Both approaches do not bound the offset error globally. To bound the error introduced by the piecewise arcs and lines approximation, a curve-line and a curve-arc maximum global distance computation is required. Such computation is traditionally performed using a finite set of samples. A bound on the maximum error over the entire curve region cannot be guaranteed using such a technique. In the second method, a finite number of samples are examined to estimate the error for the entire curve region (typically one, in the middle of the parametric domain), which again cannot insure global error bound. Both methods usually result in a piecewise representation of the approximation to the offset, a more difficult representation to use in further applications if the offset is to be used as a modeling tool. Only the use of B-spline refinement [13, 15] results in a single curve. Approximations to offsets of freeform surfaces are more difficult to determine because the subdivided components are subsurfaces. Piecewise bicubic patches have been used to approximate a surface offset of a given freeform surface [29]. This method loses continuity across patches, unlike the refinement technique [13], which can be adapted for surfaces and which maintains the original continuity.

Because of the advantages of the curve/surface B-spline refinement technique, we have used this method as the basis of this implementation for bounding the global error. However, the method presented here for bounding the error is *not* limited to this type of representation.

Trimming the loops formed by the self-intersection curves of the offset is considered a difficult problem [12]. An attempt has been made to make the calculation using numerical techniques and to perform a direct search for cusps as a mean of detecting and identifying self-intersections [35]. However, an approximation to the offset may have no cusps simply because it is just an approximation. For surfaces, unidimensional successive searches have been used to isolate self-intersection points by minimizing the ratio of the Euclidean space distance (which goes to zero at

a self-intersection point) over the parametric space distance (which should be nonzero at such point) [2]. Because this method converges to a local minimum, the initial guess location is crucial but is picked at *random*. Thus, robustness is not guaranteed. Self-intersection curves have been traced using surface "walking" techniques [2] that can also be combined with the detection methods developed here.

Section 3.1 develops the method for bounding the error and then shows how to use that information to isolate the regions with maximum error. Then, we show how to apply local improvement steps iteratively, so convergence to a prespecified tolerance is assured. In section 3.2, we attempt to improve offset approximations by perturbing control points using an analysis of the error function. Section 3.3 extends this method to support a variable offset operator that can be used as a modeling tool. Section 3.4 shows how to use the tools developed in section 3.1 to robustly detect and trim loops formed by self-intersections of the offset.

3.1 A Global Bound for the Offset Operator

Let $C(t)$ be a planar regular parameterized curve, which without loss of generality, is assumed to be in the $x-y$ plane. An offset curve for $C(t)$ by an amount d is defined mathematically as:

$$\hat{\mathcal{C}}_d(t) = C(t) + N(t)d \tag{3.1}$$

where $N(t)$ is the unit normal to the curve at t. Becuase $N(t)$ flips its direction by 180^o at inflection points, a different definition for $N(t)$ should be used to define a manufacturing or design offset:

Definition 3.1 *The* offset binormal, $B_o(t)$, *to a planar curve in the* $x-y$ *plane is a unit vector in* $+z$ *direction. Then the* offset normal, $N_o(t)$, *is defined as* $N_o(t) = B_o(t) \times T(t)$, *where* $T(t)$ *is the unit tangent to the curve.*

Throughout this Chapter, and unless otherwise specified, only the offset normal, $N_o(t)$, is used:

$$\mathcal{C}_d(t) = C(t) + N_o(t)d \tag{3.2}$$

Similarly for surfaces, an offset surface for surface $S(u,v)$ by an amount d is mathematically defined as:

$$\mathcal{S}_d(u,v) = S(u,v) + n(u,v)d \tag{3.3}$$

where $n(u,v)$ is the surface unit normal to the surface at parameter values (u,v).

In this Chapter, we will concentrate on characterizing methods for the NURBs representation because the Bézier representation is a subset of it. Given two freeform NURBs curves $C_1(t)$, $C_2(t)$, their sum, difference (equation (2.6)), and product (equation (2.13)) is also a NURBs curve as seen in Chapter 2. Derivatives of NURBs curves are also NURBs curves (equation (2.4)), as are constant functions (i.e., equation 2.10).

Therefore, if $N_o(t)$ $(n(u,v))$ could be computed and represented as a NURBs, so could $\mathcal{C}_d(t)$ $(\mathcal{S}_d(u,v))$, respectively. Unfortunately, however, the general form of a normal involves a square root which is usually not representable as either a polynomial or a piecewise polynomial. Thus, offsets of freeform curves and surfaces will, in general, be approximations.

Let $\mathcal{C}_d^a(t)$ be an approximation to the offset curve of $C(t)$ by an amount d (equation (3.2)), and let $\delta(t) = \mathcal{C}_d^a(t) - C(t)$ be the difference curve. Ideally, if $\mathcal{C}_d^a(t) \equiv \mathcal{C}_d(t)$, then $\delta(t) \equiv N_o(t)d$.

Two tests could be applied to $\delta(t)$ to determine the accuracy of the offset approximation. First, the deviation of $\delta(t)$ from the direction of $N_o(t)$ could be measured, by testing whether $\delta(t)$ is orthogonal to the curve tangent. If $\hat{T}(t) = C'(t)$, then $T(t) = \frac{\hat{T}(t)}{||\hat{T}(t)||}$ is the unit tangent of $C(t)$. $\left\langle \frac{\hat{T}(t)}{||\hat{T}(t)||}, \frac{\delta(t)}{||\delta(t)||} \right\rangle$ measures the cosine of the angle between $\hat{T}(t)$ and $\delta(t)$, and is equal to zero everywhere along the exact offset

curve. However, finding $T(t)$ and $\|\delta(t)\|$ requires representing square roots, and therefore is impractical when using a piecewise rational representation. However, the square of this inner product,

$$\frac{\left\langle \hat{T}(t), \delta(t) \right\rangle \left\langle \hat{T}(t), \delta(t) \right\rangle}{\|\hat{T}(t)\|^2 \|\delta(t)\|^2}, \tag{3.4}$$

can be represented.

Although equation (3.4) is representable as a piecewise rational, it is a complex process. The equation requires at least six curve products (more if the curves are rational), each of which doubles the degree.

Instead, a second test that measures the magnitude of $\delta(t)$ can be applied to determine the accuracy of $\mathcal{C}_d^a(t)$. Computationally, it is much more attractive. Current offset techniques usually test accuracy by evaluating this magnitude on a set of sampled points. Direct representation of $\|\delta(t)\|$ would require the representation of a square root, so $\psi(t) = \|\delta(t)\|^2$ is used instead and compared with d^2:

$$\psi(t) = \|\delta(t)\|^2 = \delta_x(t)^2 + \delta_y(t)^2 + \delta_z(t)^2 \tag{3.5}$$

where $\delta_x(t)$, $\delta_y(t)$ and $\delta_z(t)$ are the components of $\delta(t)$.

Equation (3.5) can be directly represented using multiplication and addition which are computable for rationals and piecewise rationals. Hereafter, assume $\psi(t)$ can be computed and represented as a scalar NURBs curve. For exact offsets, ψ is a constant value curve equal to d^2. By subtracting d^2 from ψ, the difference curve is obtained.

$$\epsilon(t) = \psi(t) - d^2. \tag{3.6}$$

The extremal values of the coefficients of ϵ provide a global error measure. It is important to examine the consequences for computing $\epsilon(t)$ instead of $\varepsilon(t) = \|\delta(t)\| - d$, the Euclidean error between the exact offset curve and its approximation:

$$\epsilon(t) = \psi(t) - d^2 = \|\delta(t)\|^2 - d^2 = (\varepsilon(t) + d)^2 - d^2 = \varepsilon(t)^2 + 2d\varepsilon(t) \approx 2d\varepsilon(t) \tag{3.7}$$

In other words, by computing the differences of the squared magnitude, the resulting error bound is scaled by the magnitude of twice the offset distance, $2d$, which is a constant and therefore easy to control. $\varepsilon(t)^2$ has been ignored because it is much smaller than $2d\varepsilon(t)$, when the error converges to zero.

The problem of finding the global offset error has been reduced to a problem of finding the extrema of a freeform explicit curve. Because the values of a scalar B-spline curve over an interval lie between the maximum and minimum values of the coefficients of the nonzero B-spline functions, a simple and computationally efficient way of locally bounding the curve is immediately available.

The error between a C^2 continuous function and its Schoenberg variation diminishing spline approximation over a knot vector $\{t_i\}$ is $O(|\{t_i\}|^2)$, where $|\{t_i\}| = max_i\{t_{i+1} - t_i\}$. By using a sequence of Schoenberg variation diminishing spline approximations to $N_o(t)$, each one based on a knot vector that is a refinement of the previous one, and a sequence, $\{C_i(t)\}$, of refined representations to C, based on the same sequence of knot vectors, we form a convergent sequence of approximations to C_d. If the approximation is close over one interval, it is unnecessary to refine over that interval just to make the mesh norm smaller, because the approximation error is based on maximum error bounds over local regions. Hence, we need only refine over intervals where the error is large, as determined by the extrema of ϵ.

We derive an iterative algorithm in which each step uses the direct polygon transformation method [13] to compute offset approximations. The criterion for proceeding to the next step uses the magnitude of the extrema of $\epsilon(t)$. Then, the locations of the extrema are used to refine $C(t)$ (going from $C_i(t)$ to $C_{i+1}(t)$) and to create a new approximation to the offset. The process terminates when the magnitudes of the extrema of ϵ are within the tolerance.

Algorithm 3.1 retains its curve refinement history in the $C_i(t)$ sequence. The last curve in the sequence can be offset to within a provided tolerance by an amount

Algorithm 3.1

Input:
τ, required offset curve tolerance.
$C(t)$, input curve.
d, offset distance.

Output:
$C_d^a(t)$, offset curve approximation within τ accuracy.

Algorithm:
$C_0(t) \Leftarrow C(t)$.
$i \Leftarrow 0$.
Do
  Compute offset approximation $C_d^a(t)$ for $C_i(t)$.
  Compute offset error $\epsilon(t)$ for $C_i(t)$, $C_d^a(t)$.
  $C_{i+1}(t) \Leftarrow C_i(t)$ refined at $\epsilon(t)$ highest error region(s).
  $i \Leftarrow i + 1$.
While ($\epsilon(t)$ highest error $> \tau$).

d. Because the algorithm "knows" more about the curve, improvements can be applied in a more optimal way than simply subdividing the curve at its midpoint as has been done in the past. Even for polynomial representations such as Bézier curves, it is common to split the curve at the middle of the parametric domain if the accuracy of the offset is not good enough. Using the global error measure, one can now split the curve near the parameter value with the highest error. This will usually result in requiring fewer subdivisions to achieve a given tolerance.

One can compute and refine the curve at the maxima of $\epsilon(t)$ only in each iteration. However, simultaneous refinement of all regions whose respective errors were larger than allowable was found to be much faster. The computation of $\epsilon(t)$ is much more demanding than single knot insertion. By using simultaneous refinement, this computation is fully exploited.

Figure 3.1 shows four stages of algorithm 3.1, using global refinement, operating on a chess pawn cross section. Single knots have been inserted in all parametric regions whose error was above the tolerance level. The number of control points and the respective error function $\epsilon(t)$ for each iteration are also provided in Figure 3.1. The error is improved by almost an order of magnitude on each iteration up to the required tolerance of 0.0001.

Finding approximations to offsets of surfaces are usually more difficult, but the above method can be applied to finding errors of offset surfaces as well. $\delta(t)$, $\psi(t)$ and $\epsilon(t)$ would be simply explicit surfaces instead of explicit curves, i.e., $\delta(u,v)$, $\psi(u,v)$ and $\epsilon(u,v)$. In Figure 3.2, this error bounding extension surface is used to automatically iterate, refine, and improve an offset B-spline surface to a specified tolerance. It is interesting to compare the two offset surfaces in Figure 3.2. They both have the same tolerance but the offset distance is different. The offset error increases as d becomes larger and therefore more refinements are required to achieve the same accuracy.

3.2 Better Approximation of Offsets

In section 3.1, a technique was developed to provide a global bound using a global error function. This error function can be used to attempt to reduce the maximum error by perturbing the control points instead of refinement, as in section 3.1. Ideally, for each control point, the gradient direction that maximizes the change in the error function would be computed and the control point would be moved in that direction. Such a computation is extremely expensive and slow and a compromise must be made. By refining and offsetting in the normal direction, it is known the offset approximation converges to the exact offset. Therefore, the normal direction is a simple candidate for a preferred direction to use. We will also see that this

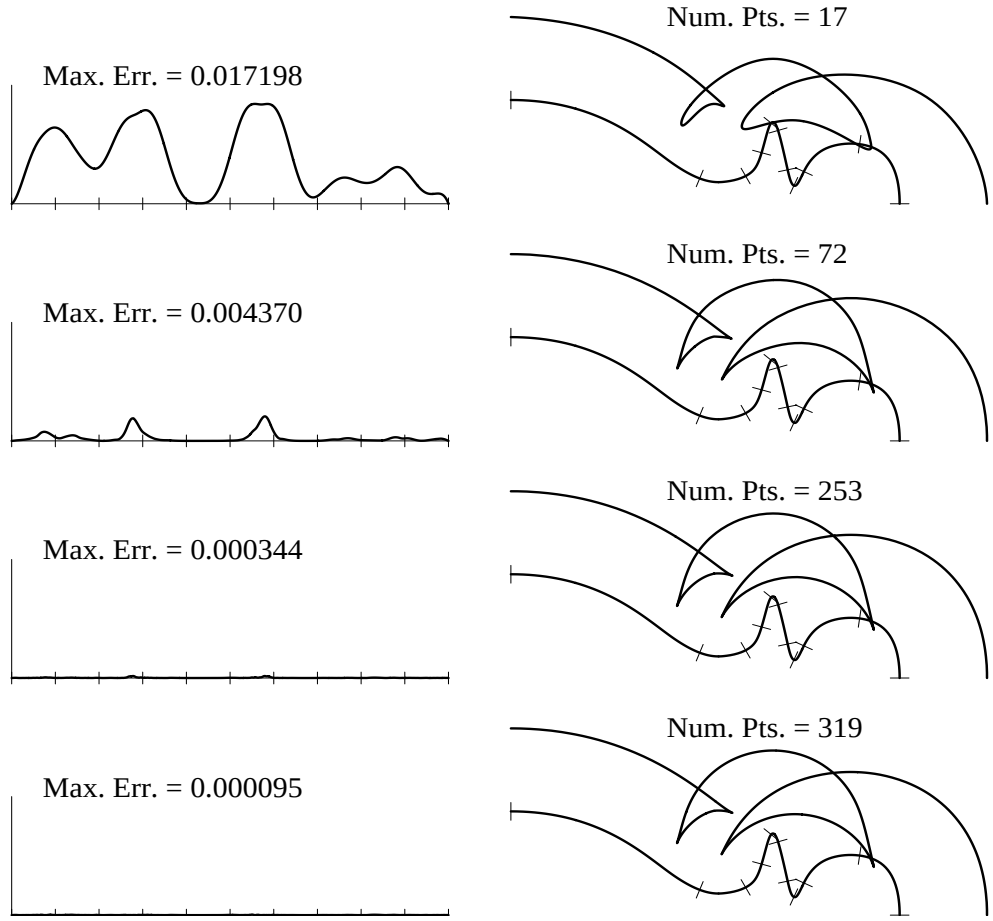

Figure 3.1. Four stages in global error bounding $\epsilon(t)$ and simultaneous refinement.

direction allows an exact representation of offset of quadratic circular curves with no refinement at all. The iterative process follows in algorithm 3.2.

In each iteration, the error function is computed and each control point is moved in the normal direction by the error amount at the node parameter value associated with this control point. This process repeats itself until no improvement is gained in the maximum error (i.e., no convergence) or the required tolerance is being achieved.

Figure 3.3 shows a unit circle composed of four 90 degree quadratic arcs. The first offset is obviously underestimated, but it converges quite quickly to the exact offset by moving only the corner points. These points have nonzero error, as can be seen from Figure 3.4 which also shows the respective error function as the process

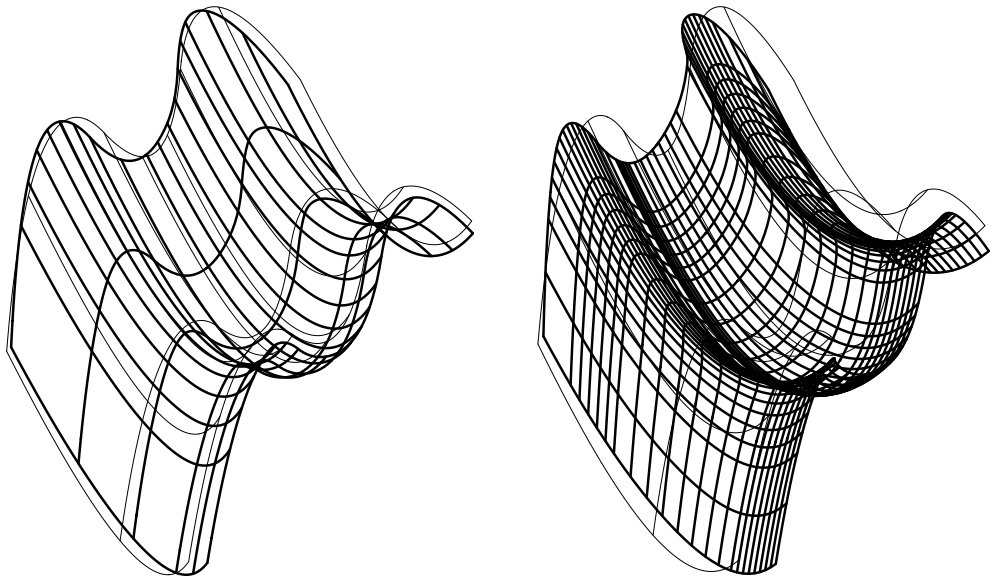

Figure 3.2. Error bounded offset surface, using simultaneous auto refinement.

converges. The points in the error function in Figure 3.4 at which the error is always zero correspond to the end points of the four 90 degrees Bézier segments forming the circle. Because the normals for the corner control point node values are in the direction (vectors $(\pm a, \pm a)$) pointing to the corner control points of a larger similarly represented circle, this process converges to an exact offset circle, with no refinement.

In Figure 3.5, the quadratic curve consists of three arcs of 120 degrees and three lines. The offset error along the line is zero and no improvement is applied there. The arcs can be improved to the exact representation. The required tolerance of 0.01 terminated this process at that accuracy as can be seen in Table 3.1.

Figure 3.6 is a case in which exact representation of the offset as a NURBs does not exist. Control points perturbation can improve the result, but refinement is still necessary to meet the required tolerance of 0.04 as can be seen from Table 3.2.

Figure 3.7 shows the same process applied to a unit sphere. This time the process does not converge to the exact representation because the normals at the node values of the corner points are not in the exact direction (vectors $(\pm a, \pm a, \pm a)$).

Algorithm 3.2

Input:
τ, required offset curve tolerance.
$C(t)$, input curve.
$C_d^a(t)$, offset approximation to input curve.
d, offset distance.

Output:
$\hat{C}_d^a(t)$, improved curve approximation.

Algorithm:
$C_0^a(t) \Leftarrow C_d^a(t)$.
$i \Leftarrow 0$.
$MaxErr \Leftarrow Infinity$.
Do
  Compute offset error $\epsilon(t)$ for $C(t)$, $C_i^a(t)$.
  $C_{i+1}(t) \Leftarrow C_i(t)$ perturbed according to $\epsilon(t)$ at node values.
  $LastMaxErr \Leftarrow MaxErr$.
  $MaxErr \Leftarrow \mathbf{min}(LastMaxErr, \epsilon(t)$ highest error$)$.
  $i \Leftarrow i + 1$.
While $(MaxErr > \tau$ and $MaxErr < LastMaxErr)$.

Even so, the improvement gained is quite significant. The right side of Figure 3.7 is the regular offset while the left side shows the same surface (and same number of control points) after perturbing it. Table 3.3 provides the convergence steps for this case, up to the prespecified tolerance of 0.01. Figure 3.7 right is stage 1 of Table 3.3 while Figure 3.7 left is stage 6.

3.3 The Offset Operator as a Modeling Tool

The offset operator can be used as a modeling tool. In fact, one can extend the global error finding method developed in section 3.1 and allow variable distance offsets as well. Given a parameter value, t, one needs to specify the offset distance required at that location. A scalar explicit distance function $d(t)$ (or $d(u, v)$ for

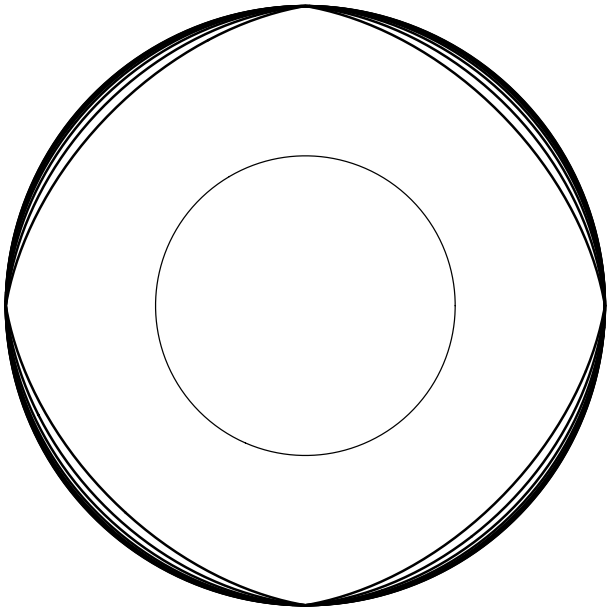

Figure 3.3. Control points perturbation converges to exact offset circle.

Figure 3.4. The error function convergence to zero, of the circle in Figure 3.3.

surfaces) having the same domain as $C(t)$ ($S(u,v)$) can be used. The only change that must be made to the method developed in section 3.1 is that equation (3.6) should now read:

$$\epsilon(t) = \psi(t) - d^2(t), \tag{3.8}$$

where d, which used to be constant, is now a distance function. In equation (3.7), it was shown that the global error bound depends on d, so now the extrema of $d(t)$ are used to bound the error. Algorithm 3.1 described in section 3.1 is identical to the one that should be used here. Figures 3.8 and 3.9 show some simple examples of the operator's power, for both curves and surfaces.

Table 3.1. Convergence errors of Figure 3.5 offset curve using perturbation.

Step	Error	Comments
1	0.49	
2	0.387	
3	0.291	
4	0.211	
5	0.149	
6	0.104	
7	0.071	
8	0.048	
9	0.033	
10	0.022	
11	0.015	
12	< 0.01	Tolerance is met.

Table 3.2. Convergence errors of Figure 3.6 offset curve using perturbation.

Step	Error	Comments
1	0.22	
2	0.197	
3	0.187	
4	0.182	
5	0.179	
$\vdots$	$\vdots$	
16	0.178	No improvement - refinement stage
21	0.083	
22	0.040	
23	< 0.04	Tolerance is met.

3.4 Trimming Self-Intersection Loops

Two types of loops are sometimes created in $C_d^a(t)$ when $C(t)$ is a C^1 continuous curve. If $\kappa(t)$, the curvature of $C(t)$, is larger than $\frac{1}{d}$, where d is the offset distance, a loop will be formed (see Figure 3.10). Because this loop is local to a region in which the curvature is too high, this type of loops will be referred to as a *local loop*. However, not all loops resulting from offset operations are of this kind. Some of the loops formed, as can be seen in Figure 3.11, are the result of two separate regions

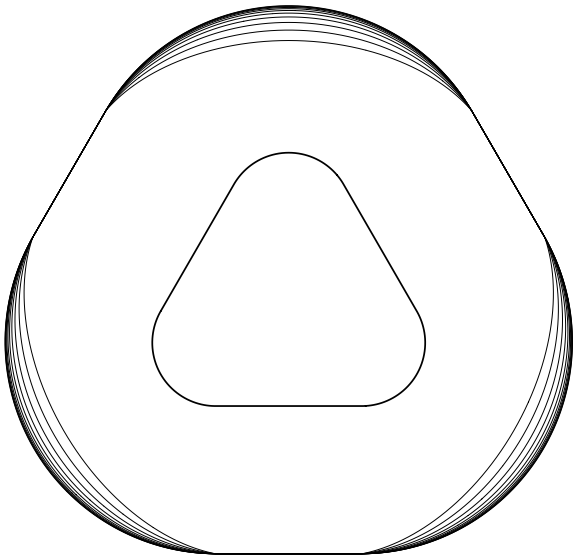

Figure 3.5. Error function convergence to zero, for three 120 degrees arcs in curve.

in $C(t)$ so close that the offset curve in those regions intersects itself. This type of loop is referred to as a *global loop*.

Detection of these loops is a difficult problem. A search for cusps was suggested as a method to detect local loops [35]. However, because $\mathcal{C}_d^a(t)$ is only an approximation, it is possible that no cusps will be formed (see first (top) stage of Figure 3.1). Moreover, the cusps, when detected, must be grouped in pairs, which is not a natural process using this technique. We use a more robust method to correctly detect all loops.

Let $\mathcal{T}(t)$ be the tangent vector to $\mathcal{C}_d(t)$ and let $\kappa(t)$ be the curvature of $C(t)$. Luckily, local loops have a distinct characteristic that when $\kappa(t_0) = \frac{1}{d}$, $\|\mathcal{T}(t_0)\| = 0$, and $\mathcal{C}_d(t)$ has a cusp at t_0 (see [28] and appendix 1). So, if $C(t)$ is curvature continuous, each time $\kappa(t) = \frac{1}{d}$ and $N(t) = N_o(t)$, $\|\mathcal{T}(t)\| = 0$. If $\kappa(t) > \frac{1}{d}$ and the normals coincide, $\mathcal{T}(t)$ flips its direction 180^o. When $\kappa(t)$ continuously changes from $< \frac{1}{d}$ to $> \frac{1}{d}$ and then back to $< \frac{1}{d}$ and the normals coincide, two cusps will be formed in $\mathcal{C}_d(t)$ at the places where $\kappa(t) = \frac{1}{d}$.

Using this characteristic, the cusp pairs can be identified by finding the zero set

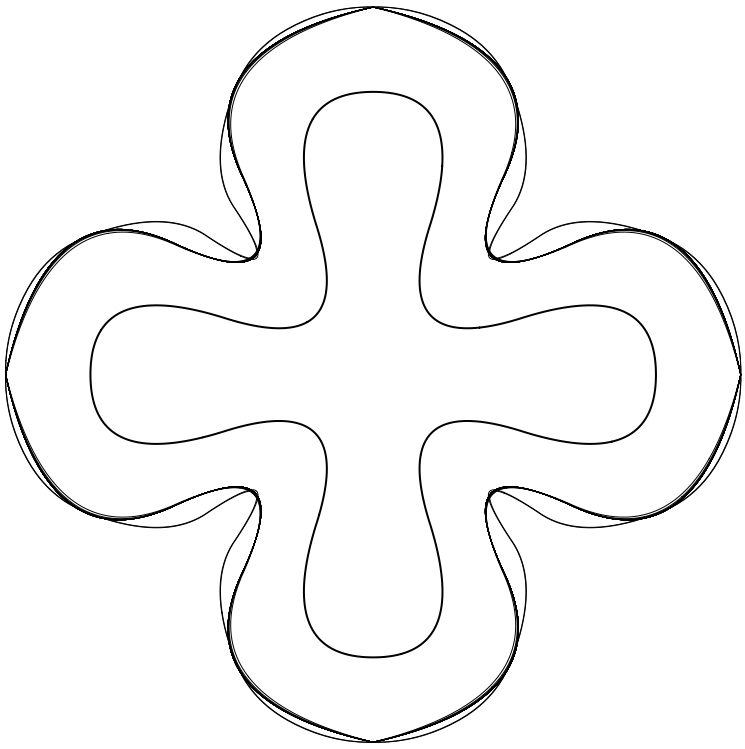

Figure 3.6. The error function does not convergence to zero, for general curves.

of $\tau(t) = \langle \mathcal{T}(t), T(t) \rangle$. The regions where $\tau(t)$ is negative are the regions where $\mathcal{T}(t)$ flips its direction (i.e., normals coincide and $\kappa(t) > \frac{1}{d}$). Figure 3.12 demonstrates this process on the pawn cross section in Figure 3.1. The tangent curves $T(t)$ ((a) in Figure 3.12) and $\mathcal{T}(t)$ ((b) in Figure 3.12) have been derived. Their dot product ((c) in Figure 3.12), $\tau(t) = \langle T(t), \mathcal{T}(t) \rangle$, is computed and used to identify the two local loops in the resulting offset approximation in its two negative regions ((d) in Figure 3.12). Once the two loops have been identified, they can be trimmed away ((e) in Figure 3.12).

The usage of $\tau(t)$ to identify local loops make this process more robust, even if no cusps are formed in the offset approximation. The tangent vector, $\mathcal{T}(t)$, still flips its direction and still makes $\tau(t)$ negative (Figure 3.12 (c)). Furthermore, by detecting the negative regions of $\tau(t)$ the cusps are virtually paired because each cusp pair is the negative $\tau(t)$ region boundary.

Table 3.3. Convergence errors of Figure 3.7 offset sphere using perturbation.

Step	Error	Comments
1	2.574	
2	1.43	
3	0.964	
4	0.804	
5	0.733	
6	0.691	
⋮	⋮	
20	0.616	No improvement - refinement stage
21	0.296	
22	0.238	
23	0.186	
24	0.146	
25	0.115	
26	< 0.01	Tolerance is met.

Once a local loop has been identified using $\tau(t)$, the algorithm splits the curve into three parts, the region before the first cusp, the region after the second cusp, and the region between the two cusps. The third part, between the cusps, must be deleted. The first two should then be intersected against each other to find the self intersection point using standard curve-curve intersection algorithms [15, 43, 60], trimmed properly to the intersection point, and then merged back. See Figures 3.10 and 3.11 for some examples.

Global loops have no such characteristic and are therefore more difficult to isolate. It is necessary to find all the self-intersections of a curve. However, a curve which is monotone in one dimension can never intersect itself. Therefore, one way to approach this problem is to split the curve into monotone subcurves, intersect all the subcurves against each other using curve-curve intersection algorithms, and isolate all the self-intersection points if any. Loops can now be formed by tracing the self-intersection points along the parameter space. Given an intersection point P_i, when $C(t_i^1) = C(t_i^2)$, the sign of the dot product $\langle T(t_i^1), N_o(t_i^2) \rangle$ can be used

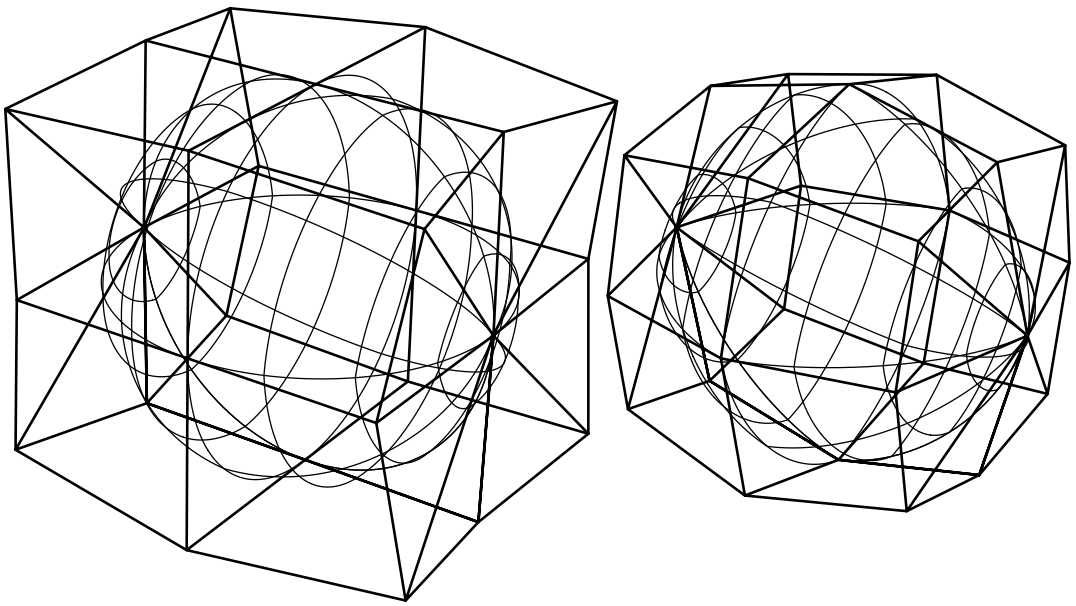

Figure 3.7. Control points perturbation can also improve offset surface accuracy.

to determine if a loop is to be purged or not. Given P_i, the normal $N_o(t_i^2)$ defines the relative position of the original and offset curve. If the dot product is negative, it means the intersecting curve (with tangent $T(t_i^1)$) in P_i is closer locally (P_4 in Figure 3.13) to the original curve than the offset amount. Because curves are continuous, it implies the whole loop is closer than the offset amount and therefore should be removed (loop 4 in Figure 3.13). Similarly, the dot product is found to be positive in P_5 in Figure 3.13 so in the neighborhood of P_5, loop 5 distance to the offset curve in the N_5 direction is larger than the offset amount and therefore loop 5 is locally (and globally) valid. The loops are tested while following the parameter values of the curve from its beginning to its end. For each intersection of an untested loop i, the tangent T_i of the current curve parameter is computed along with the offset normal N_i of the *other* curve at the intersection point i. Using the example in Figure 3.11, loop 1 is tested first. $\langle T_1, N_1 \rangle$ is found to be negative and therefore loop 1 is purged. Because $\langle T_2, N_2 \rangle$ is positive loop 2 should not be purged, etc.

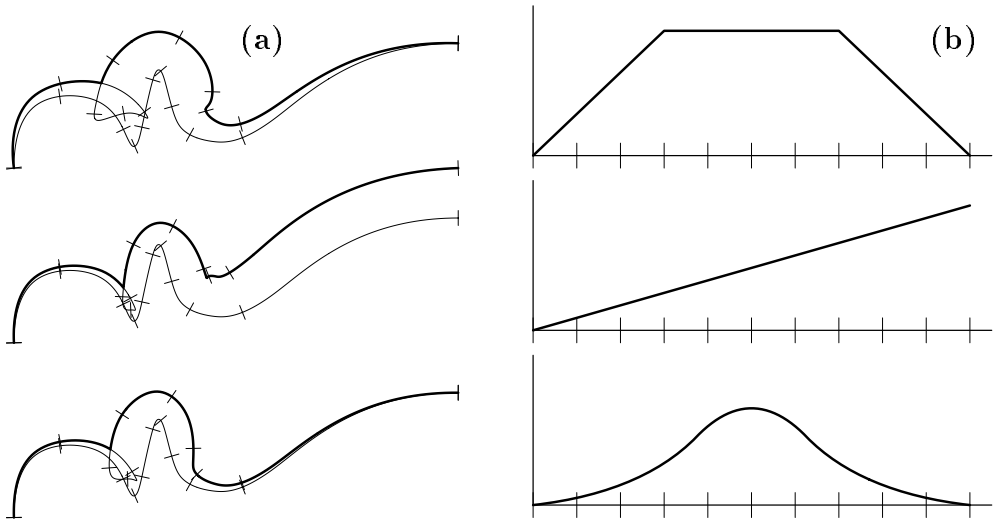

Figure 3.8. Variable distance offset (a) using a scalar distance function (b).

This approach has been used to trim out the global loops of Figure 3.11.

The curve offset local loop detection method may be extended to surfaces as well. If the surface radius is smaller than the offset distance, the normal of the offset surface may flips its direction. If both principal curvatures are the same and equal to κ (i.e., an oblique point which is locally a sphere of radius $\frac{1}{\kappa}$), then an offset by more than $\frac{1}{\kappa}$ will cause both tangents in the isoparametric directions to be flipped or the normal of the offset will point to the same direction. If, however, the two principal curvatures are different (say $\kappa_1 > \kappa_2$), the normal to the surface will be flipped when the offset distances passes $\frac{1}{\kappa_1}$, and flipped back when later the distance grows beyond $\frac{1}{\kappa_2}$. Because exact spherical shapes are fairly rare and simple to deal with, the normal flipping may be a useful tool in *detection* of self-intersections. Let $N(u,v)$ be the normal surface to the original surface $S(u,v)$ and $\mathcal{N}(u,v)$ be the normal surface to the offset surface $\mathcal{S}(u,v)$, and define

$$\nu(u,v) = \langle N(u,v), \mathcal{N}(u,v) \rangle \,. \tag{3.9}$$

Equation (3.9) can be used to detect self-intersections. If $\nu(u,v) < 0$, there must be a self-intersection. In Figure 3.14 the apex of $S(u,v)$ is an oblique point and near

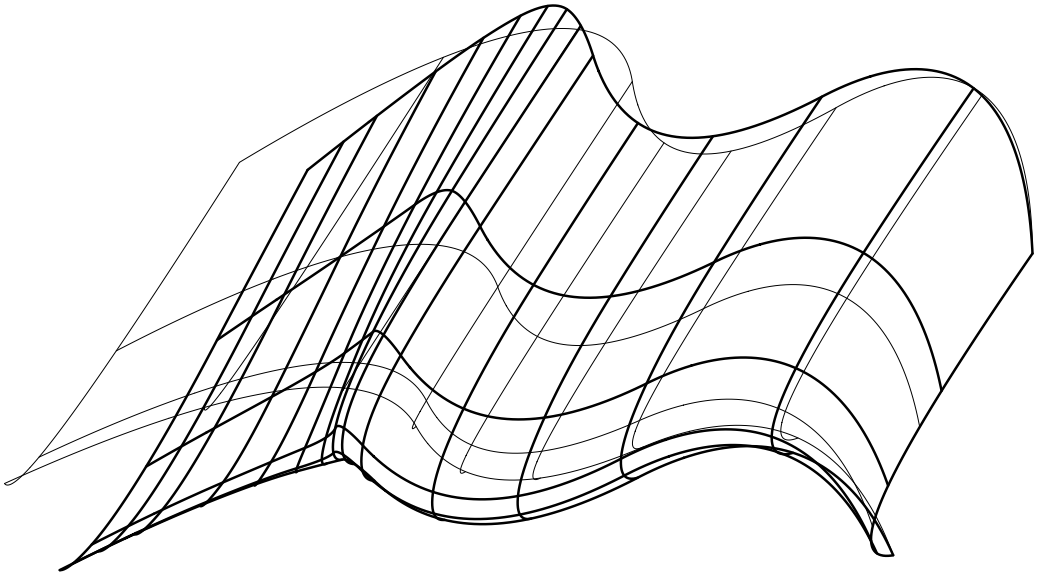

Figure 3.9. Variable distance surface offset (u direction linear, v constant).

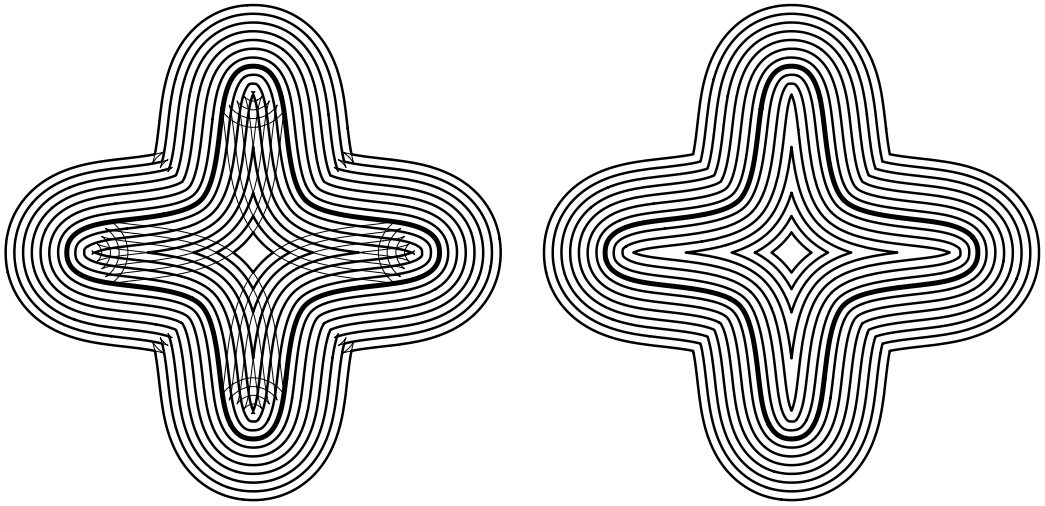

Figure 3.10. Offset operation local loops are trimmed using a distinct characteristic.

it both $N(u,v)$ and $\mathcal{N}(u,v)$ point to the same direction because both $\frac{1}{\kappa_1} < d$ and $\frac{1}{\kappa_2} < d$ or both tangents to the $\mathcal{S}(u,v)$ in the isoparametric directions are flipped. However, in the intermediate region of $\mathcal{S}(u,v)$ the u direction (surface of revolution circular direction) curvature has reached the offset distance and so the tangents in the u isoparametric direction are flipped, while the tangents in the v isoparametric direction are not. In that region, obviously $\frac{1}{\kappa_1} > d > \frac{1}{\kappa_2}$ and ν is negative, which

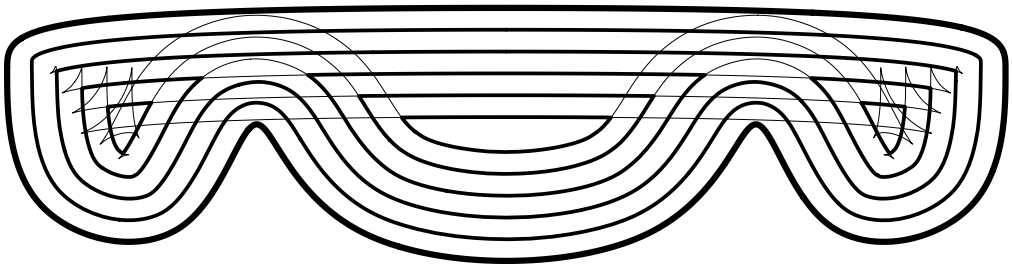

Figure 3.11. Global loop are being trimmed using numerical techniques.

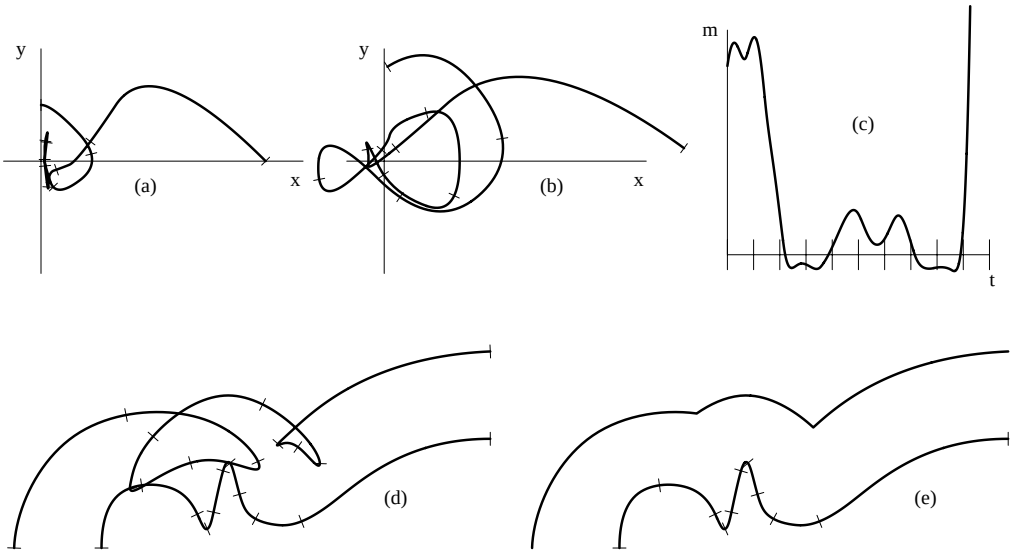

Figure 3.12. Product of a curve and its offset tangents used to identify local loops.

together signal the self-intersection.

Trimming surface loops are much more difficult because, in general, they are not isoparametric. Because an analytic approach was not feasible, an approach which subdivided the surface into polygons and detected self-intersections on this approximation was used. To simplify the process, it was assumed the the original surface was completely visible from the z direction (envisioning a 3 axis pocket for NC applications). The z was used as the sweeping axis, in algorithm 3.3, to minimize the number of polygon-polygon intersection tests, in the detecting of possible self-intersections in the offset. **minimumZ** and **maximumZ**, in algorithm 3.3,

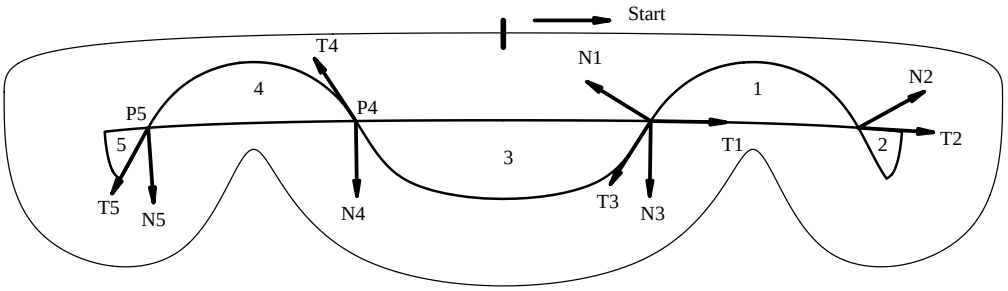

Figure 3.13. Global loop classification is based on $\langle N_i(t_i^1), T_i(t_i^2)\rangle$ sign.

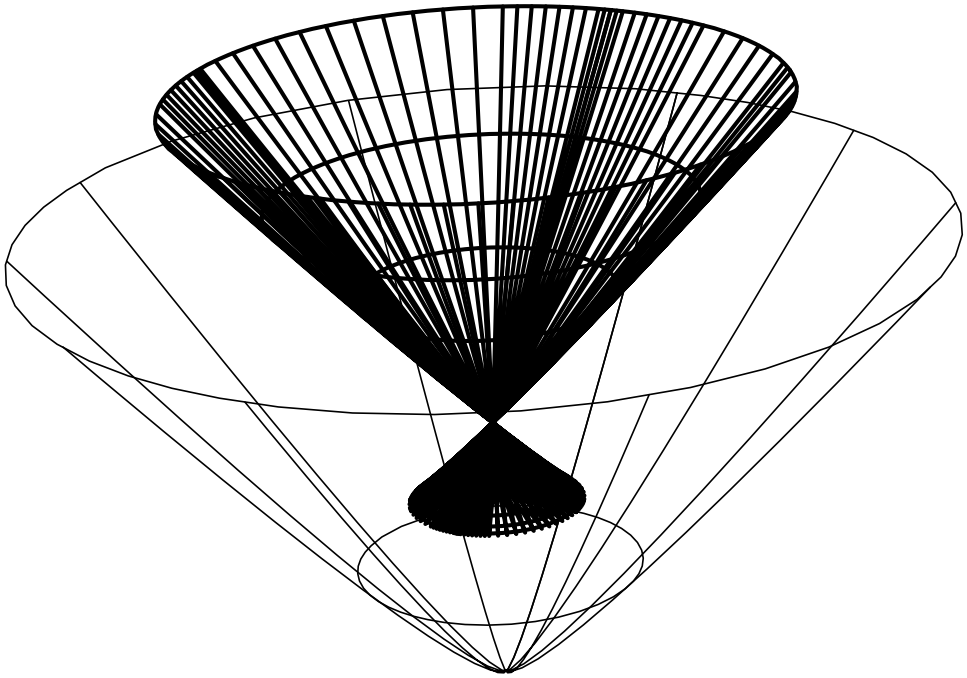

Figure 3.14. Offset surface self-inter. may be detected using $\langle N(u,v), \mathcal{N}(u,v)\rangle$ sign.

return the z extrema of given object, **IntersectPolyPoly** finds the linear segment of two intersecting polygons, and finally **isSurface** and **isFlat** are two predicates.

As if these difficulties are not enough, the topology of the self-intersection can be extremely complex. Figure 3.15 left shows an offset surface of a simple surface. The centered region of the surface has large enough curvature to cause the offset surface to intersect itself. Extremely complex self-intersection loops are generated as can be seen on the right of Figure 3.15 which shows the self-intersection curves