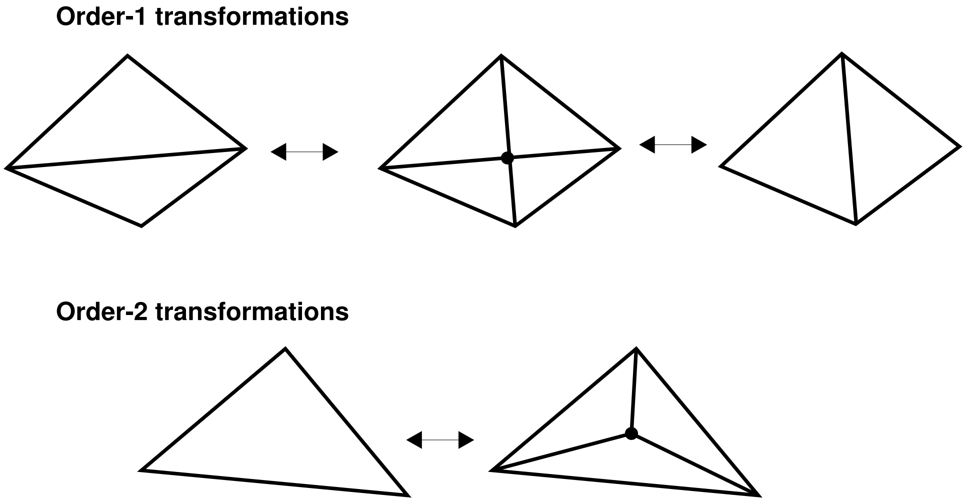

Figure 2.17: The Alexander moves. The order-1 moves operate on edges, splitting them to add nodes and collapsing them to delete nodes. The order-2 moves operate on faces, splitting them to add nodes or clipping off tetrahedral "corners" to delete nodes. The order-1 moves are sufficient to transform between any two triangulations of the same surface.

Chapter 3

Overview

This chapter gives an overview of our approach to free-form shape design, beginning with the user's view. The user is presented with a very simple model for the way surfaces behave: they may be pinned down at arbitrary points and along curves, all the while maintaining globally fair shapes or locally copying externally controlled tool shapes. Detail may be added without limit, through the accumulation of additional control curves and shape tools. Additionally, surfaces may be cut up and smoothly pasted together along arbitrary curves, so that complex topologies may be built up from simpler ones.

We start with a construction example that shows what it is like to design a surface using these tools. We then show how such curve and surface behavior can be precisely characterized in terms of a simply formulated (if not simply solved) optimization problem. User-defined control points and curves act as geometric constraints on the possible shapes; the automatic fairing and shape-copying behaviors are then realized by optimizing the shapes subject to these geometric constraints. Shapes defined this way are sometimes called *variational* shapes [HB91a, HB93], because the resulting optimization problems are properly stated using the calculus of variations[CH37].

One of the limitations of such a variational modeling approach is that there is generally no way to explicitly solve for optimal curve or surface shapes. This precludes operating directly on exact, explicit representations of the variational shapes. Our modeler will instead construct approximations to the ideal shapes, continually updating the approximation as the user interacts with the model. In this work, piecewise-linear (p.l.) curves and triangulated surface meshes are used to build approximations, instead of the smooth patches used in previous work, because they simplify certain aspects of the calculations and allow approximations to be computed at interactive speeds.

Even though the user is only presented with approximate renderings of the shape under construction, the user will be able to create and manipulate these variational shapes directly and unambiguously, regardless of the coarseness of the approximation. This is because each action by the user is interpreted as operating on the variational specification, not the particular approximation that is being displayed.

Ultimately, this representational and approximation machinery will be treated as a "black box". This will allow us to build an interactive modeler that operates on variational curves and surfaces as its basic shape representation, much as a conventional modeler might

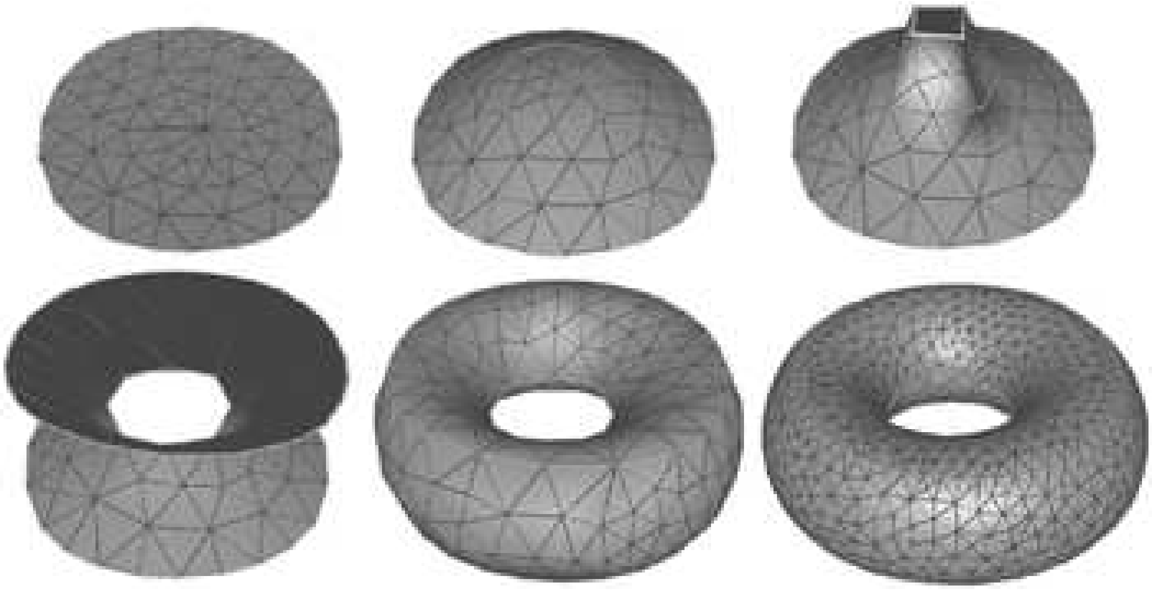

Figure 3.1: a) A closed curve is created and skinned to make a disc. b) Two closed curves are drawn on the disc and elevated c) A hole is cut in center of disc and the new boundary curve elevated. d) The upper curve is expanded to match the lower. e-f) Two boundary curves are skinned to make a single toroidal surface passing through the three control curves.

operate on B-splines or Bezier patches. Unlike conventional modelers, this approach allows one to create and interact with surfaces of unrestricted, mutable topology; add arbitrary amounts of detail; and incorporate a wide variety of convenient shape controls into a single structured free-form shape.

3.1 The user's view

A simple construction example will demonstrate the user's view of this approach to creating free-form shapes. In Figure 3.1 a torus is built in a series of simple steps. These are perhaps not the most straightforward set of steps for specifying such a shape, but the point of this example is to exhibit a number of useful tools during the intermediate stages.

In the first frame, four *control points* have been placed in roughly a square, and then a curve created that passes smoothly through them. The designer may re-shape this curve by moving the original control points, or by grabbing and re-shaping the curve at arbitrary points in-between (which then become new control points). In the next frame, a surface has been created that uses this curve as its boundary, yielding a disc. The disc's shape is thus indirectly controlled by the control points that shape its boundary curve.

In frames (c–d), a pair of curves are drawn on the surface — again, by defining control points and connecting them with smooth closed curves, but this time requiring that the curves be embedded in the surface. When the designer moves or re-shapes these *control curves*, the surface follows, always assuming a smooth shape that passes through them. In frame (e), the interior of the inner curve has been "burned out" to turn the disc into an annulus with the inner curve now serving as a boundary. This boundary is subsequently re-shaped (f), and then it and the original boundary are "skinned" with a new piece of surface that is smoothly joined to the original, creating a closed toroidal surface (g).

3.2 Shape design as functional minimization

In the previous example we spoke of curves that pass smoothly through control points, and surfaces that pass smoothly through control curves, or whose shapes copy parameterized shape tools. Such descriptions can be made mathematically precise by interpreting them as specifications for a shape optimization problem. For instance, the curve in Figure 3.1 is constrained to pass through its four fixed control points, in a specified order. Subject to these geometric and topological *constraints*, it takes on a shape that minimizes an *fairness function* (Section 2.5.2). The fairness function measures total curvature, and minimizing it causes the curve to iron out undesirable bulges or wiggles as it redistributes its curvature over its length. Similarly for surfaces: the surface in Figure 3.1 is constrained to pass through three control curves. Subject to this geometric constraint, and the topological constraint that it remain a torus, it also minimizes a curvature-based objective function to give it a fair shape.

This is an extremely concise way of describing a wide range of free-form shapes. For the curve above, a handful of control point positions, their topological ordering , and a curvature objective function are enough to completely determine the shape. Similarly for the surface, a handful of control curves, a given surface topology, and a surface objective function determine its shape everywhere. This is very different from what is done in conventional curve and surface modelers today, where free-form shapes are described as collections of B-splines, Bezier patches, or other explicit forms. Instead, these shapes are described *implicitly* as the solutions of variational optimizations. This abstracts away from the details of any particular surface representation, and makes it trivial to augment a shape with additional controls. They are simply stated as additional objective and constraint terms contributing to the variational form.

Our approach to representing and operating on variational specifications is developed in Chapter 4 and Chapter 7 (this work is also described in [WW94]). Briefly, a triangulated surface mesh records the surface topology, while sequences of connected edges record curve topology. Topological features within it — embedded curves and bounded regions — are each tagged with a "region controller" telling whether to copy the region's shape from some other piece of control geometry or to solve for a region shape that blends smoothly with neighboring regions. Variational specifications will be built up incrementally and interactively by the user. The construction example above indicates some of the ways in which natural operations by the user can provide the information needed to construct and modify a variational specification; more examples of this kind of interactive specification are discussed below and in Chapter 7.

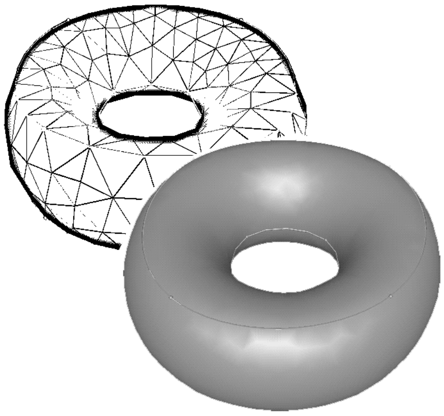

Figure 3.2: Triangulated meshes, augmented with static information about topological regions, geometric constraints and objective functions, serve as our representation of smooth variational shapes.

3.3 Approximating variational shapes

Although it is easy to pose variational problems that characterize free-form shapes, solving them is another matter. In such problems, one is solving for an optimal *function* (*e.g.*, a position function telling where the surface is), rather than a single numerical value or discrete set of values. Techniques from the calculus of variations occasionally allow one to find explicit solutions in terms of special functions[CH37], but only for limited classes of objective functions over geometrically simple domains. Things are almost never this simple for the kinds of shape objective functions, constraints, and unrestricted domain topologies considered here. Even when optima are known to exist, it will not generally be possible to find explicit functional forms for them (Section 2.5.1).

3.3.1 Pointwise approximation

Because we cannot directly solve for the optimal shapes, we will instead approximate them using an explicit surface representation. Given the variational specification above, an approximation could be set up using any of a variety of smooth surface representations — piecewise smooth polynomial patches, subdivision surfaces, *etc.*, as has been done in previous work. The approach to be described, instead of using a smooth representation, uses the same triangulated surface meshes that record curve and surface topology, by associating a 3D position with each node of the triangulation and thereby immersing the mesh as a piecewise linear surface.

A fair amount of machinery goes towards computing node positions to approximate variational shapes at interactive speeds, despite (or sometimes because of) the simplicity of the underlying representation. At the lowest level, one must be able to compute over the mesh as if it were a sampling of some underlying smooth surface. To do this, a generalized finite-difference scheme is used, in which a truncated Taylor series is fitted to each nodal neighborhood in the mesh. This allows surface derivatives to be computed at each of the nodes. Given this local reconstruction scheme, approximate solutions to smooth variational minimizations may then be computed by evaluating the integrals numerically at mesh nodes and solving for shapes that directly minimize this mesh objective.

3.3.2 Maintaining a quality mesh

As will be seen in Chapter 5, the shape objective functions used are really only capable of moving surface nodes in directions normal to the fitted surface to improve the shape. They have nothing useful to say about the distribution of nodes relative to each other over the triangulated surface. As it happens, this distribution is important for the accuracy and stability of the shape approximation calculations, because the triangulated surface essentially serves as the computational mesh over which the generalized finite-difference scheme operates. This leads us to consider the problem of maintaining a uniform nodal distribution as the surface shape changes. This is posed as a variational minimization which will slide nodes around on the triangulated surface, to optimize the quality of the computational mesh. In addition to controlling the relative nodal distribution, the absolute nodal density will be regulated as surface areas grow or shrink, using an automatic refinement procedure. To ensure that the mesh doesn't contain "skinny" triangles, dynamic surface Delaunay triangulation scheme will be used, so that a quality surface triangulation will be present at all times as surface shape and topology changes. Incorporating such a re-triangulator into the modeler affords an unrelated but important benefit: because of the way the mesh connectivity adapts as shape changes, surface "features" are free to slide around relative to each other within the mesh (Section 7.3.4). This would allow, for example, a designer of an automobile hood to slide an air intake scoop around on the hood surface to adjust its position.

3.4 Modeling with variational shapes

The variational tools described here will be much more convenient for the designer of free-form curve and surface shapes than using fixed representation parameters like B-spline control points. The designer can create any number of control points (and control curves) and place them anywhere. Surfaces may be cut apart and stitched together into more complex global topologies. Nevertheless, even with this approach there are modeling operations that might be conceptually simple from a designer's point of view but will in fact require a coordinated set of changes to the underlying variational specification. This makes it useful to consider higher-level modeling operations expressed as simple operations on the lower-level variational surfaces. Having already developed the machinery to approximate such surfaces, we can safely hide these details in a computational black-box and use variational shape specification as the basic representation on which to build a free-form surface modeler.

In Chapter 7, we consider a number of basic modeling operations cast in terms of this variational substrate. For example, we discuss how a designer might go about specifying changes to surface topology in the course of constructing a model. The handle-attachment in Figure 7.2 is an example. The conceptually simple operation of merging the two surfaces will be automated by the modeler as a series of operations that result in an appropriately

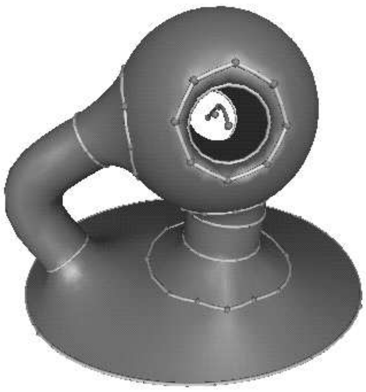

Figure 3.3: A model that mixes variational and explicitly represented shapes. A sphere tool controls a portion of the surface, to which supporting handles have been attached, and through which a hole has been drilled.

shaped hole being cut in the torus, and a blending skirt being added to join the torus and cylinder at the hole.

The cylinder tool itself demonstrates another important feature of this approach: the mixing of variational and explicit surfaces in a single model. The cylinder's shape is not defined through functional minimization, but explicitly as a constant-radius offset from a backbone curve (which is itself a variational curve, in this example). There are any number of such shapes that may properly be considered "free-form" under a broad definition, but that contain symmetries with respect to one or more defining curves, or have cross-sections that must match some specified functional form. Explicit definitions are a much more sensible way to specify such shapes, such as generalized sweeps[SK91, SK92]. Therefore, we consider ways of incorporating such explicit shapes in a variational model. The ability to smoothly join variational shapes and externally defined shapes in a common framework lets one a) "glue" these external shapes together with variational blend surfaces and b) modify their apparent topologies (independently of their explicit representations), by boring holes in them or attaching handles using trim-and-stitch operations (Figure 3.3).

3.4.1 Designing with approximations

Since a designer is only presented with approximate renderings of what are supposed to be smooth variational shapes, the question arises — how does one make sense of the designer's manipulation of these approximate shapes? One must never lose sight of the fact that the implicitly defined variational surface is the "real" surface. The user will interact only with these approximate renderings, but always with the understanding that operations will be interpreted as implicitly defining an ideal smooth shape (by modifying the definitions of the geometric constraints that frame the surface). The result is a method for directly manipulating these smooth surfaces, regardless of the coarseness of their explicit approximations.

A potential drawback of this approach is that modeling operations cannot depend on these approximations to reveal something about the exact location of the variational sur-

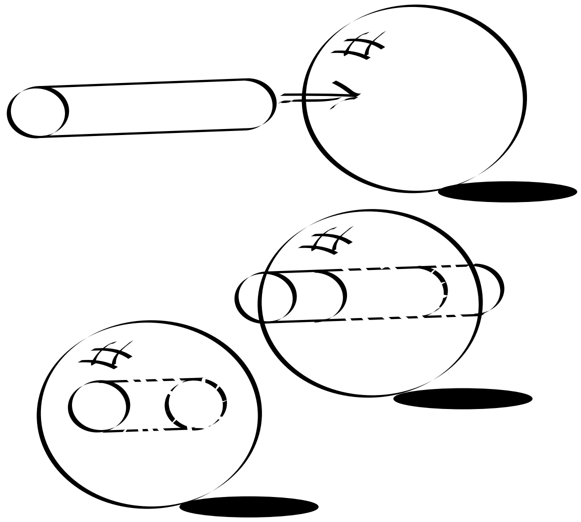

Figure 3.4: CSG-like boundary operations for surface construction: the cylinder and sphere are trimmed against their intersection curve, and then joined to make a single toroidal surface.

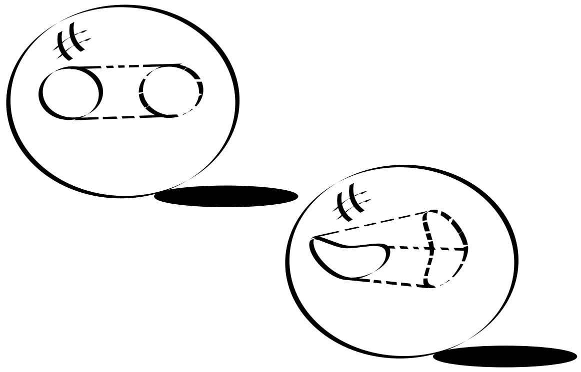

Figure 3.5: Rather than remaining dependent on the original sphere and cylinder, the original intersection curve becomes an independent control curve, and may be reshaped.

face. For example, one shouldn't look for points of intersection between two approximate surfaces in order to answer the question, "do the variational surfaces intersect?" Because of discretization error, whether or how two approximate surfaces intersect says nothing about the true intersection topology.

This is not as big a drawback as it might seem at first. As an example, consider the curved boundary-representation operations of[Rie89], in which intersecting surfaces are trimmed against each other and joined along their intersection curves [TTSC91]. If variational surfaces are being used, one cannot generally compute their intersection, which would seem to rule out this style of trim-and-stitch construction. But the limitation disappears if instead the trimming operation is conceived as taking a snapshot of the intersection between the two approximate surfaces, then *redefining* the parent surfaces to interpolate this independent curve as their new boundary (we construct a variational curve to approximate this intersection shape by sampling the explicit mesh intersection somewhat coarsely and using these as point constraints). This has the advantage over passive surface intersection that the intersection curve, and optionally its tangent ribbon, becomes an independent control curve in the composite surface, and can subsequently be directly reshaped by the designer (Figure 3.5).

3.4.2 Topological design

We are distinguishing between topological design and shape design because it is very natural and convenient to think about these as separate phases of a 3D design process. While from a design standpoint it is not particularly interesting to point out that a donut is topologically equivalent to a coffee mug, if one wants to consider designing a family of coffee mugs one might expect to fix the topology fairly early in the process, then manipulate or deform the shape while leaving the topology unchanged as the mug shape is refined. In this case, various disjoint regions might correspond to different components or features of the mug, each of whose shapes will be controlled in a different way. Contrast this with a volumetric sculpting process, in which mugs must be "carved" out of blocks of material. Here topology and shape are inextricably bound up together, so that in changing the shape one might accidentally change the topology (*i.e.*, gouging a hole in the mug while thinning a wall).

Summary

We have discussed what it might be like to design free-form curves and surfaces as variational shapes, and how one can build up non-trivial topologies gradually, using sequences of simple "surgical" operations. Variational shapes are described implicitly in terms of the constraints they satisfy and the quality measures they optimize. A modeler that operates on such variational specifications will need to be able to compute approximations to the optimal shapes, quickly. Some of the issues involved in computing such approximations were raised, to be fleshed out in upcoming chapters.

Chapter 4

Variational Shape Specifications

Synopsis

> Variational shape specifications are skeletal recipes for smooth curve and surface shapes. The recipes include topological information, local shape information in the form of geometric constraints, and shape objective functions. We represent these specifications as meshes of connected curve and surface elements, and tag each element with an appropriate "shape controller", depending on how the element's shape is to be computed. Here we develop algorithms for building, modifying, and tagging topological meshes to represent variational specifications.

This chapter develops our basic methods of specifying variational shapes, and of representing these specifications in the computer. We are *not* concerned here with explicit curve and surface shapes. Rather, we are concerned with the information needed to specify a variational optimization that implicitly defines a particular free-form shape. Given such a specification, methods of approximating the associated surface using any of a variety of explicit curve and surface representations will be taken up in later chapters.

Variational specifications have been present to varying degrees in previous variational surface modelers [CG91, WW92, MS92], implicit in the stitching-together of surface elements, the mechanics of local refinement schemes, and the objective functions used. But they have not really been considered as first-class entities unto themselves, independent of the explicit surface representations used by the modelers in question. Doing so is not particularly complicated; but it is a prerequisite to the kind of modeling we want to do. Ultimately, the designer will communicate with the modeler by creating and operating on such an abstract variational specification, and thereby avoid specializing the operations to any particular piecewise surface representation.

4.1 Ingredients

Any curve or surface model can be seen as containing two fundamentally different kinds of information: that related to topology, and that related to its immersion or shape. The topological information, discussed in Chapter 2, gives a decomposition of the model as a set of connected components. The immersion tells where each of the model's domain points is located in space.

In this work, a model's topology will always be represented explicitly, so that the designer can have explicit control over it. As was discussed in Section 2.2, not all shape representations allow the topology and the immersion to be treated separately. In representations where shape and topology are so intimately linked, there is no guarantee that the topology won't change along with the shape.

Immersed free-form curves and surfaces, as discussed in Chapter 2, are most commonly represented as explicit coordinate functions that map a 1D or 2D parametric domain into space. In contrast, a variational shape specification has no such explicit immersion. Rather, the immersion is characterized implicitly, in terms of geometric constraints it should satisfy and geometric measures it should maximize or minimize subject to these constraints. A geometric constraint might take the form of a control point or curve that the shape must interpolate (*i.e.*, explicit coordinate assignments for some subset of the model); or it might require the model to have a prescribed normal at some point or (though not considered here) maintain a prescribed area or volume. Geometric objective functions to be optimized subject to these constraints could include the goals of minimum surface area or curvature.

So three kinds of information are recorded in a variational specification: topology, geometric constraints, and geometric objective functions. Below, we consider how this information will be represented, using meshes, and discuss a number of mesh transformation algorithms necessary for building topological specifications. The mesh's shape will not be important at this point (an important aspect of the algorithms developed here is that the meshes needn't even have 3D coordinates associated with them). The problem of using these specifications to compute a variational shape will be taken up in the next three chapters.

4.2 Topology

In our modeler, points, curves, and surfaces are the only topological elements that will be needed. A natural way of representing these domains uses simplicial complexes (Chapter 2); and a natural way of representing these simplicial complexes on a computer is as lists of nodes (for 1D complexes) and triangulated meshes of nodes (for 2D complexes). With this, the seemingly abstract task of specifying and representing topology becomes the very concrete task of building a mesh. User-level tools for creating and modifying model topologies are discussed in Chapter 7. This section addresses lower-level representation issues — how to represent and operate on topological meshes

4.2.1 Topological representation

We begin by outlining how mesh specifications are to be represented, from the bottom-up, beginning with the simplest elements. There is nothing particularly novel or deep about this representation scheme; it is offered for definiteness in the discussions to follow. Any of a variety of boundary-representation schemes from solid modeling might have been used here (Chapter 2), but these contain much more element grouping and incidence information than is needed for our discussion.

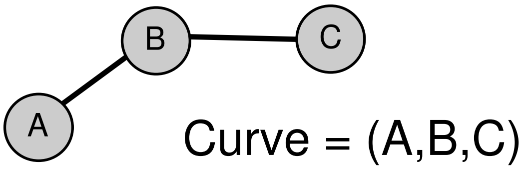

Figure 4.1: 1.5in

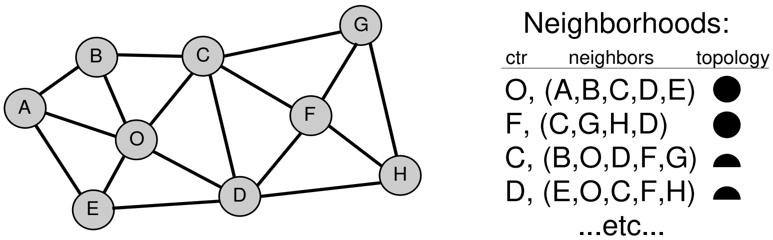

Figure 4.2: A 2D topological domain (a surface) is represented as a list of neighborhoods, each of which has a center node and a list of neighbor nodes.

Primitive elements

The simplest topological element is a point (a *node*, in the data structures below). Curve and surface elements will be built up as lists of nodes.

A 1D topological domain (a curve) is represented as a list of nodes.

A 1D interval (a curve segment) will be represented as a list of nodes, with edges implied between successive nodes (Figure 4.1). For open curves, the first and last nodes represent the boundary points; for closed curves, an edge is implied between the first and last nodes in the list. No more than one edge may connect the same two curve nodes (this restriction is important for embedded surface curves, below); thus, a closed curve must contain at least 3 nodes.

A 2D patch (a surface mesh) is represented as list of neighborhoods, each having a center node and an ordered list of neighbor nodes. (Figure 4.2). Triangular faces are implied between successive neighbor nodes. For nodes on a boundary of the surface, the first and last neighbors will lie on the boundary as well, and the neighborhood is isomorphic to a half-disc. For nodes in the surface interior, the neighborhood is isomorphic to a full disc, and a triangular face is implied between the first and last neighbors. Mechanically, neighbor lists work much as if they represent an (open/closed) embedded curve that bounds the (boundary/interior) neighborhood.

Curves that are embedded in a surface are represented as sequences of edge-connected surface nodes (Figure 4.3). It is necessary that the curve nodes be connected by surface edges so that the embedded curve can be treated as a restriction of the mesh to a 1D domain. A closed surface curve must contain at least 3 nodes, so that it will enclose one or more triangles on the surface and thus divide the surface into nonempty "inside" and "outside" regions. This constraint will simplify the mesh transformation algorithms in later sections.

For the use of some of the algorithms below and in Chapter 6 an explicit list of mesh

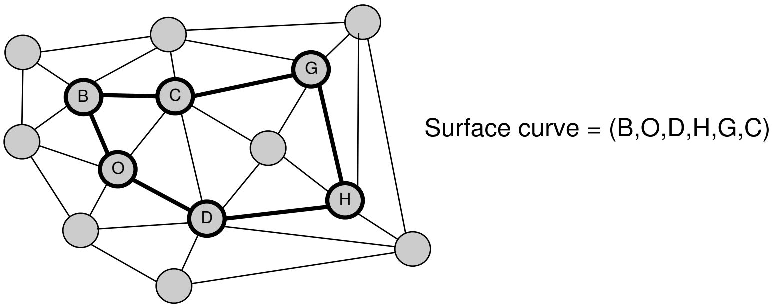

Figure 4.3: An embedded surface curve is a list of edge-connected surface nodes.

edges will be needed. The edge-list is easily derived from the neighbor lists of the mesh nodes, with each node-neighbor pair corresponding to an edge. A list of triangular faces can be similarly collected. The cost of these traversals might seem to be $\mathcal{O}(nodes^2)$ because there is no limit to the number of neighbors a single node may have, but this is not so. It can be shown (using the Euler characteristic (Chapter 2) and a simple counting argument) that for a triangulated surface of genus g, the total number of edges is bounded linearly by the number of vertices:

$$E \leq 3V + 6(g-1)$$

(equality holds when the surface is closed). Further, the faces are related to the edges by

$$F = 2/3E,$$

and thus the cost of the traversal is linear in the number of nodes.

This node-based mesh representation was chosen for simplicity, and because ordered neighbor traversal will be used so often in the neighborhood-based computations of Chapters 5 and 6. Other ways of representing triangulations might also have been used here that would maintain edge and face lists at all times, thus eliminating the separate linear-cost collection steps in favor of more tedious but constant-cost bookkeeping.

Continuous collections of elements: Regions

Points on a curve break it into disjoint intervals; and closed curves on a surface break it up into disjoint patches. Such regional decompositions of curve and surface domains will be very important to us, for organizing computations and data within a mesh. A model's topological mesh will generally be decomposed into an assortment of regions of dimension 0, 1, or 2 (Figure 4.4). The decomposition will be represented by maintaining a list of regions, and by labeling nodes, edges, and faces with the region to which each belongs. Note that an element may only belong to one region; thus, a region will not generally include its boundary elements, as they are often treated separately as regions of their own. This is closely related to the topological notion of a *cell-decomposition* (Chapter 2) and is a useful abstraction because cells do not depend on the particular triangulations of their components or their relative levels of refinement.

Special nodes or edges in a mesh that define embedded curves or control points are referred to as *source* edges and nodes in the triangulation literature. Given a mesh and

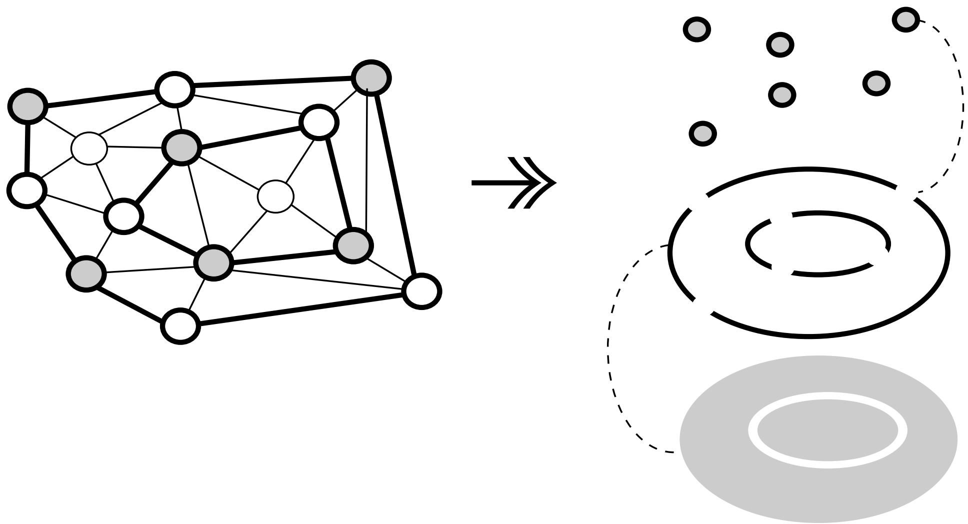

Figure 4.4: A surface mesh will be decomposed into disjoint 0D, 1D,and 2D regions. In the mesh on the left, the grey nodes and heavy edges indicate 0D and 1D regions bounding 1D and 2D regions, respectively. Unshaded elements are “filler”.

a collection of source nodes and edges, one may sweep through and assemble a list of regions, labeling each of the mesh elements using a standard connected-components algorithm ([CLR90]):

Algorithm: label-regions

```
1. For each source node:
    create a 0D region and label the node.

2. For each source edge:
    if neither of the edge's nodes is tagged with a 1D region,
      create a new 1D region
    else if both nodes are tagged with different 1D regions,
      union the different regions
    tag the edge and any untagged end node with the new 1D region
      (leave 0D labels in place)

3. For each surface triangle:
    if none of the triangle's edges is tagged with a 2D region,
      create a new 2D region
    else if more than one edge is tagged, with different 2D regions,
      union the different regions
    tag the face and unlabeled nodes and edges with the new region
    (leave 0D, 1D tags in place)
```

An efficient implementation of this algorithm uses a standard Union-Find data structure[CLR90] to tag and merge regions in essentially constant time (amortized over a sequence of oper-

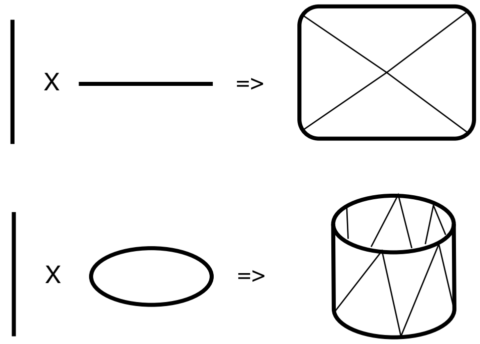

Figure 4.5: Constructing topological sheets and cylinders as products of 1-dimensional spaces

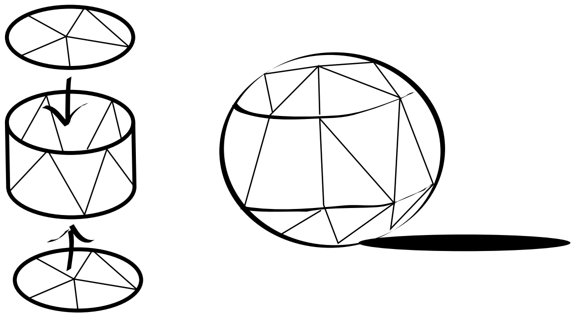

Figure 4.6: Capping the ends of a cylinder to make a topological sphere

ations). So, given the linear relationship between the numbers of edges, nodes, and faces, the cost of performing this region labeling is essentially linear in the number of mesh nodes.

4.2.2 Specifying topology by building a mesh

When it comes down to actually constructing meshes, this will sometimes be done "from scratch" and at other times by operating on an existing mesh to transform it. We want to avoid ever having to repair or transform a mesh by triangulating an unadorned set of vertices, because doing something topologically appropriate can be difficult if not impossible. Likewise, we want to avoid having the user perform any kind of surgery on an existing mesh that might leave it in an inconsistent state. Therefore, a set of mesh construction and transformation primitives will be developed, each of which is guaranteed to leave a topologically valid mesh in its wake.

The only meshes directly constructed are Cartesian products of an interval with either another interval or a circle. These are nothing more than planar sheets and cylinders, and they are constructed in obvious ways (Figure 4.5).

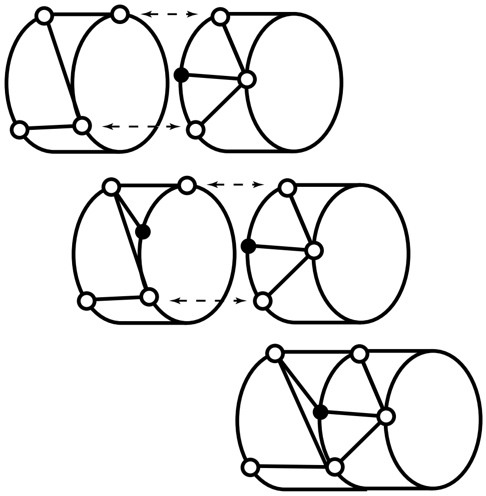

Figure 4.7: Two cylindrical surfaces are glued together after their boundaries have been made compatible through an edge-split.

Mesh surgery

Instead of making a cylinder from scratch, one might equally well have made a sheet, identified a pair of intervals on its boundary, and stitched them together. Other, more complex topologies will be built up in exactly this way, through surgery on boundary complexes of simpler meshes. A simple example is Figure 4.6, in which the open ends of a cylinder are closed by adding sheets to make a topological sphere. The basic surgical operations needed for meshes are cutting and gluing along embedded curves.

The mechanics of cutting are straightforward. Given a closed, embedded curve dividing a surface into inside and outside regions, the curve is "copied" by creating a corresponding list of new, unconnected surface nodes. Any 0D tags (control point indicators) are cloned as well, to propagate the way the curve is broken up into 1D regions. Then each of the original nodes' neighbor lists is split into inside and outside halves, and the inside neighbors are reassigned to the new curve's nodes. A similar operation is possible on an open embedded curve to cut a "slit" in a surface. In this case, only the interior nodes of the curve are copied and split, and the original curve boundary nodes are opened up into half-discs to connect the two new boundary curves into a single closed loop.

Gluing is, conceptually, the inverse of splitting. But the only time the two operations are algorithmic inverses is when the target is a pair of boundary curves whose regions and nodes correspond exactly (*i.e.*, two boundary curves just created by a splitting operation). In this case, gluing is a simple matter of identifying corresponding boundary nodes, discarding one set of nodes, and adding their neighbor lists to the corresponding nodes on the other curve. The two boundary curves become a single interior curve.

The general gluing operation is a bit more complicated, because it must handle surfaces bounded by curves with different numbers of nodes (Figure 4.7). The curves must first be brought into correspondence. We assume we are given a 1-1 correspondence between source nodes on the curves, which implicitly gives a correspondence between regions on the curves

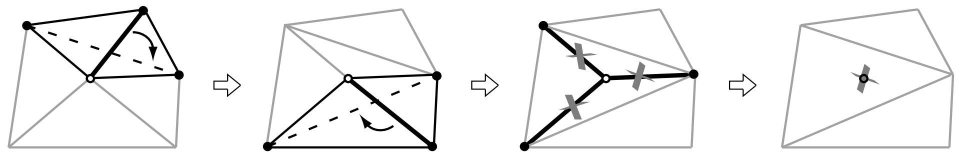

Figure 4.8: Node deletion: a sequence of edge-flips whittles away at the white node until an inverse face-split can be used to remove the node.

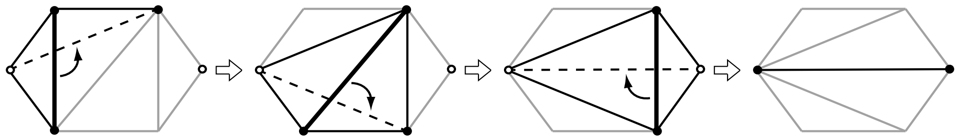

Figure 4.9: Edge insertion: a sequence of edge-flips modifies the mesh so that an edge is introduced between the white nodes.

as well. Note that whatever higher-level process is calling for the merge may need to add or delete source nodes on either curve to make the 1-1 correspondence possible. Each of the regions is then brought into node-to-node correspondence, with additional nodes being created as needed through edge splitting (described below). The merger then proceeds as above.

4.2.3 Mesh transformations

In this section we discuss *mesh transformations* — operations that transform between different triangulations of the same topological surface. These will be needed for purely topological reasons, *e.g.*, to insert p.l. curves into surface meshes or eliminate redundant mesh nodes. The transformations will also be used in later chapters in dealing with 3D meshes, and therefore we may dwell occasionally on issues related to node positions. But the intent for each of these algorithms is that it not depend on node positions in order to do its job, and that mesh topologies are always preserved or changed in controlled ways.

Our basic transformations will be the Alexander moves outlined in Figure 2.17. The order-1 refinement is more commonly known as an *edge split*, the order-2 refinement a *face split*. An edge split and an inverse edge split can be combined to exchange the diagonal of a mesh quadrilateral; this operation is commonly known as an *edge flip*, and treated as atomic. Note that boundary edges cannot be flipped (there is no quadrilateral), and edges that are part of an embedded surface curve should not be flipped or the curve's continuity will be disrupted.

Deleting nodes

In working with triangulations in the plane, node deletion is almost always accomplished by discarding the node and its edges and then re-triangulating that portion of the mesh. In working with abstract surfaces, planar re-triangulation is not an option. Figure 4.8 illustrates our node deletion algorithm. The simplest case is an infield, non-source node:

For degree-3 nodes, deletion is the inverse of the Face-split operation. For nodes of higher degree, iteratively flip away edges until the node is degree-3, then apply the inverse Face-split. It should be clear that this procedure always terminates in a degree-3 node that may then be removed.

Boundary nodes are handled similarly: they are first reduced to degree-2 by edge-flips, and are then deleted along with their associated face. In effect, the corner is snipped away, so that an edge connecting the triangle's other two nodes becomes a new boundary edge.

Deleting source nodes through which an embedded curve passes requires much more explanation, though little more actual work. Exactly two source edges meet at a source node — else the node would be a curve endpoint or curve intersection point and not a candidate for removal. Furthermore, following these two edges out to their other endpoints, one may assume these neighbor source nodes are not connected by a source edge — else, the closed curve would only contain 3 edges and again the node would not be a candidate for removal. Reduction to degree-3 proceeds as before using edge-flips (no source edges are flipped). A natural consequence of this reduction is that a (non-source) edge will be placed between the two neighbor source nodes, if one was not there to begin with. After the degree-3 node has been deleted the curve will have been disrupted. The curve's continuity may be restored by incorporating that non-source edge connecting the source nodes into the curve and marking it as a source edge to avoid future disruptions. Another way one might delete nodes uses the "edge-collapse" operation of [HDD$^+$93], which is more general than an inverse edge split because it can be used to remove nodes having any number of neighbors. Though it is conceptually simpler than the above, we found that implementing it to correctly handle source edges is more complicated than the approach taken here.

Remark on shape preservation

So far, our algorithms have been purely topological. All references to mesh components have been through nodes and neighbor lists, with no reference to actual 3D vertex coordinates. We could continue in this vein with the remaining mesh transformations. But later these meshes will be immersed in 3D as p.l. approximations to smooth shapes. It would be nice if mesh transformation operations, in addition to being topology-preserving, would also preserve shape as much as possible. This is not an absolute requirement like topology-preservation, because it will officially be someone else's job to worry about mesh shape (Chapters 5 and 6). But it will be helpful to other calculations if the mesh transformations do not perturb the shapes needlessly, and there are some simple things one can do toward that end.

When splitting edges or faces in 3D, clearly one can compute the new node's position to center it on the element just split. Deleting a node using edge flips can cause the mesh to fold back on itself. Since it does not matter topologically which edges are flipped or in what order, one may attempt to minimize folding by always flipping the edge that yields the flattest dihedral angle. This does not guarantee that the surface will keep its shape, but generally does a good job. If for some reason it was crucial that the surface shape be disrupted even less, a procedure that perturbed node positions could be devised.

We note in passing that node deletion, face splitting, and edge insertion (below) do not leave particularly "nice" triangulations when applied to 3D meshes; triangle sizes and aspect ratios can become rather uneven. Again, this is not a topological issue. If it is important that a nice triangulation be maintained over an immersed mesh (it will be for us), a re-triangulation step should follow these operations. Chapter 6 discusses a simple, topology-safe re-triangulator. We have found it *much* simpler to apply a single universal mesh improvement algorithm to a neighborhood after any one of these transformations, rather than complicate the individual transformations by devising each to leave a nice triangulation behind.

Edge insertion

The edge-insertion operation (Figure 4.9) is a basic part of our algorithm for inserting curves into a mesh. Given two nodes in the mesh, it adjusts the mesh so that an edge connects two nodes. First, a sequence of abutting triangles that connect these initial and final nodes is found. The triangles' union is a polygonal "channel" with no interior nodes, and the edge to be inserted will run down the middle of this channel. Looking down the channel from the initial to the final node, there are edges crossing the channel (like rungs of a ladder). We sweep from one end of the channel to the other, flipping each of these edges in succession. The final flip inserts the desired edge. Note that the channel must not be crossed by any source edge, because such an edge cannot be flipped out of the way (though see the discussion on curve insertion, below).

As with node deletion, this procedure can leave small folds in a p.l. surface due to unlucky edge-flips (particularly when three or more nodes are nearly collinear). One possible solution to this problem is to change the order in which channel edges are flipped, choosing flips that minimize creasing. Dyn, *et al.*[DGR93], show that in the planar case, there is always a channel and edge-flipping sequence that avoids folding. In the non-planar case, choosing the crossing edge that yields the flattest dihedral is a reasonable generalization, and given a flat mesh it reduces to their planar algorithm (assuming the channel itself is straight enough that the inserted edge won't touch or cross its boundary).

Inserting curves

In order to operate on surface curves (*e.g.*, to constrain a surface along a control curve, or prepare a mesh for surgery), such curves must be explicitly embedded in the mesh. Given a surface in 3D, one might use a "cookie-cutter" approach, extruding the immersed curve normal to the surface to make a cutting ribbon, and intersecting this ribbon with the surface to introduce new vertices and edges. Unfortunately, this offers no insight about inserting a curve as a purely topological operation, in the absence of a 3D mesh immersion. And, even for an immersed mesh, a robust implementation of this naive approach would be plagued with the kinds of "general position" problems that complicate so many algorithms from computational geometry.

A much simpler approach uses the edge-insertion operation to connect a sequence of nodes in the mesh. These nodes can be introduced through edge- or face-splitting operations. For a p.l. surface, this allows the curve to be "drawn" on the surface as a sequence of points, and their positions can guide the channel-finding part of the edge-insertion transformations. For a purely topological version of the operation, these must be guided by

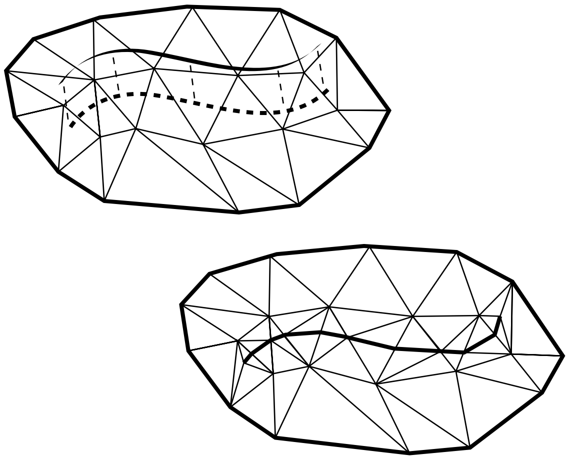

Figure 4.10: A p.l. curve is inserted into the mesh via edge splitting.

structural information.

If there is no requirement that the inserted curve contain only the originally specified nodes, another way to connect these nodes begins by finding polygonal channels connecting the nodes, as with edge-insertion. But then, instead of flipping the cross-channel edges out of the way, these edges are split, creating a sequence of edges that runs through the middle of the channel to connect the initial and final nodes (Figure 4.10). This completely avoids the folding problems discussed earlier for p.l. surfaces, at the expense of creating a more densely sampled curve. It also accommodates cross-channel source edges, by creating explicit intersection points between the new curve and existing curves.

4.3 Attaching shape specifications to the mesh

The structures and operations of the previous section will be used to create a mesh representation of the "topology-part" of a free-form shape specification. In this section we consider how to represent the "shape-part" of such specifications, information related to computing an immersion for the domain.

The shape specifications we want to support are mixtures of explicit positional information (*i.e.*, specific points that curves or surfaces should interpolate) and implicit information (*i.e.*, extremize a fairness objective). A variational specification must distribute such implicit shape information over the mesh, in the form of geometric constraints, objectives, and dependencies, telling how to compute an immersion. Think of it as a recipe for shape.

The mesh representation includes a decomposition into 0, 1, and 2 dimensional regions. Shape information will be incorporated into the mesh by giving each region its own "shape controller", a tag indicating how an immersion for that portion of the domain is to be computed. This is a little different from a completely explicit surface modeler, where a collection of surface patches would suffice to define a surface, because there is a mix of point, curve, and surface controllers, all interacting through constraint and objective function dependencies.

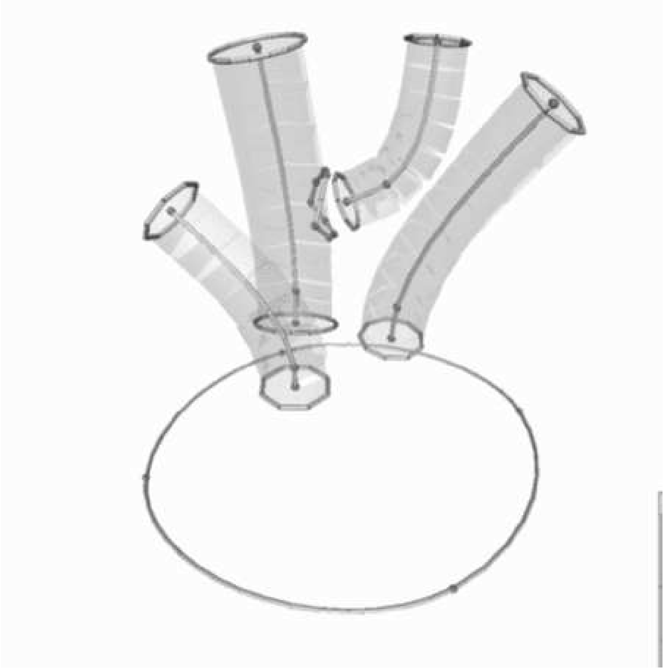

Figure 4.11: Constraint and objective skeleton for the Party Hat. The constraint curves and cylindrical offset surfaces (controlled by individual backbone curves) provide a partial specification for the overall surface shape.

4.3.1 Geometric Constraints

In this work we will consider *interpolation* and *normal* or *ribbon* constraints (Chapter 2). An interpolation constraint says that the surface must pass through a particular point or curve, or incorporate part of an explicitly defined surface region into its overall shape (external shape-copying). A normal constraint says the shape should have prescribed normal vectors within a specified region; this lets us force a curve or surface to be tangent to some other shape. The computational issues involved in enforcing these constraints will be discussed Chapter 5. Here we are concerned with representing the constraint information rather than computing a satisfying shape.

Interpolation constraints will ultimately be implemented by copying the shape of some "source" piece of geometry. The mesh region will be tagged with a pointer to this source, for use by the process responsible for computing the region's shape. There are no restriction on the particular representation of such source geometry — it could be a piecewise smooth polynomial, an algebraic surface, or a p.l. shape represented by some other mesh. It will be the region controller's job to sample this source immersion and use it to construct a matching shape for the region. As a concrete example, surface control curves are implemented by having an embedded curve copy the position of a corresponding free-standing control curve, carrying the surrounding surface with it. The topological mesh region corresponding to the embedded curve would be tagged with a pointer to this control curve.

These kinds of one-way constraints allow one to establish a hierarchy of control, recorded in the mesh by constraint dependencies. Continuing the example above, the free-standing control curve might be a variational curve forced to interpolate selected control points. The curve nodes corresponding to these control points would be tagged with pointers to free-standing control points. Changing the position of a control point would cause the control curve to change shape, which would then cause the surface to change shape.

We only consider normal constraints anchored to point or curve constraints. The normal information is stored similarly to the positional interpolation information, as pointers to externally represented tangent planes or ribbons.

4.3.2 Specifying creases and corners

The shape computation scheme to be used in Chapter 5 assumes that regions are smooth throughout their interiors, and that discontinuities may only occur at region boundaries. Two kinds of continuity will be supported between faired regions: G^0 (positional continuity) and G^2 (positional, normal, and curvature continuity). A G^0 join between two curves or surfaces is a crease or corner; a G^2 join has the same degree of smoothness as the interior. The fact that this maximum continuity is G^2 and not G^1 or G^n is a function of the approximation scheme adopted in Chapter 5, and might be different if a different approach were used.

To record this continuity information, in keeping with the spirit of labeling regions with shape-related information, region boundaries might be tagged with the desired continuity. But continuity is a matter of communication between adjacent regions: for C^0 continuity, only positional information along the boundary is shared; for C^2 , normal and curvature information propagates across the boundary. It will turn out to be computationally convenient to have the mesh directly reflect this communication. Thus, a C^0 join will be represented as two independent meshes constrained to interpolate a shared boundary curve. A C^2 join will be represented simply as a single mesh with an interior embedded curve, as has been discussed all along. Creasing a surface means splitting its mesh along the crease curve and constraining both sides to interpolate it.

4.3.3 Objective functions

The final step in constructing a variational specification is to associate an objective function with any unconstrained regions in the mesh. These are functions that measure some aspect of shape at a point of evaluation and return a "goodness" rating for that point. Integrating the measure over a piece of geometry measures the overall quality of the shape relative to other possible shapes. The modeler will use this to determine what to do with curve and surface shapes in-between constraints, by seeking shapes that minimize this measure subject to any geometric constraints that may be present.

In this work we use objective functions whose minimization yields "fair" surfaces. The functions measure total curvature over curves and surfaces, and are based on the bending energy of a thin beam or plate:

$$\mathcal{E}_{\text{curve}} = \int_{\text{curve}} \kappa^2 ds, \tag{4.1}$$

$$\mathcal{E}_{\text{surf}} = \int_{\text{surf}} (\kappa_1^2 + \kappa_2^2)\, dA. \tag{4.2}$$

Minimizing these functions distributes curvature over curves or surfaces to eliminate unwanted bulges or wiggles; they are discussed in more detail in Chapter 5. Other objective functions one might consider, though they are not addressed in this work, include surface area (Section 2.5.2) or variation of curvature. The tradeoffs between the thin-plate and other fairness measures are discussed in more detail in the next chapter.

Taken together, this information — topology, geometric constraints, and piecewise objective functions — implicitly describes a variational shape. Though one cannot solve directly for an immersion of a given topological domain, a specification like this can be used to set up approximations based on any of a variety of piecewise smooth representations. This will be taken up in the following chapter. Further implementation issues, such as how to coordinate the various shape computations with topological operations, and how to build a free-form shape modeler using this general approach, will be addressed in Chapter 7.

Summary

We have described a method of representing and constructing a variational shape specification. The basic representation is a tagged mesh: the mesh represents topological information, and the tagged regions within it indicate how the shapes of various curve and surface components are to be computed. Tagged meshes are not new — they are basic to any boundary-representation scheme in constructive solid geometry. The intended use, however, is novel, as are the topology-safe mesh transformations for node deletion and curve insertion. These transformations may also find use beyond our application, in general polygonal mesh manipulations.

Chapter 5

Approximating Variational Shapes

Synopsis

In this chapter we present a method of approximating shapes that minimize a geometric thin-plate objective function while satisfying point and normal interpolation constraints. Triangulated surface meshes are used as approximations to smooth surface shapes (the same meshes used to record topology in the previous chapter). Surface curvature is estimated at mesh nodes by fitting a quadratic surface function at each mesh neighborhood. The mesh is driven towards an optimal approximating shape by minimizing a sequence of quadratic approximations to the geometric objective function (Figure 5.1).

The skeletal shape specifications developed in the previous chapter gives implicit representations of curve or surface shapes. In order to move from there to an explicit 3D representation — which will be needed to render, export, or do just about anything useful with the geometry — the corresponding variational minimization problem must be solved. But it may not be possible to solve such problems in closed form, due to the complexity of the domain, objective function, or boundary conditions[BCO81]. Generally, the best one can hope to do is compute approximations to the true solutions.

5.1 Overview of the approximation method

At an abstract level, our approach to variational approximation is typical. We have been given an infinite-dimensional problem: find an immersion for the domain curve and surface regions of the skeletal specification, mapping their points into space in a way that satisfies the given geometric conditions. This is converted into a finite-dimensional problem by choosing an explicit representation for the immersed curves and surfaces, then re-stating the constraints and objectives in terms of these representation parameters. Because of the particular constraint and objective functions we use, the discretization step will yield a multivariate optimization that is nonlinear in the representation parameters. The optimal representation parameter values are solved for — thus “fitting” the curve or surface to the variational shape — by minimizing a succession of quadratic approximations to the nonlinear problem.

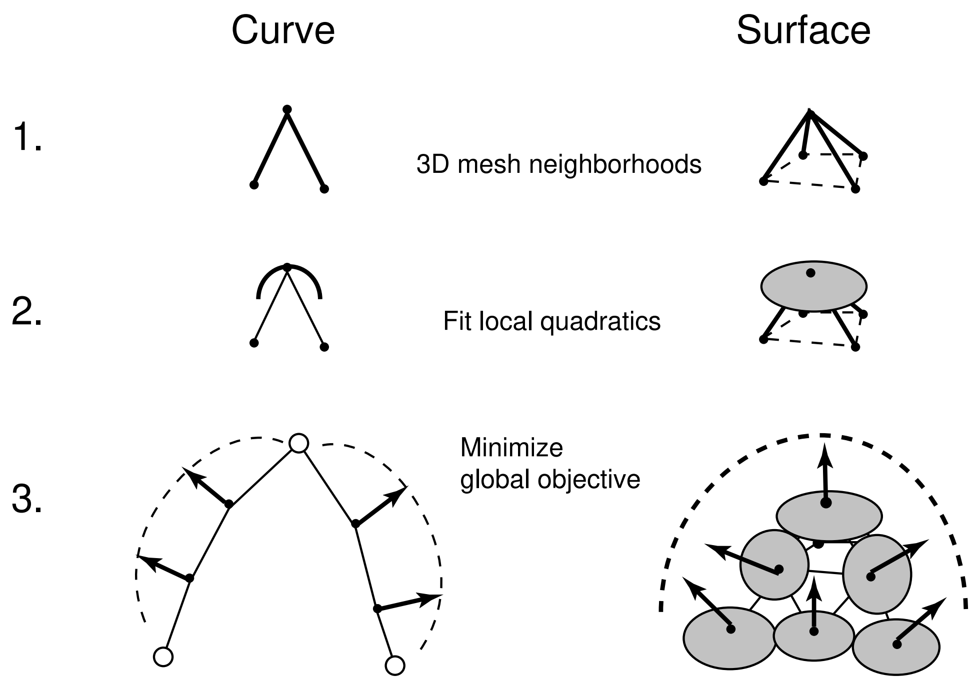

Figure 5.1: Synopsis of approximation approach: 1) Use a p.l. mesh to approximate smooth shapes. 2) Estimate curvature at mesh nodes. 3) Drive the mesh towards an optimal shape by minimizing a discretized curvature integral.

What is unusual about our approach is that we will take as the approximating representation p.l. curves and triangulated surface meshes — the same meshes used in the previous chapter to represent region topologies. The task then is to compute a 3D position for each mesh node.

5.2 Why a mesh?

An explicit representation scheme must be chosen for curves and surfaces in $\mathcal{R}^3$. Section 2.5.4 discussed earlier work that used piecewise smooth curve and surface patches (tensor-product B-splines, Bezier triangles, subdivision surfaces) as representations for variational surface approximations, along with the pros and cons of each for use in an interactive modeler. In this work, we will use a triangulated mesh to represent approximate smooth surfaces. The requirement that computations over arbitrary topologies be feasible at interactive speeds was the principal reason for this choice; if we were willing to wait seconds or hours for a surface to be computed [HKD93, LP88, NLL90, MS92], just about any linear patch scheme would work.

Instead, we will give up smoothness and directly immerse the triangle meshes of the previous chapter to make a p.l. approximation of the free-form shape. While not as visually appealing as a smooth representation, it will be fast, and we expect that in an interactive design tool it will be OK to serve up such faceted approximations quickly, with the promise that a high-quality smooth surface can be fit to the final design in a post-processing step. Of course, smooth functions (such as a curvature-based objective function) cannot be evaluated directly on such a non-smooth mesh. Section 5.3 addresses this by fitting a quadratic patch to the neighborhood around each node. Thus, the representation may be viewed as either a p.l. surface with curvature estimates at the nodes, or as a discontinuous union of quadratic discs, depending on your preference for finite difference or finite element methods, respectively[ZM83].

A side-effect of the combined shape/topology roles to be played by our meshes is that they will generally contain many more than the minimum number of simplices needed to express the topology. A denser mesh will be used to more accurately approximate free-form shapes by providing sample points. Nothing about our use of topological meshes will demand a minimal set of vertices, so it is alright to enrich the mesh for this purpose. When, in Section 6.3, a mesh refinement scheme is developed as part of the surface approximation machinery, the mesh transformation operations of Section 4.2.3 will be used to ensure that the underlying topology will not be changed by coarsening or refining the mesh. (this is in contrast with multi-resolution polygonal modeling schemes in which holes might disappear along with other fine detail as the model is coarsened[HG94]). Note that it will always be possible to derive a minimal mesh from a richer one by repeatedly un-refining until no legal moves are left, effectively "dehydrating" the mesh to leave its topologically essential components.

5.3 Smooth mesh neighborhoods

If a mesh is to be used to approximate variational shapes, one must compute over the mesh as if it was a sampling of a smooth surface. To do so, in addition to sample point positions one must also be able to evaluate surface first and second derivatives at each of these points, in order to compute curvatures and normals. If the mesh was regular, a standard finite difference stencil could be applied to the node and its neighbor positions to estimate these derivatives. With irregular mesh neighborhoods, this approach breaks down since a single regular stencil cannot be used.

As it happens, the way that finite-difference stencils are constructed is by making a truncated Taylor series expansion of a low-degree polynomial fitted to a mesh neighborhood[Lan56]. This approach does generalize to irregular mesh neighborhoods[FW60]. Since surface curvature must be computed, the first and second derivative terms of the series will be retained, effectively fitting a quadratic patch to the neighborhood. This is essentially the approach taken in several earlier scattered data approaches (*e.g.*, [Law77, SZ90, Ham93]). What is new here is the way neighborhood parameterizations are constructed with which to conduct the fitting.

5.3.1 Building neighborhood parameterizations

A garden-variety rectangular finite-difference mesh has global parameterization — the (u, v) plane. As has been discussed, global parameterizations generally do not exist for the surfaces we want to model. So instead a separate parameterization will be constructed for each mesh neighborhood, and the neighborhood quadratic will be fitted with respect to this parameterization.

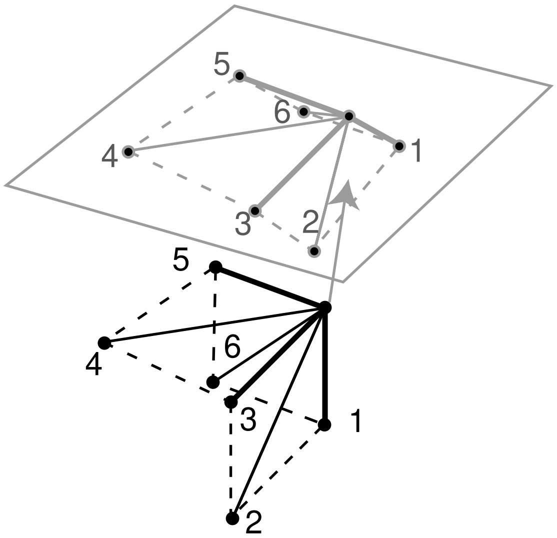

Figure 5.2: A failed neighborhood projection: projecting onto a plane does not guarantee a one-to-one relationship between surface points and plane points, and this can scramble the angular ordering of the neighbors.

As discussed, a usual way to parameterize this fit is by projection: find a normal, then project the neighborhood onto the tangent plane.

A weakness of the projection method is that it requires reliable estimates of curve and surface normals in order to determine a tangent plane on which to project the parameterization. If the normal used isn't close to the ultimately fitted normal, the expectation is the shape of the fitted surface will not be a good match to the neighborhood geometry. A Taylor expansion of a smooth surface using geometric information such as the true curvature and normal would differ from this fitted quadratic.

A more serious problem with such a projected parameterization is related to the behavior of the projection and subsequent fitting when a p.l. surface neighborhood is folded (*i.e.*, has high polyhedral curvature). In such situations, the resulting projection of the neighborhood on to the parameter plane may not be one-to-one (Figure 5.2). This has the effect of scrambling the angular order of the neighbors about the node, and thus the topology of the fitted polynomial will not match the topology of the folded mesh, and lead to erroneous derivative estimates. This folding is perhaps less disastrous (though still anomalous) when applied to the analysis of static objects (as in [SW92]), since such neighborhoods will be rare for densely-sampled surfaces. But in our interactive modeler, there is nothing to stop the user from tugging on a node and momentarily placing the surface in a badly folded configuration. Ultimately, the fairing scheme will "iron out" such transient folding, but only if the numerics honor the topology of these folded neighborhoods so that high polyhedral curvature is reflected in the smooth curvature estimate.

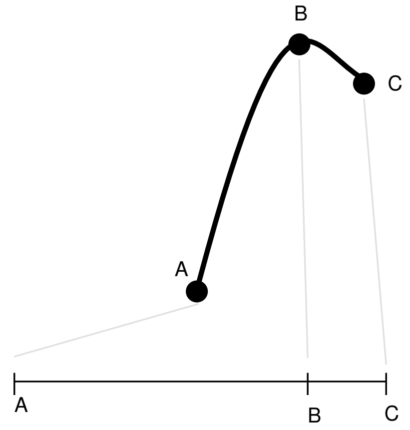

Figure 5.3: The straight-line distance between successive interpolation points is taken as an approximation of their arc-length separation along the curve.

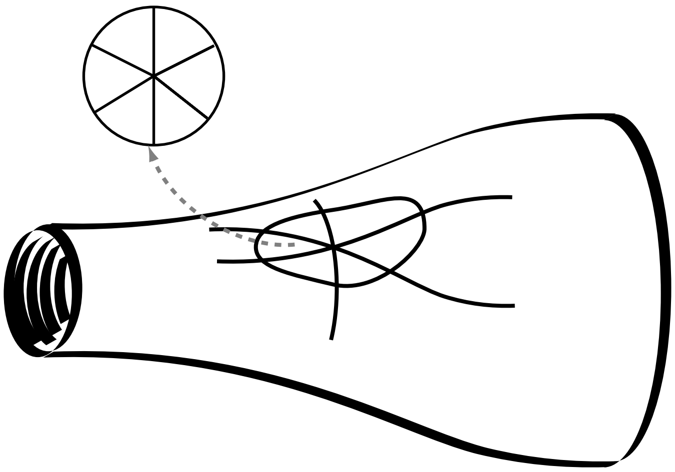

Figure 5.4: Geodesic polar map at a surface point.

Parameterizing curve neighborhoods

Rather than rely on an accurate normal projection to capture metric information (or ignore it altogether by using a uniform parameterization), an arc-length parameterization of the curve will be mimicked by directly by measuring the Euclidean distances between neighbors and taking that as their parametric separation. This is the well-known chord-length approximation to an arc-length parameterization[Far90, Epp76] (Figure 5.3).

Parameterizing surface neighborhoods

Neighbor distances on the mesh surface can be directly measured as in the curve parameterization; but also needed are *tangent plane* angular separations between the geodesic paths

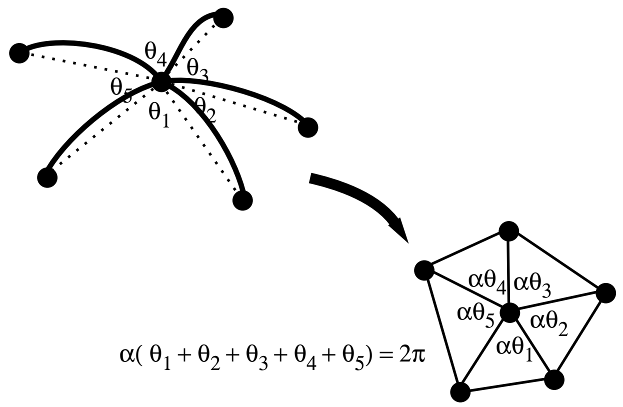

Figure 5.5: Parameterizing a surface neighborhood: angles between neighbors are measured in 3D, then scaled to sum to 2π. The distances between neighbors and the center are measured in 3D analogously to the chord-length approximation to arc-length along a curve.

to various neighbor points. This is the surface analogue of an arc-length parameterization — the so-called *geodesic polar map*[O'N66] (Figure 5.4). It is simply a generalization to geometric surfaces of polar coordinates (r, θ) in the plane, where r now indicates a distance traveled along a geodesic path away from the point, and θ indicates the initial direction to be taken. Unlike a curve, which can be given a single global arc-length parameterization, geodesic maps must be constructed separately for each point on a curved surface.

In the spirit of the chord-length approximation to arc-length, we will measure the angular separation between each successive pair of neighbors *in 3D*. These angles are then "projected" onto a plane by uniformly scaling them so that they sum to 2π for nodes in the surface interior and π for nodes on the boundary (Figure 5.5). The Euclidean distance from the center node to each of its neighbors is taken as the radial parametric separation. This procedure yields an (r_i, θ_i) for each neighbor (by convention, the first neighbor is at $\theta = 0$). From this the corresponding (u_i, v_i) are computed by a standard polar to Cartesian transform in 2D. By analogy to the chordal parameterization for curves, we will refer to this as a *faceted parameterization* for surface neighborhoods.

5.3.2 Surface coordinate fitting

Given a suitable parameterization of the neighborhood, it is a straightforward task to fit a truncated Taylor series expansion for each of the surface x, y, and z coordinate functions about the neighborhood center. In the following, $\mathbf{p}_0$ will designate the position of the node at the neighborhood center, $\mathbf{p}_1 ... \mathbf{p}_n$ the positions of $\mathbf{p}_0$'s n neighbors, and $(u_0, v_0)...(u_n, v_n)$ their parametric coordinates ($(u_0, v_0) = (0, 0)$).

For each of the coordinate functions we seek the coefficients of a quadratic:

$$s(u, v) = d_0 + d_1\, u + d_2\, v + \frac{d_3}{2} u^2 + d_4\, uv + \frac{d_5}{2} v^2 \tag{5.1}$$

$$= \mathbf{b}(u, v)\mathbf{d}, \tag{5.2}$$

where $\mathbf{b}(u,v)$ is the basis row vector $[1, u, v, \frac{1}{2}u^2, uv, \frac{1}{2}v^2]$, and $\mathbf{d}$ a column vector of coefficients. We want $s(u_0, v_0) \equiv s(0,0) = p_0$; this requires $d_0 = p_0$ (here p_i is taken to mean one of p_{ix}, p_{iy}, p_{iz}, since the same fitting procedure applies to each). The remaining coefficients will be determined such that the $s(u_i, v_i)$ are a least-squares fit to the p_i.

For completeness, here are the details of the calculation: first, shift the neighborhood's 3D origin to p_0, yielding the vector of shifted neighbor positions $\mathbf{q} = [p_1 - p_0, ..., p_n - p_0]^T$. The sample matrix $\mathbf{S}$ for this shifted, center-constrained system is built by evaluating the basis vector $\mathbf{b}(u_i, v_i)$ for each of the neighbors and collecting these rows into a matrix, then deleting its first column:

$$\mathbf{S} = \begin{bmatrix} u_1 & v_1 & \frac{1}{2}u_1^2 & u_1 v_1 & \frac{1}{2}v_1^2 \\ & & \vdots & & \\ u_n & v_n & \frac{1}{2}u_n^2 & u_n v_n & \frac{1}{2}v_n^2 \end{bmatrix}.$$

Then $\mathbf{S}[d_1, ..., d_5]^T = \mathbf{q}$, and the least-squares solution for $d_1 ... d_5$ is

$$\begin{aligned} [d_1, ..., d_5]^T &= [\mathbf{S}^T\mathbf{S}]^{-1}\mathbf{S}^T\mathbf{q} && (5.3) \\ &= \mathbf{Zq}. && (5.4) \end{aligned}$$

The shifted formulation in q is used instead of one in p because it reduces by one the rank of the matrix that must be inverted. The mesh's node and neighbor positions will of course be stored as absolute positions, not shifted so that p_0 is the origin. But now that there is a way to compute Z, the above may be recast more conveniently in terms of p_i instead of q_i. Let $\mathbf{P}$ represent the $(n+1) \times 3$ matrix of the $\mathbf{p}_i$'s x, y, z coordinates. The vector form for the reconstructed surface positions $\mathbf{s}(u,v)$ is then

$$\begin{aligned} \mathbf{s}(u,v) &= \mathbf{b}(u,v) \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -\sum Z_{1i} & Z_{11} & \cdots & Z_{1n} \\ \vdots & \vdots & \vdots & \\ -\sum Z_{ni} & Z_{n1} & \cdots & Z_{nn} \end{bmatrix} \begin{bmatrix} p_{0x} & p_{0y} & p_{0z} \\ p_{1x} & p_{1y} & p_{1z} \\ & \vdots & \\ p_{nx} & p_{ny} & p_{nz} \end{bmatrix} && (5.5) \\ &= \mathbf{b}(u,v)\mathbf{BP}. && (5.6) \end{aligned}$$

This is the form that will be evaluated when surface derivatives are computed in the upcoming sections.

The fitting procedure is somewhat complicated by fact that not all neighborhoods have five neighbors. And even if a node has five immediate neighbors, they might be positioned parametrically so as to make the full quadratic fit ill-conditioned (*i.e.*, if two neighbors are nearly collinear). We cannot look beyond the immediate neighbors to bring in additional nodes, as is common in planar least-squares schemes[FN90], because an angular ordering for nodes beyond the immediate neighbors is not determined by the surface triangulation.

In cases where there are not enough neighbors for a full fit, or where the full fit is poorly conditioned, we throw away some of the polynomial basis functions. For each node (of sufficient degree), an initial fit of the full basis $[1, u, v, \frac{1}{2}u^2, uv, \frac{1}{2}v^2]$ is attempted. The condition number of this fit, $c = \|\mathbf{S}^T\mathbf{S}\| \cdot \|(\mathbf{S}^T\mathbf{S})^{-1}\|$, is then computed (where the matrix norms are Frobenius norms[GVL89], $\|A\|_F = \sqrt{\sum_i \sum_j a_{ij}^2}$).

If the fit was ill-conditioned (say, $c > 1000$), or if there were too few neighbors, a fit is attempted with the reduced basis functions $[1, u, v, \frac{1}{2}(u^2+v^2)]$ for interior nodes[1]. Boundary nodes, which will rarely have enough neighbors for a full fit, are treated specially: the parameterization is constructed so that the two boundary nodes lie on the $\pm u$ axis, and the basis functions $[1, u, v, \frac{1}{2}u^2]$ are used. This lets a surface curve along its boundary while remaining flat in the infield direction. As before, the condition number is evaluated and an ill-conditioned fit rejected. As a last resort, a planar fit (for boundary or infield nodes) is made with the basis functions $[1, u, v]$.

A shortcoming of this approach is that the somewhat arbitrary choice of basis functions could lead to instability over time. A neighborhood that is nearly well-conditioned might switch back and forth between different sets of basis functions on successive parameterization/fitting operations when used in the iterative minimization procedure later in this chapter (we have not actually observed such behavior). Better would be to consistently use the same set of basis functions and optimize some auxiliary norm in the underdetermined case. Barth's approach[Bar90] is an example: an orthogonal decomposition of $\mathbf{S}^T\mathbf{S}$ splits the range into well-conditioned and ill-conditioned subspaces. Ill-conditioned components can be left out of the fit, and then values chosen for them that minimize a separate norm, *e.g.*, coefficient magnitudes. Unfortunately, the orthogonal decomposition adds a good deal of computational expense, and it will thus degrade the overall interactivity of the modeler. We do not use such a scheme because we have not observed the instability problems it is intended to solve. Our meshing procedures (Chapter 6) do a fair job of keeping neighborhoods well-conditioned so that this borderline behavior doesn't arise.

5.3.3 Curve coordinate fitting

Fitting curve neighborhoods is so mathematically and algorithmically similar to surface fitting that the details will be omitted. A quadratic curve segment will be fit to each nodal neighborhood in the curve, parameterized as discussed earlier. The fitting procedure above is used with the monomial basis functions $b(t) = [1, t, \frac{1}{2}t^2]$. The vector form for the fitted curve position function $\mathbf{c}(t)$ is then

$$\mathbf{c}(t) = \mathbf{b}(t)\mathbf{BP}, \tag{5.7}$$

where $\mathbf{B}$ is a $3\times npts$ array computed as in Equation 5.6.

5.4 Geometric objective functions

In Section 2.5.2 we discussed a geometric thin plate function that gives a measure of curves' and surfaces' total curvature. Shapes that minimize this measure are free of unwanted wiggles, creases, or bulges (*i.e.*, they are *fair*). Here we revisit curve and surface elastica, and with some mathematical manipulation arrive at forms that can be simply and efficiently evaluated over the quadratic mesh neighborhoods of the previous section.

[1] It is tempting to damp the second-order terms in the system matrix $\mathbf{S}^T\mathbf{S}$ by adding a constant on the diagonal. This will insure well-conditioning without these repeated fitting attempts. We tried this, but it led to curvature estimation errors that noticeably degraded the fairing computations of the next section.

5.4.1 Thin spline curve functional

Our chosen curve fairing functional mimics behavior of a physical spline by minimizing *strain energy*, approximated as the arc-length integral of curvature:

$$\mathcal{E}_{\text{curve}} = \int \kappa^2 ds \tag{5.8}$$

$$= \int (\mathbf{c}_{ss})^2 ds, \tag{5.9}$$

where, as before, $(\mathbf{c}_{ss})^2$ indicates the dot product of the second arc-length derivative $\mathbf{c}_{ss}$ with itself.

Recall that the chordal parameterization of our p.l. curves is an approximation to an arc-length parameterization. We will therefore take the fitted $\mathbf{c}_{tt}$ from Equation 5.7 as an approximation to $\mathbf{c}_{ss}$. Unlike the linearizations discussed in Section 2.5.2, this approximation inherits metric information from its special parameterization, and will not suffer the same distortions as in Figure 2.7 (the ultimate result is illustrated in Figure 8.1). This use of an approximate arc-length parameterization in order to have vvc_{tt}^2 approximate κ^2 appears in Hagen's work[HS90, HB91b].

5.4.2 Thin plate surface functional

By analogy to curve fairing, we take as the surface fairing objective function the integral of the squared principal curvatures over a smooth surface

$$\mathcal{E}_{\text{surf}} = \int_S (\kappa_1^2 + \kappa_2^2)\, dA, \tag{5.10}$$

where dA is the differential area form. This, and approximations to it, have been used as approximations the strain energy of a thin elastic plate. In order to express the thin plate functional in terms of a general parametric surface, some definitions and results from differential geometry ([Ham93, Spi79b]) are needed:

Begin with a surface $\mathbf{s}$ parameterized by u, v and immersed in $\mathcal{R}^3$,

$$\mathbf{s} = \mathbf{s}(\mathbf{u}) = (x(u,v), y(u,v), z(u,v)).$$

In the remainder of this section, all definitions and formulae assume $\mathbf{s}$ is being evaluated at a particular point $\mathbf{u}_0 \in \mathcal{R}^2$, though we will avoid notational clutter and not indicate this explicitly.

The partial derivatives $\mathbf{s}_u$ and $\mathbf{s}_v$ are a basis for the tangent plane at each surface point, though they are not necessarily orthonormal. The matrix of metric coefficients

$$\mathbf{G} = \begin{pmatrix} \mathbf{s}_u \cdot \mathbf{s}_u & \mathbf{s}_u \cdot \mathbf{s}_v \\ \mathbf{s}_v \cdot \mathbf{s}_u & \mathbf{s}_v \cdot \mathbf{s}_v \end{pmatrix},$$

where $i, j \in u, v$, give rise to the *first fundamental form*, I:

$$I(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{G} \mathbf{y},$$

for tangent vectors $\mathbf{x}, \mathbf{y} \in \mathcal{R}^2$ (with respect to the tangent basis vectors $\mathbf{s}_u$ and $\mathbf{s}_v$). Similarly, the normal section curvature coefficients

$$\mathbf{L} = \begin{pmatrix} \mathbf{s}_{uu} \cdot \mathbf{n} & \mathbf{s}_{uv} \cdot \mathbf{n} \\ \mathbf{s}_{uv} \cdot \mathbf{n} & \mathbf{s}_{vv} \cdot \mathbf{n} \end{pmatrix},$$

where $\mathbf{n}$ is a unit surface normal, *i.e.*,

$$\mathbf{n} = \frac{\mathbf{s}_u \times \mathbf{s}_v}{||\mathbf{s}_u \times \mathbf{s}_v||}$$

give rise to the *second fundamental form*, II:

$$II(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{L} \mathbf{y}.$$

A basic result from differential geometry says that the eigenvalues (both real) of the matrix

$$\mathbf{A} = \mathbf{G}^{-1}\mathbf{L}$$

are the principal curvatures κ_1 and κ_2 ([Spi79b, III.51]). That means a matrix $\mathbf{Q}$ exists that diagonalizes the matrix $\mathbf{A}$ by mapping $\mathbf{s}_u$ and $\mathbf{s}_v$ to two orthogonal unit eigenvectors:

$$\mathbf{QAQ^{-}1} = \begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix}.$$

Other important quantities related to $\mathbf{A}$ are the Gaussian curvature:

$$K = \kappa_1 \kappa_2 = det(\mathbf{A}) = det(\mathbf{L})/det(\mathbf{G}),$$

and the mean curvature

$$H = \frac{\kappa_1 + \kappa_2}{2} = trace(\mathbf{A})/2.$$

(recall that det and *trace*, like the eigenvalues themselves, are invariant under similarity transformation and thus do not depend on the choice of parameterization).

We are interested in the sum $\kappa_1^2 + \kappa_2^2$, the geometric thin plate integrand. This quantity is not so nicely related to $\mathbf{A}$, $\mathbf{L}$, or $\mathbf{G}$. It may be written as:

$$\kappa_1^2 + \kappa_2^2 = (\kappa_1 + \kappa_2)^2 - 2\kappa_1\kappa_2 = 4H^2 - 2K,$$

which leads to a rather complicated expression in terms of the elements of $\mathbf{L}$ and $\mathbf{G}$ (that is nonetheless quadratic in the elements of $\mathbf{L}$). Rather than pursue this line further, we note two facts: first, if the tangent plane basis vectors are orthonormal (that is, $||\mathbf{s}_u|| = ||\mathbf{s}_v|| = 1$ and $\mathbf{s}_u \cdot \mathbf{s}_v = 0$), as they are for a geodesic parameterization, then $\mathbf{G}$ will be orthogonal. Second, the sum of the squares of the elements of any positive definite symmetric matrix (the Frobenius norm, denoted $|| \cdot ||_F^2$) equals the sum of its squared eigenvalues (because this norm is invariant under isometry). Thus, for $\mathbf{G}$ orthogonal $\kappa_1^2 + \kappa_2^2$ may be computed directly as $||\mathbf{L}||_F^2$, since $\mathbf{Q}$ in Equation 5.4.2 must be orthogonal in this case. Then, in terms of $\mathbf{L}$'s elements,

$$\mathcal{E}_{\text{surf}} = \int_S ((\mathbf{s}_{uu} \cdot \mathbf{n})^2 + 2(\mathbf{s}_{uv} \cdot \mathbf{n})^2 + (\mathbf{s}_{vv} \cdot \mathbf{n})^2) dA,$$

which is also quadratic in $\mathbf{L}$'s elements. It is worth pointing out that

$$\kappa_1^2 + \kappa_2^2 \neq \frac{||\mathbf{L}||_F^2}{||\mathbf{E}||_F^2},$$

though this is a tempting-looking formula.

Recall that faceted parameterizations are taken as approximations to geodesic maps for the neighborhoods. As with curves above, the derivatives of the fitted surface quadratic (Equation 5.6) are taken as approximations to derivatives wrt such a geodesic map, taking advantage of the built-in metric information from the parameterization to justify simplifying the curvature expressions. Similar use of a local quadratic fit to estimate II, and subsequently, surface curvature at a point, has appeared in [SZ90] and [Ham93], though in that work projection onto an estimated surface normal is used to parameterize the neighborhood.

5.5 Discretized objective functions

Having selected a surface representation in the form of a collection of nodes (and associated edges/faces), the objective function can be *discretized* by evaluating it only at the neighborhood centers (where the curvature estimates are best).

5.5.1 Discretized curve objective function

For a point-sampled curve, the integral in Equation 5.9 is approximated as an area-weighted sum of integrands over sample points:

$$\hat{E}_{\text{curve}} = \sum_{\text{c}\in\text{nodes}} (\mathbf{c}_{tt}(0))^2 a_c, \tag{5.11}$$

where $\mathbf{c}$ is the local curve function of Equation 5.7, and a_c the width of the curve segment associated with node c (nominally, 1/2 the distance to each neighbor). Because each of the local $\mathbf{c}$ reconstructions is quadratic, each $\mathbf{c}_{tt}$ is constant within its neighborhood. Interestingly, this means that one could just as well view the numerical integration as being exact over a collection of discontinuous quadratic curve elements, since $\int_0^a \ddot{\mathbf{c}}$ is simply $\ddot{\mathbf{c}}(0)\, a$ in this special case (this will also be true of the surface objective function, below, so that one may think of the surface as being a union of quadratic patches).

5.5.2 Gradient and Hessian of the curve objective function

Substituting (5.7) and evaluating the derivatives of the basis functions $\mathbf{b}(0)$ yields $\hat{E}_{\text{curve}}$ as a function of the node and its neighbors' positions $\mathbf{P}$:

$$\hat{E}_{\text{curve}} = \sum_{\text{i}\in\text{nodes}} (\mathbf{B}_3\mathbf{P})_i \cdot (\mathbf{B}_3\mathbf{P})_i\, a_i, \tag{5.12}$$

where $\mathbf{B}_3$ is the third row of the nodal neighborhood basis matrix, corresponding to the curve's second-order coefficients. We will be less concerned with evaluating $\hat{E}_{\text{curve}}$ than its gradient and Hessian with respect to $\mathbf{P}$, because the absolute magnitude of $\hat{E}_{\text{curve}}$ is not important — it is at a minimum when its *gradient* has vanished. As is customary, this objective function is linearized with respect to the metric information a by treating the a_i as constant, yielding an objective that is quadratic in $\bar{\mathbf{P}}$ (of course, a is re-evaluated whenever the parameterization changes). The resulting gradient and Hessian will be used to set up a quadratic approximation to the minimization problem, and ultimately solve a series of such problems to converge on the true minimum.

In actually computing the gradient and Hessian with respect to the elements of $\mathbf{P}$, a global "flattened" $\mathbf{P}$ (a single vector of concatenated x,y,and z values from every node in the mesh) will be needed, denoted $\bar{\mathbf{P}}$. This $\bar{\mathbf{P}}$ is the vector of independent variables whose values will be found in the course of minimizing the objective. If, for simplicity, there was only a single neighborhood ($N = 1$), the corresponding Hessian for $\hat{E}_{\text{curve}}$ would be a $3n \times 3n$ block-diagonal matrix, with the outer product $\mathbf{B}_3^T\mathbf{B}_3$ replicated once along the diagonal for each of x, y, and z, like so:

$$\hat{E}_{\text{curve}} = (\bar{\mathbf{P}^T}_x \bar{\mathbf{P}^T}_y \bar{\mathbf{P}^T}_z) \begin{pmatrix} \mathbf{B}_3 \otimes \mathbf{B}_3 & 0 & 0 \\ 0 & \mathbf{B}_3 \otimes \mathbf{B}_3 & 0 \\ 0 & 0 & \mathbf{B}_3 \otimes \mathbf{B}_3 \end{pmatrix} \begin{pmatrix} \bar{\mathbf{P}}_x \\ \bar{\mathbf{P}}_y \\ \bar{\mathbf{P}}_z \end{pmatrix} a \tag{5.13}$$

$$= \bar{\mathbf{P}^T}\mathbf{H}\bar{\mathbf{P}} \tag{5.14}$$

When there are many nodes ($N >> 1$), on the other hand, $\bar{\mathbf{P}}$ gets correspondingly longer, and while there will be more neighborhoods overall, the size n of a neighborhood doesn't typically increase. The Hessian for any particular neighborhood will be very sparse, with $3(n+1)^2$ nonzero entries scattered over the $3N \times 3N$ node coordinate array. The global Hessian is then the sum of these very sparse neighborhood Hessians. We will not attempt to write out an expression for the global $\mathbf{H}$, since we will never actually have to generate and store it as a monolithic matrix. Instead, the solver (below) will only require that one compute matrix-vector products with $\mathbf{H}$, and this can be done by summing the matrix-vector product with each neighborhood Hessian.

5.5.3 Discretized surface objective function

Analogously to the curve objective, for a triangulated surface the objective integral (Equation 2.10) is approximated as an area-weighted sum of integrands over sample points:

$$\hat{E}_{\text{surf}} = \sum_{\mathbf{s} \in \text{nodes}} \left((\mathbf{s}_{uu}(0,0) \cdot \mathbf{n})^2 + 2(\mathbf{s}_{uv}(0,0) \cdot \mathbf{n})^2 + (\mathbf{s}_{vv}(0,0) \cdot \mathbf{n})^2 \right) a, \tag{5.15}$$

where $\mathbf{s}(0,0)$ is the fitted surface function for each neighborhood (Equation 5.6) evaluated at its center node, $\mathbf{n}$ the fitted surface normal function, and a the neighborhood's associated area (nominally, 1/3 the area of each of the triangles in its parametric neighborhood).

5.5.4 Gradient and Hessian of the surface objective function

As with curves, the derivatives in the equation above are simply dot products with rows of $\mathbf{B}$:

$$\mathbf{s}_{uu}(0,0) = \mathbf{B}_3\mathbf{P}, \mathbf{s}_{uv}(0,0) = \mathbf{s}_{vu}(0,0) = \mathbf{B}_4\mathbf{P}, \mathbf{s}_{vv}(0,0) = \mathbf{B}_5\mathbf{P},$$

where $\mathbf{B}_j$ is the jth row of the neighborhood basis matrix fitted in Section 5.3. Substituting these yields $\hat{E}_{\text{surf}}$ as a function of $\mathbf{P}$:

$$\hat{E}_{\text{surf}} = \sum_{\text{i}\in\text{nodes}} \left((\mathbf{B}_3\mathbf{P}\cdot\mathbf{n})_i^2 + 2(\mathbf{B}_4\mathbf{P}\cdot\mathbf{n})_i^2 + (\mathbf{B}_5\mathbf{P}\cdot\mathbf{n})_i^2 \right) a_i. \tag{5.16}$$

The dependency of $\hat{E}_{\text{surf}}$ on the node positions is given by its gradient with respect to $\mathbf{P}$. $\hat{E}_{\text{surf}}$ is linearized with respect to the $\mathbf{n}_i$ and a_i (and implicitly, $\mathbf{G}$) by taking them to be constant when computing this gradient.

The effect of the linearization is easy to picture, if one considers only a neighborhood at a time. The dot-product against $\mathbf{n}$ means that the gradient is only sensitive to motion in the normal direction. Thus, the objective function treats the surface like a height-field with respect to the tangent plane at the neighborhood center. This suggests that one might reduce the number of free variables by allowing each sample to move only in the direction of its current normal. Instead solving directly for n new point positions, requiring that a system of $3n \times 3n$ equations be solved, the n normal offsets α from the current point positions are found, resulting in only an $n \times n$ system:

$$\hat{E}_{\text{surf}} = \sum_{\text{i}\in\text{nodes}} \left((\mathbf{B}_3(\mathbf{P}+\alpha\mathbf{N})\cdot\mathbf{n})^2 + 2(\mathbf{B}_4(\mathbf{P}+\alpha\mathbf{N})\cdot\mathbf{n})^2 + (\mathbf{B}_5(\mathbf{P}+\alpha\mathbf{N})\cdot\mathbf{n})^2 \right) a, \tag{5.17}$$

where α is a vector of offsets and $\mathbf{N}$ a list of neighbor normal vectors $\mathbf{n}_j$ (ordered just like $\mathbf{P}$). Thus, $\alpha\mathbf{N}$ is a list of *offset* vectors from the $\mathbf{P}$ computed by scaling each $\mathbf{n}_j$ by its corresponding α_j. It is these α_j that must be solved for, rather than the $\mathbf{P}$. To minimize $\hat{E}_{\text{surf}}$, compute a minimizing α using the current sample positions, normals, and areas; then add $\alpha\mathbf{N}$ to $\mathbf{P}$ to move the points to these new positions. To picture this more restricted situation, imagine the surface as a tent, with the samples as tent-poles arranged at various angles, propping it up to give it shape. This new $\hat{E}_{\text{surf}}$ controls shape by shortening or lengthening the tent-poles. After a length change, the poles will be re-oriented to point in the new surface normal directions. In our experience, this normal-offset scheme yields surfaces visually indistinguishable from those produced by the original unconstrained computation of $\mathbf{P}$, while requiring the solution of a much smaller system of equations. It also yields more stable behavior in conjunction with the mesh smoothing operations developed in the next chapter, because the shape optimization doesn't compete with our sample re-distribution process, which moves nodes tangent to the surface.

Noting that the linearizations with respect to $\mathbf{N}$, a, and $\mathbf{P}$ make $\hat{E}_{\text{surf}}$ a quadratic function of α, it should be possible to rearrange terms to put Equation 5.17 in the form $\alpha\mathbf{H}\alpha + \mathbf{g}\cdot\alpha + C$, for use in gradient and Hessian calculations. But standard vector notation has broken down for us in Equation 5.17 because of the way α scales $\mathbf{N}$, and rearrangement will only exacerbate the problem. With apologies in advance (there really seems to be no appealing way to continue the derivation), we will re-write the summand of Equation 5.17 using index notation[2], which will then allow us to re-arrange its terms unambiguously:

$$\hat{E}_{\text{surf}} = ((B_{3i}(P_{ij}+\mathcal{D}_{ikm}\alpha_k N_{mj})n_j)^2 + 2(B_{4i}(P_{ij}+\mathcal{D}_{ikm}\alpha_k N_{mj})n_j)^2 + (B_{5i}(P_{ij}+\mathcal{D}_{ikm}\alpha_k N_{mj})n_j))a, \tag{5.18}$$

[2] In index notation, an unsubscripted quantity is a scalar, one subscript denotes a vector, and two denote a matrix. Under the *summation convention*, the appearance of any index twice in a term implies summation, so that $M_{ij}v_j$ means $\sum_j M_{ij}v_j$, which is matrix M times vector v.

where the neighborhood index from the earlier summation has been dropped (all quantities are specific to the particular neighborhood being evaluated) and the diagonalizing constant $\mathcal{D}_{ijk} = 1$ if $i = j = k$, and 0 otherwise [3]. Now, let

$$(BB)_{ij} = B_{3i}B_{3j} + 2B_{4i}B_{4j} + B_{5i}B_{5j} \tag{5.19}$$

within each neighborhood. Some rearrangement of Equation 5.18 yields

$$\hat{E} = \sum_{\text{nodes}} n_j(N_{mj}\alpha_k\mathcal{D}_{ikm} + P_{ij})(BB)_{iq}(P_{qr} + \mathcal{D}_{qst}\alpha_s N_{tr})n_r\, a \tag{5.20}$$

$$= \sum_{\text{nodes}} \left(\begin{array}{c} \alpha_k(n_j N_{mj}\mathcal{D}_{ikm}(BB)_{iq}\mathcal{D}_{qst}N_{tr}n_r)\alpha_s \\ + \\ (2n_j P_{ij}(BB)_{iq}\mathcal{D}_{qst}N_{tr}n_r)\alpha_s \\ + \\ n_j P_{ij}(BB)_{iq}P_{qr}n_r \end{array} \right) a, \tag{5.21}$$

which is the desired form. The neighborhood Hessians and gradients of $\hat{E}_{\text{curve}}$ or $\hat{E}_{\text{surf}}$ can be read off as

$$H_k s = \frac{\partial^2 \hat{E}}{\partial\alpha_i\partial\alpha_j} = n_j N_{mj}\mathcal{D}_{ikm}(BB)_{iq}\mathcal{D}_{qst}N_{tr}n_r a, \tag{5.22}$$

$$g_i = \frac{\partial \hat{E}}{\partial\alpha_i} = H_{ij}\alpha_j + 2n_j P_{kj}(BB)_{kq}\mathcal{D}_{qit}N_{tr}n_r a \tag{5.23}$$

The global Hessian and gradients are simply the sums over all neighborhoods of these local versions. A word on evaluating terms like $(BB)_{iq}\mathcal{D}_{qst}N_{tr}n_r$: $\mathcal{D}$ is a notation aid, not meant to be used explicitly in computation. Instead, first evaluate the product $N_{tr}n_r$. The resulting vector is then used to scale the columns of $(BB)_{iq}$.

5.5.5 Geometric point and curve constraints

Point interpolation

Point interpolation constraints for a curve or surface are enforced by simply freezing the positions of their associated nodes during minimization. We might also have placed the point interpolation constraint on the interior of an edge or face, resulting in a more general linear constraint analogous to the point-constrained B-splines of [WW92]. But refinement is cheap and easy in our p.l. representation, so having to create a node for each constraint point is no hardship.

Of course, just because a node's position is frozen does not mean that the node disappears from the calculations altogether: a frozen node is no longer considered an independent variable, but it still contributes constant terms to the gradient. For curves, split $\bar{\mathbf{P}}$ into unconstrained and constrained parts $\bar{\mathbf{Q}}$ and $\bar{\mathbf{R}}$, which partitions $\mathbf{H}$ into blocks representing cross-terms between the constrained and unconstrained points:

$$\hat{E} = [\bar{\mathbf{Q}}^T\bar{\mathbf{R}}^T] \begin{bmatrix} \mathbf{H}^{QQ} & \mathbf{H}^{QR} \\ \mathbf{H}^{RQ} & \mathbf{H}^{RR} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{Q}} \\ \bar{\mathbf{R}} \end{bmatrix} \tag{5.24}$$

[3] $\mathcal{D}_{ijk}a_ib_jc_k = a_0b_0c_0 + a_1b_1c_1 + \cdots a_nb_nc_n$. $\mathcal{D}$ is a notational trick for constructing a diagonal scaling matrix from a vector (and exploiting the symmetries of this operation).

The gradient of $\hat{E}$ with respect to the active nodes is then

$$g = \mathbf{H}^{QQ}\bar{\mathbf{Q}} + 2\mathbf{H}^{QR}\bar{\mathbf{R}}. \tag{5.25}$$

For surfaces, because constrained α values are always fixed at 0, only the $\mathbf{H}^{QQ}$ terms need to be evaluated.

In upcoming chapters, where an automatic re-triangulation scheme is developed to run in parallel with the fairing computation, these point interpolation constraints will become sliding (nonparametric) interpolation constraints.

Curve and region interpolation

Curve and region interpolation constraints, in which an embedded curve or surface region is constrained and controlled separately from the faired surface, are implemented as collections of point constraints over the affected areas. Region controllers (described in Chapter 7) will be responsible for positioning these constraint points and controlling the mesh refinement within these constrained regions.

Continuity across control regions

Recall that a surface forced to interpolate a closed embedded control curve is represented as three disjoint mesh regions — the inside, the outside, and the embedded curve itself. The nodes in these disjoint regions do communicate with each other, through the fitted neighborhood quadratics. Thus, even though the nodes in the embedded curve have their positions fixed, these nodes' derivatives incorporate positional information from nodes on either side of the curve, and the continuity across this fixed curve is thus the same (G^2 in the limit) as across any unconstrained mesh edge. If a crease in the surface is desired along the embedded curve, the surface mesh may be split along the curve as discussed in Chapter 4, and the two independent surfaces forced to interpolate the same crease curve without sharing any other neighborhood information across their boundaries. The dihedral angle at which these surfaces meet may be controlled using a ribbon constraint (below).

Curve tangency at a point

In addition to constraining curve positions at selected points, we must be able to constrain a curve to be tangent to a given line at a given point. Suppose $\mathbf{l}(t) = \mathbf{x}_0 + t\mathbf{T}$, where $\mathbf{x}_0$ and $\mathbf{T}$ are a point and a unit vector, respectively. The curve $\mathbf{c}$ is forced to be tangent to $\mathbf{l}$ at $\mathbf{x}_0$ by constraining a point $\mathbf{c}_i$ to remain at $\mathbf{x}_0$ as above, and also enforcing the following linear constraint:

$$\dot{\mathbf{c}}_i(0) = \mathbf{T} \tag{5.26}$$

$$\mathbf{B}_2\mathbf{P}_i = \mathbf{T} \tag{5.27}$$

($\mathbf{B}_2$ is the second row of $\mathbf{B}$, as in previous sections). This constraint cannot be properly enforced by freezing node positions, as was done with position constraints; instead, the technique of Lagrange multipliers[Str86] is used, to be described below.

Surface boundary ribbons

In addition to directly enforced point constraints, we consider another more complicated constraint on surface boundaries: the *ribbon* constraint, which controls cross-boundary tangents along a surface boundary curve. Recall from Section 5.3 that our local fitting scheme aligns surface boundaries in the parametric u direction. Thus, enforcing a cross-boundary tangency constraint at a point amounts to enforcing

$$\mathbf{s}_v(0,0) = \mathbf{R} \tag{5.28}$$

$$\mathbf{B}_2(\mathbf{P} + \alpha\mathbf{N}) = \mathbf{R} \tag{5.29}$$

at the boundary mesh node ($\mathbf{B}_2$ is the second row of $\mathbf{B}$). This can be turned into a ribbon if tangent vectors R are given for each point on the boundary curve (a cross-boundary tangent function). Again, such linear constraints will be enforced through the use of Lagrange multipliers.

5.6 Minimizing the objective functions

The curve and surface objective functions $\mathcal{E}_{\text{curve}}$and $\mathcal{E}_{\text{surf}}$are nonlinear functions of the mesh node positions. A minimizing set of node positions will be computed by solving a sequence of quadratic approximations to the true nonlinear problem (*quadratic steps* in a sequential quadratic program[GMW81]).

The quadratic sub-problems are minimizations of the linearized $\hat{E}_{\text{curve}}$ and $\hat{E}_{\text{surf}}$ objective functions just described. Given mesh and curve shapes, we parameterize node neighborhoods, fit a quadratic function to each node neighborhood, then take a step towards the minimum of the linearized objective functions evaluated with respect to the current fitted neighborhoods. The idea behind the iteration is that near the nonlinear objective function's true minimum it can be well-approximated by a quadratic function, so that minimizing this function (which is easy) also minimizes the nonlinear version (which is hard). The assumption is that near the minimum, control parameter gradients are small, and therefore the ignored gradient terms due to linearization will not matter so much. This all depends on beginning the iterations with a mesh shape that is "reasonable", or the iterations will diverge and not home in on the minimum. This issue of good initial guesses is discussed further in Chapter 7.

A quadratic step involves moving towards the minimum for the quadratic objective functions $\hat{E}_{\text{curve}}$ or $\hat{E}_{\text{surf}}$. The minimization, subject to point constraints (Equation 5.24), is performed by solving for a value of $\mathbf{Q}$ that zeros the gradient, which means solving the linear system $\mathbf{H}^{QQ}\bar{\mathbf{Q}} = -2\mathbf{H}^{QR}\bar{\mathbf{R}}$. There may be additional linear constraints that must be maintained, such as those contributed by tangency requirements. To enforce these, concatenate them as rows of a single matrix equation $\mathbf{C} = \mathbf{t}$, where $\mathbf{t}$ is the vector of fixed constraint values. Then use the technique of Lagrange multipliers[Str86]: an additional free variable $\mathbf{y}_i$ is added for each of the constraints, and one solves the augmented system

$$\begin{pmatrix} \mathbf{H}^{QQ} & \mathbf{C}^T \\ \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \bar{\mathbf{Q}} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} 2\mathbf{H}^{QR}\bar{\mathbf{R}} \\ t \end{pmatrix}$$