e$. We assume that the entry point is given and so we need to compute, t_e, P_e, and the various coefficients of a parametric representation of the curve from P_0 to P_e. The value t_e along with the parametric representation are used for displaying the curve and the value P_e serves as the entry point for the next triangle. If P_e lies on one of the edges, then t_e must satisfy one of the equations

$$f_a(x(t_e), y(t_e)) = 0, \qquad a = i, j, k \tag{21.4}$$

where

$$\begin{aligned} f_i(x,y) &= (x - x_j)(y_k - y_j) - (y - y_j)(x_k - x_j) \\ f_j(x,y) &= (x - x_k)(y_i - y_k) - (y - y_k)(x_i - x_k) \\ f_k(x,y) &= (x - x_i)(y_j - y_i) - (y - y_i)(x_j - x_i). \end{aligned}$$

But not all of the roots of $f_a(x(t), y(t))$, $a = i, j, k$ are candidates to be t_e. It must be the case that t_e is the smallest of these roots and that P_e is actually on the appropriate edge

and not simply on the line containing this edge. The point P_e could potentially be on any of the edges, including the edge containing P_0, but some simple tests and modifications can possibly eliminate some edges and portions of edges where P_e might be found. Along an edge, the flow is either always in, always out, or there is a special point where the flow is parallel to the edge and the direction of flow relative to the triangle changes at this point. This is further illustrated in Figure 21.4. The change of direction point P_Δ is easy to compute. For example, if the direction changes on edge e_k, then

$$P_\Delta = tP_i + (1-t)P_j \tag{21.5}$$

where

$$tV_i + (1-t)V_j = c(P_j - P_i)$$

for some constant, c, and $0 < t < 1$.

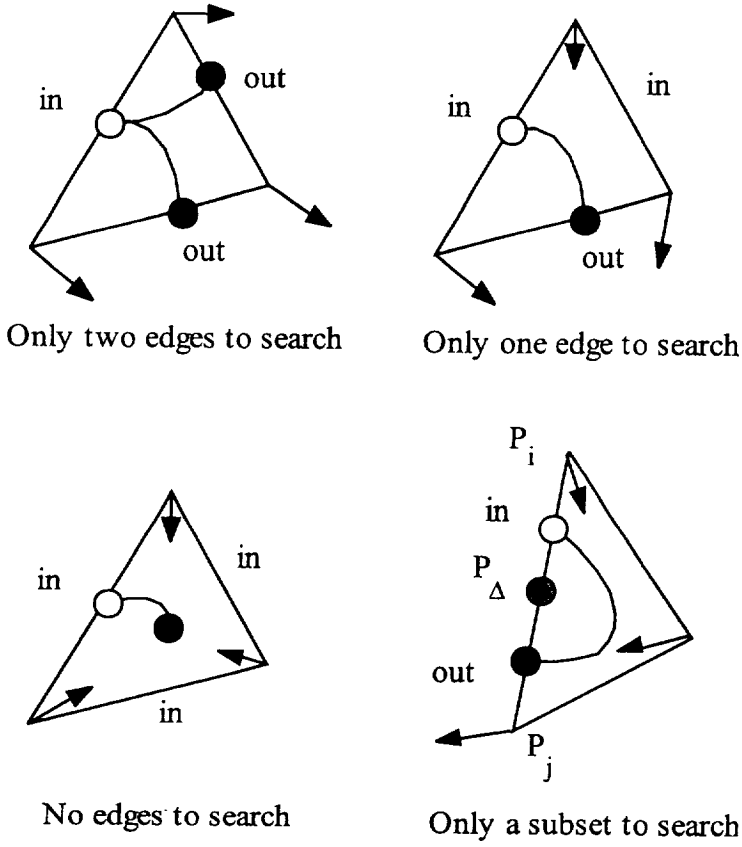


Figure 21.4: Exit point search strategy regions.

We now turn our attention to the details of the parametric representation of the tangent curve. The tangent curve

$$P(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

over a particular triangular cell is defined as

$$\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \end{pmatrix} = A \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} + B \tag{21.6}$$

with the initial conditions

$$\begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}.$$

The 2×2 matrix A and the 2×1 vector B are determined as in Equation (21.3) or equivalent other means. (For example, see Section 21.4 on barycentric coordinates.) The general solution of this initial value problem is of the form

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = P(t) = \Phi_1(t)E_1 + \Phi_2(t)E_2 + C \tag{21.7}$$

where the particular functions Φ_1, Φ_2, and the coefficients E_1, E_2, and C depend on the eigenvalues of A. There are five separate cases:

Case 1. A has two real, nonzero eigenvalues, $0 \neq r_1 \neq r_2 \neq 0$.

$$P(t) = e^{tr_1}E_1 + e^{tr_2}E_2 + P_c, \tag{21.8}$$

$$E_1 = \left(\frac{A - r_2 I}{r_1 - r_2}\right)(P_0 - P_c), \qquad E_2 = \left(\frac{A - r_1 I}{r_2 - r_1}\right)(P_0 - P_c), \qquad AP_c + B = 0.$$

Case 2. A has one zero and one nonzero eigenvalue, $0 = r_1, r_2 \neq 0$.

$$P(t) = tE_1 + \left(e^{tr_2} - 1\right)E_2 + P_0, \tag{21.9}$$

$$E_1 = \left(I - \frac{A}{r_2}\right)B, \qquad E_2 = \frac{A}{r_2}\left(P_0 + \frac{B}{r_2}\right).$$

Case 3. A has only a zero eigenvalue, $r_1 = 0 = r_2$.

$$P(t) = tE_1 + t^2E_2 + P_0, \tag{21.10}$$

$$E_1 = AP_0 + B, \qquad E_2 = \frac{AB}{2}.$$

Case 4. A has a single real, nonzero eigenvalue, $r_1 = r_2 \neq 0$.

$$P(t) = e^{tr_1}E_1 + te^{tr_1}E_2 + P_c, \tag{21.11}$$

$$E_1 = P_0 - P_c, \qquad E_2 = (A - r_1 I)(P_0 - P_c), \qquad AP_c + B = 0.$$

Case 5. A has complex eigenvalues, $\mu + \lambda i, \mu - \lambda i, \lambda \neq 0$.

$$P(t) = cos(\lambda t)e^{\mu t}E_1 + sin(\lambda t)e^{\mu t}E_2 + P_c \tag{21.12}$$

$$E_1 = P_0 - P_c, \qquad E_2 = \left(\frac{A - \mu I}{\lambda}\right)(P_0 - P_c), \qquad AP_c + B = 0.$$

In Cases 1, 4, and 5, A is nonsingular and it is guaranteed that there is a unique critical value satisfying $AP_c + B = 0$. In Cases 2 and 3, it is possible that no critical values exist or even that an entire line of critical values exist. Not all cases occur with equal frequency. Cases 1 and 5 are the predominate ones, Cases 2 and 4 are much less likely to occur, and Case 3 is extremely rare. This is explained by taking a look at Figure 21.5. In this example, we determined the case for a triangle where we varied the value of the vector field at one vertex of the triangle. The flow at two of the vertices is fixed and illustrated by the arrows drawn at these vertices. The flow at the vertex marked with the white circle is taken be a vector which is based at the vertex and emanating out to an arbitrary point in the plane. We classify this point on the basis of the case classification associated with this flow. The interior to the parabolic bounded region is Case 5. The boundary is Case 4 except for the degenerate subcase of Case 3 indicated by the black box on the boundary. Case 2 consists of a line tangent at this point. Of course the tolerances used for determining these regions affect their relative occurrence in actual practice. It is interesting and instructive to look at some examples of tangent curves for the various cases. We have included samples for each of the cases in Figure 21.6. The data for these examples is the same as in Figure 21.5. The boxes of Figure 21.5 indicate the tip of the one vector of the triangle for the particular example of Figure 21.6. In each of these images, the triangular domain is shown along with arrows at each vertex indicating the flow at these points. Particular tangent curves are determined by using initial values along a particular edge. We use 10 equally spaced points along these edges. The flow is shown at each of these initial points and the tangent curve is traversed out in both negative and positive parameter domain for a fixed amount. Some additional examples are shown in Figure 21.7. For these examples, the flow data at each vertex is indicated by the line segment and again we use initial values at 10 equally spaced values along an edge and the tangent curves are traversed out in both positive and negative parameter directions.

Displaying the tangent curve and computing the roots for the determination of t_e depends upon the evaluation of $P(t)$ given in its various forms by the five cases above. Depending upon the computing resources available, it is possible to structure efficient methods for performing these computations. In some cases, it is desirable to compute $P(i\Delta t)$, $i = 0, 1, 2, \ldots$, where Δt is some fixed increment of the parameter t. This may be the case in a situation where these values are used to display the curve and the curve is determined

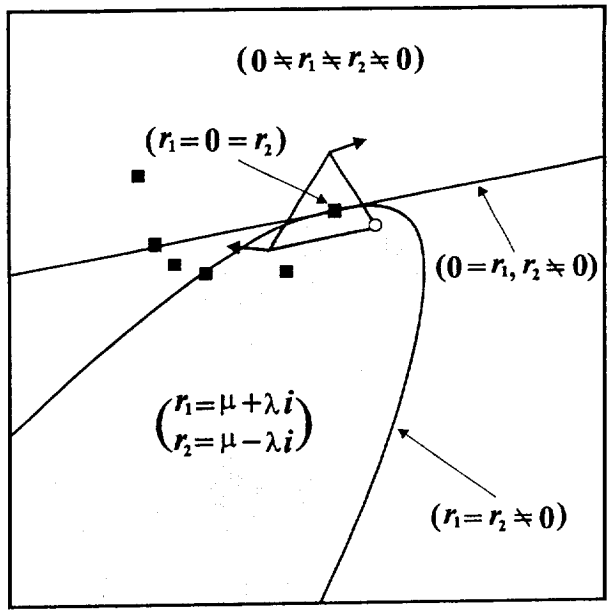


Figure 21.5: The five case regions.

to exit the triangle once one of the functions f_i, f_j, or f_k (of Equation (21.4)) is found to change sign from $a\Delta t$ to $(a+1)\Delta t$. The exponentials of Cases 1, 2, 4, and 5 can be computed by the computation of a single $\exp(\Delta t)$ and subsequent multiplications. The sin and cos functions of Case 5 can be computed with the formulas

$$sin(t + \Delta t) = cos(t)sin(\Delta t) + sin(t)cos(\Delta t)$$

$$cos(t + \Delta t) = cos(t)cos(\Delta t) - sin(t)sin(\Delta t).$$

We should also point out that the formulas which form the basis of the incremental methods of Section 21.3.2 can also be used to evaluate the curve at equally spaced parameter values for display purposes.

21.2.2 Parametric Edge, Implicit Tangent Curve

As we have previously mentioned, there are two general approaches to computing the exit point $P(t_e)$. A parametric representation for the tangent curve can be substituted into an implicit representation for the line segment as in Section 21.2.1. Or a parametric (or explicit) representation of the line segment can be substituted into an implicit representation of the curve. In this section, we discuss the latter approach. In this approach, either an explicit, $y = ax + b$ or $x = cy + d$, or a parametric $(x(s), y(s)) = sP_i + (1-s)P_j$, is used to represent the line segment and an implicit representation

$$F(x(t), y(t)) = 0$$

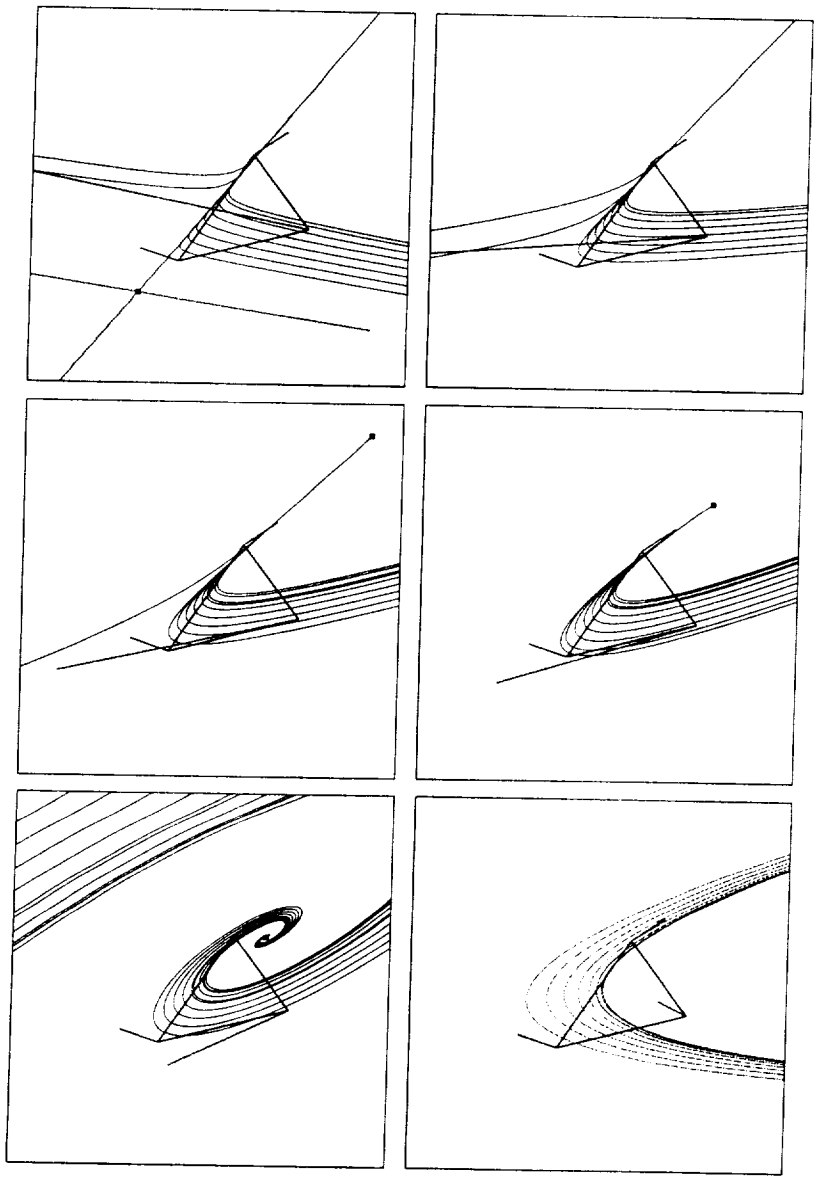


Figure 21.6: Examples of tangent curves for all five cases.

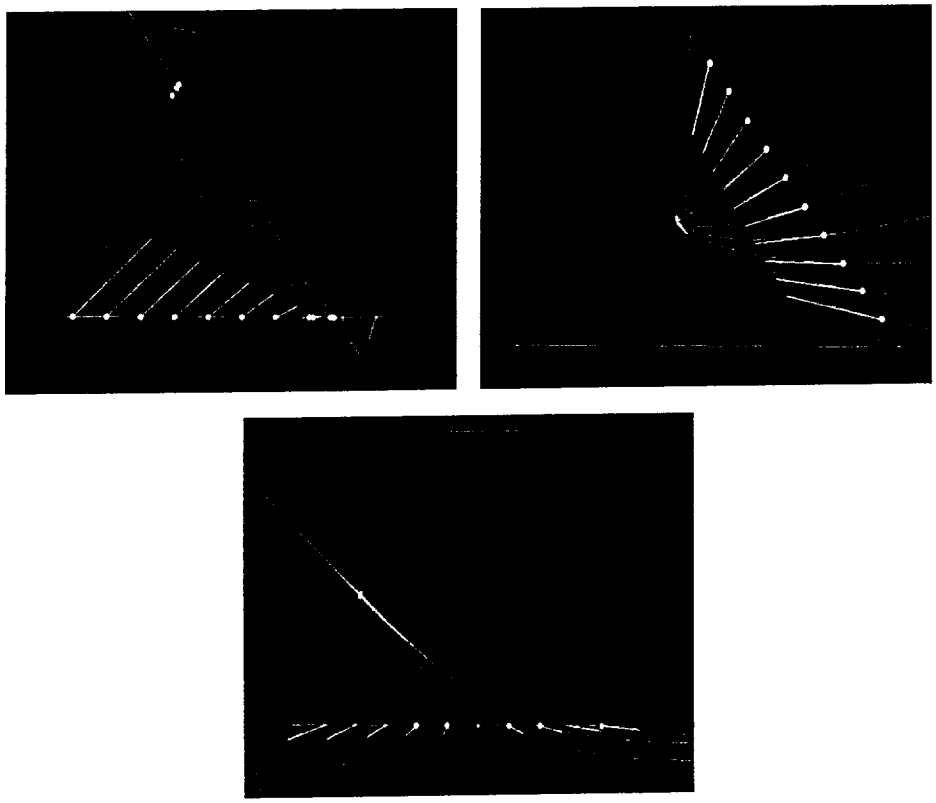


Figure 21.7: Additional examples of tangent curves. *(Upper-left)* Case 1 with $r_1 \cdot r_2 < 0$. *(Upper-right)* Case 1 with $r_1 \cdot r_2 > 0$. *(Lower)* Case 2.

is required for the curve. For the edge e_k, this leads to one of the following root finding problems:

i) $F(x, ax+b)$, $\quad min\{x_i, x_j\} \leq x \leq max\{x_i, x_j\}$

ii) $F(cy+d, y)$, $\quad min\{y_i, y_j\} \leq y \leq max\{y_i, y_j\}$

iii) $F(x(s), y(s))$, $\quad 0 \leq s \leq 1$.

Up to now, the missing component for this approach is an implicit representation of the streamline curve. We now take up this topic. Again, there are five separate cases. In each of the five cases covered below, there are specified two vectors, E_1 and E_2. We let $E = (E_1, E_2)$. If $|E| \neq 0$ then the two vectors E_1 and E_2 are linearly independent, and we can use the change of variables

$$\begin{pmatrix} x \\ y \end{pmatrix} = E \begin{pmatrix} X \\ Y \end{pmatrix} + C. \tag{21.13}$$

The point C is either P_0 for Cases 2 and 3 or P_c for Cases 1, 4, and 5. We now give the details for each separate case.

Case 1. $(0 \neq r_1 \neq r_2 \neq 0)$

If $|E| \neq 0$ then the change of variables of Equation (21.13) with $C = P_c$ changes the parametric curve to

$$X(t) = e^{r_1 t}$$
$$Y(t) = e^{r_2 t}$$

which leads to the implicit equation

$$F(X, Y) = X^{r_2} - Y^{r_1} = 0 \tag{21.14}$$

with $X, Y \geq 0$ and $t = \frac{ln(X)}{r_1} = \frac{ln(Y)}{r_2}$.

If $|E| = 0$ then P_0 is on one of the lines passing through P_c in the direction of the eigenvectors of A, and either $E_1 = 0$ or $E_2 = 0$ (or $E_1 = E_2 = 0$ if $P_0 = P_c$). The tangent curve will be a straight line. If $E_2 = 0$ then an implicit equation for the tangent curve is

$$F(x, y) = (x - x_c)e_{21} - (y - y_c)e_{11} = 0$$

and in the case that $E_1 = 0$, we have

$$F(x, y) = (x - x_c)e_{22} - (y - y_c)e_{12} = 0$$

Note that these last two implicit equations are in terms of the original variables, (x, y). See Figure 21.8.

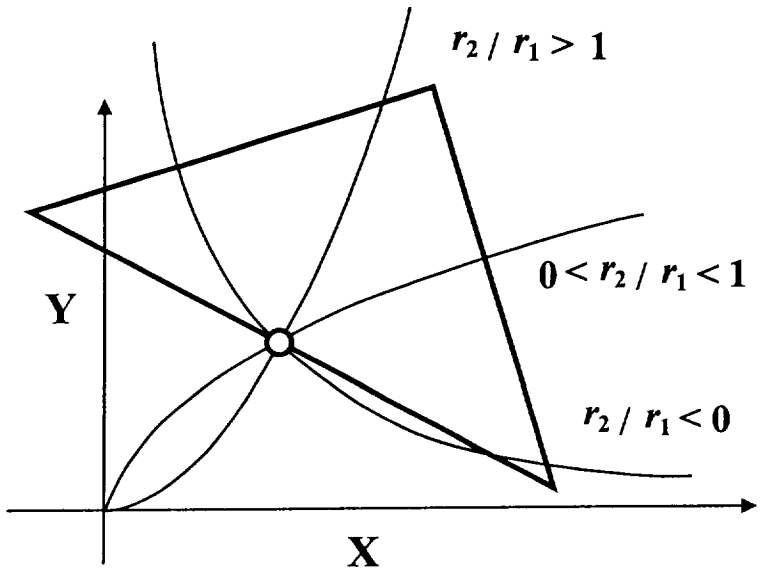


Figure 21.8: Typical curves in the transformed variables for Case 1.

Case 2. $(0 = r_1, r_2 \neq 0)$

If $|E| \neq 0$, then the change of variables given by Equation (21.13) with $C = P_0$ will yield

$$X(t) = t$$

$$Y(t) = e^{tr_2} - 1$$

and

$$F(X,Y) = Y - e^{r_2 X} + 1 = 0 \tag{21.15}$$

with $X \geq 0, \quad Y \geq -1, \quad t = X.$

If $|E| = 0$, then either $E_2 = 0$ and (in the original variables) we have the implicit equation

$$(x - x_0)e_{21} - (y - y_0)e_{11} = 0,$$

or $E_1 = 0$ and we have the equation

$$(x - x_0)e_{22} - (y - y_0)e_{12} = 0.$$

See Figure 21.9.

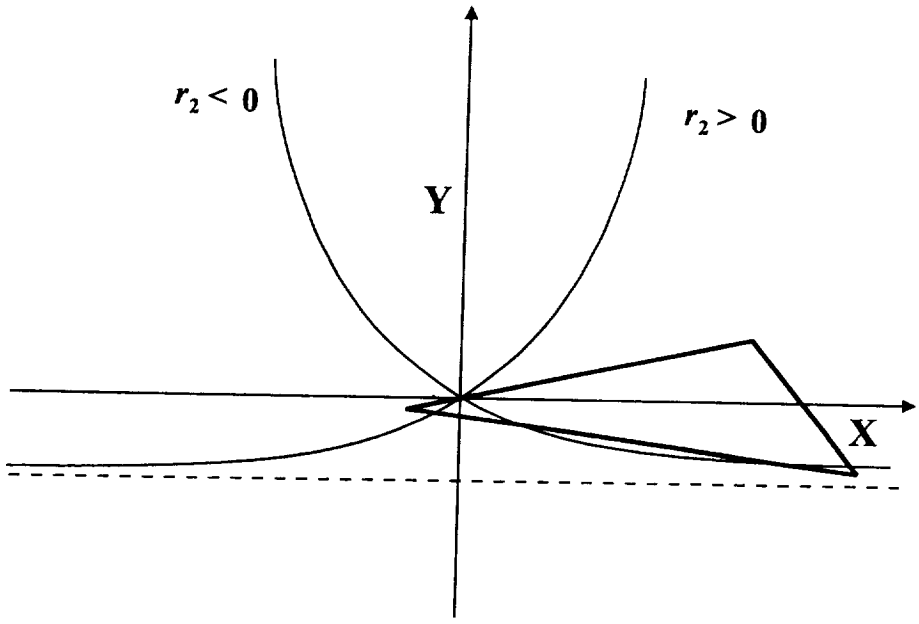


Figure 21.9: Typical curve in the transformed variable for Case 2.

Case 3. $(r_1 = 0 = r_2)$

If $|E| \neq 0$, then Equation (21.13) with $C = P_0$, can be used to obtain

$$Y - X^2 = 0, \qquad X \geq 0, \qquad Y \geq 0; \qquad t = X = \sqrt{Y}. \tag{21.16}$$

If $|E| = 0$, then $E_2 = 0$ and (in the original variables) we have the implicit equation

$$(x - x_0)e_{21} - (y - y_0)e_{11} = 0.$$

Case 4. $(r_1 = r_2 \neq 0)$

If $|E| \neq 0$, then the change of variables of Equation (21.13) leads to

$$Xln(X) - rY = 0, \qquad X, Y \geq 0, \qquad t = \frac{Y}{X}. \tag{21.17}$$

If $|E| = 0$, then the implicit equation (in the original coordinates) is

$$(x - x_c)(y_0 - y_c) - (y - y_c)(x_0 - x_c) = 0.$$

Case 5. $(\mu + \lambda i, \mu - \lambda i, \lambda \neq 0)$

The change of variable given by Equation (21.13) leads to

$$\frac{Y}{X} = tan\left(\frac{\lambda}{2\mu} ln(X^2 + Y^2)\right), \qquad t = \frac{ln(X^2 + Y^2)}{\mu}. \tag{21.18}$$

21.3 Incremental Methods

The basic idea behind incremental methods consists of starting at the entry point $P(0) = P_0$ and then successively computing $P(t_i)$, $i = 1, 2, 3, \ldots$ for increasing values of the parameter t and stopping when $P(t_i)$ leaves the triangle. These successive values that are computed along the way to finding the exit point can also be used as the basis for displaying the curve. Often it is efficient and convenient to use equally spaced values of the parameter; that is, $t_i = i\Delta t, i = 1, 2, 3, \ldots$. At least this can be the default strategy and the value of Δt can be adjusted every so often, depending on accuracy estimates or the spacing requirements between $P(t)$ and $P(t + \Delta t)$.

In this section, we cover two types of incremental methods. In Section 21.3.1 we discuss briefly the application of conventional methods such as Euler's and Runge-Kutta to the special case at hand of linearly varying vector fields. In Section 21.3.2 we discuss some incremental methods for computing values on the curve which are based upon the exact solutions which were explicitly given in Section 21.2. Before we proceed, we wish to briefly discuss two topics that come up in the implementation of incremental methods. Namely, how to choose Δt and how to test when the curve leaves the triangle. We take up the latter topic first.

The functions $f_i(x, y)$, $f_j(x, y)$, and $f_k(x, y)$ of Equation (21.4) were carefully defined so that $f_a(x, y) > 0$ for points (x, y) on the same side of e_a as P_a, $a = i, j,$ or k. This assumes that the points P_i, P_j, and P_k are listed in counterclockwise order. So by testing these functions, we can determine if the next point leaves the triangle and also which edge it leaves from and this can then be used to determine the data for the next triangle it enters. A particularly efficient way to compute all three values is based upon the identity

$$f(x, y) = x(f(1, 0) - f(0, 0)) + y(f(0, 1) - f(0, 0)) + f(0, 0) \tag{21.19}$$

where

$$f(x, y) = \begin{pmatrix} f_i(x, y) \\ f_j(x, y) \\ f_k(x, y) \end{pmatrix}.$$

As we previously mentioned, it is sometimes useful to be able to approximately control the distance between $P(t)$ and $P(t+\Delta t)$ by choosing an appropriate Δt. Say, for example, it is desired that

$$||P(t + \Delta t) - P(t)|| \approx \delta.$$

The mean value theorem implies

$$P(t + \Delta t) - P(t) = \Delta t P'(\tau), \qquad t \leq \tau \leq t + \Delta t.$$

Equation (21.6) then yields

$$P(t + \Delta t) - P(t) = \Delta t(A(P(\tau) + B), \qquad t \leq \tau \leq t + \Delta t.$$

Now a local estimate as in

$$\Delta t = \frac{\delta}{||AP(t) + B)||}$$

can be used or an overall estimate as in

$$\Delta t = \frac{\delta}{max\{||V_i||, ||V_j||, ||V_k||\}}.$$

This last estimate is based upon the property that $||AP + B||$ is bounded by $(\max\{||V_i||, ||V_j||, ||V_k||\})$ for P in the triangle of interest.

21.3.1 Conventional Methods Applied to Linearly Varying Vector Fields

It is possible to treat Equation (21.1) as an ordinary differential equation and to use standard and conventional numerical methods to compute approximations to $P(t)$. The most popular numerical methods in this context are incremental methods. The particular computational formulas for a sampling of these methods are:

Euler's:

$$P(t + \Delta t) \cong P(t) + \Delta t \cdot V(p(t))$$

Runge-Kutta 2^{nd} Order:

$$P(t + \Delta t) \cong P(t) + \tfrac{1}{2}(V_1 + V_2)$$
$$V_1 = \Delta t \cdot V(P(t))$$
$$V_2 = \Delta t \cdot V(P(t) + V_1)$$
$$P(t + \Delta t) = P(t) + \tfrac{1}{2}(V_1 + \Delta t \cdot V(P(t) + V_1))$$

Runge-Kutta 4^{th} Order:

$$P(t + \Delta t) \cong P(t) + \tfrac{1}{6}(V_1 + 2V_2 + 2V_3 + V_4)$$
$$V_1 = \Delta t \cdot V(p(t))$$
$$V_2 = \Delta t \cdot V(p(t) + \tfrac{1}{2}V_1)$$
$$V_3 = \Delta t \cdot V(p(t) + \tfrac{1}{2}V_2)$$
$$V_4 = \Delta t \cdot V(p(t) + \tfrac{1}{2}V_3)$$

Adaptive Runge-Kutta-Fehlberg:

$$P(t + \Delta t) \cong P(t) + \tfrac{16}{135}V_1 + 0V_2 + \tfrac{6656}{12825}V_3 + \tfrac{28561}{56430}V_4 - \tfrac{9}{50}V_5 + \tfrac{2}{55}V_6$$
$$\text{error estimate } = \tfrac{1}{360}V_1 + 0V_2 - \tfrac{128}{4275}V_3 - \tfrac{2197}{75240}V_4 + \tfrac{1}{50}V_5 + \tfrac{2}{55}V_6$$
$$V_1 = \Delta t \cdot V(p(t))$$
$$V_2 = \Delta t \cdot V(p(t) + \tfrac{1}{4}V_1)$$

$$V_3 = \Delta t \cdot V(p(t) + \tfrac{3}{32}V_1 + \tfrac{9}{32}V_2)$$
$$V_4 = \Delta t \cdot V(p(t) + \tfrac{1932}{2197}V_1 - \tfrac{7200}{2197}V_2 + \tfrac{7296}{2197}V_3)$$
$$V_5 = \Delta t \cdot V(p(t) + \tfrac{439}{216}V_1 - 8V_2 + \tfrac{3680}{513}V_3 - \tfrac{845}{4104}V_4)$$
$$V_6 = \Delta t \cdot V(p(t) - \tfrac{8}{27}V_1 + 2V_2 - \tfrac{3544}{2565}V_3 + \tfrac{1859}{4104}V_4 - \tfrac{11}{40}V_5)$$

The above formulas can be simplified somewhat when the special property that $V(P) = AP + B$ is utilized. The formulas in this case become:

Euler's:

$$P(t + \Delta t) \cong (I + \Delta tA)P(t) + \Delta tB$$

Runge-Kutta 2^{nd} Order:

$$P(t + \Delta t) \cong (I + \Delta tA + \tfrac{(\Delta tA)^2}{2})P(t) + \Delta t(I + \tfrac{\Delta tA}{2})B$$

Runge-Kutta 4^{th} Order:

$$P(t + \Delta t) \cong \left(\sum_{i=0}^{4} \tfrac{(\Delta tA)^i}{i!}\right) P(t) + \Delta t \left(\sum_{i=0}^{3} \tfrac{(\Delta tA)^i}{(i+1)!}\right) P(t)$$

Adaptive Runge-Kutta-Fehlberg:

$$P(t + \Delta t) \cong \sum_{i=0}^{5} \tfrac{(\Delta tA)^i}{i!} + \tfrac{(\Delta tA)^6}{2080} P(t) + \Delta t \left(\sum_{i=0}^{4} \tfrac{(\Delta tA)^i}{(i+1)!} + \tfrac{(\Delta tA)^5}{2080}\right) B$$

Error Estimate $\cong (\Delta tA)^5(\tfrac{\Delta tA}{2080} - \tfrac{1}{780})$

21.3.2 Incremental Methods for the Exact Solution

Here we are looking for a formula of the form

$$P(t + \Delta t) = G(\Delta t)P(t) + H(\Delta t)$$

which will allow computation of points at equal parameter values on the curve to be computed with only the application of an affine transformation. Repeated application of the basic differential equation which characterizes $P(t)$ leads to

$$P'(t) = AP(t) + B$$
$$P''(t) = AP'(t) = A^2P(t) + AB$$
$$P'''(t) = AP''(t) = A^3P(t) + A^2B$$

which leads to the expansion

$$\begin{aligned} P(t + \Delta t) &= (I + \Delta tA + \frac{(\Delta tA)^2}{2!} + \cdots)P(t) \\ &\quad + (\Delta t + (\Delta t)^2\frac{A}{2!} + (\Delta t)^3\frac{A^2}{3!} + \cdots)B \\ &= E(\Delta t)P(t) + F(\Delta t)B \end{aligned}$$

where $E(\Delta t) = I + F(\Delta t)A$. Both of these series for $E(\Delta t)$ and $F(\Delta t)$ have limits that can be approximately calculated in a variety of ways. One, of course, is to simply truncate the series. If we utilize the expressions for the solution given in Section 21.2.1, we arrive at the following particularly efficient methods which rely only upon the need to compute the functions e^t, $sin(t)$, and $cos(t)$.

The matrix E referred to below is (E_1, E_2) where these two vectors are defined in the different cases covered in Sections 21.2.1 and 21.2.2.

Case 1. $(0 \neq r_1 \neq r_2 \neq 0)$
If $|E| \neq 0$ then

$$P(t + \Delta t) - P_c = E \begin{pmatrix} e^{\Delta t r_1} & 0 \\ 0 & e^{\Delta t r_2} \end{pmatrix} E^{-1}(P(t) - P_c).$$

If $|E| = 0$ then P_0 is on one of the lines passing through P_c in the direction of the eigenvectors of A and either $E_1 = 0$ or $E_2 = 0$ (or $E_1 = E_2 = 0$ if $P_0 = P_c$). The tangent curve will be a straight line and so

$$P(t + \Delta t) = P(t) \pm \Delta t(P_c - P_0).$$

Case 2. $(0 = r_1, r_2 \neq 0)$

$$P(t + \Delta t) = [I + \frac{(e^{\Delta t r_2} - 1)}{r_2} A]P(t) + \Delta t[I + \frac{(e^{\Delta t r_2} - \Delta t r_2 - 1)}{\Delta t r_2^2} A]B.$$

Case 3. $(r_1 = 0 = r_2)$

$$P(t + \Delta t) = (I + \Delta t A)P(t) + (\Delta t + \frac{(\Delta t)^2 A}{2})B.$$

Case 4. $(r_1 = r_2 \neq 0)$
If $|E| \neq 0$, then

$$P(t + \Delta t) - P_c = E e^{\Delta t r_1} \begin{pmatrix} 1 & 0 \\ \Delta t & 1 \end{pmatrix} E^{-1}(P(t) - P_c).$$

If $|E| = 0$, then $P(t)$ is a straight line and so

$$P(t + \Delta t) = P(t) \pm \Delta t(P_c - P_0).$$

Case 5. $(\mu + \lambda i, \mu - \lambda i, \lambda \neq 0)$

$$P(t + \Delta t) - P_c = E e^{\mu \Delta t} \begin{pmatrix} cos(\lambda \Delta t) & -sin(\lambda \Delta t) \\ sin(\lambda \Delta t) & cos(\lambda \Delta t) \end{pmatrix} E^{-1}(P(t) - P_c).$$

21.4 Working Directly with Barycentric Coordinates

In this section, we discuss the representation and computation of tangent curves using barycentric coordinates rather than conventional Cartesian coordinates. There are some potential advantages to the use of barycentric coordinates depending upon the particular application. One advantage to barycentric coordinates is that it is trivial to determine whether or not a point on a tangent curve is inside the triangular domain or not, as all three barycentric coordinates must be positive in this case. Another less obvious advantage is for applications where the triangular domain is not located in the normal xy-plane. We will show some examples of this later where the triangular domains consist of a triangulated approximation of the sphere. Here, even though the flow vectors are 3D vectors, they are in the plane of the domain and so all that we have developed in this chapter still applies. It would be possible to use a 3D affine transformation to move the problem to the plane, solve it, and transform back. But there is no need to do this transformation if we do all calculations in barycentric coordinates. But it should also be noted that most graphics routines require Cartesian coordinates and so the particular application environment has to be taken into consideration.

For completeness, we give some background on barycentric coordinates. Given a point

$$P = \begin{pmatrix} x \\ y \end{pmatrix},$$

barycentric coordinates, b_i, b_j, and b_k of this point relative to the triangle T_{ijk} with vertices P_i, P_j, and P_k are defined by the relationships

$$\begin{aligned} \begin{pmatrix} x \\ y \end{pmatrix} &= b_i P_i + b_j P_j + b_k P_k \\ 1 &= b_i + b_j + b_k \end{aligned} \tag{21.20}$$

There are several alternative ways of defining or determining the barycentric coordinates of a point. For example,

$$b_i = \frac{A_i}{A}, \qquad b_j = \frac{A_j}{A}, \qquad b_k = \frac{A_k}{A},$$

where A_i, A_j, and A_k represent the areas of the subtriangle shown in Figure 21.10 and A is the area of T_{ijk}. Also,

$$b_i = \frac{\begin{vmatrix} x - x_k & x_j - x_k \\ y - y_k & y_j - y_k \end{vmatrix}}{\begin{vmatrix} x_i - x_k & x_j - x_k \\ y_i - y_k & y_j - y_k \end{vmatrix}}, \qquad b_j = \frac{\begin{vmatrix} x - x_i & x_i - x_k \\ y - y_i & y_i - y_k \end{vmatrix}}{\begin{vmatrix} x_j - x_k & x_i - x_k \\ y_j - y_k & y_i - y_k \end{vmatrix}},$$

$$b_k = \frac{\begin{vmatrix} x - x_j & x_i - x_j \\ y - y_j & y_i - y_j \end{vmatrix}}{\begin{vmatrix} x_k - x_j & x_i - x_j \\ y_k - y_j & y_i - y_j \end{vmatrix}}. \tag{21.21}$$

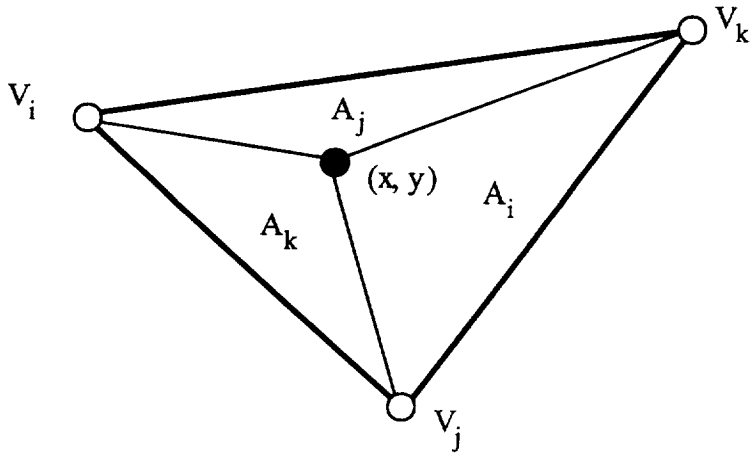


Figure 21.10: Areas leading to barycentric coordinates.

We can note that b_i, b_j, and b_k are simply linear functions of x and y. In fact, b_i is just the linear function which is one at P_i and zero at P_j and P_k. It will be convenient to explicitly denote this dependence so that we have

$$b_i = b_i \begin{pmatrix} x \\ y \end{pmatrix}$$

and the relationship

$$\begin{pmatrix} x \\ y \end{pmatrix} = P_i b_i \begin{pmatrix} x \\ y \end{pmatrix} + P_j b_j \begin{pmatrix} x \\ y \end{pmatrix} + P_k b_k \begin{pmatrix} x \\ y \end{pmatrix} = (P_i, P_j, P_k) \begin{pmatrix} b_i \begin{pmatrix} x \\ y \end{pmatrix} \\ b_j \begin{pmatrix} x \\ y \end{pmatrix} \\ b_k \begin{pmatrix} x \\ y \end{pmatrix} \end{pmatrix}.$$

Also we will denote the column vector consisting of the three barycentric coordinates as

$$b \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b_i \begin{pmatrix} x \\ y \end{pmatrix} \\ b_j \begin{pmatrix} x \\ y \end{pmatrix} \\ b_k \begin{pmatrix} x \\ y \end{pmatrix} \end{pmatrix}.$$

It can be noted that

$$b \begin{pmatrix} x \\ y \end{pmatrix} = xb \begin{pmatrix} 1 \\ 0 \end{pmatrix} + yb \begin{pmatrix} 0 \\ 1 \end{pmatrix} + (1 - x - y) b \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad (21.22)$$

and

$$b(\alpha P + (1-\alpha)Q) = \alpha b(P) + (1-\alpha)b(Q). \tag{21.23}$$

The 3×3 matrix whose columns consist of the barycentric coordinates of the three flow vectors at each of the vertices will be denoted by

$$V = (b(V_i), b(V_j), b(V_k)).$$

It should be noted that if

$$Q = \begin{pmatrix} q_i \\ q_j \\ q_k \end{pmatrix}$$

has the property that $q_i + q_j + q_k = 1$ (as will be the case when Q represents the barycentric coordinates of a point) then VQ will have this same property. It is also true that if $q_i + q_j + q_k = 0$ then the entries of VQ will also sum to zero.

It is easy to verify that

$$A = (P_i, P_j, P_k)V\left(b\begin{pmatrix}1\\0\end{pmatrix} - b\begin{pmatrix}0\\0\end{pmatrix},\ b\begin{pmatrix}0\\1\end{pmatrix} - b\begin{pmatrix}0\\0\end{pmatrix}\right) \tag{21.24}$$

and

$$A\begin{pmatrix}x\\y\end{pmatrix} + B = (P_i, P_j, P_k)Vb\begin{pmatrix}x\\y\end{pmatrix}. \tag{21.25}$$

Also

$$Vb\begin{pmatrix}0\\0\end{pmatrix} = b(B)$$

and if there exists a point P_c such that

$$AP_c + B = \begin{pmatrix}0\\0\end{pmatrix}$$

then

$$b\begin{pmatrix}0\\0\end{pmatrix} = Vb(P_c).$$

We should point out that the 3×3 matrix V has the same eigenvalues as A and if $AE = rE$ for a 2D vector E and scalar r, then

$$V(b(E) - b\begin{pmatrix}0\\0\end{pmatrix}) = r(b(E) - b\begin{pmatrix}0\\0\end{pmatrix}).$$

In addition V has the eigenvalue 1.

The tangent curve $P(t)$ which was previously given in Cartesian coordinates will be denoted in barycentric coordinates by

$$p(t) = b(P(t)) = \begin{pmatrix} b_i(t) \\ b_j(t) \\ b_k(t) \end{pmatrix}$$

and the basic differential equation which characterizes $P(t)$ now takes the form

$$p'(t) = Vp(t) - b\begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{21.26}$$

with $p(0) = b(P_0)$.

Using the same notation as in Equation (21.7) we can now represent the tangent curve which is the solution of Equation (21.26) in barycentric coordinates by

$$p(t) = \Phi_1(t)\left[b(E_1) - b\begin{pmatrix} 0 \\ 0 \end{pmatrix}\right] + \Phi_2(t)\left[b(E_2) - b\begin{pmatrix} 0 \\ 0 \end{pmatrix}\right] + b(C) \tag{21.27}$$

where, as before, each of the cases selected by the eigenvalues of A (or V now) determine the bases functions $\Phi_1(t)$ and $\Phi_2(t)$ and the vectors E_1, E_2, and C. We now discuss incremental methods in terms of barycentric coordinates. In this context we are looking for a formula of the form

$$p(t + \Delta t) = g(\Delta t)p(t) + h(\Delta t). \tag{21.28}$$

Similar to before, we can use repeated applications of Equation (21.26) to yield

$$\begin{aligned} p(t + \Delta t) &= (I + \Delta tV + \frac{(\Delta tV)^2}{2!} + \cdots)p(t) \\ &\quad -(\Delta tI + \frac{(\Delta t)^2 V}{2!} + \frac{(\Delta t)^3 V^2}{3!} + \cdots)b\begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ &= e(\Delta t)p(t) + f(\Delta t)b\begin{pmatrix} 0 \\ 0 \end{pmatrix}. \end{aligned} \tag{21.29}$$

Also, we can note that $e(\Delta t) = I + f(\Delta t)V$. These series for $e(\Delta t)$ and $f(\Delta t)$ can be approximated in a variety of ways. As we saw earlier, if we truncate the series, then we have the equivalent of an Euler or Runge-Kutta method applied to the special case of linearly varying vector fields. The same type of results holds in the case of barycentric coordinates. The results are:

Method	$e(\Delta t)$	$f(\Delta t)$
Euler	$(I + \Delta t V)$	$-(\Delta t I)$
Runge-Kutta 2^{nd} Order	$(I + \Delta t V + \frac{1}{2}\Delta t^2 V^2)$	$-(\Delta t I + \frac{1}{2}\Delta t^2 V)$
Runge-Kutta 4^{th} Order	$(I + \Delta t V + \frac{1}{2}\Delta t^2 V^2 + \frac{1}{6}\Delta t^3 V^3 + \frac{1}{24}\Delta t^4 V^4)$	$-(\Delta t I + \frac{1}{2}\Delta t^2 V + \frac{1}{6}\Delta t^3 V^2 + \frac{1}{24}\Delta t^4 V^3)$
Adaptive Runge-Kutta-Fehlberg	$(I + \Delta t V + \frac{1}{2}\Delta t^2 V^2 + \frac{1}{6}\Delta t^3 V^3 + \frac{1}{24}\Delta t^4 V^4 + \frac{1}{120}\Delta t^5 V^5 + \frac{1}{2080}\Delta t^6 V^6)$	$-(\Delta t I + \frac{1}{2}\Delta t^2 V + \frac{1}{6}\Delta t^3 V^2 + \frac{1}{24}\Delta t^4 V^3 + \frac{1}{120}\Delta t^5 V^4 + \frac{1}{2080}\Delta t^6 V^5)$
Adaptive Runge-Kutta-Fehlberg error estimation	$(\frac{-1}{780}\Delta t^5 V^5 + \frac{1}{2080}\Delta t^6 V^6)$	$-(\frac{-1}{780}\Delta t^5 V^5 + \frac{1}{2080}\Delta t^6 V^6)$

Similar to what was done in Section 21.3.2, we can develop exact representations utilizing transcendental functions. The details for the various cases follow. The matrix D referred to below is

$$\left(b(E_1) - b\begin{pmatrix} 0 \\ 0 \end{pmatrix}, b(E_2) - b\begin{pmatrix} 0 \\ 0 \end{pmatrix}\right)$$

and $D^+ = (D^t D)^{-1} D^t$. The two vectors E_1 and E_2 are defined in the different cases covered in Sections 21.2.1 and 21.2.2.

Case 1. $(0 \neq r_1 \neq r_2 \neq 0)$

If $|E| \neq 0$, then

$$p(t + \Delta t) - b(P_c) = D\begin{pmatrix} e^{\Delta t r_1} & 0 \\ 0 & e^{\Delta t r_2} \end{pmatrix} D^+ (p(t) - b(P_c)).$$

If $|E| = 0$, then P_0 is on one of the lines passing through P_c in the direction of the eigenvectors of A and either $E_1 = 0$ or $E_2 = 0$ (or $E_1 = E_2 = 0$ if $P_0 = P_c$). The tangent curve will be a straight line and so

$$p(t + \Delta t) = p(t) \pm \Delta t (b(P_c) - b(P_0)).$$

Case 2. $(0 = r_1, r_2 \neq 0)$

$$\begin{aligned} p(t + \Delta t) \;=\; & \left(I + \frac{V}{r_2}\left(\frac{V - I}{r_2 - 1}\right)(e^{\Delta t r_2} - 1) + V\left(\frac{r_2 I - V}{r_2 - 1}\right)(e^{\Delta t} - 1)\right) p(t) \\ & -\Delta t \left(I + \frac{V}{\Delta t r_2^2}\left(\frac{V - I}{r_2 - 1}\right)(e^{\Delta t r_2} - \Delta t r_2 - 1)\right. \\ & \left. + \frac{V}{\Delta t}\left(\frac{r_2 I - V}{r_2 - 1}\right)(e^{\Delta r} - \Delta r - 1)\right) b\begin{pmatrix} 0 \\ 0 \end{pmatrix}. \end{aligned}$$

Case 3. $(r_1 = 0 = r_2)$

$$\begin{aligned} p(t+\Delta t) &= (I + \Delta t V + (e^{\Delta t} - \Delta t - 1)V^2)p(t) \\ &\quad -(\Delta t I + \frac{(\Delta t)^2 V}{2} + (e^{\Delta t} - \frac{(\Delta t)^2}{2} - \Delta t - 1)V^2)b \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \end{aligned}$$

Case 4. $(r_1 = r_2 \neq 0)$
If $|E| \neq 0$, then

$$p(t+\Delta t) - b(P_c) = De^{\Delta t r_1} \begin{pmatrix} 1 & 0 \\ \Delta t & 1 \end{pmatrix} D^+(p(t) - b(P_c))$$

If $|E| = 0$, then the tangent curve is a straight line and so

$$p(t+\Delta t) = p(t) \pm \Delta t(b(P_c) - b(P_0)).$$

Case 5. $(\mu + \lambda i, \mu - \lambda i, \lambda \neq 0)$

$$\begin{aligned} &p(t+\Delta t) - b(P_c) \\ &\qquad = De^{\mu \Delta t} \begin{pmatrix} cos(\lambda \Delta t) & -sin(\lambda \Delta t) \\ sin(\lambda \Delta t) & cos(\lambda \Delta t) \end{pmatrix} D^+(p(t) - b(P_c)). \end{aligned}$$

21.5 Topological Methods

Topological methods provide a clear and uncluttered means of visualizing a two-dimensional vector field. They give a good overview of the flow with relatively little information, but they can require some effort to compute. They were first introduced into the visualization literature by Helman and Hesselink [17]. They consist of a collection of tangent curves which separate the flow into regions. The tangent curve boundaries of these regions connect together certain critical points. Critical points are locations where the flow is zero.

In a nutshell, there are two main steps to computing a topological graph. First, all critical points are computed and classified on the basis of the nature of the flow near the critical point. In the second step, certain tangent curves are started at critical points and traversed out until they either link up with other critical points or leave the domain. In this section, we will discuss what is necessary to compute a topological graph within the special context of this chapter which is piecewise linear vector fields over triangulated domains. An example of a topological graph is shown in Figure 21.11.

We now discuss some details of the critical point computation and classification step of the algorithm. In a general context with the vector field

$$V(x,y) = \begin{pmatrix} u(x,y) \\ v(x,y) \end{pmatrix},$$

a critical point, P_c, is simply a position where

$$V(P_c) = \begin{pmatrix} u(x_c, y_c) \\ v(x_c, y_c) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

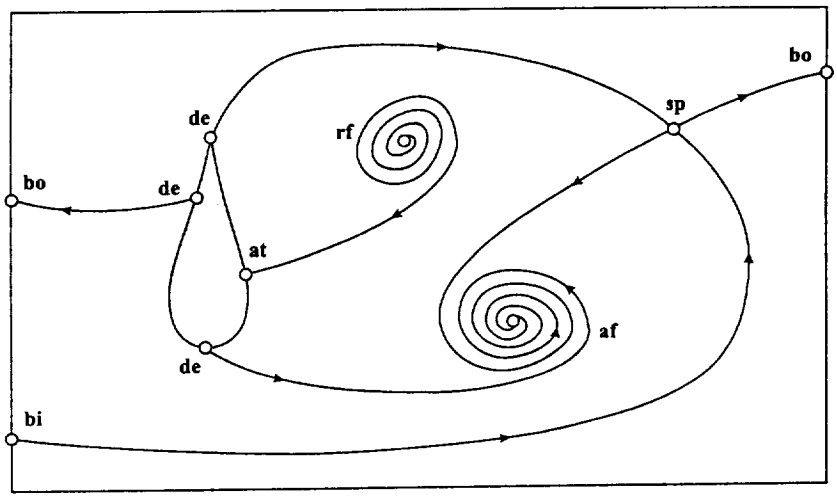


Figure 21.11: A topological graph of a two dimensional vector field.

The local behavior close to a critical point is determined by the Jacobian

$$J(x_c, y_c) = \begin{pmatrix} u_x(x_c, y_c) & u_y(x_c, y_c) \\ v_x(x_c, y_c) & v_y(x_c, y_c) \end{pmatrix}.$$

This is due to the fact that the flow field is approximated by the linear terms of its expansion near the critical point. That is,

$$V(P) \cong V(P_c) + (P - P_c)J(P_c) = (P - P_c)J(P_c).$$

Using techniques similar to those used in Section 21.2.1, it is determined that there are six different types of critical points (see Figure 21.12):

i) Saddle Point $J(P_c)$ has two real, nonzero eigenvalues which differ in sign
ii) Attracting Node $J(P_c)$ has two negative eigenvalues
iii) Repelling Node $J(P_c)$ has two positive eigenvalues
iv) Attracting Focus $J(P_c)$ has complex eigenvalues $\mu + \lambda i$, $\mu - \lambda i$, $\lambda \neq 0$, $\mu < 0$
v) Repelling Focus $J(P_c)$ has complex eigenvalues $\mu + \lambda i$, $\mu - \lambda i$, $\lambda \neq 0$, $\mu > 0$
vi) Center $J(P_c)$ has purely imaginary eigenvalues $+\lambda i$ and $-\lambda i$, $\lambda \neq 0$

In general, the computation of critical points can be a rather formidable problem. For example, in the case of curvilinear grids, cells must be searched for possible candidate cells and then a nonlinear system (two equations in two unknowns) of equations must be solved. This normally requires some iterative method which could possibly fail to converge or converge to a point which is actually not a critical point. In the case of a linearly varying vector field, the situation is much simpler. Only the linear system

$$A \begin{pmatrix} x_c \\ y_c \end{pmatrix} + B = 0$$

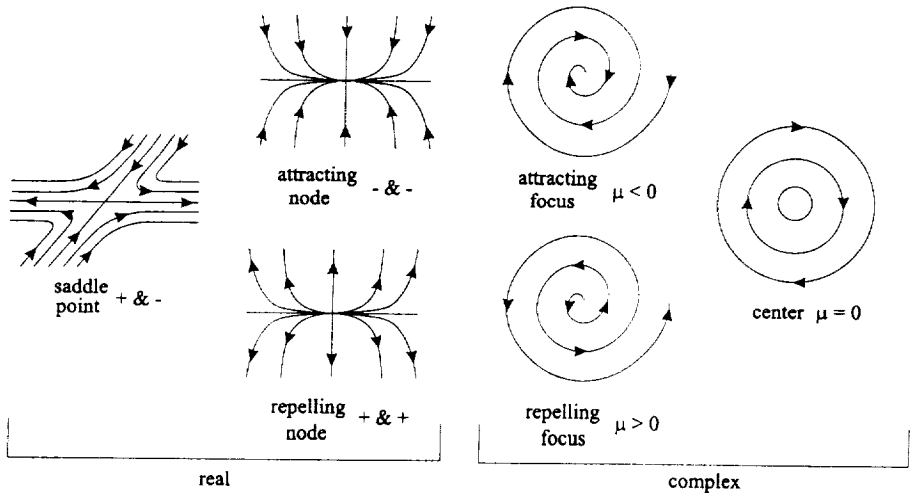


Figure 21.12: Characterization of critical values.

must be solved. Note that A and B are defined the same as earlier in Equation (21.2). We next determine if this is a "real" critical point by deciding whether or not P_c is actually in the triangle. This determination can be made by computing the barycentric coordinates of P_c (or possibly a scaling of them) and checking to see if all three of them are positive. More discussion on barycentric coordinates can be found in the previous Section 21.4.

Often, a flow field will have an interior boundary which represents an object about which the flow is being analyzed. All the points on this interior boundary are critical points as the flow is forced to be zero here. Some of these points are of special interest. These are the points of attachment (at) and detachment (de) which belong to tangent curves which separate the flow along and near the inner boundary. In some previous discussion in the literature, the characterization of these points has not been so rigorous, but here in the context of piecewise linear flow fields, we can be very precise. Consider a triangle with only one vertex on the inner boundary. This vertex will necessarily be a critical point. If the eigenvalues of the Jacobian for this triangle indicate that this critical point is classified as a saddle point, then it is a candidate for a point of attachment or detachment. It will be a point of attachment if the eigenvector (or its negative) associated with the negative eigenvalue lies between the two edges of the triangle sharing this critical point. If an eigenvector associated with the positive root lies in the triangle, then this critical point will be a point of detachment. See Figure 21.14.

It should be noted that it is possible for a single point to be both a point of attachment and a point of detachment. This is illustrated in Figure 21.15. We should also point out that it is possible for a single point on the inner boundary to belong to two different triangles which have only this point on the inner boundary and because of this, this single point could be classified as a certain type of critical point for one triangle and another (or the same) type of critical point as a vertex for a different triangle. An example of this is shown

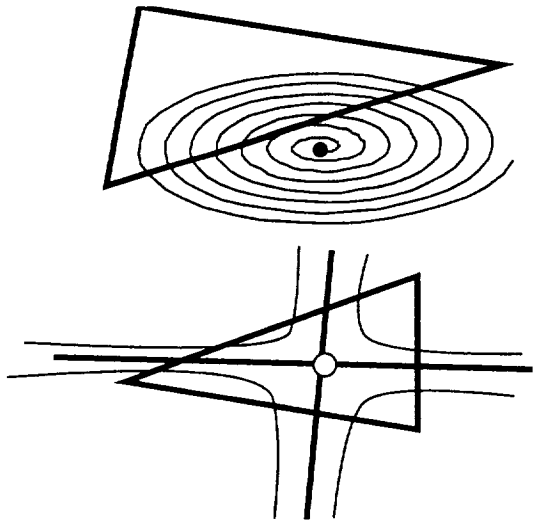


Figure 21.13: Bottom critical point is "real," but not the top one.

in Figure 21.16 where the critical point on the upper-left portion of the inner boundary is a point of attachment for one triangle and a point of detachment for an adjacent triangle.

The second major step in computing a topological graph consists of the linking algorithm. From each saddle point which is interior to a triangle, four tangent curves will emanate—two associated with each eigenvector of the Jacobian. One curve emanates in the direction of the eigenvector and one in the negative direction of the eigenvector. The curves emanating in the directions of the eigenvector associated with the positive eigenvalue will be traversed in positive parameter direction and these curves will move along with the flow. They have the chance to link up with attracting nodes or foci or possibly other saddle points along the eigenvectors of inward flow to the saddle point. The two curves emanating in the direction of the eigenvectors associated with the negative eigenvalue will be traversed in a negative parameter direction and will move along in a direction opposite (or negative) to the direction of the flow field. They have the chance to link up with repelling nodes or foci or other saddle points. From each point of detachment, one curve will emanate and be traversed in positive parameter direction. From each point of attachment, one curve will emanate and be traversed in a negative or opposite direction to the flow. The algorithm is complete when each of these curves has linked to another critical point or leaves the domain either by encountering the outer boundary or the inner boundary. An example of the results of this algorithm are shown in Figure 21.16. We should point out that this linking algorithm does not produce all separating tangent curves for it is possible that in addition to the center points, some repelling or attracting nodes or foci could be left unconnected to any tangent curve. However, this does not occur when the domain is a triangulated approximation to the sphere.

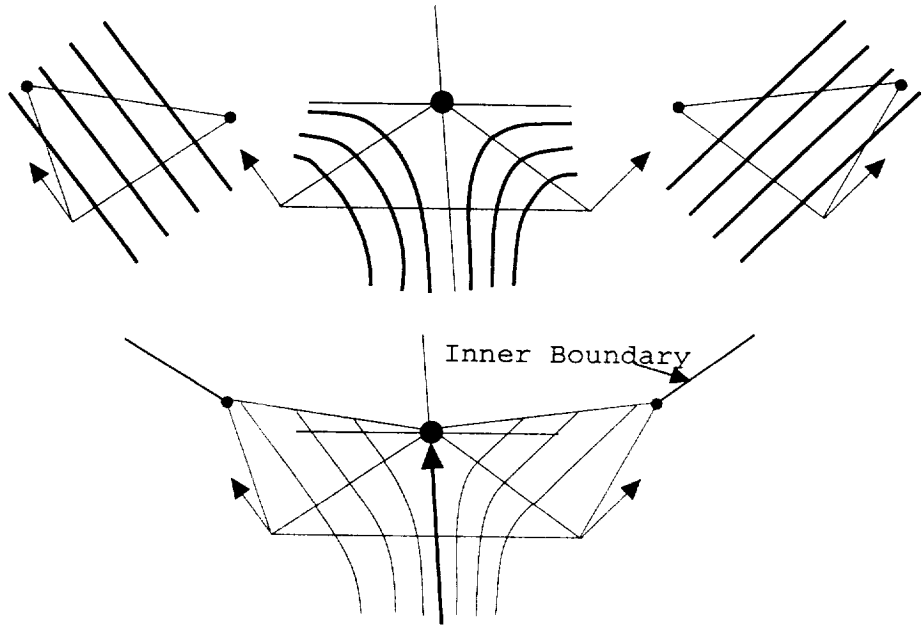


Figure 21.14: Diagram to support definition of point of attachment.

21.6 Examples

The first example that we include uses a simple quadratic equation to specify the vector field. The equations are given in Equation (21.30). One reason for including this type of example is to allow other implementors to easily verify and compare their software results.

$$\begin{aligned} U(x,y) &= -0.103209 + 0.051511x - 0.302699y \\ &\quad +0.037546xy - 0.232875x^2 + 0.611528y^2 \\ V(x,y) &= 0.143656 + 0.687847x - 0.144779y \\ &\quad -0.213010xy - 1.029676x^2 + 0.246278y^2 \end{aligned} \tag{21.30}$$

As we have mentioned earlier, the methods described here can be applied to any triangulated domain. In Figure 21.18 we show the topological graph and some additional tangent curves (in cyan) for a vector field defined over a triangulation of a spherical domain. Similar to Figure 21.17, two different resolutions of the triangulation are shown. In the right column, each triangle of the left column has been replaced by four subtriangles. We have intentionally used flat shading rather than Gourard shading for the rendering of the sphere so that the triangulation is apparent.

In the next example, we illustrate the use of the methods developed here to visualize a multiresolution model of a vector field over a curvilinear grid. The topological graph is

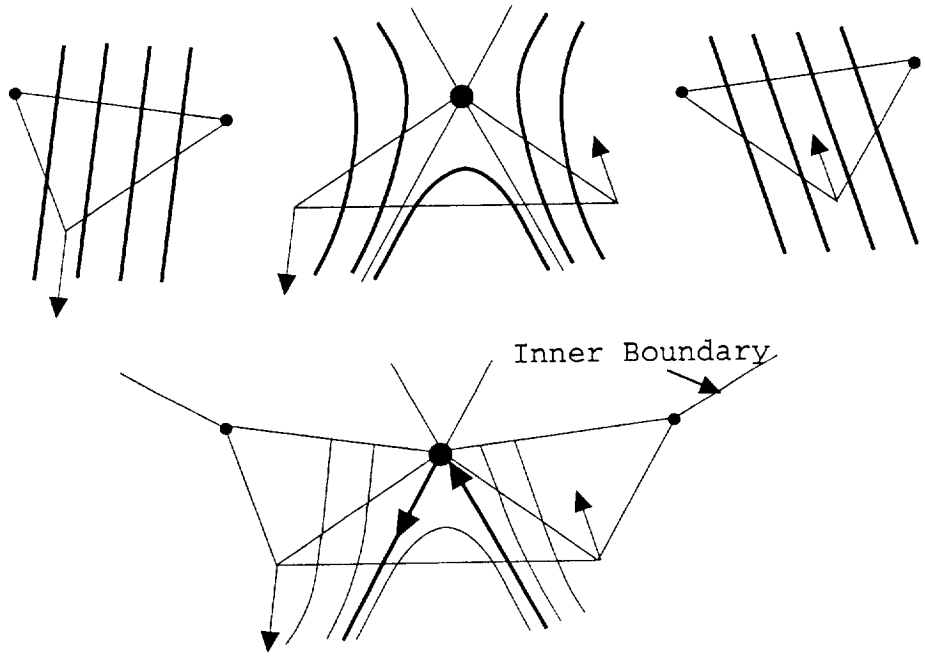


Figure 21.15: A point on the inner boundary which is both a point of attachment and a point of detachment.

shown at four different levels of approximation in Figure 21.19. One unique feature allowed by these methods is that no matter how coarse the resolution becomes, the boundaries have not changed from the original data. This is a particularly important aspect for the inner boundary which is often the focus of attention for a flow analysis. In Figure 21.20, we show the topological graph for some partial reconstructions of the flow field. These methods allow the user to zoom in and out and only reconstruct the portion of interest.

In Figure 21.21 we show results similar to those of Figures 21.19 and 21.20 except that now the domain is a triangulated sphere and the reconstruction is not done on the basis of regions, but on the basis of the magnitude of the coefficients of the wavelet basis functions. This data represents "real" data provided to us by Roger Crawfis and Nelson Max of Lawrence Livermore National Laboratory. It is one time step and one tier of simulated wind velocity data.

The data used for the examples shown in the images of Figure 21.22 and Figure 21.23 was provided to us by Yasuo Nakajima, Nissan Motor Company, Japan. Actually, the domain of this data is three-dimensional and not two-dimensional as required for the algorithms covered here. Figure 21.23 shows one slice through this 3D data. The 3D velocity vectors were projected into the plane of the slice. Many of the results of this chapter have been extended to 3D domains where the vector field is assumed to vary linearly over a tetrahedrization of the domain. See [33] for more discussion on tetrahedrizing a 3D curvilinear grid. The tangent curves and critical points shown in Figure 21.22 were computed using

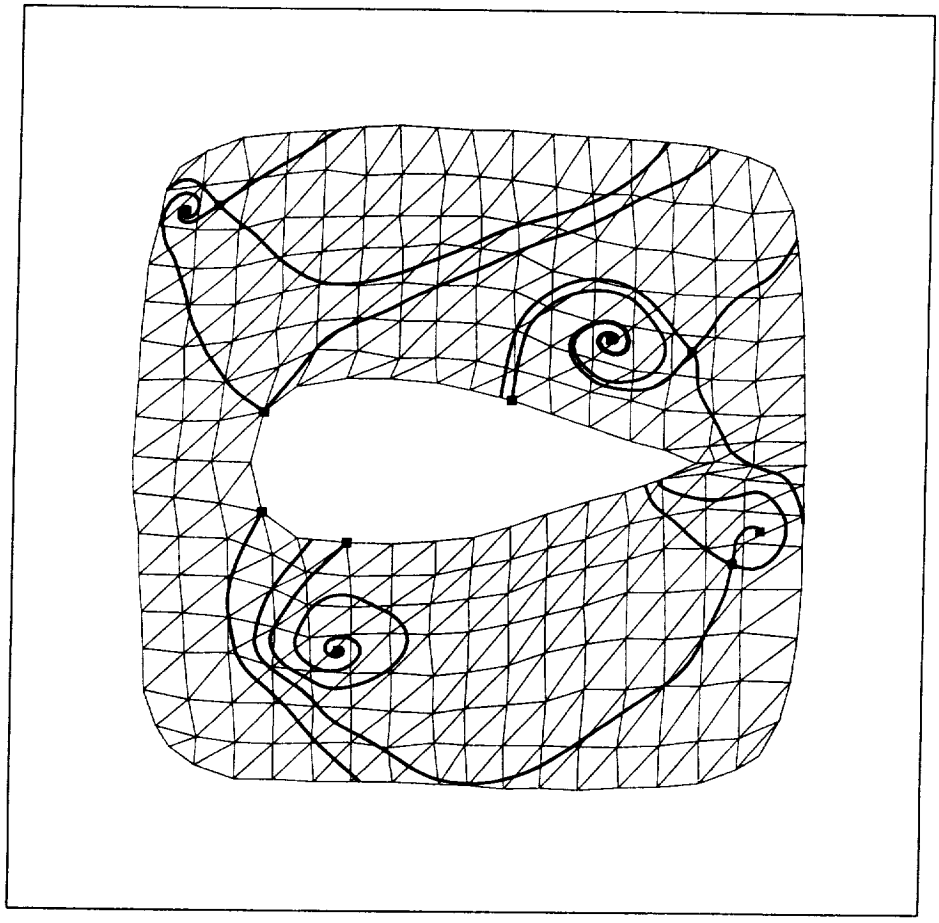


Figure 21.16: Topological graph for a curvilinear grid. A point on the inner boundary is both a point of attachment and a point of detachment.

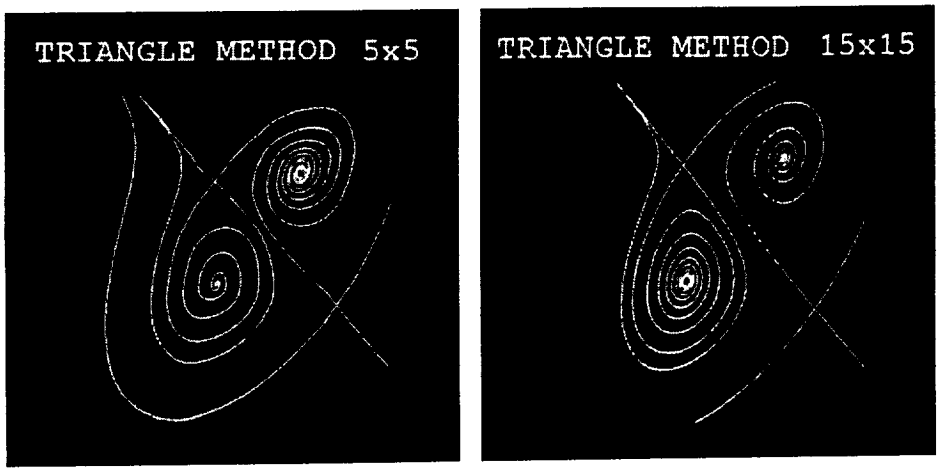


Figure 21.17: Explicit method used to compute topological graph for vector field given by Equation (21.30). Two different resolutions for the triangulation are shown.

extensions of the algorithms discussed here. See [22] for more details.

21.7 Conclusions and Remarks

1. One of the advantages of the explicit methods we have developed here is that accuracy can be monitored and controlled. Runge-Kutta and other incremental methods for solving ODEs are notorious for "wandering off" the true solution and once an error is made, there is no way to recover because from the erroneous point forward, the method is attempting to solve a different (wrong) problem than the one it set out to solve in the beginning. Variable step size methods which estimate the error and attempt to control it by adaptively changing the step size are helpful in this regard, but the problem is that the error is only estimated and only guesses about the proximity of the computed solution to the desired solution can be made. This is the case for the R-F-K method covered in Sections 21.3.1 and 21.4. In the proposed method, the accumulated error is a result of how accurately the intersection of a cell boundary and a particular explicitly defined tangent curve are computed. This is formulated as a root finding problem and so this computation can be done as accurately as deemed necessary.

2. Another advantage of the present method is speed and efficiency. Since the tangent curve is known explicitly for each triangle, parameters pertaining to this definition can be precomputed and stored and then used when a particular tangent curve penetrates this triangle. The global shape of the tangent curve is known by its sequence of entry and exit points for each triangle it intersects and so the overall appearance of the curve is not seriously degraded if a local approximation for each cell is used. We have used a parametric cubic Hermite curve on each triangle with good success.

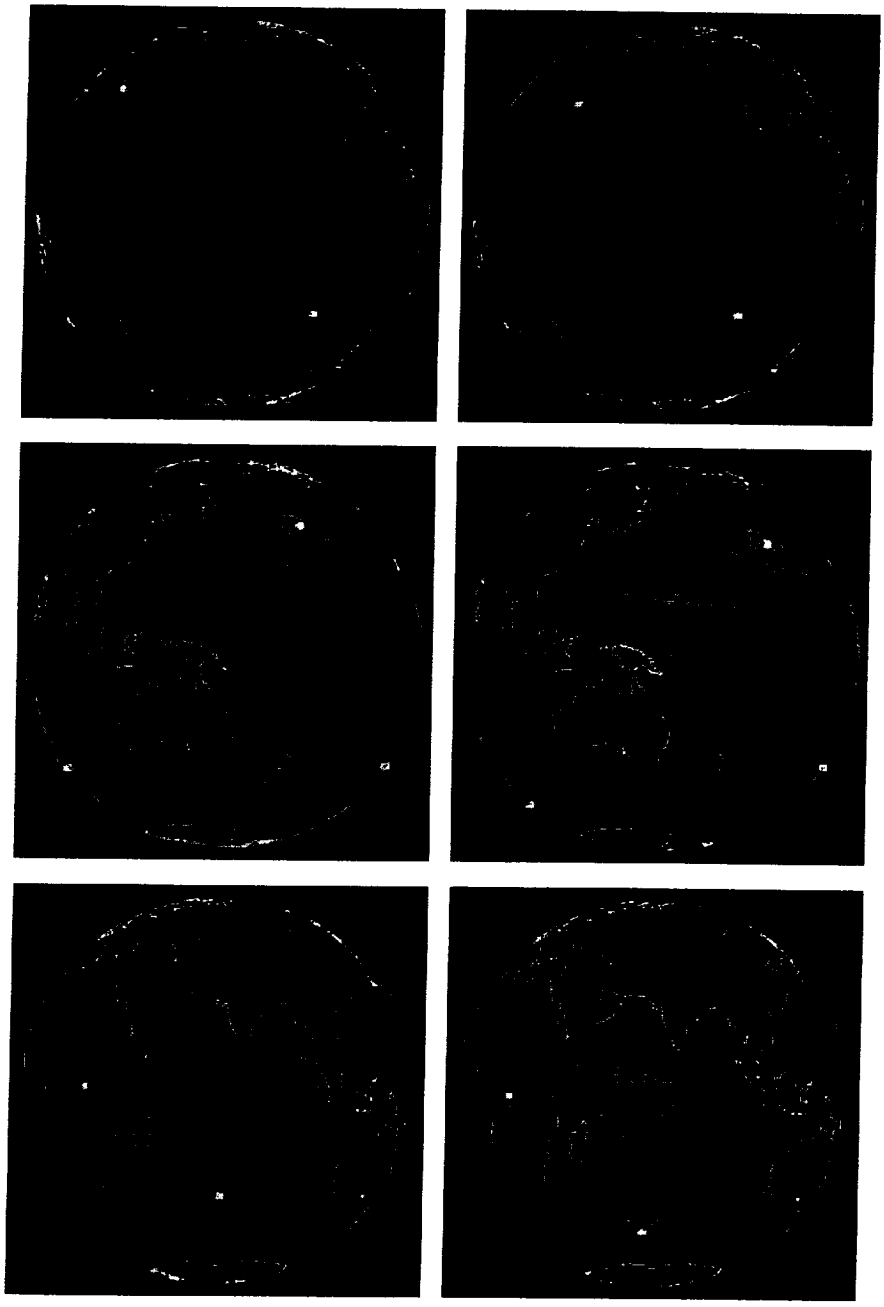


Figure 21.18: Explicit method used to compute critical values and tangent curves for a vector field defined over a spherical domain.

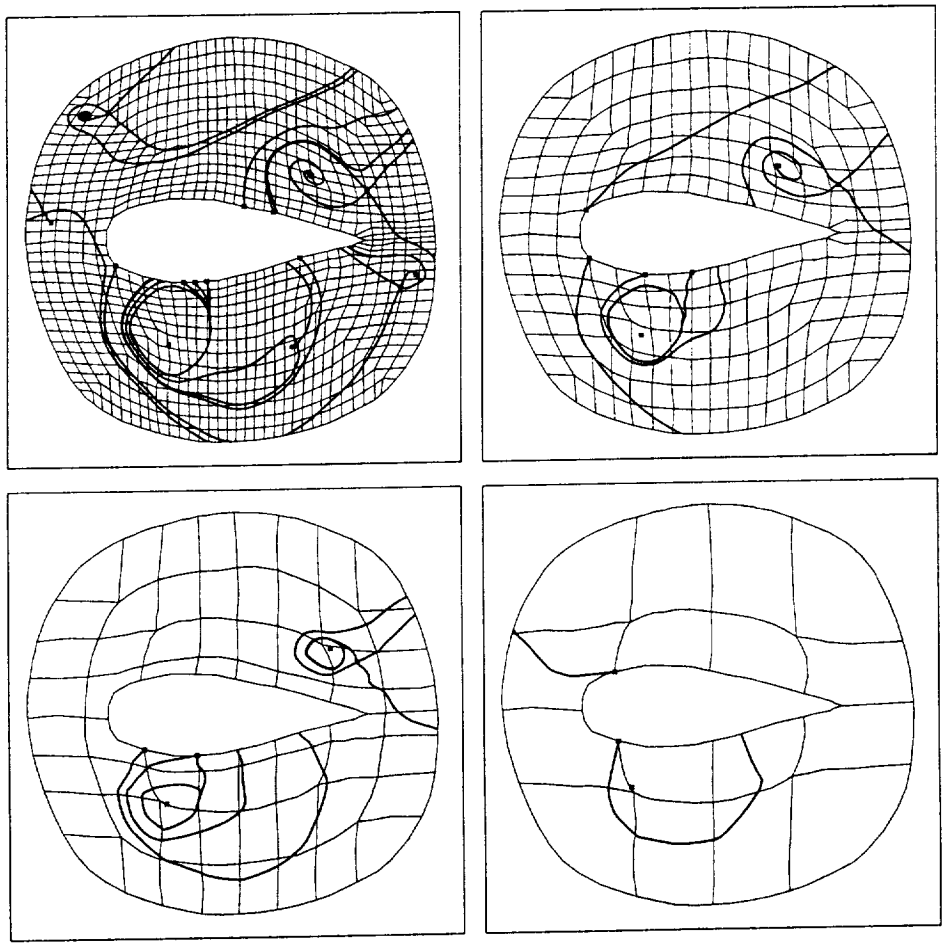


Figure 21.19: The explicit method is used to compute the topological graph for several different resolutions of a curvilinear grid.

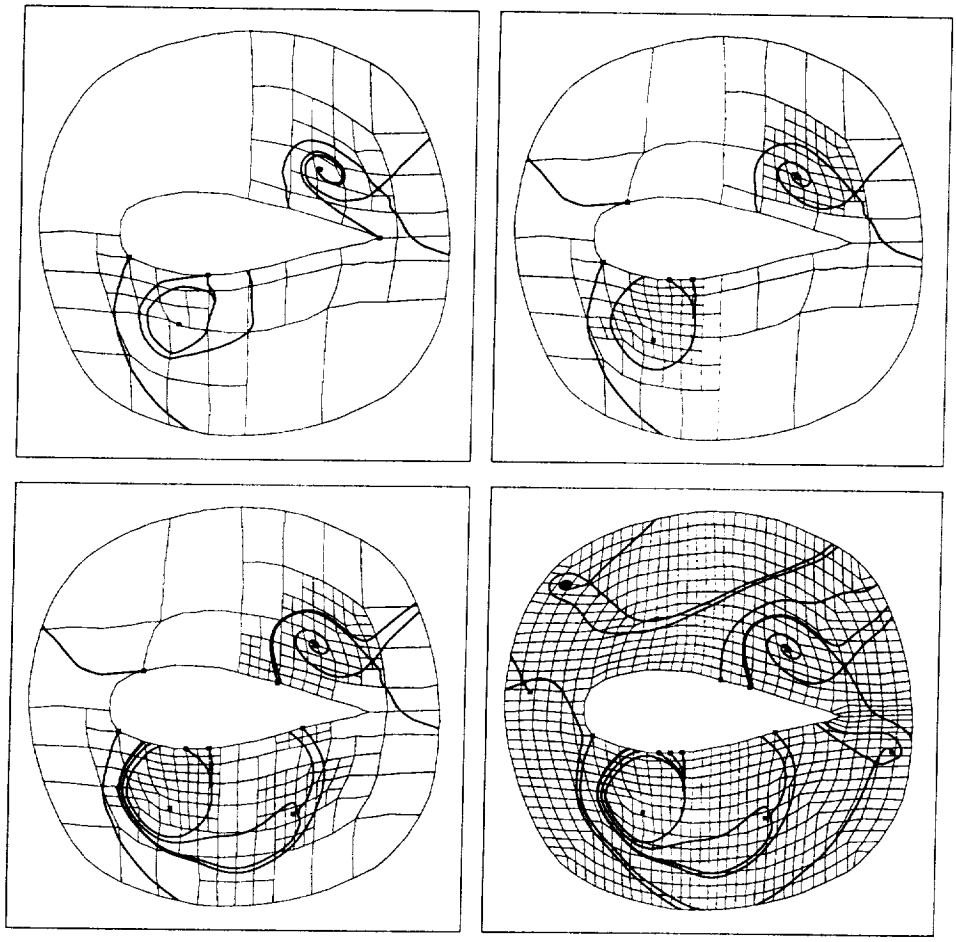


Figure 21.20: Partial wavelet reconstruction of the flow over a curvilinear grid indicating the efficiencies of zooming in or out.

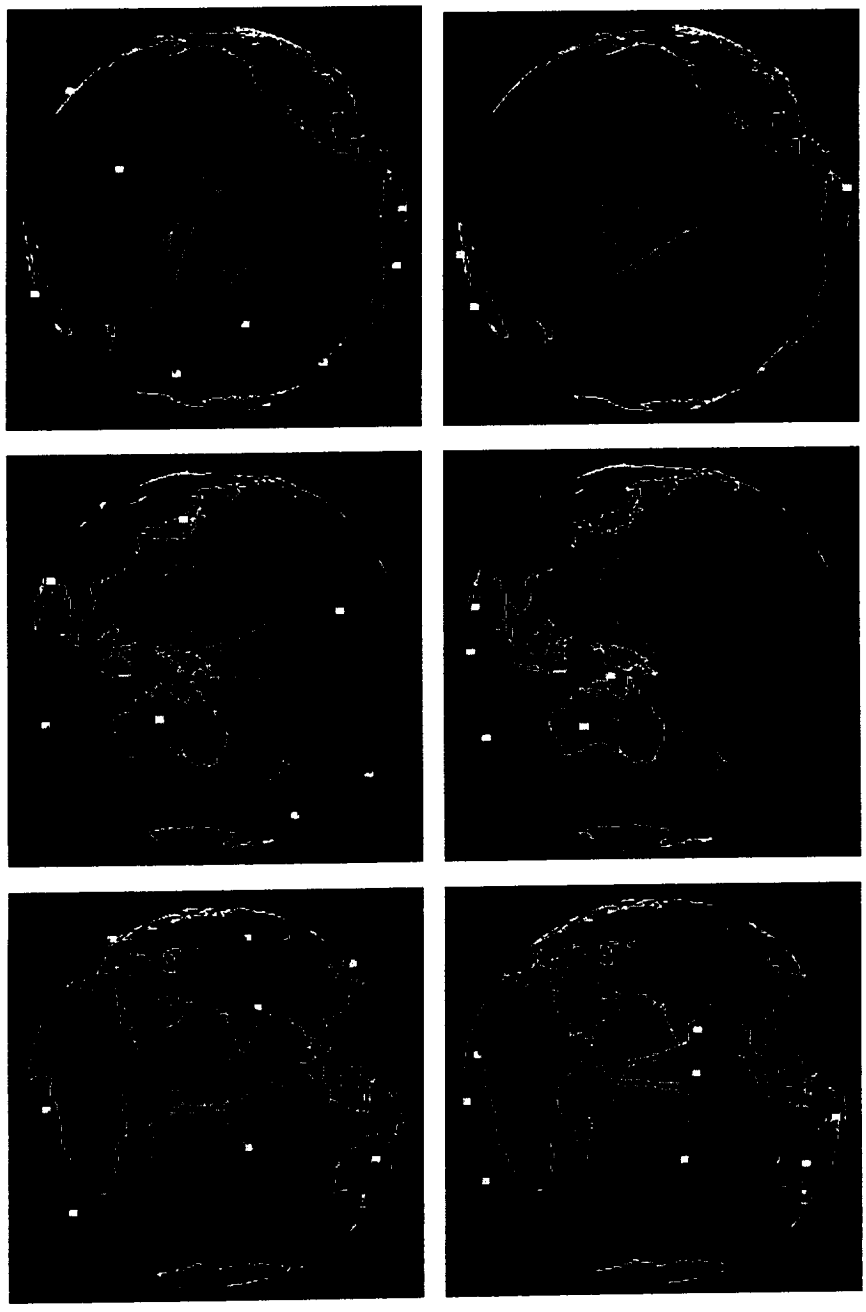


Figure 21.21: Topological graphs for wind data over the earth are computed using the explicit method. The right column is wavelet reconstruction based on the largest 3% of the wavelet coefficients.

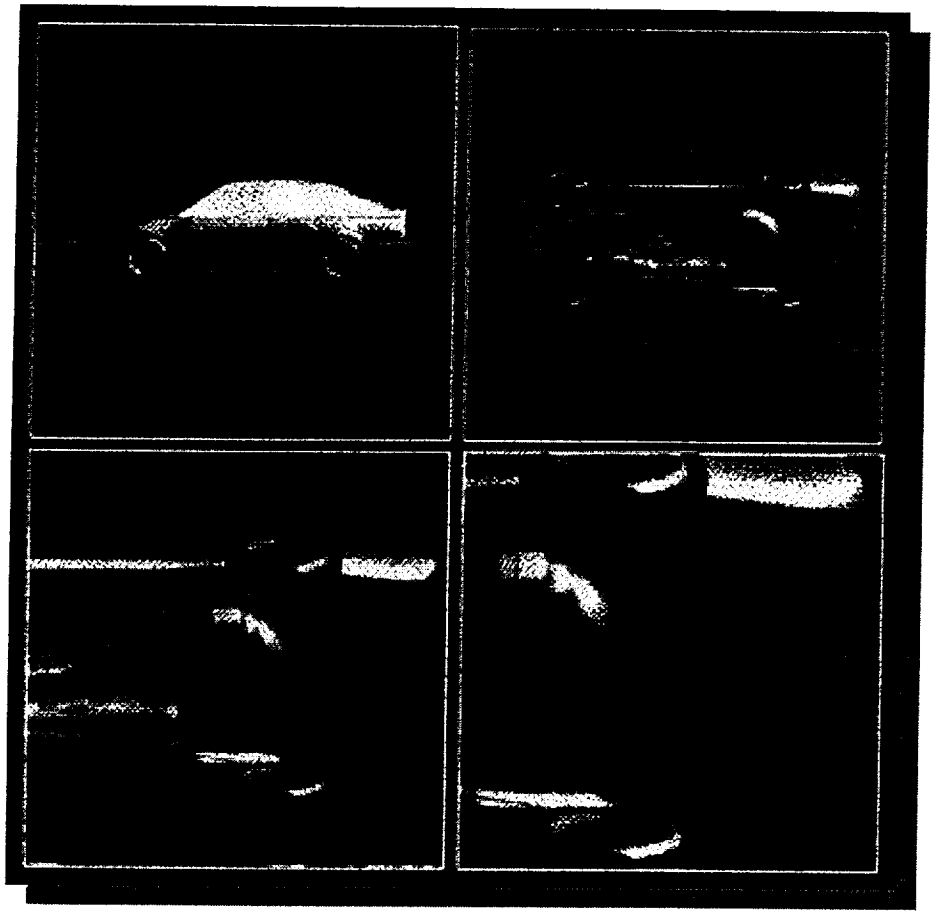


Figure 21.22: The explicit method is used to compute tangent curves linking critical values for a 3D vector field.

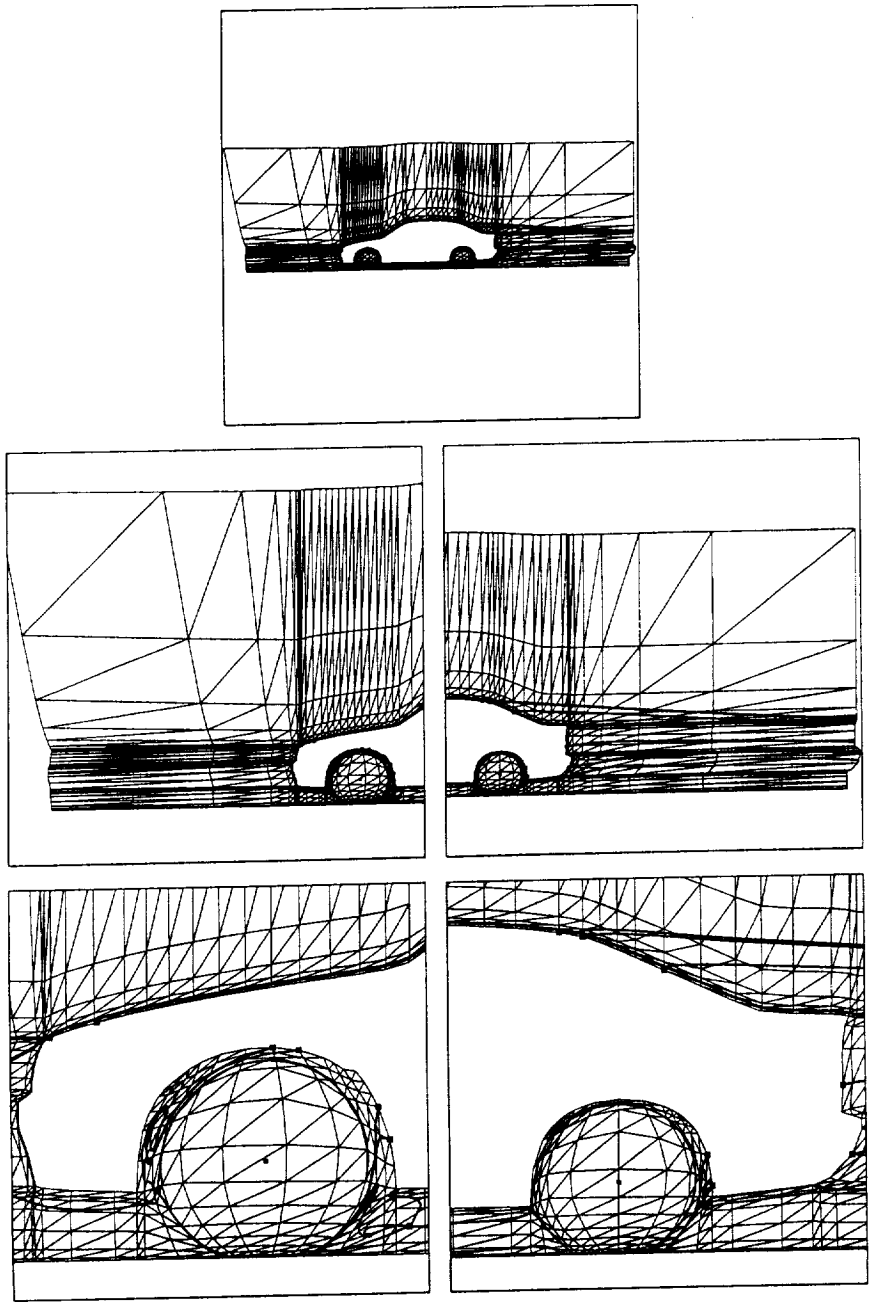


Figure 21.23: One slice of the data of Figure 21.22 showing the topological graph.

3. Because of the general nature of the approach of the methods covered here, they can be extended to any domain which consists of a collection of triangles. This includes manifolds of arbitrary topological type if a triangulated approximation of the domain is acceptable.

4. The computation of critical points is greatly simplified. Critical points are crucial to topological methods. If the vector field which has been defined by a particular cell and extended to the entire domain of E^2 has a critical value, then it can be computed by solving a linear system of equations. A solution to the linear system of equations is a "real" critical value for the piecewise defined vector field if it lies within the cell, otherwise it is not. Normally, the computation of critical values requires a rather tedious computation involving testing whether or not a cell might have a critical value, followed by the solution to a nonlinear system of equations by Newton's methods or some other iterative scheme. See [17], for example.

5. The last advantage we mentioned is that the present method does all of its computations in physical coordinates as opposed to computational coordinates. To some this might initially appear to be a disadvantage since computational coordinates are introduced for the specific purpose of their namesake. But if the domain is triangulated and linear variation is assumed over each triangle, we not only gain a method that is affine invariant, but there is no need to map the data to computational coordinates, solve the problem and map the solution back; we simply compute the solution directly in physical space.

6. We first mentioned the basic ideas of these methods and reported on some preliminary results at a presentation given in 1991 at the first Dagstuhl Seminar on Scientific Visualization. At this time, Nelson Max pointed out that he had mentioned the idea in an earlier paper [29]. Sawada [42] has also mentioned the idea of using explicit representations. We presented some preliminary results on the 3D extension of the methods covered here at a technology assessment workshop held the Summer of 1993 in Darmstadt, Germany. See [40] for the proceedings of this workshop. Figure 21.22 has previously appeared in [34].

Acknowledgments

This research was supported by the National Aeronautical and Space Administration under NASA-Ames Grant, NAG 2-990. Partial support was also provided by the North Atlantic Treaty Organization under grant RG0097/88. The data for the example of Figures 21.22 and 21.23 was provided by Yasuo Nakajima of Nissan, Japan. The data for the example of Figure 21.21 was provided by Roger Crawtis and Nelson Max of Lawrence Livermore National Laboratory.

Bibliography

[1] F.H. Bertrand and P.A. Tanguy, "Graphical Representation of Two-Dimensional Fluid Flow by Stream Vectors," *Communications in Applied Numerical Methods*, Vol. 4, 1988, pp. 213–217.

[2] J. Buckmaster, "Perturbation Technique for the Study of Three-Dimensional Separation," *Phys. Fluids*, Vol. 15, 1972, pp. 2106–2113.

[3] P.G. Buning, "Numerical Algorithms in CFD Post-Processing," *von Karman Institute for Fluid Dynamics*, Lecture Series 1989-07.

[4] B.J. Cantwell, "Similarity Transformations of the Two-Dimensional Unsteady Stream Function Equations," *J. Fluid Mech.*, Vol. 85, 1978, pp. 257–271.

[5] M.S. Chong, A.E. Perry and B.J. Cantwell, "A General Classification of Three-Dimensional Flow Fields," *Physics of Fluids A*, Vol. 2, No. 5, 1990, pp. 765–777.

[6] D. Darmofal and R. Haimes, "An Analysis of 3D Particle Path Integration Algorithms," *Proc. 1995 AIAA CFD Meeting*, San Diego, Calif., 1995.

[7] A. Davey, "Boundary Layer Flow at a Point of Attachment," *J. Fluid Mech.*, Vol. 10, 1961, pp. 593–610.

[8] R. Denzer, "Application of Visualization in Environmental Protection," *Focus on Scientific Visualization*, H. Hagen, M. Muller and G.M. Nielson, editors, Springer, 1993, pp. 73–82.

[9] R.R. Dickinsion, "Interactive Analysis of the Topology of 4D Vector Fields," *IBM Journal of Research and Development*, Vol. 35, No. 1/2, 1991, pp. 59–66.

[10] D.S. Ebert and R.E. Parent, "Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-Buffer Techniques," *Computer Graphics*, Vol. 24, No. 4, 1990, pp. 357–366.

[11] M. Fruehauf, "Combining Volume Rendering with Line and Surface Rendering," *Eurographics '91*, F.H. Post and W. Barth, editors, North Holland, 1991, pp. 21–32.

[12] R.S. Gallagher, "Span Filtering: An Optimization Scheme for Volume Visualization of Large Finite Element Models," *Proceedings of Visualization '91*, G.M. Nielson and L. Rosenblum, editors, IEEE Computer Society Press, 1991, pp. 68–75.

[13] R.S. Gallagher and J.C. Nagtegaal, "An Efficient 3-D Visualization Technique for Finite Element Models and Other Coarse Volumes," *Computer Graphics*, Vol. 23, No. 3, 1989, pp. 185–194.

[14] M. Geiben and M. Rumpf, "Visualization of Finite Elements and Tools for Numerical Analysis," *Advances in Scientific Visualization*, F.H. Post and A.J.S. Hin, editors, Springer, 1992.

[15] R.B. Haber and D.A. McNabb, "Visualization Idioms: A Conceptual Model for Scientific Visualization Systems," *Visualization in Scientific Computing*, G.M. Nielson, L. Rosenblum and B. Shriver, editors, IEEE Computer Society Press, 1990, pp. 74–92.

[16] B. Hamann, D. Wu, and R. Moorhead, "Flow Visualization with Surface Particles," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 3, Sep. 1995, pp. 210–217.

[17] J. Helman, and L. Hesselink, "Representation and Display of Vector Field Topology in Fluid Flow Data Sets," *IEEE Computer*, Vol. 22, No. 8, 1989, pp. 27–36.

[18] J. Helman and L. Hesselink, "Surface Representation of Two- and Three-Dimensional Fluid Flow Topology," *Proceedings Visualization '90*, A. Kaufman, editor, IEEE Computer Society Press, 1990, pp. 6–13.

[19] J. Helman and L. Hesselink, "Visualizing Vector Field Topology in Fluid Flows," *IEEE Computer Graphics and Applications*, Vol. 11, No. 3, pp. 36–46.

[20] W. Hibbard and D. Santek, "Visualizing Large Data Sets in the Earth Sciences," *IEEE Computer*, Vol. 22, No. 8, pp. 53–57.

[21] S. Hultquist, "Interactive Numeric Flow Visualization Using Stream Surfaces," *Computer Systems in Engineering*, Vol. 1, Nos. 2–4, 1990, pp. 349–353.

[22] I.-H. Jung, *Topological Visualizing Method of Vector Fields in Fluid Data Sets*, M.S. thesis, Arizona State University, Department of Computer Science and Engineering, June 1993.

[23] J.C.R. Hunt, C.J. Abell, J.A. Peterka and H. Woo, "Kinematical Studies of the Flows Around Free or Surface-Mounted Obstacles; Applying Topology to Flow Visualization," *J. Fluid Mech.*, Vol. 86, 1978, pp. 179–200.

[24] D.N. Kenwright, *Dual Stream Function Methods for Generating Three-Dimensional Stream Lines*, Ph.D. thesis, University of Auckland, Department of Mechanical Engineering, Aug. 1993.

[25] D.N. Kenwright and D.A. Lane, "Interactive Time-Dependent Particle Tracing Using Tetrahedral Decompostion," *IEEE Transaction on Visualization and Computer Graphics*, Vol. 2, No. 2, June 1996, pp. 120–129.

[26] M.J. Lighthill, "Attachment and Separation in Three-Dimensional Flow," *Laminar Boundary Layers*, L. Rosenhead, editor, Oxford University Press, 1963, pp. 72–82.

[27] R. Lohner and J. Ambrosiano, "A Vectorized Particle Tracer for Unstructured Grids," *J. Computational Physics*, Vol. 91, 1990, pp. 22–31.

[28] K.-L. Ma and P.J. Smith, "Cloud Tracing in Convection-Diffusion Systems," *Proceedings of Visualization '93*, G. Nielson and L. Rosenblum, editors, IEEE Computer Society Press, 1993, pp. 253–260.

[29] N. Max, *Private Communication*, 1991.

[30] W. Merzkirch, *Flow Visualization*, Academic Press, 1987.

[31] C. Moler and C. Van Loan, "Nineteen Dubious Ways to Compute the Exponential of a Matrix," *SIAM Review*, Vol 20, No. 4, Oct. 1978, pp. 801–836.

[32] G.M. Nielson, "Modeling and Visualizing Volumetric and Surface-on-Surface Data," *Focus on Scientific Visualization*, H. Hagen, M. Muller and G.M. Nielson, editors, Springer, 1993, pp. 191–240.

[33] G.M. Nielson, "Tools for Triangulation and Tetrahedrization," Chapter 20 in this volume.

[34] G.M. Nielson and A. Kaufman, "Visualization Graduates," *Computer Graphics and Applications*, Vol. 14, No. 5, Sep. 1994, pp. 17–18.

[35] T.V. Pathomas, J.A. Schiavone and B. Julesz, "Applications of Computer Graphics to The Visualization of Meteorological Data," *Computer Graphics*, Vol. 22, No. 4, pp. 327–335.

[36] A.E. Perry and B.D. Fairlie, "Critical Points in Flow Patterns," *Adv. Geophys.*, Vol. 18B, 1974, pp. 299–315.

[37] A.E. Perry and D.K.M. Tan, "Simple Three-Dimensional Motions in Coflowing Jets and Wakes," *J. Fluid Mech.*, Vol. 141, 1984, pp. 197–231.

[38] A.E. Perry and M.S. Chang, "A Description of Eddying Motions and Flow Patterns Using Critical-Point Concepts," *Ann. Rev. Fluid Mech.*, Vol. 19, 1987, pp. 125–155.

[39] F.H. Post and T. van Walsum, "Fluid Flow Visualization," *Focus on Scientific Visualization*, H. Hagen, M. Muller and G.M. Nielson, editors, Springer, 1993, pp. 1–40.

[40] L. Rosenblum et al., editors, *Scientific Visualization: Advances and Challenges*, Academic Press and IEEE Computer Society Press, 1994.

[41] A. Sadarjoen, T. van Walsum, A.J.S. Hin, and F.H. Post, "Particle Tracing Algorithms for Curvilinear Grids," *Proceedings Fifth Eurographics Workshop on Visualization in Scientific Computing*, Rostock, Germany, May 1994.

[42] K. Sawada, "Visualization of Unsteady Vortex Motion in the Flow over a Delta Wing," *Proc. of the 5th Int. Symp. on Computational Fluid Dynamics*, Sendai, Vol. III, 1993, ISCFD.

[43] W.J. Schroeder, C.R. Volpe and W.E. Lorensen, "The Stream Polygon: A Technique for 3D Vector Field Visualization," *Proceedings Visualization '91*, L. Rosenblum and G.M. Nielson, editors, IEEE Computer Society Press, 1991, pp. 126–132.

[44] N. Srinivasan, *An Efficient Method for the Computation of Tangent Curves*, M.S thesis, Arizona State University, Department of Computer Science and Engineering, May 1995.

[45] J. Stolk and J.J. van Wijk, "Surface-Particles for 3D Flow Visualization," *Advances in Scientific Visualization*, Springer, 1992.

[46] T. Strid, A. Rizzi and J. Oppelstrup, "Development and Use of some Flow Visualization Algorithms," *von Karman Institute for Fluid Dynamics*, Lecture Series 1989-07.

[47] M. Tobak and D.J. Peake, "Topology of Two-Dimensional and Three-Dimensional Separated Flows," *AIAA 12th Fluid and Plasma Dynamics Conference*, Williamsburg, Va., July 23–25, 1979.

[48] M. Tobak and D.J. Peake, "Topology of Three-Dimensional Separated Flows," *Ann. Rev. Fluid Mech.*, Vol. 14, 1982, pp. 61–85.

[49] S.K. Ueng, K. Sikorski, and K.-L. Ma, "Efficient Streamline, Streamribbon, and Streamtube Constructions on Unstructured Grids," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 2, 1996, pp. 100–110.

[50] J.J. van Wijk, "A Raster Graphics Approach to Flow Visualization," *Eurographics '90*, D.A. Duce and C.E. Vandoni, editors, North Holland, pp. 251–259.

[51] J.J. van Wijk, "Spot Noise—Texture Synthesis for Data Visualization," *Computer Graphics*, Vol. 25, No. 4, 1991, pp. 309–318.

[52] W.J. Yang, editor, *Handbook of Flow Visualizaton*, Hemisphere Publishing, 1989.

2

Chapter 20

Tools for Triangulations and Tetrahedrizations and Constructing Functions Defined over Them

Gregory M. Nielson

20.1 Introduction

This chapter is about triangulations and tetrahedrizations and functions defined over them. The original and main goal was to provide some information about tetrahedra and tetrahedrizations and functions defined over them, but it was quickly realized that many of these topics are easier to describe and understand with some background on their two-dimensional analogs. Therefore, it was decided to also include material on triangulations. While some of the material exists elsewhere in the literature, much is new and appears here for the first time. The intended purpose for this chapter is to serve as a survey/tutorial in the area of data modeling and visualization. As data modeling and visualization becomes more sophisticated in its application domains and begins to deal with data sets that are more complex than Cartesian grids, there will be the need for tools to deal with these data sets. We feel that the tools and techniques covered here are very basic and will prove to be useful in a variety of contexts in data visualization.

Now we have some comments about the organization of this chapter. While tetrahedrizations are the goal, researchers have dealt with triangulations for a much longer period of time than tetrahedrizations and so triangulations and related matters are much better understood. The next section is a survey of triangulations and related matters of interest in modeling and visualization. The following section is on tetrahedrizations and we attempt to follow the same flow of information as in the section on triangulations as well as possible. We use the phrase "as well as possible" because some aspects of triangulations do not generalize to tetrahedrization and some facts known about triangulations and triangu-

lar domains are yet to be known about tetrahedrizations and tetrahedral domains. On the other hand, there are topics of interest to tetrahedrization which have no 2D counterpart of interest—for example, visibility sorting for tetrahedrizations. The outline of this chapter is very simple. In Section 20.2 we go through a list of topics on triangulations and triangular domains and then in Section 20.3 we repeat these topics with reference to tetrahedrizations and tetrahedral domains.

20.2 Triangulations

20.2.1 Basics

Definitions, Data Structures, and Formulas for Triangulations

In order to avoid any possible confusion and problems later, it is usually best to be a little precise and formal about the definition of a triangulation. We start with a collection of points in the plane, P = $\{p_i = (x_i, y_i), i = 1, \ldots, N\}$ and a domain of interest, D, which contains all of the points of P. We assume that the boundary of D is a simple (does not intersect itself), closed polygon. Often D is the convex hull of P, but in general, it need not be convex. In fact the boundary does not have to be a single polygon so that the domain is not even simply connected. (*Connected* means that there is a path joining any two points and *simply connected* means that the complement is connected.) Roughly speaking, a triangulation is a decomposition of D into a collection of triangles which are formed from vertices of P. Since we are eventually interested in defining functions over D in a piecewise manner over each triangle, we must require that the triangles do not overlap so as not to have any ambiguities. Thus we require the collection of triangles of the triangulation to be mutually exclusive and collectively exhaustive. In order to continue this formalism to a precise definition, we need some additional notation. A single triangle with vertices p_i, p_j, and p_k is denoted by T_{ijk} and the list of triples which represents the triangulation is denoted by I_t. A triangle T_{ijk} is a closed 2D point set that includes its three edges which comprise its boundary. The interior of T_{ijk}, denoted by Int(T_{ijk}) is open and does not include the boundary. The edge joining p_i and p_j is denoted by e_{ij} and $N_e = \{ij : ijk \text{ in } I_t \text{ for some } k\}$ is used to refer to the collection of all edges. Formally, the definition of a triangulation requires the following:

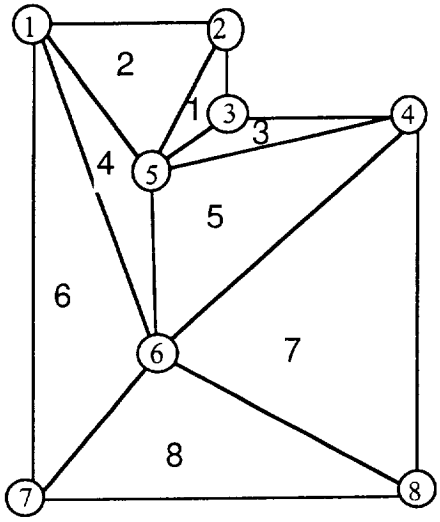
i) No triangle $T_{ijk}, ijk \in I_t$ is degenerate. That is, if $ijk \in I_t$ then p_i, p_j and p_k are not collinear.

ii) The interior of any two triangles do not intersect. That is, if $ijk \in I_t$ and $\alpha\beta\gamma \in I_t$ then Int(T_{ijk})$\cap$ Int($T_{\alpha\beta\gamma}$) $= \emptyset$.

iii) The boundary of two triangles can only intersect at a common edge.

iv) The union of all the triangles is the domain D $= \cup_{ijk \in I_t} T_{ijk}$.

Examples of valid triangulations are shown in Figure 20.1 and Figure 20.2. Note that the example of Figure 20.1 is not convex and that of 20.2 is not simply connected. Even though the diagrams of Figure 20.3 and Figure 20.4 look all right, the actual triangulations



I_t

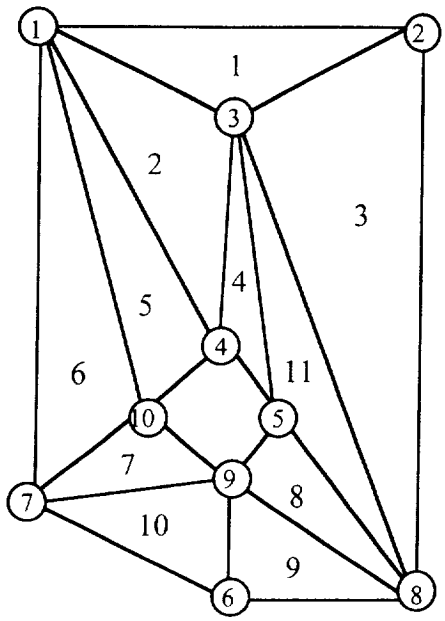
i	j	k
3	5	2
5	1	2
4	5	3
6	1	5
4	6	5
6	7	1
8	6	4
7	6	8

Figure 20.1: A triangulation of a nonconvex domain.

given by the corresponding I_t's do not represent valid triangulations. In the case of Figure 20.3 the triangle T_{465} is degenerate. Even if this triangle is eliminated, what remains is not a valid triangulation because condition iii) would then be violated since edge e_{46} contains p_5. This example would become a valid triangulation if the point p_5 were to be moved slightly to the right so as not to be on the edge e_{46}. The information of Figure 20.4 is not a valid triangulation because condition ii) is violated.

We now want to make some assertions about the possibility of triangulating a domain containing a collection of data points that is bounded by a simple, closed polygon. First we note that in the case that the domain contains no interior data points, it is always possible to form a triangulation. Just for the sake of interest, we mention two ways that this can be accomplished. The first way is based upon the fact that every simple closed polygon with more than three vertices can be split into two polygons. This leads to an algorithm that recursively splits each subpolygon until only triangles are left. The following argument which guarantees that each simple closed polygon has a diagonal has been discussed in [16]. A diagonal is an edge between two vertices that lies inside the polygon and does not intersect the polygon except at the endpoints.

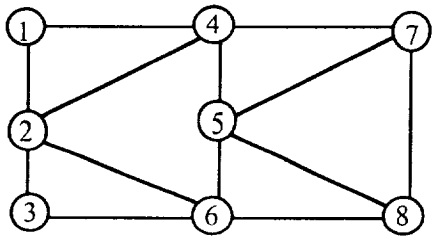
Splitting a polygon: Let b be the vertex with minimum x-coordinate and ab and bc be its two incident edges as is shown in Figure 20.5. If ac is not cut by the polygon, then ac is a diagonal. Otherwise there must be at least one polygon vertex inside T_{abc}. Let d be the vertex inside abc furthest from the line through a and c. Now edge bd cannot be cut by the polygon, since any edge intersecting bd must have one endpoint further from line ac. The second approach leads to an iterative algorithm. We first give a definition. A vertex, p_i, of a simple, closed polygon is called *protruding*, provided the following conditions hold:



I_t

i	j	k
3	1	2
4	1	3
2	8	3
5	4	3
10	1	4
7	1	10
7	10	9
8	9	5
6	9	8
6	7	9
5	3	8

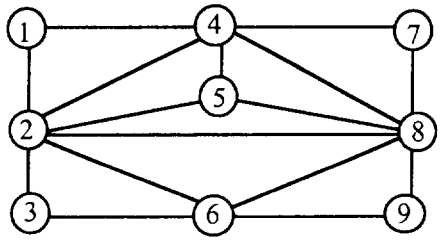
Figure 20.2: A triangulation of a domain which is not simply connected.



I_t

i	j	k
1	4	2
2	4	6
2	6	3
4	6	5
4	7	5
5	7	8
5	8	6

Figure 20.3: Not a valid triangulation.



I_t

i	j	k
1	4	2
2	4	5
2	5	8
4	8	5
4	7	8
2	4	8
2	6	3
2	8	6
8	9	6

Figure 20.4: Not a valid triangulation.

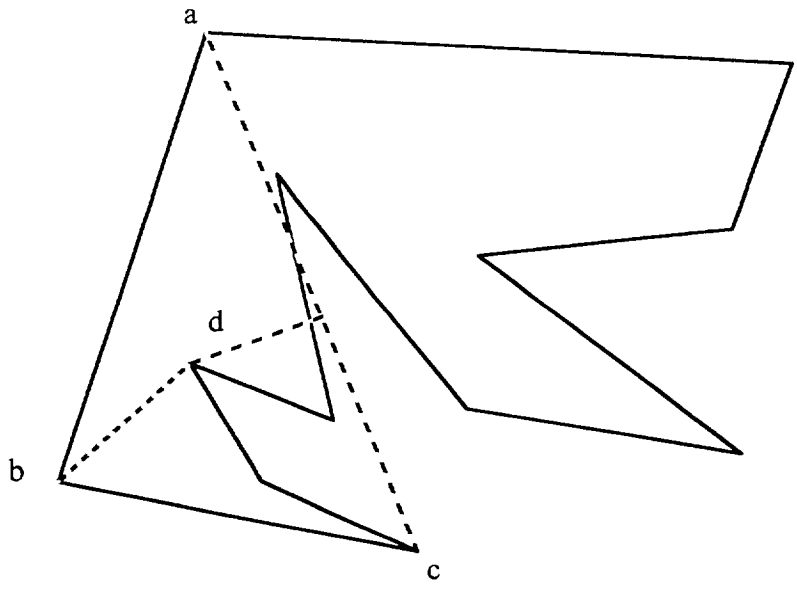


Figure 20.5: Any polygon with more than three vertices can be split.

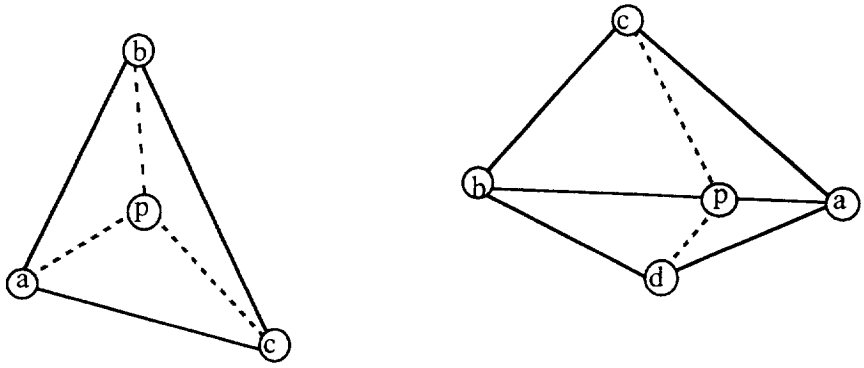


Figure 20.6: Insertion of an interior point.

i) The interior angle Θ_i, between the edges, $p_{i-1}p_i$ and p_ip_{i+1} is less than or equal to π. (Cyclic notation is used here so that $p_{N+1} = p_1$.)

ii) The triangle $T_{i-1,i,i+1}$ contains no other vertices of the polygon than p_{i-1}, p_i or p_{i+1}.

iii) The interior of $T_{i-1,i,i+1}$ is contained in the interior of D.

It is an easy matter to prove that every simple, closed polygon has at least one protruding vertex. (The proof is left to the reader. Some people call them ears and so there must be two of them!) We can triangulate the polygon-bounded domain by successively removing protruding vertices. This approach to triangulating the region bounded by a simple closed polygon is called the "boundary stripping algorithm." It is easy to implement, but in a theoretical sense, it is not competitive with other algorithms (see, for example, the papers of Narkhede and Manocha [175] and Fournier and Montuno [94], among others).

Once the boundary of D has been triangulated, it is a relatively simple matter to build a triangulation including the interior points. This can be done by simply inserting them sequentially in a manner which we now describe.

Insertion of an interior point: If the point to be inserted, p, lies in the interior of the triangle T_{abc}, we replace T_{abc} with the three triangles: $T_{abp}, T_{bcp}, T_{cap}$. If p lies on an edge shared by T_{abc} and T_{bad}, then we replace the two triangles T_{abc} and T_{bad} with the four triangles $T_{bcp}, T_{dbp}, T_{pca}, T_{pad}$. These two cases are illustrated in Figure 20.6.

It is also possible to generalize the insertion idea to include an edge. Once we are armed with this capability, we know that we can triangulate any polygon-bounded domain: simply connected or multiply connected (that is, with holes).

Insertion of an interior edge: Assume that the one endpoint, p, lies in the triangle T_{abc} and that the other endpoint, q, lies in the triangle T_{xyz}. See Figure 20.7. Collect all of the triangles from T_{abc} to T_{xyz} which are intersected by edge pq and form a region R

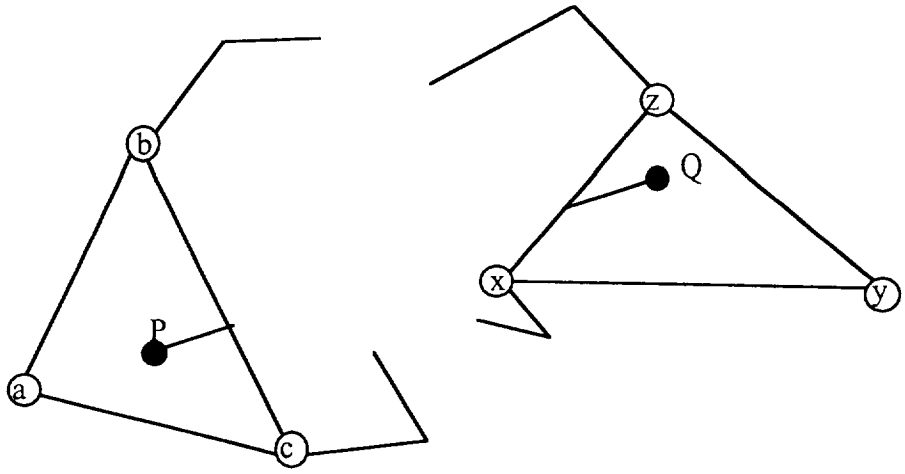
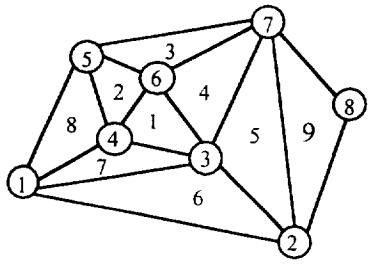


Figure 20.7: Insertion of an interior edge.

with polygon boundary D. We can split D with polygon $dpqw$, where d is the vertex of T_{abc} not on the edge common with the other triangles whose union is R, and w is the analogous vertex of T_{xyz}. Now we know that each of these two domains can be triangulated. The union of these two triangulations, which each contain the edge pq, can replace the previous triangulation of D.

In addition to I_t, which represents the triangulation, it is often worthwhile to generate and maintain some auxiliary information about the neighbors of each triangle. This information is useful for traversal algorithms and evaluation algorithms which have a searching component that determines the particular triangle containing a point where a function defined piecewise over the triangulation is to be evaluated. One very common and particularly useful data structure is that which is illustrated in Figure 20.8. The first three columns contain the data of I_t, with the additional constraint that in reading from left to right (cyclically), the vertices of each triangle are traversed in a clockwise order. The next three columns contain the indices of the triangles which are neighbors to this triangle. The character ϕ indicates that the triangle has an edge that is part of the boundary of D. The entries of these three columns are also in a special order. The fourth column contains the index of the triangle which shares the common edge with vertex indices specified in the second and third columns. Similar relationships hold for the fifth and sixth columns. The information represented by this data structure is called a "triangular grid." The neighborhood information contained in the last three columns does not contain any "new" information over that of I_t, but it is often the case (and this depends, of course, on the application) that it is useful data which is worth generating a priori.

Another data structure for representing a triangulation which is useful for some applications is illustrated by the example shown in Figure 20.9, which represents the same triangulation as that of Figure 20.8. Here, for each vertex, a list of all vertices which are joined by an edge of the triangulation is given. This list is given in counterclockwise order



Triangles			Neighbors		
p_i	p_j	p_k	N_{jk}	N_{ki}	N_{ij}
3	4	6	2	4	7
4	5	6	3	1	8
6	5	7	ϕ	4	2
3	6	7	3	5	1
2	3	7	4	9	6
2	1	3	7	5	ϕ
1	4	3	1	6	8
1	5	4	2	7	ϕ
2	7	8	ϕ	ϕ	5

Figure 20.8: An example that defines a *triangular grid structure.*

Vertex	Joining Vertices
1	2, 3, 4, 5
2	8, 7, 3, 1
3	1, 2, 7, 6, 4
4	3, 6, 5, 1
5	1, 4, 6, 7
6	3, 7, 5, 4
7	6, 3, 2, 8, 5
8	2, 7

Figure 20.9: The data that defines the data point contiguity list.

around each vertex. This is called the *data point contiguity list.* We mention this particular data structure because of its convenience for dealing with the optimal Delaunay triangulation discussed in the next section. Also, it is very useful for computing the parameters of the Minimum Norm Network method [179], which is one of the most effective C^1 interpolation methods for scattered data.

Even though there are a number of possible triangulations for any given domain D, the number of triangles is fixed once the boundary has been specified. More precisely, if N_b represents the number of vertices on the boundary and N_i the number of interior vertices so that $N = N_b + N_i$, then the following formulas hold:

$$N_t = 2N_i + N_b - 2$$

and

$$N_e = 3N_i + 2N_b - 3,$$

where N_t is the total number of triangles and N_e is the total number of edges. The importance of these formulas (not so much what the values in the formulas are, but more the fact

that some fixed formula holds) will show up in the next section. If we let M_i represent the number of points joining to p_i then it is easy to see that

$$\sum_{i=1}^{N} M_i = 2N_e$$

and so we have that the "average valence" of a point is given by

$$\bar{M} = \frac{\sum_{i=1}^{N} M_i}{N_i + N_b} = 6 - 2\frac{N_b + 3}{N}$$

which is approximately 6. For a sphere (or any domain homeomorphic to a sphere) we have no boundary points and so $N = N_i$ and the analogous formulas are

$$N_t = 2(N-1), \quad N_e = 3(N-1), \quad \bar{M} = \frac{6}{N}(N-1).$$

Some Special Triangulations

One of the simplest triangulations results from splitting the rectangles of a Cartesian grid. A Cartesian grid involves two monotonically increasing sequences, $x_i, i = 1, \ldots, n$ and $y_j, j = 1, \ldots, m$. The grid points have coordinates (x_i, y_j) and these points mark out a cellular decomposition of the domain consisting of rectangles. See Figure 20.10. Forming an edge with one of the diagonals of these rectangular cells leads to a triangulation of the domain. In Figure 20.11 is shown a triangulation where a consistent choice for the diagonal is made. In Figure 20.12 is shown a triangulation with mixed choices for the diagonals. In some applications where dependent ordinate values are known, it is possible to base the choice of the diagonal upon some criteria such as minimum jump in normal vector (see Section 20.2.4) or whether or not the diagonal vertices are separated or connected based upon the hyperbolic contours at the mean value (see the *asymptotic decider* criteria discussed in [186]). In general for this type of triangulation which results from a Cartesian grid, it is not necessary to maintain the triangular grid structure (see Figure 20.8) as this information can be directly inferred from the natural labeling of $p_{ij} = (x_i, y_j)$. Only the information which indicates which diagonal is selected needs to be made available.

We now want to discuss some special triangulations which result from curvilinear grids. A curvilinear grid is specified with two "geometry arrays" $(x_{ij}, y_{ij}), i = 1, \ldots, M; j = 1, \ldots, N$. A cell C_{ij} consists of the quadrilateral with the boundary delineated by (x_{ij}, y_{ij}) to (x_{i+1j}, y_{i+1j}) to (x_{ij+1}, y_{ij+1}) back to (x_{ij}, y_{ij}). It is assumed that these four points form a simple (nonintersecting) polygon so that the quadrilateral is actually well-defined. This condition obviously puts some geometric constraints on the values of the geometry arrays that specify a curvilinear grid.

An example of a curvilinear grid is shown in Figure 20.13. In this case the cell C_{73} degenerates to a triangle because (X_{83}, Y_{83}) and (X_{84}, Y_{84}) are the same point and the cell C_{83} degenerates to an edge because, in addition, (X_{93}, Y_{93}) and (X_{94}, Y_{94}) are the same point. The cells $C_{33}, C_{43}, C_{53}, C_{63}$, and C_{73} have been removed from the domain creating the hole in the interior.

The domain (the union of all of its cells) can be triangulated simply by triangulating each of the cells, by choosing a diagonal to an edge of the triangulation. An example related

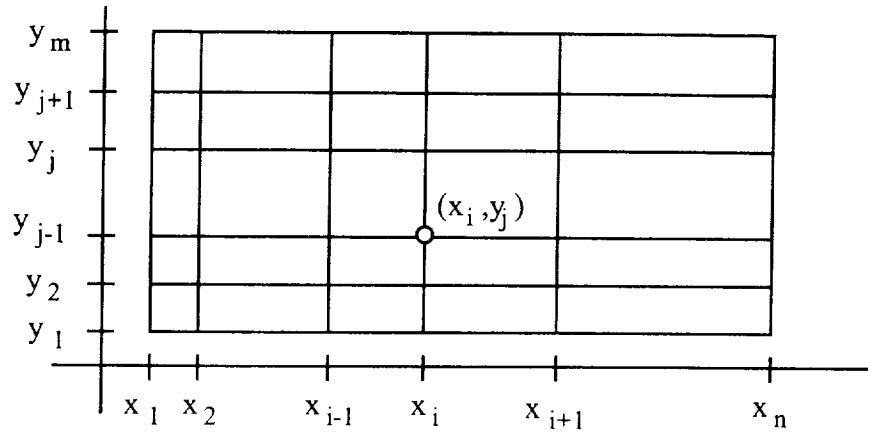


Figure 20.10: Cartesian Grid.

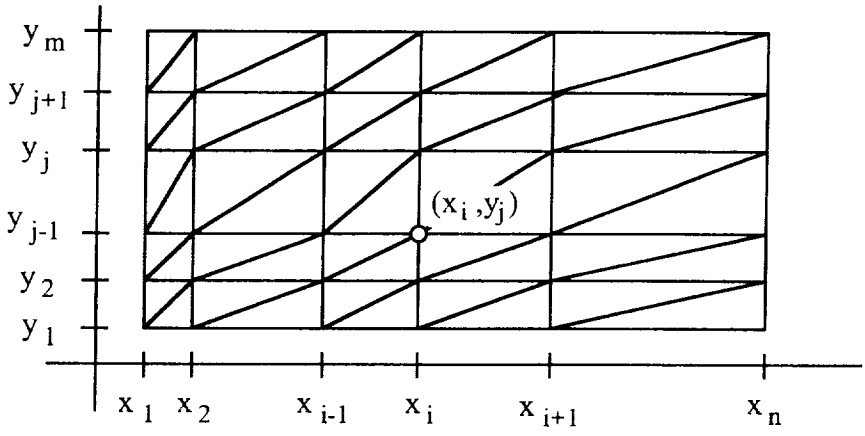


Figure 20.11: Triangulation from Cartesian grid with uniform diagonal choice.

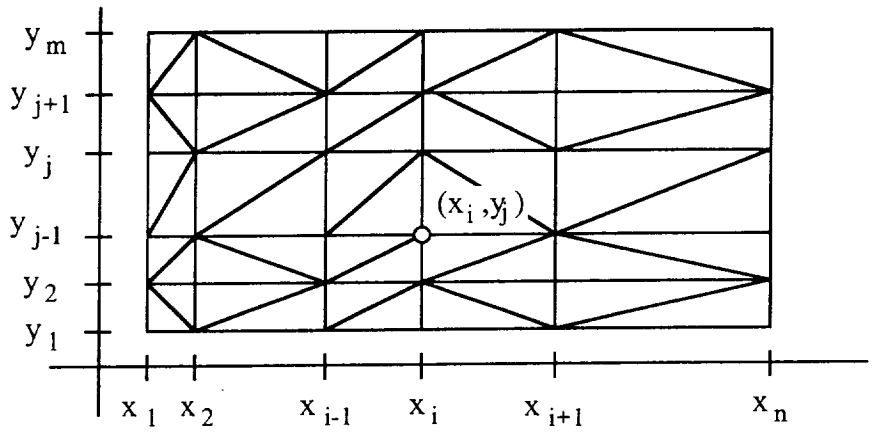


Figure 20.12: Triangulation from Cartesian grid with mixed diagonals.

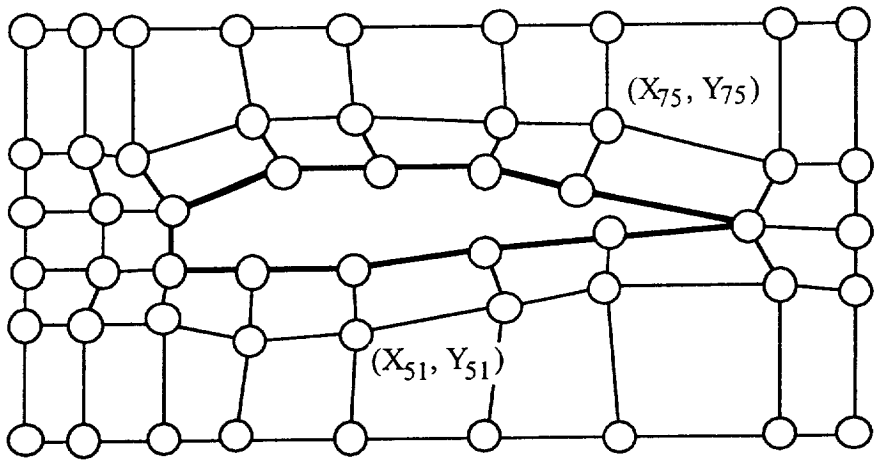


Figure 20.13: An example of a curvilinear grid.

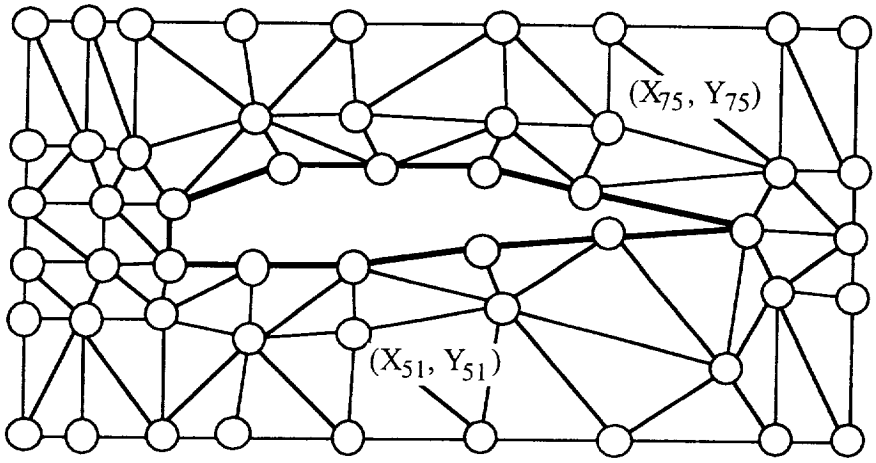


Figure 20.14: Triangulation resulting from curvilinear grid.

to the grid of Figure 20.13 is shown in Figure 20.14. Here we have modified the grid by moving the point (X_{72}, Y_{72}) a little. This serves to point out that if the cell is not convex, then there may be only one choice for the diagonal.

We now discuss some special triangulations obtained by subdividing an existing triangulation. We briefly mention a couple of possibilities. The first is based upon inserting an additional point into the interior of an existing triangle and thereby forming three new triangles. This is illustrated in Figure 20.15. This particular type of subdivision is sometimes referred to as the Clough-Tocher split because of its association with a very well known finite element shape function defined over a triangular domain.

Another way to subdivide an existing triangulation is to insert a new point on an existing edge and split the two triangles (unless the edge is on the boundary) which share this edge. If all edges are split simultaneously we obtain yet another triangulation where each previous triangle is replaced by four new ones. Two different ways for forming triangles from these points is shown in Figure 20.16 and Figure 20.17, respectively. These types of subdivision are particularly interesting due to the nested properties of function spaces which are defined in a piecewise manner over the embedded subdivisions. This can lead to wavelets and their related multiresolution analysis. For the efficient application of these triangulations, it is important to have a method of labeling the triangles which allows an efficient algorithm for finding the labels of all neighbors of a triangle. The labeling scheme illustrated in Figure 20.17 has these properties. We call it the *divide and flip* scheme and have found it to be very useful for implementations. It is related to the spherical quadtrees discussed by Fekete [85]. The first step of the subdivision applied to the triangulation of Figure 20.8 is shown in Figure 20.18.

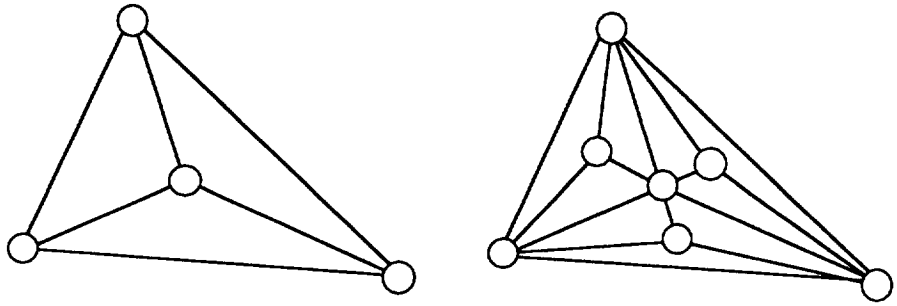


Figure 20.15: Subdivision by inserting a new point that is interior to an existing triangle.

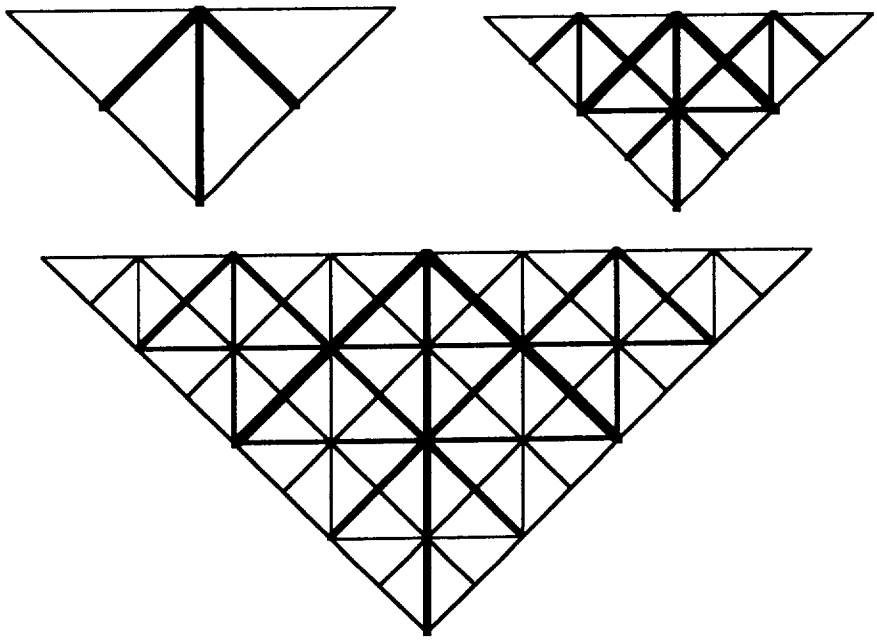


Figure 20.16: Nested subdivision triangulation.

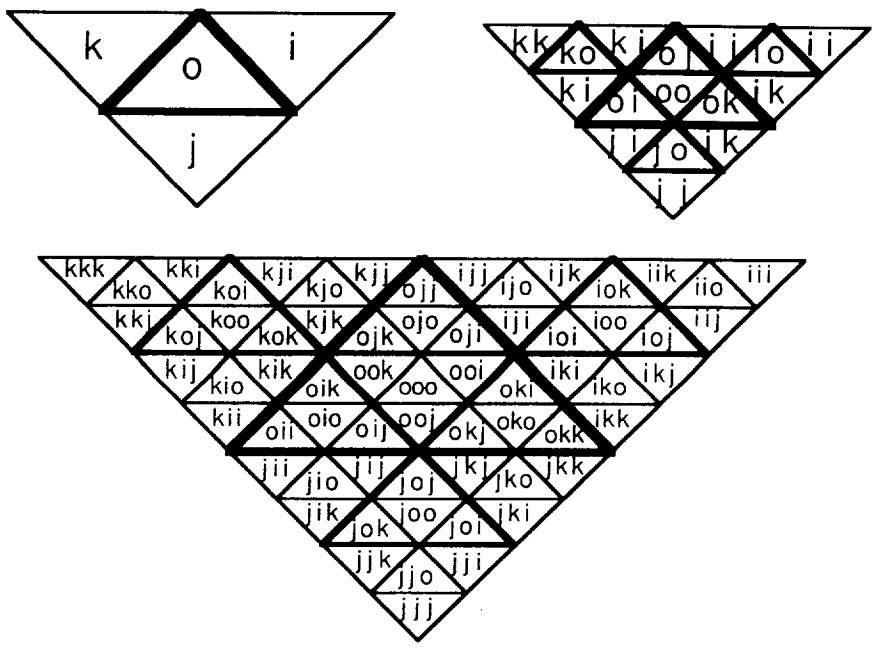


Figure 20.17: The *divide and flip* labeling scheme for a nested subdivision triangulation.

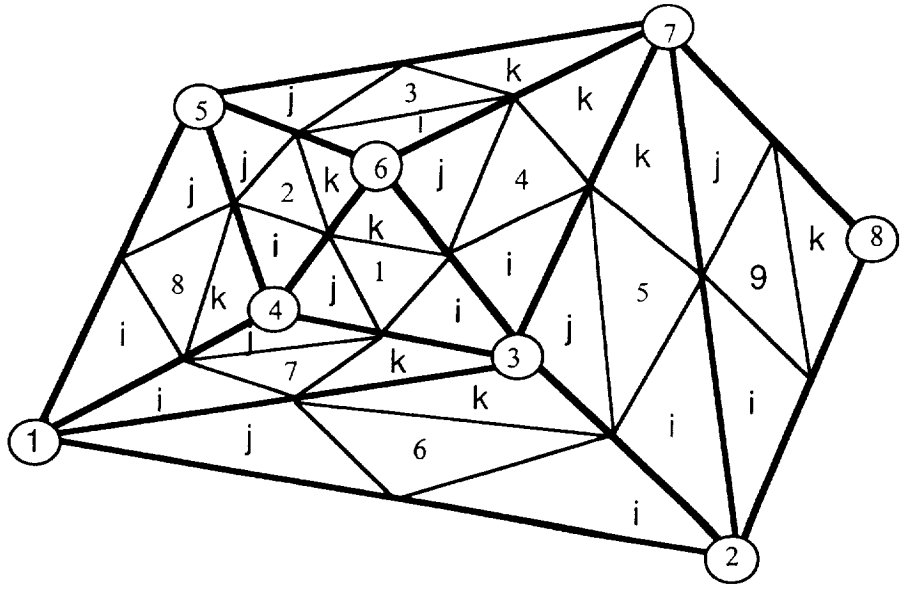


Figure 20.18: A triangulation obtained by splitting each edge of an existing triangulation and forming triangles as indicated in Figure 20.17.

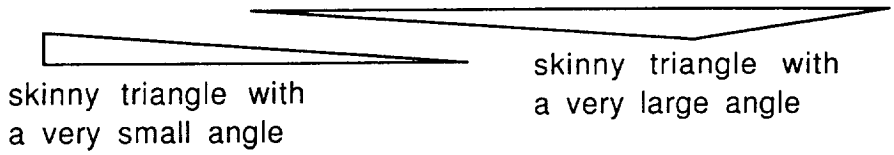


Figure 20.19: Examples of poorly shaped triangles.

20.2.2 Optimal Triangulations

Types and Characterizations

There are many possible triangulations of a given, polygon-bounded domain D. For some applications (but not all) it is desirable to avoid poorly shaped triangles. See Figure 20.19. These are triangles with very large angles or ones with very small angles. This gives rise to two types of optimal triangulations which have been discussed quite widely: the MaxMin and MinMax. Both of these optimal triangulations have a similar method of characterization. Associated with each triangulation there is a vector with N_t entries representing either the largest or smallest angle of each triangle. The entries of each vector are ordered and then a lexicographic ordering of the vectors is used to impose an ordering on the set of all triangulations. In the case of the MinMax criterion, A_i is the largest angle of a triangle and the entries of each vector, A_t, are ordered so that

$$A_t = (A_1, A_2, \ldots, A_{N_t}), A_i \geq A_j, i < j.$$

The smallest of these vectors, based on their lexicographic ordering, associates with the optimal triangulation. In the case of the MaxMin criteria, a_i is the smallest angle and the entries of each vector are ordered the other way so that

$$a_t = (a_1, a_2, \ldots, a_{n_t}), a_i \leq a_j, i < j.$$

The largest of these vectors represents the optimal triangulation in the MaxMin sense. In Figure 20.20, six data points are shown which have a total of ten possible triangulations, which are shown in Figure 20.21. The associated vectors for the MinMax criterion are

$$\begin{aligned}
A_{T_0} &= (2.84, 2.36, 1.99, 1.77, 1.57) \\
A_{T_1} &= (2.98, 2.84, 1.99, 1.91, 1.57) \\
A_{T_2} &= (2.98, 2.42, 1.91, 1.88, 1.57) \\
A_{T_3} &= (2.84, 2.36, 2.32, 1.99, 1.40) \\
A_{T_4} &= (2.42, 2.36, 1.88, 1.77, 1.57) \\
A_{T_5} &= (2.98, 2.42, 1.95, 1.91, 1.27) \\
A_{T_6} &= (2.42, 2.36, 2.32, 1.88, 1.40) \\
A_{T_7} &= (2.42, 2.36, 2.32, 1.50, 1.50) \\
A_{T_8} &= (2.42, 2.36, 1.95, 1.74, 1.50) \\
A_{T_9} &= (2.42, 2.36, 1.95, 1.77, 1.27)
\end{aligned}$$

which we rearrange into decreasing order to obtain

$$\begin{aligned}
A_{\tau_1} &= (2.98, 2.84, 1.99, 1.91, 1.57) \\
A_{\tau_5} &= (2.98, 2.42, 1.95, 1.91, 1.27) \\
A_{\tau_2} &= (2.98, 2.42, 1.91, 1.88, 1.57) \\
A_{\tau_3} &= (2.84, 2.36, 2.32, 1.99, 1.40) \\
A_{\tau_0} &= (2.84, 2.36, 1.99, 1.77, 1.57) \\
A_{\tau_6} &= (2.42, 2.36, 2.32, 1.88, 1.40) \\
A_{\tau_7} &= (2.42, 2.36, 2.32, 1.50, 1.50) \\
A_{\tau_9} &= (2.42, 2.36, 1.95, 1.77, 1.27) \\
A_{\tau_8} &= (2.42, 2.36, 1.95, 1.74, 1.50) \\
A_{\tau_4} &= (2.42, 2.36, 1.88, 1.77, 1.57)
\end{aligned}$$

which implies the following ordering

$$\tau_4 < \tau_8 < \tau_9 < \tau_7 < \tau_6 < \tau_0 < \tau_3 < \tau_2 < \tau_5 < \tau_1$$

and so τ_4 is the optimal triangulation in the MinMax sense. On the other hand, the associated vectors for MaxMin criteria sorted in increasing order are

$$\begin{aligned}
a_{\tau_1} &= (0.02, 0.04, 0.35, 0.46, 0.50) \\
a_{\tau_2} &= (0.02, 0.11, 0.42, 0.46, 0.50) \\
a_{\tau_5} &= (0.02, 0.11, 0.50, 0.58, 0.88) \\
a_{\tau_3} &= (0.04, 0.14, 0.35, 0.37, 0.66) \\
a_{\tau_0} &= (0.04, 0.14, 0.35, 0.46, 0.62) \\
a_{\tau_6} &= (0.11, 0.14, 0.37, 0.42, 0.66) \\
a_{\tau_7} &= (0.11, 0.14, 0.37, 0.46, 0.70) \\
a_{\tau_4} &= (0.11, 0.14, 0.42, 0.46, 0.62) \\
a_{\tau_8} &= (0.11, 0.14, 0.57, 0.58, 0.70) \\
a_{\tau_9} &= (0.11, 0.14, 0.58, 0.62, 0.88)
\end{aligned}$$

which results in the following ordering

$$\tau_1 < \tau_2 < \tau_5 < \tau_3 < \tau_0 < \tau_6 < \tau_7 < \tau_4 < \tau_8 < \tau_9$$

and so τ_9 is the optimal triangulation in the case of the MaxMin criterion.

In the case where D is the convex hull of the points of P, there is an important relationship between the MaxMin triangulation and the Dirichlet tessellation. The Dirichlet tessellation is a partition of the plane into regions $\mathrm{R}_i, i = 1, \ldots, N$ called Thiessen regions. The Thiessen region R_k consists of all points in the plane whose closest point among $p_i, i = 1, \ldots, n$ is p_k. A Dirichlet tessellation is usually illustrated by drawing the boundaries of the Thiessen regions. The collection of these edges is sometimes referred to as the Voronoi diagram (see [252]). An example is shown in the left image of Figure 20.24. In the right image of Figure 20.24 is shown the MaxMin triangulation, which is also called the Delaunay triangulation. It is dual to the Dirichlet tessellation in that the edges of this optimal triangulation join vertices which share a common Thiessen region boundary. We have included the great circles in the left image of this figure so as to point out another important property of the Dirichlet tessellation and its companion Delaunay triangulation.

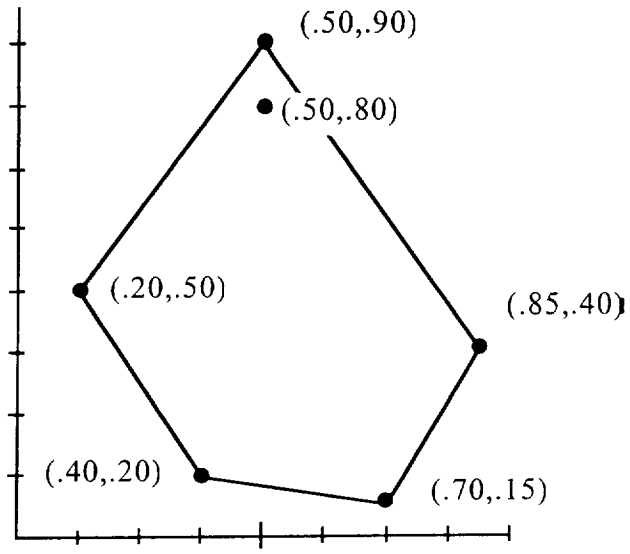


Figure 20.20: Six data points.

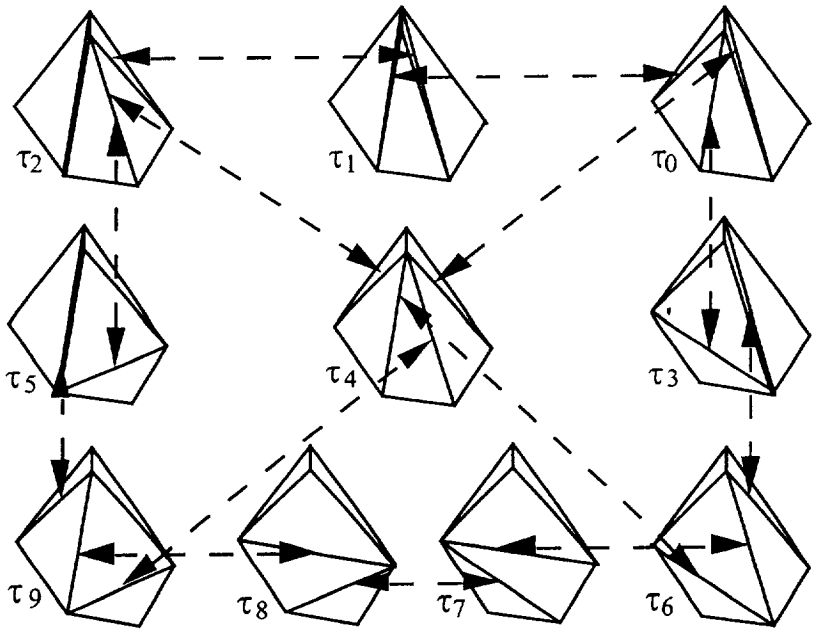


Figure 20.21: Ten triangulations of six data points.

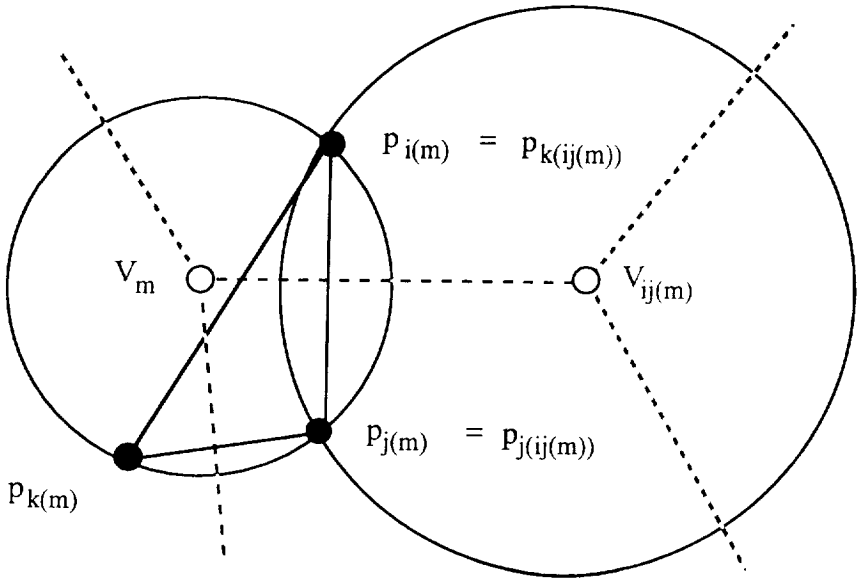


Figure 20.22: Drawing the Dirichlet tessellation from the triangular grid structure.

By definition, the edges of the Thiessen regions meet at triads (possibly more than three edges meet in some special, neutral/cyclic cases) which are equally distant to three points. These three points will form a triangle of the optimal triangulation and the great circle will not contain any other data points. See Figure 20.25.

We can be a little more formal about these properties if we introduce some notation. Recall that $I_t = \{(i(m), j(m), k(m)), m = 1, \ldots, N_t\}$ so that the three data points $p_{i(m)}, p_{j(m)}, p_{k(m)}$ will be the vertices of a triangle of the triangulation. We assume that the neighbor information of the triangular grid is given by three arrays $ij(m), jk(m)$, and $ki(m), m = 1, \ldots, N_t$. Let V_m be the point which is equidistant from $p_{i(m)}, p_{j(m)}$, and $p_{k(m)}$ and $C_m = \{p : \|p - V_m\| \leq \|V_m - p_{a(m)}\|, a = i, j \text{ or } k\}$ be the circumcircle (disk) for this triangle which has V_m as its center. The Delaunay triangulation is characterized by the fact that C_m does not contain any other data points $p_i, i = 1, \ldots, N$ other than $p_{i(m)}, p_{j(m)}$, and $p_{k(m)}$. The points V_m are the vertices of the Voronoi diagram. In order to draw the Voronoi diagram we simply start with some V_m and draw the edges to the three points that are joined to it, namely, $V_{ij(m)}, V_{jk(m)}$, and $V_{ki(m)}$. If any one of $ij(m), jk(m)$, or $ki(m)$ is zero (say $ij(m)$, indicating the edge joining $p_{i(m)}$ and $p_{j(m)}$ is on the boundary of the convex hull), then we draw the ray emanating from V_m in the direction perpendicular to the appropriate edge (which is $p_{i(m)}p_{j(m)}$ if $ij(m) = 0, p_{j(m)}p_{k(m)}$ if $jk(m) = 0$, and $p_{k(m)}p_{i(m)}$ if $ki(m) = 0$). If we go through the list of triangles and draw three edges for each V_m we will actually be drawing each edge (not each ray) twice. We can avoid this duplication by testing (for example) whether or not $m > ij(m), m > jk(m), m > ki(m)$ before we draw the corresponding edge.

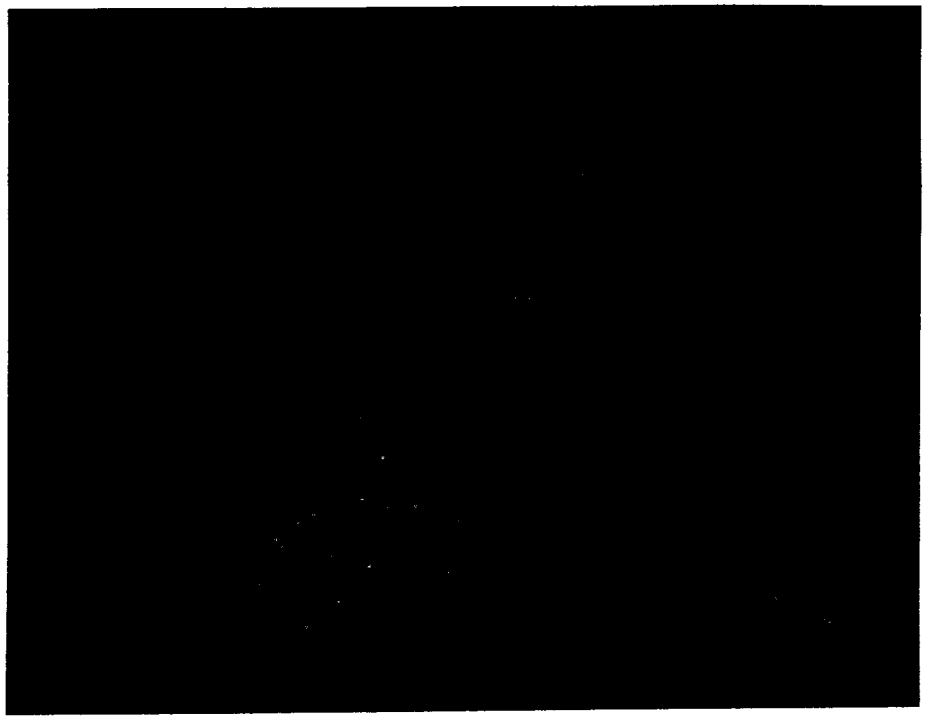


Figure 20.23: Spherical triangulation and tessellation.

Because of this relationship between the Dirichlet tessellation and the optimal MaxMin triangulation, we can extend the idea of MaxMin or Delaunay triangulation to any domain where we can compute the distance between two points. The sphere provides an interesting and useful example. Here the distance between two points p and q is easily computed as $\cos^{-1}(p \cdot q)$ so the Dirichlet tessellation is also easy to compute. An example is shown in the right image of Figure 20.23. The left image depicts the triangulation which is dual to this tessellation.

There have been many other criteria for characterizing optimal triangulations that have been studied and discussed in the literature. Some turn out to be equivalent to those we have mentioned here and some only appear to be similar, so one needs to be rather careful. Even though the terminology can be similar, the criterion of minimizing the maximum angle is not the same as the MinMax criterion we have described here. It is easily the case that the two quite different triangulations with different vectors A_τ (as defined above) could have the same maximum angle and could both be a triangulation which minimizes the maximum angle. The example of Figure 20.20 has this property. Each of the triangulations $\tau_6, \tau_7, \tau_9, \tau_8$, and τ_4 have a maximum angle of 2.42, which turns out to be a minimum, so any one of these triangulations would satisfy the criterion of minimizing the maximum angle, while only τ_4 satisfies the MinMax criterion described here. Overall, the topic of optimal triangulations can be rather technical, and one has to be careful when comparing results found in the literature.

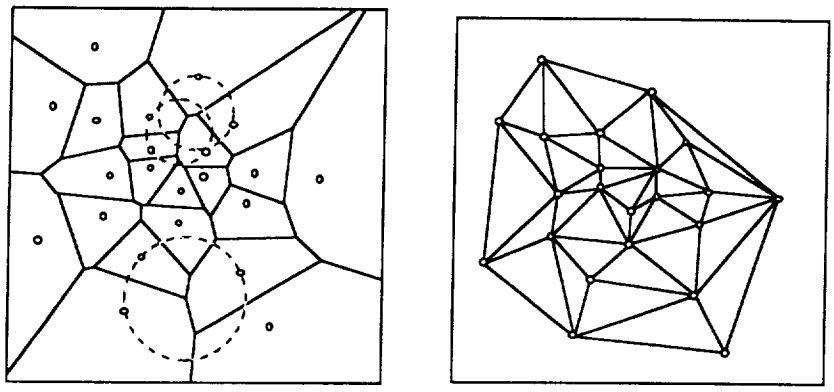


Figure 20.24: The Dirichlet tessellation and its dual triangulation.

Algorithms for Delaunay Triangulations

In this section we discuss some ideas and techniques leading to algorithms for computing the Delaunay triangulation of a set of points in the plane. In general, this is a very rich and full area of research and here we can only provide a glimpse. The literature is very abundant with both practical and theoretical papers on this subject. There is not a single "best" algorithm. The choice depends upon the particular application and the tools and resources available. It is a good strategy to be armed with a collection of ideas, tools, and techniques so that an effective algorithm can be custom-designed for the application at hand. Our approach for the material for this section is based upon a discussion of the ideas behind a few selected algorithms. Our selection is based upon potential usefulness of the ideas and on which would be representative. In addition, we are particularly interested in those ideas which extend most easily to three dimensions. But, just for the sake of interest, we have included the description of one 2D algorithm which does not extend at all to 3D!

The Swapping Algorithm of Lawson [139]: The basic operation of this algorithm consists of swapping the diagonal of a convex quadrilateral. Lawson [138] showed that any triangulation of the convex hull can be obtained from any other triangulation by a sequence of these operations. (Later this property was established for nonconvex domains by Dyn and Goren [66].) Furthermore, Lawson proved that if the choice of the diagonal is made on the basis of the MaxMin criterion for the quadrilateral only, eventually the global optimal triangulation will be obtained. In other words, for this criterion, a local optimum is a global optimum. A typical implementation of this type of algorithm would insert new points (say, in sorted x-order) in the interior of an existing triangulation or connect to all points on the boundary which are visible from the new point. This new triangulation is then optimized by testing and possibly swapping the diagonals of convex quadrilaterals. It is interesting to note that this type of algorithm will not necessarily produce the MinMax because for this criterion, a local extreme is not necessarily a global optimum. The example of Figure 20.20 illustrates this. Based upon the MinMax criterion, τ_4 is optimal and τ_8 is a local minimum. Locally optimal swaps of diagonals from τ_8 would never lead to τ_4. The algorithm could

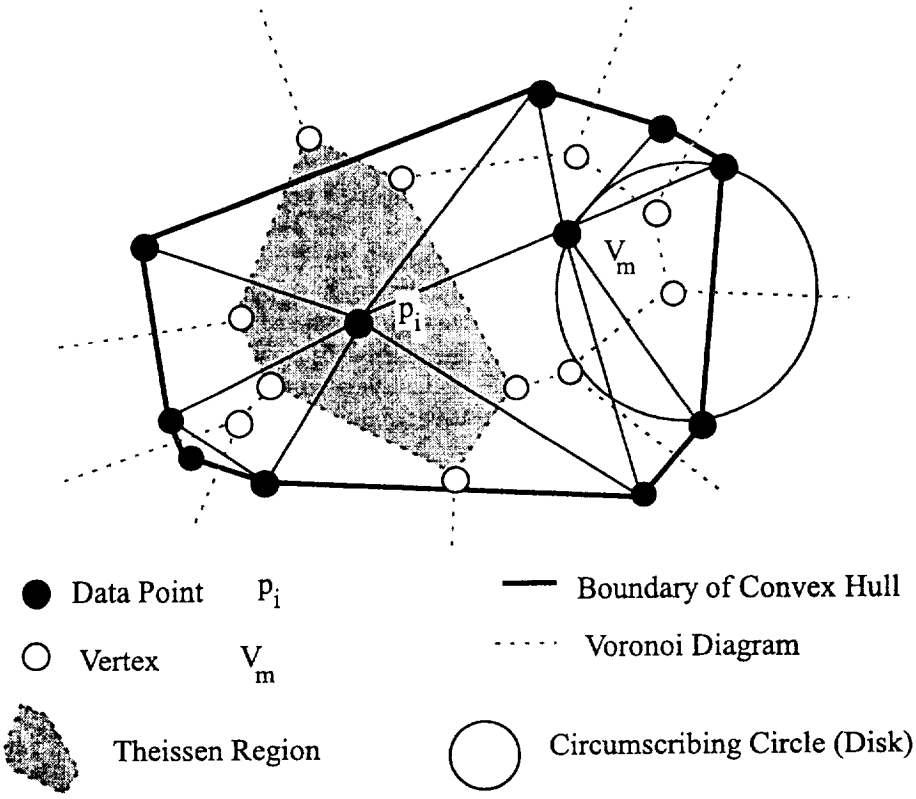


Figure 20.25: Notation and terminology for Delaunay triangulation and Dirichlet tessellation.

easily get trapped in a local extreme at τ_8. The ideas of simulated annealing can be used to develop algorithms which can escape from these local extrema. See Schumaker [225], for example.

The Algorithm of Green and Sibson [107]: This algorithm depends heavily upon a particular data structure used to store the Delaunay triangulation (or Dirichlet tessellation). For each object (a Dirichlet tile or window boundary constraint) is recorded in a "contiguity list" consisting of all objects with which it is contiguous. This data structure is very similar to the contiguity list structure we described in Figure 20.9 but it also includes some window boundary constraints. New points are inserted sequentially. We quote directly from [107] to how this is done.

> The contiguity list for the new point is then built up in reverse (that is, clockwise) order and subsequently standardised. We begin by finding where the perpendicular bisector of the line joining the new point to its nearest neighbour meets the edge of the nearest neighbour's tile, clockwise round the new point. Identifying the edge where this happens gives the next object contiguous with the new point and this is in fact the first to go onto its contiguity list. The new perpendicular bisector is then constructed and its incidence on the edge of this new tile is examined to obtain the subsequent contiguous object: successive objects are added to the contiguity list in this way until the list is completed by the addition of the nearest neighbour. Whilst this being done old contiguity lists are being modified: the new point is inserted in each and any contiguities strictly between the entry and exit points of the perpendicular bisector are deleted, the anticlockwise-cyclic arrangement of the lists making both this and the determination (sic) of the exit very easy.

This insertion algorithm requires the computation of the nearest existing data point to the data point that is to be inserted. The authors discuss an algorithm which takes advantage of the tessellation computed so far. In the authors' words: "Simply start at an arbitrary point and "walk" from neighbour to neighbour, always approaching the new point, until the point nearest to it is found."

The Algorithm of Bowyer [21]: Bowyer described an algorithm for inserting a new point (lying in the convex hull) into an existing Delaunay triangulation. An example given by Bowyer and which we include in Figure 20.27 serves to define this data structure. In the terminology of Bowyer, the forming points for a vertex are simply the vertices of the triangle which has this particular vertex as the center of its circumcircle. Since each triangle gives rise to a vertex, giving a list of indices of the forming points for each vertex (as Bowyer does) is equivalent to giving a list of indices of the data points which comprise each triangle of triangulation. Except for a change in ordering, the neighboring vertices are exactly the same as the indices of the triangle neighbors as given in the triangular grid data structure of Figure 20.26.

In order to insert a new point (Q in Figure 20.27) within the current convex hull of the data points, Bowyer [21] gives the following algorithm:

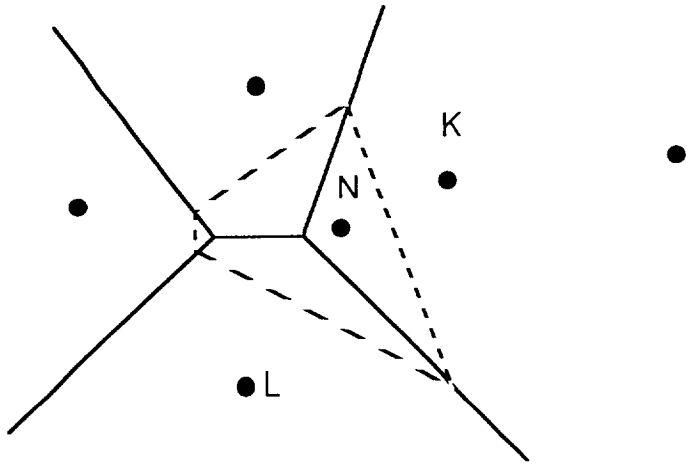


Figure 20.26: An aid to the Green and Sibson algorithm.

1. Identify a vertex currently in the structure that will be deleted by the new point (say V_4). Such a vertex is any that is *nearer to the new point than to its forming points.*
2. Perform a tree search through the vertex structure starting at the deleted vertex looking for others that will be deleted. In this case the list will be: $\{V_4, V_3, V_5\}$.
3. The points contiguous to Q are all the points forming the deleted vertices: $\{P_2, P_5, P_4, P_3, P_7\}$.
4. An old contiguity between a pair of those points will be removed ($P_2 - P_4$ say) if all of its vertices $\{V_4, V_3\}$ are in the list of deleted vertices.
5. In this case the new point has five new vertices associated with it: $\{W_1, W_2, W_3, W_4, W_5\}$. Compute their forming points and neighbouring vertices. The forming points for each will be the point Q and two of the points contiguous to Q. Each line in the tessellation has two points around it (the line $V_3 - V_2$, for example, is formed by P_3 and P_4). The forming points of the new vertices and their neighbouring vertices may be found by considering vertices pointed to by members of the deleted vertex list that are not themselves deleted, and finding the rings of points around them. Thus W_5 points outwards to V_2 from Q and is formed by $\{P_3, P_4, Q\}$.
6. The final step is to copy some of the new vertices, overwriting the entries of those deleted to save space.

The Algorithm of Watson [254]: This algorithm relies on the property of a Delaunay triangulation that a triple of data point indices (i, j, k) will be in I_t provided the circumcircle of p_i, p_j, and p_k contains no other data points. As with the other algorithms, this algorithm is based upon inserting a new point into an already existing Delaunay triangulation. The general philosophy of Watson's approach is described by the following two steps:

Vertex	Forming points			Neighboring vertices		
	1	2	3	1	2	3
V_1	P_6	P_4	P_5	V_4	ϕ	V_6
V_2	P_1	P_4	P_3	V_3	ϕ	V_7
V_3	P_2	P_3	P_4	V_2	V_4	V_5
V_4	P_2	P_5	P_4	V_1	V_3	ϕ
V_5	P_7	P_3	P_2	V_3	ϕ	ϕ
V_6	P_6	P_8	P_4	V_7	V_1	ϕ
V_7	P_1	P_8	P_4	V_6	V_2	ϕ

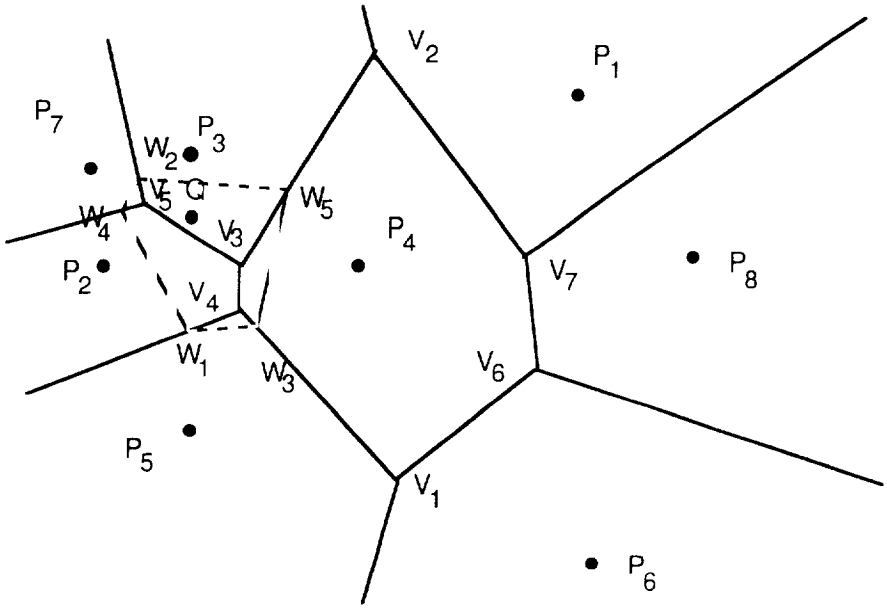


Figure 20.27: Illustrating the algorithm of Bowyer [21].

1. Find all triangles whose circumcircle contains the point to be inserted.

2. For each of these triangles, form three new triangles from the point to be inserted and the three edges of this triangle and test to see if any of these three new triangles contain any other data points. If not, then add this new triangle to the triangulation.

More details for this general approach are given in the flow diagram of Figure 20.28, which is based upon the flow diagram of [254].

Watson [254] describes a number of features and details to make the basic algorithm efficient and eventually discusses a particular implementation which he says has an expected running time which is observed to increase not more than $N^{3/2}$.

The Embedding/Lifting Approach: Algorithms of this type are based upon a very interesting relationship that exists between the three-dimensional convex hull of the lifted points $(x_i, y_i, x_i^2 + y_i^2)$ and the Delaunay triangulation. Faces on the convex hull are designated as being either in the upper or lower part. The lower part consists of faces which are supported by a plane that separates the point set from $(0, 0, -\infty)$. The Delaunay triangulation is obtained directly from the projection onto the xy-plane of the lower part of the convex hull. See [27] and [68]. An algorithm for computing the convex hull which is based on an initial sort followed by a recursive divide-and-conquer approach has been described by Preparata and Hong [202]. This algorithm is also covered in [68] and [203]. Theoretically the algorithm is optimal time $O(n \cdot \log(n))$, but Day [49] reports that empirical data implies a worst-case complexity of $O(N^2)$. Day covers many of the details and special-case issues of practical interest for implementation which are often brushed over in more theoretical papers.

Divide-and-Conquer Algorithms: The general structure of this type of algorithm is to divide the data set into subsets A and B, solve the problem for A and solve the problem for B and merge the results into a solution for A $\cup$ B. See Figure 20.29. Divide-and-conquer algorithms can lead to theoretically optimal algorithms, but often fail to be competitive in practical usage. The merging portion is often the most troublesome in trying to maintain bounds on the running times and complexity of the algorithm.

20.2.3 Visibility Sorting of Triangulations

This is an example of an area that is interesting in 3D but not in 2D. It is possible to make a definition of a visibility sort for a triangulation which is completely analogous to that of a tetrahedrization, but there does not appear to be any application or use for such a property. We defer further discussion on visibility sorting to Section 20.3.3.

20.2.4 Data-Dependent Triangulations

The topic of data-dependent triangulations arises within the context of determining a modeling function $F(x, y)$ for the data $(F_i; x_i, y_i), i = 1, \ldots, N$. A relatively simple approach to defining a modeling function is to first form a triangulation of the convex hull of the independent data $(x_i, y_i), i = 1, \ldots, N$ and then define F to be piecewise linear over this

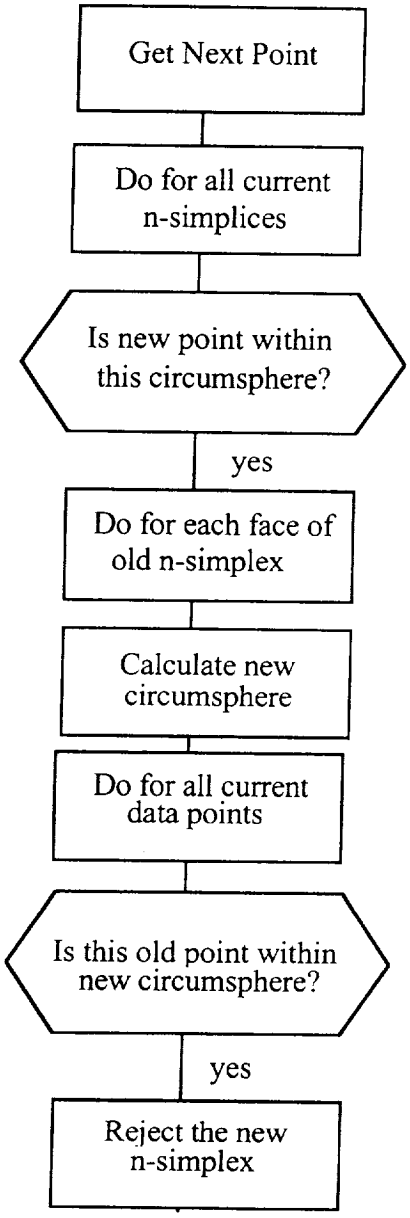


Figure 20.28: Flow diagram for Watson's algorithm.

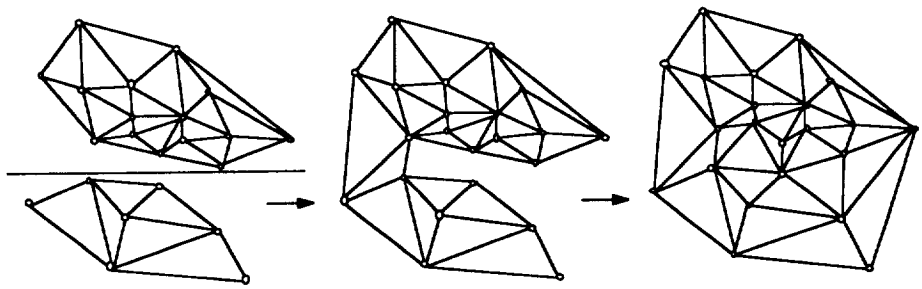


Figure 20.29: Divide-and-conquer algorithms.

triangulation. This will yield a C^0 (continuous) function which interpolates the data; that is, $F(x_i, y_i) = F_i, i = 1, \ldots, N$. We denote this function by $F_T(x, y)$. Any triangulation of the independent data $(x_i, y_i), i = 1, \ldots, N$ will suffice for this approach. While we are well aware of the many desirable properties of the Delaunay triangulation, it might very well be the case that some other triangulation whose choice would depend upon the values $F_i, i = 1, \ldots, N$ would lead to some desirable properties for the modeling function F. This is the basic idea of data-dependent triangulation. Of course, there are potentially many ways to accomplish this, but we choose for this discussion here to briefly describe the criteria called "nearly C^1" as proposed in [67]. An ordering is imposed on the collection of all possible triangulations of the convex hull in the following manner. First a local cost function (see Figure 20.30) for each edge $e_i = 1, \ldots, N_{ie} = N_e - N_b$ is defined and denoted by $S(F_T, e_i)$. (We will shortly describe the four examples of local cost functions covered in [67].) If T and T' are two triangulations, then

$$T \leq T'$$

provided the vector

$$(s(F_T, e_1), s(F_T, e_2), \ldots, s(F_T, e_{N_{ie}}))$$

is lexicographically less than or equal to

$$(s(F_{T'}, e_1), s(F_{T'}, e_2), \ldots, s(F_{T'}, e_{N_{ie}})).$$

It is assumed that the components of these vectors are arranged in nonincreasing order. The goal is then to find the optimal data-dependent triangulation which is defined by having the smallest associated vector under this lexicographical ordering. Since there are only a finite (albeit possibly very large) number of possible triangulations, we know that a global minimum exists even though it may not be unique and it may not be so easy to compute. The algorithm used in [67] is similar to the swapping algorithm of Lawson (which we have described above in Section 20.2.2) in that an initial triangulation is obtained and then an internal edge of a convex quadrilateral is considered. If $T' < T$, where T' is the same triangulation as T except the diagonal of the convex quadrilateral has been switched, then this switch is made and other edges are considered for potential swapping. Since each swap moves strictly lower in the lexicographic ordering, we are guaranteed that this algorithm

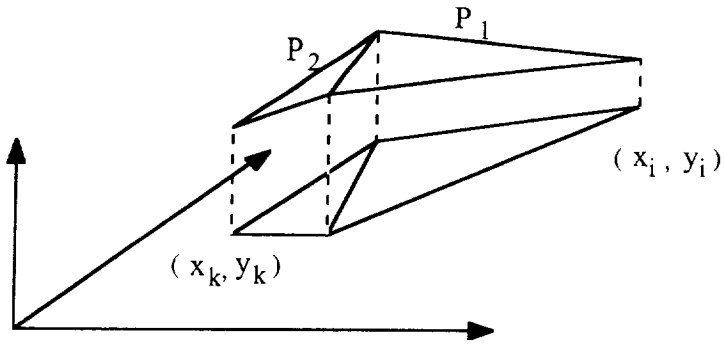


Figure 20.30: Notation for local cost function definitions.

will eventually converge after a finite number of steps. This means that swapping any edge would not move to a smaller triangulation. This limit triangulation may not be the global minimum; it is only guaranteed to be a local minimum and steps to find the global minimum must do more than swap diagonals which improve (with respect to the ordering) the triangulation.

We now describe the four local edge cost functions used in [67]. Let $P_1 = a_1x+b_1y+c_1$ and $P_2 = a_2x + b_2y + c_2$ be the two planes defined over the two triangles of a convex quadrilateral.

i) The angle between normals: The local cost function is taken as the acute angle between N_1 and N_2, which are the respective normals for P_1 and P_2.

$$s(F_T, e) = \cos^{-1}(A)$$

where

$$A = \frac{a_1a_2 + b_1b_2 + 1}{\sqrt{(a_1^2 + b_1^2 + 1)(a_2^2 + b_2^2 + 1)}}$$

ii) The jump in normal derivative: This cost function is the difference between the derivative of P_1 and P_2. This derivative is taken in the direction perpendicular to the edge dividing the two triangles.

$$s(F_T, e) = [n_x(a_1 - a_2) + n_y(b_1 - b_2)]$$

where (n_x, n_y) is a unit vector perpendicular to the edge e.

iii) The deviations from linear polynomials: The cost function measures the error between P_1 and P_2, evaluated at the other point of the quadrilateral.

$$s(F_T, e) = \sqrt{(P_1(x_i, y_i) - F_i)^2 + (P_2(x_k, y_k) - F_k)^2}$$

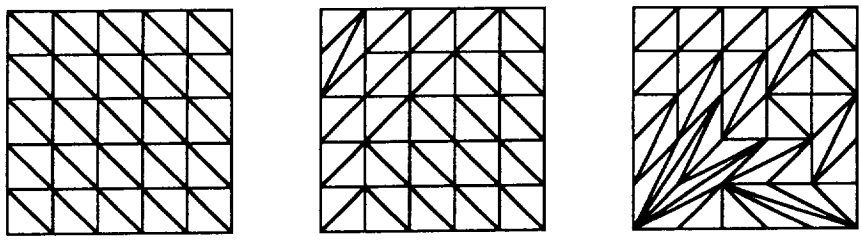


Figure 20.31: Examples of data-dependent triangulations.

iv) The distance from planes: This cost function measures the distance between the planes P_1 and P_2 and the corresponding vertex of the quadrilateral.

$$s(F_T, e) = \sqrt{\frac{(P_1(x_i, y_i) - F_i)^2}{a_1^2 + b_1^2 + 1} + \frac{(P_2(x_k, y_k) - F_k)^2}{a_2^2 + b_2^2 + 1}}$$

Some typical results are given in [67] which confirm the expectation that using the optimal data-dependent triangulation improves the overall fitting properties of F_T over that of the Delaunay triangulation, which, by the way, is used as the initial triangulation for the swapping algorithm. It is observed that long, thin triangles tend to appear where the data seems to indicate a function that is increasing (or decreasing) relatively rapidly in a certain direction. The use of the data-dependent triangulation generally gives an overall reduction in errors when certain test functions are used to generate the data.

As we have mentioned, the local swapping algorithm used in [67] can only find a local minimum. In order to move more closely to the globally optimal data-dependent triangulation, Schumaker [225] and Quak and Schumaker [204,205,206] have involved the tools of simulated annealing. More details on this are contained in Section 20.3.6 on data-dependent tetrahedrizations. We include here the results of one example described by Schumaker. The data consists of

$$(F_{ij}; x_i, y_j); \quad x_i, y_j = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0;$$

where

$$F(x, y) = (y - x^2)_+.$$

Three triangulations are shown in Figure 20.31. The first is the Delaunay triangulation of the independent data. The next is the triangulation which results from the local swapping algorithm of [67] using the local cost function of "angle between normals." The last is the triangulation after simulated annealing has been applied. The associated vectors for each of these triangulations is given in Figure 20.33 and the piecewise linear approximations over these triangulations are shown in Figure 20.32.

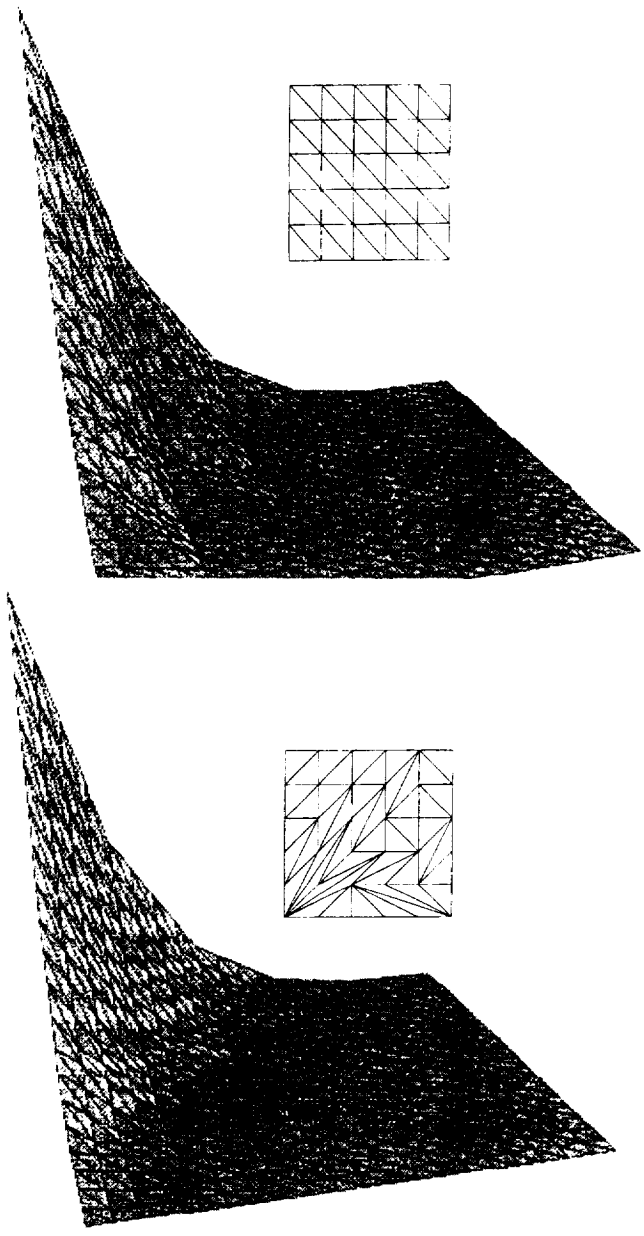


Figure 20.32: The graphs of Schumaker's example. See [225].

Angles between normals for Delaunay triangulation:

55.077	48.155	44.684	39.801	39.588	38.378	37.734	35.445	33.992
33.786	33.561	33.162	30.470	28.898	28.287	27.284	27.284	26.003
23.633	21.958	20.814	17.886	16.066	15.942	15.642	11.310	10.302
9.661	7.294	7.294	7.294	6.843	0.649	0.649	0.459	0.458
0.458	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000							

Angles between normals for locally optimal triangulation:

35.993	30.590	26.070	23.610	21.813	21.558	16.563	16.521	15.793
12.810	11.929	11.310	10.646	10.261	9.622	8.844	8.707	8.321
8.076	8.047	5.794	5.563	3.777	0.649	0.649	0.459	0.459
0.458	0.458	0.458	0.448	0.020	0.020	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000							

Angles between normals for annealed triangulation:

26.070	22.929	22.113	20.049	17.257	16.563	16.521	13.031	12.505
11.929	10.389	10.270	10.261	8.954	8.321	7.844	5.962	5.794
5.256	1.652	1.480	1.025	0.649	0.648	0.459	0.458	0.458
0.448	0.447	0.020	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000							

Figure 20.33: Angles for the data-dependent triangulation.

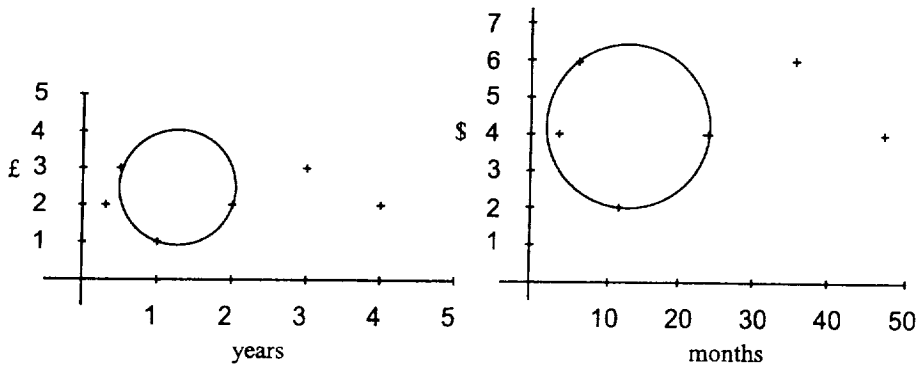


Figure 20.34: Two different units used to measure the same data lead to two different Delaunay triangulations.

20.2.5 Affine Invariant Triangulations

The desirable properties of the Delaunay triangulation have been previously discussed. Unfortunately, this optimal triangulation is not invariant under affine transformations, and this means that methods for analyzing and visualizing data that use this particular triangulation can be affected by the choice of units used to measure the data. This could be considered an undesirable property. In this section we describe a relatively new method for characterizing and computing an optimal triangulation which is invariant under affine transformations. Before we proceed with the discussion of these techniques, we wish to motivate further the desirability of affine invariance.

As we have mentioned earlier, one of the main purposes for triangulations and tetrahedrizations is their use in defining functions in a piecewise manner over the domain of a data set. It would be undesirable if the happenstance of the choice of units used to measure the data were to affect the definition of a data modeling function. But this does happen with the Delaunay triangulation. The example of Figure 20.34 points this out. This data represents the independent data; the dependent data is not given as it is not important in this context. The data is the same in both the left and right graphs of Figure 20.34; the only difference is that in the left graph we have used years and £ (pounds, British monetary unit, approximately equal and assumed here to be exactly equal to two US dollars), and in the right graph we have used months and dollars. If we use the units of years and £ then we can see that the three vertices (1yr, 1£), (0.5yr, 3£), (2yr, 2£) will mark out a triangle to be included in the list of triangles for the Delaunay triangulation. But on the other hand, if we use months and $ we can see that the circumcircle defined by these same three vertices (12mon, 2$), (6mon, 6$), (24mon, 4$) contains the data point (4mon, 4$). Therefore, these three vertices will not comprise a triangle of the Delaunay triangulation if these units are used. This simple example points out the possible effects of the choice of the units of measurement. The choice of the units of measurement is the same as a change in scale, $x \leftarrow ax$ and $y \leftarrow by$. Uniform scale changes of the type $x \leftarrow ax$, $y \leftarrow ay$ will not affect the Delaunay triangulation.

We now discuss how to avoid this problem. It would be possible to simply normalize

all data ranges to one unit by scaling by the range. But this approach would mean that rotations of the data could have an effect on the Delaunay triangulation, meaning the final data model would be affected by rotations of the data. In other words, the placement and alignment of the axes for the measurement of the data would have an effect on the data modeling function and subsequently on our analysis of the data, and this we would like to have the opportunity to avoid. It would, in general, be useful to have a characterization (and subsequent algorithms) for an optimal triangulation which is not affected by affine transformation. An affine transformation is a map of the form

$$(x, y) = A(x, y) + c$$

where A is a 2×2 matrix and c is a two-dimensional point. Affine transformations include not only scale changes and rotations, but also translations, reflections, and shearing transformations. The approach to such an optimal triangulation covered here is through the duality that exists between the conventional Delaunay triangulation and the Dirichlet tessellation. As we described previously, the characterization of the Delaunay triangulation (as a MaxMin triangulation), it was heavily dependent upon angles, and angles are affected by scaling transformations; so it should be no surprise that the Delaunay triangulation is also affected by scaling transformations. But the definition of the Dirichlet tessellation uses only distance and we know that the Delaunday triangulation is dual to (a direct result of) the Dirichlet tessellation. The approach here is to use a method of measuring distance which is invariant under affine transformations. The Dirichlet tessellation based upon this new method of measuring distance will have a dual which will serve as our optimal triangulation. Rather than use the standard Euclidean norm $||(x, y)||^2 = \sqrt{x^2 + y^2}$ we propose the use of the following norm

$$||(x, y)||_V^2 = (x, y) \begin{pmatrix} \frac{\Sigma_y^2}{\Sigma_x^2\Sigma_y^2 - (\Sigma_{xy})^2} & \frac{-\Sigma_{xy}}{\Sigma_x^2\Sigma_y^2 - (\Sigma_{xy})^2} \\ \frac{-\Sigma_{xy}}{\Sigma_x^2\Sigma_y^2 - (\Sigma_{xy})^2} & \frac{\Sigma_x^2}{\Sigma_x^2\Sigma_y^2 - (\Sigma_{xy})^2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{20.1}$$

where

$$\Sigma_x^2 = \frac{\sum_{i=1}^N (x_i - \mu_x)^2}{N}, \qquad \mu_x = \frac{\sum_{i=1}^N x_i}{N}$$

$$\Sigma_y^2 = \frac{\sum_{i=1}^N (y_i - \mu_y)^2}{N}, \qquad \mu_y = \frac{\sum_{i=1}^N y_i}{N}$$

$$\Sigma_{xy} = \frac{\sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)}{N}$$

and

$$V = \begin{pmatrix} x_1 - \mu_x & x_2 - \mu_x & \cdots & x_N - \mu_x \\ y_1 - \mu_y & y_2 - \mu_y & \cdots & y_N - \mu_y \end{pmatrix}.$$

We have used the subscript of V on the norm to explicitly indicate that this method of measuring distance is dependent upon the data set. Change the data set and you change how you measure distance, but the distance between any two data points will remain constant. This norm and its use within the context of scattered data modeling was first described in [181]. This norm has the property that it is invariant under affine transformations. More precisely,

$$\|P - Q\|_V = \|T(P) - T(Q)\|_{T(V)} \tag{20.2}$$

for any two points $P = (x, y)$ and $Q = (u, v)$ and any affine transformation

$$T(P) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}.$$

Here, $T(V)$ (used as a subscript in Equation (20.2)) is the transformed data

$$T(V) = \begin{pmatrix} T\begin{pmatrix} x_1 - \mu_x \\ y_1 - \mu_y \end{pmatrix} & T\begin{pmatrix} x_2 - \mu_x \\ y_2 - \mu_y \end{pmatrix} & \cdots & T\begin{pmatrix} x_N - \mu_x \\ y_N - \mu_y \end{pmatrix} \end{pmatrix}.$$

Figure 20.35 illustrates the properties of this new method of measuring distance. Each of the data sets shown in this figure are affine images of each other. Starting in the upper left and moving in a clockwise direction, the transformations are: counterclockwise rotation of 44 degrees; a scaling in x by a factor of 2; a scaling in y by a factor of 0.4. The four ellipses in each figure represent points which are $1/4$, $1/2$, $3/4$, and 1 unit(s) from their center point as measured with the affine invariant norm. In Figure 20.36 we show the Dirichlet tessellation of these four affinely related data sets, and in Figure 20.37 we show the corresponding dual triangulation. As one can see, the triangulation is unchanged by these transformations.

As a comparison, we have also included the Delaunay triangulation based upon the standard Euclidean norm in Figure 20.38. And as we indicated earlier, we can see that triangulation results are affected by the transformations. Not all triangles are changed, but some are.

And now we suggest some practical information on how to incorporate this feature into an algorithm for computing triangulations. If you already have a procedure for computing an optimal triangulation, then it is possible to modify it slightly to achieve the results we have described in this section. Say, for example, that the procedure is based upon Lawson's algorithm and that there is a subprocedure which decides whether or not to switch the diagonal of a quadrilateral formed from two triangles. It might be that this procedure is based solely on Euclidean distance. That is, the center and radius of the circumcircle of three points are determined and the distance to the center from the fourth point is computed so as to make this decision. In order to modify this subprocedure, we need only to replace the use of the Euclidean norm with the affine invariant norm described here. The equations for computing circumscribing circles (ellipses) for a quadratic norm in general are given in [182]. If, on the other hand, the procedure you are already using is known to be rotation invariant, then there is an even easier way to affect the results of the affine invariant triangulation. This is based upon the factorization of the matrix which defines the affine invariant norm. We denote this matrix by $A(V)$ so that we have

$$\|(x, y)\|_V^2 = (x, y)A(V)\begin{pmatrix} x \\ y \end{pmatrix}$$

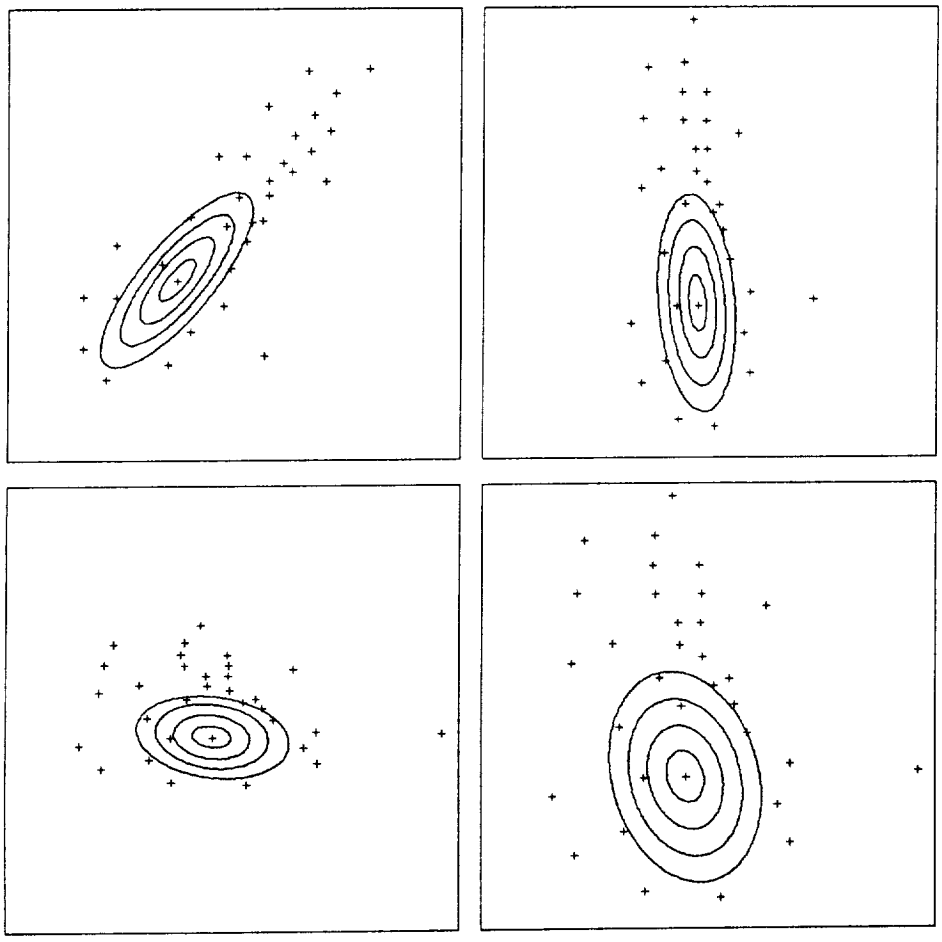


Figure 20.35: Affine transformations of a data set and points equally distant (affine invariant norm) from a point.

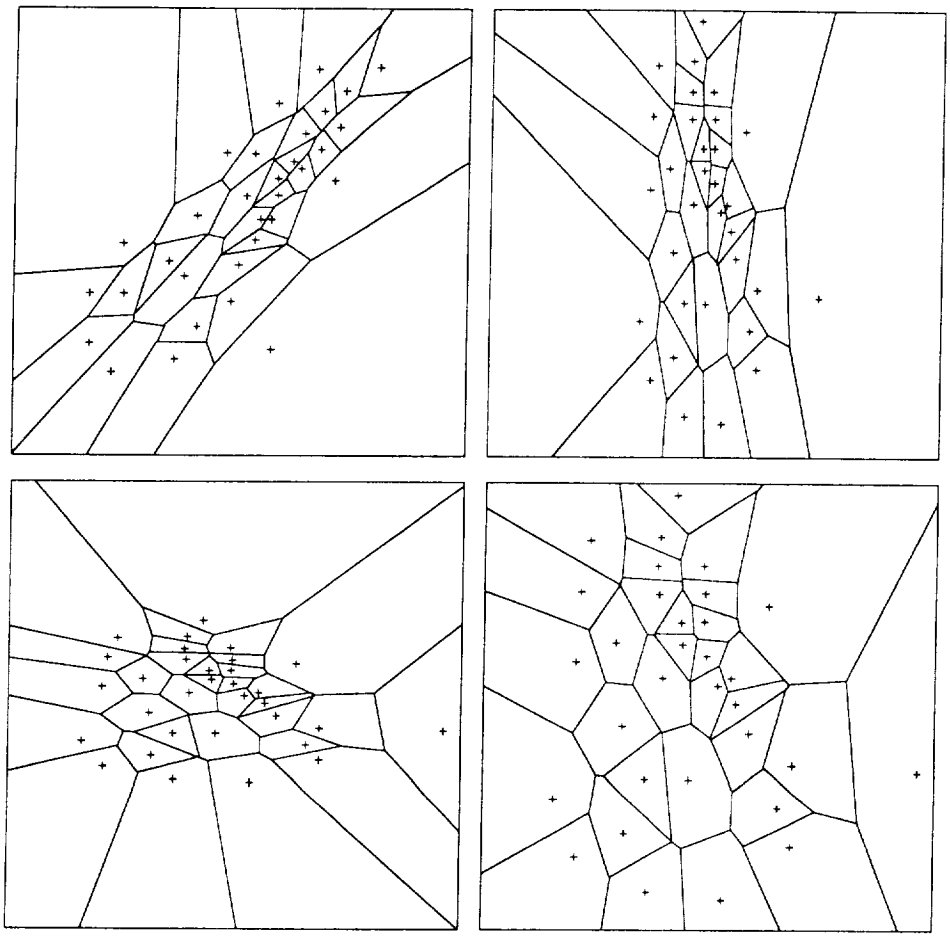


Figure 20.36: The Dirichlet tessellation (affine invariant norm) of affine transformations of a given data set.

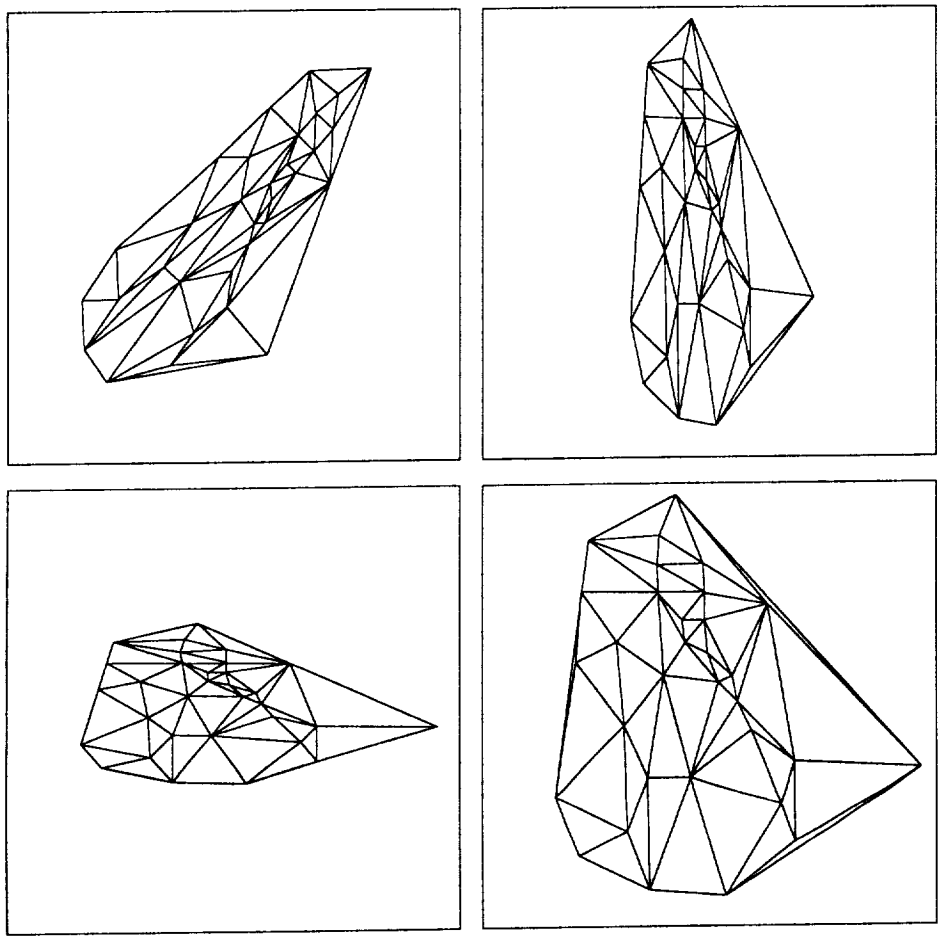


Figure 20.37: The triangulation dual to the Dirichlet tessellation (affine invariant norm) of a given data set and some affine transformations.

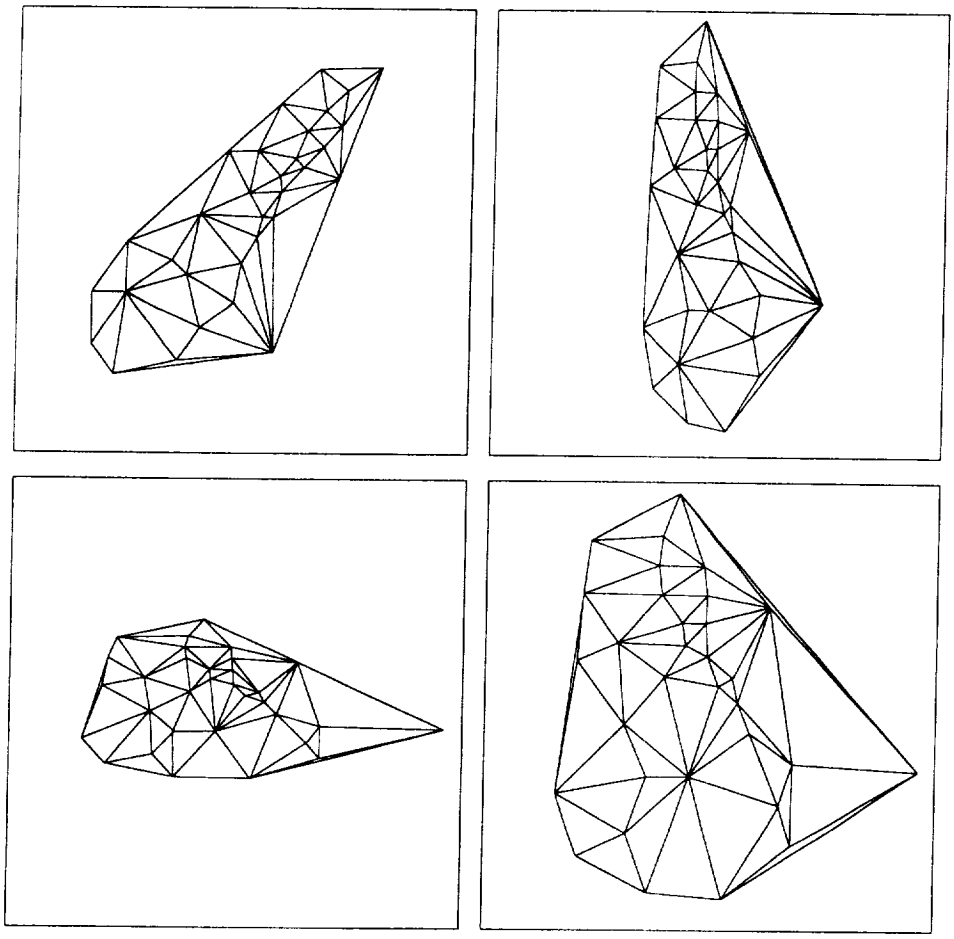


Figure 20.38: The Delaunay triangulation of a data set and some affine transformations.

The matrix $A(V)$ can be factored (Cholesky) into

$$A(V) = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{pmatrix} = L(V)L(V)^*.$$

Here the notation $L(V)^*$ denotes the transpose of $L(V)$.

Using this factorization, we have that

$$\|(x,y)\|_V^2 = (x,y)L(V)L(V)^* \begin{pmatrix} x \\ y \end{pmatrix} = \|(x,y)L(V)\|^2$$

which means measuring distances with the affine invariant norm is the same as measuring distance in the standard Euclidean but with the points transformed by multiplying by $L(V)$. This means that we can achieve the result of the optimal affine invariant triangulation by computing the standard Delaunay triangulation on the transformed data

$$(X_i, Y_i) = (x_i, y_i)L(V)$$

In summary, we need only to compute

$$l_{11} = \sqrt{a_{11}}, \quad l_{21} = \frac{a_{21}}{\sqrt{a_{11}}}, \quad l_{22} = \sqrt{\frac{a_{11}a_{22} - a_{21}^2}{a_{11}}}$$

where $a_{11} = \dfrac{\Sigma_y^2}{\Sigma_x^2\Sigma_y^2 - (\Sigma_{xy})^2}$, $a_{21} = \dfrac{-\Sigma_{xy}}{\Sigma_x^2\Sigma_y^2 - (\Sigma_{xy})^2}$, and $a_{22} = \dfrac{\Sigma_x^2}{\Sigma_x^2\Sigma_y^2 - (\Sigma_{xy})^2}$ and to apply any rotation invariant triangulation algorithm to the transformed data

$$\begin{aligned} X_i &= l_{11}x_i + l_{21}y_i \\ Y_i &= l_{22}y_i, \qquad i = 1, \ldots, N. \end{aligned}$$

20.2.6 Interpolation in Triangles

We now take up the topic of interpolating into (or over) a single triangular domain. The interpolants we describe here form the basic building blocks for constructing the global interpolants which have piecewise definitions over the individual triangles of a triangulation. The domain here is a single triangle, $T = T_{ijk}$ with vertices V_i, V_j, and V_k, and the data consists of values given on the boundary of the triangular domain. We need to differentiate between two types of boundary data. If the data consists of function and certain derivative values specified only at the vertices (or possibly other points such as midpoints), then we call this *discrete data*. If, on the other hand, the data is provided on the entire boundary of the triangle, we refer to this type of data as *transfinite data*. The importance of an interpolant which will match transfinite data is that it serves as a prototype for developing a large variety of discrete interpolants. This is accomplished through the process of *discretization*, where the data required for a transfinite interpolant is provided by means of using some interpolation scheme only on the boundary, discrete data. For example, given only data values at the vertices, we can use linear interpolation along an edge to produce

the transfinite data required by the transfinite interpolant, or if we also have data values at the midpoints, we could use quadratic interpolation.

There is a second concept which is rather important for interpolants defined over triangles and this has to do with the degree of continuity of the global interpolant. Often, we require that the global interpolant at least be continuous. We call such an interpolant a C^0 interpolant. If the global interpolant has continuous first-order derivatives, we say it is a C^1 interpolant. A C^0 interpolant for a single triangle is one which interpolates to boundary data consisting of only position values, either at the vertices only (and possibly points along the edges) or on the entire boundary. A C^1 interpolant for a single triangle is one which will interpolate to first-order derivative data specified on the boundary. But this must be done in a manner so as to guarantee C^1 continuity across the boundary edges. So, for example, if the cross-boundary derivative varies quadratically along an edge, then the data on this edge must be sufficient to uniquely determine this derivative, so that on an adjoining triangle we will have exactly the same cross-boundary derivative. For this reason, it is common for C^1 interpolants to have linearly varying cross-boundary derivatives which are determined by their values at the two endpoint vertices.

Combining the two concepts of discrete and transfinite data and C^0 and C^1 data leads to four types of triangular interpolants. This general area of interpolation in triangles is fairly rich and well developed, and we urge the really interested reader to follow the citations into the literature (for example [177], [178], and [189]) after taking a look at the sampling we have chosen to include here. We first cover C^0, discrete interpolants, then a sampling of three C^0, transfinite interpolants. This is followed by the description of a C^1, discrete interpolant. We have chosen to include a discretized version of the minimum norm triangular interpolant (see [178]). Another rather popular C^1, discrete interpolant, is the Clough/Tocher interpolant often mentioned in conjunction with the finite element method. Much has been written about this interpolant in the past and so we do not include it here. This section is concluded with a description of a C^1, transfinite interpolant, called the side-vertex interpolant [177]. It is one of the easiest to describe and the most versatile to use. It also generalizes rather nicely to a tetrahedral domain.

C^0, Discrete Interpolation in Triangles

The lowest-degree polynomial, C^0, discrete interpolant, is linear and unique. Given the data $F(V_i)$, $F(V_j)$, and $F(V_k)$, the coefficients of the linear function

$$F(x, y) = a + bx + cy$$

which interpolates this data can be found by solving the linear system of equations

$$\begin{aligned} a + bx_i + cy_i &= F(V_i) \\ a + bx_j + cy_j &= F(V_j) \\ a + bx_k + cy_k &= F(V_k). \end{aligned}$$

Another path to this basic linear interpolant is via barycentric coordinates. Given a point $V = (x, y)$, barycentric coordinates, b_i, b_j, and b_k of this point relative to the triangle T_{ijk}

are defined by the relationships

$$\begin{pmatrix} x \\ y \end{pmatrix} = b_i V_i + b_j V_j + b_k V_k$$
$$1 = b_i + b_j + b_k.$$

The linear interpolant now takes the form

$$F(x, y) = F(V) = b_i F(V_i) + b_j F(V_j) + b_k F(V_k).$$

There are several alternative ways of defining or determining the barycentric coordinates of a point. For example,

$$b_i = \frac{A_i}{A} \quad b_j = \frac{A_j}{A} \quad b_k = \frac{A_k}{A}$$

where A_i, A_j, and A_k represent the areas of the subtriangle shown in Figure 20.39 and A is the area of T_{ijk}. Also,

$$b_i = \frac{\begin{vmatrix} x - x_k & x_j - x_k \\ y - y_k & y_j - y_k \end{vmatrix}}{\begin{vmatrix} x_i - x_k & x_j - x_k \\ y_i - y_k & y_j - y_k \end{vmatrix}} \qquad b_j = \frac{\begin{vmatrix} x - x_i & x_i - x_k \\ y - y_i & y_i - y_k \end{vmatrix}}{\begin{vmatrix} x_j - x_i & x_i - x_k \\ y_j - y_i & y_i - y_k \end{vmatrix}}$$

$$b_k = \frac{\begin{vmatrix} x - x_j & x_i - x_j \\ y - y_j & y_i - y_j \end{vmatrix}}{\begin{vmatrix} x_k - x_j & x_i - x_j \\ y_k - y_j & y_i - y_j \end{vmatrix}}.$$

Given the values at the three vertices and the three midpoints of a triangle, there is a unique quadratic which interpolates this data,

$$\begin{aligned} Q(x, y) = {} & F(V_i) b_i (b_i - b_j - b_k) + F(M_{jk}) 4 b_j b_k \\ & + F(V_j) b_j (b_j - b_i - b_k) + F(M_{ik}) 4 b_i b_k \\ & + F(V_k) b_k (b_k - b_i - b_j) + F(M_{ij}) 4 b_i b_j \end{aligned}$$

where $M_{jk} = (V_j + V_k)/2$, $M_{ik} = (V_i + V_k)/2$ and $M_{ij} = (V_i + V_j)/2$.

A common way to specify a cubic along an edge is to use the Hermite form which involves the first-order directional derivatives along the edges

$$F'_{ki}(V_i) = (x_k - x_i) F_x(V_i) + (y_k - y_i) F_y(V_i)$$

which are further illustrated in Figure 20.40.

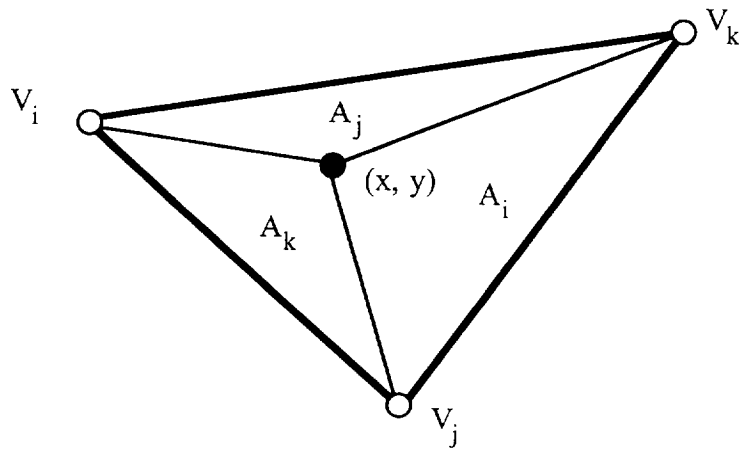


Figure 20.39: Areas leading to barycentric coordinates.

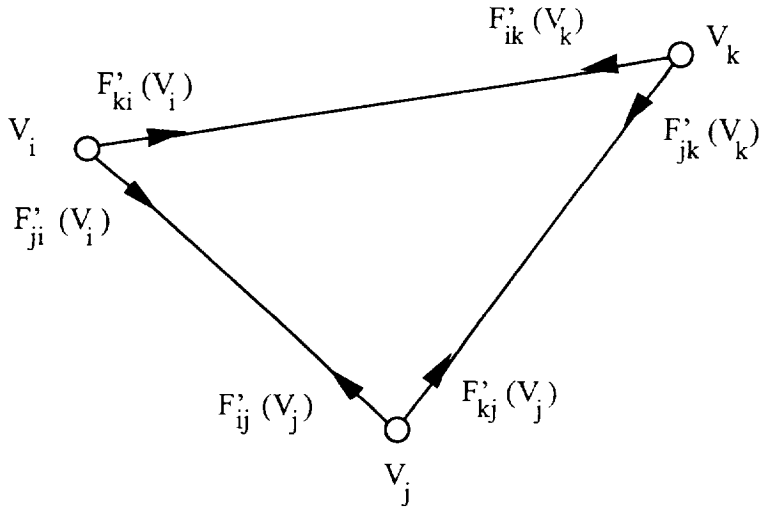


Figure 20.40: The notation for the six directional derivatives.

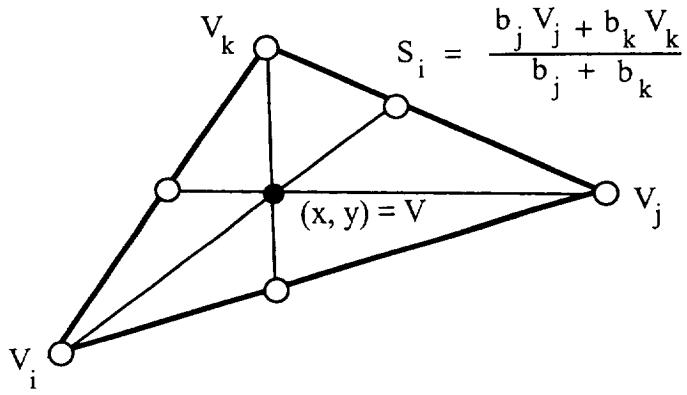


Figure 20.41: The side-vertex interpolant notation.

The six directional derivatives at the three vertices along with $F(V_i)$, $F(V_j)$ and $F(V_k)$ do not uniquely determine a cubic since the bivariate cubics are of dimension 10. The interpolant

$$\begin{aligned} C(x,y) \;=\; & F(V_i)b_i^2 + F'_{ki}(V_i)b_i^2 b_k + F'_{ji}(V_i)b_i^2 b_j \\ & + F(V_j)b_j^2 + F'_{ij}(V_j)b_j^2 b_i + F'_{kj}(V_j)b_j^2 b_k \\ & + F(V_k)b_k^2 + F'_{ik}(V_k)b_k^2 b_i + F'_{jk}(V_k)b_k^2 b_j \\ & + w b_i b_j b_k \end{aligned}$$

will match this function and derivative data for any value of w. This remaining degree of freedom represented by w can be absorbed by a variety of conditions. For example, it can additionally be required that the interpolant match some predescribed value at the centroid. Another common choice is

$$\begin{aligned} w \;=\; & 2[F(V_i) + F(V_j) + F(V_k)] \\ & + \frac{1}{2}[F'_{ki}(V_i) + F'_{ji}(V_i) + F'_{ij}(V_j) + F'_{kj}(V_j) + F'_{ik}(V_k) + F'_{jk}(V_k)] \end{aligned}$$

which guarantees quadratic precision and is a result of discretization of a number of transfinite interpolants (see [189]). Quadratic precision means that whenever the data comes from a bivariate quadratic function the interpolant will become this very same quadratic polynomial.

C^0, Transfinite Interpolation in Triangles

In this section, we give only a sampling of three interpolants which will interpolate to arbitrary function values on the boundary of a triangular domain, $T = T_{ijk}$. More information on this general topic can be found in [189].

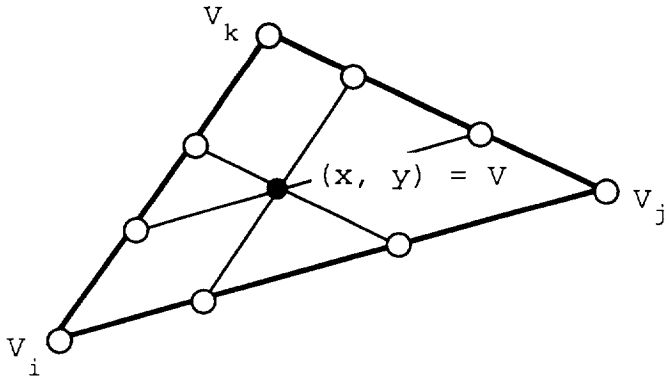


Figure 20.42: The evaluation points (stencil) for side-side interpolant.

The Side-Vertex Interpolant: The side-vertex interpolant is built from three basic interpolants which are defined by linear interpolation along line segments joining a vertex and the opposing side. See Figure 20.41. In terms of barycentric coordinates, we have

$$\begin{aligned} A_i[F] &= b_i F(V_i) + (1-b_i)F(S_i), \\ A_j[F] &= b_j F(V_j) + (1-b_j)F(S_j), \\ A_k[F] &= b_k F(V_k) + (1-b_k)F(S_k) \end{aligned}$$

where $S_i = \frac{b_j V_j + b_k V_k}{b_j + b_k}$, $S_j = \frac{b_i V_i + b_k V_k}{b_i + b_k}$, $S_k = \frac{b_i V_i + b_j V_j}{b_i + b_j}$. Each of these interpolants will interpolate to arbitrary function values on one edge of the triangular domain. In order to obtain an interpolant which matches arbitrary values on the entire boundary of T_{ijk}, we form the Boolean sum of these three interpolants:

$$\begin{aligned} A[F] &= A_i \oplus A_j \oplus A_k[F] = A_i[F] + A_j[F] + A_k[F] \\ &\quad -A_i[A_j[F]] - A_j[A_k[F]] - A_k[A_j[F]] + A_i[A_j[A_k[F]]] \\ &= (1-b_i)F(S_i) + (1-b_j)F(S_j) + (1-b_k)F(S_k) \\ &\quad -b_i F(V_i) - b_j F(V_j) - b_k F(V_k) \end{aligned}$$

The Side-Side Interpolant: The side-side interpolant (Figure 20.42) is based upon the basic operation of linear interpolation along edges which are parallel to the edges of T_{ijk}. There are three of these interpolants:

$$\begin{aligned} P_i[F] &= \frac{b_k F(b_i V_i + (1-b_i)V_k + b_j F(b_i V_i + (1-b_i)V_j}{b_k + b_j} \\ P_j[F] &= \frac{b_i F(b_j V_j + (1-b_j)V_i + b_k F(b_j V_j + (1-b_j)V_k}{b_i + b_k} \\ P_k[F] &= \frac{b_i F(b_k V_k + (1-b_k)V_i + b_j F(b_k V_k + (1-b_k)V_j}{b_i + b_j} \end{aligned}$$

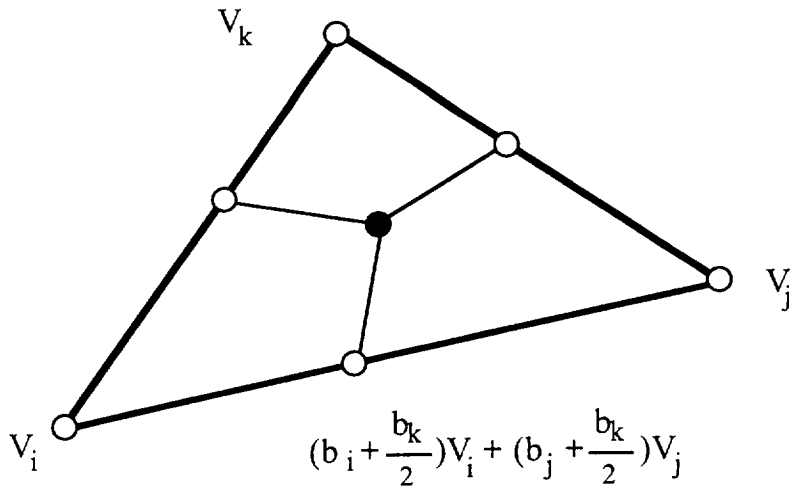


Figure 20.43: The stencil of the C^* interpolant.

Unlike the basic interpolants of the side-vertex interpolant, these interpolants do not commute and so their triple Boolean sum is not well defined. However, it is possible to form the average of all double Boolean sums (each of which interpolates to the entire boundary) to arrive at the following affine invariant interpolant

$$\begin{aligned} Q^*[F] \;=\; & \frac{b_k F(b_i V_i + (1-b_i)V_k) + b_j F(b_i V_i + (1-b_i)V_j)}{b_k + b_j} \\ & + \frac{b_i F(b_j V_j + (1-b_j)V_i) + b_k F(b_j V_j + (1-b_j)V_k)}{b_i + b_k} \\ & + \frac{b_i F(b_k V_k + (1-b_k)V_i) + b_j F(b_k V_k + (1-b_k)V_j)}{b_i + b_j} \\ & - b_i F(V_i) - b_j F(V_j) - b_k F(V_k). \end{aligned}$$

The C* Interpolant: The third and final transfinite, C^0 interpolant we describe utilizes the stencil illustrated in Figure 20.43.

$$\begin{aligned}
C^*[F](b_i,b_j,b_k) \;=\; & \frac{b_ib_j}{\left(b_i+\frac{b_k}{2}\right)\left(b_i+\frac{b_k}{2}\right)}F\left(\left(b_i+\frac{b_k}{2}\right)V_i+\left(b_j+\frac{b_k}{2}\right)V_j\right)\\
&+\frac{b_ib_k}{\left(b_i+\frac{b_j}{2}\right)\left(b_k+\frac{b_j}{2}\right)}F\left(\left(b_i+\frac{b_j}{2}\right)V_i+\left(b_k+\frac{b_j}{2}\right)V_k\right)\\
&+\frac{b_jb_k}{\left(b_j+\frac{b_i}{2}\right)\left(b_k+\frac{b_i}{2}\right)}F\left(\left(b_j+\frac{b_i}{2}\right)V_j+\left(b_k+\frac{b_i}{2}\right)V_k\right)\\
&-\frac{2b_ib_jb_k}{(b_j+2b_k)(b_k+2b_j)}F(V_i)\\
&-\frac{2b_ib_jb_k}{(b_i+2b_k)(b_k+2b_i)}F(V_j)\\
&-\frac{2b_ib_jb_k}{(b_i+2b_j)(b_j+2b_i)}F(V_k)
\end{aligned}$$

which can be written in the form

$$\begin{aligned}
C^*[F](b_i,b_j,b_k) \;=\; & b_iF(V_i)+b_jF(V_j)+b_kF(V_k)\\
&+W_k\left\{F(Q_k)-\left(b_i+\frac{b_k}{2}\right)F(V_i)-\left(b_j+\frac{b_k}{2}\right)F(V_j)\right\}\\
&+W_j\left\{F(Q_j)-\left(b_i+\frac{b_j}{2}\right)F(V_i)-\left(b_k+\frac{b_j}{2}\right)F(V_k)\right\}\\
&+W_i\left\{F(Q_i)-\left(b_j+\frac{b_i}{2}\right)F(V_j)-\left(b_k+\frac{b_i}{2}\right)F(V_k)\right\}
\end{aligned}$$

where

$$W_i=\frac{4b_jb_k}{(2b_j+b_i)(2b_k+b_i)},\quad W_j=\frac{4b_ib_k}{(2b_i+b_j)(2b_k+b_j)},\quad W_k=\frac{4b_ib_j}{(2b_i+b_k)(2b_j+b_k)}$$

$$\begin{aligned}
Q_i &= \left(b_j+\frac{b_i}{2}\right)V_j+\left(b_k+\frac{b_i}{2}\right)V_k,\\
Q_j &= \left(b_i+\frac{b_j}{2}\right)V_i+\left(b_k+\frac{b_j}{2}\right)V_k,\\
Q_k &= \left(b_i+\frac{b_k}{2}\right)V_i+\left(b_j+\frac{b_k}{2}\right)V_j.
\end{aligned}$$

In this form of C* we can see that it consists of linear interpolation plus a correction term. It can easily be verified that C* is precise for all quadratic functions. That is, if f is a quadratic, bivariate polynomial, then $C^*[f]=f$.

C^1, Discrete Interpolation in Triangles

A commonly used 9-parameter, C^1 interpolant is

$$\begin{aligned} C_\Delta[F](x,y) \;=\; & \sum_{(i,j,k)\in I} \{F(V_i)\left[b_i^2(3-2b_i)+6wb_i(b_k\alpha_{ij}+b_j\alpha_{ik}\right] \\ & +F'_{ki}(V_i)\left[b_i^2b_k+wb_i(3b_k\alpha_{ij}+b_j-b_k)\right] \\ & +F'_{ji}(V_i)\left[b_i^2b_j+wb_i(3b_j\alpha_{ik}+b_k-b_j)\right]\}, \end{aligned}$$

where

$$F'_{ki}(V_i) = (x_k - x_i)F_x(V_i) + (y_k - y_i)F_y(V_i),$$

$$F'_{ji}(V_i) = (x_j - x_i)F_x(V_i) + (y_j - y_i)F_y(V_i),$$

$$w = \frac{b_ib_jb_k}{b_ib_j + b_ib_k + b_jb_k}, \qquad I = \{(i,j,k),(j,k,i),(k,i,j)\},$$

and

$$\alpha_{ij} = \frac{\|e_{jk}\|^2 + \|e_{ik}\|^2 - \|e_{ij}\|^2}{2\|e_{ik}\|^2}$$

We use $\|e_{ij}\|$ to denote the length of edge e_{ij}. This 9-parameter, C^1 interpolant is a discretized version of a transfinite, C^1, triangular interpolant, which is described in [178]. The derivatives which are in a direction perpendicular to an edge vary linearly along an edge. This guarantees that when two of these interpolants share a common edge the two surface patches will join with continuous first-order derivatives. It is possible to discretize the same transfinite interpolant and use an additional three parameters consisting of cross-boundary derivatives at the midpoints of the three edges. This leads to an interpolant that has all first-order derivatives varying quadratically along the edges. For a comparison of the C_Δ interpolant to the Clough/Tocher interpolant within the context of triangle-based scattered-data models, see Franke and Nielson [97].

C^1, Transfinite Interpolation in Triangles

In this section, we extend the problem of interpolating to transfinite data on the boundary to include also the requirement that the interpolant match user-specified transfinite derivative data on the boundary. These types of interpolants can be used to construct surfaces over triangulated domains which are C^1—that is, functions which have continuous first-order partial derivatives. One of the most versatile and easily described C^1, transfinite interpolants is the C^1, side-vertex interpolant [177].

Earlier, we saw that the basic building blocks of the C^0, side-vertex interpolant consisted of linear interpolation along lines joining a vertex and its opposing side. In order to extend these ideas to C^1 data, we make use of the univariate cubic, Hermite interpolation applied along rays emanating from a vertex and joining to the opposing edge. See

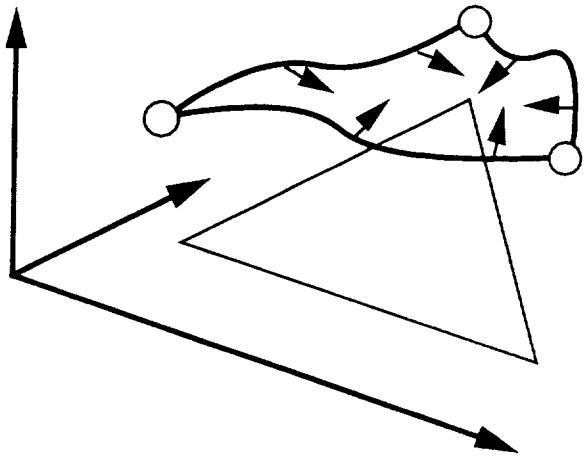


Figure 20.44: The data for C^1 interpolants position and derivative boundary values.

Figure 20.41. Cubic Hermite interpolation will match position and derivatives at the two ends of the interval. We assume that position and derivative information is available on the entire boundary of a triangle T_{ijk}.

$$\begin{aligned} S_i[F](p) &= b_i^2(3-2b_i)F(V_i) + b_i^2(b_i-1)F'(V_i) \\ &\quad +(1-b_i)^2(2b_i+1)F(S_i) + b_i(1-b_i)^2F'(S_i) \end{aligned}$$

where $F'(V_i) = \dfrac{(x-x_i)F_x(V_i)+(y-y_i)F_y(V_i)}{1-b_i}$ and

$F'(S_i) = \dfrac{(x-x_i)F_x(S_i)+(y-y_i)F_y(S_i)}{1-b_i}$. $S_i[F]$ has the property that it interpolates to the boundary data provided by F at V_i and on the entire opposing edge e_{kj}. It also matches first-order derivatives on this edge and at V_i. It does not necessarily interpolate F or its derivatives on the other two edges. In order to have an interpolant for the entire boundary of the triangular domain, we could try to construct one using the ideas of Boolean sums as was done earlier for the C^0, side-vertex interpolant. Even though the interpolants S_i, S_j, and S_k commute so that their Boolean sums are well defined, this approach does not work (see [177]) and so the use of convex combination techniques has been suggested. This leads to the interpolant

$$S[F] = \frac{b_j^2b_k^2S_i[F] + b_i^2b_k^2S_j[F] + b_j^2b_i^2S_k[F]}{b_i^2b_j^2 + b_j^2b_k^2 + b_i^2b_k^2}$$

which has the property that it matches F and its first order derivatives on the entire boundary of the triangular domain. In the case where the boundary information has been discretized with cubically varying (Hermite) position values and linearly varying cross-boundary derivatives, it is possible to obtain a final interpolant with simpler weights in the convex combination. Namely,

$$S[F] = \frac{b_jb_kS_i[F] + b_ib_kS_j[F] + b_jb_iS_k[F]}{b_ib_j + b_jb_k + b_ib_k}.$$

20.3 Tetrahedrizations

In this section we follow the outline of the previous section as well as possible. Since the dimension is one less and since bivariate problems have been considered for a much longer period of time, the development in the 3D domain is not as rich as it is in the 2D domain. So we cannot parallel the previous section exactly, but most everything generalizes or leads to something interesting and often useful.

20.3.1 Basics

Definitions, Data Structures, and Formulas for Tetrahedrizations

Our definition of a tetrahedrization follows very closely to that given for a triangulation at the beginning of Section 20.2.1. We start with a collection of points $p_i = (x_i, y_i, z_i), i = 1, \ldots, N$ which we assume are not collectively coplanar. We denote this collection of points by P. A tetrahedrization consists of a list of 4-tuples which we denote by I_t. Each 4-tuple, $ijkl \in I_t$ denotes a single tetrahedron with the four vertices p_i, p_j, p_k, and p_l. The following conditions must hold:

i) No tetrahedron $T_{ijkl}, ijkl \in I_t$ is degenerate. That is, if $ijkl \in I_t$ then p_i, p_j, p_k, and p_l are not coplanar.

ii) The interiors of any two triangles do not intersect. That is, if $ijkl \in I_t$ and $\alpha\beta\gamma\delta \in I_t$ then $\text{Int}(T_{ijkl}) \cap \text{Int}(T_{\alpha\beta\gamma\delta}) = \emptyset$.

iii) The boundary of two triangles can intersect only at a common triangular face.

iv) The union of all the triangles is the domain $\text{D} = \cup_{ijkl \in I_t} T_{ijkl}$.

We should point out that condition iii) must hold in the strictest sense, and so tetrahedra joining as shown on the right side of Figure 20.45 are not allowed. The reason for this condition (and all the others) is the same as the reason for the conditions of a triangulation; that is, we eventually wish to be able to define C^0 functions in a piecewise manner over the domain consisting of the union of all tetrahedra. The triangular grid data structure for representing triangulations (illustrated in Figure 20.8) generalizes very nicely to a structure for representing tetrahedrizations. For example, in Figure 20.46, we show a tetrahedrization of the cube into five tetrahedra. We saw earlier in the case of triangulations that once the boundary is specified, the number of triangles comprising the triangulation was fixed, and moreover, we had a simple approach for determining a formula for the number of triangles that existed in the triangulation. This property allowed for the definition of the vectors of angles which lead to the criterion for optimal triangulations and was therefore rather important. It would be nice if everything extended to 3D in a straightforward manner. That is, we would like to say that any polyhedron can be decomposed into tetrahedra and that there is a fixed formula of the following form $N_t = aN_b + bN_i + c$, whereas before, N_b and N_i are the number of vertices on the boundary and interior, respectively. Unfortunately, this is not the case and, in fact, the situation is much worse than that. We saw earlier that any polygon-bounded region can be triangulated using only the vertices of the polygon. This is one of the first areas where matters differ significantly when going from 2D to 3D. It turns

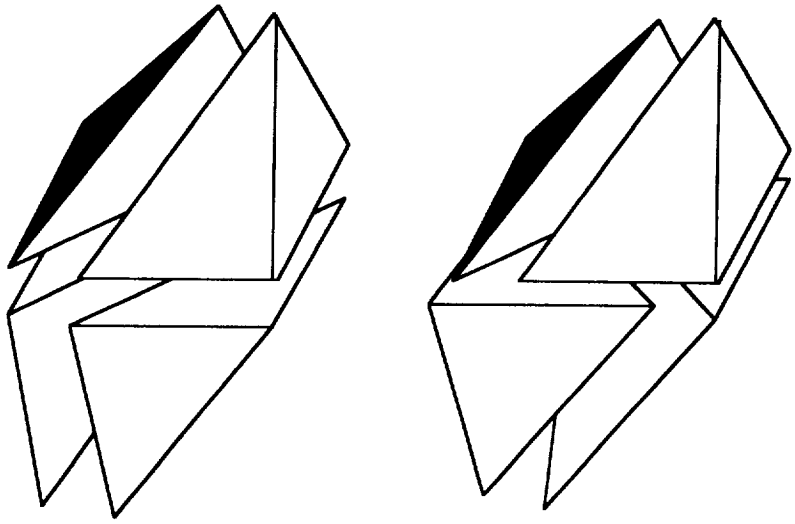
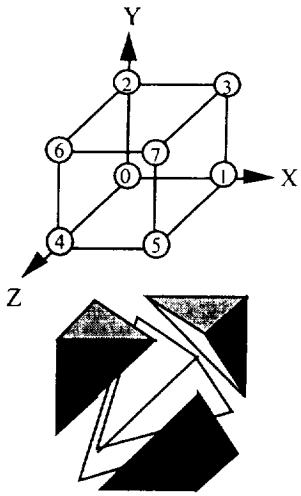


Figure 20.45: The configuration indicated by the diagram on the left is acceptable, while that on the right is not acceptable for a tetrahedrization. It is eliminated by condition iii) listed above.



Tetrahedra				Neighbors			
p_i	p_j	p_k	p_l	N_{jkl}	N_{ikl}	N_{ijl}	N_{ijk}
7	4	5	1	ϕ	ϕ	5	ϕ
7	1	3	2	ϕ	ϕ	5	ϕ
2	4	0	1	ϕ	ϕ	5	ϕ
7	2	6	4	ϕ	ϕ	5	ϕ
2	4	7	1	1	2	3	4

Figure 20.46: An example which defines the tetrahedral grid data structure.

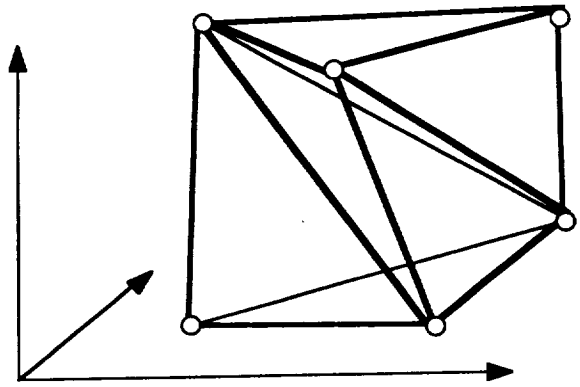


Figure 20.47: The twisted prism of Schoenhardt [221], which cannot be tetrahedrized.

out that not every polyhedron can be tetrahedrized. The example illustrated in Figure 20.47 is originally due to Schoenhardt [221]. It can be visualized as a prism which has been twisted until each face (a quadrilateral comprised of two triangles) has "buckled" inward. Any tetrahedron we form from these vertices must include an edge which lies outside the domain of the "twisted prism," so it is clear that the object cannot be tetrahedrized.

One very basic operation does carry over in a straightforward manner from 2D to 3D—the process of inserting an additional vertex into the interior of an existing tetrahedrization. See Figure 20.48. If the new vertex p lies interior to an existing tetrahedron, say T_{abcd}, then this tetrahedron is simply replaced with the four tetrahedron, $T_{abcp}, T_{abdp}, T_{bcdp}, T_{acdp}$, adding a net increase of three tetrahedra. If the new vertex p lies on the common triangular face of two tetrahedra, then these two tetrahedra are replaced with six new tetrahedra, $T_{abcp}, T_{bcdp}, T_{abdp}, T_{aecp}, T_{ecdp}, T_{aedp}$, resulting in a net increase of four new tetrahedra. This latter aspect of the number of tetrahedra increasing which is different here from the 2D case is that net increase in the number of tetrahedra depends on the actual location of the interior point to be inserted. This observation points out that not only can the number of ways that a data set is tetrahedrized vary, but also the number of tetrahedra can vary. We will illustrate this further with some examples that do not even have interior points.

We have already seen (Figure 20.46) the decomposition of a cube into five tetrahedra. It is also possible to tetrahedrize the cube into six tetrahedra. This is illustrated in Figure 20.49.

It is interesting to note that from the exterior, the tetrahedrization of Figure 20.49 looks exactly the same as that of Figure 20.46 because all external edges are the same. Another interesting connection between these two tetrahedrizations of the cube is that one can be obtained from the other by "swapping" operations, similar to those used in the Lawson algorithm for computing optimal triangulations. Previously, in the case of triangulations, there was the possibility of two triangulations of a convex quadrilateral. The analogous situation in 3D is the tetrahedrization of the region formed by five vertices when two tetrahedra meet at a common triangular face. If the line segment joining the two vertices not on the common face intersects the interior of the common face, then, analogous to the convex quadrilateral case in 2D, there is the possibility of an alternate tetrahedrization. But what is

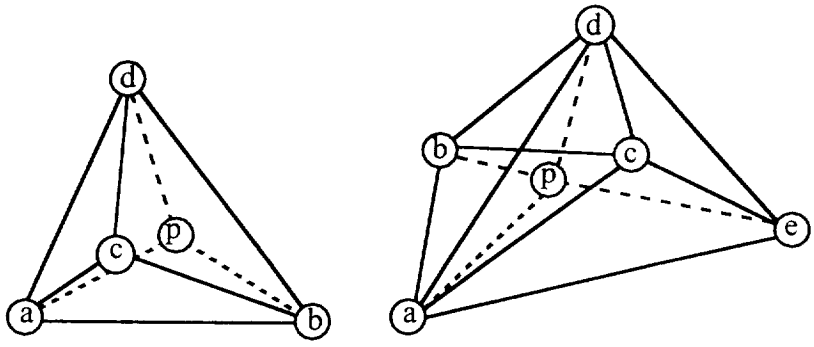
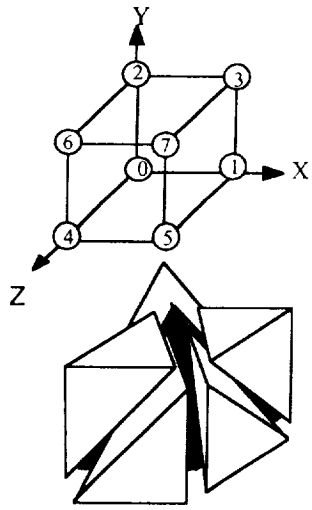


Figure 20.48: Inserting a point interior to an existing tetrahedrization. On the left, the new point is interior to a tetrahedron, and on the right it is on a common face of two tetrahedra.



Tetrahedra				Neighbors			
p_i	p_j	p_k	p_l	N_{jkl}	N_{ikl}	N_{ijl}	N_{ijk}
4	7	6	2	ϕ	ϕ	2	ϕ
4	5	7	2	4	1	3	ϕ
4	1	5	2	4	2	6	ϕ
5	1	7	2	5	2	3	ϕ
1	3	7	2	ϕ	4	ϕ	ϕ
4	1	2	0	ϕ	ϕ	ϕ	3

Figure 20.49: A tetrahedrization of the cube into six tetrahedra.

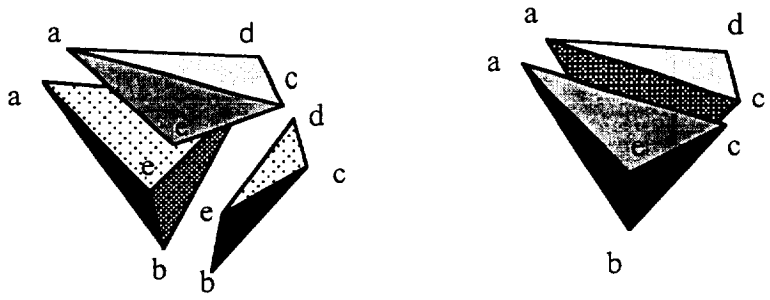


Figure 20.50: Two different tetrahedrizations of five points.

really different from the 2D case is that the number of tetrahedra changes from two to three! This is illustrated in Figure 20.50. This basic operation was applied to the center and upper, back-right tetrahedra of Figure 20.46 to arrive at the tetrahedrization of Figure 20.49.

Another example worth noting in this context is the case where $p_i = (i, i^2, i^3), i = 1, \ldots, N$. The (Delaunay) tetrahedrization of the convex hull of this set of points consists of the tetrahedra with vertices p_i, p_{i+1}, p_j, and p_{j+1}, of which there are a total of $((N-2)(N-1))/2$ tetrahedra. Bern and Eppstein [16] point out that this example provides an upper bound on the number of tetrahedra in a tetrahedrization of an N-vertex polyhedron, and that a lower bound is provided by the fact that any tetrahedrization of a simple polyhedron has at least $N - 3$ tetrahedra.

Some Special Tetrahedrizations

Following the pattern established in the earlier sections on triangulations, we first discuss tetrahedrizations related to Cartesian grids followed by tetrahedrizations associated with curvilinear grids. A 3D Cartesian grid (Figure 20.51) involves three monotonically increasing sequences, $x_i, i = 1, \ldots, N_x$, $y_j, j = 1, \ldots, N_y$ and $z_k, k = 1, \ldots, N_z$. The grid points have coordinates (x_i, y_j, z_k) and these points mark out a cellular decomposition of the domain consisting of regular parallelepipeds. Each of these cells can be tetrahedrized in a manner similar to that given for the cube in the previous section. Probably the most popular is the tetrahedrization involving five tetrahedra shown in Figure 20.46. So as to not end up with a nontetrahedrization with problems similar to those shown on the right side of Figure 20.45, it is necessary to "alternate" the tetrahedrization from one cell to the next so that adjoining cells have the same diagonal on the common faces. This alternate tetrahedrization is not really different but is just a rotation of its companion. It is shown in Figure 20.52. Another popular choice is the tetrahedrization shown in the upper-left corner of Figure 20.53. It has the advantage that all of the tetrahedra are the same shape (up to mirror images). Actually, it turns out that there are six different tetrahedrizations of a cube (parallelepiped). See Nielson [183]. We have previously shown pictures of two of them in Figure 20.46 and Figure 20.49. The other four are shown in Figure 20.53.

All six tetrahedrizations of the cube are comprised of five primitive tetrahedra, which are shown in Figure 20.54. We use the names 0F, 1F, 2Fr, 2Fl and 3F for these tetrahedra to

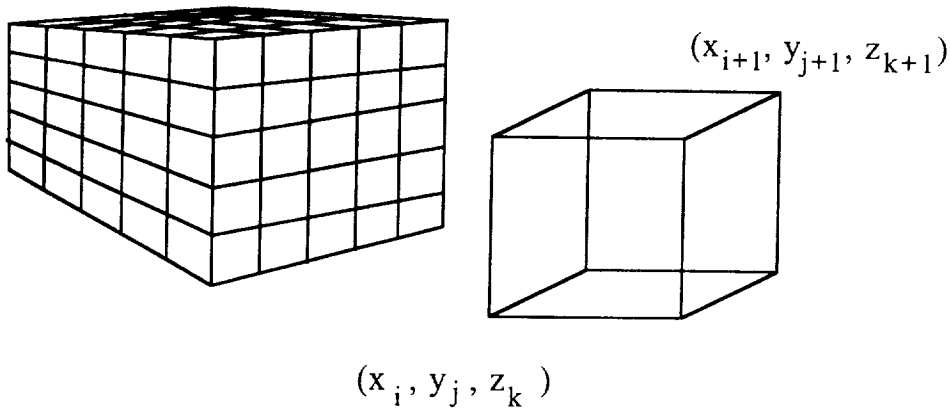


Figure 20.51: Three-dimensional Cartesian grid.

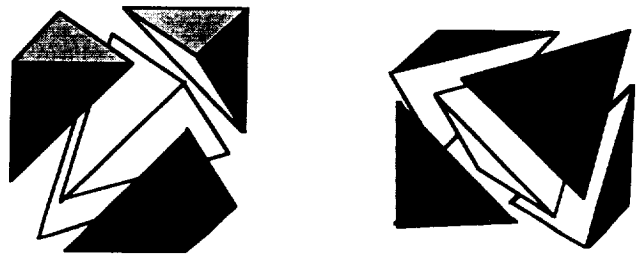


Figure 20.52: The two alternating tetrahedrizations with five tetrahedra of the cell of a 3D Cartesian grid. (One can be rotated to the other.)

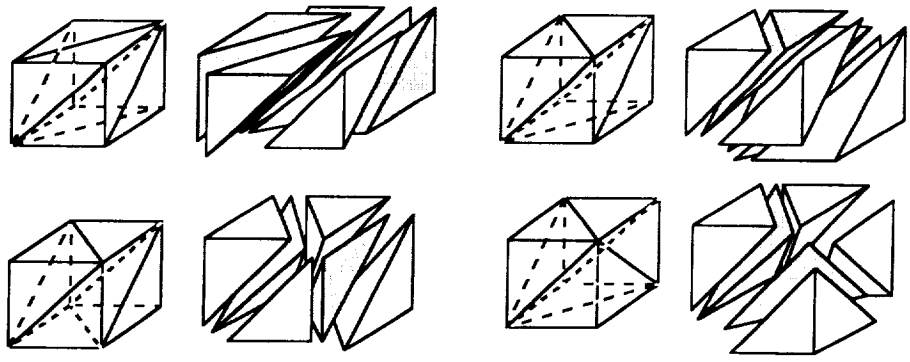


Figure 20.53: Four different tetrahedrizations of the cube, each with six tetrahedra.

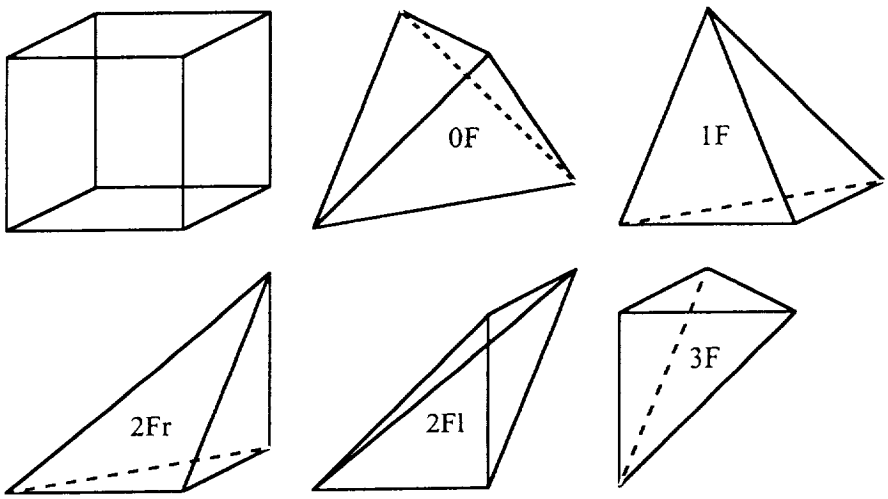


Figure 20.54: The five primitive tetrahedra comprising the tetrahedrizations of the cube.

indicate the number of exterior faces for each tetrahedra. There are two different primitive tetrahedra with two exterior faces; one is a mirror-image version of the other and so it cannot be rotated to the other. The tetrahedron 0F has volume $1/3$ and all the others have volume $1/6$. During informal discussion we most often use the names 3F = "corner," 2Fr or 2Fl = "right wedge" or "left wedge," 1F = "kite" and 0F = "equi" or "fatboy."

In a joining similar to that shown in Figure 20.50, three 1F tetrahedra can come together to form the same exact shape formed by a 0F and a 3F together. Also a 2Fl and 2Fr together form the same shape as a 1F and a 3F, but two 2Fr's or two 2Fl's cannot share a common face and remain inside a unit cube. There are four tetrahedrizations (each composed of three primitive tetrahedra) of the prism making up half of the cube. They are 3F, 1F, 2Fl; 3F, 1F, 2Fr; 2Fr, 2Fl, 2Fr; and 2Fl, 2Fr, 2Fl. In Figure 20.55 we show the dual graphs of the six tetrahedrizations of the cube. A node is a primitive tetrahedron and an arc is a common triangular face. As expected, in each case the "names" add to twelve.

Each of these six tetrahedrizations has unique and interesting properties. The tetrahedrization of Figure 20.46 and Figure 20.49 both "swap" diagonals on all three pairs of opposing faces. The tetrahedrization shown in the lower right of Figure 20.53 swaps the diagonals of two pair of opposing faces and that of the upper right swaps one pair. The two tetrahedrizations on the left of Figure 20.53 do not swap any diagonals of any opposing faces. The tetrahedrization of the upper left of Figure 20.53 can be realized with three cuts of the entire cube, while the others cannot. This particular tetrahedrization also has the unique property of being composed of only 2F primitives whose faces are all right triangles, and all six of them share the diagonal of the cube as a common edge. This tetrahedrization has been discussed and used widely. It is called the CFK triangulation of the cube after Coxeter [47], Freudenthal [79], and Kuhn [137]. A replacement rule can be used to generate this tetrahedrization. Using the labeling scheme of Figure 20.46, we start with the four vertices $P_{2^i-1}, i = 0, 1, 2, 3$ and replace each vertex V_j, other than V_0 and

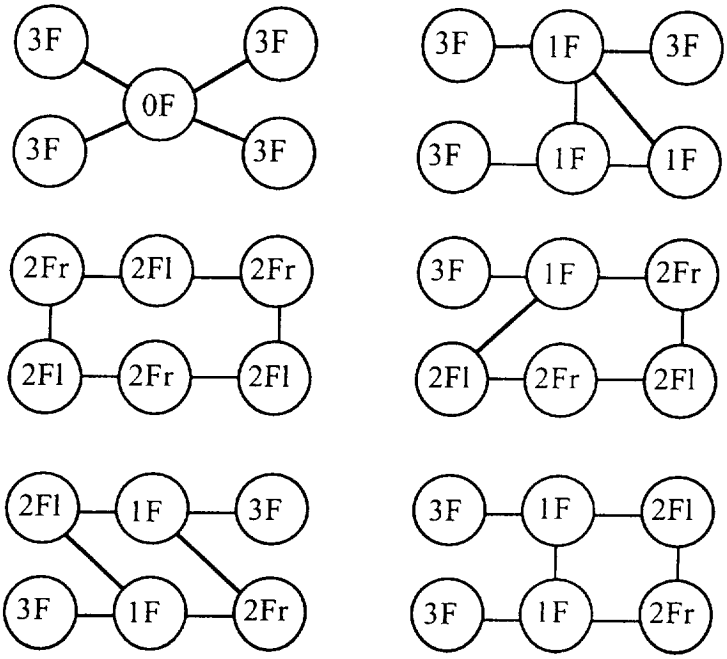


Figure 20.55: The six tetrahedrizations of the cube shown as dual graphs. (These are the only tetrahedrizations of the cube.)

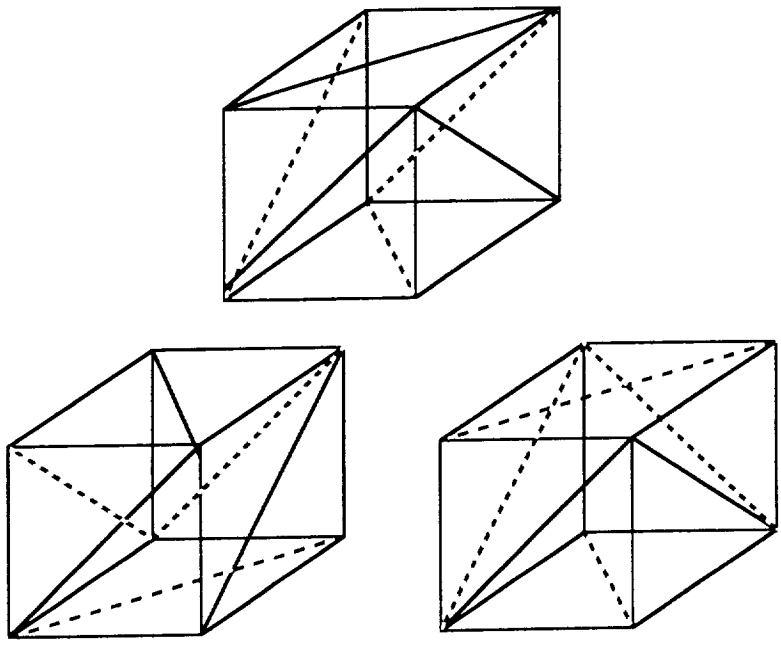


Figure 20.56: Face triangulations that are not consistent with any tetrahedrization of the cube.

V_7, with $V_{j+1} + V_{j-1} - V_j$. Explicitly, this will successively generate the six tetrahedra: $p_0p_1p_3p_7$; $p_0p_2p_3p_7$; $p_0p_2p_6p_7$; $p_0p_4p_6p_7$; $p_0p_4p_5p_7$; $p_0p_1p_5p_7$. The CFK triangulation generalizes to n-dimensions as does the "replacement" algorithm for generating the simplicial decomposition.

It is interesting to note that not all possible face triangulations are realized by the six possible tetrahedrizations of the cube. In addition to the five different face triangulations which are realizable (note that two tetrahedrizations have the same face triangulations) there are three others which cannot be realized. They are shown in Figure 20.56. In order to determine these eight unique face triangulations, we start with the $64 = 2^6$ face triangulations and then group them into these eight equivalence classes by rotations.

Theorem: It is impossible to tetrahedrize a cube and yield face triangulations as shown in Figure 20.56.

Proof: We give only the proof for the case in the top center, as the others are similar. We use the same labeling as shown in Figure 20.49. We start with the face 457. Only vertex 0 can be attached to the face 457, which gives the tetrahedron 0457. The internal face 047 must be shared by some other tetrahedron. Any vertex, however, cannot be joined to the face of 457 without violating the conditions of the face triangulations, so this completes the argument.

Earlier we discussed triangulations related to curvilinear grids. We now take up the

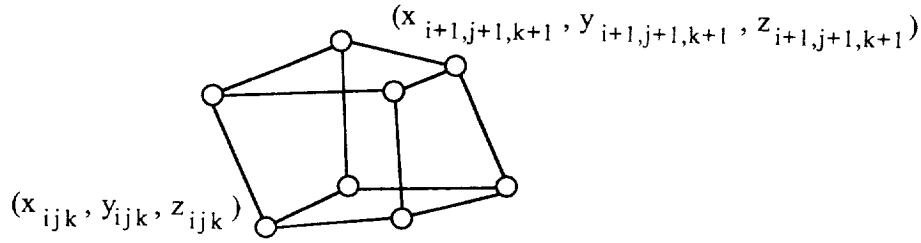


Figure 20.57: Single cell of a 3D curvilinear grid.

topic of tetrahedrization of 3D curvilinear grids. Analogous to the 2D situation, a 3D curvilinear grid is specified by three geometry arrays $x_{ijk}, y_{ijk}, z_{ijk}$, $i = 1, \ldots, N_x$; $j = 1, \ldots, N_y$; $k = 1, \ldots, N_z$. In the 2D case a cell C_{ij} consisted of the quadrilateral with vertices $(x_{ij}, y_{ij}), (x_{i+1,j}, y_{i+1,j}), (x_{i,j+1}, y_{i,j+1}), (x_{i+1,j+1}, y_{i+1,j+1})$, and the cells serve as a decomposition of the domain.

In the 3D case, matters are not as straightforward as we might expect, and there are some areas where we need to be concerned. These have mainly to do with just exactly what comprises a cell. In 3D the cell C_{ijk} has the eight vertices $(x_{abc}, y_{abc}, z_{abc}), a = i, i+1, b = j, j+1, c = k, k+1$, but there is not always a consistent definition for the cell boundaries. We mention briefly some possible choices. If the geometry arrays are constrained so that each collection of four vertices of the six "faces" of the cells are coplanar, then an obvious choice for the cell boundaries is this common planar quadrilateral. In this case the cells are hexahedron, and it is relatively easy to determine whether or not an arbitrary point (x, y, z) is in a particular cell or not. Often this planarity condition does not hold, and cell boundaries are taken to be the parametrically defined (hyperbolic) surface obtained by substituting 0 or 1 for any of the parameter values s, t, u in the following trilinear mapping:

$$\begin{aligned} C_{i,j,k}(s,t,u) \;=\; & (1-s)(1-t)(1-u)P_{i,j,k} + (1-s)(1-t)uP_{i,j,k+1} \\ & +(1-s)t(1-u)P_{i,j+1,k} + (1-s)tuP_{i,j+1,k+1} \\ & +s(1-t)(1-u)P_{i+1,j,k} + s(1-t)uP_{i+1,j,k+1} \\ & +st(1-u)P_{i+1,j+1,k} + stuP_{i+1,j+1,k+1} \end{aligned}$$

where

$$P_{i,j,k} = (x_{i,j,k}, y_{i,j,k}, z_{i,j,k})$$

Given a point (x, y, z) in the cell C_{ijk}, the value (s, t, u) which associates with it via the trilinear mapping is called the corresponding *computational coordinate*. In fact, in order to determine whether or not an arbitrary point is in this type of cell or not requires that we solve the three nonlinear equations which represent this association. This can be a considerable problem from a computational point of view. Most methods use some heuristics to obtain an initial approximation for some type of Newton's method. Another choice for the cell boundaries, in the event the four vertices of a face are not coplanar, is to choose them to be piecewise planar. That is, a diagonal edge is selected and the

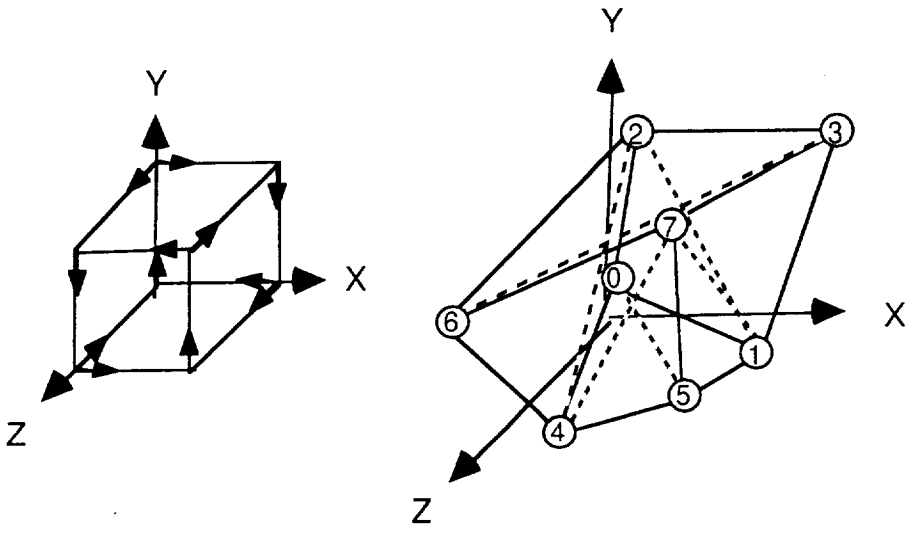


Figure 20.58: A curvilinear grid cell (polyhedron) that cannot be tetrahedrized.

boundary between the two cells consists of the two triangles which result. Often the cell would be further decomposed into tetrahedra, thus leading to an overall tetrahedrization of the curvilinear grid. We should point out that not all choices for the diagonals can lead to a tetrahedrization of the cell. In order to be specific about this, consider the cell illustrated in Figure 20.58. This cell was created from a unit cube by cutting notches in the faces so as to force the diagonal edges $p_2p_7, p_4p_1, p_3p_5, p_3p_0, p_0p_6, p_6p_5$ to be exterior to the cell. If the depth of the notches is e, then this results in the points $p_0 = (0, e, 0)$, $p_1 = (1-e, 0, e)$, $p_2 = (e, 1, e)$, $p_3 = (1, 1-e, 0)$, $p_4 = (e, 0, 1-e)$, $p_5 = (1, e, 1)$, $p_6 = (0, 1-e, 1)$, $p_7 = (1-e, 1, 1-e)$. Note that p_6, p_3, p_4 and p_1 all lie in the plane $x + z - 1 = 0$ and p_2, p_7, p_0 and p_5 are in the plane $x - z = 0$.

Theorem: The polyhedron of Figure 20.58 cannot be tetrahedrized.

Proof: Consider the triangle face with vertices p_6, p_4, and p_7. In any tetrahedrization, this face must be joined to some vertex to form a tetrahedron. By considering the remaining five vertices p_5, p_0, p_2, p_1, and p_3 we find that only p_3 would not lead to a tetrahedron with an edge which is outside the cell. If the tetrahedron p_6, p_4, p_7, and p_3 is included in the list of tetrahedra, then the interior triangle face $p_3p_4p_7$ must connect to another vertex (besides p_6) to form a tetrahedron. But a consideration of each of the possible vertices p_5, p_1, p_2, and p_0 each lead to an edge which is exterior to the cell and this concludes the argument.

We conclude this discussion on the tetrahedrization of the cells of a curvilinear grid by pointing out that some hexahedra will decompose into seven tetrahedra. Consider the cell of Figure 20.57 and let the six faces be planar, but assume that the four diagonal points p_{ijk}, $p_{i+1,j+1,k+1}$, $p_{i,j,k+1}$ and $p_{i+1,j+1,k}$ are not coplanar, so that they will form a tetrahedron. Remove this tetrahedron, leaving two prisms with two planar quadrilateral

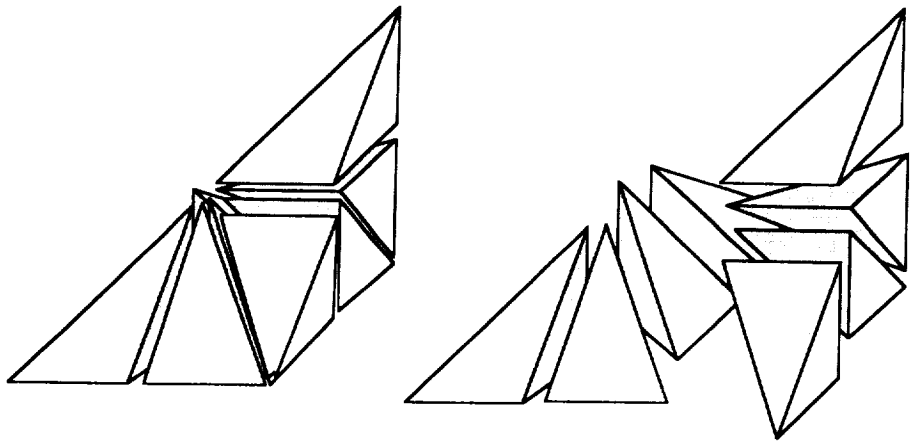


Figure 20.59: Nested tetrahedral subdivision analogous to that of Figure 20.16.

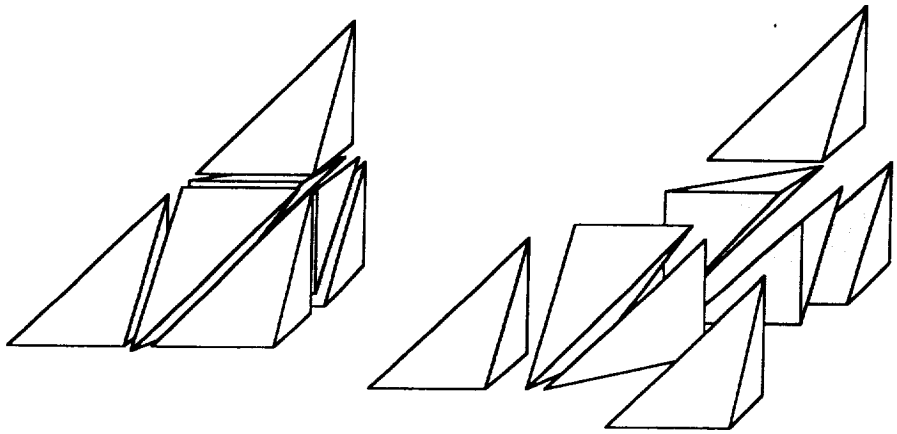


Figure 20.60: Symmetric nested tetrahedral subdivision.

faces which can each be decomposed into three tetrahedra. We should point out that we have observed cases where this decomposition was the Delaunay tetrahedrization.

In Section 20.2.1 we described two different approaches leading to nested subdivision triangulations and pointed out their potential value in multiresolution approximations. These both have analogs in 3D and are shown in Figures 20.59 and 20.60, respectively. The first one is based upon recursive subdivision and the second one is called "symmetric" subdivision and is related to the CFK tetrahedrization of the cube [170]. It is composed of six 2Fr's and two 2Fl's and is the same shape and twice the size of one 2Fr.

It should be noted that if primitive tetrahedra of the shape shown in Figure 20.61 are assembled as in Figure 20.60, then we obtain a composite tetrahedron which is twice the size and exactly the same shape as the primitive tetrahedron. This particular tetrahedrization of

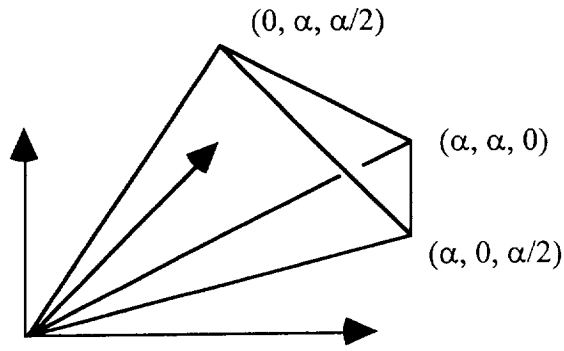


Figure 20.61: A tetrahedron that can be tetrahedrized into eight tetrahedra, each of which are the same shape as the original yet half the size.

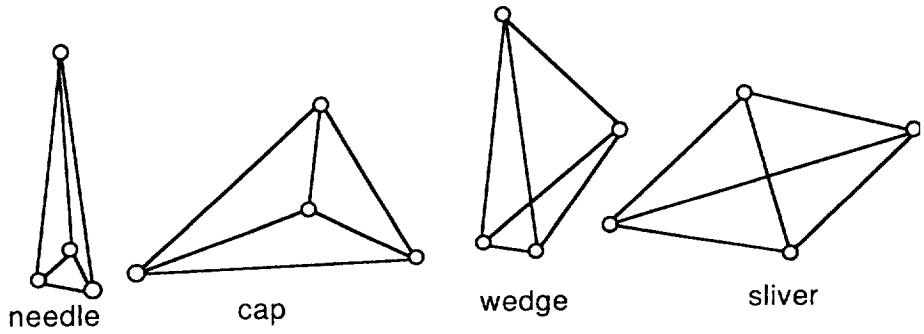


Figure 20.62: Examples of poorly shaped tetrahedra.

tetrahedra is related to the Delaunay tetrahedrization of the BCC lattice, which is the union of the lattices $\{(i, j, k) : i, j, \text{and } k \text{ are integers}\}$ and $\{(i + \frac{1}{2}, j + \frac{1}{2}, k + \frac{1}{2}) : i, j, \text{and } k \text{ are integers}\}$. See also Senechal [229] for a discussion of tetrahedra that can be decomposed into similar tetrahedra.

20.3.2 Algorithms for Delaunay Tetrahedrizations

Analogous to the examples of Figure 20.19, examples of poorly shaped tetrahedra are shown in Figure 20.62. The sliver has small dihedral angles, but need not have any small planar angles. Several measures of the quality of tetrahedrizations have been proposed. See Baler [12] and Field [86]. Take as an example the ratio of the inradius (radius of inscribed sphere) and the circumradius. The problem here is that there is no apparent way to order the collection of all tetrahedrizations of a point set. The approach of lexicographically ordering the associated vectors of angles, as we described in Section 20.2.2, does not extend to 3D because the number of tetrahedra in a tetrahedrization is not necessarily fixed. Nevertheless, the Delaunay tetrahedrization of the convex hull which is dual to the Dirichlet

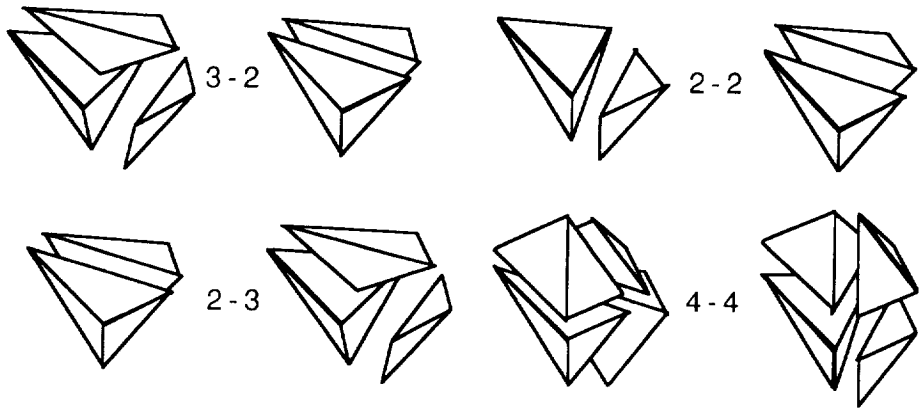


Figure 20.63: Different cases of swapping for 3D version of Lawson's algorithm.

tessellation is well defined (in the absence of neutral cases where points lie on a common sphere), so the remainder of this section is devoted to a discussion of the extension of the previously discussed 2D algorithms for computing the Delaunay triangulations to the case of 3D tetrahedrizations.

Extension of Lawson's Algorithm (Incremental Flipping): It is possible to extend this algorithm to 3D, but the extension is not as simple as one might expect. The first major difference that one encounters is the character of the basic swapping step. In 2D we take an edge and consider the quadrilateral formed by the two triangles which share this edge. If the quadrilateral is convex, we can swap the diagonal if this step moves us closer to the optimal solution, which can easily be determined by applying the circle inclusion test. Two triangles are replaced by two other triangles. But the analogous steps in 3D can lead to a situation where the two tetrahedra sharing a face can be replaced with three tetrahedra. See Figure 20.63 for an example.

Joe [122] showed that if the points are inserted in a particular manner, then incremental flipping will lead to the optimal Delaunay tetrahedrization. Edelsbrunner and Shah have generalized these results [72]. See also [82]. Software based upon these ideas is provided by the Software Development Group at the National Center for Supercomputing Applications and is available at the World Wide Web site:
http://www.ncsa.uiuc.edu/SDG/Brochure/Overview/ALVIS.overview.html.

Extensions of the Algorithm of Green and Sibson: There does not seem to be an apparent method of extending this type of algorithm to 3D. The algorithm is dependent upon the "contiguity list," and here lies the difficulty to extend to 3D. We included this algorithm in our selection of 2D algorithms so that this very point could be made. Some concepts extend easily to 3D and others do not.

Bowyer's Algorithm for 3D: It is a straightforward exercise to extend Bowyer's 2D algorithm to 3D. In fact, the original paper of Bowyer [21] describes the algorithm for

arbitrary dimensions. Bowyer also mentions that with some care, the algorithm can be extended to other domains. In [164] there is a brief discussion of Bowyer's algorithm along with some code.

Watson's Algorithm for 3D: The original description of Watson's algorithm applies to arbitrary dimension. In Watson's paper [254] results for 2, 3, and 4 dimensions are reported. Information on implementing this algorithm in 3D is given by Field in [86] and [87]. It is also the basis for the 3D algorithms discussed in [29].

Embedding/Lifting Algorithms for 3D: Software for computing general dimension convex hulls and Delaunay tetrahedrizations, based on the relationship mentioned earlier in Section 20.2.2, are provided by the Geometry Center, University of Minnesota at the WWW site:
http://freeabel.geom.umn.edu/software/download/qhull.html.

20.3.3 Visibility Sorting of Tetrahedra

We first give a motivation for the definition of and the need for a visibility sort. We use the example of volume rendering, which is a means of graphing (visualizing) a density function (cloud) $\delta(x, y, z)$ defined over a 3D domain (which is often a cube). A viewpoint V is selected along with a projection plane. A rectangular portion of the projection plane is subdivided into a rectangular array of subrectangles which associate directly with the pixels of an image to be generated. The value for each pixel is defined by

$$F(i,j) = \int_0^D \delta(s)C(s)e^{-\int_s^D \delta(u)\,du}\,ds + F_0 e^{-\int_0^D \delta(u)\,du} \tag{20.3}$$

where the integral is taken along the ray emanating from the viewpoint and passing through the center of the subrectangle associated with the pixel at location (i, j), F_0 is the background intensity, and D is a distance along the ray sufficiently large so that the ray completely passes through the domain of interest. The function C, also defined over the same domain as δ, is called the color function and governs the color of light emanating (by reflection, say) from a point within the density cloud. In actual application the integrals are approximated by numerical schemes based upon sampled values of the integrand. The sample values are often obtained by some simple interpolation into the cells covering the domain. And these cells are often a result of the positions where δ has been measured. If we let $0 = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = D$ be the distances from the viewpoint to each sampled value along the ray, then the upper Riemann sum approximation to this integral is

$$F_n = \sum_{i=0}^{n} \Delta x_i \delta(x_i) C_i \prod_{j=i+1}^{n} t_j, \tag{20.4}$$

where $C_i = C(x_i), t_j = e^{-\Delta x_i \delta(x_i)}$ and $\Delta x_i = x_i - x_{i-1}$. This discrete approximation can be computed by the *compositing process*

$$F_i = t_i F_{i-1} + I_i, \tag{20.5}$$

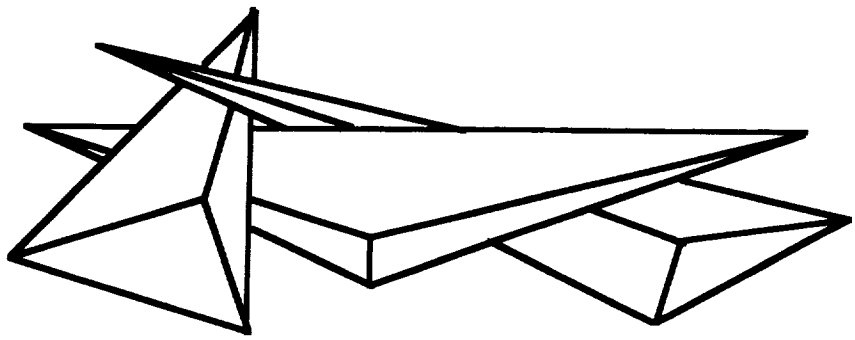


Figure 20.64: An example of three tetrahedra that cannot be visibility ordered.

where $I_i = \Delta x_i \delta(x_i) C_i$.

Another way to view this compositing process is as a simple model of transparency, where an object of thickness Δx_i attenuates the incoming light intensity F_{i-1} by the factor t_i, and this object emits light of intensity I_i. Algorithms which accumulate these values into a frame buffer (with each location holding the value for a pixel) can either be image-space-oriented or object-space-oriented. Image-space algorithms proceed along the lines of our development here and accumulate all contributions for a pixel along a particular ray. Object-space algorithms compute exactly the same values but the calculations are done in a different order. These algorithms sequentially process each cell by accumulating into the proper location of the frame buffer all contributions of a particular cell. Due to the nature of the compositing process, it is mandatory that these accumulations be done in the proper order. It is this latter approach which motivates the definition of and need for visibility sorting in this context.

Definition of Visibility Order: Let T and T' be tetrahedra of a tetrahedrization and let V be the center of perspective projection. If there is a ray emanating from V which intersects T' before T, then T is said to precede T' and we write $T < T'$.

The purpose of a visibility sort is to find a linear ordering of all of the tetrahedra of a tetrahedrization so that the ordering relation is never violated.

Definition of Visibility Ordering: A visibility ordering of a tetrahedrization is a sequence, $n_1, n_2, \ldots, n_T$ which has the property that whenever $T_{n_i} < T_{n_j}$ then $i < j$. The implication of the definition of visibility ordering for splatting or object-space traversal algorithms for volume rendering is that a tetrahedron T must be processed (sampled and composited into the frame buffer) before T' whenever $T < T'$.

A couple of items should be noted at this point. The relation of visibility order is not, in the strict mathematical sense, a partial ordering. A partial ordering is required to be i) transitive: $x < y$, $y < z$ implies $x < z$; ii) antisymmetric: $x < y$ and $y < x$ implies $x = y$; and iii) reflexive: $x < x$. It is entirely possible that a visibility order could not exist at all due to the presence of cycles as shown in Figure 20.64.

Knuth [136] has discussed in some detail (including MIX programs) the topological sort algorithm as a means of "embedding a partial order in a linear order." A linear ordering

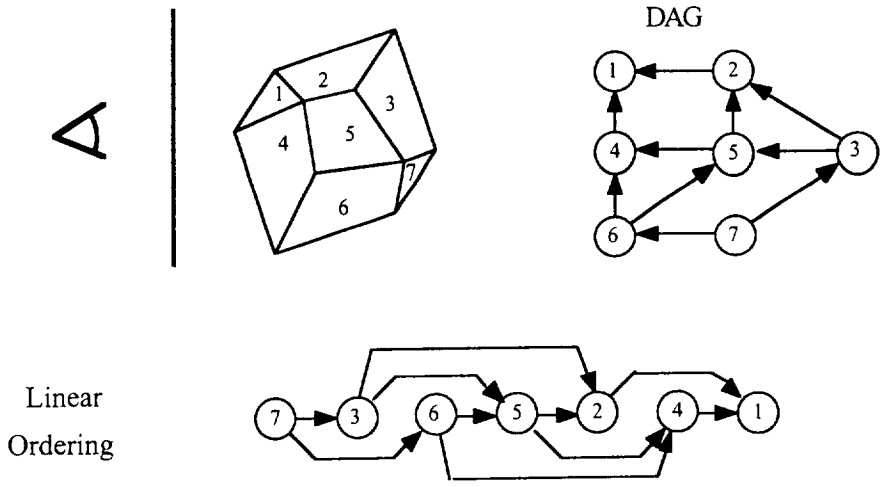


Figure 20.65: An example of the topological sort algorithm.

is a partial ordering where either $x < y$ or $y < x$ for all x, y. Even though this does not strictly apply in the context of a general tetrahedrization, the basic ideas (mainly due to the manner in which it is described) are very useful for developing visibility-sorting algorithms for specific applications, so we include a description of the topological sort algorithm here.

Topological Sort Algorithm: The topological sort algorithm as described by Knuth [136] starts with a directed, acyclic graph (DAG). The DAG can be represented with a diagram using nodes and arrows. See Figure 20.65. The nodes represent the elements of the set to be ordered, and an arrow from node x to node y represents the relation of the partial ordering, $x < y$. The algorithm is simple. Any node that has no incoming arrow is removed from the DAG (with all of its attached arrows) and placed in the linear ordering. This process is repeated until the DAG is empty. It is easy to prove (left to the reader) that if the DAG represents a partial ordering, a linear ordering will always be produced by this algorithm.

Max [166] has discussed the application of the ideas of the topological sort algorithm to the problem of producing a visibility sort for a cellular decomposition of a domain. Max defines the order relation in the following way. The DAG contains an arrow for each face common to two cells x and y. The arrow is directed from x to y if the viewpoint is on the same side of the face as x, meaning that y must be processed before x. Max mentions that the topological sorting algorithm will be successful "if every ray through the data volume intersects it in a single sequence of adjacent cells." Of course, if the cell complex contains cycles (see Figure 20.64), then a visibility sort is not possible. Williams [257] discusses similar algorithms applied to a very general cellular decomposition which may contain empty cavities.

We conclude this section with some rather interesting properties about the special case of the Delaunay tetrahedrization of the convex hull of a collection of 3D points. The *power* of a tetrahedra is defined as $D^2 - R^2$, where D is the distance to the viewpoint from the

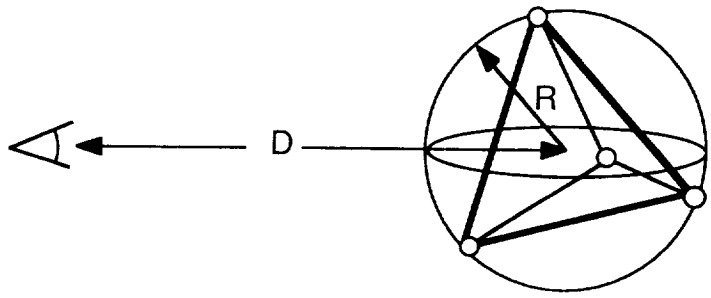


Figure 20.66: Elements of the definition of the power of a tetrahedron.

center of the circumsphere of the tetrahedron and R is the radius of the circumscribing sphere. See Figure 20.66. A visibility sort can be accomplished by a simple sort based upon the power. This property is covered in [69] and used by Max, Hanrahan, and Crawfis [167]. We caution the reader that this approach breaks down in the presence of neutral cases where possibly several tetrahedra have the same power (as in the case of decomposing the cube). One additional interesting observation in this context is that a sort based upon the power of the tetrahedra does not require the neighborhood information that is required for the algorithms using the ideas of topological sorting. Another method which does not use adjacency information is described by Stein, Becker, and Max [240].

20.3.4 Data-Dependent Tetrahedrizations

Lee [148] has investigated the topic of data-dependent tetrahedrizations. This work generalizes from 2D to 3D the ideas and techniques of [67] and [225]. Similar to the algorithms of [225], simulated annealing is used. The initial tetrahedrization is the Delaunay tetrahedrization of the convex hull of the independent data site locations. Local swapping of tetrahedra is performed based upon random values compared to an annealing schedule and a cost function. This "randomness" of the simulated annealing approach allows the algorithm to escape local extrema of the cost function. Local swapping for 2D simply involves the choice of one or the other of the diagonals of a quadrilateral. In 3D the situation is more complex. There are four cases shown in Figure 20.63 which are the same as those used in the 3D version of Lawson's algorithm. In the first case, three triangles are swapped for two. The second case is the reverse of the first—two tetrahedra are replaced by three. The third case is where two triangles are on the boundary of the convex hull and the two tetrahedra can be swapped for two other tetrahedra. In the last case four tetrahedra are swapped for four other tetrahedra.

In Section 20.2.4, we described the cost function used by Dyn, Levin, and Rippa [67]. Analogous to these cost functions for 2D, Lee [148] uses the following criterion for 3D:

Gradient Difference: Let T_1 and T_2 be two tetrahedra with a common triangular face. Let G_1 be the gradient of the linear function which interpolates the data at the four vertices of T_1, and let G_2 be the similar gradient for the linear interpolant of T_2. The gradient difference is defined as $||G_1 - G_2||$.

Method	RMS Error
Delaunay	.007475
Difference in Gradient	.005445
Jump in Normal Derivative	.004361

Figure 20.67: Errors for the piecewise linear interpolant using different tetrahedrization.

Jump in Normal Direction Derivatives: Let $L_1(x, y, z) = a_1x+b_1y+c_1z+d_1$ be the linear function which interpolates to the data at the four vertices of T_1 and let $L_2(x, y, z) = a_2x + b_2y + c_2z + d_2$ be the similar function for T_2. Let $N = (n_x, n_y, n_z)$ be the normal (normalized) of the common triangular face of T_1 and T_2. $D_1 = a_1n_x + b_1n_y + c_1n_z$ is the directional derivative of L_1 in the direction of N. $D_2 = a_2n_x + b_2n_y + c_2n_z$ is the analogous value for T_2. The jump in normal direction criterion is $|D_1 - D_2| = |(a_1 - a_2)n_x + (b_1 - b_2)n_y + (c_1 - c_2)n_z)|$.

Some example results reported by Lee [148] are repeated here in Figure 20.67. This example involves a test function, $F(x, y, z) = (\tanh(9y-9x-9z)+1)/9$, which provides the dependent data. The piecewise linear interpolant over the tetrahedrization is compared to the test function. The RMS errors are based upon evaluations of the functions and this approximation over a $20 \times 20 \times 20$ Cartesian grid. The dependent data site locations are taken to be 1000 random points in the unit cube.

In Figure 20.68 are some graphs which can be considered as 3D analogs of the graphs shown in Figure 20.31 of Section 20.2.4. Similar to the 2D case, the data-dependent tetrahedrization involves some badly shaped tetrahedra. This is the cost of having an optimal (or nearly optimal) piecewise linear approximation.

20.3.5 Affine Invariant Tetrahedrizations

In this section we extend the results of Section 20.2.5 on affine invariant triangulations to that of affine invariant tetrahedrizations. Prior to discussing the characterization and computation of this type of tetrahedrization, we make some comments about the need for such a tetrahedrization over and above those reasons for the 2D case. It appears that as the dimension of the independent data increases, our need to be concerned about lack of affine invariance also increases.

One source of 3D independent data is the case of time-varying 2D data. In some cases the data measurement locations might stay fixed over time and in some cases they may vary over time. Let us say, for example, that we have a vector field which is known (by means of a numerical simulation) at the locations of a 2D curvilinear grid $(x_{ij}, y_{ij}), i = 1, \ldots, N_x;\ j = 1, \ldots, N_y$. As time proceeds, the vector field varies, but the dependent data site locations stay fixed. So in this case, we have data which can be represented as $(V_{ijk}; x_{ij}, y_{ij}, t_k),\ i = 1, \ldots, N_x,\ j = 1, \ldots, N_y,\ k = 1, \ldots, N_t$. If the definition of a modeling function $F(x, y, t)$, designed to interpolate the data, $F(x_{ij}, y_{ij}, t_k) = F_{ijk}$,

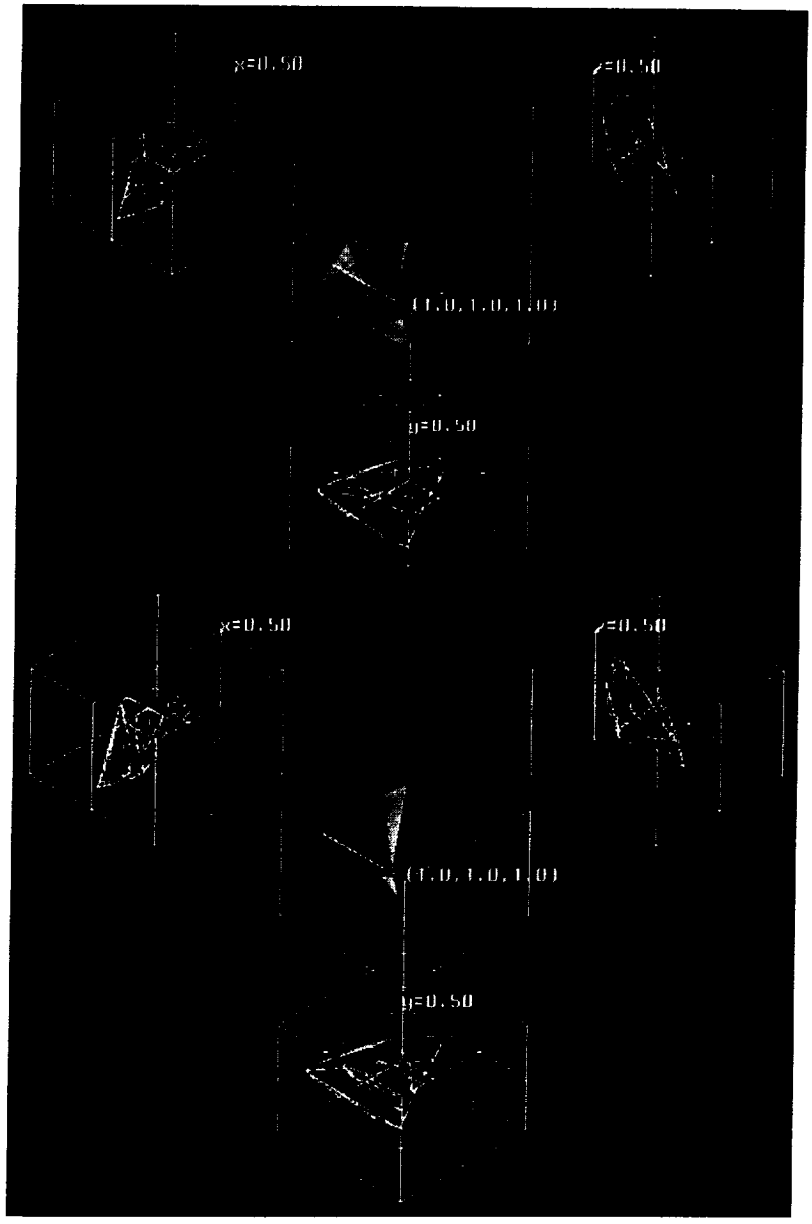


Figure 20.68: Data-dependent tetrahedrization compared to the Delaunay tetrahedrization.

is based upon a tetrahedrization of the 3D independent data (x_{ij}, y_{ij}, t_k), then this model will not necessarily be affine invariant, and the units used to measure and represent the physical coordinates and time could have an effect on the modeling function $F(x, y, t)$ and subsequently an effect on the visualization and analysis. The same problem could also occur for a time-varying vector field over a curvilinear grid which also varies over time—that is, data of the type $(F_{ijk}; x_{ijk}, y_{ijk}, t_k)$, $i = 1, \ldots, N_x$, $j = 1, \ldots, N_y$, $k = 1, \ldots, N_t$. In general, any tetrahedrization of the independent data of (F_{ijk}, x_i, y_j, z_k), where the choice of the units of measurement used for the independent data could lead to a nonuniform scaling, could have the problem of being dependent on the choice of the units used. If each of the variables use the same units there will be no problems of this type, because a scale transformation of the form $x \leftarrow ax$, $y \leftarrow ay$, $z \leftarrow az$, where the scale change is uniform for each variable, will not affect the tetrahedrization. It is only the nonuniform scaling $x \leftarrow ax$, $y \leftarrow by$, $z \leftarrow cz$ which creates the problem. An example of a scale change affecting the tetrahedrization is shown in Figure 20.69. Here there are 10 data points. In the right image, the data has been scaled in the y variable by a factor of 2. Not only does the tetrahedrization change, but even the number of tetrahedra changes. The Delaunay tetrahedrization of the original 10 data points has 18 tetrahedra and the scaled data has 13 tetrahedra.

We now describe the 3D version of the affine invariant norm, which leads (by way of the Dirichlet tessellation) to an affine invariant tetrahedrization. Actually, we can define it so that it is clear what the generalization is for any dimension. Let

$$\|(x, y, z)\|_V^2 = (x, y, z)(VV^*)^{-1} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

where V is the $3 \times N$ matrix of translated data values

$$V = \begin{pmatrix} x_1 - \mu_x & x_2 - \mu_x & \cdots & x_N - \mu_x \\ y_1 - \mu_y & y_2 - \mu_y & \cdots & y_N - \mu_y \\ z_1 - \mu_z & z_2 - \mu_z & \cdots & z_N - \mu_z \end{pmatrix}.$$

As with the 2D case, there are some different approaches to modifying an existing tetrahedrization procedure. Probably the simplest is to preprocess the data with the transformation given by the lower triangular matrix, $L(V)$ which results from the Cholesky decomposition of $(VV^*)^{-1}$

$$L(V)L(V)^* = (VV^*)^{-1}.$$

Explicitly in the 3D case, we use the transformed data

$$\begin{aligned} X_i &= l_{11}x_i + l_{21}y_i + l_{31}z_i \\ Y_i &= l_{22}y_i + l_{32}z_i \\ Z_i &= l_{33}z_i \end{aligned}$$

where

$$l_{11} = \sqrt{a_{11}}, \quad l_{21} = \frac{a_{11}}{l_{11}}, \quad l_{31} = \frac{a_{13}}{l_{11}},$$

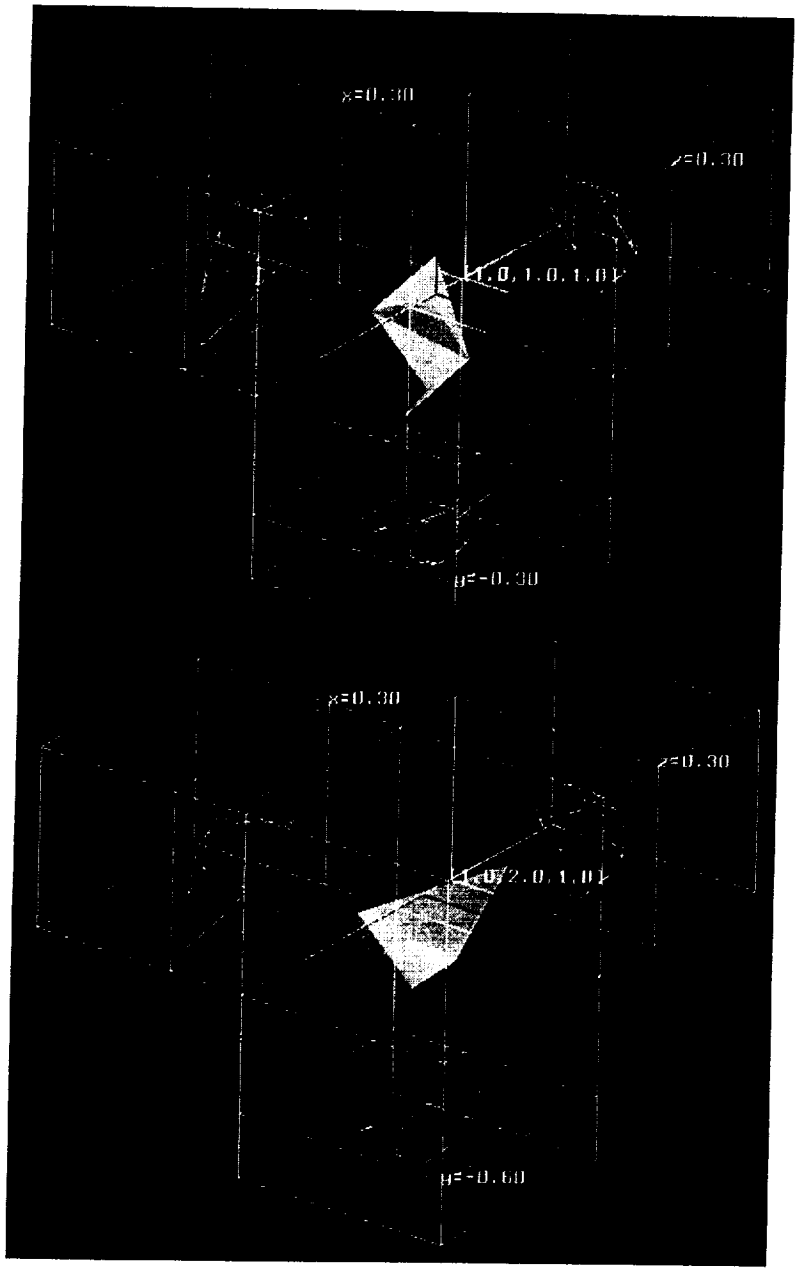


Figure 20.69: Delaunay tetrahedrization of 10 data points and a scaled version of the same data points.

$$l_{22} = \sqrt{a_{22} - (l_{21})^2}, \quad l_{32} = \frac{a_{33} - l_{21}l_{31}}{l_{22}},$$

$$l_{33} = \sqrt{a_{33} - (l_{31})^2 - (l_{32})^2}$$

$$\begin{aligned} A &= (a_{ij}) = (VV^*)^{-1} = L(V)L(V)^* \\ &= \frac{1}{\det}\begin{pmatrix} \Sigma_y^2\Sigma_z^2 - \Sigma_{yz}^2 & -(\Sigma_{xy}\Sigma_z^2 - \Sigma_{xz}\Sigma_{yz}) & \Sigma_{xy}\Sigma_{yz} - \Sigma_{xz}\Sigma_y^2 \\ -(\Sigma_{xy}\Sigma_z^2 - \Sigma_{xz}\Sigma_{yz}) & \Sigma_x^2\Sigma_z^2 - \Sigma_{xz}^2 & -(\Sigma_{yz}\Sigma_x^2 - \Sigma_{xz}\Sigma_{xy}) \\ \Sigma_{xy}\Sigma_{yz} - \Sigma_{xz}\Sigma_y^2 & -(\Sigma_{yz}\Sigma_x^2 - \Sigma_{xz}\Sigma_{xy}) & \Sigma_x^2\Sigma_y^2 - \Sigma_{xy}^2 \end{pmatrix} \end{aligned}$$

where

$$\begin{aligned} \Sigma_x^2 &= \frac{\sum_{i=1}^N (x_i - \mu_x)^2}{N}, \qquad \mu_x = \frac{\sum_{i=1}^N x_i}{N} \\ \Sigma_y^2 &= \frac{\sum_{i=1}^N (y_i - \mu_y)^2}{N}, \qquad \mu_y = \frac{\sum_{i=1}^N y_i}{N} \\ \Sigma_z^2 &= \frac{\sum_{i=1}^N (z_i - \mu_z)^2}{N}, \qquad \mu_z = \frac{\sum_{i=1}^N z_i}{N} \\ \Sigma_{xy} &= \frac{\sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)}{N}, \\ \Sigma_{yz} &= \frac{\sum_{i=1}^N (y_i - \mu_y)(z_i - \mu_z)}{N}, \\ \Sigma_{xz} &= \frac{\sum_{i=1}^N (x_i - \mu_x)(z_i - \mu_z)}{N}, \end{aligned}$$

and

$$\det = \Sigma_x^2\Sigma_y^2\Sigma_z^2 + 2\Sigma_{xy}\Sigma_{yz}\Sigma_{xz} - \Sigma_x^2(\Sigma_{yz})^2 - \Sigma_y^2(\Sigma_{xz})^2\Sigma_z^2(\Sigma_{xy})^2.$$

We conclude this section with some examples illustrating this affine invariant norm and its use in characterizing affine invariant tetrahedrizations. In Figure 20.70 there are four graphs of 13 data points. The transparent ellipsoids represent all the points that are 0.50 and 1.0 units from the center point using the affine invariant norm. The different graphs show the data after it has undergone an affine transformation. The original data is displayed in the upper left. The upper right shows the data after it has been rotated by 44 degrees about the z axis. The lower right is after it has subsequently been scaled in the x variable by a factor of 1.5. The lower left is after it has been scaled in y by a factor of 0.6. A close examination of these graphs will show that the relative distances (as measured by the affine invariant norm) between points is unchanged by these transformations. Figure 20.71 shows an affine invariant tetrahedrization. In comparison, the conventional Delaunay tetrahedrization is shown in Figure 20.72.

20.3.6 Interpolation in Tetrahedra

As with the bivariate case covered in Section 20.2.6, there are two concepts of interest for interpolation in tetrahedra. The first is concerned with the amount of boundary data that

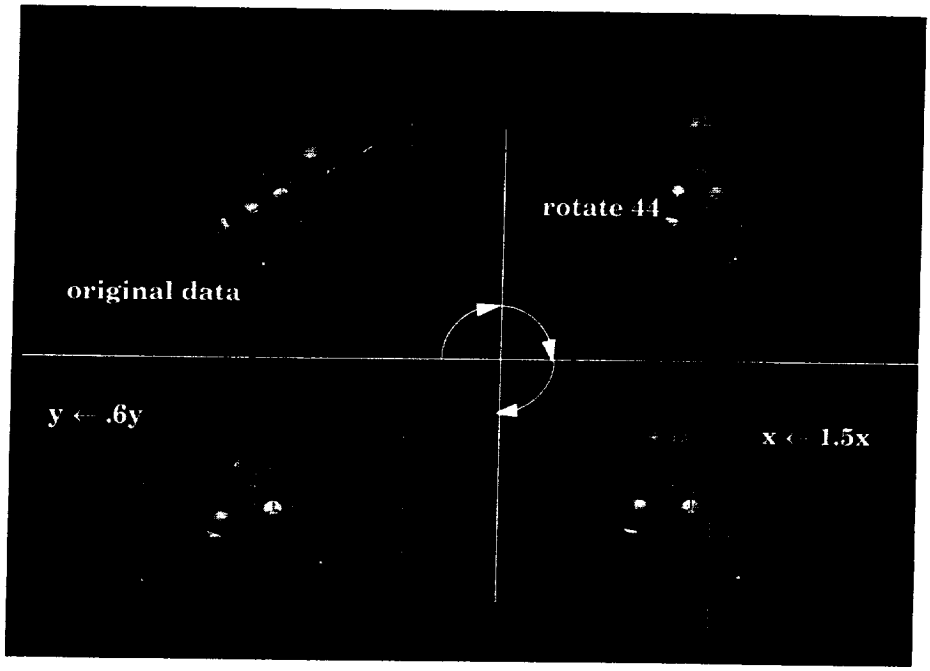


Figure 20.70: Examples illustrating the affine invariant norm. The ellipsoids are 0.50 and 1.0 units from the center point.

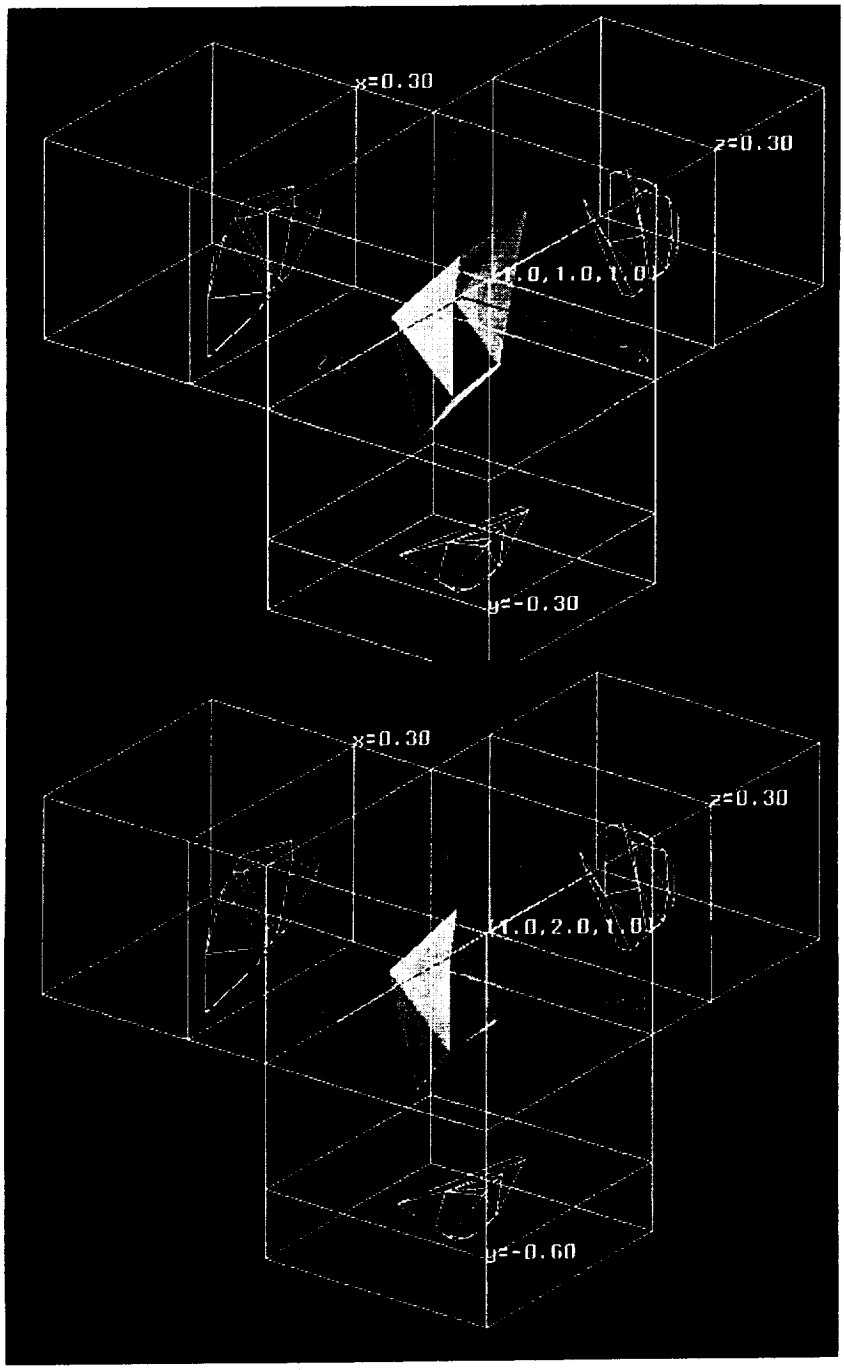


Figure 20.71: Examples of affine invariant tetrahedrization.

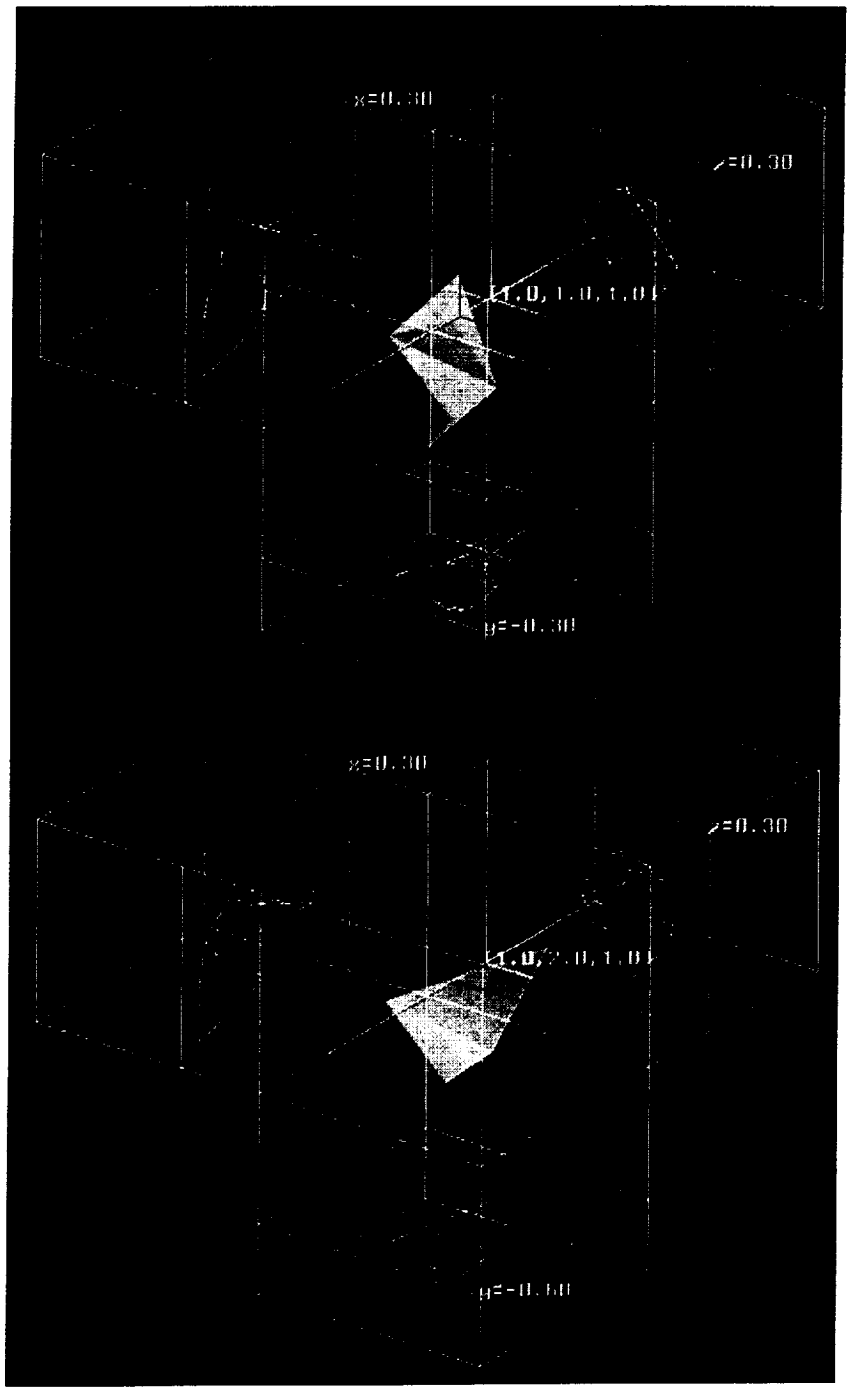


Figure 20.72: Delaunay tetrahedrization of the same data as in Figure 20.71.

is provided or assumed to be available. This can be discrete data provided at a finite number of locations (usually the vertices or midpoints) or transfinite data where boundary data values are assumed to be available at all locations on the boundary. The second concept relates to the degree of continuity of a piecewise defined interpolant using the local interpolants described here. C^0 interpolants use only boundary position data and lead to overall interpolants which are continuous. C^1 interpolants utilize first-order derivative information and lead to global interpolants which have all first-order derivative continuous. These two concepts lead to four possibilities which we discuss below.

C^0, Discrete Interpolation in Tetrahedra

Analogous to the bivariate linear interpolant which will match predescribed values at the three vertices of a triangle, there is a unique trivariate linear interpolant which will match data at the four vertices of a tetrahedra, T_{ijkl}. Given $F(V_i), F(V_j), F(V_k)$ and $F(V_l)$, the coefficients of this linear function which interpolates this data

$$F(x, y, z) = a + bx + cy + dz$$

can be found by solving the linear system of equations

$$\begin{aligned} a + bx_i + cy_i + dz_i &= F(V_i) \\ a + bx_j + cy_j + dz_j &= F(V_j) \\ a + bx_k + cy_k + dz_k &= F(V_k) \\ a + bx_l + cy_l + dz_l &= F(V_l) \end{aligned}$$

As before, it is also possible to use barycentric coordinates. The barycentric coordinates of a point $V = (x, y, z)$ are defined by the relationships

$$\begin{aligned} V &= b_i V_i + b_j V_j + b_k V_k + b_l V_l \\ 1 &= b_i + b_j + b_k + b_l \end{aligned}$$

and the linear interpolant has the form

$$F(x, y, z) = F(V) = b_i F(V_i) + b_j F(V_j) + b_k F(V_k) + b_l F(V_l). \tag{20.6}$$

As before, there are several ways of defining or computing barycentric coordinates. The analog of the ratios of areas we saw before is the ratio of volumes of subtetrahedra,

$$b_i = \frac{Vol(T_{Vjkl})}{Vol(T_{ijkl})}, \quad b_j = \frac{Vol(T_{iVkl})}{Vol(T_{ijkl})}, \quad b_k = \frac{Vol(T_{ijVl})}{Vol(T_{ijkl})}, \quad b_j = \frac{Vol(T_{ijkV})}{Vol(T_{ijkl})}$$

where T_{Vjkl} is the tetrahedron with vertices V, V_j, V_k, and V_l and similar definitions for the other subtetrahedra. The volume of a tetrahedron, T_{abcd}, with vertices a, b, c, and d is

$$Vol(T_{abcd}) = \frac{1}{6}[(d - a) \cdot ((b - a) \times (c - a))]$$

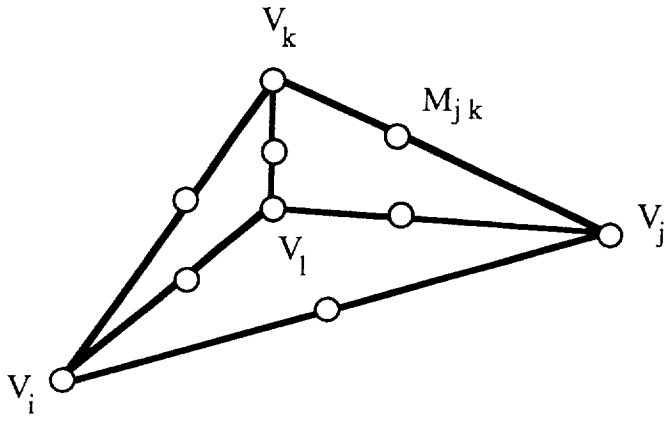


Figure 20.73: Data site locations for trivariate quadratic interpolation.

Also determinants can be used,

$$b_i = \frac{\begin{vmatrix} x-x_j & x-x_k & x-x_l \\ y-y_j & y-y_k & y-y_l \\ z-z_j & z-z_k & z-z_l \end{vmatrix}}{\begin{vmatrix} x_i-x_j & x_i-x_k & x_i-x_l \\ y_i-y_j & y_i-y_k & y_i-y_l \\ z_i-z_j & z_i-z_k & z_i-z_l \end{vmatrix}}, \quad b_j = \frac{\begin{vmatrix} x-x_i & x-x_k & x-x_l \\ y-y_i & y-y_k & y-y_l \\ z-z_i & z-z_k & z-z_l \end{vmatrix}}{\begin{vmatrix} x_j-x_i & x_j-x_k & x_j-x_l \\ y_j-y_i & y_j-y_k & y_j-y_l \\ z_j-z_i & z_j-z_k & z_j-z_l \end{vmatrix}},$$

$$b_k = \frac{\begin{vmatrix} x-x_i & x-x_j & x-x_l \\ y-y_i & y-y_j & y-y_l \\ z-z_i & z-z_j & z-z_l \end{vmatrix}}{\begin{vmatrix} x_k-x_i & x_k-x_j & x_k-x_l \\ y_k-y_i & y_k-y_j & y_k-y_l \\ z_k-z_i & z_k-z_j & z_k-z_l \end{vmatrix}}, \quad b_l = \frac{\begin{vmatrix} x-x_i & x-x_j & x-x_k \\ y-y_i & y-y_j & y-y_k \\ z-z_i & z-z_j & z-z_k \end{vmatrix}}{\begin{vmatrix} x_l-x_i & x_l-x_j & x_l-x_k \\ y_l-y_i & y_l-y_j & y_l-y_k \\ z_l-z_i & z_l-z_j & z_l-z_k \end{vmatrix}}.$$

Given the values at the four vertices and the six midpoints of a tetrahedron, there is a unique trivariate quadratic which interpolates this data,

$$\begin{aligned} Q(x,y,z) \;=\; & F(V_i)b_i(b_i-b_j-b_k-b_l) + F(V_j)b_j(b_j-b_i-b_k-b_l) \\ & +F(V_k)b_k(b_k-b_i-b_j-b_l) + F(V_l)b_l(b_l-b_i-b_j-b_k) \\ & +F(M_{ik})4b_ib_k + F(M_{jl})4b_jb_l + F(M_{ij})4b_ib_j \\ & +F(M_{jk})4b_jb_k + F(M_{il})4b_ib_l + F(M_{kl})4b_kb_l \end{aligned} \tag{20.7}$$

where $M_{ij} = (V_i + V_j)/2$ and the other midpoints are defined similarly. See Figure 20.73.

C^0, Transfinite Interpolation in Tetrahedra

As before in Section 20.2.6, we give a sampling of interpolants. One is a generalization of the side-vertex interpolant and the other is a generalization of the C^* interpolant. Both of

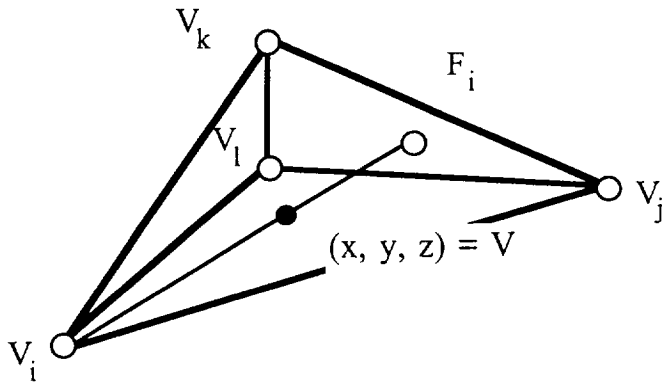


Figure 20.74: Notation for the face-vertex interpolant.

these bivariate interpolants were discussed previously in Section 20.2.6.

The C^0, Face-Vertex Interpolant: Analogous to the basic interpolants used to construct the side-vertex interpolant, we have the interpolants which consist of linear interpolation along edges joining a vertex and the opposing face

$$\begin{aligned} A_i[F] &= b_i F(V_i) + (1-b_i)F(F_i) \\ A_j[F] &= b_j F(V_j) + (1-b_j)F(F_j) \\ A_k[F] &= b_k F(V_k) + (1-b_k)F(F_k) \\ A_l[F] &= b_l F(V_l) + (1-b_l)F(F_l) \end{aligned} \tag{20.8}$$

where $F_i = \dfrac{b_j V_j + b_k V_k + b_l V_l}{b_j + b_k + b_l}$, $F_j = \dfrac{b_i V_i + b_k V_k + b_l V_l}{b_i + b_k + b_l}$, $F_k = \dfrac{b_i V_i + b_j V_j + b_l V_l}{b_i + b_j + b_l}$ and $F_l = \dfrac{b_i V_i + b_j V_j + b_k V_k}{b_i + b_j + b_k}$. See Figure 20.74. Computing the Boolean sum of these four interpolants leads to

$$\begin{aligned} A[F] = \quad & (1-b_i)F(F_i) + (1-b_j)F(F_j) + (1-b_k)F(F_k) + (1-b_l)F(F_l) \\ & -(b_k+b_l)F(S_{kl}) - (b_i+b_l)F(S_{il}) - (b_j+b_l)F(S_{jl}) \\ & -(b_j+b_k)F(S_{jk}) - (b_i+b_k)F(S_{ik}) - (b_i+b_j)F(S_{ij}) \\ & +b_i F(V_i) + b_j F(V_j) + b_k F(V_k) + b_l F(V_l) \end{aligned} \tag{20.9}$$

where $S_{mn} = \dfrac{b_m V_m + b_n V_n}{b_m + b_n}, mn = kl, il, jl, jk, ik, ij.$

The C^* Interpolant (for a tetrahedron): The analog of the bivariate C^* interpolant described in Section 20.2.6 is

$$\begin{aligned} C^*[F] \;=\; & b_i F(V_i) + b_j F(V_j) + b_k F(V_k) + b_l F(V_l) \\ & + W_l \left\{ F(Q_l) - \left(b_i + \frac{b_l}{3}\right) F(V_i) - \left(b_j + \frac{b_l}{3}\right) F(V_j) - \left(b_k + \frac{b_l}{3}\right) F(V_k) \right\} \\ & + W_k \left\{ F(Q_k) - \left(b_i + \frac{b_k}{3}\right) F(V_i) - \left(b_j + \frac{b_k}{3}\right) F(V_j) - \left(b_l + \frac{b_k}{3}\right) F(V_l) \right\} \\ & + W_j \left\{ F(Q_j) - \left(b_i + \frac{b_j}{3}\right) F(V_i) - \left(b_k + \frac{b_j}{3}\right) F(V_k) - \left(b_l + \frac{b_j}{3}\right) F(V_l) \right\} \\ & + W_i \left\{ F(Q_i) - \left(b_j + \frac{b_i}{3}\right) F(V_j) - \left(b_k + \frac{b_i}{3}\right) F(V_k) - \left(b_l + \frac{b_i}{3}\right) F(V_l) \right\} \end{aligned} \tag{20.10}$$

where $Q_l = \left(b_i + \frac{b_l}{3}\right) V_i + \left(b_j + \frac{b_l}{3}\right) V_j + \left(b_k + \frac{b_l}{3}\right) V_k$,

$W_l = \dfrac{27 b_i b_j b_k}{(3b_i + b_l)(3b_j + b_l)(3b_k + b_l)}$ and the other Q's and W's are defined in a similar manner.

C^1, Transfinite Interpolation in Tetrahedra

The C^1, Face-Vertex Interpolant: It is a straightforward process to extend the C^1, transfinite side-vertex interpolant to a tetrahedral domain, T_{ijkl}. It is called the C^1, face-vertex interpolant and we assume that position and derivative information is available at all locations on the four faces which make up the boundary of the tetrahedron T_{ijkl}. The basic face-vertex operator is defined as

$$\begin{aligned} S_i[F](p) \;=\; & b_i^2(3 - 2b_i)F(V_i) + b_i^2(b_i - 1)F'(V_i) \\ & + (1 - b_i)^2(2b_i + 1)F(F_i) + b_i(1 - b_i)^2 F'(F_i) \end{aligned} \tag{20.11}$$

where $F'(V_i) = \dfrac{(x - x_i)F_x(V_i) + (y - y_i)F_y(V_i) + (z - z_i)F_z(V_i)}{1 - b_i}$ and

$F'(F_i) = \dfrac{(x - x_i)F_x(S_i) + (y - y_i)F_y(S_i) + (z - z_i)F_z(S_i)}{1 - b_i}$. The point F_i is the intersection point of the ray from V_i through V and the face opposite V_i, and the derivatives are taken in the direction of this same ray. See Figure 20.75. If we form the convex combination

$$S[F] = \frac{b_j^2 b_k^2 b_l^2 S_i[F] + b_i^2 b_k^2 b_l^2 S_j[F] + b_j^2 b_l^2 b_i^2 S_k[F] + b_j^2 b_k^2 b_i^2 S_l[F]}{b_j^2 b_k^2 b_l^2 + b_i^2 b_k^2 b_l^2 + b_j^2 b_l^2 b_i^2 + b_j^2 b_k^2 b_i^2}$$

then $S[F]$ will match position and derivative values on the entire boundary of T_{ijkl}.

C^1, Discrete Interpolation in Tetrahedra

For a C^1, discrete interpolant, we assume that position and first-order derivative information is given at all four vertices of the tetrahedron T_{ijkl}. Since there are three (linearly independent) directional derivatives at each vertex, this amounts to a total of 16 data values. The

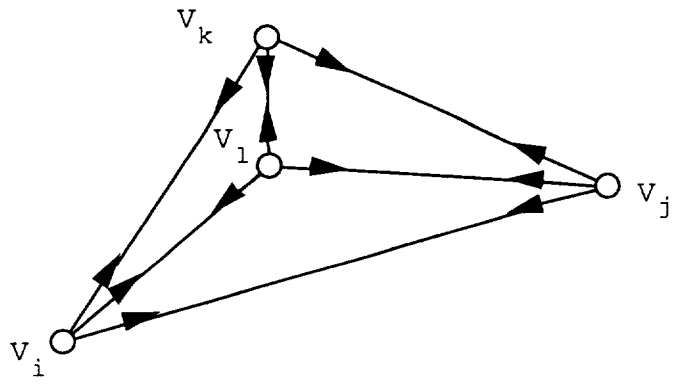


Figure 20.75: The data for a 16 parameter, C1 interpolant over a tetrahedron.

method for describing an interpolant that will match these 16 pieces of data—and which also has the property that all first-order derivatives across a face with common data will be continuous—is somewhat different from the previous interpolants we have described so far. Our description (and subsequent implementation) is based upon a two-step procedural discretization process. We use the transfinite interpolant of the previous section. In order to apply this transfinite interpolant, we need to define position and derivative values on the entire boundary of T_{ijkl}. First we assume that information is known on all the edges of the tetrahedra, and we describe how to extend it to the entire boundary. Second, we describe how to provide this transfinite edge data from only the discrete data at the vertices. If we know both position and derivative information on the edges, then we can use any C^1 transfinite planar triangular interpolant to define position values on the interior points of the face triangles. For example, the side-vertex method itself could be used. Specifying position information on a face also implies some information about the derivatives on the interior of a triangle. Namely, all directional derivatives in a direction parallel to the face triangle are determined; so, in order to completely specify all derivatives, we need only provide a definition for the derivative perpendicular to the face. For this we use the C^0 version of the side-vertex interpolant which interpolates position data only and not derivatives, but we apply it to the edge data consisting of derivatives normal to a face. We now describe the second step of the discretization, which is how to compute edge information when only the point and derivative values are known at the four vertices. For position only on an edge, we simply use univariate cubic Hermite interpolation. This will also specify one directional derivative on the edge—namely $\frac{\partial F}{\partial e_{ij}}$, which will vary as a quadratic polynomial. In order to get a C^1 join from one tetrahedron to the next, the other two directional derivatives must vary linearly along this edge. This is accomplished by specifying the gradient, ∇F, by the relationship

$$\begin{aligned} \nabla F_{ij}(p) &= (1-t)\nabla F_i + t\nabla F_j \\ &\quad + \left[\frac{\partial F}{\partial e_{ij}}(p)((1-t)\nabla F_i + t\nabla F_i, e_{ij})\right] e_{ij} \end{aligned} \tag{20.12}$$

where $\nabla F_i = (F_x(p_i), F_y(p_i), F_z(p_i))$ and $t = \frac{\|p-p_i\|}{\|p_j-p_i\|}$. This interpolation of the gradient is consistent with the value $\frac{\partial F}{\partial e_{ij}}$ already specified because $(\nabla F_{ij}(p), e_{ij}) = \frac{\partial F}{\partial e_{ij}}$ and it also has the property that for $(n, e_{ij}) = 0$,

$$(\nabla F_{ij}(p), n) = (1-t)(\nabla F_i, n) + t(\nabla F_j, n),$$

and so we have linear interpolation for any derivative in a direction perpendicular to e_{ij}. This completes the definition of the 16-parameter, C^1, tetrahedral interpolant which is based upon the face-vertex interpolant. Examples and more discussion on this interpolant can be found in [187]. The Clough-Tocher interpolant has been generalized to n-dimensional by Worsey and Farin [261]. Other C^1, discrete interpolants for a tetrahedral domain are discussed in [2], [3], and [260], but each have some problem or drawback. The method of [2] is based upon the side-side, transfinite method of interpolation and apparently it has a problem with the linear independence of the discretized data. The method of [3] requires C^2 data for a C^1 interpolant and the method of [260] has a problem similar to its bivariate precursor [199] and [198]. This problem lies in the constraint that the center of the circumcircle of each triangle must lie interior to the triangular domain.

Acknowledgments

We wish to acknowledge the support of the National Aeronautical and Space Administration under NASA-Ames Grant, NAG 2-990. Also, partial support was provided by the North Atlantic Treaty Organization under grant RG 0097/88. We wish to thank Herbert Edelsbrunner for the idea of the dual graphs of Section 20.3.1 and other insightful discussions about tetrahedrizations. We wish to thank Kun Lee for his help in generating the images of Sections 20.3.4 and 20.3.5.

Bibliography

[1] A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor, "A Linear Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon," *Disc. and Comp. Geometry 4*, 1989, pp. 591–604.

[2] P. Alfeld, "A Discrete C^1 Interpolant for Tetrahedral Data," *The Rocky Mountain Journal of Mathematics*, Vol. 14, No. 1, Winter 1984, pp. 5–16.

[3] P. Alfeld, "A Trivariate Clough-Tocher Scheme for Tetrahedral Data," *Computer Aided Geometric Design*, Vol. 1, No. 2, 1984, pp. 169–181.

[4] T. Asano and R. Pinter, "Polygon Triangulation: Efficiency and Minimality," *J. Algorithms*, Vol. 7, 1986, pp. 221–231.

[5] D. Avis, and B.K. Bhattacharya, "Algorithms for Computing d-Dimensional Voronoi Diagrams and their Duals," *Advance in Computing Research*, Vol. 1, pp. 159–180.

[6] D. Avis and K. Fukuda, "A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra," *Proceedings 7th Annual ACM Symposium Computational Geometry*, 1991, pp. 98–104.

[7] D. Avis and G.T. Toussaint, "An Efficient Algorithm for Decomposing a Polygon into Star-Shaped Polygons," *Pattern Recogn.*, Vol. 13, No. 6, 1981, pp. 395–398.

[8] F. Aurenhammer, "Voronoi Diagrams—A Survey of a Fundamental Geometric Data Structure," *ACM Computing Surveys*, Vol. 23, 1991, pp. 345–405.

[9] I. Babuska and A. Aziz, "On the Angle Condition in the Finite Element Method," *SIAM J. Numer. Analysis*, Vol. 13, 1976, pp. 214–227.

[10] P.I. Baegmann, M.S. Shepard, and J.E. Flaherty, "A Posterori Error Estimation for Triangular and Tetrahedral Quadratic Elements Using Interior Residuals," *Internat. J. Numer. Meth. Eng.*, Vol. 34, 1992, pp. 979–996.

[11] F. Bagemihl, "On Indecomposable Polyhedra," *American Mathematical Monthly*, Sep. 1948, pp. 411–413.

[12] T.J. Baker, "Automatic Mesh Generation for Complex Three-Dimensional Regions Using a Constrained Delaunay Triangulation," *Eng. with Computers*, Vol. 5, 1989, pp. 161–175.

[13] B.S. Baker, E. Grosse, and C.S. Rafferty, "Nonobtuse Triangulation of Polygons," *Disc. and Comp. Geom.*, Vol. 3, 1988, pp. 147–168.

[14] G. Baszenski and L.L. Schumaker, "Use of Simulated Annealing to Construct Triangular Facet Surfaces," *Curves and Surfaces*, P.-J. Laurent, A. Le Mehaute and L.L. Schumaker, editors, Academic Press, Boston, Mass., 1991, pp. 27–32.

[15] M. Bern and D. Eppstein, "Polynomial-Size Nonobtuse Triangulation of Polygons," *Proc. 7th ACM Symp. Comp. Geometry*, 1991, pp. 342–350.

[16] M. Bern, and D. Eppstein, "Mesh Generation and Optimal Triangulation," *Computing in Euclidean Geometry*, F. K. Hwang and D.-Z. Du, editors, World Scientific, Singapore, 1992, pp. 23–90.

[17] M. Bern, D. Dobkin, and D. Eppstein, "Triangulating Polygons with Large Angles," *Proc. 8th ACM Sym. Comp. Geometry*, 1992.

[18] M. Bern, D. Eppstein, and F. Yao, "The Expected Extremes in a Delaunay Triangulation," *Int. J. Comp. Geometry and Applications*, Vol. 1, 1991, pp. 79–92.

[19] J. Bloomenthal, "Polygonization of Implicit Surfaces," *CAGD*, Vol. 5, 1988, pp. 341–355.

[20] C. Borgers, "Generalized Delaunay Triangulations of Nonconvex Domains," *Computers & Mathematics with Applications*, Vol. 20, No. 7, 1990, pp. 45–49.

[21] A. Bowyer, "Computing Dirichlet Tessellations," *Computer J.*, Vol. 24, 1981, pp. 162–166.

[22] C. Bradford, D. Barber, D.P. Dobkin, and H. Huhdanpaa, *The Quickhull Algorithm for Convex Hull*, Technical Report GCG53-93, Geometry Center, University of Minnesota, July 1993.

[23] J. Bramble and M. Zlamal, "Triangular Elements in the Finite Element Method," *Math. Comp.*, Vol. 24, 1970, pp. 809–820.

[24] K.E. Brassel and D. Reif, "A Procedure to Generate Thiessen Polygons," *Geograph. Anal.*, Vol. 11, 1979, pp. 289–303.

[25] W. Brostow, J.P. Dussault, and B.L. Fox, "Construction of Voronoi Polyhedra," *J. Comp. Physics*, Vol. 29, 1978, pp. 81–92.

[26] J.L. Brown, "Vertex Based Data Dependent Triangulations," *Computer Aided Geometric Design*, Vol. 8, 1991, pp. 239–251.

[27] K.Q. Brown, "Voronoi Diagrams from Convex Hulls," *Inform. Process. Lett.*, Vol. 9, 1979, pp. 223–228.

[28] J.C. Cavendish, "Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method," *Int. J. for Numer. Methods in Engr.*, Vol. 8, 1974, pp. 679–696.

[29] J.C. Cavendish, D.A. Field, and W.H. Frey, "An Approach to Automatic Three-Dimensional Finite Element Mesh Generation," *Int. J. Numer. Meth. Eng.*, Vol. 21, 1985, pp. 329–347.

[30] M.S. Chang, N.-F. Huang, and C.Y. Tang, "Optimal Algorithm for Constructing Oriented Voronoi Diagrams and Geographic Neighborhood Graphs," *Information Processing Letters*, Vol. 35, No. 5, Aug. 1990, pp. 255–260.

[31] R.C. Chang and R.C.T. Lee, "On the Average Length of Delaunay Triangulations," *BIT*, Vol. 24, 1984, pp. 269–273.

[32] S. Chattopodhyay and P.P. Das, "Counting Thin and Bushy Triangulations," *Pattern Recognition Letters*, Vol. 12, No. 3, 1991, pp. 139–144.

[33] B. Chazelle, "Convex Partitions of Polyhedra: A Lower Bound and Worst-Case Optimal Algorithm," *SIAM J. Comput.*, Vol. 13, 1984, pp. 488–507.

[34] B. Chazelle, "Triangulating a Simple Polygon in Linear Time," *Disc. and Comp. Geometry*, Vol. 6, 1991, pp. 485–524.

[35] B. Chazelle and D. Dobkin, "Decomposing a Polygon into its Convex Parts," *ACM Proceedings of the 11th Symposium on Theory of Computing*, 1979, pp. 38–48.

[36] B. Chazelle, H. Edelsbrunner, L.J. Guibas, J.E. Hershberger, R. Reidel, and M. Sharir, "Selecting Multiply Covered Points and Reducing the Size of Delaunay Triangulations," *Proc. 6th ACM Symp. Comp. Geometry*, 1990, pp. 116–127.

[37] B. Chazell and J. Incerpi, "Triangulating a Polygon by Divide and Conquer," *Proceedings of the 21st Allerton Conference on Communications, Control and Computing*, 1983, pp. 447–456.

[38] B. Chazelle and J. Incerpi, "Triangulation and Shape Complexity," *ACM Trans on Graphics*, Vol. 3, 1984, pp. 135–152.

[39] B. Chazelle and L. Palios, "Triangulating a Nonconvex Polytope," *Disc. and Comp. Geometry*, Vol. 5, 1990, pp. 505–526.

[40] L.P. Chew, "Constrained Delaunay Triangulations," *Algorithmica*, Vol. 4, 1989, pp. 97–108.

[41] B.K. Choi, H.Y. Shin, Y.I. Yoon, and J.W. Lee, "Triangulation of Scattered Data in 3D Space," *CAD*, Vol. 20, 1988, pp. 239–248.

[42] K.L. Clarkson, R.E. Tarjan, and C.J. Van Wyk, "A Fast Las Vegas Algorithm for Triangulating a Simple Polygon," *Discrete and Computational Geometry*, Vol. 4, 1989, pp. 423–432.

[43] A.K. Cline and R.J. Renka, "A Constrained Two-Dimensional Triangulation and the Solution of Closest Node Problems in the Presence of Barriers," *SIAM Journal on Numerical Analysis*, Vol. 27, No. 5, 1990, pp. 1305–1321.

[44] A.K. Cline and R.L. Renka, "A Storage-Efficient Method for Construction of a Thiessen Triangulation," *Rocky Mountain Journal of Mathematics*, Vol. 14, No. 1, Winter 1984, pp. 119–140.

[45] H.E. Cline, W.E. Lorensen, S. Ludke, C.R. Crawford, and B.C. Teeter, "Two Algorithms for the Reconstruction of Surfaces from Tomograms," *Medical Physics*, June 1988.

[46] Y. Correc and E. Chapuis, "Fast Computation of Delaunay Triangulations," *Advances in Engineering Software*, Vol. 9, No. 2, 1987, pp. 77–83.

[47] H.S.M. Coxeter, "Discrete Groups Generated by Reflections," *Ann. Math.*, Vol. 35, 1934, pp. 588–621.

[48] J.R. Davy and P.M. Dew, "A Note on Improving the Performance of Delaunay Triangulation," *New Advances in Computer Graphics: Proceedings of Computer Graphics International 89*, R.A. Earnshaw and B. Wyvill, editors, Springer, Tokyo, 1989, pp. 209–226.

[49] A.M. Day, "The Implementation of an Algorithm to Find the Convex Hull of a Set of Three-Dimensional Points," *ACM Transactions on Graphics*, Vol. 9, No. 1, Jan. 1990, pp. 105–132.

[50] L. De Floriani, "A Pyramidal Data Structure for Triangle-Based Surface Representation," *IEEE Computer Graphics and Applications*, Vol. 9, Mar. 1989, pp. 67–78.

[51] L. De Floriani, B. Falcidieno, and C. Pienovi, "Delaunay-Based Representation of Surfaces Defined over Arbitrarily Shaped Domains," *Computer Vision, Graphics, and Image Processing*, Vol. 32, 1985, pp. 127–140.

[52] L. De Floriani and E. Puppo, "An On-Line Algorithm for Constrained Delaunay Triangulation," *CVGIP: Graphical Models and Image Processing*, Vol. 54, No. 3, 1992, pp. 290–300.

[53] L. De Floriani, B. Falcidieno, G. Nagy, and C. Pienovi, "On Sorting Triangles in a Delaunay Tessellation," *Algorithmica*, Vol. 6, 1991, pp. 522–532.

[54] B. Delaunay, "Sur la sphere vide," *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7, [Bull. Acad. Sci. U.S.S.R.(VII), Classe Sci. Mat. Nat]*, 1934, pp. 793–800.

[55] P.A. Devijver and M. Dekesel, "Insert and Delete Algorithms for Maintaining Dynamic Delaunay Triangulations," *Pattern Recogn. Lett.*, Vol. 1, 1982, pp. 73–77.

[56] T. Dey, "Triangulation and CSG Representation of Polyhedra with Arbitrary Genus," *Proc. 7th ACM Symp. Comp. Geometry*, 1991, pp. 793–800.

[57] T. Dey, K. Sugihara and C.L. Bajaj, "Delaunay Triangulations in Three Dimensions with Finite Precision Arithmetic," *Computer Aided Geometric Design*, Vol. 9, No. 6, 1992, pp. 457–470.

[58] M.B. Dillencourt, "Realizability of Delaunay Triangulations," *Information Processing Letters*, Vol. 33, No. 6, 1990, pp. 283–287.

[59] G.L. Dirichlet, "Ueber die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen," *J. Reine u. Angew. Math.*, Vol. 40, 1850, pp. 209–227.

[60] H. Djidjev and A. Lingas, "On Computing the Voronoi Diagram for Restricted Planar Figures," *Proc. 2nd Workshop Algorithms Data Struct. Volume 519 of Lecture Notes in Computer Science*, Springer, 1991, pp. 54–64.

[61] D. P. Dobkin, "Computational Geometry and Computer Graphics," *Proc. IEEE*, Vol. 80, No. 9, Sep. 1992, pp. 1400–1411.

[62] D.P. Dobkin and M.J. Laszlo, "Primitives for the Manipulation of Three-Dimensional Subdivisions," *Algorithmica*, Vol. 4, 1989, pp. 3–32.

[63] D. Dobkin, S. Friedman, and K. Supowit, "Delaunay Graphs Are Almost as Good as Complete Graphs," *Disc. and Comp. Geometry*, Vol. 5, 1990, pp. 389–423.

[64] D. Dobkin, S. Levy, W. Thurston, and A. Wilks, "Contour Tracing by Piecewise Linear Approximations," *ACM Trans. on Graphics*, Vol. 9, 1990, 389–423.

[65] R.A. Dwyer, "A Faster Divide and Conquer Algorithm for Constructing Delaunay Triangulation," *Algorithmica*, Vol. 2, 1987, pp. 137–151.

[66] N. Dyn and I. Goren, "Transforming Triangulations in Polygon Domains," *Computer Aided Geometric Design*, Vol. 10, No. 6, Dec. 1993, pp. 531–536.

[67] N. Dyn, D. Levin, and S. Rippa, "Data Dependent Triangulations for Piecewise Linear Interpolation," *IMA Journal of Numerical Analysis*, Vol. 10, 1990, pp. 137–154.

[68] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[69] H. Edelsbrunner, "An Acyclicity Theorem for Cell Complexes in d Dimensions," *Combinotorica*, Vol. 18, 1990, pp. 251–260. Also: *Proceedings of the 5th Annual ACM Symposium on Computation Geometry*, 1989, pp. 145–151.

[70] H. Edelsbrunner and E.P. Muecke, "Simulation of Simplicity, A Technique to Cope with the Degenerate Cases in Geometric Computations," *ACM Trans. Graphics*, Vol. 9, 1990, pp. 66–104.

[71] H. Edelsbrunner and E.P. Muecke, "Three Dimensional Alpha Shapes," *ACM Transactions on Graphics*, Vol. 13, 1994, pp. 43–72.

[72] H. Edelsbrunner and N.R. Shah, "Incremental Topological Flipping Works for Regular Triangulations," *Proceedings of the 8th Annual ACM Symposium on Computational Geometry*, June 1992, pp. 43–52.

[73] H. Edelsbrunner and T.S. Tan, "A Quadratic Time Algorithm for the Minmax Length Triangulation," *Proc. 32nd IEEE Symp. foundations of Comp. Science*, 1991, pp. 414–423.

[74] H. Edelsbrunner and T.S. Tan, "An Upper Bound for Conforming Delaunay Triangulations," *Proc. 8th Symp Comp. Geometry*, 1992, pp. 53–62.

[75] H. Edelsbrunner, T.S. Tan, and R. Waupotitsch, "A Polynomial Time Algorithm for the Minmax Angle Triangulation," *Proc. 5th Symp Comp. Geometry*, 1990.

[76] H. Edelsbrunner, T.S. Tan, and R. Waupotitsch, "O($N^2 \log N$) Time Algorithm for the Minmax Angle Triangulation," *SIAM Journal on Scientific and Statistical Computing*, Vol. 13, No. 4, July 1992, pp. 994–1008.

[77] H. Edelsbrunner, F.P. Preparata, and D.B. West, "Tetrahedrizing Point Sets in Three Dimensions," *J. Symbolic Comp.*, Vol. 10, 1990, pp. 335–347.

[78] M. Elbaz and J.-C. Spehner, "Construction of Voronoi Diagrams in the Plane by Using Maps," *Theoretical Computer Science*, Vol. 77, No. 3, 1990, pp. 331–343.

[79] H. ElGindy and G.T. Toussaint, "On Geodesic Properties of Polygons Relevant to Linear Time Triangulation," *Visual Computer*, Vol. 5, No. 1, 1989, pp. 68–74.

[80] D. Eppstein, "The Farthest Point Delaunay Triangulation Minimizes Angles," *Comput. Geom. Theory Appl.*, Vol. 1, 1992, pp. 143–148.

[81] G. Erlebacher and P.R. Eiseman, "Adaptive Triangular Mesh Generation," *AIAA Journal*, Vol. 25, 1987, pp. 1356–1364.

[82] M.A. Facello, "Implementation of a Randomized Algorithm for Delaunay and Regular Triangulations in Three Dimensions," *Computer Aided Geometric Design*, Vol. 12, pp. 351–370, 1995.

[83] T.P. Fang and L.A. Piegl, "Delaunay Triangulation Using a Uniform Grid," *IEEE Computer Graphics and Application*, Vol. 13, No. 3, pp. 36–47, May 1993.

[84] G. Farin, "A Modified Clough-Tocher Interpolant," *Computer Aided Geometric Design*, Vol. 2, Nos. 1–3, pp. 19–27.

[85] G. Fekete, "Rendering and Managing Spherical Data with Sphere Quadtrees," *Proceedings of Visualization '90*, IEEE Computer Society Press, 1990, pp. 176–186.

[86] D. Field, "Implementing Watson's Algorithm in Three Dimension," *Pro. 2nd ACM Symp. Comp. Geometry*, 1986, pp. 246–259.

[87] D. Field, "A Generic Delaunay Triangulation Algorithm for Finite Element Meshing," *Adv. Eng. Software*, Vol. 13, 1991, pp. 263–272.

[88] D. Field, "Laplacian Smoothing and Delaunay Triangulations," *Comm. in Applied Numer. Analysis*, Vol. 4, 1988, pp. 709–712.

[89] D. Field and W.D. Smith, "Graded Tetrahedral Finite Element Meshes," *Int. J. Numer. Meth. Eng.*, Vol. 31, 1991, pp. 413–425.

[90] R. Forrest, "Computational Geometry," *Proc. Royal Society London*, Volume 321, Series 4, 1971, pp. 187–195.

[91] S. Fortune, "Numerical Stability of Algorithms for 2-d Delaunay Triangulations and Voronoi Diagrams," *Proc. 8th Annual. ACM Symposium. Comput. Geom.*, 1992, pp. 83–92.

[92] S. Fortune, "Voronoi Diagrams and Delaunay Triangulations," *Computing in Euclidean Geometry*, F.K. Hwang and D.-Z. Du, editors, World Scientific, Singapore, 1992, pp. 193–233.

[93] S. Fortune, "Sweepline Algorithm for Voronoi Diagrams," *Algorithmica*, Vol. 2, No. 2, 1987, pp. 153–174.

[94] A. Fournier and D.Y. Montuno, "Triangulating Simple Polygons and Equivalent Problems," *ACM Transaction on Graphics*, Vol. 3, No. 2, Apr. 1984, pp. 153–174.

[95] R.J. Fowler and J.J. Little, "Automatic Extraction of Irregular Network Digital Terrain Models," *Computer Graphics*, Vol. 13, No. 2, Aug. 1979, pp. 199–207.

[96] R. Franke, "Scattered Data Interpolation: Tests of Some Methods," *Math. Comp.*, Vol. 38, 1982, pp. 181–200.

[97] R. Franke and G. Nielson, "Surface Construction Based upon Triangulations," *Surfaces in Computer Aided Geometric Design*, Springer, 1983, pp. 163–179.

[98] R. Franke and G. Nielson, "Scattered Data Interpolation and Applications: A Tutorial and Survey," *Geometric Modelling: Methods and their Application*, H. Hagen and D. Roller, editors, Springer, 1990.

[99] H. Freudenthal, "Simplizialzerlegungen von beschraenkter Flachheit," *Ann. Math.*, Vol. 43, 1942, pp. 580–582.

[100] W.H. Frey and D.A. Field, "Mesh Relaxation: A New Technique for Improving Meshes," *Int. J. Numer. Neth. Eng.*, Vol. 31, 1991, pp. 1121–1133.

[101] M.R. Garey, D.S. Johnson, F.P. Preparata, and R.E. Tarjan, "Triangulating a Simple Polygon," *Inform. Process. Lett.*, Vol. 7, 1978, pp. 175–179.

[102] P.L. George and F. Hermeline, "Delaunay's Mesh of a Convex Polyhedron in Dimension d. Application to Arbitrary Polyhedra," *International Journal for Numerical Methods in Engineering*, Vol. 33, No. 5, Apr. 1992, pp. 975–995.

[103] J. Gleue, *Triangulierung und Interpolation von im R^2 unregelmaessig verteilten Daten*, HMI B 357, 1981.

[104] M.T. Goodrich, "Efficient Piecewise-Linear Function Approximation Using the Uniform Metric," *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 322–331.

[105] S. Goldman, "A Space Efficient Greedy Triangulation Algorithm," *Information Processing Letters*, Vol. 31, No. 4, 1989, pp. 191–196.

[106] T. Gonzalez and M. Razzazi, "Properties and Algorithms for Constrained Delaunay Triangulations," *Proc. 3rd Canad. Conf. Comput. Geom.*, 1991, pp. 114–117.

[107] P.J. Green and R. Sibson, "Computing Dirichlet Tessellations in the Plane," *The Computer Journal*, Vol. 21, 1978, pp. 168–173.

[108] P.J. Green and B.W. Silverman, "Constructing the Convex Hull of a Set of Points in the Plane," *The Computer Journal*, Vol. 22, No. 3, 1979, pp. 262.

[109] J.A. Gregory, "Error Bounds for Linear Interpolation on Triangles," *The Mathematics of Finite Elements and Application II*, J.R. Whiteman, editor, Academic Press, London, 1975, pp. 163–170.

[110] J.A. Gregory, "A Blending Function Interpolant for Triangles," *Multivariate Approximation*, D.G. Handscomb. editor, Academic Press, London.

[111] J.A. Gregory, "Interpolation to Boundary Data on the Simplex," *Computer Aided Geometric Design*, Vol. 2, Nos. 1–3, pp. 43–52.

[112] J.A. Gregory, "Error Bounds for Linear Interpolation on Triangles," *The Mathematics of Finite Elements and Applications II*, J. Whiteman, editor, Academic Press, London, 1975, pp. 163–170.

[113] L. Guibas and J. Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams," *ACM Trans. Graphics*, Vol. 4, 1985, pp. 74–123.

[114] L.J. Guibas, D.E. Knuth, and M. Sharir, "Randomized Incremental Construction of Delaunay and Voronoi Diagrams," *Automata, Languages and Programming*, LNCS N.443, Springer-Verlag, 1990, pp. 414–431.

[115] A.J. Hansen and P.L. Levin, "On Conforming Delaunay Mesh Generation," *Adv. Engineering Software*, Vol. 14, No. 2, 1992, pp. 129–135.

[116] D. Hansford, "The Neutral Case for the Min-Max Triangulation," *CAGD*, Vol. 7, 1990, pp. 431–438.

[117] F. Hermeline, "Triangulation automatique d'un polyedre in dimension n," *RAIRO Anal. Numer.*, Vol. 76, 1982, pp. 211–242.

[118] C. Hazelwood, "Approximating Constrained Tetrahedrizations," *Computer Aided Geometric Design*, Vol. 10, No. 1, pp. 67–87.

[119] S. Hertel and K. Mehlhorn, "Fast Triangulation of Simple Polygons," *4th Conf. Foundations of Computation Theory*, Springer LNCS 158, 1983, pp. 207–218.

[120] H. Jin and R.I. Tannel, "Generation of Unstructured Tetrahedral Meshes by Advancing Front Technique," *Internat. J. Numer. Meth. Eng.*, Vol. 36, 1993, pp. 1805–1823.

[121] B. Joe, "ree-Dimensional Triangulations from Local Transformations," *SIAM Journal Sci. Stat. Comput.*, Vol. 10, pp. 718–741, 1989.

[122] B. Joe, "Construction of Three Dimensional Delaunay Triangulations Using Local Transformations," *Computer Aided Geometric Design*, Vol. 8, No. 2, pp. 123–142, 1991.

[123] B. Joe and C.A. Wang, "Duality of Constrained Voronoi Diagrams and Delaunay Triangulations," *Algorithmica*, Vol. 9, No. 2, 1993, pp. 149–155.

[124] D.-M. Jung, "An Optimal Algorithm for Constrained Delaunay Triangulation," *Proceedings Twenty-Sixth Annual Allerton Conference on Communication, Control and Computing*, Urbana, Ill., 1988, pp. 85–86.

[125] Y.H. Jung and K. Lee, "Tetrahedron-Based Octree Encoding for Automatic Mesh Generation," *Computer Aided Design*, Vol. 25, 1993, pp. 141–153.

[126] T.C. Kao and D.M. Mount, "An Algorithm for Computing Compacted Voronoi Diagrams Defined by Convex Distance Functions," *Proc. 3rd Canad. Conf. Comput. Geom.*, 1991, pp. 104–109.

[127] T.C. Kao and D.M. Mount, "Incremental Construction and Dynamic Maintenance of Constrained Delaunay Triangulations," *Proc. 4th Canad. Conf. Comput. Geom.*, 1992, pp. 170–175.

[128] M.D. Karasick, D. Lieber, and L.R. Nackman, "Efficient Delaunay Triangulation Using Rational Arithmetic," *ACM Transactions on Graphics*, Vol. 10, No. 1, Jan. 1991, pp. 71–91.

[129] J. Katajainen and M. Koppinen, "Constructing Delaunay Triangulations by Merging Buckets in Quadtree Order," *Annales Societatis Mathematicae Polonae, Series IV, Fundamenta Informaticae*, Vol. 11, No. 3, 1988, pp. 275–288.

[130] D.G. Kirkpatrick, "A Note on Delaunay and Optimal Triangulations," *Inform. Process. Lett.*, Vol. 10, 1990, pp. 127–128.

[131] D.G. Kirkpatrick, M.M. Klawe, and R.E. Tarjan, "Polygon Triangulation in O(n log log n) Time with Simple Data Structures," *Proc. 6th Annual ACM Symposium. Comput. Geom.*, 1990, pp. 34–43.

[132] V. Klee, "On the Complexity of d-Dimensional Voronoi Diagrams," *Arch. Math.*, Vol. 34, 1980, pp. 75–80.

[133] R. Klein, "Concrete and Abstract Voronoi Diagrams," *Volume 400 of Lecture Notes in Computer Science*, Springer, 1989.

[134] R. Klein and A. Lingas, "A Note on Generalizations of Chew's Algorithm for the Voronoi Diagram of a Simple Polygon," *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993, pp. 124–132.

[135] G.T. Klincsek, "Minimal Triangulations of Polygonal Domains," *Ann. Disc. Math.*, Vol. 9, 1980, pp. 121–123.

[136] D. Knuth, *The Art of Computer Programming, Volume 1; Fundamental Algorithms*, Addison Wesley, Reading, Mass., 1973.

[137] H.W. Kuhn, "Simplicial Approximation of Fixed Points," *Proc. Nat. Acad. Sci. USA*, Vol. 61, 1968, pp. 1238–1242.

[138] C. Lawson, "Transforming Triangulations," *Discrete Mathematics*, Vol. 3, 1972, pp. 365–372.

[139] C. Lawson, "Software for C^1 Surface Interpolation," *Mathematical Software III*, J.R. Rice, editor, Academic Press, New York, 1977, pp. 161–194.

[140] C. Lawson, "Properties of n-Dimensional Triangulations," *Computer Aided Geometric Design*, Vol. 3, No. 4, pp. 231–246.

[141] C. Lawson, "C^1 Surface Interpolation for Scattered Data on a Sphere," *Rocky Mountain Journal of Mathematics*, Vol. 14, No. 1, Winter 1984, pp. 177–202.

[142] C. Lee, "Regular Triangulations of Convex Polytopes," *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, Gritzmann and B. Strumfels, editors, Amer. Math. Soc., Providence, RI, 1991, pp. 443–456.

[143] D.T. Lee, "Two Dimensional Voronoi Diagram in the L_p-Metric," *J. ACM*, Vol. 27, 1980, pp. 604–618.

[144] D.T. Lee and A. Lin, "Generalized Delaunay Triangulation for Planar Graphs," *Disc. and Comp. Geometry*, Vol. 1, 1986, pp. 201–217.

[145] D.T. Lee and C.K. Wong, "Voronoi Diagrams in L_1 (L_∞) Metrics with 2-Dimensional Storage Applications," *SIAM J. Comput.*, Vol. 9, 1980, pp. 200–211.

[146] D.T. Lee, and B.J. Schacter, "Two Algorithms for Constructing a Delaunay Triangulation," *Int. J. of Computer and Information Science*, Vol. 9, No. 3, 1980, pp. 219–242.

[147] J. Lee, "Comparison for Existing Methods for Building Triangular Irregular Network Models of Terrain from Grid Digital Elevation Models," *Int. J. of Geographical Information Systems*, Vol. 5, No. 2, July–Sep. 1991, pp. 267–285.

[148] K. Lee, *Data Dependent Tetrahedrizations*, Ph.D. thesis, Arizona State University, 1995.

[149] N.J. Lennes, "Theorems on the Simple Finite Polygon and Polyhedron," *American Journal of Mathematics*, Vol. 33, 1911, pp. 37–62.

[150] C. Levcopoulos and A. Lingas, "On Approximation Behavior of the Greedy Triangulation for Convex Polygons," *Algorithmica*, Vol. 2, 1987, pp. 175–193.

[151] B.A. Lewis and J.S. Robinson, "Triangulation of Planar Regions with Applications," *Computer J.*, Vol. 21, 1978, pp. 324–332.

[152] A. Lingas, *Advances in Minimum Weight Triangulation*, Ph.D. thesis, Linkoeping Univ., 1983.

[153] A. Lingas, "Voronoi Diagrams with Barriers and the Shortest Diagonal Problem," *Inform. Process. Lett.*, Vol. 32, 1989, pp. 191–198.

[154] D. Lischinski, "Incremental Delaunay Triangulation,"*Graphic Gems IV*, Paul S. Heckbert, editor, Academic Press, 1994, pp. 47–59.

[155] E. L. Lloyd, On Triangulations of a Set of Points in the Plane," *Proc. 18th IEEE Symp. Found. Comp. Sci.*, 1977, pp. 228–240.

[156] S. Lo, "Delaunay Triangulations of Nonconvex Planar Domains," *Int. J. Numer. Meth. Eng.*, Vol. 28, 1989, pp. 2695–2707.

[157] S. Lo, "Volume Discretizations Into Tetrahedra. I. Verification and Orientation of Boundary Surfaces," *Computers and Structures*, Vol. 39, 1991, pp. 493–500.

[158] R. Loehner and P. Parikh, Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method, Internat. J. Numer. Meht. Fluids, Vol. 8, 1988, pp. 1135–1149.

[159] M.K. Loze and R. Saunders, "Two Simple Algorithms for Constructing a Two-Dimensional Constrained Delaunay Triangulation," *Applied Numerical Mathematics*, Vol. 11, 1993, pp. 403–418.

[160] W. Lorensen and H.E. Cline, "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm," *SIGGRAPH 87 Conference Proceedings, Computer Graphics*, Vol. 21, No. 4, July 1987, pp. 163–169.

[161] G. Macedonio and M.T. Pareschi, "An Algorithm for the Triangulation of Arbitrarily Distributed Points: Applications to Volume Estimate and Terrain Fitting," *Computers & Geosciences*, Vol. 17, No. 7, 1991, pp. 859–874.

[162] G.K. Manacher, and A.L. Zobrist, "Neither the Greedy nor the Delaunay Triangulation Approximates the Optimum," *Inform. Process. Lett.*, Vol. 9, 1979, pp. 31–34.

[163] L. Mansfield, "Interpolation to Boundary Data in Tetrahedra with Applications to Compatible Finite Elements," *J. Mat. Anal. Appl.*, Vol. 56, pp. 137–164.

[164] G. Marton, "Acceleration of Ray Tracing Via Voronoi Diagrams," *Graphic Gems V*, Alan Paeth, editor, Academic Press, 1995, pp. 268–284.

[165] A. Maus, "Delaunay Triangulation and the Convex Hull of n Points in Expected Linear Time," *BIT*, Vol. 24, 1984, pp. 151–163.

[166] N. Max, "Sorting for Polyhedron Compositing," *Focus on Scientific Visualization*, H. Hagen, H. Mueller, G.M. Nielson, editors, Springer, 1993, pp. 259–268.

[167] N. Max, P. Hanrahan, and R. Crawfis, "Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions," *Computer Graphics*, Vol. 24, Nov. 1990, pp. 27–33.

[168] A. Mirante and N. Weingarten, "The Radial Sweep Algorithm for Constructing Triangulated Irregular Networks," *IEEE Computer Graphics and Applications*, May 1982, pp. 11–21.

[169] G.H. Meisters, "Polygons Have Ears," *Amer. Math. Monthly*, Vol. 82, 1975, pp. 648–651.

[170] D. Moore, "Subdividing Simplices," *Graphics Gems III*, D. Kirk, editor, Academic Press, 1992, pp. 244–249.

[171] D. Moore, "Understanding Simploids," *Graphics Gems III*, D. Kirk, editor, Academic Press, 1992, pp. 250–255.

[172] J.-M. Moreau and P. Volino, "Constrained Delaunay Triangulation Revisited," *Proc. 5th Canad. Conf. Comput. Geom.*, 1993, pp. 340–345.

[173] D.E. Muller and F.P. Preparata, "Finding the Intersection of Two Convex Polyhedra," *Theoretical Computer Science*, Vol. 7, 1978, pp. 217–236.

[174] E.J. Nadler, *Piecewise Linear Approximation on Triangulations of a Planar Region*, Ph.D. thesis, Brown University, Division of Applied Mathematics, 1985.

[175] A. Narkhede and D. Manocha, "Fast Polygon Triangulation Based on Seidel's Algorithm," *Graphic Gems V*, Academic Press, 1995, pp. 394–397.

[176] J.M. Nelson, "A Triangulation Algorithm for Arbitrary Planar Domains," *Appl. Math. Modelling*, Vol. 2, 1978, pp. 151–159.

[177] G.M. Nielson, "The Side-Vertex Method for Interpolation in Triangles," *Journal of Approx. Theory*, Vol. 25, 1979, pp. 318–336.

[178] G.M. Nielson, "Minimum Norm Interpolation in Triangles," *SIAM Journal Numer. Analysis*, Vol. 17, 1980, pp. 46–62.

[179] G.M. Nielson, "A Method for Interpolating Scattered Data Based upon a Minimum Norm Network," *Mathematics of Computation*, Vol. 40, 1983, pp. 253–271.

[180] G M. Nielson, *An Example with a Local Minimum for the Minmax Ordering of Triangulations*, Arizona State University Computer Science Technical Report TR-87-014, 1987.

[181] G.M. Nielson, "Coordinate Free Scattered Data Interpolation," *Topics in Multivariate Approximation*, C. Chui, F. Utreras, L. Schumaker, editors, Academic Press, New York, 1987, pp. 175–184.

[182] G.M. Nielson, "A Characterization of an Affine Invariant Triangulation," *Geometric Modelling*, Computing Supplementum 8, G. Farin, H. Hagen, H. Noltemeier, W. Knoedel, editors, Springer, 1993, pp. 191–210.

[183] G.M. Nielson, *How Many Ways Can a Cube Be Subdivided Into Tetrahedra?*, Arizona State University Computer Science Department Technical Report TR-95-13, 1995.

[184] G.M. Nielson and T. Foley, "A Survey of Applications of an Affine Invariant Metric," *Mathematical Methods in Computer Aided Geometric Design*, T. Lyche and L.L. Schumaker, editors, Academic Press, New York, 1989, pp. 445–467.

[185] G.M. Nielson and R. Ramaraj, "Interpolation over a Sphere," *Computer Aided Geometric Design*, Vol. 4, 1987, pp. 41–57.

[186] G.M. Nielson and B. Hamann, "The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes," *Proceedings of Visualization '91*, IEEE Computer Society Press, Los Alamitos, Calif., 1990, pp. 83–91.

[187] G.M. Nielson and K. Opitz, "The Face-Vertex Method for Interpolating in Tetrahedra," *Workshop on Computational Geometry*, A. Conte, V. Demichelis, F. Fontanella and I. Galligani, editors, World Scientific, 1993, pp. 231–244.

[188] G.M. Nielson and J. Tvedt, "Comparing Methods of Interpolation for Scattered Volumetric Data," *State of the Art in Computer Graphics—Aspects of Visualization*, D. Rogers and R.A. Earnshaw editors, Springer-Verlag, 1994, pp. 67–86.

[189] G.M. Nielson, D.H. Thomas, and J.A. Wixom, "Interpolation in Triangles," *Bull. Austral. Math. Soc.*, Vol. 20, 1979, pp. 115–130.

[190] T. Ohya, M. Iri, and K. Murota, "Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparison of Various Algorithms," *Journal of the Operations Research Society of Japan*, Vol. 27, No. 4, 1984, pp. 306–336.

[191] A. Okabe, B. Boots, and K. Sugihara, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, Wiley & Sons, 1992.

[192] A.A. Oloufa, "Triangulation Applications in Volume Calculation," *Journal of Computing in Civil Engineering*, Vol. 5, No. 1, Jan. 1991, pp. 103–121.

[193] T.K. Peucker, R.J. Fowler, and J.J. Little, "The Triangulated Irregular Network," *Proceedings ASP-ACSM Symposium on Digital Terrain Models*, 1978.

[194] C.S. Peterson, "Adaptive Contouring of Three-Dimensional Surfaces," *CAGD*, Vol. 1, 1984, pp. 61–74.

[195] L.A. Piegl and A.M. Richard, "Algorithm and Data Structure for Triangulating Multiply Connected Polygonal Domains," *Computers & Graphics*, Vol. 17, No. 5, 1993, pp. 563–574.

[196] D.A. Plaisted and J. Hong, "A Heuristic Triangulation Algorithm," *J. Algorithms*, Vol. 8, 1987, pp. 405–437,

[197] M. Pourazady and M. Radhakrishnan, "Optimization of a Triangular Mesh," *Computers and Structures*, Vol. 40, No. 3, 1991, pp. 795–804.

[198] M.J.D. Powell, "Piecewise Quadratic Approximation on Triangles," *Software for Numerical Mathematics*, D.J. Evans, editor, Academic Press, New York, 1974.

[199] M.J.D. Powell and M.A. Sabin, "Piecewise Quadratic Approximation on Triangles," *ACM Trans. on Mathematical Software*, Vol. 3, 1977, pp. 316–325.

[200] P.L. Power, "Minimal Roughness Property of the Delaunay Triangulation: A Shorter Approach," *Computer Aided Geometric Design*, Vol. 9, 1992, pp. 491–494.

[201] P.L. Power, "The Neutral Case for the Min-Max Angle Criterion: A Generalized Approach," *Computer Aided Geometric Design*, Vol. 9, 1992, pp. 413–418.

[202] F.P. Preparata and S.J. Hong, "Convex Hull of a Finite Set of Points in Two and Three Dimension," *Commun. ACM*, Vol. 20, No. 2, Feb. 1977, pp. 87–93.

[203] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[204] E. Quak and L.L. Schumaker, "Cubic Spline Fitting Using Data Dependent Triangulations," *Computer Aided Geometric Design*, Vol. 7, Nos. 1–4, 1990, pp. 293–301.

[205] E. Quak and L.L. Schumaker, "C^1 Surface Fitting Using Data Dependent Triangulations," *Approximation Theory VI*, C. Chui, L.L. Schumaker, and J. Ward, editors, Academic Press, 1989, pp. 545–548.

[206] E. Quak and L.L. Schumaker, "Least Squares Fitting by Linear Splines on Data Dependent Triangulations," *Curves and Surfaces*, P.-J. Laurent, A. Le Mehaute and L.L. Schumaker, editors, Academic Press, 1991, pp. 387–390.

[207] R.J. Renka, "Algorithm 624: Triangulation and Interpolation of Arbitrarily Distributed Points in the Plane," *ACM TOMS*, Vol. 10, 1984, pp. 440–442.

[208] V.T. Rajan, "Optimality of the Delaunay Triangulation in R^d," *Proc. 7th ACM Symp. Comp. Geometry*, 1991, pp. 357–363.

[209] P.N. Rathie, "A Census of Simple Planar Triangulations," *J. Comb. Theory B*, Vol. 16, 1974, pp. 134–138.

[210] D. Rhynsburger, "Analytic Delineation of Thiessen Polygons," *Geograph. Anal.*, Vol. 5, 1973, pp. 133–144.

[211] S. Rippa, "Minimal Roughness Property of the Delaunay Triangulation," *Computer Aided Geometric Design*, Vol. 7, 1990, pp. 489–497.

[212] S. Rippa, "Long and Thin Triangles Can Be Good for Linear Interpolation," *SIAM Journal on Numerical Analysis*, Vol. 29, No. 1, Feb. 1992, pp. 257–270.

[213] S. Rippa, *Piecewise Linear Interpolation and Approximation Schemes over Data Dependent Triangulations*, Ph.D. thesis, Tel Aviv, 1989.

[214] S. Rippa and B. Schiff, "Minimum Energy Triangulations for Elliptic Problems," *Comp. Meth. in Applied Mech. and Eng.*, Vol. 84, 1990, pp. 257–274.

[215] C.A. Rogers, *Packing and Covering*, Cambridge University Press, 1964.

[216] J. Ruppert and R. Seidel, "On the Difficulty of Tetrahedralizing 3-Dimensional Non-convex Polyhedra," *Proc. 5th ACM Symp. Comp. Geometry*, 1989, pp. 380–393.

[217] N. Sapidis and R. Perucchio, "Delaunay Triangulation of Arbitrarily Shaped Planar Domains," *Computer Aided Geometric Design*, Vol. 8, 1991, pp. 421–438.

[218] V. Sarin and S. Kapoor, "Algorithms for Relative Neighbourhood Graphs and Voronoi Diagrams in Simple Polygons," *Proc. 4th Canad. Conf. Comput. Geom.*, 1992, pp. 292–298.

[219] L. Scarlatos and T. Pavlidis, "Optimizing Triangulation by Curvature Equalization," *Proceedings of Visualization '92*, IEEE Computer Society Press, Oct. 1992, pp. 333–339.

[220] B. Schachter, "Decomposition of Polygons Into Convex Sets," *IEEE Transactions on Computing C-27*, Vol. 11, Nov. 1978, pp. 1078–1082.

[221] E. Schoenhardt, "Ueber die Zerlegung von Dreieckspolyedern in Tetraeder," *Math. Annalen*, Vol. 98, 1928, 309–312.

[222] W.J. Schroeder and M.S. Shephard, "Geometry-Based Fully Automatic Mesh Generation and the Delaunay Triangulation," *Int. J. Numer. Meth. Eng.*, Vol. 26, 1988, pp. 2503–2515.

[223] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen, "Decimation of Triangle Meshes," *SIGGRAPH '92*, Vol. 26, July 1992, pp. 65–70.

[224] L.L. Schumaker, "Fitting Surfaces to Scattered Data," *Approximation Theory II*, G.G. Lorentz, C.K. Chui, and L.L. Schumaker, editors, Academic Press, 1976, pp. 203–268.

[225] L.L. Schumaker, "Computing Optimal Triangulations Using Simulated Annealing," *Computer Aided Geometric Design*, Vol. 10, Nos. 3–4, pp. 329–345.

[226] L.L. Schumaker, "Triangulation Methods," *Topics in Multivariate Approximation*, L.L. Schumaker, C. Chui and F. Utreras, editors, Academic Press, New York, 1987, pp. 219–232.

[227] L.L. Schumaker, "Triangulations Methods in CAGD," *IEEE Computer Graphics and Applications*, Vol. 13, Jan. 1993, pp. 47–52.

[228] A. Seidel, "Constrained Delaunay Triangulations and Voronoi Diagrams with Obstacles," *1978–1988 Ten Years IIG*, H.S. Poingratz and W. Schinnerl, editors, 1988, pp. 178–191.

[229] M. Senechal, "Which Tetrahedra Fill Space?," *Math. Magazine*, Vol. 54, 1981, pp. 227–243.

[230] M.I. Shamos, *Computational Geometry*, Ph.D. dissertation, Yale University, 1978.

[231] M. Shapiro, "A Note on Lee and Schachter's Algorithm for Delaunay Triangulation," *International Journal of Computer and Information Sciences*, Vol. 10, No. 6, 1981, pp. 413–418.

[232] D N. Shenton and Z.J. Cendes, "Three-Dimensional Finite Element Mesh Generation Using Delaunay Tessellation," *IEEE Trans. on Magnetics, MAG-21*, 1985, pp. 2535–2538.

[233] D. Shirley and A. Tuchman, "A Polygonal Approximation to Direct Scalar Volume Rendering," *Computer Graphics*, Vol. 24, Nov. 1990, pp. 63–70.

[234] G.M. Shute, L.L. Deneen, and C.D. Thomborson, "An O(N log N) Plane-Sweep Algorithm for L_1 and L_∞ Delaunay triangulations," *Algorithmica*, Vol. 6, 1978, pp. 207–221

[235] R. Sibson, "Locally Equiangular Triangulations," *Computer J.*, Vol. 21, 1978, pp. 243–245.

[236] R. Sibson, "A Brief Description of Natural Neighbour Interpolation," *Chapter 2 of Interpreting Multivariate Data*, Wiley, New York, 1981.

[237] C.T. Silva, J.S.B. Mitchell, and A.E. Kaufman, "Automatic Generation of Triangular Irregular Networks Using Greedy Cuts," *Proceedings of Visualization '95*, IEEE Computer Society Press, Oct. 1995, pp. 201–208.

[238] S.W. Sloan, "A Fast Algorithm for Constructing Delaunay Triangulations in the Plane," *Advances in Engineering Software*, Vol. 9, Jan. 1987, pp. 34–55.

[239] S.W. Sloan and G.T. Houlsby, "An Implementation of Watson's Algorithm for Computing 2-Dimensional Delaunay Triangulations," *Advances in Engineering Software*, Vol. 6, 1984, pp. 192–197.

[240] C. Stein, B. Becker, and N. Max, "Sorting and Hardware Assisted Rendering for Volume Visualization," *1994 Symposium on Volume Visualization*, Washington, D.C., Oct. 1994, pp. 83–89.

[241] K. Sugihara and M. Iri, "Construction of the Voronoi Diagram for "One Million" Generators in Single-Precision Arithmetic," *Proceedings of the IEEE*, Vol. 80, 1992, pp. 1471–1484.

[242] M. Tanemura, T. Ogawa, and W. Ogita, "A New Algorithm for Three-Dimensional Voronoi Tessellation," *Journal of Computational Physics*, Vol. 51, 1983, pp. 191–207.

[243] R.E. Tarjan and C.J. Van Wyk, "An O(n log log n)-Time Algorithm for Triangulating a Simple Polygon," *SIAM J. Comput.*, Vol. 17, 1988, pp. 143–178.

[244] A.H. Thiessen, "Precipitation Averages for Large Areas," *Monthly Weather Review*, Vol. 39, 1911, pp. 1032–1034.

[245] J.F. Thompson, *Numerical Grid Generation*, North-Holland, 1982.

[246] J.C. Tipper, "Straightforward Iterative Algorithm for the Planar Voronoi Diagram," *Information Processing Letters*, Vol. 34, No. 3, Apr. 1990, pp. 155–160.

[247] J.C. Tipper, "FORTRAN Programs to Construct the Planar Voronoi Diagram," *Computers & Geosciences*, Vol. 17, 1991, pp. 597–632.

[248] G. Toussaint, "Efficient Triangulation of Simple Polygons," *Visual Comput.*, Vol. 7, 1991, pp. 280–295.

[249] G.T. Toussaint, C. Verbrugge, C. Wang, and B. Zhu, "Tetrahedrization of Simple and Not Simple Polyhedra," *CCCG Proc. of the Fifth Canadian Conference on Computational Geometry*, 1994.

[250] V.J.D. Tsai, "Delaunay Triangulation in TIN Creation: An Overview and a Linear-Time Algorithm," *Int. J. Geographical Information Systems*, Vol. 7, 1993, pp. 501–524.

[251] W.T. Tutte, "A Census of Planar Triangulations," *Canadian J. Math.*, Vol. 14, 1962, pp. 21–38.

[252] G. Voronoi, "Nouvelles applications des paramatres continusala theorie des formes quadratiques, Deuxieme Memoire, Recherches sur les parallelloedres primitifs," *J. reine angew. Mathe.*, Vol. 134, 1908, pp. 198–287.

[253] C. Wang and L. Schubert, "An Optimal Algorithm for Constructing the Delaunay Triangulation of a Set of Line Segments," *Proc. 3rd ACM Symp. Comp. Geometry*, 1987, pp. 223–232.

[254] D.F. Watson, "Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes," *Comp. J.*, Vol. 24, 1981, pp. 167–172.

[255] D.F. Watson and G.M. Philip, "Systematic Triangulations," *Computer Vision, Graphics, and Image Processing*, Vol. 26, 1984, pp. 217–223.

[256] N.D. Weatherhill and O. Hassan, "Efficient Three-Dimensional Delaunay Triangulation with Automatic Point Creation and Imposed Boundary Constraints," *Internat. J. Numer. Meth. Eng.*, Vol. 37, 1994, pp. 2005–3039.

[257] P. Williams, "Visibility Ordering Meshed Polyhedra," *ACM Transactions on Graphics*, Vol. 11, No. 2, 1992, pp. 103–126.

[258] B. Woerdenweber, *Automatic Mesh Generation of 2- and 3-Dimensional Curvilinear Manifolds*, Ph.D. dissertation, University of Cambridge, 1981.

[259] B. Woerdenweber, "Finite-Element Analysis for the Naive User," *Solid Modeling by Computers from Theory to Applications*, M.S. Pickett and J. Boyse, editors, Plenum, 1984, pp. 81–100.

[260] A.J. Worsey and B. Piper, "A Trivariate Powell-Sabin Interpolant," *Computer Aided Geometric Design*, Vol. 5, No. 3, 1988 pp. 177–186.

[261] A.J. Worsey and G. Farin, "An n-Dimensional Clough-Tocher Interpolant," *Constructive Approximation*, Vol. 3, 1987, pp. 99–110.

[262] F.F. Yao, "Computational Geometry," *Handbook of Theoretical Computer Science*, Vol. A, Chapter 7, J. van Leeuwen, editor, Elsevier and MIT Press, 1990, pp. 343–389.

[263] A. Zenisek, "Polynomial Approximation on Tetrahedrons in the Finite Element Method," *J. Approximation Theory*, Vol. 7, 1973, pp. 334–351.

3

Interval Volume Tetrahedrization

by

Gregory M. Nielson
&
Junwon Sung

Computer Science
Arizona State University
Tempe, AZ 85287-5406
nielson|jsung@asu.edu

Abstract

The interval volume is a generalization of the isosurface commonly associated with the marching cubes algorithm. Based upon samples at the locations of a 3D rectilinear grid, the mc algorithm produces a triangular approximation to the surface defined by $F(x,y,z)=c$. The interval volume is defined by $\alpha \le F(x,y,z) \le \beta$. We describe an algorithm for computing a tetrahedrization of a polyhedral approximation to the interval volume.

1. Introduction

In this paper we describe an algorithm for computing a tetrahedrization of an interval volume. The interval volume is a generalization of the isosurface commonly associated with the marching cubes algorithm (see [4]). It is assumed that a trivariate function $F(x,y,z)$ has been sampled at domain points lying on a 3D rectilinear grid. Given a threshold value, c, the mc algorithm produces a triangulated approximation to the surface $S_F(c) = \{(x,y,z): F(x,y,z) = c\}$. The interval volume is defined as $I_F(\alpha,\beta) = \{(x,y,z): \alpha \le F(x,y,z) \le \beta\}$. We describe an algorithm for computing a tetrahedrization of a polyhedral approximation to the interval volume. The uses and benefits of the interval volume have been delineated rather well in earlier papers on this subject (see Guo[3] and Fujishiro, Maeda and Sato [2]) and so we will not repeat them here. The algorithm of Guo [3] is based on the alpha shapes of Edelsbrunner and Mucke [1]. The algorithm of Fujishiro, Maeda and Sato [2] computes for each voxel the intersecton of two convex polyhedra; namely $I_F(-\infty,\beta)$ and $I_F(\alpha,+\infty)$ which are approximated by the polygon surface of the mc algorithm [4] appropriately adjusted forht ambiguous cases (see Nielson and Hamann [6]). For the discussion here, the two surfaces which bound the interval volume, $S_F(\alpha)$ and $S_F(\beta)$ are referred to as the α-surface and the β-surface respectively.

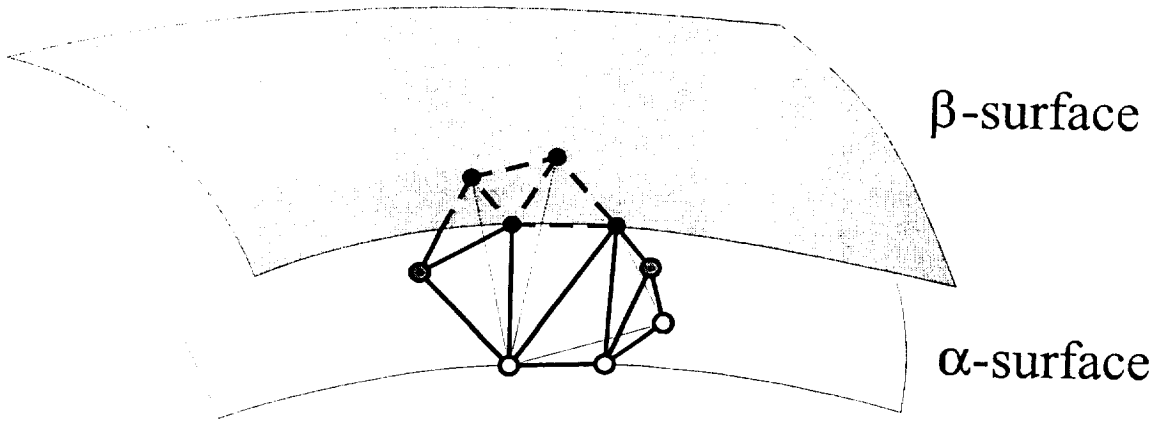


Figure 1. Tetrahedrization of the Interval Volume.

While it is impossible to anticipate all the application domains of algorithms for computing interval volumes, in this paper, we are primarily interested in applications where samples of the data function $F(x,y,z)$ over a 3D rectilinear grid are known. This includes the typical data sets associated with MRI and CAT scans. In order to make the interval volume well defined, we make some inference about the variation of the data function, $F(x,y,z)$, over the voxel domains defined by the vertices of the rectilinear grid. Our approach is very simple and leads to a two step algorithm:

Step 1. Each voxel is decomposed into tetrahedra and F is assumed to vary lineary over each tetrahedron.

Step 2. For each tetrahedron of each voxel, the interval volume is computed and decomposed into tetrahedra.

For the first step, there are several choices for decomposing a voxel into a collection of tetraheda. Since we require that the final collection of tetrahedra to be a proper tetrahedrization of the complete interval volume it is important that this first tetrahedral decomposition step also be a tetrahedrization. This implies, among other things, that the diagonals form the tetrahedization of one voxel to the next must match. We normally use the decomposition shown in Figure 2 which leads to five tetrahedra per voxel. The five-tetrahedron decomposition must be alternated from voxel to voxel with a rotated version of itself in order to maintain a proper tetrahedrization.

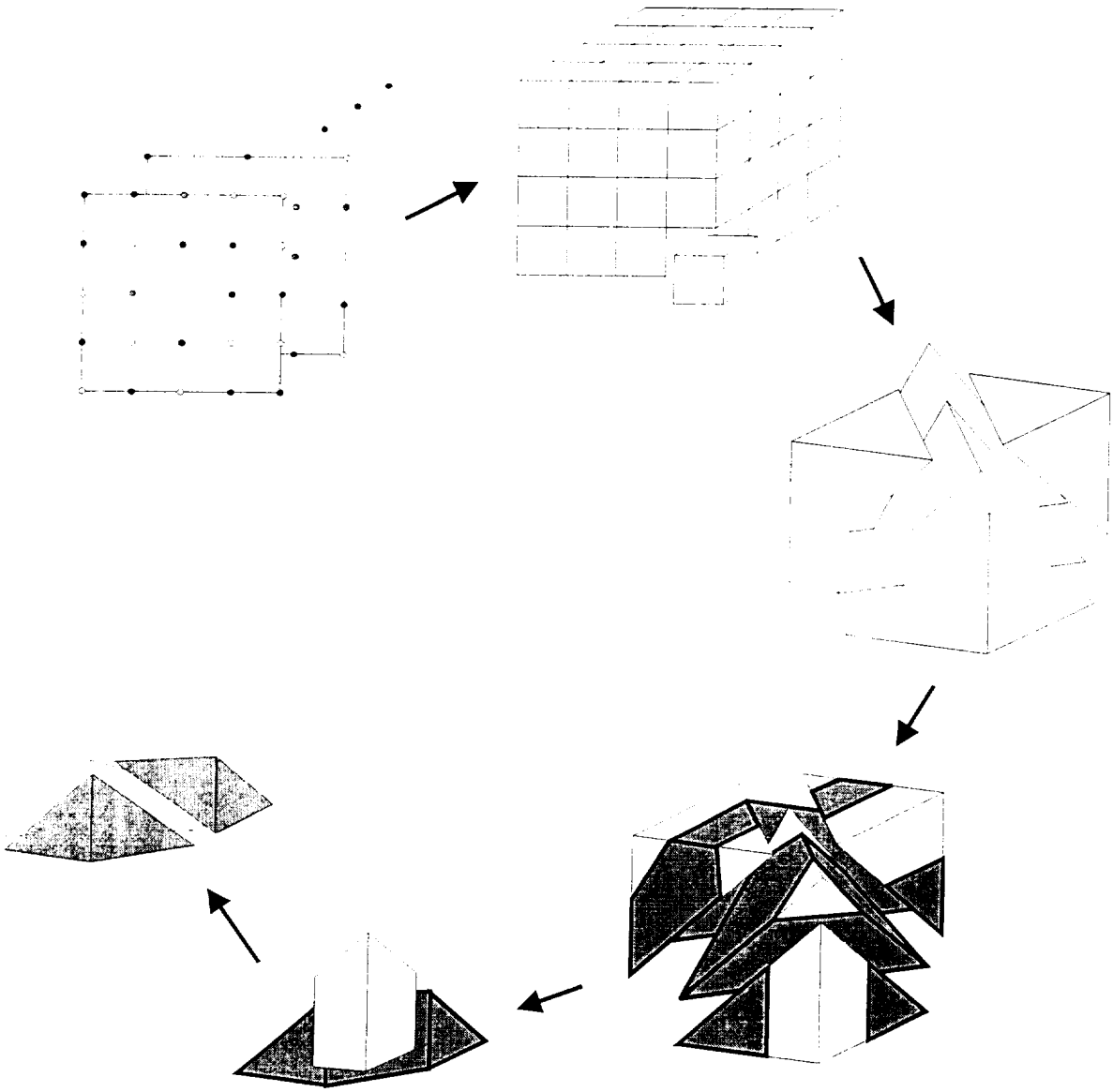


Figure 2. Steps of the Interval Volume Tetrahedrization Algorithm

2. An Algorithm for Tetrahedrizing the Interval Volume

The section is devoted to the details of the second step of our algorithm. At this point, we assume that we have a tetrahedrization of the domain of interest and the values of the data function, F, are known at the vertices of all the tetrahedra. We extend F to the interior of each tetrahedron by assuming that it varies linearly. With this assumption, the interval volume, $I_F(\alpha,\beta)$, is well defined and consists of a collection of polyhedron. Our algorithm computes the interval volume tetrahedization for each tetrahedron separately and so we need only describe the algorithm as it applies to a single tetrahedron with consideration to how the individual pieces will match up properly. The four vertices of each tetrahedron are classified according to the function value at this point. There are three possibilities. If the function value is less than or equal to α, then we call this a "white vertex" and mark it in our illustrations with an open white circle. If the function

value is strictly between α and β, the we call this a "gray vertex" and mark it with a gray filled circle. If the value is greater than or equal to β, then we call it a "black vertex" and mark it with a black filled circle. This classification leads to 15 distinct configurations which are all shown in Figure 3. All other cases can be rotated into one of these 15 equivalence class representers. The labels we use for these configurations is based upon a triple index indicating the number of white, gray and black vertices present. This naming convention is shown in Figure 4.

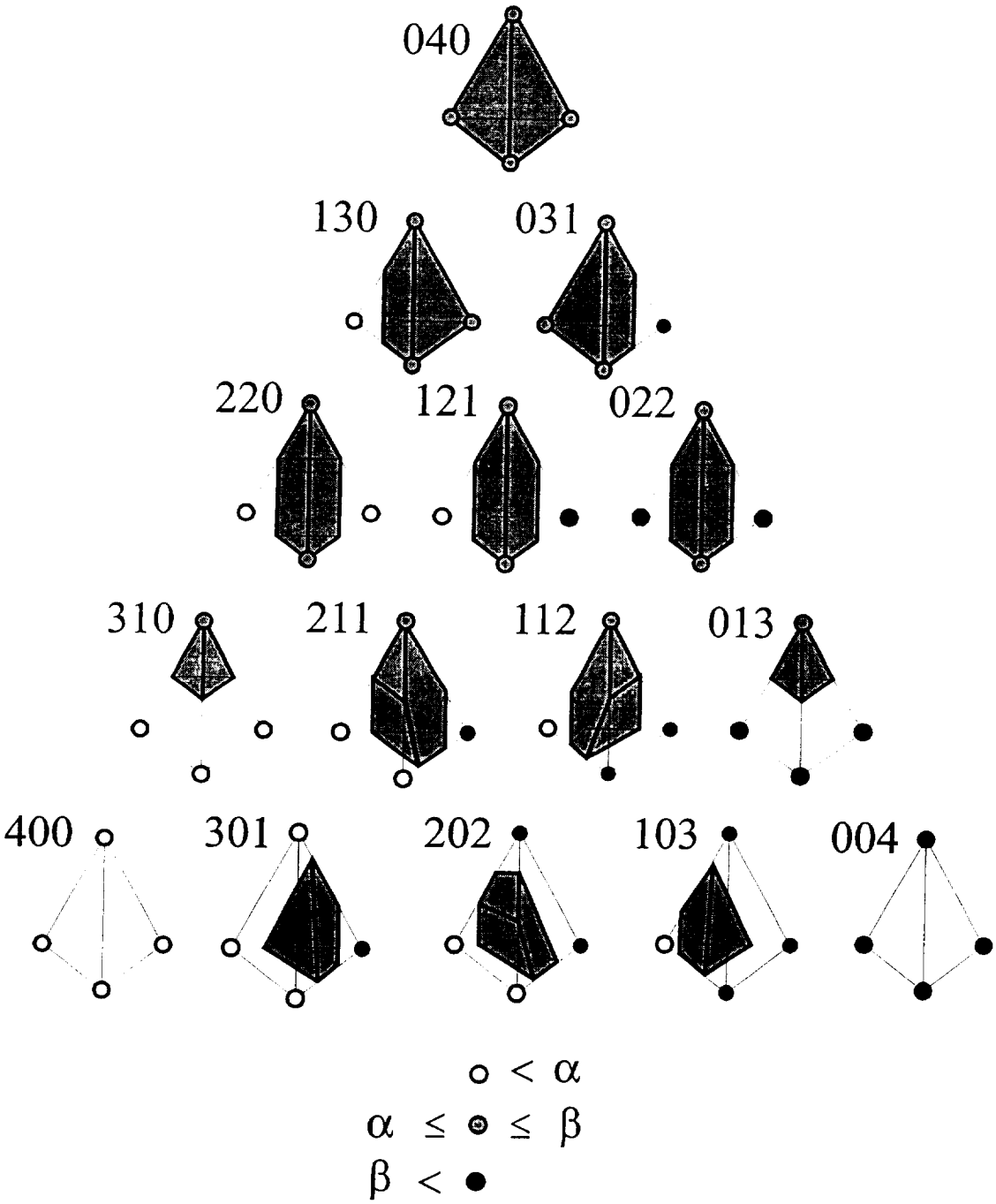


Figure 3. The fifteen distinct configurations

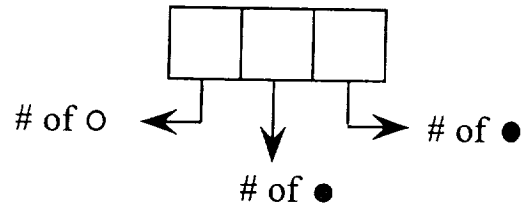


Figure 4. Method of labeling the configurations

Note that all the polyhedra representing interval volumes in Figure 3 have planar faces and are convex. The planarity of the interior quadrilaterals lying on the α- or β-surfaces is an immediate consequence of the use of linear interpolation to compute the vertices along the edges.

We next describe how to decompose each of the interval volumes of Figure 3 into tetrahedra. We first note that there are four distinct types of polyhedra, namely:

Tetrahedra: Configurations: 013, 040, 310.

Prism shaped polyhedra: Configurations: 031, 022, ,103, 130, 220, 301
Each of these polyhedra has six vertices, two opposing triangular faces and three planar quadrilateral faces.

Crystal shaped polyhedra: Configurations: 112, 121, 211
Each of these polyhedra has eight vertices, two opposing triangular faces, two planar quadrilateral faces and two planar pentagonal faces.

Cube shaped polyhedra: Configuration: 202
The polyhedra of this configuration has eight vertices and six planar quadrilateral faces.

We now describe how each of these distinct types of polyhedra can be tetrahedrized so as to lead to a global tetrahedrization of the interval volume. Since they are all convex polyhedra, there is no problem as to whether or not they can be individually tetrahedrized. (See Nielson [5].) The problem becomes interesting in trying to maintain a global tetrahedrization. Unless special provisions are made, it is possible that the tetrahedrization of a portion of the interval volume in one tetrahedra could not join with the tetrahedrization of another portion of the interval volume in a separate tetraheda in manner that would lead to an overall tetrahedrization of the interval volume. This could happen, for example, if these two portions shared a common quadrilateral face and in one instance one diagonal is chosen and in another instance the other diagonal is chosen. See Figure 5. This eliminates the possibility of having a well-defined piecewise linear function defined over this type of decomposition. In order to circumvent this problem and have a consistent choice of edges on any of the planar faces, we use the "index connection rule." The invocation of this rule assumes that we have a total ordering on the vertices so that for any two vertices one is "<" or "smaller" than the other. There are, of course, many ways to

accomplish this linear ordering of the vertices. The details of how we do it for the examples presented here is given in the next section.

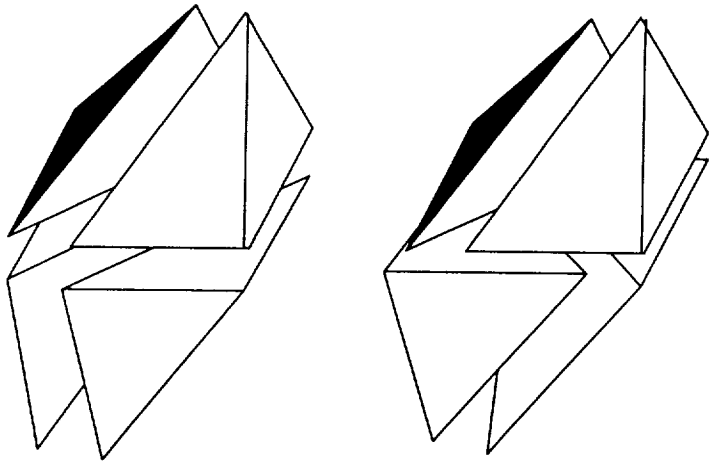


Figure 5. The "3D cracking problem" on the right.

Index Connection Rule:

Quadrilateral: The edge consisting of the unique smallest vertex joined to its opposing vertex will be contained in the tetrahedrization.

Pentagon: Each pentagonal face has a unique vertex which is gray; that is, the function value at this vertex is between α and β. Also, it is always the case on pentagonal faces that there are two adjoining α-vertices and two adjoining β-vertices. The gray vertex and its non neighbor α-vertex will form an edge as well as the gray vertex and its non neighbor β-vertex. See Figure 7.

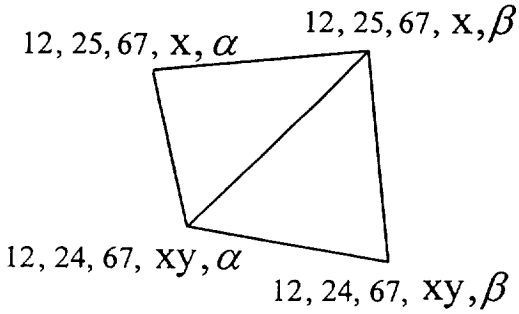


Figure 6. Index connection rule applied to quadrilateral.

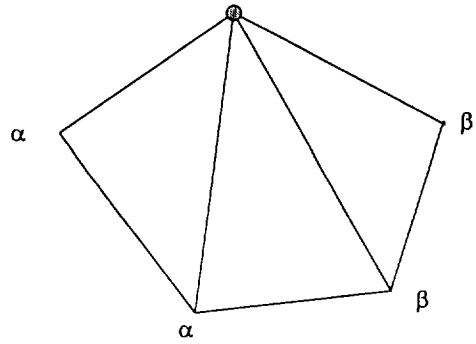


Figure 7. Index connection rule applied to pentagonal face

We now set out to prove how each of the different types of polyhedra (prism, crystal and cube) can be tetrahedrized and in a manner which is consistent with the edges required by the "index connection rule."

Prism shaped polyhedra: Configurations 031, 022, 103, 130, 220, 301. On any prism shaped polyhedron, there are eight different possible edge connections for the three quadrilateral faces. Six of them can be "realized" by a tetrahedrization of the polyhedron while two can not. See Figure 8. The edge connections shown in the upper left and lower right are the ones that can not be realized, but these particular vertex connections would never be specified by the "index connection rule" or otherwise we would have by transitivity a contradiction to the linear ordering of the vertices.

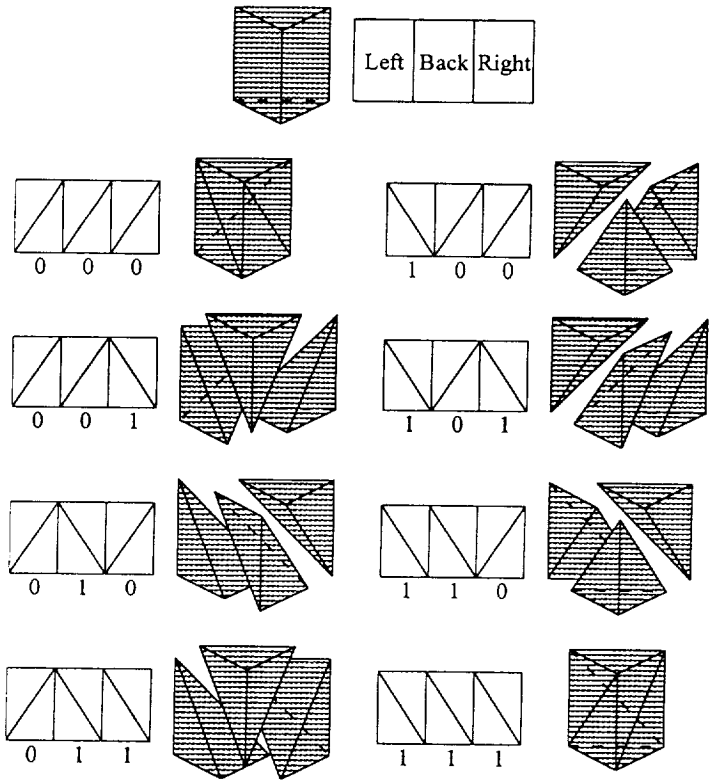


Figure 8. The eight possible edge configurations for the quadrilateral faces of the prism shaped polyhedron.

Crystal Shaped Polyhedra: Configurations: 121, 112, 211. For this type of polyhedra, we can first decompose it into two pyramids and one tetrahedron and this decomposition leads to edges on the boundary of the polyhedron which are consisitent with the edges specified by the "index connection rule" applied to pentagonal faces. This is shown for the three configurations, which lead to this case, in Figure 9. The "index connection rule" is now applied to the remaining quadrilateral faces of the pyramids and these are split into two tetrahedra each leading to a total of five tetrahedra decomposing a crystal shaped polyhedron.

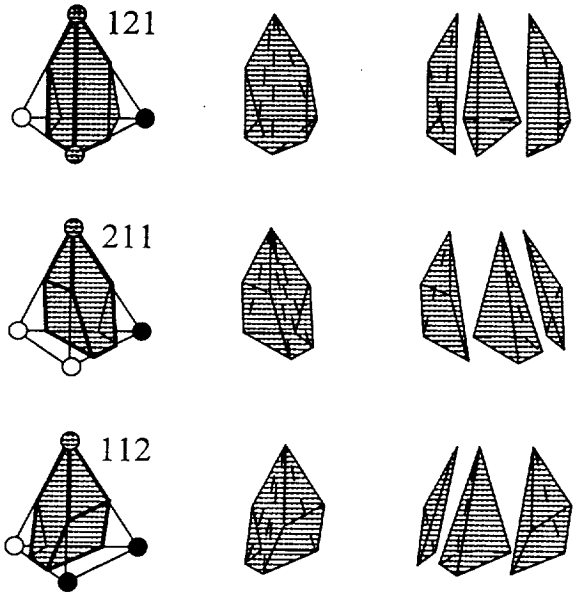


Figure 9. Decomposition of Crystal shaped polyhedra into a tetrahedron and two pyramids.

Cube Shaped Polyhedra: As with the other cases, we must show that the polyhedron can be tetrahedrized such that the edges imposed by the "index connection rule" are realized. Not all possible edge configuratons on the faces of this hexahedron can be realized by a tetrahedrization. See Nielson [5] for examples that are not realizable. But the edges imposed by the "index connection rule" are all fully realizable as we now will show. First, let us identify the smallest vertex of this polyhedron and also the second smallest. In the figures, the smallest is marked with a "**1**" and the next smallest is marked with a "**2**". There are three cases to consider. The first is when these two special vertices are located diagonally opposite from each other. Since everyone of the eight faces contains exactly one of these two special vertices, the "index connection rule" applies and the edges on all eight faces are completely specified. Fortunately, this particular configuration of edges is fully realizable by a tetrahedrization of the cubed shaped polyhedra. The tetrahedrization consistent with these edges on the faces is shown in Figure 10.

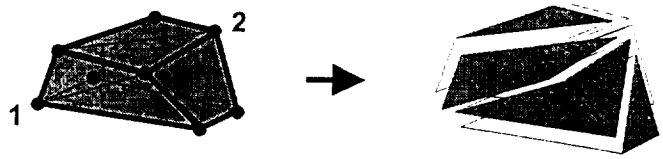


Figure 10. The first type of case for the cube shaped polyhedron.

The next case is when the smallest vertex and the next smallest vertex are diagonally located on the same face. All faces except the face opposite the one containing these two special vertices has at least one of these special vertices and so the edges are determined for these faces. This leaves ony two cases to consider. These two cases are illustrated in Figure 11. In the first case, shown on the left, all opposing faces have diagonal edges which are switched. This particular edge configuration can be realized with a

tetrahedrization consisting of five tetrahedra as shown. The other case shown on the right is realized with the tetrahedrization shown consisting of six tetrahedra.

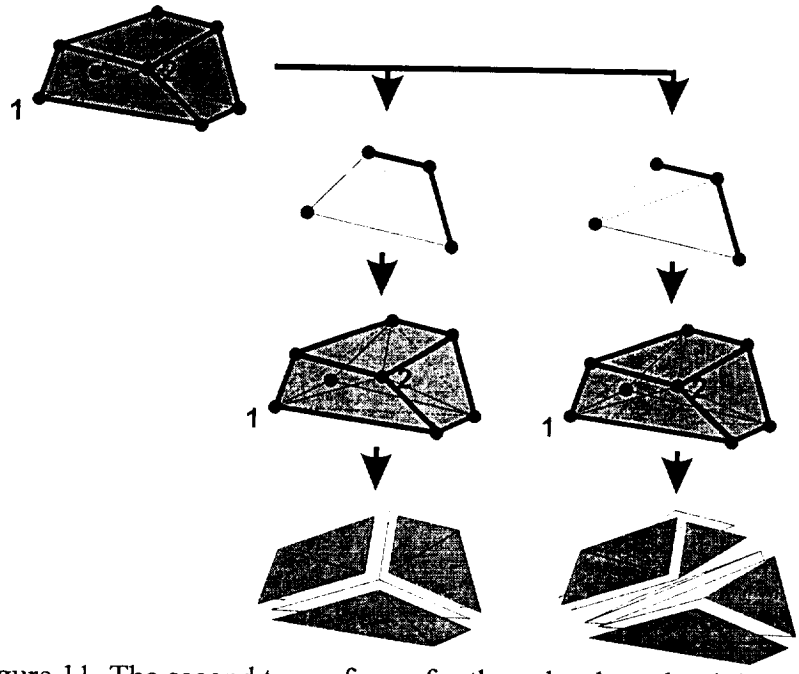


Figure 11. The second type of case for the cube shaped polyhedron

The third case is when the smallest vertex and the next smallest are on a common edge. In this case there are four subcases as the "index connection rule" leaves the edges on two adjoining faces undetermined. These four subcases and the realizing tetrahedrizations are shown in Figure 12. In all subcases, we have a tetrahedrization consisting of six tetrahedra.

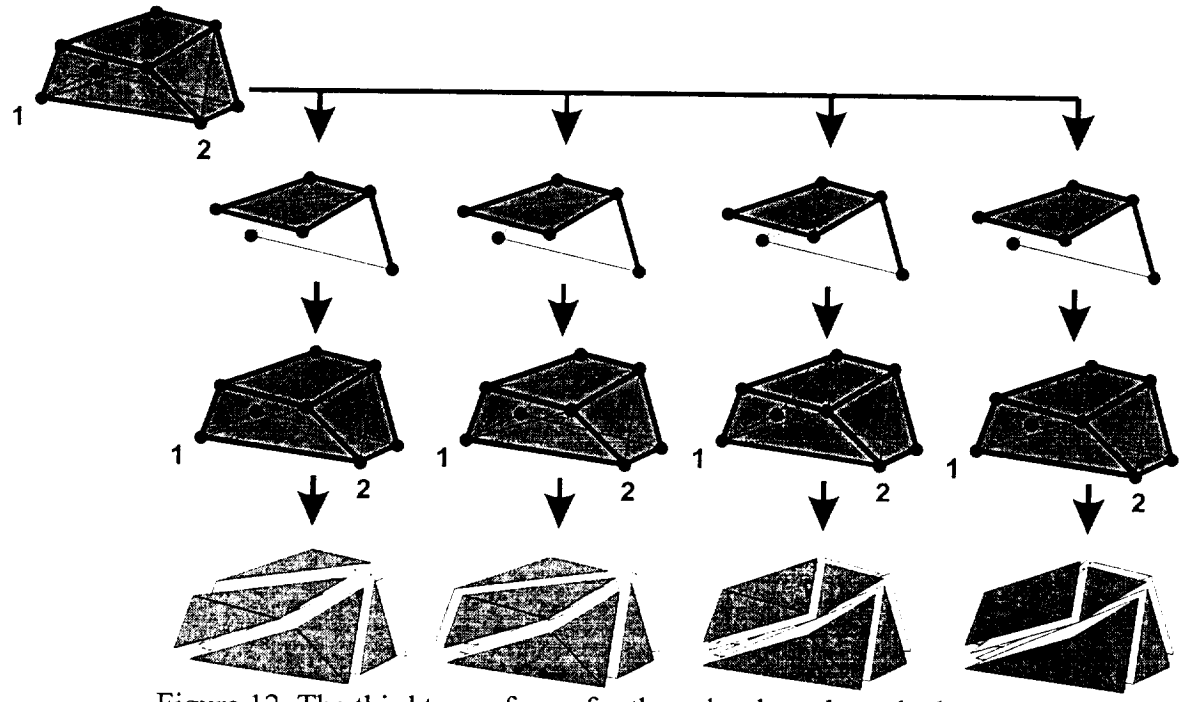


Figure 12. The third type of case for the cube shaped tetrahedron.

3. Implementation Considerations and Examples

Our implementation produces a tetrahedral grid rather than simply a list of the four vertices of each tetrahedron. The efficiency and benefits of a tetrahedral grid over this simpler data structure are evident for most applications. For compactness in the following discussion, rather than the cumbersome "tetrahedral grid representing a tetrahedrization", we often use the simpler "tetrahedrization." Our definition of a tetrahedrization starts with the collection of vertices $P_i = (x_i, y_i, z_i), \quad i = 1, \ldots, N$ which we assume are not collectively coplanar. We denote this collection of points by P. A tetrahedrization consists of a list of 4-tuples which we denote by I_t. Each 4-tuple, $ijk\ell \in I_t$ denotes a single tetrahedron with the four vertices P_i, P_j, P_k, P_ℓ. The following conditions must hold:

i) No tetrahedron $T_{ijk\ell}, ijk\ell \in I_t$ is degenerate. That is, if $ijk\ell \in I_t$ then $P_i, P_j P_k$ and P_ℓ are not coplanar.

ii) The interior of any two triangles do not intersect. That is if $ijk\ell \in I_t$ and $\alpha\beta\gamma\delta \in I_t$ then $\text{Int}(T_{ijk\ell}) \cap \text{Int}(T_{\alpha\beta\gamma\delta}) = \phi$.

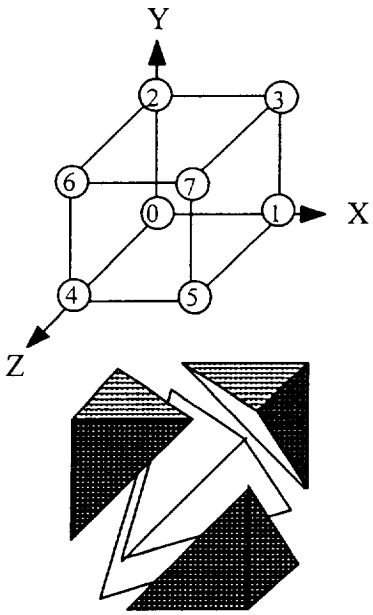
iii) The boundary of two tetrahedra can only intersect at a common triangular face.

iv) The domain is the union of all tetraheda, $D = \bigcup_{ijk\ell \in I_t} T_{ijk\ell}$.

We should point out that condition iii) must hold in the strictest sense and so tetrahedra joining as shown in right side of Figure 5 are not allowed. The reason for this

condition (and all the others) is that we eventually wish to be able to define C^0 functions in a piecewise linear manner over the domain consisting of the union of all tetrahedra.

The tetrahedral grid data structure for representing tetrahedrizations is illustrated by the example of Figure 13 where a tetrahedrization of the cube into 5 tetrahedra is shown.



Tetrahedra				Neighbors			
p_i	p_j	p_k	p_l	N_{jkl}	N_{ikl}	N_{ijl}	N_{ijk}
7	4	5	1	ϕ	ϕ	5	ϕ
7	1	3	2	ϕ	ϕ	5	ϕ
2	4	0	1	ϕ	ϕ	5	ϕ
7	2	6	4	ϕ	ϕ	5	ϕ
2	4	7	1	1	2	3	4

Figure 13. An example which defines the tetrahedral grid data structure.

As we noted earlier, the use of the "index connection rule" requires that we have a linear ordering of the vertices. The ordering we use is based upon the lexicographic ordering of a unique naming rule. The name of a vertex is specified by

$$\text{vertex\_name} = i, j, k, X|Y|Z|XY|XZ|YZ|YX|ZX|ZY|0, \alpha|\beta \qquad (1)$$

This is further illustrated in Figure 14. As the algorithm processes each tetrahedron (five per voxel) this naming scheme is used to record the tetrahedra of the interval volume for each tetrahedron. This makes it easy to invoke the "index connection rule" locally. Later we post process this tetrahedral grid into the more conventional format illustrated in Figure 13 by sequentially replacing the name (pointer) as describe in equation (1) with a simple integer pointer (or one-dimensional array index). It is also interesting to note that this particular ordering eliminates the possibility of the first and second types of cases for the cube shaped polyhedra and so only the third type of case (Figure 12) must be considered.

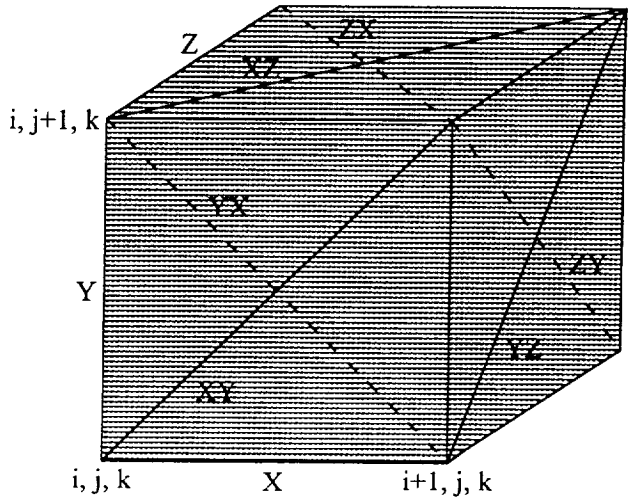


Figure 14. Vertex naming scheme.

Example 1.

This first example utilizes CAT data on a $64 \times 64 \times 68$ grid. The lower value for the interval volume is $\alpha = 50.0$ and the upper value is $\beta = 150$. There are 362,181 tetrahedra and 115,956 vertices. In Figure 15 we show the interval volume. The α-surface is colored white and the β-surface is colored red. We have cut a section of the data away so as to show the interior of the volume. Triangular faces of the interval volume which do not lie on either the α-surface or the β-surface are colored blue.


Figure 15. Display of interval volume for Example 1.

Example 2.

This next example is based upon some MRI data. The grid size is $64 \times 64 \times 58$ and we have chosen the α and β values in order to try and get a volume representation of the "skin". These are taken to be 15.0 and 40.0 respectively. These values lead to 454,916 tetrahedra and 147,711 vertices. A rendering of the interval for this example is shown in Figure 16. The left image is the interval volume and we see nothing more than the α-

surface which has been colored white. In the right image we show a cut-away with the triangular faces not lying on a either the α-surface or the β-surface colored in blue.

Figure 16. Interval volume "skin" of MRI data

Example 3.

The next example is based upon the data function

$$F(x,y,z) = 12y + 36(x-z)^2 - 5$$

and the values:

$\alpha = -0.543, \beta = 0.0$,

Domain: $\{(x,y,z)\colon \ 0 \le x \le 1, \quad 0 \le y \le 1, \quad 0 \le z \le 1\}$

Sample Grid: $\{(\frac{i}{13},\frac{j}{13},\frac{k}{13}),\ i = 0,\ldots,14;\ j = 0,\ldots,14;\ k = 0,\ldots,14\}$.

The final tetrahedrization has 4,034 tetrahedra and 1,480 vertices. A picture of the interval volume is shown in Figure 17.

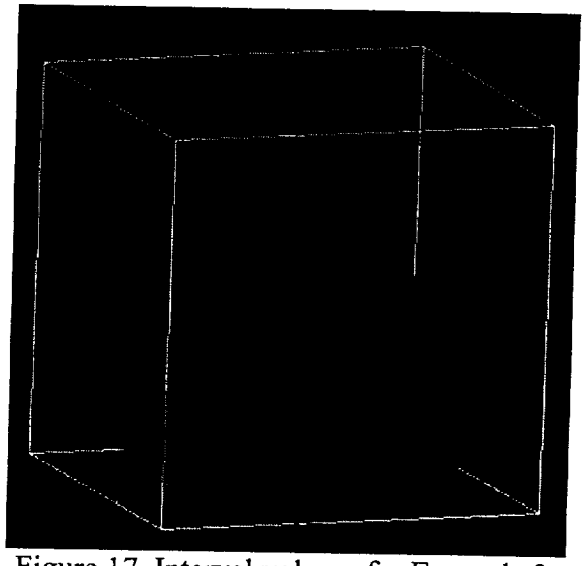


Figure 17. Interval volume for Example 3.

Example 4.

This example utilizes the data function

$$F(x,y,z) = 4(3y-1)^2 + 18(x-z)^2 - 18(x+z-1)^2 + 1$$

and the values:

$\alpha = -1.11, \beta = 1.11$

Domain: $\{(x,y,z): \quad 0 \le x \le 1, \quad 0 \le y \le 1, \quad 0 \le z \le 1\}$

Sample Grid: $\{(\frac{i}{13}, \frac{j}{13}, \frac{k}{13}), i = 0,\ldots,14; j = 0,\ldots,14; k = 0,\ldots,14\}$.

The interval volume consists of 5,782 tetrahedra and 2,092 vertices. A rendering of the interval volume is shown in Figure 18.

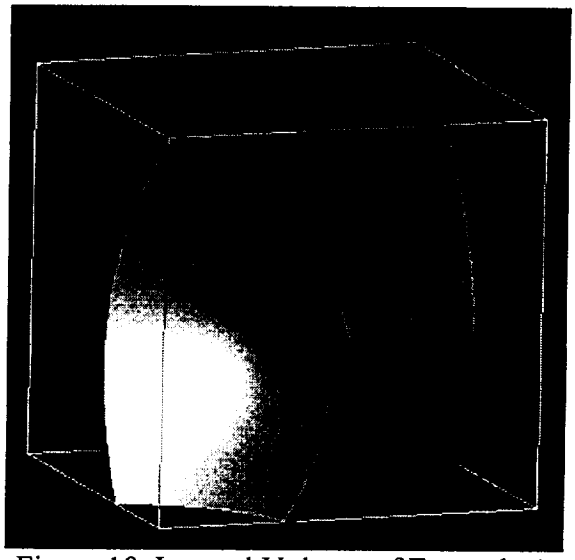


Figure 18. Interval Volume of Example 4.

Example 5.

This example is based upon the spherical data function

$$F(x,y,z) = (x-0.5)^2 + (x-0.5)^2 + (x-0.5)^2$$

and the values:

$$\alpha = 0.35, \beta = 0.37$$

Domain: $\{(x,y,z): \quad 0 \le x \le 1, \quad 0 \le y \le 1, \quad 0 \le z \le 1\}$

Sample Grid: $\{(\frac{i}{13}, \frac{j}{13}, \frac{k}{13}), i = 0,\ldots,14; j = 0,\ldots,14; k = 0,\ldots,14\}$.

The tetrahedrization has 8,500 tetrahedra and 3,224 vertices. A rendering of the interval volume is shown in Figure 19.

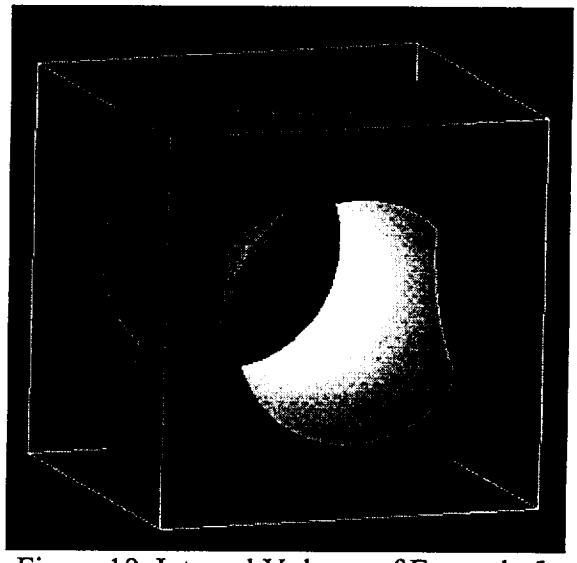


Figure 19. Interval Volume of Example 5.

Example 6.

The data function for this example is

$$F(x,y,z) = \alpha(x-0.5)^2 + \alpha(y-0.5)^2 + (\alpha-1)(z-0.5)^2, \qquad \alpha = \frac{\pi^2}{16}$$

and the values of the other parameters are:

$$\alpha = -0.02,\ \beta = 0.02$$

Domain: $\{(x,y,z):\ 0 \le x \le 1,\ \ 0 \le y \le 1,\ \ 0 \le z \le 1\}$

Sample Grid: $\{(\frac{i}{13},\frac{j}{13},\frac{k}{13}),\ i = 0,\ldots,14;\ j = 0,\ldots,14;\ k = 0,\ldots,14\}$.

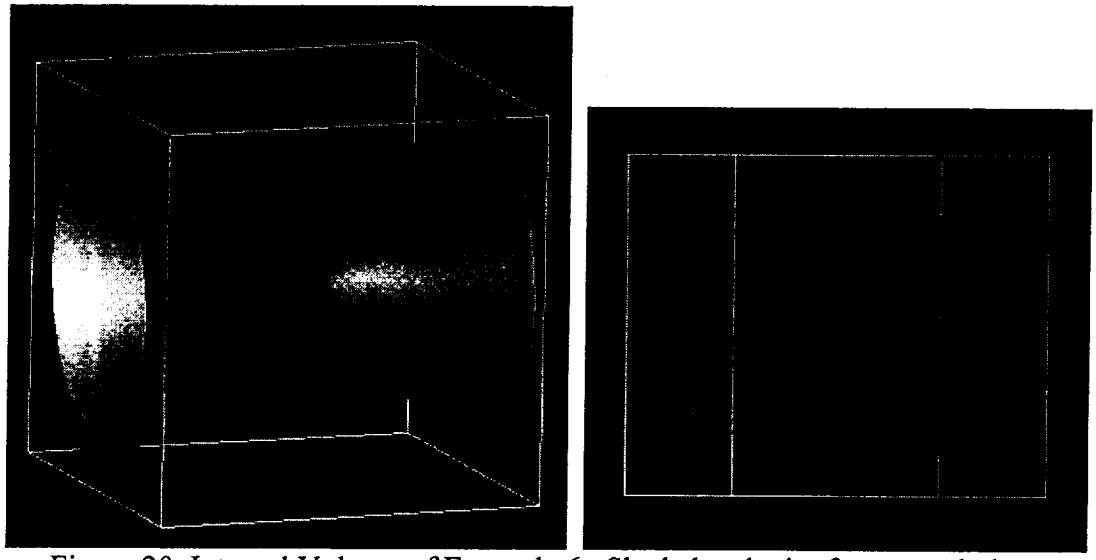


Figure 20. Interval Volume of Example 6. Shaded and wire frame renderings

In this example, we have also included a wire frame renderings so that the tetrahedrization is apparent.

It is interesting to note the frequency of the various configurations. In Figure 21, we show the statistics for the frequency of occurrence of these configurations for the six examples we have discussed here.

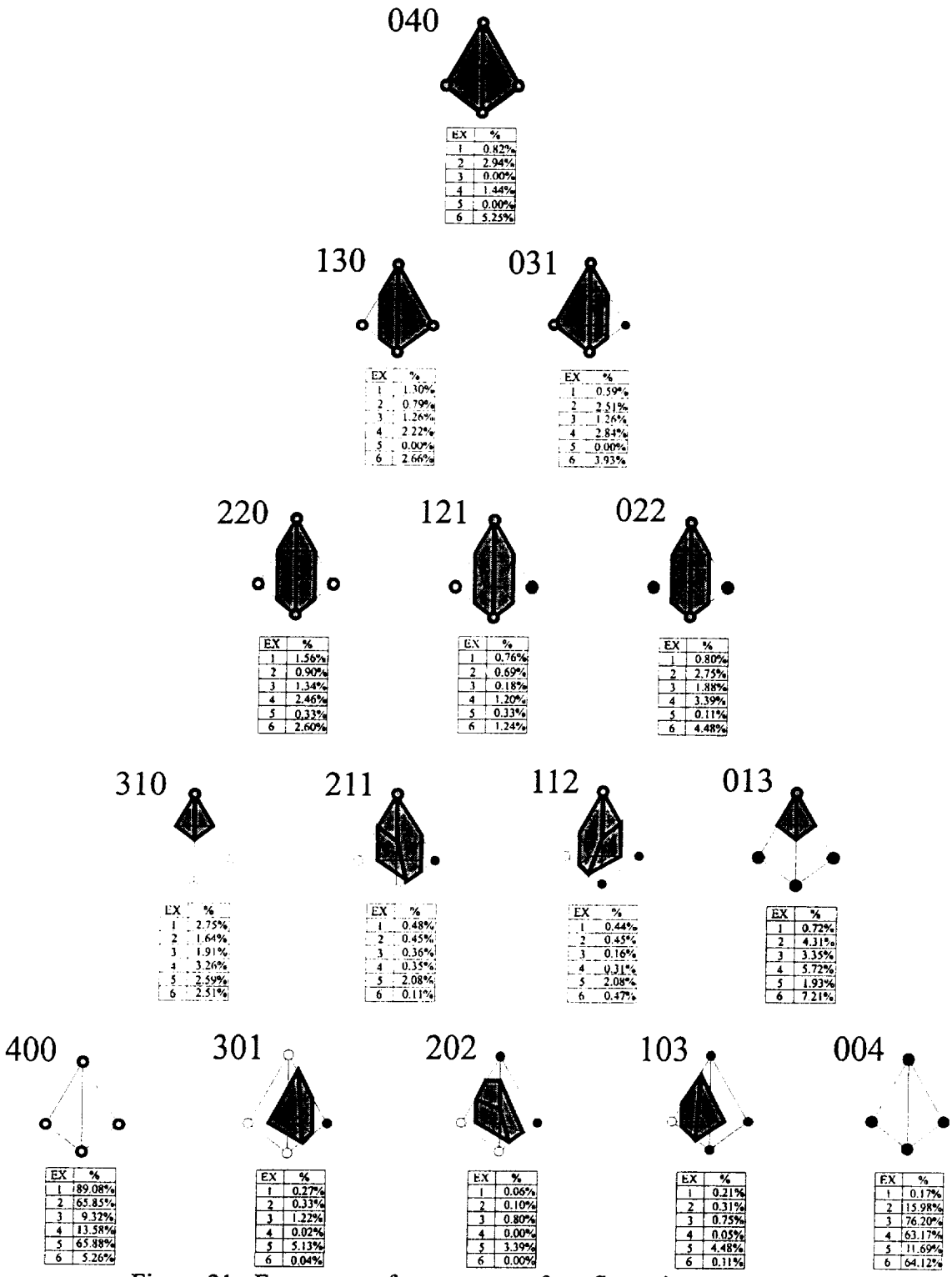


Figure 21. Frequency of occurrence of configurations.

4. Remarks

1. While our primary attention for application data sets has been rectilinear grids, for the most part, our algorithm works for any tetrahedrization of a domain. This includes tetrahedrized 3D curvilinear grids and scattered volumetric data sets.

2. Some readers may wonder why we did not use the Delaunay tetrahedrization of the convex polyhedra of Step 2 since a Delaunay tetrahedrization would lead to a two-dimensional Delaunay choice on the common faces and therefore gaurantee a matching across common faces. The problem here is to properly and efficiently deal with the neutral and near neutral cases which are far from uncommon in the applications that we have discussed here.

Acknowledgments

We wish to acknowledge the support of the National Aeronautical and Space Administration under NASA-Ames Grant, NAG 2-990 and the support of the Office of Naval Research under Grant, ONR N00014-97-1-0243.

References

1. H. Edelsbrunner and E. Mucke, Three dimensional Alpha Shapes, ACM Transactions on Graphics, Vol. 13, pp. 43-72, 1994.

2. I. Fujishiro, Y. Maeda and H. Sato, Interval volume: A solid fitting technique for volumetric data display and analysis, in Proceedings of Visualization '95, IEEE Computer Society Press, pp. 151-158, 1995.

3. Baining Guo, Interval set: A volume rendering technique generalizing isosurface extraction, in: Proceedings of Visualization '95, IEEE Computer Society Press, pp. 3-10, 1995.

4. W. E. Lorensen and H. E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, Computer Graphics, Vol. 21, No. 4, pp. 163-169, 1987.

5. G. M. Nielson, Tools for triangulations and tetrahedrizations, in: Scientific Visualization: Overviews, Methodologies, and Techniques, G. M. Nielson, H. Mueller, and H. Hagen, eds. , IEEE Computer Society Press, 1997.

6. G. M. Nielson and B. Hamann, The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes, in Proceedings of Visualization '91, pp. 83-90, 1991.

4

Haar Wavelets over Triangular Domains with applications to Multiresolution Models for Flow over a Sphere

by

Gregory M. Nielson
Il-Hong Jung
Junwon Sung

Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406
nielson|jsung|ijung@asu.edu

Abstract

Some new piecewise constant wavelets defined over nested triangulated domains are presented and applied to the problem of multiresolution analysis of flow over a spherical domain. These new nearly orthogonal wavelets have advantages over the weaker biorthogonal wavelets. In the planar case of uniform areas, the wavelets presented here converge to one of two fully orthogonal Haar wavelets which are proven to be the only wavelets of this type.

1 Introduction

In this paper we present some new Haar (piecewise constant) wavelets defined over nested triangular domains. Our target application is the multiresolution approximation and analysis of fluid flow over a spherical domain and we will show the application of these new wavelets in this context in Section 3. Even though our primary application is a spherical domain and for simplicity, our figures will be for the case of planar triangles, it should be kept in mind that the development and application is much more general. The techniques developed here apply to any triangulated manifold or surface.

The concept of the orthogonality of the wavelet basis functions is important. This relates to the ease of computation of best approximations. Fully orthogonal (or simply orthogonal) wavelets are to be preferred over the weaker biorthogonal in this regard. There do not exist orthogonal wavelets over spherical nested triangular domains (only biorthogonal) and this paper does not change that. However, the new wavelets presented here are "nearly orthogonal" in the sense that when the domain becomes planar (as spherical grids approach) the wavelets become fully orthogonal. Exactly two, planar, affine invariant, orthogonal wavelets result as limits of our nearly orthogonal wavelets and we eventually prove that these are the only planar wavelets of this type.

Often these days, papers on wavelets and multiresolution approximation and analysis will give some general basic motivation and overview. We will dispense with that here and refer the reader to anyone of the many excellent introductions currently available. See [1] for example. Rather, we will focus our introductory material on the case of Haar wavelets over nested triangular grids. First, some comments about the types of domains we are covering. We start with an initial triangulation of a domain of interest. This initial or base triangulation may be very simple. In the case of a planar application it may consist of a single triangle. In the case of a sphere it may consist of the eight spherical triangles with vertices on an octahedron or four spherical triangles with vertices on a tetrahedron. Each of the triangles of the base triangulation is subdivided into four subtriangles in a manner shown in Figure 1 by adding a vertex along each edge. Often this vertex will be the midpoint (geodesic bisector), but this need not be the case. A generic triangle of the base triangulation is denoted by T and we concentrate our attention from this level forward. All subtriangles are denoted by T_n, where n is an N-tuple made from the symbols o, i, j or k. The triangle labeling scheme that we use is illustrated in Figure 1. This labeling scheme allows us to dispense with the normal data structure used to represent a triangular as the name itself allows for the determination of all neighbors. The parent of T_{no}, T_{ni}, T_{nj} and T_{nk} is T_n. At level N there are 4^N subtriangles

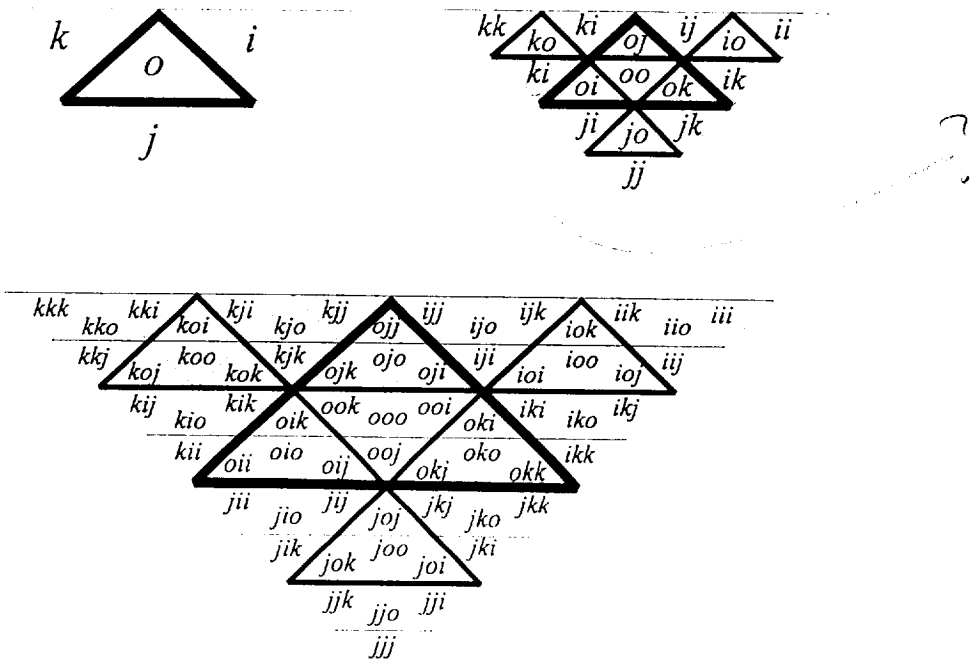


Figure 1. The Triangle Labeling Scheme for Subtriangles

We start with a function which is defined in a piecewise manner over a certain level of the nested triangular domains and we wish to find an approximation defined in a piecewise manner over the coarser, parent grid one level up. We usually think of applications starting with a rather fine resolution so that simple, piecewise constant approximation is sufficient. We want to compute this lower resolution approximation in an efficient manner and in a manner that allows the reconstruction of the original piecewise defined function. This is accomplished through a change of basis which we now illustrate by example. We can represent an arbitrary piecewise constant function over the triangular subdivision at level 2 (upper right of Figure 1) by

$$\begin{aligned} F(p) = {} & \lambda_{kk}\phi_{kk}(p) + \lambda_{ko}\phi_{ko}(p) + \lambda_{ki}\phi_{ki}(p) + \lambda_{oj}\phi_{ok}(p) + \lambda_{ij}\phi_{ij}(p) + \lambda_{io}\phi_{io}(p) + \lambda_{ii}\phi_{ii}(p) \\ & + \lambda_{ki}\phi_{ki}(p) + \lambda_{oi}\phi_{oi}(p) + \lambda_{oo}\phi_{oo}(p) + \lambda_{ok}\phi_{oj}(p) + \lambda_{ik}\phi_{ik}(p) \\ & + \lambda_{ji}\phi_{ji}(p) + \lambda_{jo}\phi_{jo}(p) + \lambda_{jk}\phi_{jk}(p) \\ & + \lambda_{jj}\phi_{jj}(p). \end{aligned} \tag{1}$$

We have arranged the terms of $F(p)$ so as to correspond with the layout of Figure 1. The value λ_{ab} represents the value of $F(p)$ over T_{ab}. The first step of the so called wavelet decomposition process yields

$$
\begin{aligned}
F(p) = \gamma_{kk}\psi_{jj}(p) + \lambda_k\phi_k(p) \; &+ \gamma_{kj}\psi_{kj}(p) + \gamma_{oj}\psi_{oj}(p) + \gamma_{ij}\psi_{ij}(p) \; + \lambda_i\phi_i(p) + \gamma_{ii}\psi_{ii}(p) \\
&+ \gamma_{ki}\psi_{ki}(p) + \gamma_{oi}\psi_{oi}(p) + \lambda_o\phi_o(p) \; + \gamma_{ok}\psi_{ok}(p) + \gamma_{ik}\psi_{ik}(p) \\
&+ \gamma_{ji}\psi_{ji}(p) \; + \lambda_j\phi_j(p) \; + \gamma_{jk}\psi_{jk}(p) \\
&+ \gamma_{jj}\psi_{jj}(p)
\end{aligned}
\tag{2}
$$

where the coarser approximation is represented by the terms $\lambda_k\phi_k(p) + \lambda_o\phi_o(p) + \lambda_i\phi_i(p) + \lambda_j\phi_j(p)$ and the so called detail or wavelet is represented by the remaining 12, ψ basis functions. We should be quick to point out that $F(p)$ has not changed from equation (1) to equation (2), only a change of basis has taken place. Still F has the property that $F(p) = \lambda_{ab}$ for $p \in T_{ab}$. We can carry this basic idea one step further and replace the coarser approximation, $\lambda_k\phi_k(p) + \lambda_o\phi_o(p) + \lambda_i\phi_i(p) + \lambda_j\phi_j(p)$, which is piecewise defined over the parent triangles with an even yet coarser approximation which is, in this second and last step, piecewise over the single triangle parent of T_i, T_j, T_k, T_o. This leads to the representation

$$
\begin{aligned}
F(p) = \gamma_{kk}\psi_{kk}(p) + \gamma_k\psi_k(p) \; &+ \gamma_{kj}\psi_{kj}(p) + \gamma_{oi}\psi_{oi}(p) + \gamma_{ij}\psi_{ij}(p) \; + \gamma_i\psi_i(p) + \gamma_{ii}\psi_{ii}(p) \\
&+ \gamma_{ki}\psi_{ki}(p) \; + \gamma_{oi}\psi_{oi}(p) + \lambda\phi(p) \quad + \gamma_{ok}\psi_{ok}(p) + \gamma_{ik}\psi_{ik}(p) \\
&+ \gamma_{ji}\psi_{ji}(p) \; + \gamma_j\psi_j(p) \; + \gamma_{jk}\psi_{jk}(p) \\
&+ \gamma_{jj}\psi_{jj}(p).
\end{aligned}
\tag{3}
$$

From equation (2) to equation (3) the 12 wavelet terms from equation (1) remain and 3 additional wavelet terms have been added along with the coarser (coarsest) approximation $\lambda\phi(p)$. A general step in this decomposition is accomplished by successively processing groups of four terms which are associated with a particular triangle. The terms

$$
\begin{aligned}
&\lambda_{nk}\phi_{nk}(p) + \lambda_{no}\phi_{no}(p) + \lambda_{ni}\phi_{ni}(p) \\
&+ \lambda_{nj}\phi_{nj}(p)
\end{aligned}
$$

are replaced by the terms

$$
\begin{aligned}
&\gamma_{nk}\psi_{nk}(p) + \lambda_n\phi_n(p) + \gamma_{ni}\psi_{ni}(p) \\
&+ \gamma_{nj}\psi_{nj}(p)
\end{aligned}
$$

using the relationship

$$\begin{pmatrix} \phi_n(P) \\ \psi_{ni}(P) \\ \psi_{nj}(P) \\ \psi_{nk}(P) \end{pmatrix} = \begin{pmatrix} h_{no} & h_{ni} & h_{nj} & h_{nk} \\ g_{nio} & g_{nii} & g_{nij} & g_{nik} \\ g_{njo} & g_{nji} & g_{njj} & g_{njk} \\ g_{nko} & g_{nki} & g_{nkj} & g_{nkk} \end{pmatrix} \begin{pmatrix} \phi_{no}(p) \\ \phi_{ni}(p) \\ \phi_{nj}(p) \\ \phi_{nk}(p) \end{pmatrix}. \tag{4}$$

The above relationship (4) is called the *refinement relationship* within the context of wavelets and embodies the information for a change of basis functions. See Figure 2.

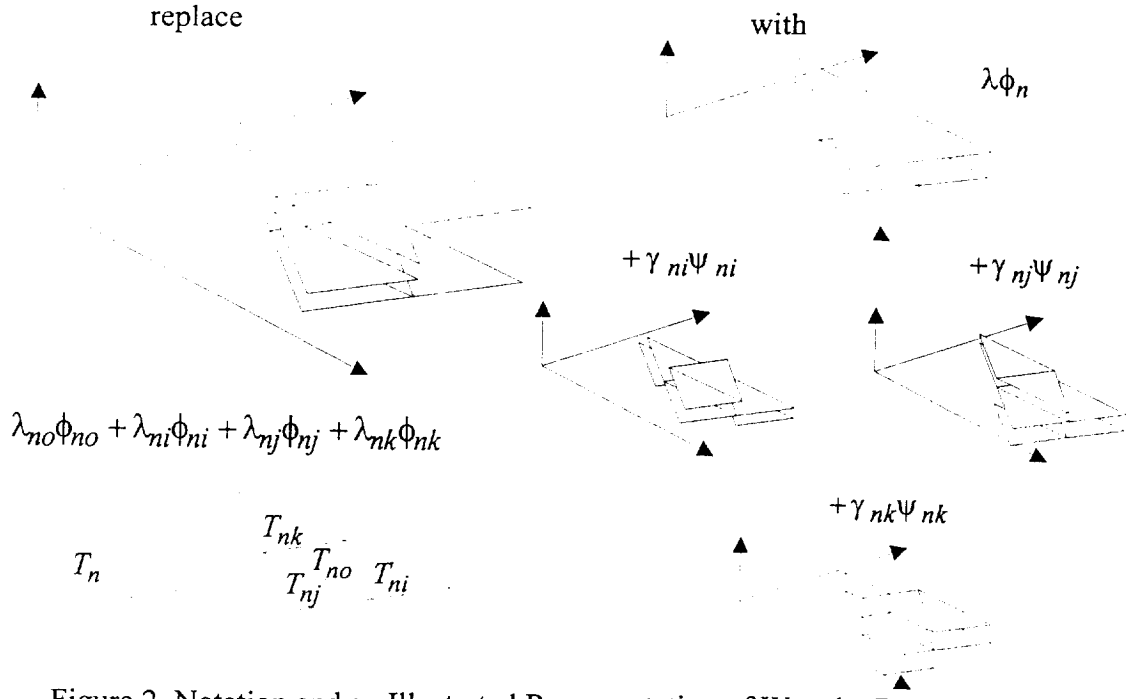


Figure 2. Notation and an Illustrated Representation of Wavelet Decomposition

Actually in the computation of the coefficients of the wavelet decomposition leading from (1) to (3) we need the inverse of the refinement relationship; namely,

$$\begin{pmatrix} \phi_{no}(p) \\ \phi_{ni}(p) \\ \phi_{nj}(p) \\ \phi_{nk}(p) \end{pmatrix} = \begin{pmatrix} \tilde{h}_{no} & \tilde{g}_{noi} & \tilde{g}_{noj} & \tilde{g}_{nok} \\ \tilde{h}_{ni} & \tilde{g}_{nii} & \tilde{g}_{nij} & \tilde{g}_{nik} \\ \tilde{h}_{nj} & \tilde{g}_{nji} & \tilde{g}_{njj} & \tilde{g}_{njk} \\ \tilde{h}_{nk} & \tilde{g}_{nki} & \tilde{g}_{nkj} & \tilde{g}_{nkk} \end{pmatrix} \begin{pmatrix} \phi_n(P) \\ \psi_{ni}(P) \\ \psi_{nj}(P) \\ \psi_{nk}(P) \end{pmatrix} \tag{5}$$

because

$$\begin{pmatrix} \lambda_n & \gamma_{ni} & \gamma_{nj} & \gamma_{nk} \end{pmatrix} = \begin{pmatrix} \lambda_{no} & \lambda_{ni} & \lambda_{nj} & \lambda_{nk} \end{pmatrix} \begin{pmatrix} \tilde{h}_{no} & \tilde{g}_{noi} & \tilde{g}_{noj} & \tilde{g}_{nok} \\ \tilde{h}_{ni} & \tilde{g}_{nii} & \tilde{g}_{nij} & \tilde{g}_{nik} \\ \tilde{h}_{nj} & \tilde{g}_{nji} & \tilde{g}_{njj} & \tilde{g}_{njk} \\ \tilde{h}_{nk} & \tilde{g}_{nki} & \tilde{g}_{nkj} & \tilde{g}_{nkk} \end{pmatrix} \tag{6}$$

This inverse relationship is called the *decomposition relationship*. The representation of equation (3) is called the wavelet decomposition and from this representation, we can easily reconstruct any of the piecewise defined approximations at any level and this can be done either locally or globally. Also, from this decomposition, we can easily reconstruct the original function locally by simply adding up the terms whose support intersects the local region of interest. Other oracles can be invoked for different purposes and applications. An oracle is a means by which a certain subset of the basis functions of the wavelet decomposition are selected to form a certain beneficial approximation of the original function. Compression of the original function is a very popular application. Here a certain subset (say those with the largest coefficients) of the terms of the wavelet decomposition are maintained. If the coefficients of the discarded terms are nonzero, then there will be some loss as a result of the compression. With regard to this, the concept of orthogonality is important. Let us explain why by noting a very basic property of best approximations and orthogonal basis functions. Suppose that we wish to solve the problem of

$$\min_{\{a_1, a_2, \cdots, a_N\}} \left\| F - \sum_{i=1}^{N} a_i g_i \right\|$$

where $\|F\|^2 = (F, F)$. If the basis functions are orthonormal, i. e. $(g_i, g_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$ then the solution of (6) is easily computed as

$$G = \sum_{i=1}^{N} (F, g_i) g_i \, .$$

Furthermore this implies that whenever you have a representation of a function of the form

$$F = \sum_{i=1}^{K} \alpha_i g_i$$

where the basis functions are orthonormal, then α_i must be (F, g_i) and if we wish to compute a best approximation to F from among a subset of these basis functions, we need only keep those of interest. That is, if F is given by (7) then the solution of

$$\min_{a_i, i \in K_s} \left\| F - \sum_{i \in K_s} a_i g_i \right\|$$

is

$$G = \sum_{i \in K_s} \alpha_i g_i \ .$$

This only works if the basis functions are orthogonal.

Biorthogonal wavelets have the property that the collectively the $\psi's$ are orthogonal to the $\phi's$, but individually, the $\psi's$ are not orthogonal to each other and so by truncating a wavelet representation, we don't necessarily get the best approximation. As long as we do the approximation in "chunks" it is OK. That is, as long as we involve all the wavelet basis functions from a particular level the best approximation property holds, but if we use an oracle which does not keep this in mind, then we are not making the best use of the storage for the coefficients. For example, an approximation based upon a certain percentage of the largest coefficients of the biorthogonal wavelet decomposition will not necessarily give the best compression in the least squares sense.

There is a second concept that is really quite important when it comes to the practical application of these techniques. This has to do with how the actual data is mapped to the names and variables used to describe and program these methods. For example suppose that we start with a triangular array of trixel data $F_{ij}, i, j = 1, \cdots, 2^N, i + j \le 2^N$ and we plan to apply the wavelets described here to obtain some compression or lower resolution approximation. In order to apply these techniques, we need to make some association of the data to the subtriangle domains illustrated in Figure 1. There are six possible ways of making this association with each being an affine map of the other. But there is no particular reason why one would be preferred over the other. We certainly do not want the results of our wavelet decompositon and expansions to depend upon which choice is made. Therefore, it is important that are methods give the same results in any case and be invariant to these affine transformations or equivalently be invariant under the labelings used. Some authors have called this property "symmetry."

2. Examples of Haar Wavelets over Triangular Domains

Within the context that we have established here, a particular type of Haar wavelet is completely determined by the refinement equations of (4). We begin this section by presenting two previously discussed Haar wavelets. The first is mentioned in [3] where it is attributed to Mitrea and Girardi & Sweldens. The refinement equation for these wavelets is

$$\begin{pmatrix} \phi_n(P) \\ \psi_{ni}(P) \\ \psi_{nj}(P) \\ \psi_{nk}(P) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \dfrac{-\alpha_n}{2\alpha_{no}} & \dfrac{\alpha_n}{2\alpha_{ni}} & 0 & 0 \\ \dfrac{-\alpha_n}{2(\alpha_{no}+\alpha_{ni})} & \dfrac{-\alpha_n}{2(\alpha_{no}+\alpha_{ni})} & \dfrac{\alpha_n}{2\alpha_{nj}} & 0 \\ \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nk})} & \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nk})} & \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nk})} & \dfrac{\alpha_n}{2\alpha_{nk}} \end{pmatrix} \begin{pmatrix} \phi_{no}(p) \\ \phi_{ni}(p) \\ \phi_{nj}(p) \\ \phi_{nk}(p) \end{pmatrix}$$

where α_n represents the area of triangle T_n and $\alpha_{ni}, \alpha_{nj}, \alpha_{nk}, \alpha_{no}$ represent the areas of the subtriangles $T_{ni}, T_{nj}, T_{nk}, T_{no}$ respectively. These Haar wavelets are orthogonal in that $\psi_{ni}, \psi_{nj}, \psi_{nk}$ are all orthogonal to ϕ_n and also $\psi_{ni} \perp \psi_{nj}$, $\psi_{ni} \perp \psi_{nk}$, $\psi_{nj} \perp \psi_{nk}$. Also $\int_{T_n} |\psi_{ni}| = \int_{T_n} |\psi_{nj}| = \int_{T_n} |\psi_{nk}| = \alpha$. Unfortunately, these wavelets are not affine invariant which means that the results of any application of these wavelets would depend upon the happenstance of how the vertices were labeled.

Another type of wavelet (see [3]) is defined by the refinement equation

$$\begin{pmatrix} \phi_n(P) \\ \psi_{ni}(P) \\ \psi_{nj}(P) \\ \psi_{nk}(P) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{ni})} & \dfrac{\alpha_n}{2\alpha_{ni}} & \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{ni})} & \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{ni})} \\ \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nj})} & \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nj})} & \dfrac{\alpha_n}{2\alpha_{nj}} & \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nj})} \\ \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nk})} & \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nk})} & \dfrac{-\alpha_n}{2(\alpha_n-\alpha_{nk})} & \dfrac{\alpha_n}{2\alpha_{nk}} \end{pmatrix} \begin{pmatrix} \phi_{no}(p) \\ \phi_{ni}(p) \\ \phi_{nj}(p) \\ \phi_{nk}(p) \end{pmatrix}. \quad (7)$$

These wavelets are affine invariant, but this comes at the cost of giving up the orthogonality. These wavelets are only biorthogonal. Also, as above, $\int_{T_n} |\psi_{ni}| = \int_{T_n} |\psi_{nj}| = \int_{T_n} |\psi_{nk}| = \alpha$.

We now present our two new types of *nearly orthogonal* wavelets. The first has the refinement equation

$$\begin{pmatrix} \phi_n(p) \\ \psi_{ni}(p) \\ \psi_{nj}(p) \\ \psi_{nk}(p) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1-\dfrac{\alpha_{no}^2+\alpha_{ni}\alpha_{no}}{\Delta} & 1-\dfrac{\alpha_{ni}^2+\alpha_{ni}\alpha_{no}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{nj}-\alpha_{ni}\alpha_{nj}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{nk}-\alpha_{ni}\alpha_{nk}}{\Delta} \\ 1-\dfrac{\alpha_{no}^2+\alpha_{nj}\alpha_{no}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{ni}-\alpha_{nj}\alpha_{ni}}{\Delta} & 1-\dfrac{\alpha_{nj}^2+\alpha_{no}\alpha_{nj}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{nk}-\alpha_{nj}\alpha_{nk}}{\Delta} \\ 1-\dfrac{\alpha_{no}^2+\alpha_{nk}\alpha_{no}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{ni}-\alpha_{nk}\alpha_{ni}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{nj}-\alpha_{nk}\alpha_{nj}}{\Delta} & 1-\dfrac{\alpha_{nk}^2+\alpha_{no}\alpha_{nk}}{\Delta} \end{pmatrix} \begin{pmatrix} \phi_{no}(p) \\ \phi_{ni}(p) \\ \phi_{nj}(p) \\ \phi_{nk}(p) \end{pmatrix} \quad (8)$$

where $\Delta = \alpha_{no}^2 + \alpha_{ni}^2 + \alpha_{nj}^2 + \alpha_{nk}^2$ and $\int_{T_n} |\psi_{ni}| = \dfrac{2(\alpha_{no} + \alpha_{ni})(\alpha_{nj}^2 + \alpha_{nk}^2)}{\Delta}$

and the second has the refinement equation

$$\begin{pmatrix} \phi_n(p) \\ \psi_{ni}(p) \\ \psi_{nj}(p) \\ \psi_{nk}(p) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 - \dfrac{\alpha_{no}^2 - 3\alpha_{ni}\alpha no}{\Delta} & -3 + \dfrac{3\alpha_{ni}^2 - \alpha_{ni}\alpha_{no}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{nj} + 3\alpha_{ni}\alpha_{nj}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{nk} + 3\alpha_{ni}\alpha_{nk}}{\Delta} \\ 1 - \dfrac{\alpha_{no}^2 - 3\alpha_{nj}\alpha_{no}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{ni} + 3\alpha_{nj}\alpha_{ni}}{\Delta} & -3 + \dfrac{3\alpha_{nj}^2 - \alpha_{no}\alpha_{nj}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{nk} + 3\alpha_{nj}\alpha_{nk}}{\Delta} \\ 1 - \dfrac{\alpha_{no}^2 - 3\alpha_{nk}\alpha_{no}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{ni} + 3\alpha_{nk}\alpha_{ni}}{\Delta} & \dfrac{-\alpha_{no}\alpha_{nj} + 3\alpha_{nk}\alpha_{nj}}{\Delta} & -3 + \dfrac{3\alpha_{nk}^2 - \alpha_{nj}\alpha_{nk}}{\Delta} \end{pmatrix} \begin{pmatrix} \phi_{no}(p) \\ \phi_{ni}(p) \\ \phi_{nj}(p) \\ \phi_{nk}(p) \end{pmatrix} \tag{9}$$

where $\Delta = \alpha_{no}^2 + \alpha_{ni}^2 + \alpha_{nj}^2 + \alpha_{nk}^2$ and $\int_{T_n} |\psi_{ni}| = \dfrac{2\alpha_{ni}\left(\alpha_{no}\alpha_{ni} + 3\left(\alpha_{no}^2 + \alpha_{nj}^2 + \alpha_{nk}^2\right)\right)}{\Delta}$.

Both of these new wavelets are biorthogonal and affine invariant but not orthogonal. But these new wavelets have an advantage over other wavelets of this type. As the areas of the subtriangles become more nearly equal, these wavelets tend to fully orthogonal wavelets. This is not true of the wavelets given by equation (7) for example. If we let $\dfrac{\alpha_n}{4} = \alpha_{ni} = \alpha_{nj} = \alpha_{nk} = \alpha_{no}$ the refinement equation for these wavelets becomes

$$\begin{pmatrix} \phi_n \\ \psi_{ni} \\ \psi_{nj} \\ \psi_{nk} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ -\dfrac{2}{3} & 2 & -\dfrac{2}{3} & -\dfrac{2}{3} \\ -\dfrac{2}{3} & -\dfrac{2}{3} & 2 & -\dfrac{2}{3} \\ -\dfrac{2}{3} & -\dfrac{2}{3} & -\dfrac{2}{3} & 2 \end{pmatrix} \begin{pmatrix} \phi_{no} \\ \phi_{ni} \\ \phi_{nj} \\ \phi_{nk} \end{pmatrix} \tag{10}$$

and so we have the innerproducts: $\int_{T_n} \psi_{ni}\psi_{nj} = \int_{T_n} \psi_{ni}\psi_{nk} = \int_{T_n} \psi_{nk}\psi_{nj} = -\dfrac{4\alpha_n}{9}$. In the case of the nearly orthogonal wavelets, we have, for uniform areas, the refinement equations

$$\begin{pmatrix} \phi_n \\ \psi_{ni} \\ \psi_{nj} \\ \psi_{nk} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \dfrac{1}{2} & \dfrac{1}{2} & -\dfrac{1}{2} & -\dfrac{1}{2} \\ \dfrac{1}{2} & -\dfrac{1}{2} & \dfrac{1}{2} & -\dfrac{1}{2} \\ \dfrac{1}{2} & -\dfrac{1}{2} & -\dfrac{1}{2} & \dfrac{1}{2} \end{pmatrix} \begin{pmatrix} \phi_{no} \\ \phi_{ni} \\ \phi_{nj} \\ \phi_{nk} \end{pmatrix} \tag{11}$$

and

$$\begin{pmatrix} \phi_n \\ \psi_{ni} \\ \psi_{nj} \\ \psi_{nk} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{2} & -\frac{5}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{3}{2} & \frac{1}{2} & -\frac{5}{2} & \frac{1}{2} \\ \frac{3}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{5}{2} \end{pmatrix} \begin{pmatrix} \phi_{no} \\ \phi_{ni} \\ \phi_{nj} \\ \phi_{nk} \end{pmatrix} \tag{12}$$

and it can easily be verified that in both cases $\phi_n \perp \psi_{ni}$, $\phi_n \perp \psi_{nj}$, $\phi_n \perp \psi_{nk}$ and $\psi_{ni} \perp \psi_{nj}$, $\psi_{ni} \perp \psi_{nk}$, $\psi_{nj} \perp \psi_{nk}$. So, in both cases, as the areas of the subtriangles become uniform the wavelets are become mutually orthogonal. It is interesting that these two wavelets are the only affine invariant, fully orthogonal Haar wavelets.

Theorem: In the case where the areas of the subtriangles are all equal, there are only two types of fully orthogonal, affine invariant Haar wavelets; namely those defined by the refinement equations given in equations (11) and (12).

Proof: If we impose the affine invariance conditions then we must start with refinement and decomposition relationships of the general form

$$\begin{pmatrix} \phi_n \\ \psi_{ni} \\ \psi_{nj} \\ \psi_{nk} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ g_o & g_\ell & g_m & g_m \\ g_o & g_m & g_\ell & g_m \\ g_o & g_m & g_m & g_\ell \end{pmatrix} \begin{pmatrix} \phi_{no} \\ \phi_{ni} \\ \phi_{nj} \\ \phi_{nk} \end{pmatrix} \tag{13}$$

and

$$\begin{pmatrix} \phi_{no} \\ \phi_{ni} \\ \phi_{nj} \\ \phi_{nk} \end{pmatrix} = \begin{pmatrix} \frac{1}{4} & \tilde{g}_o & \tilde{g}_o & \tilde{g}_o \\ \frac{1}{4} & \tilde{g}_\ell & \tilde{g}_m & \tilde{g}_m \\ \frac{1}{4} & \tilde{g}_m & \tilde{g}_\ell & \tilde{g}_m \\ \frac{1}{4} & \tilde{g}_m & \tilde{g}_m & \tilde{g}_\ell \end{pmatrix} \begin{pmatrix} \phi_n \\ \psi_{ni} \\ \psi_{nj} \\ \psi_{nk} \end{pmatrix} \tag{14}$$

If we now impose the biorthogonal conditions along with the requirement that (13) be the inverse of (14), we have that

$$g_\ell = \frac{5\tilde{g}_o - \tilde{g}_\ell}{4\tilde{g}_o(3\tilde{g}_\ell + \tilde{g}_o)}$$

$$g_o = \frac{1}{4\tilde{g}_o}$$

$$g_m = \frac{-(3\tilde{g}_o + \tilde{g}_\ell)}{4\tilde{g}_o(3\tilde{g}_\ell + \tilde{g}_o)}.$$

Now if we further impose the conditions of full orthogonality, we have the requirement

$$(\tilde{g}_\ell - \tilde{g}_o)(3\tilde{g}_\ell + 5\tilde{g}_o) = 0,$$

which give rise to two solutions. The first solution ($\tilde{g}_\ell = \tilde{g}_o$) leads to the wavelets of equation (11) and the second solution ($\tilde{g}_\ell = -\frac{5}{3}\tilde{g}_o$) leads to the wavelets of equation (12) and so these are the only, orthogonal, affine invariant Haar wavelets over subdivided triangular grids.

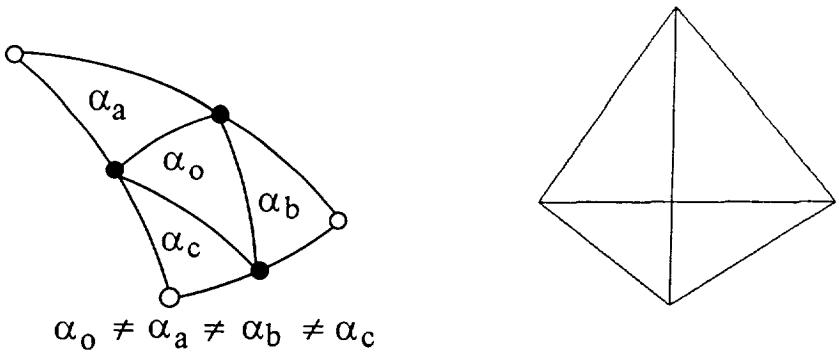
3. Application Examples

We have applied our new techniques on a variety of data sets, but some of the most interesting are for the case of a spherical domain. We present some of those results in this section. First we mention a few things about the triangular grids for a spherical domain. The construction of the grid starts with a base triangulation of the sphere. Often this is based upon the vertices of a tetrahedron (4 triangles) , octahedron (8 triangles) or icosahedron (20 triangles). Each triangle of the base triangulation is subdivided into four triangles by bisecting the geodesic edges at their midpoints. This process is continued. Figure 3 illustrates this process with the base triangulation being a spherical tetrahedron. Even though the triangles of Figure 3 are drawn with straight edges, the actual triangles used for the wavelets are spherical triangles with geodesic arcs as edges. For the sphere, it is impossible to arrange for the areas of the subtriangles to be equal while in fact the areas of these subtriangles can be quite different. For example, in the first level of the case of a tetrahedron as the base triangulation vertices, the area of the center subtriangle is four times the area of the the other three subtriangles! As we go deeper into the subdivision, this ratio diminishes to one. It is this aspect of spherical triangles that is exploited by the nearly orthogonal wavelets we have presented here in equations (8) and (9). Because of this significant difference in the areas for the tetrahedron at lower levels, we usually use a base triangulation that utilizes the vertices of an octahedron and this includes the examples presented here.

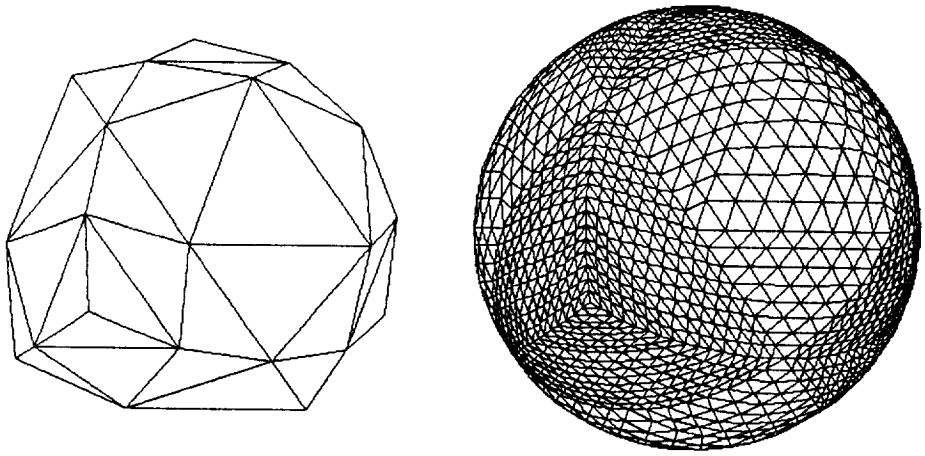
We have implemented the wavelets of equation (7) and the new wavelets of (8) and (9) and applied them to a variety of spherical data sets. For spherical images, the results of Table 1 are typical. Here we took a spherical image at level 7 (32,768 trixels) and computed the wavelet decomposition and then compared the RMS errors of the approximations using 10%, 50% and 90% of the terms with the largest normalized coefficients. Most generally the nearly orthogonal wavelets out performed the wavelets of (7), but there does not seem to be a clear choice between the two different nearly orthogonal wavelets.

	(7)	(8)	(9)
10%	12.63	12.70	12.37
50%	3.89	3.88	3.66
90%	0.329	0.324	0.303

Table 1. RMS errors for three different types of wavelets



Tetra, Level 1



Level 3 Level 6

Figure 3. Spherical Triangular Grid

More interestingly, we have applied our new wavelets to data consisting of a vector field defined over the sphere. In the example of Figure 4, we took a known vector field function and evaluated it to obtain the data. We applied the wavelet of equation (8). In the left column we show three views of the reconstruction of the wavelet decomposition using only 1% of the coefficients. This means we started with a vector valued function which had 2048 coefficients (one for each triangle). The first time we apply the decomposition equations we obtain a function which is piecewise constant over 512 triangles and plus the 1536 detail (or wavelet) functions. No information is lost, just a change of basis. The next step yields a lower resolution approximation consisting of 128

terms and 384 detail at this level plus the 1536 detail from the first decomposition step. This is continued two more times. This final expansion (of 2048 terms) is scanned and the 20 (1%) functions with the largest coefficients are summed to obtain the partial reconstruction. In the right column we show the reconstruction based upon the largest 409 (20%) coefficients. It is indistinguishable from the original function. It is interesting to note that the main features of the flow are maintained with an approximation which uses only 20 out of 2048 terms and the flow an be compressed to 409 terms (20%) with no visual loss. The graphs of Figure 4 and Figure 5 are computed using the tangent curve advection and topological methods described in [2].

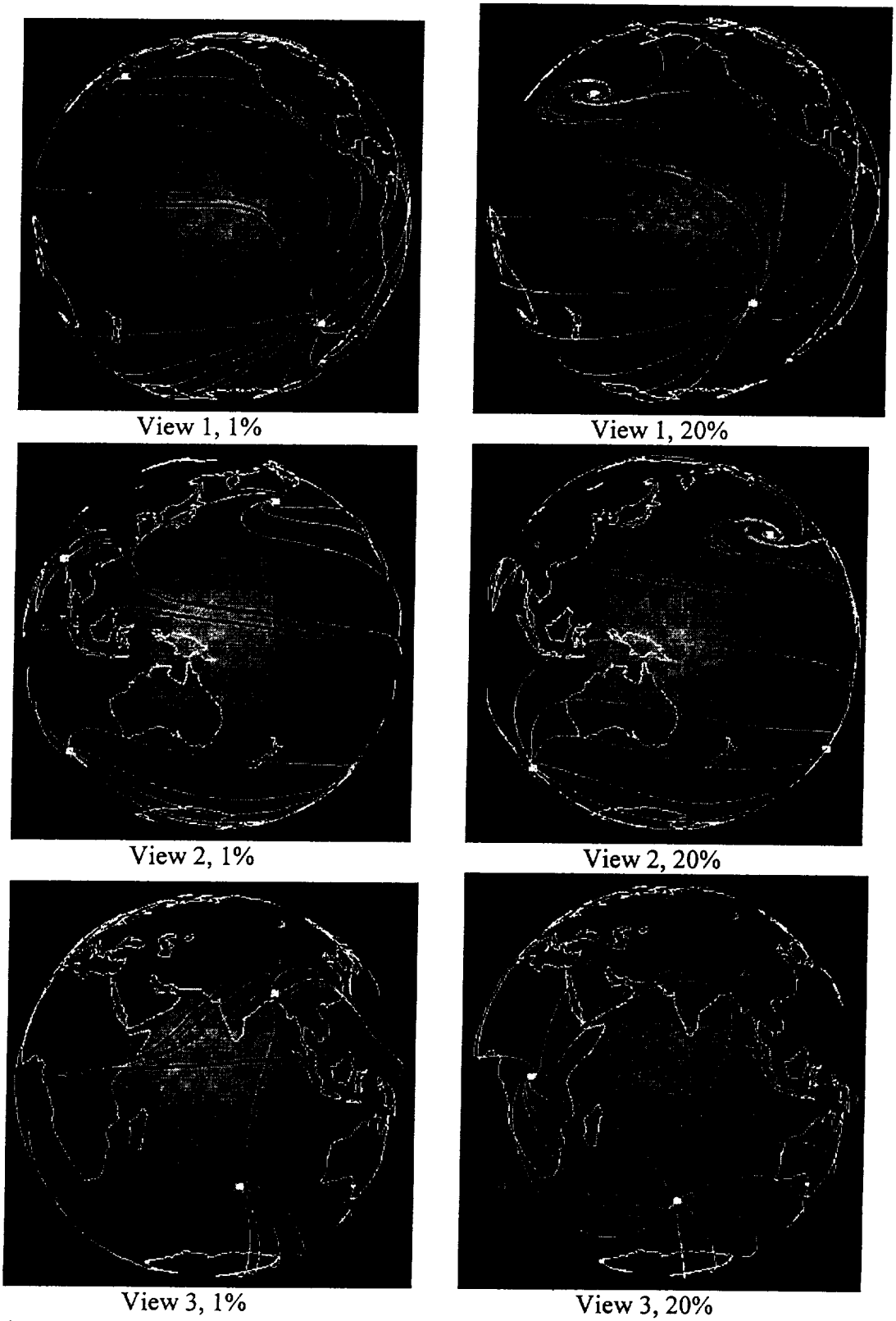


Figure 4. Partial (nearly orthogonal) spherical wavelet reconstruction of a flow field defined over a spherical domain.

In this next example shown in Figure 5, we applied the wavelets of equation (8) to some "real" data. This is data we obtained from Roger Crawfis and Nelson Max of Lawrence Livermore National Laboratory. Actually it is simulated data resulting from a global weather model. This data represents one slice in elevation and one time step. The first column shows three different views using a topological graph of the original data. In the second column the reconstruction based upon the 3% largest coefficients is shown. This low resolution approximation allows us to get an overview of the flow without the clutter and detail. The use of these types of models for analysis and visualization of scientific data is potentially very useful and a very exciting area of research.

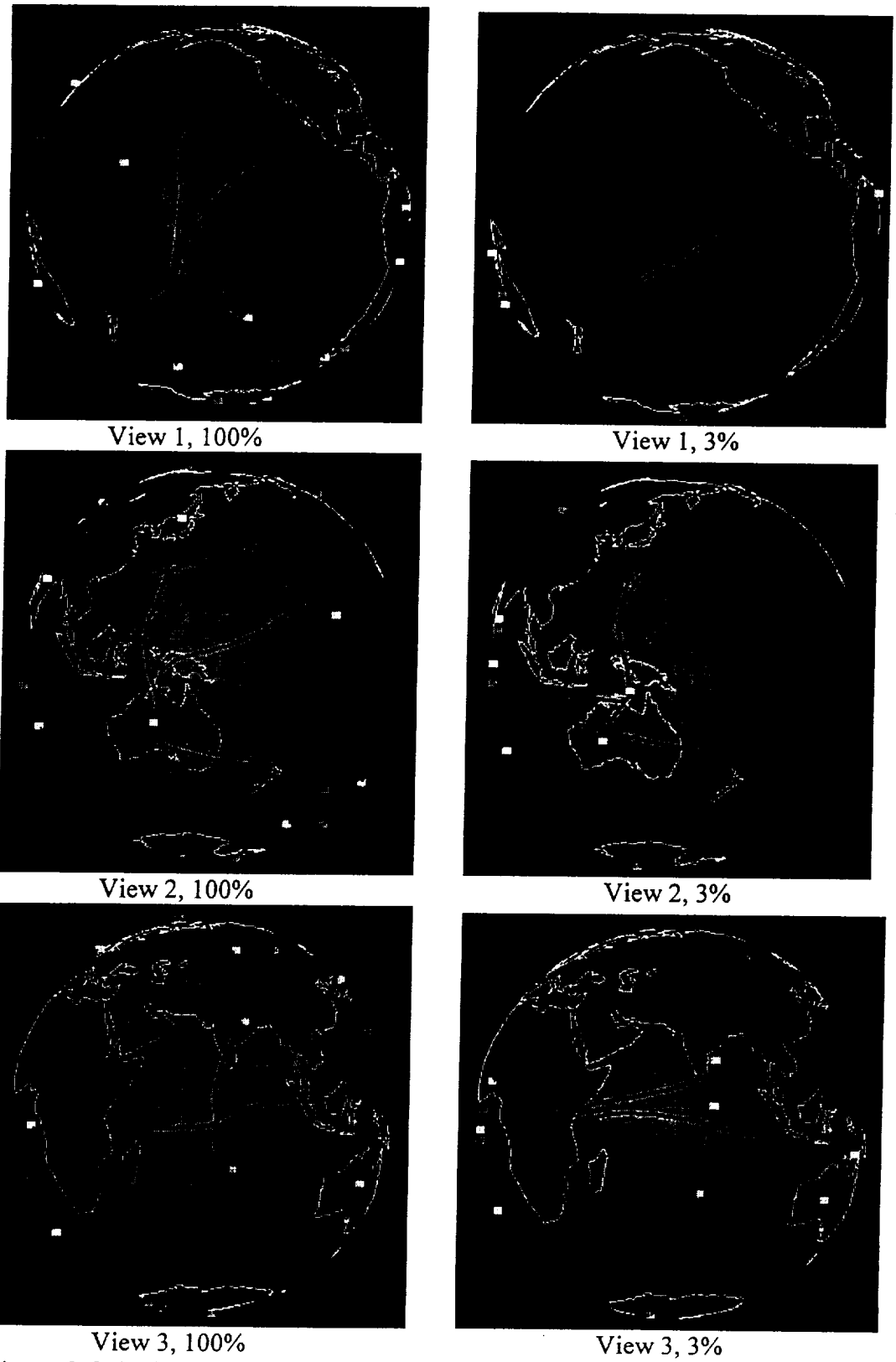


Figure 5. Spherical wavelet reconstruction of wind data from global weather model

Acknowledgments

We wish to acknowledge the support of the National Aeronautical and Space Administration under NASA-Ames Grant, NAG 2-990 and the support of the Office of Naval Research under grant N00014-97-1-0243.

References

1. E. J. Stollnitz, T. D. DeRose, and D. H. Salesin, Wavelets for Computer Graphics: Theory and Applications, Morgan Kaufmann, San Francisco, 1996

2. G. M. Nielson, I.-J. Jung, N. Srinivasan, J. Sung, and J.-B. Yoon, Tools for Computing Tangent Curves and Topological Graphs for Visualizing Piecewise Linearly Varying Vector Fields over Triangulated Domains, in: Scientific Visualization: Overviews, Methodologies and Techniques, G. Nielson, H. Hagen and H. Mueller, eds., IEEE Computer Society Press, Los Alamitos, CA, 1997, pp.517-557. www.computer.org/cspress

3. P. Schroeder and W. Smeldens, Spherical Wavelets: Efficiently Representing Functions on the Sphere, Computer Graphics Proceedings, SIGGRAPH 1995, pp. 161-172.

5

Computing the Separating Surface for Segmented Data

by

Gregory M. Nielson
Computer Science
Arizona State University
Tempe, AZ 85287-5406
nielson@asu.edu

and

Richard Franke
Mathematics
Naval Postgraduate School
Monterey, CA 93943-5216
rfranke@nps.navy.mil

March 27, 1997

Abstract

An algorithm for computing a triangulated surface which separates a collection of data points that have been segmented into a number of different classes is presented. The problem generalizes the concept of an isosurface which separates data points that have been segmented into only two classes: those for which data function values are above the threshold and those which are below the threshold value. The algorithm is very simple, easy to implement and applies without limit to the number of classes.

1. Introduction and Algorithm

In this paper we describe an algorithm for computing a triangulated surface which separates regions of different types. We assume that we have a collection of data points $V_i, i = 1,...,n$ and that each of these data points has been classified into one of several possible classes $c_i, i = 1,..., M$. This includes, for example, medical scanning device data that has been postprocessed by some segmentation procedure into different tissues or organ classes or physical simulation data that has been classified by material properties such solid, liquid or gas. For our algorithm, we assume that the data points V_i are the vertices of a tetrahedrization of the domain of interest. Two important application areas are where the data points lie on a 3D rectilinear grid or 3D curvilinear grid. In either case we preprocess the data by subdividing each voxel or hexahedron (curvilinear grid cell) into tetrahedra and proceed with our algorithm. See Figure 1 and Nielson [4]. Note that in the 6 tetrahedra split, each cube is split exactly as shown. In the 5 tetrahedra split, the mirror image of the split alternates in each direction with the one shown.

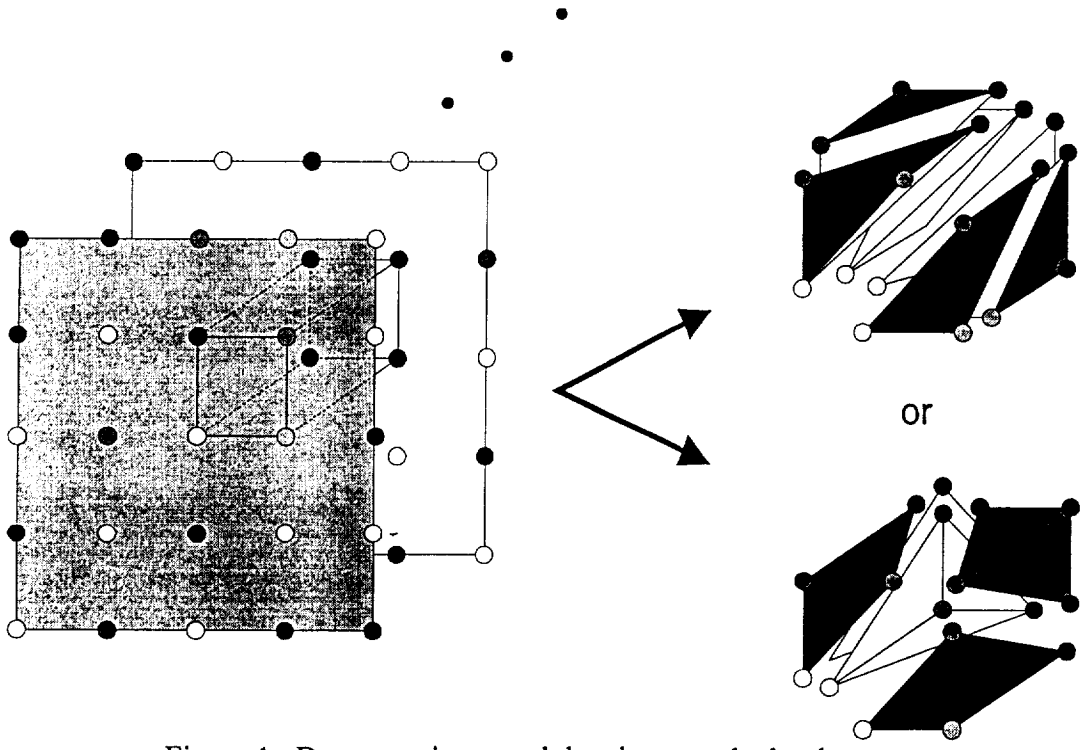


Figure 1. Decomposing voxel data into tetrahedra data.

Our goal is to produce a triangulated surface which separates the components (connected subsets) of the regions, $R_i, i = 1,..., M$, each containing the data of type $c_i, i = 1,..., M$. This surface can be viewed as a generalization of the isosurface often associated with the marching cube algorithm (see [3]). In the context of the mc algorithm the discrete vertices lying on a 3D rectilinear grid are classified into only two possible classes: either the value of the data function, δ, is above the threshold of the isosurface or

below this threshold. The isosurface then separates these two classes of data points into two regions. In the more general situation where there are several possible classes for data points, the separating surface is defined as $S = \bigcup_{i,j=1,\ldots M, j \neq j} (R_i \cap R_j)$. In the spirit of the mc algorithm, our algorithm processes each tetrahedron separately. Let V_i, V_j, V_k, V_l be the vertices of an arbitrary tetrahedron. If two vertices, say V_i and V_j are classified differently, we make reference to a point m_{ij} along the edge joining V_i and V_j. This point is where this edge intersects the surface separating the vertices V_i and V_j. This separating point can be anywhere on this edge and in some default situations a reasonable choice would be the midpoint. We mention some other considerations for selecting this point later. If three points of a face, V_i, V_j and V_k, are classified differently, we must make reference to a point m_{ijk} lying somewhere interior to this face. Again, for the topological aspects of our algorithm, it is not important where exactly this point lies on the face, but some practical considerations which we discuss later lead to reasonable choices for this point. And finally if all four points are classified differently, we need to reference a point m_t lying interior to the tetrahedron. This notation is further illustrated in Figure 2.

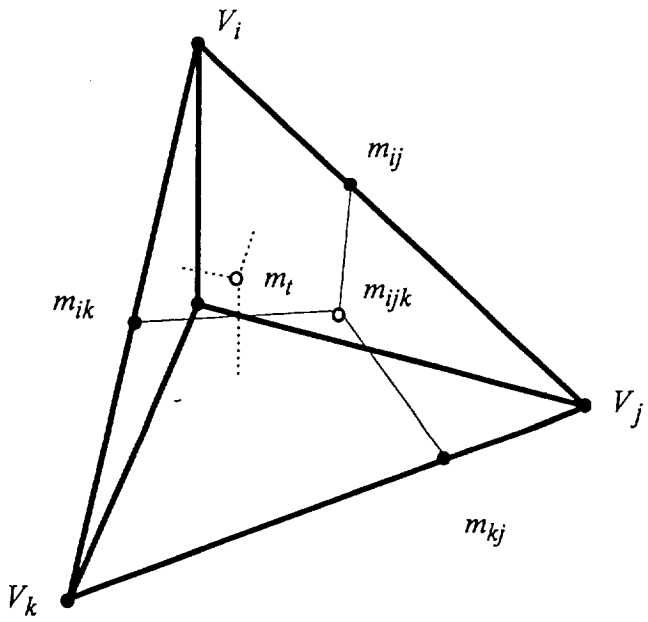


Figure 2. Notation used for vertices, mid-edge, mid-face and mid-tetrahedron points.

The strength of our tetrahedral-based algorithm is its simplicity and subsequent ease of implementation. There are only five cases to be considered: (0) all vertices are classified as one type (trivial case; no separating surface intersects the tetrahedron), (1) three vertices are of one class and one other vertex is of another class, (2) two vertices are of one class and two vertices are of another class, (3) two vertices of one class and the other

two vertices are of second and third classes, and (4) each vertex is of a different class. Because any configuration in one of these five cases can be rotated into a standard configuration, standardized algorithms can be used assuming that (local) vertices are labeled V_i, V_j, V_k and V_l.

The face of a tetrahedron having vertices of more than one type must be split. This can be seen in Figures 3, 4, 5 and 6 for the four nontrivial cases. When the vertices on a particular face of the tetrahedron are of only two types, the face is split along the line segment joining the mid-edge points on that face, say the points m_{il} and m_{jl}. This occurs in cases (1)-(3). When the vertices on a face are all three of a different type, the face must be split not only at the mid-edge points, but also at the mid-face point m_{ijk} interior to this face. The face is then divided by the line segments joining the mid-face point to the mid-edge points. When four different types are present then we must involve the mid-tetrahedron point m_t. The separating surface is to be represented as a union of triangles, so quadrilaterals that naturally occur in our algorithm must be triangulated by including one diagonal or the other. We adopt the convention that we will impose those diagonals that are consistent with a certain tetrahedrization of the four hexahedra that occur in case (4). See Figure 6. Since each hexahedron has a vertex of the cube as one vertex, we adopt the triangulation of the faces by diagonals from the tetrahedron vertices to the mid-face points, and the mid-edge points to the mid-tetrahedron point. We should point out that unless certain restrictions are put on the mid-edge, mid-face and mid-tetrahedron points, those quadrilaterals may not be planar. This causes no particular problem, although we note that the separating surface will be slightly different if different choices were made when triangulating those quadrilaterals.

In case (3), there is a dilemma as to whether the exterior quadrilateral faces (V_i, V_j, m_{jl}, m_{il} and V_i, V_j, m_{jk}, m_{ik} in Figure 5) would be divided from V_i to m_{jl} and V_i to m_{jk}, or from V_j to m_{il} and V_j to m_{ik} when tetrahedrizing the subvolumes. We have adopted the strategy that the order of the vertices in the description of the tetrahedrized volume will determine that; we choose V_i or V_j according to which has priority in the input list. Because we maintain order when sorting the unique classes of vertices for a particular tetrahedron, symbolically the first vertex is V_i. Hence we diagonalize the interior separating quadrilaterals using the line segments through m_{jkl} and m_{il}, and m_{jkl} and m_{ik}.

In case (2), we again wish to be consistent with some tetrahedrization of the volumes, two triangular prisms in this case. Thus, we follow the rule of connecting the priority vertex to the opposite mid-edges for each prism, i.e., V_i to m_{jk} and m_{jl}, and V_k to m_{il} and m_{jl}. After this is done for each prism, it is seen that the diagonal on the separating quadrilateral is arbitrary, and we choose the m_{jk} to m_{il} segment.

For completeness we list the triangles comprising the separating surface in each case.

Case (1):
Triangle: m_{il}, m_{jl}, m_{kl}

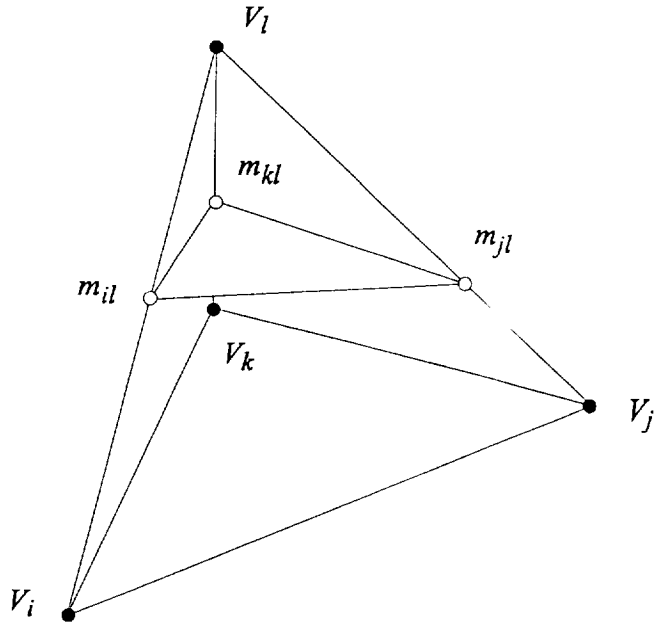


Figure 3. Case (1): Three vertices of one class and one vertex of another class.

Case (2):
Triangles: m_{ik}, m_{jk}, m_{il}; m_{il}, m_{jk}, m_{jl}

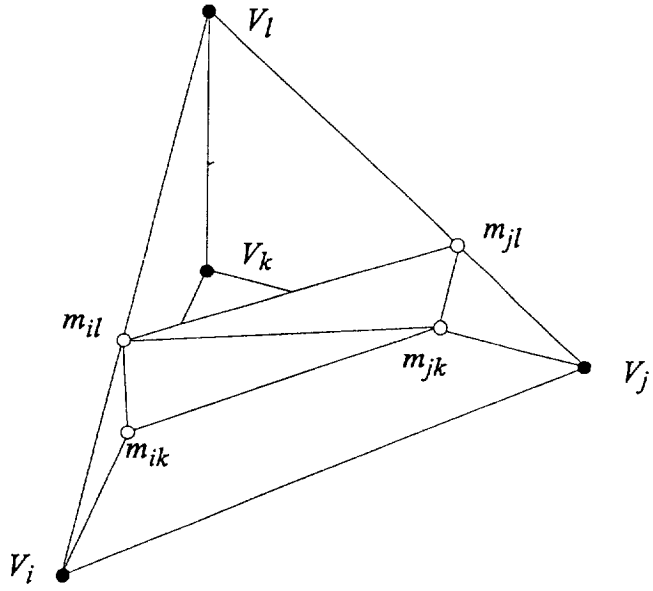


Figure 4. Case (2): Two vertices of one class and two vertices of another class.

Case (3):

Triangles: m_{kl}, m_{ikl}, m_{jkl}; m_{jk}, m_{ik}, m_{jkl}; m_{ik}, m_{jkl}, m_{ikl};
m_{jl}, m_{il}, m_{jkl}; m_{il}, m_{ikl}, m_{jkl}

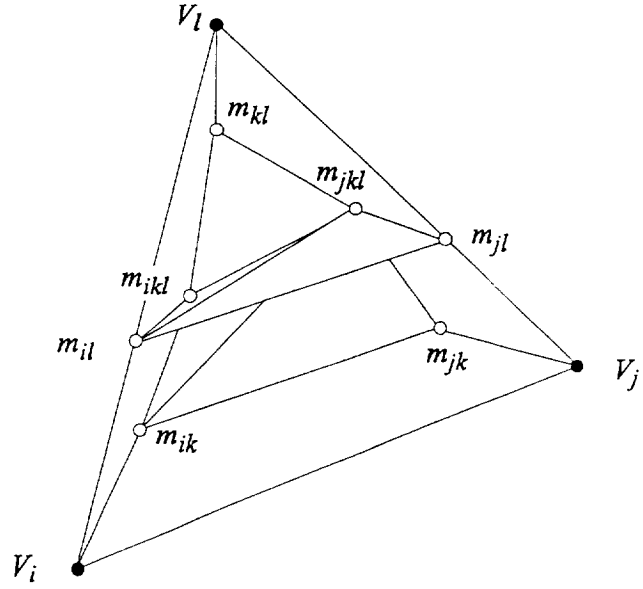


Figure 5. Case (3): Two vertices of one class and two other vertices each of another class.

Case (4):

Triangles: m_{ij}, m_{ijk}, m_t; m_{ij}, m_{ijl}, m_t; m_{jk}, m_{ijk}, m_t; m_{jk}, m_{jkl}, m_t;
m_{ik}, m_{ijk}, m_t; m_{ik}, m_{ikl}, m_t; m_{jl}, m_{jkl}, m_t; m_{jl}, m_{ijl}, m_t;
m_{il}, m_{ijl}, m_t; m_{il}, m_{ikl}, m_t; m_{kl}, m_{ikl}, m_t; m_{kl}, m_{jkl}, m_t

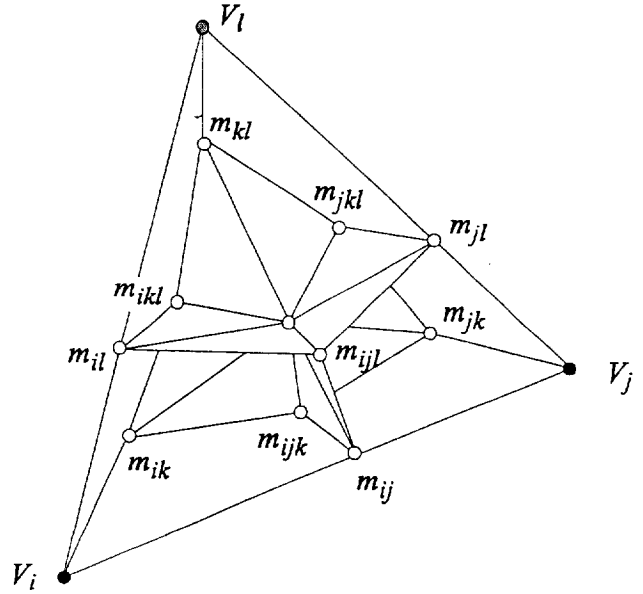


Figure 6. Case (4): Each vertex is a different class.

For the general description of our algorithm, we have kept the location of the mid-edge, mid-face and mid-tetrahedron points arbitrary. It is easy to present this way and also this allows for maximum flexibility. In some applications where there is no additional information on which to base any bias or adjustment, one just as well select these point to be the actual geometric midpoints. That is,

$$\begin{aligned} m_{ij} &= \frac{V_i + V_j}{2} \\ m_{ijk} &= \frac{m_{ij} + m_{ik} + m_{jk}}{3} \\ m_t &= \frac{m_{ijk} + m_{jkl} + m_{ikl} + m_{ijl}}{4} \end{aligned} \tag{1}$$

In some other applications where there is additional information some weights may be used to compute these values. For example, if data points are classified (or segmented) on an interval of values of some data function, then it might be useful to weight accordingly the computation of the mid-edge value. Assume that an arbitrary point $V = (x, y, z)$ is classified by the rules:

$$V \text{ is of class } c_\alpha \text{ provided } \alpha \le \delta(V) \le m$$

and

$$V \text{ is of class } c_\beta \text{ provided } m \le \delta(V) \le \beta .$$

Further assume that V_a and V_b are vertices that are classified as c_α and c_β, respectively. If we now consider the values of δ varying linearly along the edge joining V_a and V_b we could choose the mid-edge point to be the point where δ becomes equal to m which is the point where the classification changes from c_α to c_β . That would be

$$m_{ab} = \left(\frac{m - \delta(V_a)}{\delta(V_b) - \delta(V_a)} \right) V_b + \left(\frac{\delta(V_b) - m}{\delta(V_b) - \delta(V_a)} \right) V_a .$$

We have also used the following approach which is based upon the idea of a preference or probability matrix. The user specifies the off-diagonal values of an $M \times M$ matrix $P = \left(p_{ij} \right)$. These values serve as the weights for computing the mid-edge points. Let vertex V_a be of class c_α and V_b be of class c_β be vertices of the same tetrahedron. Then the edge joining V_a and V_b will intersect the separating surface at $p_{\alpha\beta} V_b + p_{\beta\alpha} V_a$. Since the separating point must be a convex combination of the vertices we require that $0 \le p_{ij} \le 1$ and $p_{ij} + p_{ji} = 1, i \ne j$. The interpretation of the matrix P can be in terms of

the "strength" of various classes relative to other classes, or it can be used to cause the separating surface to come close to (or stay further away from) certain classes of points. For example, it may be desirable to not overestimate the volume associated with a particular class, and in that case the values in the row of the matrix P associated with that class should be close to zero, forcing the separating surface close to the vertices of that class.

2. Examples

The first example has three regions. Points above the plane $z = 0$, and outside the sphere $x^2 + y^2 + z^2 = 0.25$, are of one type. Points below the plane and outside the sphere are in a second class and the points inside the sphere form the third class. Over the domain $\{(x,y,z): -0.025 \le x \le 0.625, -0.625 \le y \le 0.625, -0.625 \le z \le 0.625\}$ we formed a grid of size $14 \times 26 \times 26$ and classified the points on this grid according to the ideal. We then applied our algorithm, using the 5 tetrahedra per cube split. In this case we used the formulas of equation (1) for determining the mid-edge, mid-face and mid-tetrahedron points. The results are shown in Figure 7.

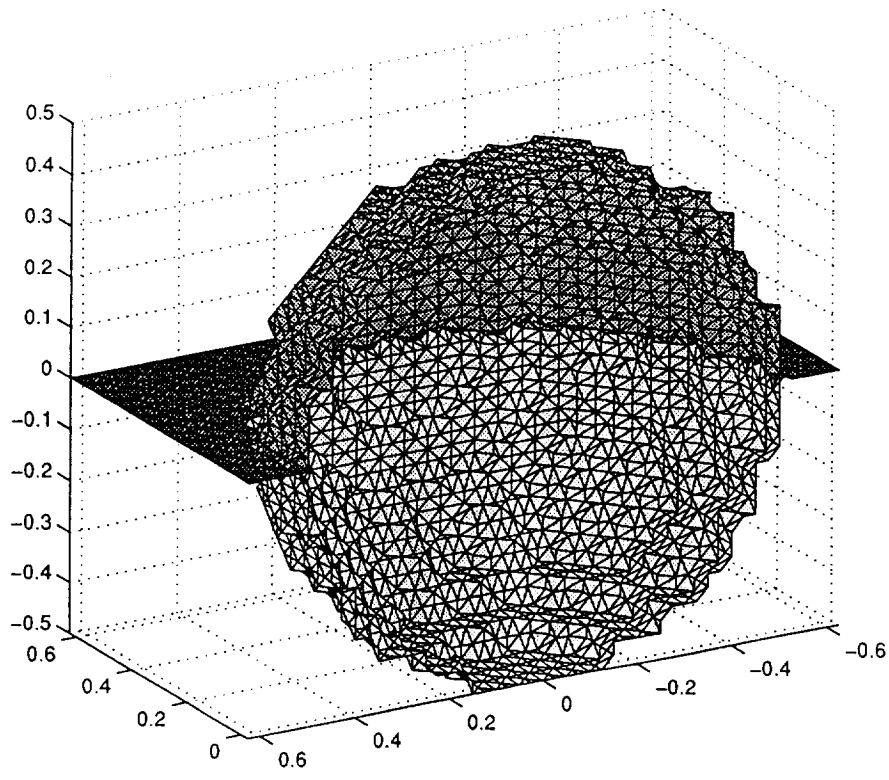


Figure 7. An example with three regions.

One of the features of our algorithm is that it is designed for scattered data. Our next example illustrates its use in that context. Because the algorithm we used tetrahedrized the convex hull, points near the boundary may be in tetrahedra with large aspect ratios. This causes distortion around the boundaries, so in Figure 8 the separating triangles near the boundary have been deleted. We note, however, that if the proper tetrahedrization is performed, the separating triangles could be processed using subsets of the entire data set because our algorithm guarantees a proper match across tetrahedron boundaries. For Figure 8, we generated 2000 random points in the region $\{(x,y,z): -0.2 \le x \le 1, -1 \le y \le 1, -1 \le z \le 1\}$. These points were then classified according to the same scheme as for the previous example. The point set was tetrahedrized and our algorithm applied with separating points being taken according to equation (1). To avoid the distraction of poor edge behavior, we then eliminated each triangle in the separating surface whose median point fell outside the region $\{(x,y,z): 0 \le x \le 0.75, -0.75 \le y \le 0.75, -0.75 \le z \le 0.75\}$. The results are given in Figure 8. The separating surface is necessarily jagged, but the proper character is shown.

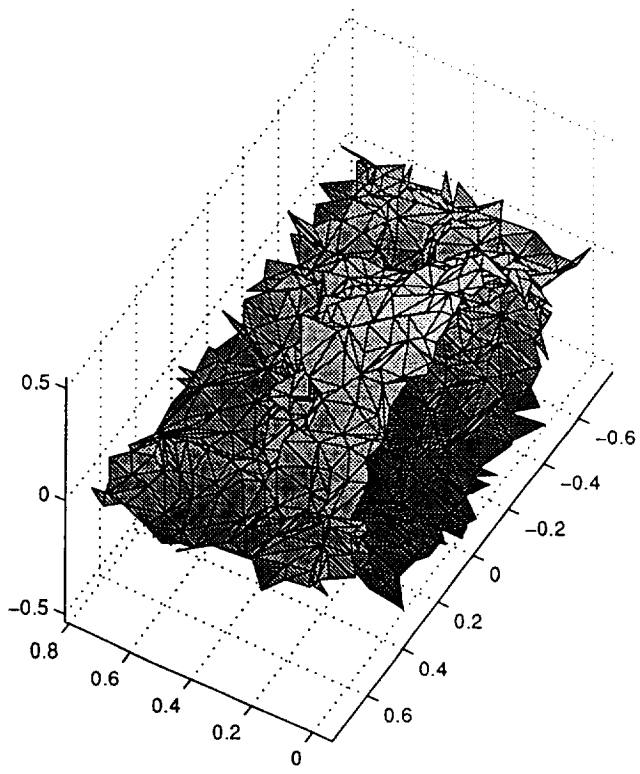


Figure 8. An example with three regions, random points.

The final example has five different regions. These regions are defined relative to several conic surfaces, and the volumes are described sequentially, with a given class overriding a lower numbered one. Above the paraboloid $z = 0.5\left(x^2 + y^2\right)$ the class is 1, while below (or on) the paraboloid the class is 3; inside the sphere $x^2 + y^2 + \left(z - 0.75\right)^2 = 0.4$, the class is 2; inside the sphere $x^2 + y^2 + \left(z + 1\right)^2 = 0.8$, the class is 4; and finally, inside the ellipsoid $2x^2 + \left(y - 0.5\right)^2 + \left(z - 0.1\right)^2 = 0.6$, the class is 5. We formed a $41 \times 41 \times 41$ grid over the domain $\left\{(x,y,z): -1 \le x \le 1, -1 \le y \le 1, -1 \le z \le 1\right\}$ and classified the points according to the definitions of the various regions. Using the 6 tetrahedra per cube split, we ran our algorithm on this data using the P matrix $P_{ij} = 1 - 0.2\left(j - i\right)$ for i<j. Because of the dense set of separating triangles, the results are shown as a shaded object in Figure 9.

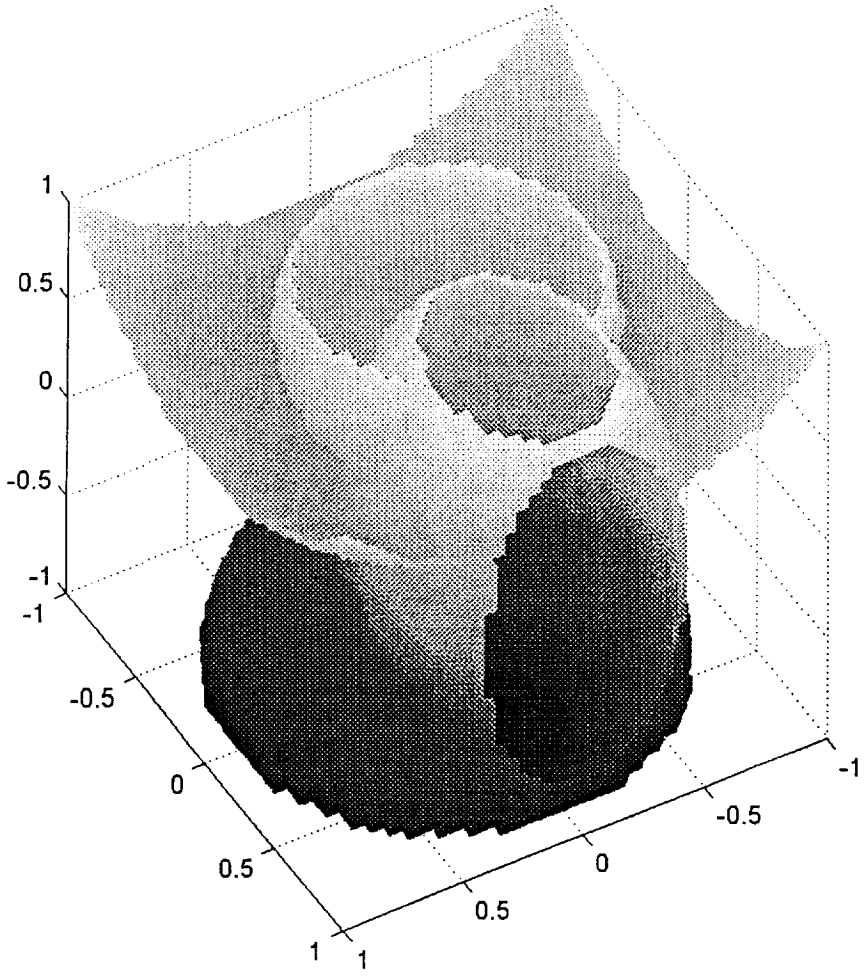


Figure 9. A surface separating five regions.

3. Summary and Remarks

An algorithm which produces a solution to a problem similar to that discussed here is described in [1]. This algorithm is based on case tables for the various configurations of classified vertices of a cube (or voxel) rather than a tetrahedron as used here. The number of equivalence classes of configurations (under rotations and possibly also mirroring transformations) for two and three classes of vertices on a cube is manageable but for more than three classes the authors mention that a table look-up approach is probably not viable due to the large number of different configurations.

It is possible to use the techniques of this paper to build the case tables for cube scenario, but the results are different (possibly simpler) than what is presented in [1] due to the difference of using trilinear interpolation compared to piecewise linear interpolation over tetrahedron. For example, in the case where $S = \{(x,y,z): R_\bullet(x,y,z) = R_\circ(x,y,z)\}$ is the separating surface with $R_\bullet(x,y,z) = xyz + (1-x)(1-y)z + (1-x)y(1-z) + x(1-y)(1-z)$ and $R_\circ(x,y,z) = (1-x)(1-y)(1-z) + xy(1-z) + x(1-y)z + (1-x)yz$ in the case of trilinear interpolation and $R_\bullet$ and $R_\circ$ appropriately defined for the piecewise linear case, the piecewise linear case (using the 6 tetrahedra split) leads to a separating surface which is a manifold topologically equivalent to a plane, but the trilinear interpolation method leads to a surface that is topologically equivalent to the intersection of three planes!

The algorithm presented here assumes that the data has been segmented into various classes and cannot be applied until this is accomplished. The problem of segmenting data is a highly nontrivial and currently unsolved problem. In no way does this present simple algorithm add to the solution of this problem, but possibly a more general algorithm which produces a tetrahedrized volume representation of the regions for different classes could be a useful tool in this regard. In a future paper, we will present such an algorithm.

Acknowledgments

We wish to acknowledge the support of the National Aeronautical and Space Administration under NASA-Ames Grant, NAG 2-990 and the support of the Office of Naval Research under grant N00014-97-1-0243. The second author was on sabbatical leave from NPS.