29 Computer-Aided Geometric Design Techniques for Surface Grid Generation

Bernd Hamann

Brian A. Jean

Anshuman Razdan

29.1 Introduction

This chapter focuses on three computer-aided geometric design (CAGD) techniques that are often needed to "prepare" a complex geometry for grid generation. Standard grid generation methods, as discussed in [George 1991], [Knupp and Steinberg 1993], and [Thompson et al., 1985], assume that parametric surfaces are well parametrized and free of undesired discontinuities.We describe CAGD techniques that are extremely helpful for the preparation of complex geometries for the grid generation process.

0-8493-2687-7/99/$0.00+$.50

29.1.1 Surface Refinement and Reparametrization

One problem that has plagued most grid generation systems is that poorly parametrized surfaces create a poorly distributed grid. This is due to the fact that the grid distributions are performed in the parametric domain and then mapped back to physical space. It is desirable that the parametrization reflect the geometry of the surface in physical space, i.e., the parametrization should mimic the surface in physical space. The more a distribution of points in parametric space resembles the corresponding distribution of points in physical space, the better the parametrization is. Imagine using a uniform parametrization on a sample of points that are not distributed evenly; if one distributes points evenly in parameter space, they are not uniform in physical space. There are ways to achieve a uniform distribution in physical space, but most approaches are based on iterative procedures. To eliminate the need for such iteration procedures, a *chord length* parametrization can be used — it best represents the surface in physical space.

29.1.2 Approximation of Discontinuous Geometries

Grid generation is concerned with discretizing surfaces and surrounding volumes in three-dimensional (3D) space — in this context, it is important that a given geometry is continuous. Before one can generate grids for geometries containing discontinuities, one must approximate the geometry by a set of surface patches which are continuous. The most common problems are gaps between neighbor patches, surfaces with undesired intersections, and overlapping surfaces. We have developed an interactive technique that can be used to approximate a faulty, i.e., discontinuous, geometry by a continuous one. One obtains approximating surface patches by projecting local approximants, *Coons patches*, onto the discontinuous geometry, see [Coons 1974]. Each approximating surface patch is constructed by specifying four boundary curves, computing a Coons patch interpolating the four curves, and projecting the Coons patch onto the given geometry. In the end, one has replaced the entire geometry by a new set of continuous surface patches.

29.1.3 Surface–Surface Intersection

Accurate computation of surface–surface intersection (SSI) curves is essential in many engineering applications including numerical grid generation. SSI curves represent important features that must be captured by the grid. These curves are typically used to trim surfaces; for example, the location where an airplane wing meets the fuselage would be given in terms of the intersection of the wing and the fuselage. Often, geometries defined in terms of standard data exchange formats either do not contain the needed intersection curves, or the curves given in the file may not be in a form that is suitable for the grid generator. A good SSI algorithm must be capable of treating analytic surfaces (e.g., cylinders, cones, etc.), parametric surfaces (e.g., NURBS — nonuniform rational B-splines), surfaces described by discrete data points (e.g., resulting from stereo lithography, Plot3D surface grids, etc.), and combinations of these types. Accuracy must be good enough for packing of grid points allowing high-aspect-ratio cells near the intersection curves, which typically requires that the curves be accurate to at least 10^{-6} units. A good method should be robust and should require a minimum of user input. A user should have to specify only the surfaces to be intersected and the requested tolerance. Use in an interactive environment requires that the method be reasonably fast, i.e., the solution of all but the most demanding problems should take only a few seconds on a state-of-the-art workstation.

29.2 Surface Refinement and Reparametrization — Underlying Principles and Best Practices

The relationship between the parametrization and the control point net of a NURBS surface reflects the relationship between parameter space and range. Each Bézier segment of a curve, for example, may have a normalized local parametrization (t, $0 \leq t \leq 1.0$). However, there exists a global parametrization that determines the relationship between each segment and the whole curve. The requirements for C^2 continuity between two segments are that the second derivatives at the common break point should match — "from the left and right." The notion of C^r, $r \geq 1$ depends on the interplay between the domain and range configurations. In other words, the first- and the second-order derivatives are dependent on the global parametrization of a NURBS curve. As a rule of thumb, a better curve is obtained if the geometry (range) of the NURBS curve is incorporated into the parametrization. Several parametrization schemes exist, such as *uniform, chord length, centripetal,* and one due to Nielson and Foley [1989]. Each scheme has some favorable aspects; see [Foley 1986, 1987] for a detailed discussion.

In the context of grid generation, the grid can be *smoothed* to correct problems resulting from bad underlying parametrization, but this procedure is rather time-consuming. The other alternative is to reparametrize the surface. The process does not change the geometry in physical space. Without going into the detail, we state that reparametrization is independent of the degree of the rational-polynomial basis functions of NURBS, see [Farin 1995] (see also Chapter 28). The reparametrization will then create a "smooth" parametric domain that will promote high quality grids without jeopardizing accuracy.

The goal then is to *refine* a given, poorly parametrized surface, i.e., to construct a surface that is *chord length* parametrized. A surface $\mathbf{s}(u,v)$ with knot sequence $\{u_0, \ldots, u_{L+2m-2}\}$ and $\{v_0, \ldots, v_{M+2m-2}\}$ is said to be *chord length* parametrized if it has the following properties:

$$\frac{\left\|\mathbf{s}\left(u_{i+1}, v_j\right) - \mathbf{s}\left(u_i, v_j\right)\right\|}{\left\|\mathbf{s}\left(u_{i-1}, v_j\right) - \mathbf{s}\left(u_i, v_j\right)\right\|} \approx \frac{\Delta u_i}{\Delta u_{i-1}} \tag{29.1}$$

and

$$\frac{\left\|\mathbf{s}\left(u_i, v_{j+1}\right) - \mathbf{s}\left(u_i, v_j\right)\right\|}{\left\|\mathbf{s}\left(u_i, v_{j-1}\right) - \mathbf{s}\left(u_i, v_j\right)\right\|} \approx \frac{\Delta v_i}{\Delta v_{i-1}} \tag{29.2}$$

where

$$\Delta u_i = u_{i+1} - u_i, \quad \Delta v_i = v_{i+1} - v_i,$$

and $\| \; \|$ denotes the Euclidean norm, see [Farin 1997].

For interrogation and analysis of a surface, it is desirable that the parametrization and control points reflect the above situation. This is to enable the surface evaluation parameter values to be used as input for subsequent analysis. The question then is, *Can the surface be redefined, within a given tolerance, such that the parametrization is in tune with the geometry of the surface?* In other words, given a poorly parametrized NURBS surface, can one construct a redefined NURBS surface that approximates the given surface within a given tolerance such that it has the properties of a chord length parametrization.

Some of the related research in the areas of reparametrization and curve and surface approximation is reviewed in the following. Previous work related to this research can be categorized in two areas, the first being reparametrization and the second being curve and surface approximation/interpolation methods.

Some work has been done in the area of reparametrization of curves. In [Fuhr and Kallay 1982], a method is described for interpolating a monotone data sequence with a C^1 monotone rational *B-spline* curve of degree 1. If the original curve C and the reparametrization function f are rational *B-splines,* then the reparametrized curve $\tilde{C} = C \, C \, f$ is also a rational *B-spline.* The degree of $\tilde{C}$ is the product of the degrees of C and f. This results in a C^1-continuous spline. We need to achieve C^2 continuity. The algorithm mentioned above ensures that the degree is not raised. This is useful in coming up with a common parametrization for opposite boundary curves on a surface.

In [Crampin et al. 1985] an algorithm is described to transmit a curve by sending discrete points off the original curve, such that the curve can be regenerated at the other end. In order to interpolate a curve effectively, few points should be placed where the radius of curvature is large, but many where it is small.

Yu and Soni [1995] use reparametrization to create grids with different parameter distributions. The reparametrization in the curve case is achieved as follows: Let us consider a NURBS curve with resolution n (number of points), and let

1. $s_1(i)$, $i = 1, \ldots, n$, be the parametric values associated with the desired distribution of the curve in physical space, and
2. $s_2(i)$, $i = 1, \ldots, n$, be the normalized chord length of the curve evaluated at parametric values $s_1(i)$, $\mathrm{i} = 1, \ldots, n$.

The $s_2(i)$ values are known, and the $s_1(i)$ values are to be determined such that $\|s_2(i) - s_1(i)\|$ is minimized for all $i = 1, \ldots, n$. This is accomplished by an iterative process. The initial values of $s_1(i)$ are set to be the same as those at which $s_2(i)$ and $s_3(i)$ are evaluated. If the absolute difference $s_2(i) - s_3(i)$ is smaller than a certain tolerance, $s_1(i)$ is set as the desired parametric value. If the difference of $s_2(i) - s_3(i)$ is negative and the absolute value of this difference is greater than the tolerance, $s_1(i)$ should be shifted to a value between $s_1(i-1)$ and $s_1(i)$. The same strategy is applied to the case where $s_2(i) - s_3(i)$ is positive. In this case, the value of $s_1(i)$ should be shifted to a value between $s_1(i)$ and $s_1(i+1)$. The algorithm is further extended to deal with reparametrization of surfaces. Nevertheless, this approach cannot be used directly for the reparametrization of surfaces, it leaves many questions open.

Kim [1993] has attempted to come up with knot placement for NURBS interpolation. He plots the distance between the interpolation points as a monotonically increasing function $f(s)$ over its parametrization. The parametrization is obtained from one of the several methods commonly used. The function can be piecewise linear, piecewise rational–quadratic, or piecewise linear–rational B-spline interpolation. Knot placement is done by dividing the function space into equal number of segments and projecting the division onto the parametric space. This is then used for determining the parametrization.

29.2.1 Approaches to Solving the Problem

Although many different approaches may be applied to solve the problem at hand, the following two are considered:

1. Modify the control parameters of the given surface, with only minimal changes to the surface, i.e., change weights, control points, etc., with the result that the surface exhibits the property of *chord length* parametrization.
2. Construct a new NURBS surface that approximates the given surface, within a given tolerance, such that the surface is chord length parametrized.

29.2.2 Modifying the Existing Surface

There are four design parameters available in the NURBS case that control its behavior (it is assumed that the knot sequence does not have any multiplicities except at the ends); these are degree, control points, weights associated with the control points (we will only deal with the case $w_i > 0$), and parametrization.

Raising (or lowering) the degree does not affect the parametrization and therefore is a non-issue in our case. However, if one were to represent a NURBS curve with its approximation, the approximating polynomial should be at least a cubic. This is due to the fact that cubics are the lowest degree which can represent true space curves.

Modifying the control points modifies the surface itself. It is very difficult to predict the behavior of the surface when its control points are modified. Let us consider the curve case. Moving a control point means affecting (degree – 1) segments on each side. In order to not change the curve itself, the affected neighboring segments would also have to be changed (by moving their control points). This can start a "chain reaction" and convergence might be a problem.

Changing the weights is similar to changing the control points. However, in conjunction with the parametrization, it is possible to keep the curve or surface the same. Thus, it would be a matter of finding the new parametrization (the desired one), and we could possibly change the poorly parametrized curve to a chord length parametrized one without changing the curve itself. Here, the problem is to find the desired knot sequences themselves. This approach, although theoretically appealing, requires as input something that is not known. This first approach, though ideal, does not always result in a convergent solution.

29.2.3 The Surface Approximation Scheme

The second approach is to find another surface as close as possible to the original surface. Let ϵ_{max} be the value which indicates the maximum Euclidean distance the two surfaces are apart from each other. If $\mathbf{s}(u, v)$ is the given surface with knot sequence $\{u_0, \ldots, u_M\}$, $\{v_0, \ldots, v_N\}$, and $\mathbf{r}(u,v)$ with knot sequence $\{u_0, \ldots, u_K\}$, $\{v_0, \ldots, v_L\}$ is the approximation to the surface, then we want

$$\max_{u_i, v_j, u_k, v_l} \left\{ \min \left\{ \left\| \mathbf{s}\left(u_i, v_j\right) - \mathbf{r}\left(u_k, v_l\right) \right\| \right\} \right\} < \varepsilon, \tag{29.3}$$

where $\mathbf{r}(u_k, v_l)$ is the closest point on $\mathbf{r}(u, v)$ for a given point $\mathbf{s}(u_i, v_j)$ on $\mathbf{s}(u, v)$ and ε is the ϵ_{max} bound placed on the *healing* process.

This approach is used in Razdan [1995] to solve the problem at hand. The approximation is based on the assumption that adequate points can be found on the surface, such that when an interpolating surface is passed through them, the resulting surface will be very close to the original surface. The problem then reduces to finding these interpolation points. If, however, the number of points is insufficient, then the error estimation process identifies the point $\mathbf{s}(u_i, v_j)$ on $\mathbf{s}(u, v)$ where the maximum deviation, ϵ_{max}, occurs between $\mathbf{s}$ and $\mathbf{r}$. This information can be used to insert a knot in surface $\mathbf{r}$ such that $\mathbf{r}$ is now forced to interpolate to $\mathbf{s}(u_i, v_j)$.

The construction of the new surface is a two-step process. First, the four boundary curves are determined, then the interior is filled. The reason for tackling the boundary curves first is twofold. The boundary curves provide the spatial bounds to the *filling process*. Second, it works out well to fill using the *outside-in* approach. All computations are based on how well the surface is discretized. We have found that surface evaluation at a density of 10 × 10 points per knot segment is sufficient.

29.2.4 Boundary Curve Approximation

Approximation of the boundary curves is the first step towards approximating the surface. Each boundary curve is treated individually. The steps for approximating a NURBS curve are:

1. Estimate the number of interpolation points needed.
2. Find interpolation points on the given curve (while keeping such points to a reasonable number).
3. Pass a C^2-continuous interpolating NURBS curve through these points.

The technique for choosing interpolation points uses arc length and curvature distribution characteristics of the given curve. It also uses the *adaptive knot selection* scheme to properly place the knots on the interpolating curve. Details on how this is done can be found in [Razdan 1995]. The underlying principle is to capture as much of the geometric properties of the original curve as possible while trying to keep the number of interpolation points to a minimum. Figure 29.1 is an example of a NURBS curve and its approximation using this technique.

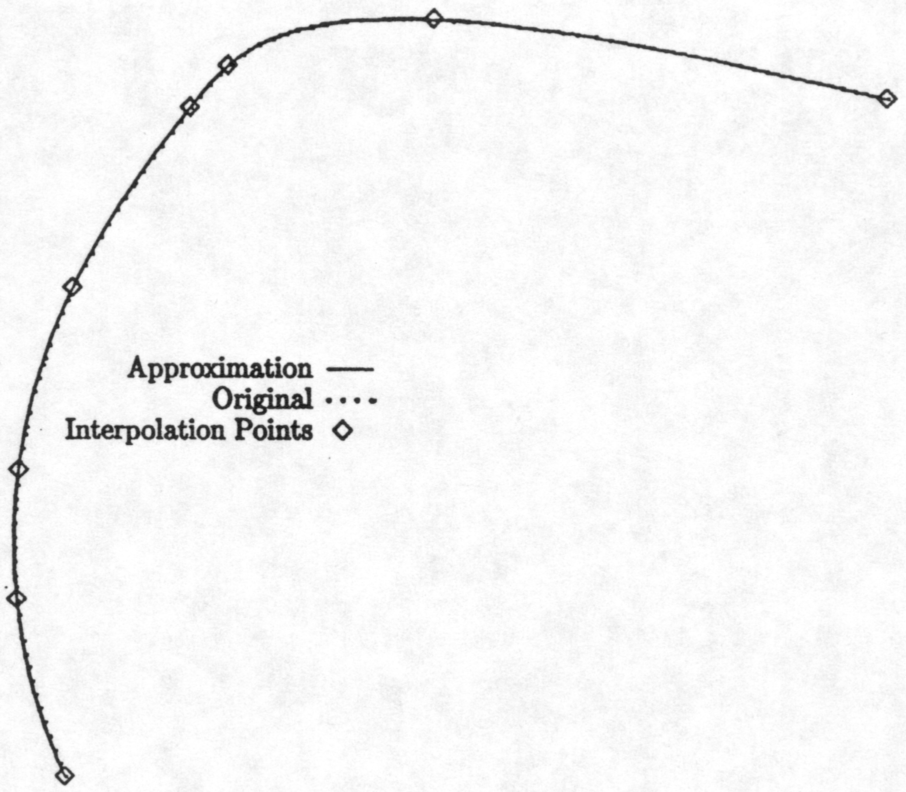

FIGURE 29. 1 NURBS curve interpolation using arc length and curvature.

29.2.5 Finding an Interpolating Surface

The next step in the process is to combine information (parametrization) of opposite boundary curves into one. Although the interpolation points required to describe each of the two boundary curves independently are available, the distribution of these points will not, in general, be satisfactorily represented by a common knot sequence (parametrization). This is due to the fact that the distributions of points in each set depends on the individual curves' curvature and arc length distributions. Choosing a knot sequence of either one of the boundary curves will result in the same initial problem. However, if somehow both curves and the interior surface did have the same distribution of interpolation points, a single parametrization could be used without a problem. But at this time the interior interpolation points are not *fixed.* This is dealt with as follows. First, a *reconciliation* process of the opposite boundary curves is performed to resolve the inequitable interpolation points distribution on the two boundary curves. This ensures that the knot sequences computed after the reconciliation step of the opposite boundary curves will be the same. Two knot sequences result, one for each parametric direction of the approximated surface. Second, the interior interpolation points on the surface are located such that they satisfy the parametrizations in both directions that resulted from the reconciliation process.

We describe briefly the *reconciliation* process between boundary curves. The distance (arc length) between neighboring interpolation points of one boundary curve is computed and tabulated. The same is done for its counterpart, the opposite boundary curve. Next, distances in each set are represented as the fraction of the total arc length of the respective curves. Once the two sets are compiled, they are *reconciled.* For every point in one set, a corresponding point is sought in the second set (and vice-versa) that is the same fraction of distance away from the starting end of the curve it belongs to. If there is no such point within a tolerance, then an auxiliary point is inserted into the set that does not have the point. At the end of this process, both sets have points that are similarly distributed along the length of the original boundary curves. Similarity in distribution means that the ratio of distances between the neighboring points is similar in the two sets. In other words, we have inserted auxiliary knots into the curves so that both curves have the right distribution of knots or points for interpolation. This in turn is nothing but chord length parametrization. This process is applied to the other set of opposite boundary curves.

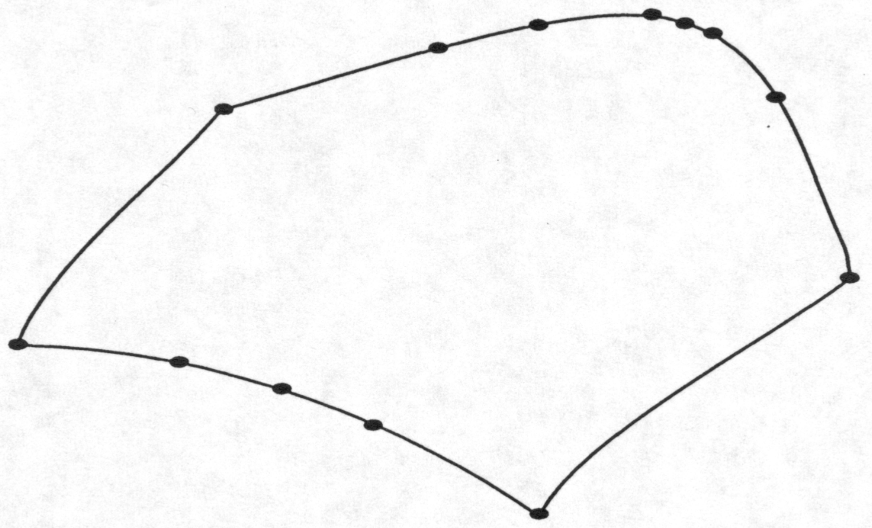

FIGURE 29.2 Boundary interpolation points before reconciliation.

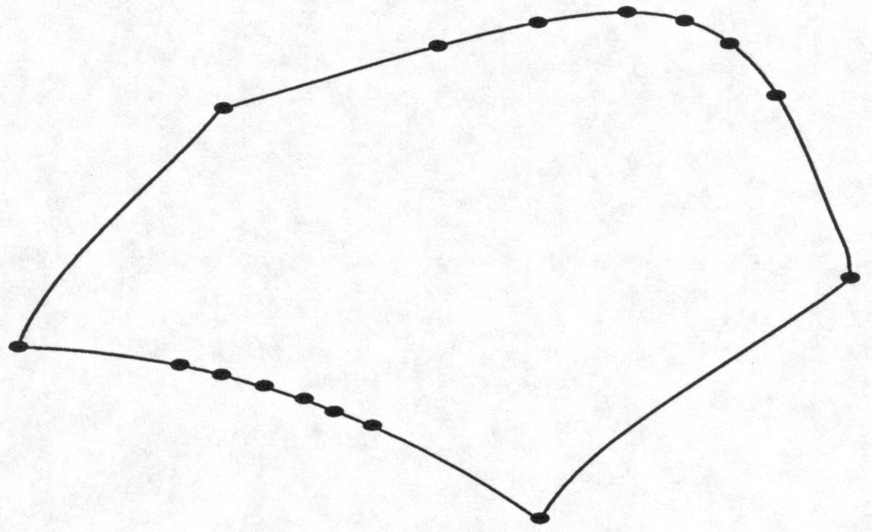

FIGURE 29.3 Boundary interpolation points after reconciliation.

The set of interpolation points is now fixed for all four boundary curves. Figures 29.2 and 29.3 show two sets of interpolation points before and after the reconciliation process. This process of adaptively generating knot sequences based on curvature information and arc length (chord length) is called the *RCA parametrization* (reconciled curvature arc length parametrization).

29.2.6 Finding Interior Interpolation Points

Once the outer framework of the boundary curves is accomplished, the interior of the surface is constructed. This is an iterative process. The parameter values at which the points on the boundary curves will be interpolated are marked on the domain rectangle of the original surface. The corresponding points on the opposite edges are joined with straight lines in the domain rectangle. The intersections of these lines provide (u,v) values in the parameter space of the original surface. This leads to an initial guess for the internal points of the new surface. In Figure 29.4, these points are marked as (1,1), (1,2), (2,1), and (2,2). As is evident from the figure, these points do not reflect the parametrization of the surface. For example, let $u_0 = 0.0$, $u_1 = 0.33$, and $u_2 = 0.66$. In general, point (1,1) will not be half the distance between points (1,0) and (1,2). In an ideal situation the process would stop here. However, for a poorly parametrized surface, this is the starting point for the iterative process.

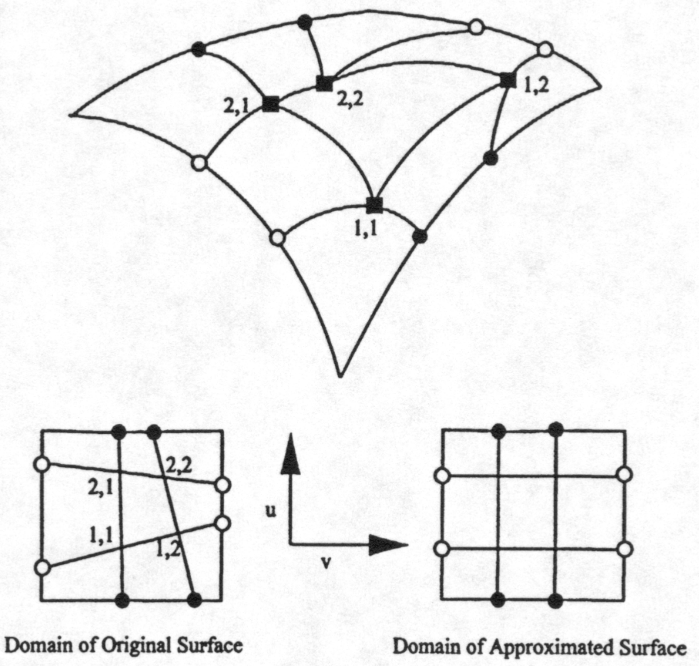

FIGURE 29.4 Surface and its domain rectangle.

We describe an algorithm to find the interior points. As stated above, the first guess of internal interpolation points will probably not satisfy the chosen parametrization. The algorithm iteratively moves each interior point $\mathbf{x}_{i,j}$ a finite distance in the domain and evaluates its relationship (distance) with its immediate four neighbors, $\mathbf{x}_{i-1,j}$, $\mathbf{x}_{i+1,j}$, $\mathbf{x}_{i,j-1}$, and $\mathbf{x}_{i,j+1}$, with respect to the new parametrization. It attempts to find the local minimum for placing this point. The points are always moved in the domain. This is important as it is guaranteed that the corresponding point in the range will always lie on the given surface. The evaluation, whether a particular choice (point location on the surface) is *good* or *bad*, is done based on a *penalty factor.* A high penalty factor means "not good." The penalty factors of all the interior points are computed and sorted in descending order. The point with highest penalty factor is tackled first, since it is most likely to be moved. The algorithm keeps track of points moved in an iteration in a two-dimensional array. In the iterative process, a point is a candidate for relocation if any of its four neighbors have moved since the last iteration. In the case when none of the four neighbors have moved, then local conditions have not changed and repeating the process will not improve the situation. On the other hand, if the local conditions have changed, i.e., one or more of the neighbors has moved since the last iteration, then it is likely that the current point is not the optimum point any more. Thus, it makes sense to apply the algorithm again. The iterative process is terminated when all the points occupy optimum positions. Figure 29.5 shows an example *before* and *after* this procedure is applied.

29.3 Approximation of Discontinuous Geometries — Underlying Principles and Practices

29.3.1 The Algorithm and References

The essential procedure used to approximate a geometry is the construction of a single local approximant. This procedure consists of these steps:

1. Creating four (or selecting four existing) curves as boundary curves for an initial local approximant (Coons patch).
2. Constructing a bilinear Coons patch from the four boundary curves.

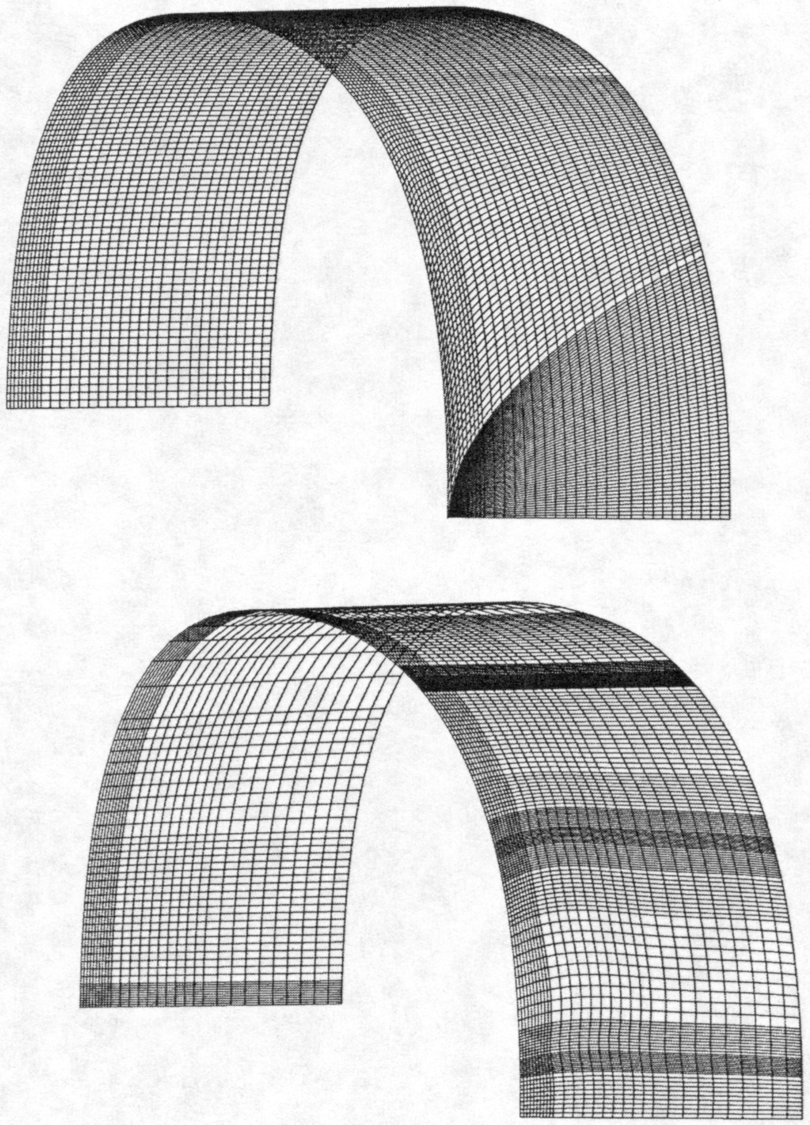

FIGURE 29.5 Surface before and after *healing*.

3. Projecting a curvilinear grid on the Coons patch onto the original geometry.
4. Determining "artificial projections" for those points in the curvilinear mesh that cannot be projected — due to possible gaps in the original surface.
5. Interpolating the points resulting from steps 3 and 4.

One has to perform step 1 interactively, while all other steps can be performed without user interaction. The local surface approximant obtained as the result of this procedure is a bicubic B-spline surface, which is guaranteed to lie within a certain distance of the original surfaces. The distance measure is based on shortest (perpendicular) distances between points on an approximant and the original surfaces. We compute this distance measure only in regions where there is a "clear" correspondence between an approximant and the original surfaces and do not compute it for those parts of an approximant covering a discontinuity.

Once all local approximants are determined and their topology (connectivity) is known, a final step ensures that the overall resulting approximation is continuous by enforcing continuity along shared boundary curves of the local approximants.

The methods that we rely on to approximate a discontinuous geometry are covered in great detail in the literature dealing with CAGD methods for curves and surfaces. References include [Farin 1995, 1997], [Faux and Pratt 1979], [Piegl 1991a, 1991b], and [Piegl and Tiller 1996].

29.3.2 Computing the Initial Coons Patch

The initial local approximant is used to smooth rough data, guide the choices of interpolation points, and serve as a reference for filling in gaps. A user has to specify four continuous curves whose endpoints meet to form a single closed curve — the boundary of a Coons patch. The four boundary curves can span across multiple original surface patches; they can even be above or below the given geometry.

In order to obtain a reasonable surface grid for the Coons patch implied by the four boundary curves, we use a *discrete Coons patch* construction. First, we compute points on the boundary curves distributed uniformly with respect to arc length. We then associate parameter values $(u_{I,0},v_{I,0})$, $(u_{I,N},v_{I,N})$, $(u_{0,J},v_{0,J})$, and $(u_{M,J},v_{M,J})$, $I=0, \ldots, M$, $J=0, \ldots, N$, defining the uniformly distributed points on the boundary curves in 3D Euclidean space. The points $\mathbf{x}_{I,J}=(x_{I,J}, y_{I,J}, z_{I,J})$ on the discrete Coons patch are thus defined as

$$
\begin{aligned}
\mathbf{x}_{I,J} = {} & \left[(1-u_{I,J})\; u_{I,J}\right]\begin{bmatrix}\mathbf{x}_{0,J}\\ \mathbf{x}_{M,J}\end{bmatrix} + \left[\mathbf{x}_{I,0}\;\mathbf{x}_{I,N}\right]\begin{bmatrix}(1-v_{I,J})\\ v_{I,J}\end{bmatrix} \\
& -\left[(1-u_{I,J})\; u_{I,J}\right]\begin{bmatrix}\mathbf{x}_{0,0} & \mathbf{x}_{0,N}\\ \mathbf{x}_{M,0} & \mathbf{x}_{M,N}\end{bmatrix}\begin{bmatrix}(1-v_{I,J})\\ v_{I,J}\end{bmatrix},
\end{aligned}
\tag{29.4}
$$

where $u_{I,J}$, $J=0, \ldots, N$, varies linearly between $u_{I,0}$ and $u_{I,N}$ and $v_{I,J}$, $I=0, \ldots, M$, varies linearly between $v_{0,J}$ and $v_{M,J}$, respectively. In general, the points $\mathbf{x}_{I,J}$ do not lie on the given surface patches.

29.3.3 Projecting the Coons Patch onto the Original Surfaces

In order to project each point $\mathbf{x}_{I,J}$ on the Coons patch onto the original surfaces, one must know the approximate surface normal at $\mathbf{x}_{I,J}$, which is used as projection direction. We approximate the unit normal vector at $\mathbf{x}_{I,J}$ by

$$
\mathbf{n}_{I,J} \approx \frac{(\mathbf{x}_{I+1,J}-\mathbf{x}_{I-1,J})\times(\mathbf{x}_{I,J+1}-\mathbf{x}_{I,J-1})}{\left\|(\mathbf{x}_{I+1,J}-\mathbf{x}_{I-1,J})\times(\mathbf{x}_{I,J+1}-\mathbf{x}_{I,J-1})\right\|}.
\tag{29.5}
$$

The points $\mathbf{x}_{I,J}$, their associated normal vectors $\mathbf{n}_{I,J}$, and an absolute offset distance d define points on an *upper* and a *lower offset surface* of the initial Coons approximant. The points on the upper offset surface are denoted by $\mathbf{a}_{I,J}$ and the ones on the lower offset surface by $\mathbf{b}_{I,J}$:

$$
\mathbf{a}_{I,J}=\mathbf{x}_{I,J}+d\,\mathbf{n}_{I,J} \quad \text{and} \quad \mathbf{b}_{I,J}=\mathbf{x}_{I,J}-d\,\mathbf{n}_{I,J}.
\tag{29.6}
$$

We relate the offset distance d to the extension of the Coons patch by setting

$$
d=\tfrac{1}{8}\left(\left\|\mathbf{x}_{M,0}-\mathbf{x}_{0,0}\right\|+\left\|\mathbf{x}_{M,N}-\mathbf{x}_{M,0}\right\|+\left\|\mathbf{x}_{0,N}-\mathbf{x}_{M,N}\right\|+\left\|\mathbf{x}_{0,0}-\mathbf{x}_{0,N}\right\|\right).
\tag{29.7}
$$

The choice of d is very important, since it determines the set of original surfaces to be considered for the final local approximation. It is not clear at this point what is the best value for d given an arbitrary geometry. The offset surface construction is shown in Figure 29.6.

It is assumed that the convex set S defined by all the points $\mathbf{a}_{I,J}$ and $\mathbf{b}_{I,J}$ contains the original surfaces that must be considered by the local approximation procedure. The convex hull of S is approximated by computing the 3D bounding box for all the points $\mathbf{a}_{I,J}$ and $\mathbf{b}_{I,J}$. Original surfaces are considered for the local approximation procedure only if they lie partially inside this bounding box. The surfaces lying inside the bounding box are evaluated, using some predefined resolutions, and the resulting point sets

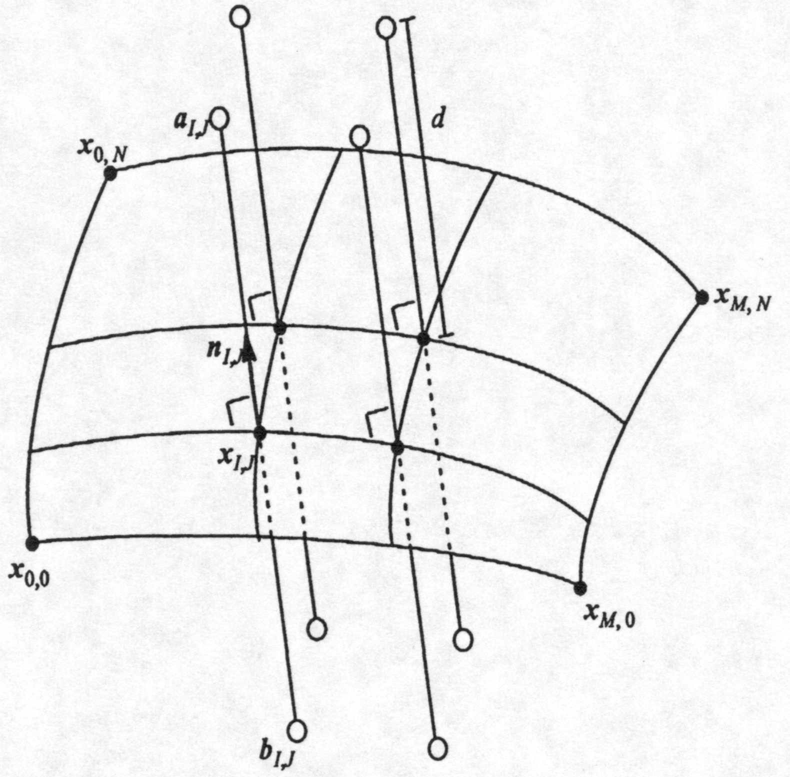

FIGURE 29.6 Local offset surface construction.

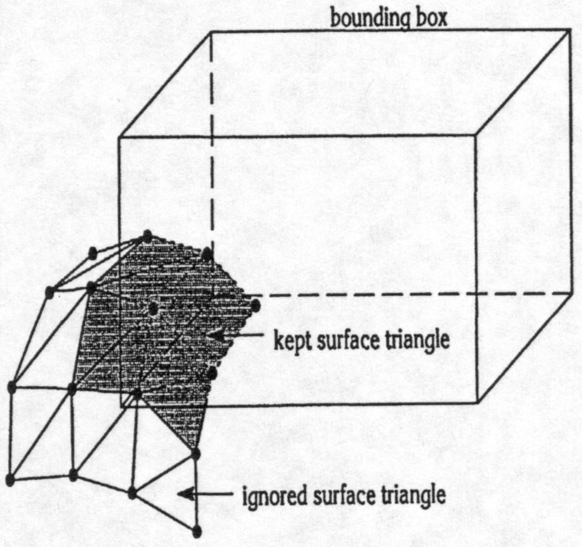

FIGURE 29.7 Clipping original surfaces and surface triangles.

are triangulated. The resulting triangles are then also clipped against the same bounding box — one needs to consider only those triangles lying inside the bounding box when projecting a point $\mathbf{x}_{I,J}$ onto the original surfaces. This is illustrated in Figure 29.7.

Next, each point $\mathbf{x}_{I,J}$ is projected in direction of $\mathbf{n}_{I,J}$ onto the triangles inside the bounding box. The projection of $\mathbf{x}_{I,J}$ must satisfy the condition that it lie between $\mathbf{a}_{I,J}$ and $\mathbf{b}_{I,J}$. In general, it is possible to obtain zero, one, or multiple projections for each point $\mathbf{x}_{I,J}$. If more than one intersection is found, the point closest to the point $\mathbf{x}_{I,J}$ is identified and used in the subsequent steps. If no intersection is found, a bivariate *scattered data approximation* method will be used later to derive "artificial projections."

If the parametric representation of the original surfaces is known the projections, computed as projections onto triangles in a surface triangulation, can be mapped to points that lie exactly on the given surfaces. A projection obtained from intersecting a line segment $\mathbf{a}_{I,J}\mathbf{b}_{I,J}$ and a surface triangle can be expressed as a barycentric combination of the vertices of this triangle. Let $\mathbf{p}_{I,J}$, be a projection, and let $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ be the vertices of the triangle containing the projection. We can express the projection in barycentric form as

$$\mathbf{p}_{I,J} = \bar{u}_1\mathbf{p}_1 + \bar{u}_2\mathbf{p}_2 + \bar{u}_3\mathbf{p}_3 \tag{29.8}$$

and can use the barycentric coordinates in this expression to compute the parameter tuple

$$(\bar{u},\bar{v}) = \bar{u}_1(u_1,v_1) + \bar{u}_2(u_2,v_2) + \bar{u}_3(u_3,v_3) \tag{29.9}$$

where (u_i,v_i) is the parameter tuple of vertex $\mathbf{p}_i$ We can now evaluate the associated original parametric surface $\mathbf{s}(u,v)$ using the parameter tuple $(\bar{u},\bar{v})$ and replace $\mathbf{p}_{I,J}$ by $\mathbf{s}(\bar{u},\bar{v})$. We will denote the points obtained by this "map-onto-real-surface step'" by $\mathbf{y}_{I,J}$. (If the parametric representation of the original surfaces is not known, we simply use the intersections with the surface triangulations as final approximation conditions $\mathbf{y}_{I,J}$.)

29.3.4 Computing Additional Approximation Conditions

Due to existing discontinuities, gaps, in the original surfaces the projection strategy might not yield any intersection points for certain line segments $\overline{\mathbf{a}_{I,J}\mathbf{b}_{I,J}}$. We must estimate "artificial projections" to obtain a complete $(M + 1) \times (N + 1)$ array of points required later in the construction of a local B-spline approximant. We use a bivariate scattered data approximation technique, called *Hardy's reciprocal multiquadric method*, see [Franke 1982].

Each intersection point $\mathbf{p}_{I,J}$, obtained by intersecting line segment $\overline{\mathbf{a}_{I,J}\mathbf{b}_{I,J}}$ with the surface triangles, can be written as a linear combination of $\mathbf{a}_{I,J}$ and $\mathbf{b}_{I,J}$, i.e.,

$$\mathbf{p}_{I,J} = \mathbf{p}(t_{I,J}) = (1-t_{I,J})\mathbf{a}_{I,J} + t_{I,J}\mathbf{b}_{I,J}, \quad t_{I,J} \in [0,1]. \tag{29.10}$$

The values $t_{I,J}$ are computed (and stored) when projecting points on the surface triangulation. These values remain unchanged, even if intersection points are mapped onto the real parametric surfaces. Hardy's reciprocal multiquadric method is used to compute a bivariate function $t(u,v)$ that interpolates all parameter values $t_{I,J}$ corresponding to intersection points that have been found. We must consider these conditions:

$$t_{I,J} = t(u_{I,J},v_{I,J}) = \sum_{j\in\{0,\ldots,N\}} \sum_{i\in\{0,\ldots,M\}} c_{i,j}\left(R + (u_{I,J}-u_{i,j})^2 + (v_{I,J}-v_{i,j})^2\right)^{-\gamma}, \\ I \in \{0,\ldots,M\}, \quad J \in \{0,\ldots,N\}, \tag{29.11}$$

where R and γ are fixed parameters and only those values $t_{I,J}$, $u_{I,J}$, $u_{i,j}$, $v_{I,J}$, and $v_{i,j}$ are considered for which an intersection point has been found. Denoting the "mean parameter spacing" in the two parameter directions by $\delta_u = \left(\frac{1}{2}M\right)\sum_{I=0}^{M}(u_{I+1,0}-u_{I,0}) + (u_{I+1,N}-u_{I,N})$ and $\delta_v = \left(\frac{1}{2}N\right)\sum_{J=0}^{N}(v_{0,J+1}-v_{0,J}) + (v_{M,J+1}-v_{M,J})$, we have found that the values $R = 0.5(\delta_u + \delta_v)$ and $\gamma = 0.001$ yield good results. Further investigation is necessary regarding appropriate choices for these parameters.

This global approach, considering *all* projections that have been found, is generally too inefficient. Therefore, we *localize* Hardy's reciprocal method by considering only a relatively small number of found projections to determine an "artificial projection." If there is no intersection between a line segment

$\overline{\mathbf{a}_{I,J}\mathbf{b}_{I,J}}$ and the surface triangulations, we use the K closest found projections. For this purpose, we identify the K found projections whose associated index tuples are closest to the index tuple (I,J). We have found that values for K between five and ten yield good projection estimates. Thus, one has to solve the linear system

$$t_k = \sum_{i=1}^{K} c_i \left(R + (u_k - u_i)^2 + (v_k - v_i)^2 \right)^{-\gamma}, \quad k = 1, \ldots, K, \tag{29.12}$$

where (u_k, v_k) and (u_i, v_i) are parameter values for which projections are known. One must solve such a linear system for each missing projection.

Having determined parameter values $t_{I,J}$ for all line segments $\overline{\mathbf{a}_{I,J}\mathbf{b}_{I,J}}$ for which no projections are nowhere obtain each needed "artificial projection" as the linear combination

$$\mathbf{z}_{I,J} = \left(1 - t_{I,J}\right)\mathbf{a}_{I,J} + t_{I,J}\mathbf{b}_{I,J}. \tag{29.13}$$

The union of all points $\mathbf{y}_{I,J}$ and $\mathbf{z}_{I,J}$ defines an $(M + 1) \times (N + 1)$ curvilinear mesh which we use for the construction of a local B-spline surface approximant.

29.3.5 Constructing a Local Surface Approximant

We use the $(M + 1) \times (N + 1)$ points $\mathbf{y}_{I,J}$ and $\mathbf{z}_{I,J}$ to define a cubic B-spline surface interpolating these points and locally approximating the given surfaces. We denote this B-spline surface as

$$\mathbf{s}(u,v) = \sum_{j=0}^{n} \sum_{i=0}^{m} \mathbf{d}_{i,j} N_i^4(u) N_j^4(v), \tag{29.14}$$

where $\mathbf{d}_{i,j}$ is a B-spline control point, $N_i^4(u)$ and $N_j^4(v)$ are the normalized B-spline basis functions of order four, and $m = 3M$ and $n = 3N$. The (normalized) knot vectors are defined by values ξ_i ($\xi_i < \xi_{i+1}$) and η_j ($\eta_j < \eta_{j+1}$) and have quadruple knots at both ends and triple knots in the interior, i.e.,

$$\begin{aligned} \mathbf{u} &= \left(u_0, u_1, \ldots, u_{m+4}\right) = \left(0,0,0,0,\xi_1,\xi_1,\xi_1,\ldots,\xi_{M-1},\xi_{M-1},\xi_{M-1},1,1,1,1\right) \quad \text{and} \\ \mathbf{v} &= \left(v_0, v_1, \ldots, v_{n+4}\right) = \left(0,0,0,0,\eta_1,\eta_1,\eta_1,\ldots,\eta_{N-1},\eta_{N-1},\eta_{N-1},1,1,1,1\right). \end{aligned} \tag{29.15}$$

For more details regarding B-splines, see, e.g., [Bartels et al. 1987], [Farin 1997], and [Piegl and Tiller 1996]. Here, we are using the indexing scheme used in [Bartels et al. 1987]. We are currently using a uniform knot spacing, i.e., $\xi_i = i/M$ and $\eta_j = j/N$.

Our construction yields local B-spline approximants degenerating to C^1-continuous, piecewise bicubic Bézier surfaces. The control points $\mathbf{d}_{i,j}$ are derived by first using a C^1 cubic interpolation scheme for all rows and columns of points to be interpolated and, second, applying C^1 continuity conditions to obtain the four interior Bézier control points of each bicubic patch constituting a single B-spline approximant. The interior Bézier control points of all bicubic patches are defined by the equations

$$\begin{aligned} \mathbf{d}_{3i\mp1,3j+1} &= \mathbf{d}_{3i\mp1,3j} + \left(\mathbf{d}_{3i,3j+1} - \mathbf{d}_{3i,3j}\right) \quad \text{and} \\ \mathbf{d}_{3i\mp1,3j-1} &= \mathbf{d}_{3i\mp1,3j} + \left(\mathbf{d}_{3i,3j-1} - \mathbf{d}_{3i,3j}\right). \end{aligned} \tag{29.16}$$

29.3.6 Error Estimation

Roughly speaking, the *error* between a locally approximating B-spline surface $\mathbf{s}(u,v)$ and the given geometry is the maximum of the minimum distances between points on the approximant and the original geometry. The existence of discontinuities and overlapping surfaces in the given geometry makes a precise definition of error impossible. We estimate the maximum distance by a discrete error measure. We use the points

$$\begin{aligned}
\mathbf{e}_{I,J} &= \mathbf{s}\left(\left(0.5\left(\xi_I+\xi_{I+1}\right),\eta_J\right)\right), \quad I=0,\ldots,(M-1),\, J=0,\ldots,N,\\
\mathbf{f}_{I,J} &= \mathbf{s}\left(\left(\xi_I,0.5\left(\eta_J+\eta_{J+1}\right)\right)\right), \quad I=0,\ldots,M,\, J=0,\ldots,(N-1), \quad \text{and} \qquad (29.17)\\
\mathbf{g}_{I,J} &= \mathbf{s}\left(\left(0.5\left(\xi_I+\xi_{I+1}\right),0.5\left(\eta_J+\eta_{J+1}\right)\right)\right), \quad I=0,\ldots,(M-1), J=0,\ldots,(N-1)
\end{aligned}$$

to compute this discrete error measure. An approximating B-spline surface most likely has its greatest deviation from the given geometry in the interior of its constituting bicubic Bézier patches due to the oscillation characteristics of bicubic spline surfaces.

We compute shortest (perpendicular) distances between points on the B-spline approximants and the original surfaces by solving the implied bivariate minimization problem to identify closest points. We do not compute shortest distances for all points $\mathbf{e}_{I,J}$, $\mathbf{f}_{I,J}$, and $\mathbf{g}_{I,J}$; whenever one of these points is associated with a gap in the given geometry we do not compute a closest point for it. If the resulting error estimate is too large for a particular B-spline approximant, the resolution parameters M and N are increased and a new B-spline approximant is computed. In principle, there is no guarantee that this process will converge for arbitrary geometries. Therefore, this iteration is terminated when a maximum resolution is reached. In practice, however, one does not have to worry about this problem as long as the user specifies a "reasonable" set of boundary curves for the initial Coons patch that is projected onto the geometry.

29.3.7 Connecting the Local B-Spline Approximants

Topologically, all B-spline approximants are four-sided entities that can have at most four neighbors. Two B-spline surfaces are neighbors when they share a common boundary curve. Bifurcations (more than two surfaces sharing the same boundary curve) in the set of all B-spline approximants are not allowed. In summary, the B-spline approximants must satisfy these topological conditions:

- Each boundary curve is shared by at most two B-spline approximants.
- A corner point of a B-spline approximant can be common to any number of approximants.
- If a corner point of B-spline approximant is shared by a second approximant then this point is also a corner point of the second approximant (*full-face interface property*).

All B-spline approximants used in the final approximation must be compatible, i.e., they must be bicubic B-spline surfaces, must have the same number of control points, and must have the same knots along common boundary curves. For an arbitrary configuration, this implies that one must use *one* global number of control points in both parameter directions, *one* global knot vector used in both parameter directions, and *one* global order.

However, it is sufficient for practical grid generation purposes to know the connectivity among all B-spline approximants and to know that the *distance* between two neighbor B-spline approximants is smaller than some predefined tolerance. In this context, a distance is implied for two neighbor B-spline approximants by the physically existing gap between the two distinct boundary curves that, topologically, define the common boundary. As long as the gaps between all neighbor B-spline approximants are negligible, then one can use them directly for the generation of a mesh. In the context of mesh generation, we must emphasize that an initial mesh is generated for the set of approximating B-spline surfaces and that this

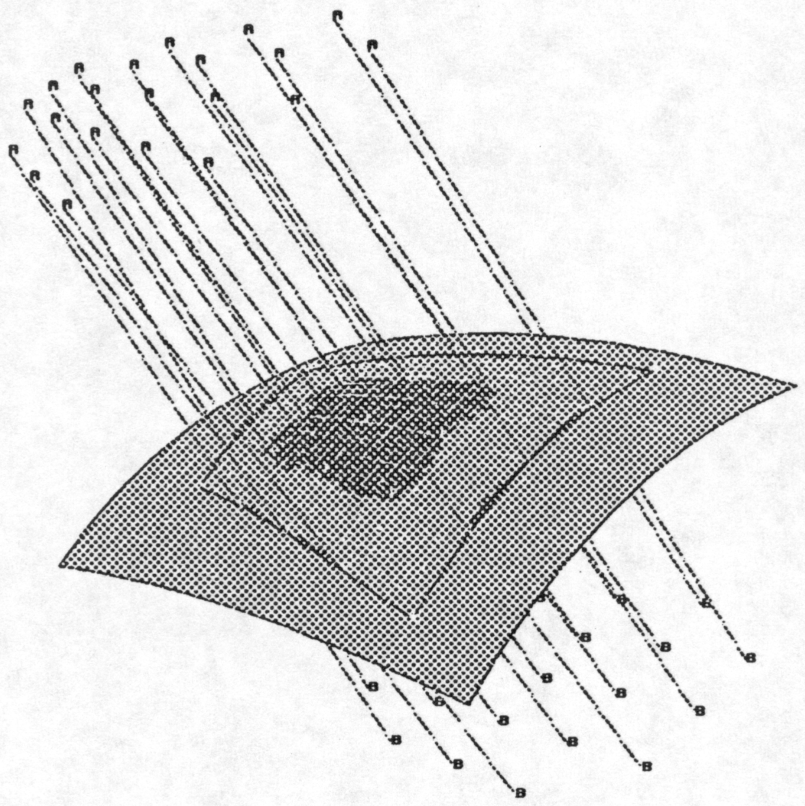

FIGURE 29.8 Approximation of part of single patch.

mesh is finally projected onto the original surfaces — unless an initial mesh point is in a gap region of the given geometry.

The conditions that must be satisfied by the B-spline approximants in order to obtain an overall tangent plane continuous approximation, also called *it gradient-continuous approximation*, are described in [Faux and Pratt 1979]. Essentially, the conditions are coplanarity conditions for certain B-spline control points along shared boundary curves and around shared corner points of B-spline approximants.

This approximation scheme for the "repair" of discontinuous geometries is explained in much more detail in [Hamann 1994] and [Hamann and Jean 1994, 1996].

29.3.8 Examples

Figures 29.8 through 29.11 show single B-spline surfaces approximating various geometries with discontinuities. One can see that the approximating surfaces are lying partially above and partially below the original surfaces. The approximating B-spline surfaces were obtained by specifying combinations of points and curves on the original geometries. Figures 29.8 and 29.9 show the line segments used to obtain sample points. Figures 29.12 and 29.13 show real-world geometries and their approximations (car body and aircraft configuration). Both figures show the original surfaces (top) and their approximations (bottom) consisting of multiple B-spline surfaces.

29.4 Surface–Surface Intersection — Underlying Principles and Best Practices

A good surface–surface intersection (SSI) algorithm should have the following characteristics:

- **Accuracy.** Grid generation for problems with high-gradient regions (such as viscous fluid flow) require a high degree of precision; a good algorithm must yield precise results.
- **Efficiency.** In an interactive environment, all but the most demanding cases should require only a few seconds to solve.

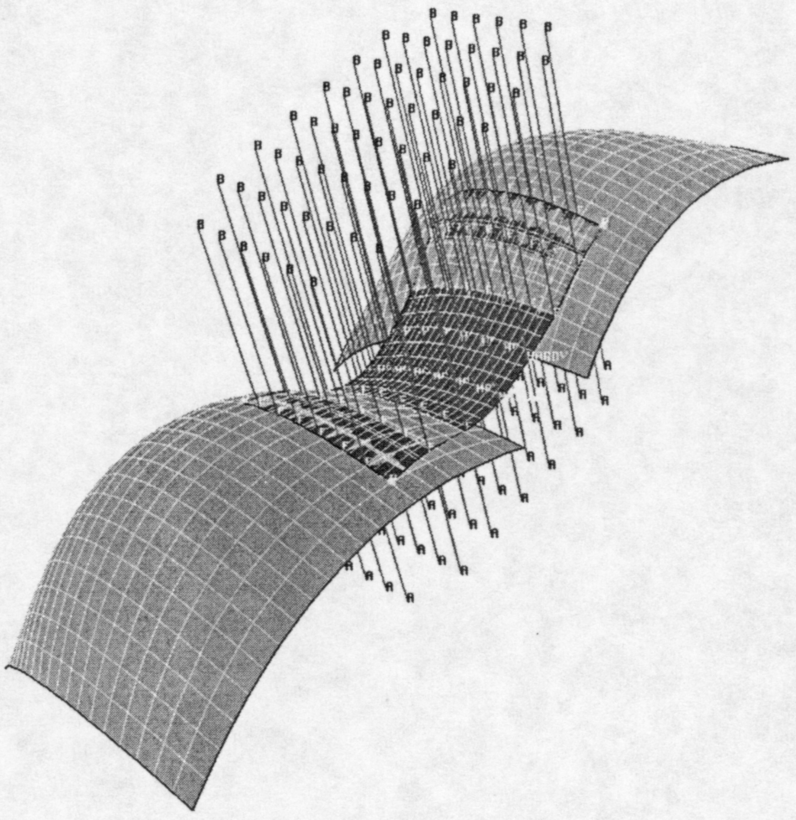

FIGURE 29.9 Approximation of surfaces with gap.

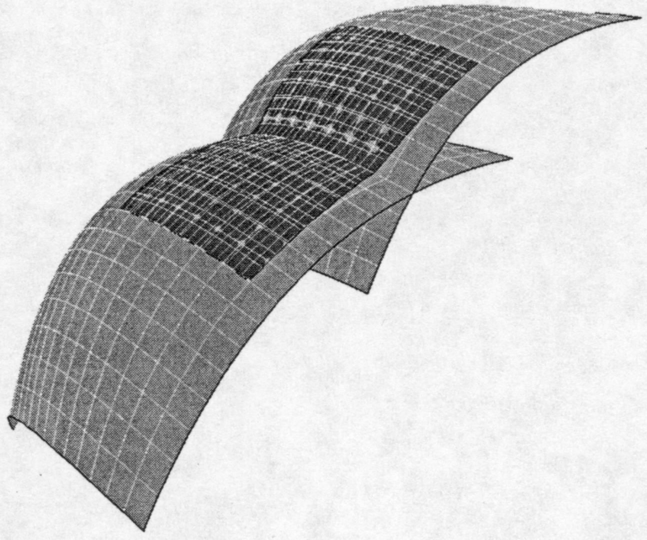

FIGURE 29.10 Approximation of intersecting surfaces.

- **Robustness**. The algorithm should correctly determine multiple intersections among multiple surfaces.
- **Simplicity**. The only action required by the user should be the specification of the surfaces to be intersected and a requested tolerance.

At present, no single algorithm possesses all of these properties. This is due to the fact that an optimal algorithm for a particular intersection problem depends on the type of surfaces involved. For example, the intersection of two planes is a line, while the intersection of two quadrics can be a curve of degree four. The representation of the surfaces must also be considered (i.e., implicit, polyhedral, or parametric). The reader can find a good survey of several types of intersection algorithms in [Hoschek and Lasser 1993].

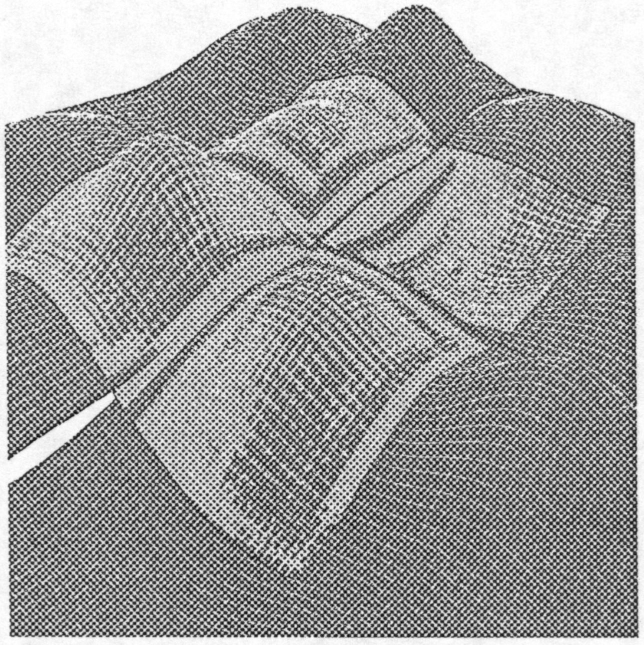

FIGURE 29.11 Approximation of highly oscillating surfaces.

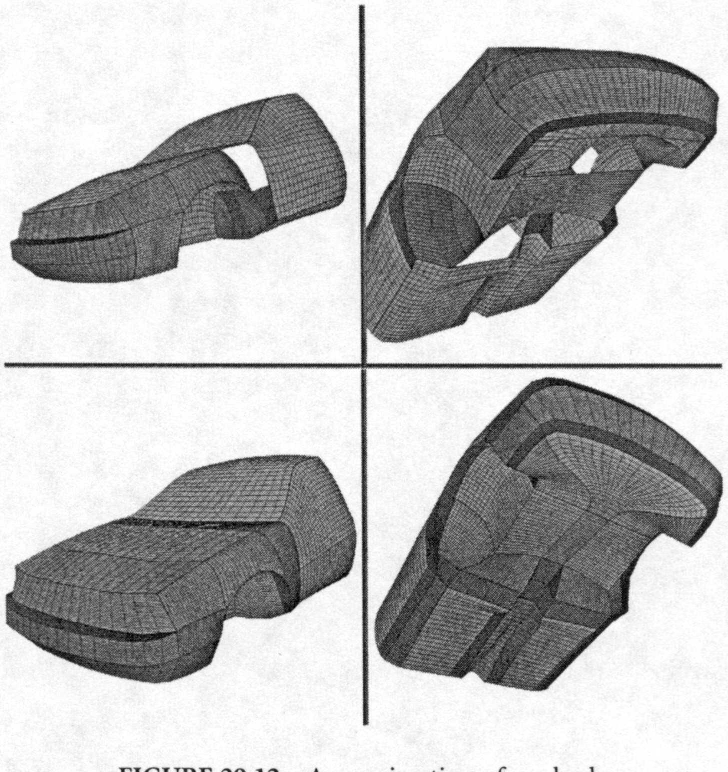

FIGURE 29.12 Approximation of car body.

29.4.1 The Intersection Algorithm.

The SSI algorithm we are describing can operate on surface triangulations, on analytically defined surfaces such as NURBS, or combinations thereof, see [Jean and Hamann 1998]. If the intersection is performed on surface triangulations, then the refinement step described below is skipped, and the piecewise linear curve produced from the intersection of triangles is the end result. If an analytical surface description is known for one or both of the surfaces involved then the refinement step is included. Surface triangulations are frequently encountered in the form of *unstructured surface grids* and are rapidly becoming a standard for data exchange using the StereoLithography format (see [3D Systems, Inc. 1989]). These are the basic steps of the algorithm:

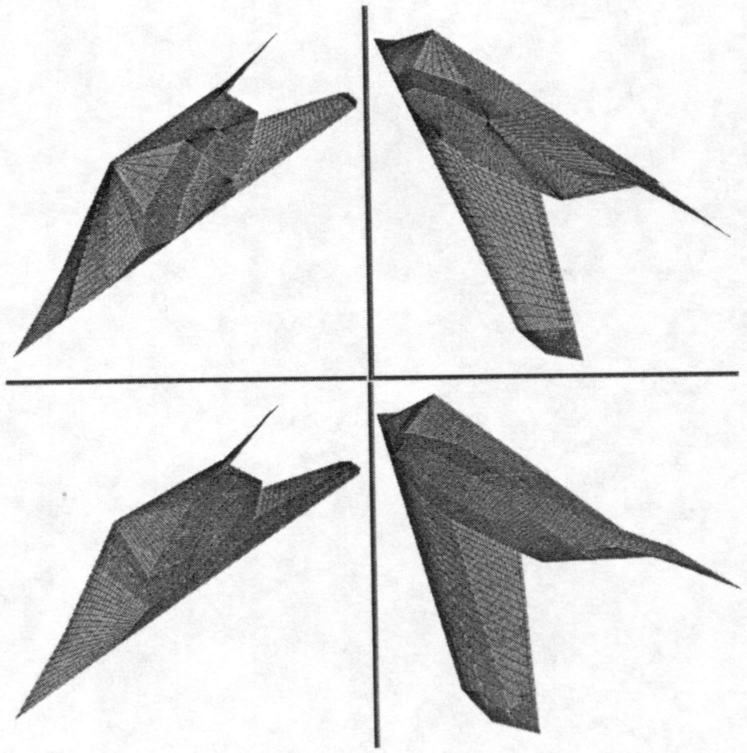

FIGURE 29.13 Approximation of aircraft.

1. Triangulate the surfaces to be intersected.
2. Store each triangulation in a space partitioning tree structure.
3. Intersect the trees to obtain a list of intersecting regions.
4. Intersect the individual triangles within each set of intersecting regions to obtain a collection of line segments.
5. Sort the line segments.
6. Find a starting point for an "intersection loop."
7. Trace the loop storing sample points which are the endpoints of the line segments.
8. If an analytic surface representation is known, refine the sample points.
9. Interpolate the sample points with a spline curve in 3D physical space, 2D parameter space, or both spaces.

The actual intersection algorithm can only operate on two surfaces at a time. When more than two surfaces are to be intersected, the driver calls the intersection operator with successive pairs of surfaces until all possible surface pairs are processed. If the desired result is the intersection of several surfaces, then additional curve-curve intersections may be necessary.

29.4.2 Triangulation

Parametric surfaces are discretized using an adaptive triangulation technique based on recursive subdivision (see [Anderson et al. 1997] and [Samet 1990]). This method triangulates the surface within a specified tolerance without using an excessive number triangles. An example of this method is shown in Figure 29.14. This "adaptive" feature allows the SSI algorithm to more accurately capture important intersection features such as singular points, i.e., points where the normals of the two surfaces are colinear or nearly colinear.

Triangles are stored as a list of vertices and a connectivity table. Each vertex in the triangulation is stored only once in order to reduce memory requirements and to eliminate the possibility of slight edge mismatches due to numerical error. A separate list of associated *uv* parameter values and *uv* connectivity (connectivity in parameter space) is maintained to allow refinement of the calculated intersection curves.

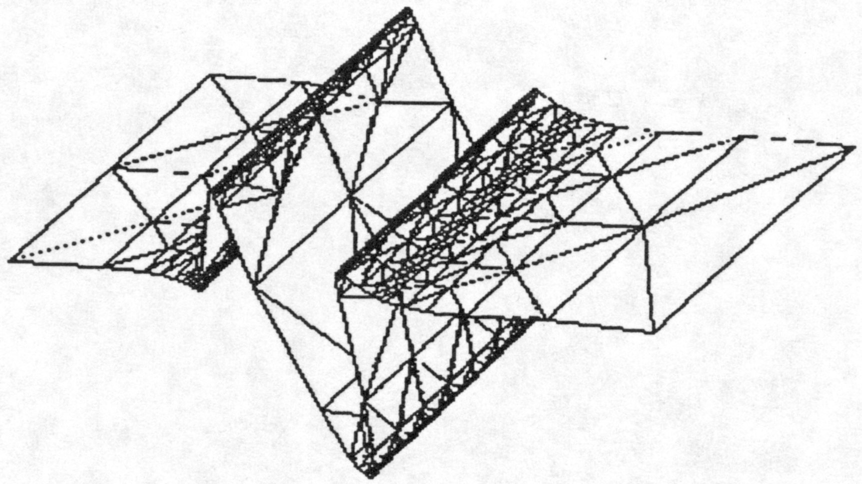

FIGURE 29.14 Adaptive surface triangulation.

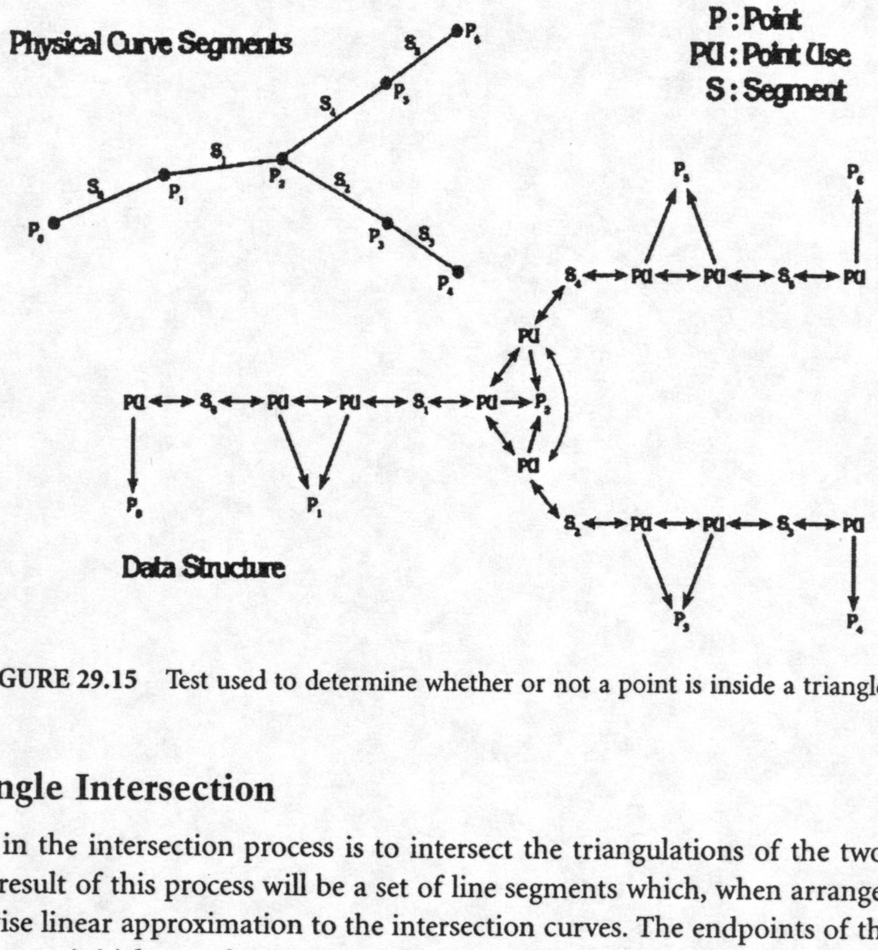

FIGURE 29.15 Test used to determine whether or not a point is inside a triangle.

29.4.3 Triangle Intersection

The first step in the intersection process is to intersect the triangulations of the two surfaces being considered. The result of this process will be a set of line segments which, when arranged properly, will provide a piecewise linear approximation to the intersection curves. The endpoints of the line segments will be used later as an initial guess for the sample points on the true intersection curves. The line segment information is used to determine the topology of the intersection curves and to order the sample points on the curves.

The method intersects two triangles by first performing a bounding box test to see if there is the possibility of the triangles intersecting. If this test is passed, then the edges of the first triangle are intersected with the plane defined by the second triangle and *vice versa.* The points resulting from the edge-plane intersections are then tested to determine if they lie inside the respective triangles. Figure 29.15 illustrates the test that determines whether a point lies inside a triangle. If the area of each of the subtriangles shown is positive, then the point is inside the triangle.

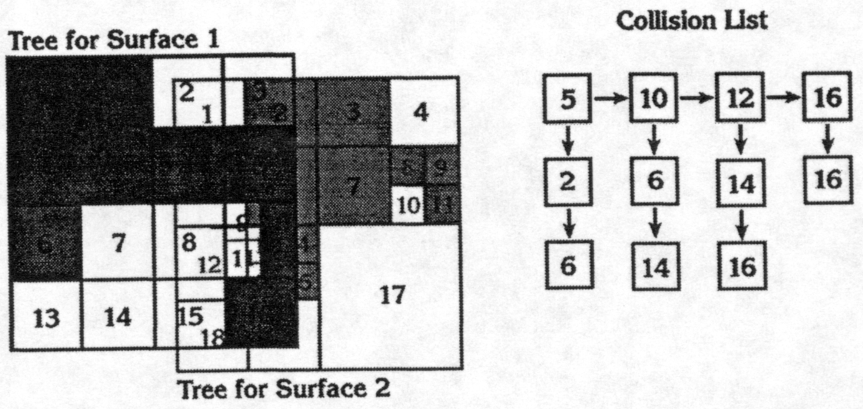

FIGURE 29.16 Intersection of quadtrees and resulting node association list.

The triangle intersection can yield one of three possible results:

1. No points are found that lie within either of the triangles, i.e., the triangles do not intersect.
2. Only one unique point is found, i.e., the triangles intersect only along the edges or at a vertex.
3. Two unique points are found, i.e., the intersection is a line segment.

The SSI method only considers intersections that result in line segments. The first case is ignored for obvious reasons. Case two is ignored because it yields only a point and therefore no information about the topology of the intersection loop.

29.4.4 Intersection Preprocessing Using a Tree Structure

The number of triangles needed to represent a surface may be quite large. The bounding box test discussed above is very fast. However, each triangle of the first surface must be compared with each triangle of the second surface. If steps were not taken to reduce the number of comparisons, this step would dominate the running time of the algorithm. There is a need to efficiently cull triangles that will not be involved in the intersection process. This is achieved by storing the triangles in a tree structure. The tree partitions the space occupied by the triangles and provides quick access to the set of triangles which inhabit a particular region. The tree type we use is a *k-d tree* (see [Samet 1990] and [Bentley 1975]). Given N triangles, the k-d tree will have at most $2N$ nodes with N leaf nodes, each containing exactly one triangle. A node is composed of a bounding box and an integer tag. The bounding box is specified by two points in space and is just large enough to contain the bounding boxes of all its children; the bounding box of a leaf node is just large enough to contain its associated triangle. We use a tag for leaf nodes to identify the triangle that is contained in the leaf.

A separate tree is constructed for each surface. One tree is chosen — it does not matter which one — as the *base tree*, and the remaining tree is referred to as the *target*. The two trees are intersected as follows:

1. Pick a leaf node in the base tree.
2. Intersect base leaf with target tree using recursive bounding box tests.
3. Associate each base leaf with each target leaf that intersects it. If the base leaf does not intersect any target leaf, then the base leaf is not considered.
4. Repeat this procedure for each leaf in the base tree.

The result of the intersection is a set of associations which encompass all possible triangle intersections for the given surfaces. Note that target leaves may be associated with multiple base leaves. However, each base leaf appears only once. This relationship is depicted in Figure 29.16. Note that a two-dimensional quadtree is used to simplify the figure. The k-d tree is a binary tree and can be searched in logarithmic time. Hence, for two surfaces represented by M and N triangles, the tree intersection can be performed in $M\log_2(N)$ time.

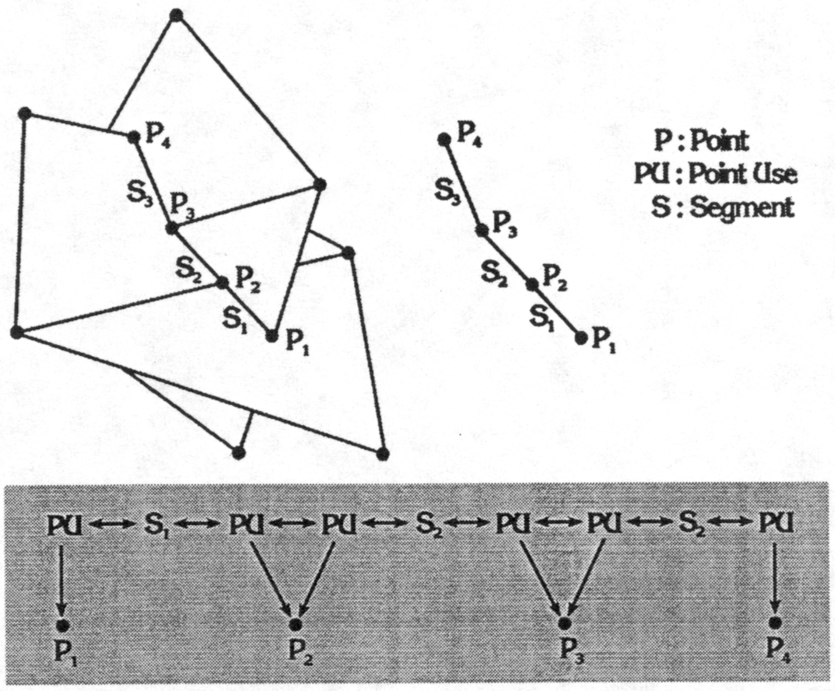

FIGURE 29.17 Data structure used to represent SSI curve topology.

29.4.5 Data Structure, Loop Detection, and Curve Tracing

The points and line segments resulting from the triangle intersection step are stored in a special topology data structure. This data structure provides explicit connectivity between the line segments and allows intersection curves to be easily identified and traced.

The *Point* structure stores the coordinates in physical space, *xyz* space, for the point as well as its associated parameter values for each of the two surfaces. A Point also has an associated circular linked list of *PointUses.* PointUse structures contain connectivity and other topological information about the Point. Each Point in the system is unique. If a new Point is computed having the same coordinates as an existing point, then a Point Use with the appropriate information is added to the list of uses for the existing Point. This list of unique points and uses builds the topology of the intersection curves as the triangles are intersected and does not depend on the order in which the intersections occur.

The PointUse structure contains topological information associated with a Point and a *Segment.* The Segment data structure provides Point connectivity information using PointUses. A Point shared by both Segments has two PointUses. Since both of these PointUses belong to the same Point, they are linked and hence the line segments are linked as well.

The PointUse structure contains a **location** field, which indicates where the PointUse is located on its associated Segment structure. This location is either zero or one indicating the start point or end point of the line segment. The **P** field and **SSISeg_PTR** field are links to the associated Point and Segment structures. So-called **prev** and **next** fields link the PointUse to others (if any) in the circular linked list of PointUses. The **InUse** field is a boolean flag used in the process of tracing the intersection curves.

Detection of individual intersection curves, *loops,* is based on PointUses. In this method, an endpoint of a curve is defined as a point with a number of PointUses not equal to two (closed curves are a special case where all points have two PointUses). If more than two PointUses are present, then the endpoint is a singular point (where three or more curves meet). Intersection curves are automatically broken at a bifurcation point.

Figure 29.17 illustrates this concept. We depict the intersection of four triangles, belonging to two different surfaces, resulting in four intersection points, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, and $\mathbf{p}_4$. This example yields one loop whose two endpoints are $\mathbf{p}_1$ and $\mathbf{p}_4$. This is a basic outline of the overall curve tracing algorithm:

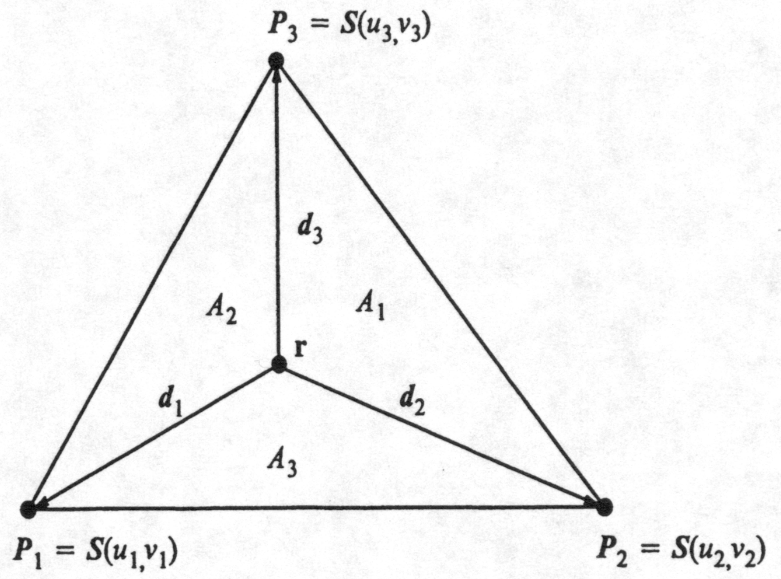

FIGURE 29.18 Stencil of data required to obtain intersection point on exact surface.

1. Find a Point with a number of PointUses $\neq 2$ and at least one PointUse with its *InUse* flag set to FALSE. If none can be found, stop.
2. Go to the PointUse with InUse set to FALSE.
3. Set PointUse $\rightarrow$ InUse = TRUE and add the Point to the ordered list of sample points on the curve. (Remark: The two triangles used to generate the segment are also stored for use in refinement steps.)
4. Go to the Segment associated with the PointUse and set Segment $\rightarrow$ InUse = TRUE.
5. Go to the opposite PointUse on the Segment.
6. Set PointUse $\rightarrow$ InUse = TRUE and add its associated Point to the ordered list of sample points on the curve.
7. If the number of PointUses associated with the present Point is two, step to the other PointUse associated with the Point (PointUse $\rightarrow$ next) and go to step 4; otherwise, continue below.
8. Store the ordered list of sample points for refinement.
9. Repeat.

Should this algorithm terminate and leave certain Segments unused, then one or more of the intersection curves are closed. Closed curves are a special case and are treated separately. Closed curves are found by picking a random starting PointUse from the remaining "unused" PointUses and proceeding with the same basic algorithm. The difference is that the algorithm terminates when the curve is traced back to its starting point.

29.4.6 Refinement

Once all possible curves have been traced, the result is an ordered set of sample points for each intersection curve. In general, these points lie on the triangulation, or, to be more specific, on the piecewise linear surface approximations, but not on the exact analytical surface. The refinement procedure described below maps the points to the surfaces and matches them, within a given tolerance, to the "true" intersection.

The first step in the refinement process is the mapping of the intersection points onto each surface. Each intersection point on the triangulation has references to the triangles containing it. Figure 29.18 shows the stencil of data required to map a point **r** (inside a triangle) to the exact underlying surface. The procedure to do this follows these steps:

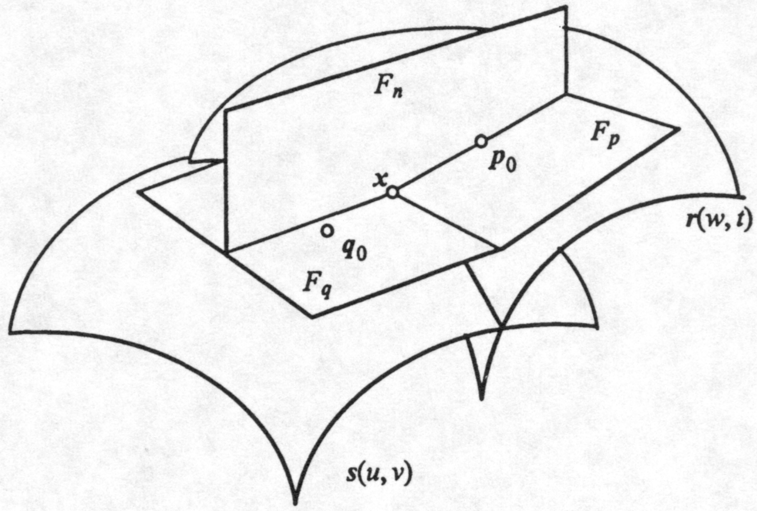

FIGURE 29.19 Point refinement using *auxilliary plane method.*

1. Find vectors $\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$ emanating from $\mathbf{r}$ and stopping at the respective triangle vertices $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$.
2. Calculate the sub-triangle areas A_1, A_2, and A_3.
3. Normalize the sub-triangle areas by dividing A_1 by the total triangle area $A_1 + A_2 + A_3$.
4. The normalized sub-triangle areas A_i are the *barycentric coordinates* of $\mathbf{r}$ with respect to the original triangle defined by $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$. Denoting the parameter values of $\mathbf{P}_i$ by (u_i,v_i), we compute $\left(\sum_{i=1}^{3} A_i u_i, \sum_{i=1}^{3} A_i v_i\right)$, which is the parameter value that we use to compute a point on the exact surface replacing $\mathbf{r}$.

The refinement technique used is the *auxiliary plane method* (see Figure 29.19) described in [Hosaka 1992]. The basic steps of this method are:

1. Denote the two "images" of $\mathbf{r}$ on the two underlying parametric surfaces $\mathbf{s}(u,v)$ and $\mathbf{r}(w,t)$ by $\mathbf{q}_0$ and $\mathbf{p}_0$; let $\mathbf{p}_0 = \mathbf{s}(u_0,v_0)$ and $\mathbf{q}_0 = \mathbf{r}(w_0,t_0)$, where u_0, v_0, w_0, and t_0 are the associated parameter values.
2. Calculate the unit normals $\mathbf{n}_p$ and $\mathbf{n}_q$ at $\mathbf{p}_0$ and $\mathbf{q}_0$.
3. Let F_p and F_q be the tangent planes at $\mathbf{p}_0$ and $\mathbf{q}_0$.
4. Calculate the distance values d_p and d_q for the distances between F_p (F_q) and the origin:

$$d_p = \mathbf{n}_p \cdot \mathbf{r}(w_0,t_0), \quad d_p = \mathbf{n}_q \cdot \mathbf{s}(u_0,v_0). \tag{29.18}$$

5. Construct a plane F_n which is orthogonal to both F_p and F_q and passes through $\mathbf{p}_0$. The unit normal $\mathbf{n}_n$ of F_n and its distance from the origin d_n are:

$$\mathbf{n}_n = \frac{\mathbf{n}_p \times \mathbf{n}_q}{\left\|\mathbf{n}_p \times \mathbf{n}_q\right\|}, \tag{29.19}$$

$$d_n = \mathbf{n}_n \cdot \mathbf{r}(w_0,t_0). \tag{29.20}$$

6. Calculate the intersection point $\mathbf{x}$ of the planes F_p, F_q, and F_n as

$$\mathbf{x} = \frac{d_p\left(\mathbf{n}_q \times \mathbf{n}_p\right) + d_q\left(\mathbf{n}_n \times \mathbf{n}_p\right) + d_n\left(\mathbf{n}_p \times \mathbf{n}_q\right)}{\left[\mathbf{n}_p, \mathbf{n}_q, \mathbf{n}_n\right]}, \tag{29.21}$$

   where $[\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3]$ is the *scalar triple product* $(\mathbf{v}_1 \times \mathbf{v}_2) \cdot \mathbf{v}_3$ of three 3D vectors, see [Hosaka 1992]. (Remark: The point $\mathbf{x}$ is an approximate intersection point and, in general, will lie neither on $\mathbf{s}(u,v)$ nor on $\mathbf{r}(w,t)$.)
7. The point $\mathbf{x}$ must be mapped back to the exact surfaces and new points $\mathbf{p}_0$ and $\mathbf{q}_0$ calculated. We compute the difference vectors $\delta\mathbf{p}_0 = \mathbf{x} - \mathbf{p}_0$ and $\delta\mathbf{q}_0 = \mathbf{x} - \mathbf{q}_0$ and compute the values

$$\acute{\mathbf{r}}_w = \mathbf{r}_w \times \mathbf{n}_p, \quad \acute{\mathbf{r}}_t = \mathbf{r}_t \times \mathbf{n}_p \tag{29.22}$$

   and

$$\acute{\mathbf{s}}_u = \mathbf{s}_u \times \mathbf{n}_q, \quad \acute{\mathbf{s}}_v = \mathbf{s}_v \times \mathbf{n}_q, \tag{29.23}$$

   where $\mathbf{r}_w$, $\mathbf{r}_t$, $\mathbf{s}_u$, and $\mathbf{s}_v$ are the partial derivative vectors (not normalized) at $\mathbf{p}_0$ and $\mathbf{q}_0$. Considering that for infinitesimally small increments the two equations $\mathbf{r}_w\delta w + \mathbf{r}_t\delta t =$
   $\delta\mathbf{p}_0$ and $\mathbf{s}_u\delta u + \mathbf{s}_v\delta v = \delta\mathbf{q}_0$ hold, we can compute the increments for the parameter values as

$$\delta w = \frac{\acute{\mathbf{r}}_t \cdot \delta\mathbf{p}_0}{\acute{\mathbf{r}}_t \cdot \mathbf{r}_w}, \quad \delta t = \frac{\acute{\mathbf{r}}_w \cdot \delta\mathbf{p}_0}{\acute{\mathbf{r}}_w \cdot \mathbf{r}_t}, \tag{29.24}$$

$$\delta u = \frac{\acute{\mathbf{s}}_v \cdot \delta\mathbf{q}_0}{\acute{\mathbf{s}}_v \cdot \mathbf{s}_u} \quad \delta v = \frac{\acute{\mathbf{s}}_u \cdot \delta\mathbf{q}_0}{\acute{\mathbf{s}}_u \cdot s_v}. \tag{29.25}$$

   The updated values of $\mathbf{p}_0$ and $\mathbf{q}_0$ are thus given by

$$\mathbf{p}_0 = \mathbf{r}\left(w_0 + \delta w, t_0 + \delta t\right), \quad \mathbf{q}_0 = \mathbf{s}\left(u_0 + \delta u, v_0 + \delta v\right). \tag{29.26}$$

8. Steps 2 through 7 are repeated until $\|\mathbf{p}_0 - \mathbf{q}_0\|$ is within a specified tolerance.

Convergence of this method is very good, even for "poor" initial values of $\mathbf{p}_0$ and $\mathbf{q}_0$. The curve defined by the triangle intersections may or may not meet the requested tolerance. Intersection points can be added or deleted as necessary. Additional intersection points are obtained using the refinement algorithm with starting points based on the known intersection points. The final representation of the curve depends on the requirements of a particular application. Common representations are piecewise linear or cubic curve representations in physical and/or parameter space.

29.5 Research Issues and Summary

In conclusion, we summarize the presented techniques, describe possible improvements, and point out remaining research issues.

29.5.1 Surface Refinement and Reparametrization

We have given techniques for *refining* the parametrization of a NURBS surface. The surface approximation method performs best when the interior surface geometry more or less follows the geometry of the boundary curves. Future work could be directed at the analysis of geodesic curvature distribution of isoparametric curves on the surface and using it as an interrogation tool. Data reduction is another research issue. In some cases, this is achieved as a side effect. Torsion characteristics of the boundary curves are not exploited. Torsion could be incorporated into the scheme in the same way as curvature and arc length to find "key interpolation points" where torsion may be a factor.

29.5.2 Approximation of Discontinuous Geometries

The interactive method for the approximation of discontinuous geometries allows the "replacement" of entire 3D geometries while preserving original boundary curves of given surfaces, if so desired. The method can be used to approximate geometries with gaps, transverse surface overlaps, and undesired surface intersections. The final approximation is a set of bicubic B-spline surfaces determining an overall continuous surface approximation — with negligible gaps between neighbor B-spline surfaces. One should explore whether it is possible to reduce the required user input further, i.e., whether one can construct local approximants automatically without having a user specify the boundary curves of the initial Coons approximants. Currently, the resulting B-spline surfaces approximating the given discontinuous geometry are stored as NURBS surfaces — with all control point weights being one. Choosing control point weights in a more "clever" way might allow the generation of equally good approximants with fewer control points.

29.5.3 Surface–Surface Intersection

The SSI algorithm discussed above is only one of many possible approaches to solving this difficult problem. The advantages of this algorithm are its speed and accuracy and the ability to operate automatically. The method relies on triangles that are locally planar approximations to the underlying surface; therefore, the algorithm can have difficulties when resolving intersection curve topologies near critical points, where critical points occur when the normals of both intersecting surfaces are exactly or nearly collinear. In the region around critical points, the intersection of the surface triangulations may not accurately reflect the intersection of the underlying surfaces, hence causing the algorithm to fail.

Acknowledgments

This work was supported by the National Grid Project consortium and the National Science Foundation under contract EEC-8907070 to Mississippi State University. Special thanks go to all members of the research and development team of the National Grid Project, which was performed at the NSF Engineering Research Center for Computational Field Simulation, Mississippi State University. Part of the work was carried out by the CAGD research group at Arizona State University.

Further Information

The following journals, magazines, and conference proceedings frequently cover topics related to the problems discussed in this chapter: *Computer-Aided Design* (Elsevier), *Computer Aided Geometric Design* (Elsevier), *Journal of Computational Physics* (Academic Press), *Transactions on Graphics* (ACM), *Transactions on Visualization and Computer Graphics* (IEEE), *The Visual Computer* (Springer-Verlag), *Computer Graphics and Applications* (IEEE), *SIGGRAPH* proceedings (ACM), and *Supercomputing* proceedings (ACM/IEEE). In addition, the *SIAM Conference on Geometric Design*, which is organized every other year, is an excellent source of information.

References

1. Anderson, J., Khamayseh, A., and Jean, B.A., Adaptive resolution refinement, Technical Report, Los Alamos National Laboratory, Los Alamos, NM, 1997.

2. Bartels, R.H., Beatty, J.C. and Barsky, B.A., *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann, Los Altos, CA, 1987.
3. Bentley, J., Multidimensional binary search trees used for associative searching, *Communications of the ACM.* 1975, 18, 9.
4. Coons, S.A., Surface patches and B-spline curves, Barnhill, R.E. and Riesenfeld, R.F., (Eds.), *Computer Aided Geometric Design*, Academic Press, San Diego, CA, 1974, pp 1–16.
5. Crampin, M., Guifo R., and Read, G.A., Linear approximation of curves with bounded curvature and a data reduction algorithm, *Computer Aided Design*, 1985, 17,6, pp 257–261.
6. Farin, G., *NURB Curves and Surfaces*, AK Peters, Wellesley, MA, 1995.
7. Farin, G., *Curves And Surfaces For Computer Aided Geometric Design*, 4th Edition, Academic Press, San Diego, CA, 1997.
8. Faux, I.D. and Pratt, M.J., *Computational Geometry for Design and Manufacture*, Ellis Horwood, New York, NY, 1979.
9. Foley, T., Local control of interval tension using weighted splines, *Computer Aided Geometric Design.* 1986, 3, 4, pp 225–230.
10. Foley, T., Interpolation with interval and point tension controls using cubic weighted *v*-splines, *ACM Trans. on Math. Software*, 1987, 13,1, pp 68–96.
11. Foley T. and Nielson G.M., A Survey of applications of an affine invariant norm, Lyche, T. and Schumaker, L.L., (Eds.), *Mathematical Methods in Computer Aided Geometric Design*, Academic Press, San Diego, CA, 1989, pp 445–467.
12. Franke, R., Scattered data interpolation: tests of some methods, *Math. Comp.* 1982, 38, pp 181–200.
13. Fuhr, R.D. and Kallay, M., Monotone Linear Rational Spline Interpolation, *Computer Aided Geometric Design*, 1982, 9, pp 313–319.
14. George, P.L., *Automatic Mesh Generation*, Wiley & Sons, New York, 1991.
15. Hamann, B., Construction of B-spline approximations for use in numerical grid generation, *Applied Mathematics and Computation*, 1994, 65, 1–3, pp 295–314.
16. Hamann, B. and Jean, B.A., Interactive techniques for correcting CAD/CAM data, Weatherill, N.P., Eiseman, P.R., Häuser, J., and Thompson, J.F., (Eds.), *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Pineridge Press, Swansea, U.K., 1994, pp 317–328.
17. Hamann, B. and Jean, B.A., Interactive surface correction based on a local approximation scheme, *Computer Aided Geometric Design*, 1996, 13, 4, pp 351–368.
18. Hosaka, M., *Modeling of Curves and Surfaces in CAD/CAM.* Springer–Verlag, New York, 1992.
19. Hoschek, J. and Lasser, D., *Fundamentals of Computer Aided Geometric Design*, AK Peters, Wellesley, MA, 1993.
20. Jean, B.A. and Hamann, B., An efficient surface-surface intersection algorithm based on surface triangulations and space partitioning trees, to appear in *Mathematical Engineering in Industry*, 1998.
21. Kim, T.W., Knot placement for NURB interpolation, M.S. thesis, Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, 1993.
22. Knupp, P.M. and Steinberg, S., *Fundamentals of Grid Generation.* CRC Press, Boca Raton, FL, 1993.
23. Piegl, L.A., On NURBS: A survey, *IEEE Computer Graphics and Applications.* 1991a, 11, 1, pp 55–71.
24. Piegl, L.A., Rational B-spline curves and surfaces for CAD and graphics, Rogers, D.F. and Earnshaw, R.A., (Eds.) *State of the Art in Computer Graphics*, 1991b, Springer-Verlag, New York, pp 225–269.
25. Piegl, L.A. and Tiller, W., *The NURBS Book*, 2nd Edition, Springer-Verlag, New York, 1996.
26. Samet, H., *The Design and Analysis of Spatial Data Structures.* Addison Wesley, New York, 1990.
27. Thompson, J.F., Warsi, Z.U.A., and Mastin, C.W., *Numerical Grid Generation*, North-Holland, New York, 1985.
28. 3D Systems, Inc., *Stereo Lithography Interface Specification.* 1989.
29. Yu, T. and Soni, B.K., Application of NURBS in numerical grid generation, *Computer Aided Design*, 1995, 27, pp 147–157.