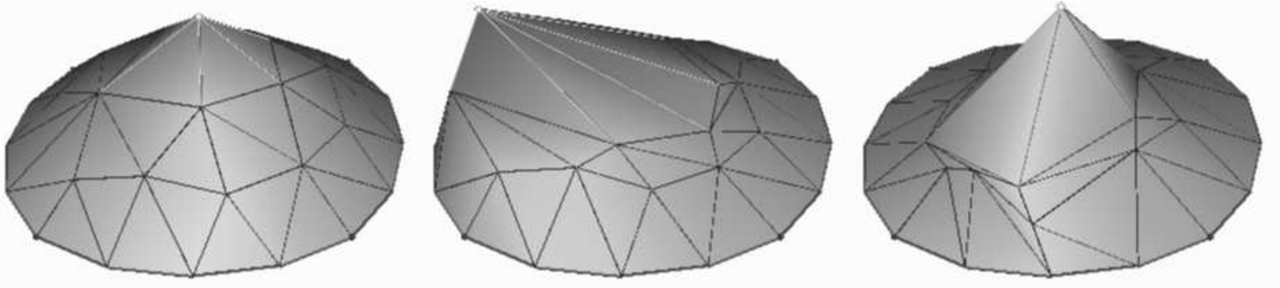


Figure 5.6: (1) A point in the center of the disk is elevated. (2) The point is moved to the side. Notice the neighborhood around the point becoming less crowded (3) After repeatedly moving the point around and finally returning it to its original place, the node distribution over the mesh has degraded to the point that the shape approximation performs poorly. The problem will be solved by the computational mesh maintainer described in the next chapter.

This system matrix is generally very sparse, as there is a zero for each pair of nodes not connected by an edge. The nonzero matrix entries will contain different values for every quadratic step taken (because the neighborhoods are re-parameterized each time). Therefore, rather than factor the Hessian explicitly (as in [WW92]), we solve the system using a conjugate-gradient method[Str86, She94], which only requires matrix-vector products with the system matrix, not an explicit representation. These vector products are computed by looping over the nodes, accumulating each neighborhood's contribution to the product. Technically, solving the linear system is $O(n^3)$ in the number of nodes; but we really expect the cost to scale as $O(n^2)$ on average because of sparsity arising from the local structure of the mesh (nodes rarely have more than 6 neighbors). The neighborhood-at-a-time approach to multiplication automatically takes advantage of the sparsity inherent in the global Hessian: since zeros in the Hessian correspond to node pairs that don't share a neighborhood, no time is wasted explicitly multiplying by such 0 entries.

Algorithm: Quadratic step (Figure 5.1)

1. `parameterize each nodal neighborhood (Section 5.3.1)`

2. `fit a quadratic function at each neighborhood (Section 5.3.2).`

3. `Compute new positions for the infield nodes by solving the system of linear equations in Equation 5.6. This may involve solving for more than one region simultaneously,` *e.g.*, `if interior control curves cut a surface into multiple regions that must meet smoothly at the control curves.`

At this point we should be through — we have a method of representing variational specifications, and a method of iteratively constructing an approximation to the corresponding variational shapes. Unfortunately, a look at Figure 5.6 shows that the iterations do not actually converge on a reasonable approximation. As the iterations progress, nodes tend to drift towards one another, clumping up near constraint points and curves, and leading to surfaces that bear little relation to the faired shapes requested.

The reason for the lack of convergence is that the relative sizes and shapes of the parametric neighborhoods in the mesh strongly influence the numerical conditioning of the calculations. As it happens, the shape functions say nothing about the distribution of nodes over a mesh, and therefore we should not be too surprised that the nodes drift about within the mesh, leaving poorly shaped triangles in their wake. Clearly, in addition to worrying about a mesh's shape, we must also be concerned with distributing and triangulating its nodes to yield a good *computational mesh*. This is taken up in the next chapter.

Summary

We have described a method of approximating thin plate surfaces of arbitrary topology using a p.l. surface mesh. A novel neighborhood parameterization scheme (*faceted* parameterizations) is used to fit a local polynomial surface at each node in an unstructured surface mesh, and thus estimate surface normals and curvatures at the nodes. This special parameterization also simplifies the evaluation of the geometric thin plate function at a node, generalizing to surfaces the technique of chordal parameterization. Approximate minimizers of the geometric elastica and thin plate fairness functions for curves and surfaces are computed through a sequence of quadratic minimizations over the mesh nodes. This approximation procedure will be used as a "region controller" to compute shapes for the faired curve and surface regions of our tagged meshes.

Chapter 6

Maintaining a Quality Mesh

Synopsis

For the mesh-based smooth surface approximation scheme of Chapter 5 to work, there must be a good computational mesh. In this chapter we discuss what that means, and address three principal concerns in maintaining a quality computational mesh: sample distribution, triangle shape, and node density.

1. **Sample Distribution:** nodes will be kept uniformly distributed by minimizing a sampling density objective function over the triangulated surface.
2. **Triangle Shape:** triangles will be kept well-shaped using an incremental surface Delaunay triangulation scheme.
3. **Sampling Density:** an appropriate sampling density will be maintained by automatically adding or deleting mesh nodes in response to changes in curve lengths or surface areas.

These mesh optimization steps will be interleaved with the shape optimization iterations of the previous chapter, so that a good computational mesh is always present for approximating minimizers of the curve and surface fairing objective functions.

In Chapter 4, we used a mesh as a representation of curve and surface topology. In Chapter 5, this same mesh was used as a triangulated approximation to a variational surface shape. In doing so, implicit use was made of the mesh in a third way — as a polyhedral manifold or *computational mesh* over which the finite-difference calculations were performed. In this chapter, we consider this third (and final) role for our meshes in more detail.

The shape optimization of the previous chapter says nothing about the relative distribution of nodes over the surface. Left to themselves, the nodes tend to drift towards one another and clump — resulting in a bad computational mesh, and subsequent failure of the fairing computations (Figure 5.6). Therefore, as the surface mesh changes size and shape during a design session, something must be done to maintain a *quality* computational mesh.

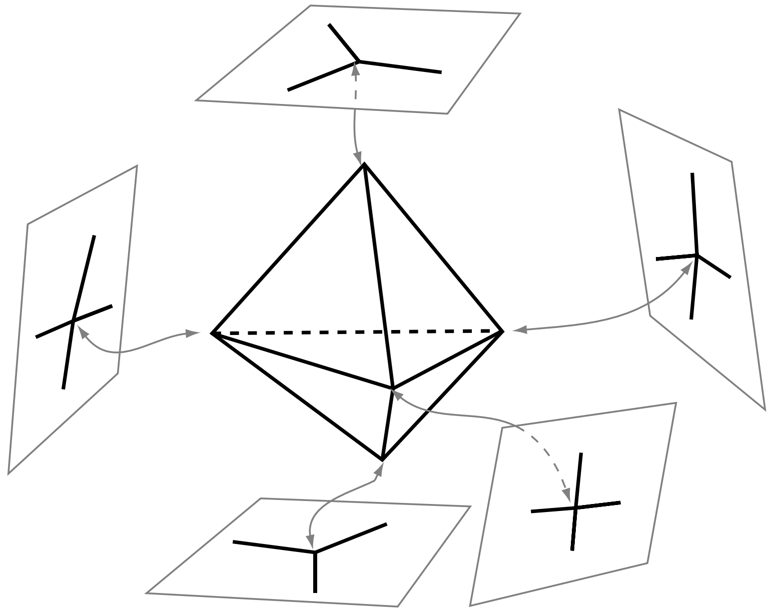


Figure 6.1: A collection of coordinate charts, one for each node-centered neighborhood.

What is a good computational mesh?

What characterizes a good mesh, and how to construct one, is properly the concern of the field of *numerical grid generation*[TWM85, Tho85]. Informally, a good mesh is one whose sample points are spread over the surface in a smooth way, and whose triangles are well-shaped. A smooth sample distribution might mean the nodes are evenly spaced (their density is constant in all directions); or, if they are dense in some areas and sparse in others, that the spacing changes smoothly (their density has a constant gradient over the surface). Triangles are well-shaped if they are not skinny

The reason quality meshes are important for these computations is that poor meshes — ones with unevenly scattered points or skinny triangles — lead to ill-conditioned numerics and magnified truncation error (the error due to our use of truncated Taylor series approximations for neighborhood shapes)[TM83, Hof82]. Schemes like the Taylor series reconstruction of Section 5.3 generally perform better as derivative estimators when a node's neighbor points aren't at wildly differing radial distances or angles.

In considering triangulated surfaces as computational meshes, recall that, strictly speaking, the computations of the previous chapter do not take place directly on the polyhedral surface. They are instead scattered over the various parameterizations constructed for each of the mesh neighborhoods. The computational domain may be more properly visualized as a triangulated surface with each node bearing a flattened picture of its neighborhood (Figure 6.1). But the geometric relationship between the mesh and these neighborhood maps is very close, with well-distributed points and well-shaped triangles on the surface generally leading to well-shaped parametric neighborhoods. We will often refer to the mesh and associated charts as the *computational mesh* (as opposed to the topological or approximating mesh) when we want to emphasize this role as domain for our smooth surface calculations.

6.1 Sample point distribution

We begin with the problem of distributing samples over a computational mesh. Because the mesh shape will be continuously changing as a result of user interaction, we rule out constructive, one-time schemes, such as those using stochastic point placement or Steiner point insertion algorithms[BE92]. A purely constructive scheme would require generating a brand-new sample distribution with each iteration of the shape optimization calculation, oblivious to the existing sampling. Instead, we consider relaxation-based schemes, which can take advantage of *temporal coherency* (the fact that the mesh shape does not change much from one time-step to the next), by using the previous mesh as a starting point for computing the next mesh. In addition to being (potentially) less work, this also avoids numerical stability problems that might arise from discontinuous changes in the domain partitioning that would likely arise for a complete re-sampling of the domain.

As discussed in Chapter 2, there are continuum methods and sample-based methods for controlling mesh vertex distribution over a physical domain. Continuum methods treat the mesh as a discretization of some continuous *computational* domain, and solve a system of grid generation equations that map the nodes into a smooth distribution over the physical domain. Generators such as the Laplacian can guarantee the mesh will not fold over on itself. Sample-based methods, on the other hand, operate on a collection of unorganized points as a discrete sampling of a computational or physical domain, with no specified interconnection topology, and use pairwise repulsion forces to distribute the points evenly over the space. As was discussed, it is unclear how to guarantee that a fixed triangulation of such a point set will not fold over on itself as the points move about on the surface. Nor do we wish to incur the computational cost that would accompany a wholesale re-triangulation of the points as in [Tur92]. For these reasons, a node positioning approach based on Laplace's equation will be used. Throughout this chapter the discussion focuses on surfaces; the reduction of these techniques to 1D, for curve re-sampling, is simple and straightforward enough that it will not be given separate treatment.

6.1.1 Laplace's equation for planar meshes

As discussed in Chapter 2, the planar Laplace's equation may be solved to smoothly map a regular computational grid (u, v) onto an irregular physical domain (x, y) :

$$u_{xx} + u_{yy} = 0, v_{xx} + v_{yy} = 0.$$

Notice that computational (u, v) are expressed as functions of the physical coordinates (x, y), reflecting the fact that we are concerned with node distribution in physical space. These equations can be obtained (via the Euler-Lagrange equation) as solutions to the variational minimization of the integral[BS82]

$$\iint ((u_x, u_y)^2 + (v_x, v_y)^2) dx\, dy.$$

The term $(u_x, u_y)^2$ measures the squared density of grid points as one moves along the u parameter line, similarly for v. Thus, minimizing this expression yields a mapping of grid points into the physical domain whose density changes smoothly. It is also possible

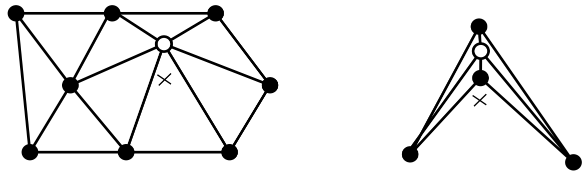


Figure 6.2: Laplacian smoothing in the plane: approximate a solution to the Laplace grid over the mesh by iteratively moving a node (white) to its neighborhood center ('x'). This breaks down for non-convex neighborhoods, whose area centers may lie outside the neighborhood polygon.

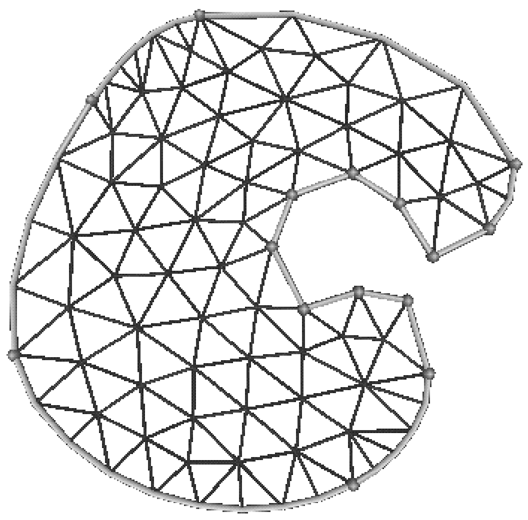


Figure 6.3: A smooth planar mesh generated by Laplacian smoothing, with boundary neighborhoods opened up into half-discs using our faceted parameterization scheme. The non-convex neighborhoods near the center are well-shaped because of the half-disc parameterization. The closely spaced boundary points in the upper left quadrant do not exert undue influence on the interior points because area centers are used instead of neighbor averages (unlike a spring system).

to view $(u_x, u_y)^2$ (and similarity for v) as measuring the energy in an imaginary spring connecting successive grid points. Minimizing the integral amounts to equilibrating the spring system, and again a smoothly changing mesh density results. Given a rectangular grid, it is straightforward to discretize and solve either of these by computing derivatives of the u and v functions using finite differences.

For an unstructured grid, things are not so simple – there are no (x, y) coordinate lines over which to measure grid density. A popular generalization extends the spring idea to unstructured meshes, minimizing the energy of a spring network that connects each node with its neighbors. If this is implemented by iteratively moving each mesh node towards the average of the neighbor points, it is known as *Laplacian smoothing*[Fie84]. One mildly objectionable feature of this approach is that, if there are fixed nodes closely spaced along the

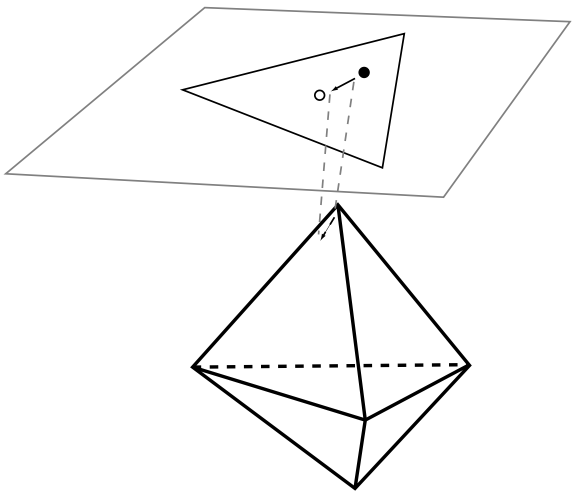


Figure 6.4: Depiction of the surface Laplacian smoothing technique for a single neighborhood. This is performed using the neigborhood's faceted parameterization rather than a projection onto the tangent plane.

mesh boundary, their springs tend to "gang up" on any shared interior nodes and pull them very close to the boundary. An alternative version of Laplacian smoothing moves nodes towards the area centers of their neighborhood *polygons*, and gives similar results but does not suffer this defect (Figure 6.3). Both methods work best with convex neighborhoods, and even better in conjunction with an incremental Delaunay triangulation scheme as reported in [Fie84]. But non-convex neighborhoods that don't contain their centroids are a problem, since the smoothing operation will cause the grid to fold back on itself (Figure 6.2). We will return to this point in a moment.

6.1.2 The surface Laplacian

Evaluating the Laplacian over a surface in 3D is a rather complicated affair, involving the so-called *Beltrami derivative* from differential geometry[Spi79b, War86], which "builds-in" a projection of derivatives onto surface tangent planes. That assumes a rectangular mesh, with the grid system re-expressed in terms of this (u, v) parameterization. For an unstructured surface mesh, things are of course even more complicated. Rather than use Beltrami derivatives, we can solve for each new node's position in 3D and then *a posteriori* project these offsets onto the nodes' tangent planes, similar to Barr, *et al.*[BCGH92].

This still requires that fitting a local quadratic to each neighborhood in order to estimate the tangent plane, an expense we would like to avoid. Analogously to the polygon centroid scheme in 2D, we will instead develop a *surface* Laplacian smoothing scheme in which nodes are moved towards their *parametric* neighborhood centers. This can be accomplished without any kind of fitting or tangent plane projection operations, by using our faceted parameterizations from the previous chapter. A smoothing iteration will consist of, for each mesh node, computing the area center of its faceted parameterization, mapping this back to a point on the polygonal surface, and sliding the node toward this point (Fig-

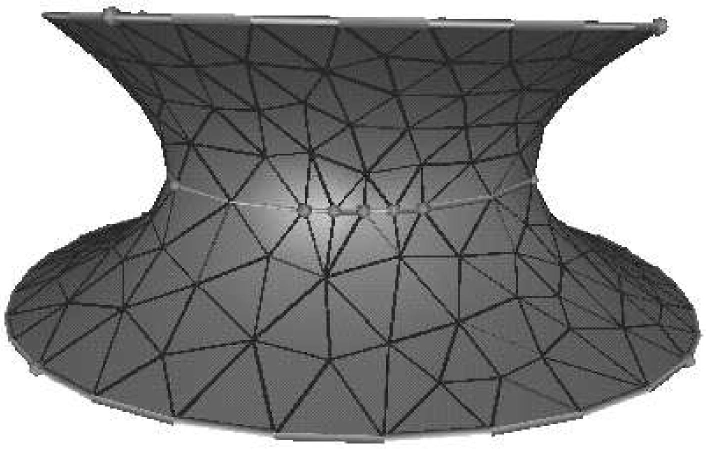


Figure 6.5: Laplacian smoothing over a surface triangulation. As with the earlier 2D example, we have crowded a few constraint points on the control curve to demonstrate infield behavior near such regions.

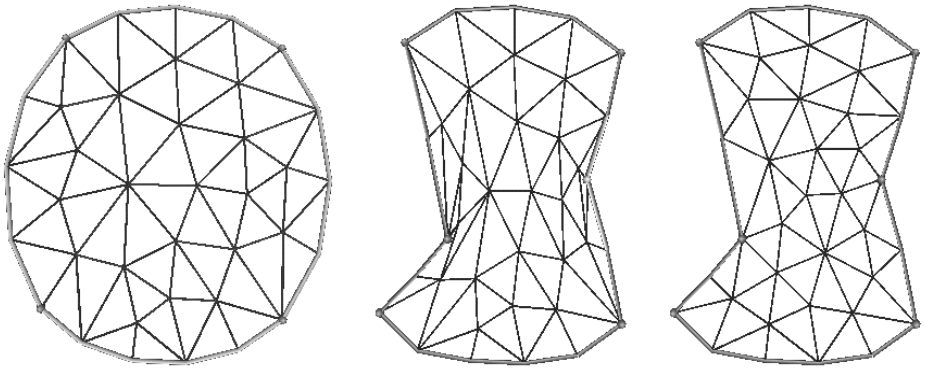


Figure 6.6: Why sample positioning isn't enough: in each figure, sample positions are controlled using Laplacian smoothing. When the original mesh (left) is deformed while its triangulation remains fixed (middle), badly shaped triangles cannot be avoided. Maintaining a Delaunay triangulation over the points (right) improves the mesh dramatically.

ure 6.4). This also has an advantage over the planar Laplacian smoothing scheme: convex boundary neighborhoods, like that in Figure 6.2, are opened up into half-discs by the faceted parameterization, and thus their area centers always lie in the interior. An example of a planar mesh so generated is Figure 6.3. A surface is demonstrated in Figure 6.5.

6.2 Surface triangulation

The sample distribution scheme above moves nodes around within their respective triangulated neighborhoods, assuming a fixed mesh connectivity. Although this does improve the quality of the triangulation by equilibrating edge lengths somewhat, it is not sufficient to yield the best possible surface triangulation over a given set of points with given topology. For example, Figure 6.6 shows a mesh whose sample distribution minimizes $\hat{\mathcal{E}}_{\mathrm{smooth}}$, yet which has many skinny triangles. The figure also shows a Delaunay triangulation (DT) over the nodes. As discussed in Chapter 2, the planar DT's max/min angle property makes it attractive for computational mesh generation, because it eliminates skinny triangles whenever possible.

In this section we outline a scheme for dynamically maintaining a surface DT over the nodes as they change position during surface re-shaping. The max/min angle property of the planar DT will carry over into the surface DT. As with the parameterization procedures of Section 5.3.1, mesh maintenance will be formulated so that it remains faithful to the given mesh topology, and never relies on the kind of projection and consistency testing found in [Tur92, SZL93].

6.2.1 The surface Delaunay triangulation

The classical DT is defined over a planar set of points, and generalizing it to 3D surfaces is not particularly straightforward. Recent work by Chew[Che93] generalizes the empty circumcircle characterization of the DT to triangulations over surfaces (Figure 6.7). It turns out that a naïve generalization using geodesic distances to inscribe circumcircles on surfaces is undesirable, because it admits strange situations like self-intersecting circumcircles (and would be expensive to test, as well). Instead, Chew defines the circumcircle of a triangle on a curved surface as the surface's intersection with the sphere that includes the three triangle vertices and whose center lies on the surface (Figure 6.8). If a local flatness assumption holds (the surface normals within the union of circumcircles associated with a given quadrilateral vary by less than $\pi/2$), it can be shown that this definition shares a consistency property with the planar circumcircle definition that makes it a reasonable generalization of the planar DT. The result is a unique triangulation that maximizes the minimum included angle in 3D.

An important consequence of this consistency property is that a surface DT can be incrementally recovered from a valid initial surface triangulation by iterative edge-flipping. One repeatedly tests quadrilaterals in the mesh to see if flipping an edge within the quadrilateral will improve the triangulation (by increasing the minimum angle), and continues flipping such edges until the DT has been restored (Figure 6.9, algorithm below). This is just like the standard planar DT edge-flipping algorithm, but angles are measured in 3D. The edge-flips preserve the topological type of the surface mesh, and thus are a topologically safe way to perform re-triangulation over a surface.

6.2.2 Constrained triangulation

In constructing the surface DT from an initial surface triangulation, there will be edges that must not be disturbed, such as those that are part of embedded control curves. A scheme that incorporates these so-called *source edges* is referred to as a *constrained Delaunay triangulation*, or CDT [DFFP85]. It enjoys the same max/min-angle property as the DT (when restricted to consider only triangulations *that include the source edges*), and thus

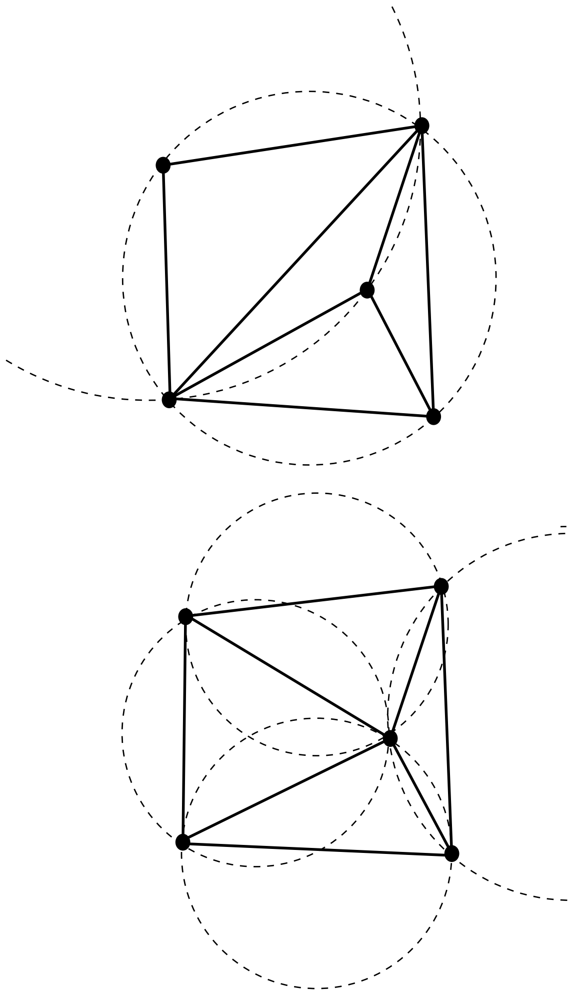


Figure 6.7: One characterization of the DT is that its triangles' circumcircles are all empty. The upper triangulation contains an edge that is not in the DT, and its associated circumcircles contain other vertices. This is not true of the lower figure, the DT of the points.

an incremental edge-flipping restoration for the CDT is also possible. One simply never considers flipping a source edge.

6.2.3 The flatness assumption

In order for edge-flipping to terminate, producing the unique surface DT, the surface must satisfy Chew's local flatness assumption that no dihedral angle exceeds $\pi/2$. Rather than enforce this requirement by refining the triangulation in highly curved neighborhoods, as suggested in [Che93], we have found that it works well in practice to relax the requirement by maintaining only an approximate DT. We temporarily *source* any edges with sharp dihedral angles, rather than allowing them to be flipped, and this preserves the algorithm's termination guarantee.

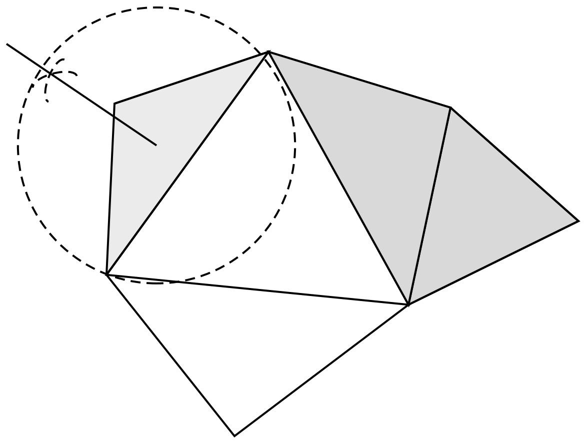


Figure 6.8: Circumcircles on a surface: for a given triangular face, there is a family of spheres through the three vertices, and their centers all lie on a line. The intersection of this line with the surface (not necessarily the given face) defines the center of a "circumsphere", and the intersection of this sphere generalizes the notion of circumcircles to curved surfaces.

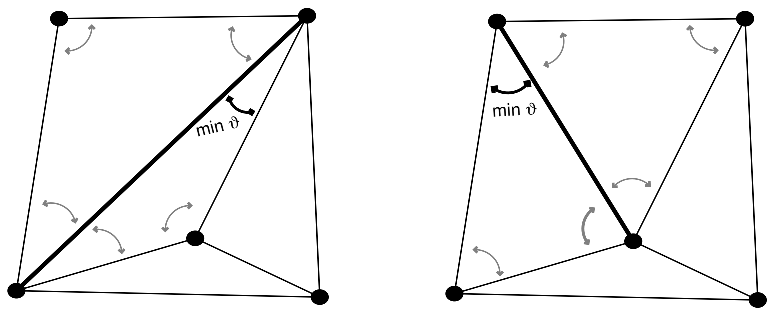


Figure 6.9: Construction of the planar Delaunay triangulation through iterative edge-flipping. The highlighted diagonal on the left is *reversed* within its quadrilateral. Flipping the edge increases the minimum included angle, and restores the DT (right).

6.2.4 An incremental surface CDT

For a valid surface triangulation, and an edge e not on the boundary, let Q_e be the quadrilateral formed by taking the two triangles on either side of e (Figure 6.9). We say that Q_e is *reversed* if e forms a smaller minimum angle with the outside edges than the other diagonal does.

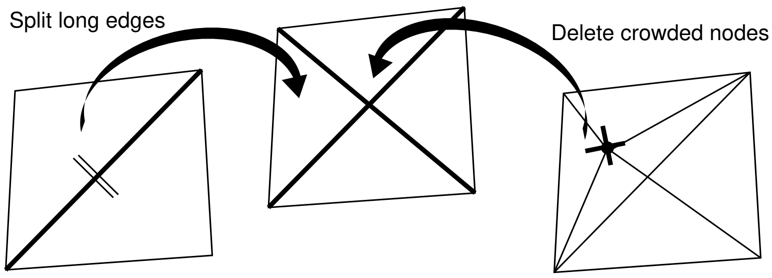


Figure 6.10: Adaptive mesh refinement: split the edges that are too long, and delete the nodes that are too crowded. The triangulator improves the mesh afterwards.

Algorithm: Restore-CDT

1. Put all non-boundary, non-source edges into a queue.
2. Remove the first edge e from the queue.
3. If Q_e is reversed, and the dihedral angle across e does not exceed $\pi/2$, and flipping e does not introduce a dihedral angle exceeding $\pi/2$, replace e in the mesh with Q_e's other diagonal. Add the non-boundary, non-source edges of Q_e to the queue if not already present.
4. Continue removing, checking, and possibly flipping edges from the queue. When the queue is empty, the CDT has been restored.

This is a very convenient and inexpensive way to dynamically maintain a CDT over a gradually changing set of vertex positions. Although the algorithm formally terminates in $\mathcal{O}(edges^2)$ flips, it is more often the case that only a few edges will need to be flipped at any one time assuming the changes in vertex positions are small and given that we began with a DT over the original positions. We have found it desirable to introduce a bit of hysteresis by only flipping edges if they increase their local minimum-angle by some small minimum (1-3 degrees). This produces an approximate CDT by making edges somewhat more reluctant to flip, but it reduces “chattering”.

6.3 Controlling the number of samples

The sample distribution and re-triangulation procedures above let us make optimal use of a given number of sample points. But as surface area grows and shrinks we would like to adjust the absolute number of samples so that the sampling density remains relatively fixed. Refinement/re-zoning schemes in mesh-based computations can be fairly complex. But given the machinery already in place, this last task is very simple. Edge lengths are measured in 3D, and an edge-split triggered if any two neighbors are too far apart. Similarly, if any node is too close to each of its neighbors, the node is destroyed using the node deletion algorithm from Chapter 4 (Figure 6.10). The sampling distribution and re-triangulation procedures above take care of restoring a quality mesh after one of these mesh transformation operations.

6.4 Mesh improvement algorithm

The techniques of the previous section can be packaged in a single "mesh improvement" step, to be applied to p.l. curves and surfaces. As with the quadratic shape steps of the previous chapter, the following steps will be iterated so that the mesh converges on a good distribution of nodes with well-shaped triangles (the interleaving of mesh shape and mesh improvement steps is discussed in the next chapter):

Algorithm: Mesh improvement step

1. `Mesh refinement: split long edges and delete crowded nodes.`
2. `Re-triangulation (surfaces only): restore the surface DT by iteratively flipping edges to eliminate skinny triangles`
3. `Mesh smoothing: adjust the distribution of the nodes by performing a curve or surface Laplacian smoothing step.`

Ordering the operations this way seems to leave the best mesh at the end of the step. The re-triangulator gets a chance to clean up after the simple edge-splitting refinement operations, and the mesh smoother then gets to optimize neighborhood shapes after the triangulator has had its say. Examples of resulting surface meshes appear throughout this document.

Summary

We have described a novel method of maintaining a good computational mesh over a triangulated surface of arbitrary topology. The method is iterative and incremental, making it appropriate for use in an interactive surface modeler in which shapes change gradually over time. Nodes are kept evenly distributed over the surface by solving a version of Laplace's equation restricted to the surface. A surface Delaunay triangulation maintains well-shaped triangles over the nodes. A simple enrichment/depletion scheme adds nodes to sparse areas of the mesh, and deletes nodes from crowded areas.

Chapter 7

Implementing the Modeler

Synopsis

We combine the shape approximation and mesh optimization computations of the previous chapters. The combination serves as a black box that, given a smoothly time-varying variational specification (stored as a tagged topological mesh), outputs shape approximations in real time. We discuss a variety of region shape controllers that can be plugged into the mesh to control the shapes of constrained regions, and consider other ways in which this "variational substrate" can be built upon to create interactive modeling application programs.

In previous chapters we have been through good deal of mathematical machinery, and have devoted some discussion to solution and minimization techniques for curve and surface fairing equations. But little has been said about how to combine all this to yield a working modeler. In this chapter we discuss the implementation of a variational surface modeler. The first four sections are devoted to algorithms and data-structures that wrap the mathematical machinery of the previous chapters into a "variational substrate." The goal is a black box that will accept variational surface specifications as input and serve up mesh approximations as output, thus encapsulating the details of shape approximations and mesh optimizations.

We begin with a discussion of shape computations, describing how the shape and sampling computations of the previous chapters fit together and are applied to our mesh structures. Next is a brief a discussion of rendering techniques for meshed surfaces. Following this is a user (programmer) description of the variational substrate — what kinds of data structures and operations one works with in building a modeler on top of this substrate. The remaining sections of the chapter discuss examples of higher-level modeling operations specified in terms of this programmer's abstraction, drawn from a prototype direct-manipulation surface modeler that uses variational curves and surfaces as its basic free-form shape representation. Just as other modelers operate directly on B-spline or Bezier patch parameters, and leave it to lower-level routines to actually render the patches, our variational modeler operates directly on a variational specification (stored as a tagged topological mesh), and leaves it to the approximation machinery to maintain a good mesh representation of the shape.

7.1 Computing variational mesh and curve shapes

As discussed in Chapter 4, our tagged curve and surface meshes are decomposed into a union of disjoint regions, and each of the regions is assigned its own shape controller. Shape controllers for variational mesh regions are computed by iteratively minimizing a sequence of quadratic approximations to the true nonlinear objective function, as discussed in Chapter 5. On the other hand, *constrained* mesh regions will typically be tied to *explicit* controllers. Explicit controllers will write their positional information directly into the mesh points they control. A controller could be as simple as an interpolation point tracking a moving mouse position, or as complicated as the examples in Section 7.4.1. We consider here at a low level how to orchestrate the computations of these various controllers to produce an interactive modeler.

7.1.1 Basic shape iteration

```
1. For each constraint point (on curves or surfaces), invoke its region controller
   to update its position.

2. For each curve segment with an explicit shape controller, invoke the controller
   to compute a new curve shape.  The controller may depend on previously computed
   constraint point positions (from step 1).

3. For each faired curve segment, execute a step towards the minimum shape
   (Section 5.6), followed by a curve mesh improvement step (Section 6.4).

4. For each surface region with an explicit shape controller, invoke the controller
   to compute a new region shape.  The controller may depend on points or curves
   computed in steps 1, 2, or 3.

5. For each faired surface region, execute a step towards the minimum shape,
   followed by a surface mesh improvement step.
```

In the computations above, faired geometry is always computed after explicitly controlled geometry has been updated, so that faired elements may adjust their shapes in response to changes in the explicit elements.

In experimenting with this combination of shape and mesh improvement iterations, we tried leaving out various components of the quadratic and mesh improvement steps, or performing the components at differing frequencies. For example, we tried reparameterizing neighborhoods every few iterations instead of every time, and similarly for refinement and re-triangulation phases. The best overall behavior, in terms of stability and speed of convergence, resulted from performing each of the steps once each time through the approximation loop. For added speed and stability, we have found it helps to turn off re-parameterization, re-triangulation, and edge refinement whenever an "upstream" shape is changing rapidly (*e.g.*, when a user is tugging on a control curve). This is because a fast-moving constraint point typically causes the attached curve or surface to momentarily elongate, and automatic refinement at this point would likely be spurious and needlessly slow down the fairing computation. Keeping the mesh topology fixed at such times allows the shape to respond quickly to gross changes in the constraints. Once the user has stopped manipulating the control, surface refinement and re-triangulation may resume at a more leisurely pace.

7.1.2 Mesh improvement for other region controllers

The mesh improvement iteration above is general enough that it may also be used with other region controllers, not just variational patches. For example, in the implicit surface shape controller below (Section 7.4.1), we only worry about attracting the mesh nodes to the implicit surface, and rely on the mesh-improvement iteration above to keep nodes dispersed over the surface and nicely triangulated. In order for this to work well, the region controller should not attempt to move nodes in tangential directions when adjusting surface shape, but only move nodes along their current surface normals, as was discussed in Section 5.5.4. Tangential motion could compete with the mesh smoothing step, and delay or even prevent convergence. Of course, a region controller is always free to manage its own node distribution if desired.

7.1.3 Convergence

There is little we can say formally about the convergence of these iterations; but in our experience the minimization has been well-behaved over a wide range of configurations. As with almost any nonlinear optimization posed as a sequence of quadratic subproblems, a caveat is that "reasonable" initial surface shapes must be used. It has not been necessary to precisely characterize what constitute good starting values for the optimization: because of the interactive nature of our system, changes to shape are generally incremental, with the endpoint of one iteration serving as the starting point of the next. It is certainly possible for a malicious user to make such a drastic change to a constraint in such a small interval of time that the computation diverges (typically as a result of the mesh optimization not keeping up with a fast-moving constraint). In practice, the visual feedback from the modeler (in the form of animated, smoothly-evolving surface shapes) has been adequate to enable the user to change constraint shapes at a safe and reasonable speed. The place where reasonable starting shapes must be generated *a priori* is when new surfaces are created through skinning operations. Our system uses only two such skinning operations, that construct cylinders and sheets, and linearly interpolating between their defining boundary curves gives reasonable starting shapes.

Finally, it should be noted that curvature minimization, while well-behaved in a wide range of surface configurations, is not always the most desirable shape objective function. There are some configurations, such as narrow cylinders, in which it performs poorly (cylinders pinch in at the waist, and may even collapse). We re-visit some other possible fairness functions (and associated difficulties) in Section 8.3.

7.1.4 Solver speeds

As discussed in Chapter 5, the equations to be solved when fairing curves and surfaces are very sparse. A conjugate gradient method is used to solve the associated linear systems, because it takes advantage of this sparsity without any additional work on our part (*i.e.*, we needn't maintain special sparse matrix data structures). Technically, each linear system requires $O(n^3)$ operations to solve, where n is the number of nodes; but we really expect performance to scale better on average because of sparse Hessian matrices arising from the local structure of the mesh. We cannot argue for this bound *a priori* as in Chapter 4 using the Euler characteristic. The worst case surface mesh is a disc with a single central neighborhood, for then the Hessian does have $O(n^2)$ entries and solution takes $O(n^3)$ operations. But the re-triangulator tends to break surfaces up into evenly-sized neighborhoods (because of the relation of the DT to the Voronoi diagram), typically having 5–7 neighbors, so that the number of nonzero entries is effectively a constant times the number of nodes. This means the solver behavior scales more like $O(n^2)$ because the matrix-vector multiplies used by the conjugate gradient solver take advantage of this sparsity.

Nonetheless, for large meshes, solution times are still much too long for interactive modeling, where one can afford no more than 100ms per iteration (assuming the mesh will be re-rendered between each quadratic step). We therefore take advantage of another feature of the conjugate gradient solver: rather than viewing it as a "black box" linear system solver, we note that it is actually an iterative solver, with successive iterations closing in on better solutions to the supplied linear system. Rather than run the method to full convergence, the conjugate gradient solver is allowed only a fixed number of iterations (10–20) per solve/redraw cycle. This slows global convergence somewhat, but lets the mesh be redrawn often enough to maintain the illusion of a smoothly deforming surface over time. We have worked comfortably with models approaching 1000 nodes, running on a Silicon Graphics Indigo (R4000).

A side-effect of bailing out of the conjugate gradient solver early is that the intermediate stages rendered are not themselves true minima, and the intermediate surfaces look as if they are moving through a viscous medium as the solution converges over time. It takes anywhere from 1 to 5 seconds for a surface to reach quiescence after a large change in a constraint position. Intermediate shapes are continuously rendered during this time.

7.2 Rendering

7.2.1 Fast rendering

In between each shape iteration, the mesh shape will be redrawn for the user. To quickly render a mesh as a "smooth" surface, we evaluate normals at mesh nodes (using the fitted node quadratics), use these to compute a lighted color for each vertex, and render each triangular facet by interpolating its vertex colors across the facet. This kind of rendering (*Gouraud shading*) is supported in hardware on the graphics workstations we use (Silicon Graphics Indigos), and surfaces thus rendered are of passable quality for all but the coarsest of refinement levels. The technique has been used in all of the "smooth" renderings in this dissertation.

7.2.2 High-quality rendering

If speed is not an issue, a higher-quality (and accordingly more complex) approach to rendering uses the mesh's vertex/normal (and possibly curvature) data to fit a network of smooth patches. One of the simplest schemes is Nielson's G^1 triangle[Nie87], but as discussed earlier, this and other purely local fitting approaches can yield unattractive curvature discontinuities across patch boundaries. Moreton's MVS interpolator[MS92], or Halstead, *et al.*'s subdivision surface fairing[HKD93] are examples of globally smooth surface interpolation schemes that could be applied to a network of patches that would fill in the triangular facets of the mesh. Alternatively, if the cost of a global scheme is acceptable, one could use the variational specification encoded in the mesh to drive a high-quality smooth patch scheme, using the same objectives and constraints that were used in the p.l. mesh approximation.

7.2.3 Orienting surfaces

An issue that arises when rendering meshes as lighted surfaces is that vertex normals must be consistently oriented "outward" across the surface. The choice of normal orientation (or its consistency) doesn't affect the shape or sampling calculations, or affect lighting calculations that take the time to determine which side of a surface is facing the viewer, so this is only an issue for quick rendering. The orientation of a mesh node's computed normal (as given by the cross-product of u and v tangent vectors) is implicitly determined by the radial ordering of the neighbors about the node; reversing the order of the neighbor list reverses the direction of the computed normal but has no other effect. In choosing an orientation for a vertex normal, we are not helped by the fact that with bordered surfaces it is not always clear which side is the "out" side, and with non-orientable surfaces like the Möbius strip a consistent choice of normal direction isn't even possible. If the surface is actually orientable, it is straightforward to maintain a consistent orientation over various mesh transformation operations, or to impose an orientation *a posteriori* by orienting a single node and then propagating the orientation to the rest of the mesh (similar to [HDD$^+$92]). For non-orientable surfaces, more careful rendering techniques must be used[1].

7.3 A programmer's view

Having exposed the inner workings of our variational substrate in some detail, it is time to see how much of that detail can be hidden. In this section we discuss a simple user/programmer model of this computational machinery used to write a simple modeler. The idea is to create and manipulate variational shape specifications as first-class objects while concealing the details of mesh-based shape approximation and mesh surgery. Our actual implementation of this low-level substrate is approximately 15,000 lines of C.

7.3.1 Objects

The basic objects available to the programmer are:

- **point**: a 3D position

[1] Any non-orientable surfaces illustrated in this document have had their normal-field discontinuities carefully moved to the back-sides.

- **c-point**: a 3D point on a curve (corresponding to a node in the p.l. curve representation).
- **curve**: a sequence of **c-points** corresponding to a smooth (curvature-continuous) curve.
- **c-region**: a continuous subset of a curve, with boundary c-points
- **s-point**: a 3D point on a surface (corresponding to a node in the surface mesh).
- **surface**: a continuous collection of s-points and a surface triangulation over them, representing a curvature-continuous surface.
- **s-curve**: an embedded surface curve (open or closed), represented as a sequence of connected s-points.
- **s-region**: a continuous subset of a surface, with boundary s-curve.

These are created and modified by various topological operations, below, and their shapes are controlled by various region controllers.

7.3.2 Topological operations

The programmer should be able to create and make controlled changes to curve and surface topology, without having to worry about maintaining the consistency of the underlying mesh representation. We provide the following topological operations, implemented in terms of the mesh constructs of Section 4.2.3 (**operation names** in bold, *parameters* in italics).

- **curve(***point list***)**: create a free-standing curve, open or closed.
- **sheet(***boundary curve***)**: given a closed boundary s-curve or free-standing curve, fill in the "hole" with a surface sheet. In the case of a surface-bounding s-curve, the sheet will meet the existing surface in a crease.
- **sweep(***c-region1, c-region2***)**: given a pair of s-curve or curve regions (or entire curves) of matching topology, create a swept surface between them, meeting any existing surface in a crease.
- **s-loop(***s-point list***)**: create an embedded surface curve.
- **burnout(***s-region***)**: destroy the surface region, leaving its boundary curves intact. If the region is a subset of some larger surface, the region's boundaries become border s-loops for that surface. This operation may actually split the surface into multiple unconnected regions (if there are unconnected regions contained within the burnout region).
- **split(***s-region***)**: split the surface along the region's boundary. This produces two independent surfaces, and clones the region boundary curve to make two independent control curves (in the case of a non-orientable surface such as a Möbius strip, there will be only one resulting surface, an effect you may have experienced if you have ever cut a paper Möbius strip lengthwise with a pair of scissors).

- **crease(**_s-region_**)**: like split, but the control curve is not cloned and instead both surfaces remain attached to the original control curve. This relaxes the continuity of the surface by introducing a crease along the curve. The operation is not defined for a non-orientable surface region.

- **smooth(**_loop_**)**: The inverse of a crease operation, this takes two surfaces sharing a boundary control curve and merges them into a single curvature-continuous surface with an internal control curve where the crease used to be.

7.3.3 Shape control and region shadowing

Shape control is organized in terms of topological regions and region controllers. Regions are created as side effects of higher-level user actions — *e.g.*, defining c-points on curves, and s-curves on surfaces (which act as region boundaries).

Interpolation constraints provide a partial specifications of curve and surface shapes, acting as a skeleton over which the topological skin will be draped (Figure 7.4). Whenever a user grabs a surface point during interaction, an appropriate constraint will be created and added to this skeleton so that the surface will be forced to follow the user's motions. A number of Higher-level shaping tools will interact with the specification by applying geometric constraints and manipulating them in coordinated ways.

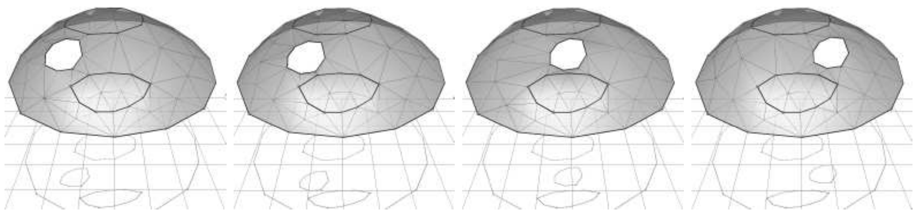
Variational control curves are free-standing variational curves attached to corresponding s-curves. A convenient way to handle such hierarchic variational shapes uses a special *region shadowing controller*. Region shadowing lets one attach p.l. surface curves (sequences of mesh edges) to free-standing p.l. control curves. It is responsible not only for copying the positions of its source nodes into those of the constrained shadowed nodes, but also for maintaining a one-to-one correspondence between nodes in the source region and the shadow region. Whenever the source region undergoes refinement, the shadow region is refined as well, to preserve the correspondence. Thus, whenever a control curve edge is split, a shadow controller must perform a split on the corresponding edge of the embedded surface curve. This has a nice effect of allowing the control curve to determine the density of *surface* sampling in the neighborhood of the curve constraint. Though we have only applied shadowing to curves, and in a degenerate sense to control points on curves (the correspondence problem is trivial), shadowing could apply equally well to surface regions, *e.g.*, to apply an "embossing stamp" to a surface.

The user-level commands to establish region controllers are:

- **set-region-shaper(**_region, shaper_**)**: shaper may be `fair` for fairness-optimization, `shadow` to constrain region to follow another region, or the programmer may build an external shape controller to be applied to mesh regions, which will be called continuously to update the region's shape along with the built-in controllers.

- **make-curve-control-pt(**_curve, vertex_**)**: Create a free-standing point, initially located at the given curve vertex, and cause the vertex to shadow it.

- **make-surf-control-curve(**_surf, vertex list_**)** Create an s-curve and a free-standing control curve that overlays it, and cause the s-curve to shadow the control curve.



A hole is interactively dragged between two fixed regions on the surface, demonstrating the incremental Delaunay triangulator at work. Global curvature minimization maintains the surface in a fair shape as it moves. The shape and position of the hole itself is determined by a free-standing PWL curve to which the surface boundary is attached.

Figure 7.1:

7.3.4 Nonparametric interpolation and sliding features

Surface features like bounded subregions and embedded curves are free to slide around relative to each other within the surface triangulation, without disrupting the mesh's global surface topology (Figure 7.1). These are the nonparametric interpolation constraints discussed in Section 2.5.3. Because mesh improvement steps are interleaved with the fairing computation, it is possible for nodes and edges to migrate across neighborhoods as a side-effect of the mesh-improvement algorithm, and thus we needn't do anything explicitly to enable this sliding behavior. This avoids a difficulty experienced by smooth patch-based surface modelers, which must maintain parametric or *material* coordinates for embedded surface features, describing the (u, v) locations of feature points on the patch. For a parametric surface, having a curve slide across the surface means adjusting its material coordinates to track its projected physical shape, and this is a messy nonlinear problem.

7.3.5 Mesh surgery revisited

Sometimes the user will want to glue together surfaces along two unrelated boundary curves — curves that have different distributions of nodes, and may not even lie atop one another prior to the gluing operation. The curves must be brought into node-to-node correspondence and superpositioned before they can be glued together. One "high road" to solving this correspondence problem casts it as a minimum edit-distance problem [CLR90], and uses a dynamic programming scheme to transform one edge loop to match the other through a sequence of edge-splits, edge-collapses, and node repositionings[SG92a].

Though we implemented a similar scheme, we subsequently found it preferable (and much simpler) to avoid the superposition problem altogether, and simply create a cylindrical blend surface that bridges the space between the two boundary curves — assuming the curves are reasonably close to one another (Figure 7.2). After attaching the blend to the boundary curves, the boundary curves become interior loops within a single unified surface, and they may be retained as control curves or freed to yield an unconstrained surface. If both curves are retained as constraint curves, they may be used to control the tightness and continuity of this blend region. On the other hand, if one or both of the curves is freed, the

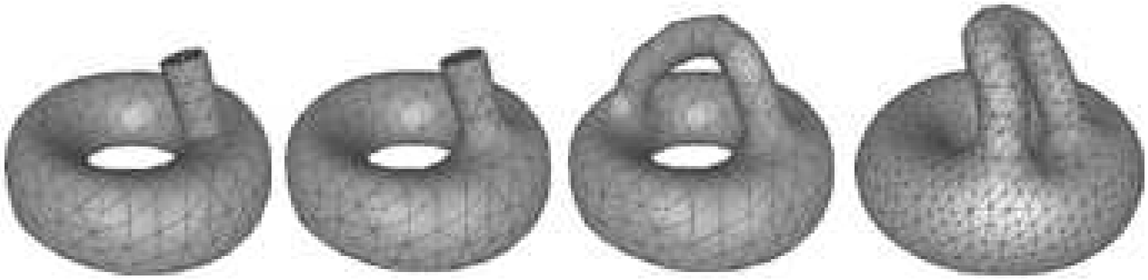


Figure 7.2: Attaching a handle: (1) one end of a cylinder is brought near the torus surface. (2) The user indicates that a *merge* is desired, and the modeler cuts a hole in the torus and attaches the cylinder with a blend surface. (3) The other side of the cylinder is similarly attached. (4) The user grabs the far handle attachment and slides it to the front, demonstrating nonparamteric interpolation constraints.

result is ultimately the same as if both curves had first been superimposed and then zipped together to join the surfaces.

7.3.6 Picking

We allow curve and surface points to be created and selected geometrically with picking operations:

- **ray-intersect-curve(***ray, curve***)**: find the point on the curve closest to the ray, locally refine the curve to place a c-point there, and return the c-point.
- **ray-intersect-surface(***ray, surface***)**: find the point on the surface closest to the ray, locally refine the surface to place a s-point there, and return the s-point.

These are implemented in the obvious way, by intersecting the ray with the face list and solving for the minimum point-edge distance.

7.4 Building on top of the substrate

Our intent is that machinery thus far described be used to manage the details of mesh maintenance and shape approximation on behalf of a higher-level modeler. This modeler in turn would offer more convenient ways of specifying and interacting with geometry. In this section we demonstrate some ways in which an external client of the substrate might extend its functionality through higher-level operations.

7.4.1 External shape tools

One approach to using our free-form surfaces combines them with standard surface shapes like generalized cylinders, spheres, etc. Our surfaces act as blends to transition between portions of these structured surfaces, but with much more topological flexibility than traditional blend surfaces.

Figure 7.3: A *Klein mug*, a variation on the famous 1-sided surface. The handle and sidewalls are controlled by cylinder tools. Variational blend surfaces join the cylinders into a globally smooth, non-orientable surface. A hole has been cut in the side to show "interior" structure.

Such structured shapes can be incorporated into a free-form model by writing region controllers that manage mesh approximations to the shapes. Given a triangulated collection of nodes (of suitable surface topology), a region controller's job is to position these nodes on its own separately-represented surface. The region's boundary points will also lie on this surface managed by the controller, so that they act as "trim-curves". The end effect is that one may punch holes in or extend simple structured shapes by trimming them and attaching variational blend surfaces smoothly along the trim curves.

Parametric curves and surfaces

The simplest region controllers are for surfaces that may be represented as functions mapping (u, v) surface coordinates to positions in $\mathcal{R}^3$, such as swept surfaces. The controller assigns a fixed (u, v) surface coordinate to each node in its region. Updating the region's shape is then a simple matter of evaluating the controller's surface function for each node. We assume that the controller will do its own mesh management — in (u, v) space — using any of a variety of standard mesh generation techniques.

Algebraic curves and surfaces

Limited kinds of algebraic curves and surfaces (Chapter 2) can also be used as region controllers. Unlike parametric surfaces, it is not clear how to implement an algebraic region controller in a completely general way, because of the intrinsic difficulty of sampling and rendering contours of algebraic functions. If the topology of the algebraic surface itself is

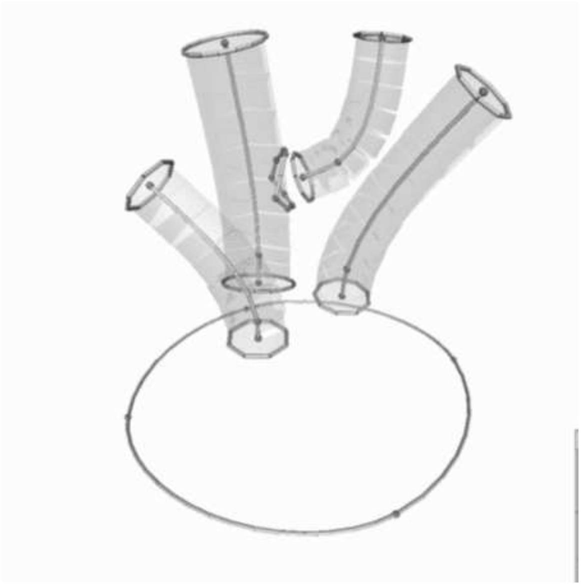


Figure 7.4: Control skeleton for a structured shape. The cylinder backbone curves, cylinder radii, cylinder attachment points and blend widths are all available to the designer as shape controls.

allowed to change interactively, as in [WH94], we would be faced with the nightmare of tracking this change and triggering mesh surgery as needed.

This didn't stop us from experimenting with a few algebraic controllers of known, fixed topology. The cylinder tool used in many of the figures is a mixture of parametric and implicit controllers. It is an implicit offset surface from a variational backbone curve (the backbone is controlled like any other variational curve). For such a restricted surface, it is possible to create an initial mesh of appropriate topology and shape, and attract the nodes to the implicit surface using a constraint technique similar to [WH94]. This works almost acceptably in practice; there are problems if the algebraic surface moves too quickly for the node position controller to track its changes differentially, and it seems advisable to implement such controllers as parametric surfaces whenever possible.

7.4.2 Deformation hierarchies

Another useful way to add structure to a free-form surface model is to group constraints and move them in coordinated ways. Deformation hierarchies may be used to this end, as in [BJWK93]. For example, if a variational curve shape is controlled by a number of constraint points, one might choose to scale, translate, or otherwise transform some or all of these control points at once in order to simplify the re-shaping of the curve. A surface model structured this way could offer the designer a small set of parameters to control the size and positioning of major components, and rely on variational curves and surfaces to keep a smooth skin stretched over the whole assembly (Figure 7.4).

7.4.3 Auto-merge

Finally, we consider an example of a high-level modeling operation that makes simultaneous changes to shape and topology using basic operations above. One of the charming behaviors of Szeliski, *et al.*'s oriented particle systems[ST92] is the way two separate particle surfaces will automatically "melt" into each other when brought in close proximity. As discussed in Chapter 2, one drawback here is that there is no explicit control over the topological change that occurs (*i.e.*, it is not clear how to bring two particle surfaces close together *without* having the merge take place).

We implemented a controlled version of this proximal merging; Figure 7.2 shows the user's view. As the two surfaces are moved close to one another, a transparent tube lights up along the line connecting their closest points, to show where an auto-merge would take place. If the user chooses to proceed, the modeler pierces each of the surfaces at its intersection with this line. Each intersection point is then "opened up" by first surrounding it with a closed s-curve, then burning out the s-curve's interior, and finally expanding the s-curve to a user-specified radius. A tubular surface is then swept between these two closed boundary curves, smoothly joining the two surfaces.

Summary

We have shown how to combine the variational specifications of Chapter 4, the mesh-based thin plate approximation scheme of Chapter 5, and the mesh maintenance scheme of Chapter 6 in an interactive surface modeler. The approach allows one to mix explicit shape controllers with implicit region fairing to make structured free-form models. A simple programming abstraction hides the details of the mesh approximation machinery in a "variational substrate" that takes tagged topological meshes as input and renders approximate shapes in real time. Our initial implementation of these ideas is a modeler that runs at interactive speeds on a Silicon Graphics Indigo class workstation, for surface models of several hundred to a thousand nodes. The example surfaces throughout this document have been produced with this modeler, and they range from 500 to 1200 nodes each. Each was created in only a few minutes by the author, using operations similar to the construction sequences in Figure 3.1 and Figure 7.2.

Chapter 8

Conclusion

In the previous chapters, we have outlined an approach to interactive modeling for fair surface shapes of arbitrary topological type. The focus has been more on the mathematical and algorithmic hurdles to be cleared than on practical application of the techniques. Nevertheless, at this point it is appropriate to compare this approach with some of the modeling approaches originally surveyed in Chapter 2. This will be followed by a discussion of general limitations of our approach, and opportunities for future research. We conclude with a summary of the work's contributions.

8.1 Comparison with other free-form surface modeling approaches

- **Polygon meshes with locally smooth skins:** As was discussed, approaches to surface design involving local skinning of point or curve networks do not produce

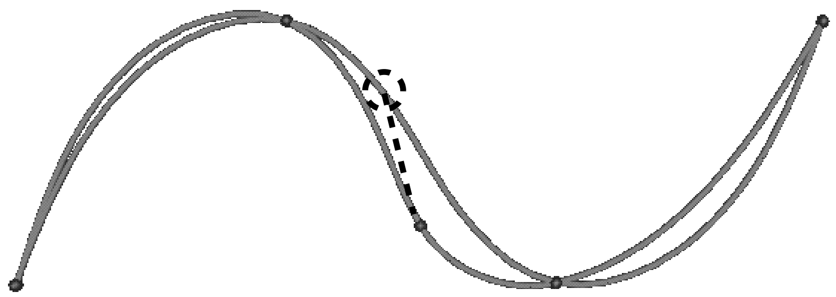


Figure 8.1: Revisiting a curve interpolation problem of Chapter 2: even though the user has grabbed and dragged a point near one of the original interpolation points, the shape remains fair. Further, it does not matter where this initial grab is made; the curve seeks the same final shape. This behavior is due to the combination of our geometric thin plate function and adaptive curve sampling.

Figure 8.2: Revisiting a surface interpolation problem of Chapter 2: even though the user has grabbed and dragged a control curve in a twisting motion as before, the mesh adapts to let the control curve slide relative to the surface, yielding a fair surface shape.

globally fair shapes. Additionally, there is no principled way to refine and unrefine such representations. Reference cannot be made to some invariant idealized shape in deciding where to place new points or reposition existing ones (the exception being the placement of new points on a subdivision surface). Our approach allows one to change mesh densities on-the-fly, while remaining faithful to an ideal variational shape.

- **Linearized thin-plate modelers:** The main drawback of previous interactive variational modelers — resulting from their use of a linearized thin-plate objective — was that surface shapes depended on an underlying surface parameterization. This parameterization easily becomes warped or stretched in the course of interactive re-shaping, and surface shapes degrade as a result. Earlier, a number of defective curve and surface shapes were shown that suffered this effect (Figure 2.7, Figure 2.9). Our way of approximating the geometric thin plate objective addresses this problem, essentially by recomputing a chordal parameterization each time a curve or surface shape changes. Illustrations of our approach's performance on these "problem" interpolations appear in Figure 8.1 and Figure 8.2.

- **Volumetric modeling:** We have spoken of variational models as *structured shapes*, because a single specification can capture an entire family of related shapes. It is instructive to compare this with unstructured, volumetric sculpting approaches. To use an analogy from 2D drawing, comparing structured and unstructured shape models is like comparing drawings produced by an object-based drawing program (*e.g.*, *MacDraw*) and a pixel-based painting program (*e.g.*, *MacPaint*), respectively. Object-based drawing programs typically do not allow one to set up dependencies between geometric elements as can be done in our structured models. For complex, highly detailed sculpted shapes (*e.g.*, a gargoyle), structured modeling may be the wrong approach: much more effort will likely be involved in designing a control-curve model than if one was to directly sculpt the object — and the resulting parameterized model may be so complex as to not be a useful representation of a family of similar objects. This is when direct volume editing (or, more likely, honest clay modeling) makes the most sense. On the other hand, there is much parameterizable structure in everyday manufactured objects. Variational design, especially when mixed with local explicit control as has been done here, is more appropriate for describing families of shapes that may have simple parametric relationships to one another while not necessarily having simple shapes or topologies.

- **Generalized sweeps and cross-sectional design:** Sweeps are appropriate when the surface topology is simply parameterized and there is one or more symmetry axis. One may then specify profile and backbone curves instead of entire surfaces. Our variational approach is more appropriate for models that have no such inherent symmetries. Then again, the generating curves in a swept surface might be best described as variational curves; and a swept surface model could be stitched into a more general variational model using the shape-copying objective, giving us the best of both worlds.

8.2 Limitations

8.2.1 Solver complexity

The most expensive computational step in our shape approximation is the large linear system that must be solved at each time-step. Even at $O(n^2)$ complexity (as discussed in Chapter 7), this will prevent model sizes from being scaled up into the thousands of nodes while maintaining interactive performance. One possible way around this is to put portions of a large, complex model to "sleep" while the user is interacting with a smaller portion. It is reasonable to expect that the extremities of such a model would not change drastically in response to modeling operations that are not nearby (when one pulls on the tail-fin of a 747, one does not expect the nose to wiggle). Disregarding nonlocal features (at least during intense local sculpting operation) is rather like putting a drop-cloth over the portions of a room that are not undergoing renovation, and would allow us to reign in the size of the linear systems that must actually be solved.

In addition to the cost of a single solver iteration, there is also the issue of global convergence: how many solver iterations are required for a surface to settle into a quiescent state after it has been perturbed in some way (*e.g.*, by control curve changes or mesh surgery)? For a very densely sampled surface such as that in Figure 7.3, the time required to reach quiescence is too large for interactive work. Recall that in our examples the user interacts with such models at a lower sampling density, and the sampling density is increased prior to final rendering. A principled approach to this would use multi-resolution sampling, so that the solver would initially solve a coarse version of the surface, then add successively finer detail using the previous coarse solution as a starting point. We experimented briefly with automatic multi-resolution reconstruction using mesh enrichment/depletion operations: whenever the user clicked a mouse, the mesh was depleted until the operation was complete, then the mesh was steadily enriched on successive solver iterations. Though this was much faster than solving the constant dense mesh, it was somewhat distracting during interactive editing to see the surface resolution changing so often. Wavelet-based surface representation schemes, which are capable of maintaining a multi-resolution sampling of the surface as an integral part of the representation, may be a better way of addressing this problem.

8.2.2 The fairness objective

As mentioned in Chapter 7, although the squared curvature objective function we use to measure surface fairness is a vast improvement over previous interactive modelers, it is not the highest-quality objective one might consider. When applied to a long, narrow tube, it causes the sides of the tube to bow in. In most other situations, the minimum is reasonable shape. But Moreton showed (using smooth patches) that even then, curvature tends to concentrate at region boundaries[Mor93]. This is perhaps counterintuitive, since a minimization of $\int \kappa^2 ds$ over a curve would seem to imply that curvature will be distributed evenly. In fact, this is only true when the curvature integration is performed over a *fixed-length* arc with suitable boundary conditions. Presumably, our curves are decreasing the magnitude of the global curvature integral by decreasing their total length, at the expense of uneven curvature distribution; surfaces behave analogously. We will consider other, higher-order fairness objectives that do not suffer this defect, in Section 8.3.

8.2.3 Self-intersection

A potentially important kind of constraint not addressed here is one that prevents surfaces from self-intersecting. This would be a nice property to have in a modeler, *e.g.*, if the surfaces were intended as boundaries of homogeneous solids. A model that depends on prevention of self-intersection in order to define its shape is unusual: a knot in a rope that has been pulled tight is an example of the kind of sliding contact constraints involved. Note that the fact that the rope is or is not knotted isn't recorded in the rope's domain topology (which is simply a cylinder), but rather is a function of its immersion in space. Unfortunately, this is a rather difficult requirement of an interactive modeler, and it is not clear how to incorporate such a constraint into our approach while maintaining interactivity.

8.2.4 The triangulated surface representation

It is perhaps a little unsatisfying that our approach to smooth surface design never actually produces a smooth surface, but only point-wise approximations. Actually, when it comes down to exporting a model, there are many situations in which a triangle mesh is a desirable representation, provided it is fine enough. Needless to say, refining a mesh to a specified export resolution is straightforward with our approach. It may be desirable follow this with a mesh simplification step[HDD+93] to obtain a more "efficient", variable density mesh rather than the uniform density mesh that would be produced by our simple procedure.

If an explicit smooth surface is needed for any reason, it is possible to fit a surface to the triangulated surface points in a post-processing step. The variational specification can be used to set up an approximation using smooth parametric patches rather than a p.l. mesh. This is exactly the computation we ruled out in Chapter 2 as being too expensive; but interactivity is not an issue here. The mesh shape can supply a starting shape for the approximating surface, and thus speed the method's convergence (coming up with good initial values is both important and nontrivial in this kind of computation). This will be a more expensive computation than the constrained point-skinning above, but should yield higher-quality results for comparable mesh resolutions.

8.2.5 The approximation procedure

If we are to consider exporting the variational specification itself as a representation of a surface shape, so that another modeler could construct its own approximate shape using its own representation, we must wonder how well these two shapes will agree. When the specification is subsequently handed off to a more accurate, non-interactive shape approximation scheme, might it be that the approximated surface is no longer where the user intended? But this is exactly where geometric constraints come into play: if the surface is supposed to be positioned in some precise way, there should be constraints in place to do this. Given that variational shapes can only be approximated, it does not seem unreasonable to offer only qualitative assurances about the behavior of faired shapes away from explicit constraints.

8.3 Future Work

Though the limitations discussed above are not likely to be relaxed by any straightforward modifications to our approach, there are other useful improvements that are more within reach.

Objective function

Using variation of curvature as our objective function, as in [MS92] would lead to even higher-quality shapes than our current curvature minimization scheme delivers. A big difficulty with implementing such a scheme over our p.l. meshes is that 3rd-derivatives are required. The finite-difference scheme cannot easily be extended to 3rd-order, because there are not generally enough neighbors to a node to resolve the additional terms. One possibility is to explicitly pull on normal vector alignments using a co-circularity objective as did Szeliski[ST91]. The difference between the directions of neighboring normal vectors acts as a kind of 2nd derivative (this goes back to the differential-geometric definition of sectional curvature). As discussed in Chapter 2, minimizing the variation between neighboring normal alignments is equivalent to minimizing a function of the 3rd derivatives over a patch.

Control curve networks

Though we formulate surface shape control using interpolated control curves, our scheme does not yet accommodate intersecting control curves. A compatibility condition[Pet91] demands that when control curves meet at a point, they must all fit a common quadratic surface form; otherwise, no there can be no smooth interpolating surface in the neighborhood of the intersection. The solution is to use a special quadratic intersection node (essentially, a surface node) to join curves at a point, much like the hub of an umbrella.

Curvature-adaptive sampling

It may be worthwhile to consider a curvature-sensitive scheme for distributing sample points across the surface. The error of our objective function integration in a neighborhood is related to the neighborhood's total curvature, and an adaptive scheme would tend to distribute this error more evenly across nodes. This requires a modification of the mesh re-sampling scheme, since the centroid-based technique does not generalize well to nonuniform spacing. We actually experimented with a curvature weighting scheme, but the results were not particularly satisfying: mesh densities didn't decrease gradually away from the high-curvature areas, but rather changed rapidly to a uniform distribution. This is to be expected, given the Laplacian's aggressive smoothing characteristics, and a different smoothing function might be better suited for solution-adaptive meshing.

Smooth surface patches

It would be interesting to try these techniques with a smooth, geometrically continuous (or approximately continuous) surface patch scheme such as subdivision surfaces or Celniker triangles[Cel90]. This would shift the local shape computations from being node-based to being a patch-based finite-element scheme. We would need to give up the use of our version of the geometric thin-plate functional (which depended on faceted parameterizations to yield a quadratic minimization problem), and consider more a general, geometric objective function. Still, it is worth asking whether one could use substantially fewer patches than nodes to obtain a given approximation accuracy, and thus reduce the overall computational burden to a comparable level.

8.4 Review of contributions

The primary contributions of this work are:

- It is the first work to address interactive, incremental design of fair free-form surfaces of arbitrary topology. The user has explicit control over the topology at all times, and builds up complex topologies from simple ones through "surface surgery."
- We developed an approximation scheme for geometric thin plate surfaces, based on triangulated surface meshes, that is suitable for use in an interactive modeler: *i.e.*, it is *speedy and robust.*
- Included as a part of our approximation approach, but useful in its own right, is a robust method of computing neighborhood parameterizations for neighborhoods of an arbitrary shape/arbitrary topology p.l. mesh.
- We developed a mesh improvement scheme for arbitrary topology surface meshes. Meshes are automatically refined to a prescribed sampling density, nodes are smoothly distributed across surfaces, and a nice triangulation is maintained. Again, the approach is speedy and robust enough for use in an interactive modeler.
- We developed an approach to "sliding" interpolation constraints, as a side-effect of our curve and surface mesh improvement schemes.

As part of this work, we built an interactive, direct manipulation surface modeler that demonstrates all of the functionality discussed above (though no one would mistake it for a full-fledged industrial design tool). The modeler itself represents a number of secondary contributions:

- It is the first modeler that uses variational curve and surface specifications as its basic shape representation (built on top of the approximation machinery above).
- It is the first *variational* modeler that allows a designer to build up surface topology in terms of "surgical operations" on the 3D surface.

Summary

In this work we have developed a new approach to designing curves and free-form surfaces on a computer. Unlike related modeling approaches based on the notions of character lines and fair surfaces, we allow surfaces to be interactively cut apart and smoothly joined along arbitrary curves, so that complex shapes and topologies can be built up from simpler ones. Unlimited amounts of detail may be added to a model simply by indicating more control points and curves. Finally, portions of a curve or surface may be made to copy externally controlled *shape tools*. This allows one to mix free-form and structured shapes within a single composite surface model of arbitrary topology.

References

[Aki84] Hiroshi Akima. On estimating partial derivatives for bivariate interpolation of scattered data. *Rocky Mountain Journal of Mathematics*, 14(1), 1984.

[Ale30] James Alexander. The combinatorial theory of complexes. *Ann. Math*, 31:292–320, 1930.

[AMR83] J. Alander, M. Mantyla, and T. Rantanen. Solid modeling with parametric surfaces. In P. J. W. ten Hagen, editor, *Eurographics '83*, pages 71–83. North-Holland, 1983.

[Baj92] Chadrajit L. Bajaj. Smoothing polyhedra using implicit algebraic splines. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[Bar90] Timothy Barth. Higher order solution of the euler equations on unstructured grids using quadratic reconstruction. In *28th Aerospace Sciences Meeting*. AIAA-90-0013, 1990.

[Bar94] Ken Bartels. Rewarming hypothermic animals with microwaves. *Veterinary Forum*, pages 28–29, March 1994.

[Bau75] B. Baumgart. A polyhedron representation for computer vision. *AFIPS Nat. Conf. Proc.*, 44:589–596, 1975.

[BB89] Richard H. Bartels and John C. Beatty. A technique for the direct manipulation of spline curves. In *Proceedings, Graphics Interface*, 1989.

[BCGH92] Alan H. Barr, Bena Currin, Steven Gabriel, and John F. Hughes. Smooth interpolation of orientation with angular velocity constraints using quaternions. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[BCO81] Eric Becker, Graham Carey, and Tinsley Oden. *Finite Elements, an introduction*. Prentice-Hall, 1981.

[BE92] Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. In F.K. Hwang and D.Z. Du, editors, *Computing in Euclidean Geometry*. World Scientific, 1992. also appears as XEROX PARC report CSL-92-1, March 1992.

[BGW88] Chanderjit Bajaj, Thomas Garrity, and Joe Warren. On the applications of multi-equational resultants. COMP TR88-83, Department of Computer Science, Rice University, November 1988.

[BJWK93] D.L. Bonner, M.J. Jakiela, M. Watanabe, and N. Kishi. Pseudoedge: nonintersected parametric quilt modeling of multiply connected objects. *Computer Aided Design*, 25(7):438–452, July 1993.

[Bli82] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1, July 1982.

[Blo94] Jules Bloomenthal. An implicit surface polygonizer. In Paul Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[Boo79] F.L. Bookstein. Fitting conic sections to scattered data. *Computer Graphics and Image Processing*, 9:56–71, 1979.

[Boy91] Carl B. Boyer. *A history of mathematics*. John Wiley and Sons, Inc., 1991.

[BS82] J.U. Brackbill and J.S. Saltzmann. Adaptive zoning for singular problems in two dimensions. *J. Comp. Phys.*, 46:342–368, 1982.

[BS91] M. Boyer and N.F. Stewart. Modeling spaces for toleranced objects. *Intl. J. Robotics Research*, 1991.

[Cav74] J. C. Cavendish. Automatic triangulation of arbitrary planar domains for the finite element method. *Int. J. for Numerical Methods in Engineering*, 8:679–696, 1974.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[Cel90] George Celniker. *ShapeWright: Finite Element Based Free-Form Shape Design.* PhD thesis, Department of Mechanical Engineering, MIT, 1990.

[CG91] George Celniker and Dave Gossard. Deformable curve and surface finite elements for free-form shape design. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 257–266, July 1991.

[CGP93] J.M. Chen, E. L. Gursoz, and F.B. Prinz. Integration of parametric geometry and non-manifold topology in geometric modeling. In J. Rossignac, J. Turner, and G. Allen, editors, *Proceedings, 2nd ACM/IEEE Symposium on Solid Modeling and Applications*, pages 53–64, 1993.

[CH37] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume I. Wiley, 1937.

[Che93] Paul Chew. Guaranteed quality mesh generation for curved surfaces. In *Proceedings of the ACM Symposium on Computational Geometry*, 1993.

[CK83] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. *Computer Graphics*, 17, July 1983. (Proceedings Siggraph '83).

[CLR90] Thomas Corman, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[CW92] George Celniker and William Welch. Linear constraints for nonuniform B-spline surfaces. In *Proceedings, Symposium on Interactive 3D Graphics*, 1992.

[DC91] B.R. Donald and D.R. Chang. On the complexity of computing the homology type of a triangulation. In *Proc. 32nd Annual Symposium on Foundations of Computer Science*, pages 650–661. IEEE, IEEE Comput. Soc. Press, October 1991.

[DE93] C.J.A. Deflinado and H. Edelsbrunner. Incremental algorithm for betti numbers of simplicial complexes. Technical Report UIUCDCS-R-1787, U. Ill. at Urbana-Champaign, January 1993.

[DeR90] Tony D. DeRose. Necessary and sufficient conditions for tangent plane continuity of Bezier surfaces. *Computer Aided Geometric Design*, 7:43–55, 1990.

[DFFP85] L. De Floriani, B. Falcidieno, and C. Pienovi. Delaunay-based representation of surfaces defined over arbitrarily shaped domains. *Computer Vision, Graphics, and Image Processing*, 32:127–140, 1985.

[dFGTV92] Luis Henrique de Figueiredo, Jonas de Miranda Gomes, Demetri Terzopoulous, and Luiz Velho. Physically based methods for polygonization of implicit surfaces. In *Proceedings, Graphics Interface '92*, pages 250–257, 1992.

[DGR93] Nira Dyn, Ifat Goren, and Shmuel Rippa. Transforming triangulations in polygonal domains. *Computer Aided Geometric Design*, 10:531–536, 1993.

[Doo78] D. Doo. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design*, pages 157–165, 1978.

[DWS93] Herve Delingette, Yasuhiko Watanabe, and Yasuhito Suenaga. Simplex based animation. In N.M. Thalman and D. Thalman, editors, *Models and Techniques in Computer Animation.* Springer-Verlag, 1993.

[EM94] H Edelsbrunner and E P Mücke. Three-dimensional alpha shapes. *Transactions on Graphics*, 13(1):43–72, 1994.

[Epp76] M. Eppstein. On the influence of parameterization in parametric interpolation. *SIAM J. Numer. Anal.*, 13:261–268, 1976.

[Far90] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1990.

[FG92] L. Fang and D. C. Gossard. Recostruction of smooth parametric surfaces from unorganized data points. In *Curves and surfaces in computer vision and graphics III*, volume 1830. SPIE, 1992.

[Fie84] D. A. Field. Laplacian smoothing and delaunay triangulations. *Comm. Appl. Numer. Methods*, 4:709–712, 1984.

[FN90] Richard Franke and Gregory Nielson. Scattered data interpolation and applications: a tutorial and survey. In Hans Hagen and Dieter Roller, editors, *Geometric Modeling: methods and applications*. Springer-Verlag, 1990.

[For92] Steve Fortune. Voronoi diagrams and Delaunay triangulations. In F. K. Hwang and D.Z. Zu, editors, *Computing in Euclidean Geometry*. World Scientific, 1992.

[Fow92] Barry Fowler. Geometric manipulation of tensor-product surfaces. In *Proceedings, Symposium on Interactive 3D Graphics*, 1992.

[Fra82] Richard Franke. Scattered data interpolation: Tests of some methods. *Mathematics of Computation*, 38(157), January 1982.

[FRC92] Helaman Ferguson, Alyn Rockwood, and Jordan Cox. Topological design of sculptured surfaces. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[FW60] G. Forsythe and W. Wasow. *Finite Difference Methods for Partial Differential Equations*, chapter 19, pages 179–182. John Wiley and Sons, 1960.

[GH91] Tinsley A. Galyean and John F. Hughes. Sculpting: An interactive volumetric modeling technique. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 267–274, July 1991.

[Gle94] Michael Gleicher. *A Differential Approach to Graphical Interaction*. PhD thesis, Carnegie Mellon University, November 1994.

[GMW81] Phillip Gill, Walter Murray, and Margaret Wright. *Practical Optimization*. Academic Press, New York, NY, 1981.

[GP74] Victor Guillemin and Alan Pollack. *Differential Topology*. Prentice-Hall, 1974.

[GVL89] Gene Golub and Charles Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1989.

[GW91] Michael Gleicher and Andrew Witkin. Differential manipulation. In *Graphics Interface '91*, June 1991.

[Ham93] B. Hamann. Curvature approximation for triangulated surfaces. *Computing/Suppl*, 8:139–153, 1993.

[HB91a] H. Hagen and G. Bonneau. Variatinal design of smooth rational Bezier curves. *Computer Aided Geometric Design*, 8(5):393–400, 1991.

[HB91b] H. Hagen and G.P. Bonneau. Variational design of smooth ratinoal bezier curves. *Computer Aided Geometric Design*, 8:393–399, 1991.

[HB93] H. Hagen Hahmann and G.-P. Bonneau. Variational surface design and surface interrogation. In R. J. Hubbold and R. Juan, editors, *Eurographics '93*, pages 448–459, Oxford, UK, 1993. Eurographics, Blackwell Publishers.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):55–64, July 1992. (Proceedings Siggraph '92).

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of Siggraph 93*, July 1993.

[HDD+94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of Siggraph 94*, July 1994.

[HG94] Paul Heckbert and Michael Garland. Multiresolution modeling for fast rendering. In *Proceedings, Graphics Interface '94*, pages 43–50, 1994.

[HH87] C. Hoffmann and J. Hopcroft. The potential method for blending surfaces and corners. In G. Farin, editor, *Geometric Modelling*. SIAM, 1987.

[HKD93] Mark Halstead, Michel Kass, and Tony DeRose. Efficient, fair interpolation using catmull-clark surfaces. In *Proceedings of Siggraph 93*, 1993.

[Hof82] Hoffman. Relationship between truncation errors of centered finite difference approximation on uniform and nonuniform meshes. *J. Comp. Phys.*, 42:469–474, 1982.

[Hos88] J. Hoschek. Spline approximation of ofset curves. *Computer Aided Geometric Design*, 5:33–40, 1988.

[HS90] H. Hagen and G. Schulze. Variational principles in curve and surface design. In Hans Hagen and Dieter Roller, editors, *Geometric Modeling: methods and applications*. Springer-Verlag, 1990.

[HSW89] J. Hoschek, F. Schneider, and P. Wassum. Optimal approximate conversion of spline surfaces. *Computer Aided Geometric Design*, 6:293–306, 1989.

[Huf75] David A. Huffman. Curvature and creases: a primer on paper. In *Proc. Conf. Computer Graphics, Pattern Recognition, and Data Structures*. IEEE, May 1975.

[Jän84] Jänich. *Topology*. Springer-Verlag, 1984.

[Kal93] Michael Kallay. Constrained optimization in surface design. In *Modeling in Computer Graphics*. Springer Verlag, 1993.

[Koe90] Jan. J. Koenderink. *Solid Shape*. MIT Press, 1990.

[KR90] Michael Kallay and Bahram Ravani. Optimal twist vectors as a tool for interpolating a network of curves with a minimum energy surface. *Computer Aided Geometric Design*, 7:465–473, 1990.

[KWT87] Michael Kass, Andrew Witkin, and Dimetri Terzopoulos. Snakes: Active contour models. *International Journal Computer Vision*, 1(4), 1987.

[Lan56] Cornelius Lanczos. *Applied Analysis*. Dover, 1956.

[Law77] C.L. Lawson. Software for C^1 surface interpolation. In J. R. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, 1977.

[LD90] Charles Loop and Tony DeRose. Generalized B-spline surfaces of arbitrary topology. *Computer Graphics*, 24(4), 1990. (Proceedings Siggraph '90).

[LGL81] V. C. Lin, D. C. Gossard, and R. A. Light. Variational geometry in computer-aided design. *Computer Graphics (SIGGRAPH '81 Proceedings)*, 15(3):171–177, August 1981.

[LH74] C.L. Lawson and R. J. Hanson. *Solving Least Squares Problems.* Prentice-Hall, 1974.

[LMD92] Michael Lounsbery, Stephen Mann, and Tony DeRose. Parametric surface interpolation. *IEEE Computer Graphics & Applications*, September 1992.

[Loo94] Charles Loop. A G^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11:303–330, 1994.

[LP88] N. J. Lott and D. I. Pullin. Method for fairing b-spline surfaces. *Computer-Aided Design*, 20(10), 1988.

[Mal89] Jean-Laurent Mallet. Discrete smooth interpolation. *ACM Transactions on Graphics*, 8(2):121–144, 1989.

[Män83] M. Mäntylä. Topological analysis of polygon meshes. *Comput. Aided Des.*, 15:228–234, July 1983.

[Mas77] William S. Massey. *Algebraic topology, an introduction.* Springer-Verlag, 1977.

[Max54] James Clerk Maxwell. On the transformation of surfaces by bending. *Transactions of the Cambridge Philosophical Society*, 9:455–469, 1854. (Reprinted, 1890, in *The Scientific Papers of J.C.Maxwell, Cambridge University Press).*

[Meh74] E. Mehlum. Nonlinear splines. In R. E. Barnhill and R. F. Reisenfeld, editors, *Computer Aided Geometric Design.* Academic Press, 1974.

[MLL+92] Steve Mann, Charles Loop, Michael Lounsbery, D. Meyers, J. Painter, Tony Derose, and K. Sloan. A survey of parametric scattered data fitting using triangular interpolants. In H. Hagen, editor, *Curve and Surface Modeling.* SIAM, 1992.

[Mor93] Henry P. Moreton. *Minimum Curvature Variation Curves, Networks, and Surfaces for Fair Free-form Shape Design.* PhD thesis, University of California, Berkeley, 1993.

[MPR+94] R. Merz, F. B. Prinz, K. Ramaswami, M. Terk, and L. E. Weiss. Shape deposition manufacturing. In *Proceedings of the Solid Freeform Fabrication Symposium.* U.T. Austin, 1994.

[MS78] B.A. Murtagh and M.A. Saunders. Large-scale linearly constrained optimization. *Math. Prog.*, 14:41–72, 1978.

[MS85] A.E. Middleditch and K.H. Sears. Blend surfaces for set theoretic volume modeling systems. *Computer graphics*, 19(3):161–170, 1985.

[MS92] Henry Moreton and Carlo Séquin. Functional minimization for fair surface design. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[MT78] Wayne Mastin and Joe Thompson. Elliptic systems and numerical transformations. *J. Math. Anal. and Appl.*, 62(52), 1978.

[NF83] Gregory Nielson and Richard Franke. Surface construction based on triangulation. In Robert Barnhill and Wolfgang Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 157–162. North-Holland, 1983.

[Nie74] Gregory Nielson. Some piecewise polynomial alternatives to splines under tension. In R.E Barnhill and Risenfeld R.F., editors, *Computer Aided Geometric Design*, pages 209–235. Academic Press, 1974.

[Nie87] Gregory Nielson. A transfinite, visually continuous, triangular interpolant. In Gerald Farin, editor, *Geometric Modelling*, pages 235–246. SIAM, 1987.

[Nie88] Gregory Nielson. Interactive surface design using triangular network splines. In *Proc. 3rd International Conf. Enginerring Graphics and Descriptive Geometry*, volume 2, pages 70–77. ASEE, 1988.

[NLL90] Horst Nowacki, Dingyuan Liu, and Xinmin Lu. Fairing bezier curves with constraints. *Computer Aided Geometric Design*, 7:43–55, 1990.

[NR83] Horst Nowacki and Dirk Reese. Design and fairing of ship surfaces. In Robert E. Barnhill and Wolfgang Boehm, editors, *Surfaces in Computer-Aided Geometric Design*, pages 121–134. North-Holland, 1983.

[O'N66] Barrett O'Neill. *Elementary Differential Geometry.* Academic Press, 1966.

[PBCF93] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(3):56–102, January 1993.

[Pet91] Jörg Peters. Smooth interpolation of a mesh of curves. *Constructive Approximation*, 7:221–246, 1991.

[Pot91] Helmut Pottmann. Scattered data interpolation based on generalized minimum norm networks. *Constructive Approximation*, 7:247–256, 1991.

[Pra87] Vaughan Pratt. Direct least-squares fitting of algebraic surfaces. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):145–152, July 1987.

[PS83] M. Plass and M. Stone. Curve fitting with piecewise parametric cubics. *Computer Graphics*, 7:229–239, 1983.

[PTVF94] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C.* Cambridge University Press, 1994.

[RA82] M. M. Rai and D. A. Anderson. Application of adaptive grids to fluid-flow problems with asymptotic solutions. *AIAA J.*, 20:496–502, 1982.

[RC86] A. A. G. Requicha and S. C. Chan. Representation of geometric features, tolerances, and attributes in solid modelers based on constructive geometry. *IEEE J. Robotics and Autom.*, RA-26(3):156–166, September 1986.

[Req80] A. A. G. Requicha. Representations for rigid solids: theory, methods, and systems. *Computing Surveys*, 12(4):437–64, Dec 1980.

[RF89] D. F. Rogers and N. G Fog. Constrained b-spline curve and surface fitting. *Computer Aided Design*, 21(10), December 1989.

[Rie89] R.F. Riesenfeld. Design tools for shaping spline models. In Tom Lyche and Larry Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 499–519. Academic Press, 1989.

[RIKM93] Mervi Ranta, Masatomo Inui, Fumihiko Kimura, and Martti Mäntylä. Cut and paste based modeling with boundary features. In J. Rossignac, J. Turner, and G. Allen, editors, *Proceedings, 2nd ACM/IEEE Symposium on Solid Modeling and Applications*, pages 23–34. ACM Press, 1993.

[Roc89] Alyn Rockwood. The displacement method for implicit blending surfaces in solid models. *ACM Transactions on Graphics*, 8(4):279–297, 1989.

[RV82] A. A. G. Requicha and H. B. Voelcker. Solid modeling: a historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9–24, 1982.

[SBG93] K. Shimada, B. Balents, and D.C. Gossard. Automatic mesh reconstruction for feature-based sculpting of deformable surfaces. *Geometric Modeling for Product Realization. IFIP TC5/WG5.2, Working Conference on Geometric Modeling; Rensselaerville, NY, USA; IFIP Transactions B (Applications in Technology)*, B-8:165–188, October 1993.

[Sch66] D.G. Schweikert. An interpolation curve using a spline in tension. *Journal of Math and Phys.*, 45:312–317, 1966.

[SG92a] Thomas Sederberg and Eugene Greenwood. A physically based approach to 2D shape blending. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[SG92b] K. Shimada and D.C. Gossard. Computational methods for physically-based fe mesh generation. *Human Aspects in Computer Integrated Manufacturing. IFIP TC5/WG 5. 3 Eighth International PROLAMAT Conference. Man in CIM; Tokyo; IFIP Transactions B (Applications in Technology); vol.B-3*, B-3:411–420, June 1992.

[She68] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings 23rd ACM National Conference*, pages 517–523, 1968.

[She94] Jonathan Richard Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, March 1994. Available by anonymous FTP to WARP.CS.CMU.EDU (128.2.209.103) as quake-papers/painless-conjugate-gradient.ps.

[SK91] Yoshiuki Shiroma and Yukinori Kakazu. A generalized sweeping method for csg modeling. In J. Rossignac and J. Turner, editors, *Proceedings, 1st ACM/IEEE Symposium on Solid Modeling and Applications*, pages 149–157. ACM Press, 1991.

[SK92] John Snyder and Jim Kajiya. Generative modeling: a symbolic system for geometric modeling. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[SM91] Biplab Sarkar and Chia-Hsiang Menq. Parameter optimization in approximating curves and surfaces to measurement data. *Computer Aided Geometric Design*, 8:267–290, 1991.

[Spi79a] Michael Spivak. *A Comprehensive Introduction to Differential Geometry*, chapter II:3, page 99. Publish or Perish, Inc., 1979.

[Spi79b] Michael Spivak. *A Comprehensive Introduction to Differential Geometry.* Publish or Perish, Inc., 1979.

[Spi79c] Michael Spivak. *A Comprehensive Introduction to Differential Geometry, Volume 4.* Publish or Perish, Inc., 1979.

[SS87] L. Shirman and C. H. Séquin. Local surface interpolation with Bezier patches. *Computer Aided Geometric Design*, 4:279–295, 1987.

[ST91] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. Technical Report 91/14, Digital Equipment Corporation, Cambridge Research Lab, December 1991.

[ST92] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[Ste84] Sarah E. Stead. Estimation of gradients from scattered data. *Rocky Mountain Journal of Mathematics*, 14(1), 1984.

[Str86] Gilbert Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, 1986.

[STT93] R. Szeliski, D. Tonnesen, and D. Terzopoulos. Modeling surfaces of arbitrary topology with dynamic particles. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'93)*, pages 82–87, New York, New York, June 1993.

[SW92] Ernest Stokely and Shang You Wu. Surface parameterization and curvature measurement of arbitrary 3-D objects: Five practical methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(8), August 1992.

[SZ90] P. T. Sander and S. W. Zucker. Inferring surface trace and differential structure from 3-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):833–854, September 1990.

[SZL93] William Schroeder, Jonathan Zarge, and William Lorensen. Decimation of triangle meshes. In *Proceedings of Siggraph 93*, July 1993.

[Ter86] D. Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-8:413–424, 1986.

[Tho85] Joe F. Thompson. A survey of dynamically-adaptive grids in the numerical solution of partial differential equations. *Applied Numerical Mathematics*, 1:3–27, 1985.

[TL25] S. Timoshenko and J. M. Lessells. *Applied Elasticity*, chapter 10, page 270. Westinghouse Technical Night School Press, East Pittsburgh, PA, 1925.

[TM83] Joe F. Thompson and Wayne Mastin. order of difference expressions of curviliear coordinate systems. In *Advances in grid generation, ASME Fluids Engineering Conference*, 1983.

[Tru83] C. Truesdell. The influence of elasticity on analysis: the classic heritage. *Bulletin of the AMS*, 9(3):293–310, 1983.

[TTSC91] H. Toriya, T. Takamura, T. Satoh, and H. Chiyokura. Boolean operations for solids with free-form surfaces through polyhedral approximation. *Visual Computer*, 7:87–96, 1991. polyhedral approximation to intersections, then re-skin surfaces to meet smoothed approximate curve. Booleans are reversable (see toriya-1986).

[Tur91] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):289–298, July 1991.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics*, 26(2):55–64, July 1992. (Proceedings Siggraph '92).

[TWM85] J.F. Thompson, Z.U.A. Warsi, and C.W. Mastin. *Numerical Grid Generation: Foundations and Applications*. North-Holland, 1985.

[War86] Z.U.A. Warsi. *J. Comput. Phys.*, 64(82), 1986.

[Wei85] K. Weiler. Edge-based data structured for solid modeling in curved-surface modeling environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.

[WGW90] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *Computer Graphics*, 24(2):11–21, March 1990. (Proceedings, 1990 Symposium on Interactive 3D Graphics).

[WH94] Andrew Witkin and Paul Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings SIGGRAPH '94*, July 1994.

[WT90] Z.U.A. Warsi and Joe. F. Thompson. Application of variational methods in the fixed and adaptive grid generation. *Comp. Math. Applic.*, 19(8):31–41, 1990.

[WW92] William Welch and Andrew Witkin. Variational surface modeling. *Computer Graphics*, 26(2), July 1992. (Proceedings Siggraph '92).

[WW94] William Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. *Proceedings Siggraph '94*, July 1994.

[ZM83] O.C. Zienkiewicz and K. Morgan. *Finite Elements and Approximation.* John Wiley and Sons, 1983.