Adaptive Isocurves Based Rendering for Freeform Surfaces

Gershon Elber and Elaine Cohen

UUCS-92-040

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

December 2, 1992

Abstract

Freeform surface rendering is traditionally performed by approximating the surface with polygons and then rendering the polygons. This approach is extremely common because of the complexity in accurately rendering the surfaces directly. Recently, several papers presented methods to render surfaces as sequences of isocurves. Unfortunately, these methods start by assuming that an appropriate collection of isocurves has already been derived. The algorithms themselves neither automatically create an optimal or almost optimal set of isocurves so the whole surface would be correctly rendered without having pixels redundantly visited nor automatically compute the parameter spacing required between isocurves to guarantee such coverage.

In this paper, a new algorithm is developed to fill these needs. An algorithm is introduced that automatically computes a set of almost optimal isocurves covering the entire surface area. This algorithm can be combined with a fast curve rendering method, to make surface rendering without polygonal approximation practical.

Adaptive Isocurves Based Rendering for Freeform Surfaces *

Gershon Elber† and Elaine Cohen
Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

December 2, 1992

Abstract

Freeform surface rendering is traditionally performed by approximating the surface with polygons and then rendering the polygons. This approach is extremely common because of the complexity in accurately rendering the surfaces directly. Recently, several papers [1, 16, 18, 25, 26, 23] presented methods to render surfaces as sequences of isocurves. Unfortunately, these methods start by assuming that an appropriate collection of isocurves has already been derived. The algorithms themselves neither automatically create an optimal or almost optimal set of isocurves so the whole surface would be correctly rendered without having pixels redundantly visited nor automatically compute the parameter spacing required between isocurves to guarantee such coverage.

In this paper, a new algorithm is developed to fill these needs. An algorithm is introduced that automatically computes a set of almost optimal isocurves covering the entire surface area. This algorithm can be combined with a fast curve rendering method, to make surface rendering without polygonal approximation practical.

1 Introduction

Most surface rendering systems render a set of polygons which approximate the model representation instead of rendering the surfaces directly. Polygon rendering is usually more efficient and numerically robust than direct surface rendering. Unfortunately, the polygonized model is only an approximation to the real surface and aliasing occurs. Intensity (Gouraud) and normal (Phong) interpolation schemes [7] were developed to overcome the visual effects caused by C^1 discontinuities across boundaries between polygons. The piecewise linear appearance of the boundary and silhouette edges is improved by increasing, as necessary, the number of polygons globally or adaptively [15]. Subdivision of trimmed surfaces into polygons for rendering purposes is also a difficult problem [24]. On the other hand, rendering the surface as a set of isocurves is appealing since this data is exact, eliminating some of the need for the anti-aliasing techniques developed for rendering of polygonal approximation. Furthermore, rendering iso curves in this way reduces the complexity necessary to support trimmed surfaces, as will be demonstrated, as well as reducing the complexity involved in texture mapping computations. Several methods were developed in

*This work was supported in part by DARPA (N00014-88-K-0689). All opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

†Appreciation is expressed to IBM for partial fellowship support of the first author.

recent years to render curves and attempt to render surfaces using isocurves [1, 16, 18, 23, 25, 26]. However, previous methods do not automatically extract an optimal or almost optimal set of isocurves from a surface S so the rendering of S using the isocurves and those methods cannot be guaranteed to not miss any pixel representing the surface in image space. In [23] this problem is partially addressed by using a heuristic subdivision based approach. The surface is subdivided each time the isocurves spacing varies more than a specified tolerance, an approach which can lead to rendering a large number of small patches in which the fixed initialization cost per curve when using the (adaptive) forward differencing algorithm would greatly increase the total cost of rendering that surface. In [25], isocurves at equally spaced intervals, $u = n\delta u$, are used and which can lead to pixels either being missed or redundantly rendered, as can be seen in the middle of Figure 1 (a) and in the center of Figure 2 (a). In [26], the isocurves are adaptively spaced using bounds extracted from the convex hull of the distance function between two adjacent isocurves, $d(v) = f(u+\delta u, v) - f(u, v)$. Each isocurve continues to span the entire u domain of the surface. Therefore, the redundancy demonstrated in Figures 1 (a) and 2 (a) would continue to reduce optimality in [26].

This paper presents an algorithm that produce curves which are not extremely redundant and do not miss pixels. The new algorithm *adaptively* extracts isocurves and covers the entire surface in an almost optimal way. The polygon primitive is replaced by an isocurve drawn with finite thickness.

In the ensuing discussion we will need the concept of *valid coverage*

Definition 1 *A set of isocurves $\mathcal{C}$ of a given surface S is called a* valid coverage with respect to some constant δ *if for any point p on S there is a point, q, on one of the isocurves in $\mathcal{C}$, such that $\|p - q\|_2 < \delta$, where $\|\cdot\|_2$ denotes Euclidean distance.*

Surface rendering algorithms using isocurves should comply with definition 1 where δ is in the order of half of the image pixel size. All pixels representing S in the image are then guaranteed to be covered by at least one isocurve. Definition 1 assumes S is in the viewing space. A surface in viewing space has its x and y coordinates aligned with the image plane coordinates u, and v. That is, $u = x$ and $v = y$. However, the z coordinate of the surface is still accessible. The viewing space automatically accounts for distant and small surfaces which requires less effort to render, since the perspective transformation has already been applied. Under some conditions, it is sufficient to compute the iso-distance using the x and y surface components only, since coverage of the image plane is the concern. Unfortunately, ignoring z may result with incorrect rendering when the right conditions are not met. We will discuss this issue further later.

The *optimality* of the valid coverage is the second concern:

Definition 2 *A coverage for a given surface is considered* optimal *if it is valid and the accumulated length of its path is minimal, over all valid coverages.*

If one could compute the parameter spacing required for a valid coverage for a given surface in a given scene, extraction of all isocurves at that spacing might be suboptimal as can be seen from the middle of the surface in Figure 1 (a) and the center of the surface in Figure 2 (a). Furthermore, since isocurves speed varies across the parameter space, local dynamic change of the parameter spacing is required as seen in Figures 1 (b) and 2 (b), to improve the coverage optimality.

Using isocurves as the coverage for a surface, we define *adjacency* and *iso-distance* between isocurves.

Definition 3 *Two isocurves of surface $S(u, v)$, $C_1(u) = S(u, v_1)$, $u \in [u_1^s, u_1^e]$ and $C_2(u) = S(u, v_2)$, $u \in [u_2^s, u_2^e]$, $v_1 \le v_2$, from a given set $\mathcal{C}$ of isocurves forming a valid coverage for S are considered* adjacent *if, along their common domain $\mathcal{U} = [u_1^s, u_1^e] \cap [u_2^s, u_2^e]$, there is no other isocurve from $\mathcal{C}$ between them. That is, there does not exist $C_3(u) = S(u, v_3) \in \mathcal{C}$, $u \in [u_3^s, u_3^e]$ such that $v_1 \le v_3 \le v_2$ and $[u_3^s, u_3^e] \cap \mathcal{U} \neq \emptyset$.*

Definition 4 *The* iso-distance *curve $\Delta_{12}(u)$ between two isocurves $C_1(u) = S(u, v_1)$, $C_2(u) = S(u, v_2)$, is $\|C_1(u) - C_2(u)\|_2$.*

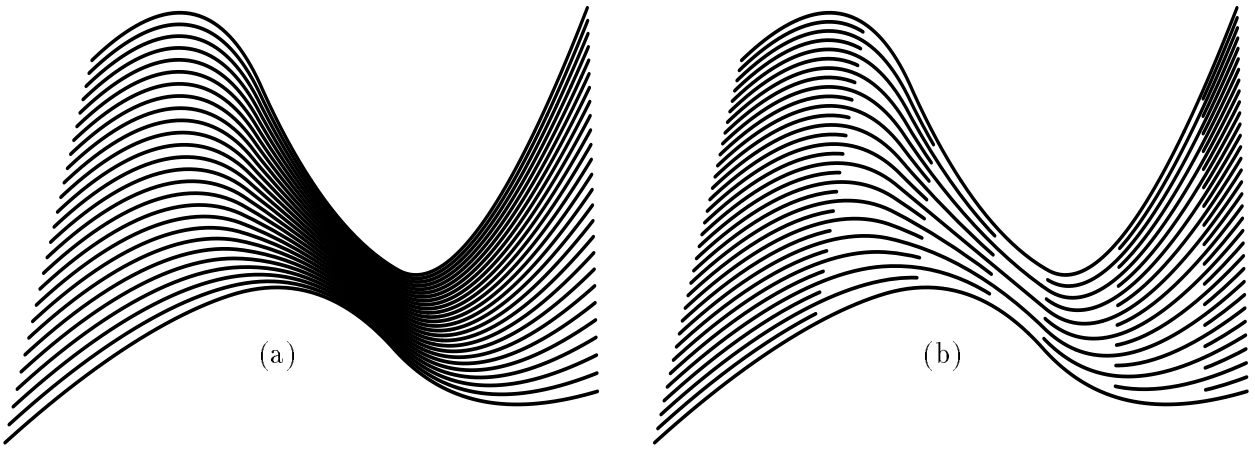

Figure 1: Isocurves are obviously not an optimal solution as a valid coverage for this surface (a). Adaptive isocurves are more optimal and their coverage is valid as well (b).

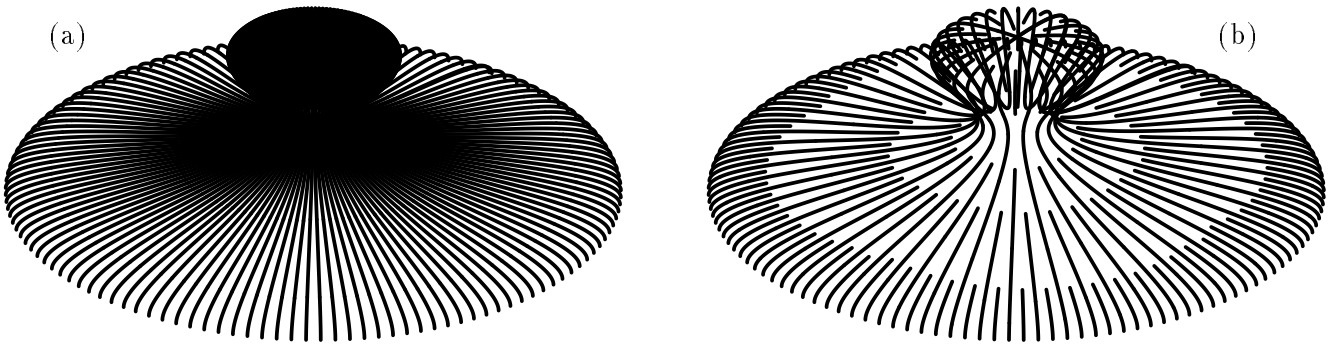

Figure 2: Utah teapot lid. Constant parameter spacing causes redundancy in coverage (a) mostly eliminated by adaptive extraction of isocurves (b). Both provide a valid coverage with respect to same δ.

Section 2 provides the background for the algorithm developed in section 3. Rendered results using this new isocurve based method are presented in section 4. The implementation uses the NURBs surface representation in the Alpha_1 solid modeler.

2 Background

Suppose $C_1(u) = S(u, v_1)$ and $C_2(u) = S(u, v_2)$ are two adjacent isocurves in a parametric surface $S(u, v)$ with common u domain $[u_s, u_e]$ from a given set of curves $\mathcal{C}$. Let R be a region of S over the domain $[u_s, u_e] \times [v_1, v_2]$. Let $\hat{R}$ be a ruled surface approximation for R, $\hat{R}(u, v) = C_1(u) * v + C_2(u) * (1 - v)$, $v \in (0, 1)$.

Lemma 1 *If* $\Delta_{12}(u) = \|C_1(u) - C_2(u)\|_2 < \delta$ *for all* u, *then* $\mathcal{C} = \{C_1(u), C_2(u)\}$ *is a* valid coverage *of* $\hat{R}$ *with respect to* $\frac{\delta}{2}$.

Proof $C_1(u)$ and $C_2(u)$ are adjacent between u_s and u_e and no isocurve in $\mathcal{C}$ exists between them by definition 3. In order for $C_1(u)$ and $C_2(u)$ to form a valid coverage for $\hat{R}$, for any point $p \in \hat{R}$, there must exist a point q in either $C_1(u)$ or $C_2(u)$ such that $\|p - q\|_2 < \frac{\delta}{2}$. Given a point $p \in \hat{R}$, there exists some v^* such that $p \in \hat{R}(u, v^*)$ curve, $p = \hat{R}(u^*, v^*)$. Therefore p is on the line from $C_1(u^*)$ to $C_2(u^*)$ which

is bounded in length to be not greater than δ, since by hypothesis $\Delta_{12}(u^*) < \delta$. In other words, either $\|C_1(u^*) - p\|_2 \leq \frac{\delta}{2}$ or $\|C_2(u^*) - p\|_2 \leq \frac{\delta}{2}$ (or both). ■

Lemma 1 provides a condition on the validity of the coverage of a ruled surface, $\hat{R}$, by two of its boundary curves, $C_1(u)$ and $C_2(u)$. One might need to further verify that $\hat{R}$ sufficiently approximates R. This might necessitate the computation of a bound on the distance between $\hat{R}$ and R [11], curvature analysis of R [9], or alternatively analysis of the speed variance of R [10]. We will refer to this approximation validity condition as $(\hat{R} \rightsquigarrow R)$.

For $\delta = 1$ pixel, the special rendering case, the surface is approximated by strips of ruled surfaces, each approximately a pixel wide, usually a more accurate approximation than the polygons used for rendering. Hence, unless subpixel results are wanted (in which case δ can be made smaller) it is not necessary to do further bounding on the distance between $\hat{R}$ and R.

As stated in lemma 1, this condition is sufficient, but is it necessary that $\Delta_{12}(u) < \delta$? For a very skewed surface resulting from a non isometric mapping this condition could be too restrictive since the isodistance could be much larger than the real minimal distance between the curves. One might expect the penalty for this assumption to be quite high. In practice, it was found to be reasonably well behaved for such surfaces as is demonstrated in section 4 since the highly non-isometric mapping is frequently isolated to be in small regions of the domain, and even then provided competitive results compared to methods using nonadaptive isoparametric curves.

The iso-distance function $\Delta_{12}(u)$ between the two isocurves $C_1(u) = (c_1^x(u), c_1^y(u), c_1^z(u))$ and $C_2(u) = (c_2^x(u), c_2^y(u), c_2^z(u))$ in the surface S can be efficiently computed and is equal to:

$$\Delta_{12}(u) = \sqrt{(c_1^x(u) - c_2^x(u))^2 + (c_1^y(u) - c_2^y(u))^2 + (c_1^z(u) - c_2^z(u))^2}. \tag{1}$$

The sum, difference, and product of two scalar curves are closed for polynomial (Bézier), piecewise polynomial (B-spline), or rational representations (NURBs). Furthermore, efficient algorithms exist, as we shall show below, for finding the form of the sum and product in the Bézier representation. On the other hand, square roots are not representable, in general, and therefore, are not closed under the above domains. Instead, one can find and use the representation for the square of the iso-distance, as is done in section 3:

$$\Delta_{12}^2(u) = (c_1^x(u) - c_2^x(u))^2 + (c_1^y(u) - c_2^y(u))^2 + (c_1^z(u) - c_2^z(u))^2. \tag{2}$$

In order to *symbolically* represent $\Delta_{12}^2(u)$ as a scalar Bézier or NURBs curve, one must be able to represent the scalar curve which is the symbolic sum, difference, and product of scalar curves, as a single Bézier or NURBs curve. Methods to represent the result of the above operators applied to Bézier or NURBs curves, as a single Bézier or NURBs curves are briefly presented below.

2.1 Curve Addition

The symbolic computation of sum and/or difference of two scalar Bézier or NURBs curves is achieved by computing the sum and/or difference of their respective control points [5, 6, 12], once the two curves are in the same space. This requirement can be met by representing them as curves with the same order (using degree raising [3, 4] on the lower order one, if necessary) and the same continuity (using refinement [2] of knot vectors for NURBs). Specifically, in $\Delta_{12}^2(u)$ computation (equation (2)), $C_1(u)$ and $C_2(u)$ are adjacent isocurves and therefore share the same order and continuity (knot vector) [12],

$$C_1(t) \pm C_2(t) \quad = \quad \sum_{i=0}^{k} P_i B_{i,\tau}^k(t) \pm \sum_{i=0}^{k} Q_i B_{i,\tau}^k(t)$$

$$= \sum_{i=0}^{k} (P_i \pm Q_i)\, B_{i,\tau}^{k}(u). \tag{3}$$

2.2 Product Curve

Representation for the product of two scalar curves is the second symbolic tool required. For Bézier curves [5, 6, 12],

$$\begin{aligned} C_1(t)C_2(t) &= \sum_{i=0}^{m} P_i B_i^m(t) \sum_{j=0}^{n} Q_j B_j^n(t) \\ &= \sum_{k=0}^{m+n} R_k B_k^{m+n}(t), \end{aligned} \tag{4}$$

$$\text{where} \qquad R_k = \sum_{\substack{i,\,j \\ i+j=k}} P_i Q_j \frac{\binom{i}{m}\binom{j}{n}}{\binom{k}{m+n}}.$$

It is also necessary to use representation of scalar products as part of representing sums of rational scalar curves [12].

Finding a representation for the product of two NURBs is far more difficult. A direct algorithmic approach has recently been developed [20] which supports symbolic computation of the coefficients of the product after finding the knot vector of the product curve. Since the direct approach is computationally expensive and complex to implement, one might choose to exploit the B-spline representation uniqueness property and compute the coefficients of the product by solving an equivalent interpolation problem [12]. Alternatively, one may consider subdividing the curves into Bézier segments at all the interior knots, compute the product, and merge the result back. However, the continuity information along the interior knots may be lost.

2.3 Zero Set of Curves

In addition to the above symbolic computation, the capability to compute the zero set of a scalar curve $C(u)$ is required. A simple subdivision based algorithm exploiting the convex hull property of both Bézier and B-spline curves can be formulated [17].

One can easily extend a zero set (root) finding algorithm to find the solution(s) of $C(u) = K$ for some constant K by subtracting K from all the coefficients of $C(u)$ to form a new curve $\hat{C}(u)$ in which the solution set of $C(u) = K$ is equivalent to that of the zero set of $\hat{C}(u)$:

$$\begin{aligned} \hat{C}(u) &= C(u) - K \\ &= \sum_{i=0}^{k} (P_i - K) B_i^k(u) \end{aligned} \tag{5}$$

using equation (3), since $\sum_{i=0}^{k} K B_i^k(u) \equiv K$.

3 Algorithm

Using the tools presented in section 2, we are ready to introduce the algorithm. Given two isocurves, $C_1(u)$ and $C_2(u)$, on a surface $S(u, v)$, one can symbolically compute the square of the iso-distance, $\Delta_{12}^2(u)$, between them. Furthermore, given some tolerance δ, one can compute the parameters along the curves where they are exactly δ iso-distance apart by computing the zero set of $(\Delta_{12}^2(u) - \delta^2)$, using equation (5) and a zero set (root) finding algorithm. By subdividing $C_1(u)$ and $C_2(u)$ at these parameters, the resulting set of pairs of curves, $\{C_1^i(u), C_2^i(u)\}$, have the property that their corresponding iso-distances are closer than δ or entirely farther apart than that, over their open interval domain. If the two curves have iso-distance less than δ, then the Euclidean distance tolerance condition is already met for that pair and the algorithm can terminate. If, however, the iso-distance between the two curves is too large, a middle isocurve between them, $C_{12}(u)$, is introduced and the same iso-distance test is invoked for the pairs $\{C_1(t), C_{12}(t)\}$ and $\{C_{12}(t), C_2(t)\}$.

Starting with the two u boundaries or two v boundaries of the surface, the algorithm invokes this iso-distance test recursively and insures that two adjacent isocurves will always be closer than some specified distance δ by verifying the iso-distance is not greater than δ. A middle isocurve is introduced only when the iso-distance is larger than δ, resulting with iso-distances between adjacent isocurves, as computed, will rarely be closer than $\frac{\delta}{2}$. Furthermore, since the resulting set of isocurves covers the entire surface S, the set of isocurves that result may serve as a valid coverage for S with distance δ.

Assuming isocurves are generated as constant v isoparametric curves, we can now formally state the algorithm. Algorithm 1 is the complete algorithm for an almost optimal extraction of isocurves to form a valid coverage. Line (1) in Algorithm 1 is the isodistance square computation as of definition 4 and computed using equation (2). If $\mathcal{Z}$ is empty, a single test at a single point may classify the pair, as is done in line (2) of Algorithm 1. If the pair is found to be close enough, no new curve is introduced and the empty set is returned in line (3). Otherwise, a new curve between the two curves is created and the algorithm is invoked recursively in lines (4) and (5). Alternatively, when $\mathcal{Z}$ is not empty, we subdivide $C_1(u)$ and $C_2(u)$ at all $u \in \mathcal{Z}$ in line (6) of the algorithm. The iso-distance between the sub-curve pairs resulted from the subdivision is always less than δ or always more than that in their entire domain. Therefore, each pair in the recursion in line (7) is classified into the $\mathcal{Z}$ empty cases above. Although omitted for clarity in Algorithm 1, the recursions invoked at line (7) of Algorithm 1 should provide $\Delta_{12}^2(u)$ so it would not be computed again. The union set returned in line (7), is the coverage set for the domain between the two curves.

The fact that the output consists of isocurves only simplifies further computation such as trimming the isocurves according to surface trimming curves, as is shown in section 4.

The resulting set of isocurves introduced through this algorithmic process forms a valid coverage while no two adjacent isocurves have iso-distance greater than the tolerance distance δ. In the limit, as δ approaches zero, no two isocurves will have iso-distance less than $\frac{\delta}{2}$. The speed in the v (ruled direction of $\hat{R}$) of a region, R, between two adjacent isocurves, $C_1(u)$ and $C_2(u)$, becomes constant, and an isocurve introduced in the middle of the parametric domain of R, will also be equally spaced between $C_1(u)$ and $C_2(u)$. Therefore, the coverage redundancy is bounded, since virtually no adjacent isocurves will have iso-distance of less than $\frac{\delta}{2}$.

A highly non-isometric mapping from parametric space to the Euclidean representation may reduce the algorithm efficiency since the iso-distance metric is a less accurate measure of the real minimum distance. In practice, it was found to be reasonably behaved.

There are some subtleties that have not yet been discussed. If the surface $VMin$ boundary is the same as the $VMax$ boundary, the algorithm will find their (zero) iso-distance below the distance tolerance δ and quit immediately. A cylinder is one such example in which the $VMin$ and $VMax$ boundary seams are shared. One should guarantee such cases are detected before invoking Algorithm 1. One way to guarantee

Algorithm 1

Input:

$S(u,v)$, input surface.
δ, maximum distance between isocurves.

Output:

S, the set of constant v isocurves of $S(u,v)$ adjacent within δ, covering S.

Algorithm:

```
    adapIsoCrvs( S, δ )
    begin
        C1(u), C2(u) ⇐ isocurves of S in u direction at VMin, VMax.
        return
            { C1(u) } ∪
            adapIsoCrvsAux( S, δ, u, VMin, VMax, C1(u), C2(u) ) ∪
            { C2(u) }.
        end
    end

    adapIsoCrvsAux( S, δ, VMin, VMax, C1(u), C2(u) )
    begin
        UMax, UMin ⇐ C1(u), C2(u) common u domain.
(1)     Δ²12(u) ⇐ squared iso-distance between C1(u) and C2(u).
        Z ⇐ zero set of (Δ²12(u) − δ²).
        if Z empty then
            R ⇐ S subsurface between C1(u) and C2(u).
            R̂ ⇐ C1(u) * v + C2(u) * (1 − v), v ∈ (0,1).
(2)         if Δ²12((UMax + UMin)/2) < δ² and (R̂⇝R) valid then
(3)             return φ.
            else
                VMid ⇐ (VMin + VMax)/2.
                C12(u) ⇐ isocurve of S at VMid from UMin to UMax.
                return
(4)                 adapIsoCrvsAux( S, δ, VMin, VMid, C1(u), C12(u) ) ∪
                    { C12(u) } ∪
(5)                 adapIsoCrvsAux( S, δ, VMid, VMax, C12(u), C2(u) ).
            end
        else
(6)         Subdivide C1(u), C2(u) at all u^i ∈ Z into {C1^i(u), C2^i(u)} pairs.
(7)         return ⋃i adapIsoCrvsAux( S, δ, VMin, VMax, C1^i(u), C2^i(u) ).
        end
    end
```

it is to insure the surface has no silhouette from the rendering direction (See [8] for silhouette detection). An alternative heuristic may be to always enforce at least one subdivision of the surface, which solves the problem for surfaces such as cylinders.

Another consideration is determination of which parametric direction should be used for isocurves extraction, u isocurves or v isocurves. In our implementation, we compute the maximum iso-distance between the u surface boundaries and the v surface boundaries, and prefer the direction with the smaller maximum. This heuristic promotes fewer, longer isocurves over numerous shorter ones in the hope that it will minimize the number of curves to be drawn.

Other image rendering aspects should be considered as well. The valid coverage is only one necessary condition. The surface normal for each pixel is also required for shading. An unnormalized representation of the surface normal, $\hat{n}(u,v) = \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v}$, can be computed symbolically [12], and represented as a vector surface whose coordinate functions are products and differences of surface partial derivatives.

Each isocurve output from Algorithm 1 is then piped into the curve renderer and is accompanied by the associated isocurve from the normal surface $\hat{n}$. The curve renderer uses the normal curve to compute the normal at all required locations. It is evident that the order of the normal curve is usually higher than that of the shape curve. Some curve renderers that use (adaptive) forward differencing [1, 16, 18, 25, 26] are implemented in hardware and are tuned to certain, usually low, orders. In such a case, the normal curve could be approximated as a sequence of lower (cubic) splines using known techniques for approximating higher order splines as lower order ones [12, 13, 14].

So far, we considered the iso-distance computed in coverage validation (definition 1) as the Euclidean distance in the viewing space. Under some conditions, it is sufficient to consult just the x and y components of the surface. If the projection of the surface in viewing space to the image plane is one to one, then only x and y need to be consulted. Ignoring z, two isocurves in S can have zero distance in the image plane (That is, they intersect in the image plane as can be seen in Figure 3 (a)), while distant apart in the viewing or object space (as seen in Figure 3 (b)), violating the one to one mapping requirement. If a surface has silhouette curves in the image plane, using only x and y in the computation of $\Delta^2_{12}(u)$ for two isocurves could result in invalid coverage, as would be the case in Figure 3, if $\Delta^2_{12}(u)$ were computed without using z. Preprocessing S to detect existence of silhouette curves in the image plane is equivalent to making sure the mapping is one to one. Therefore, one can easily determine if the iso-distance should be computed using z or without it, by determining if there are surface silhouettes.

4 Results

Several results are presented in this section, in addition to a discussion of some considerations on the complexity of the algorithm.

Figure 4 presents the well known Utah teapot model and a chess set rendered using the adaptive isocurve extraction algorithm.

Most techniques developed for enhancing image rendering quality can be applied to curve rendering. For example, Figure 5 shows a wood texture mapped version of the teapot. The fact that an isocurve is rendered only simplifies the texture mapping computation since one of the surface parameters (and the corresponding texture parameter) is fixed.

Surface isocurves are rendered into the Z-buffer one at a time with minimal memory overhead so complex scenes introduce no difficulties. Figures 4 and 5 show a complex chess scene rendered using this new algorithm.

Figures 6 and 7 demonstrates the use of solid texture to define a virtual planet and a camouflaged plane, using techniques presented in [21, 22], rendered using this isocurve based renderer.

Given a trimmed surface S, approximating it into a set of polygons for rendering is a difficult pro-

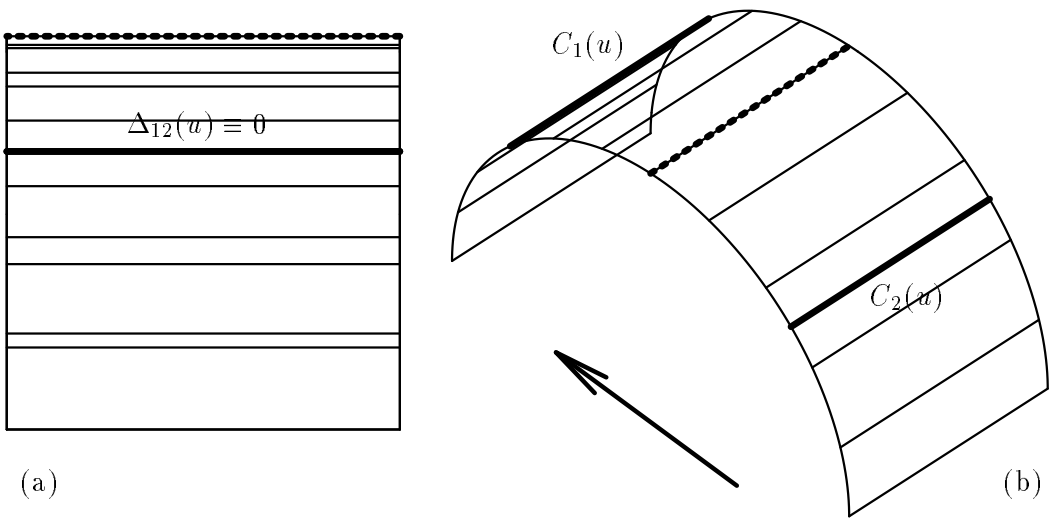

Figure 3: If S has silhouettes (dotted) in image plane, and only x and y are consulted in $\Delta_{12}(u)$ computation, $\Delta_{12}(u)$ may be found by Algorithm 1 to be wrongly zero, terminating prematurely.

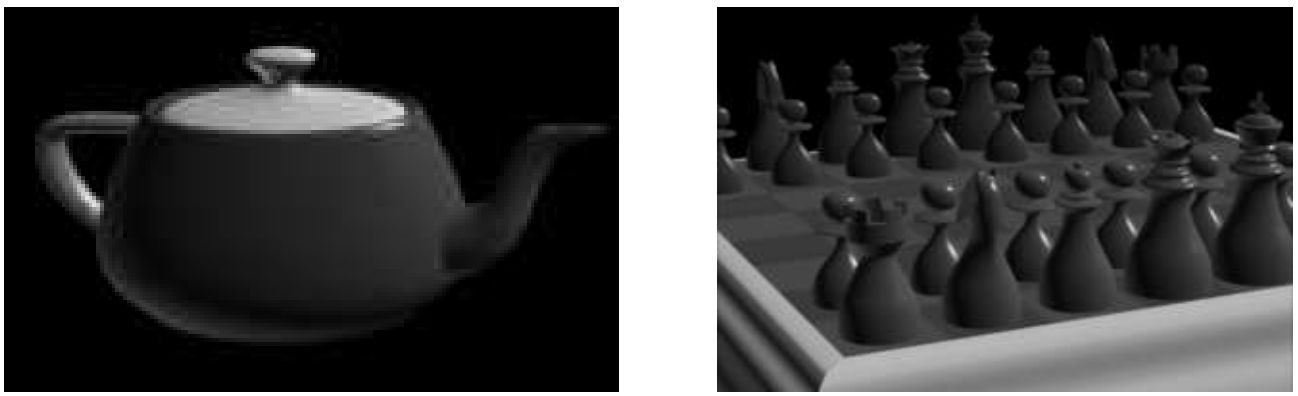

Figure 4: Utah teapot and a chess set adaptive isocurves rendered images.

Figure 5: Wood texture mapped on teapot model. Marble texture mapped chess set scene.

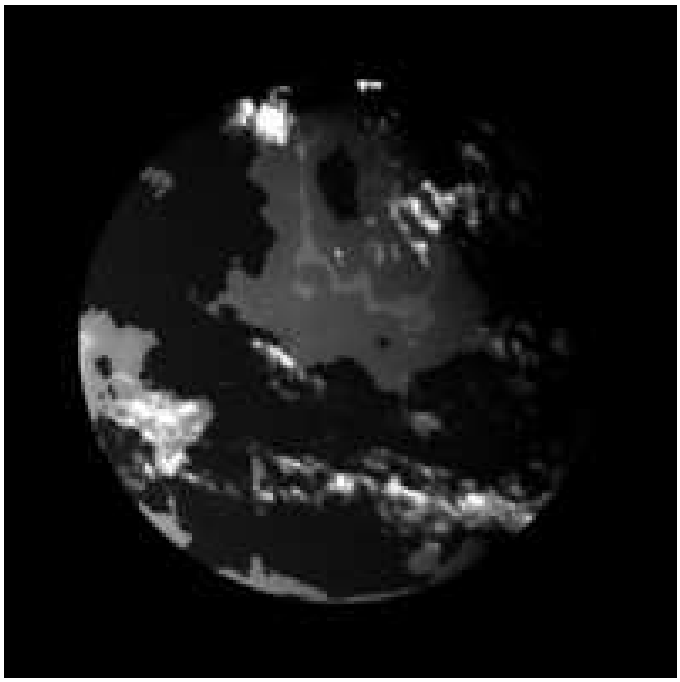

Figure 6: A virtual planet texture image using isocurves rendering.

Figure 7: A camouflage texture using isocurves rendering of an F16 model.

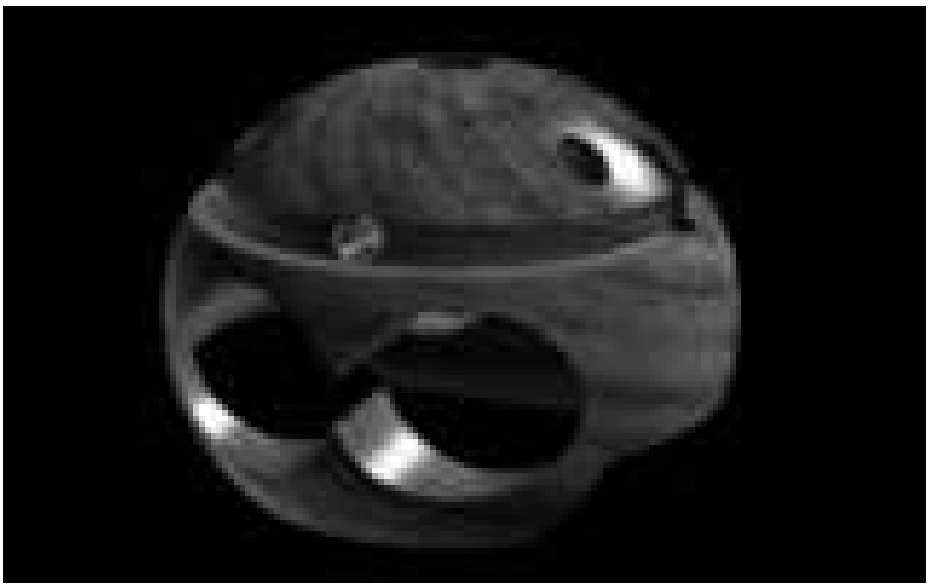

Figure 8: Trimmed NURBs model using adap. isocurves.

cess since the polygons must be properly trimmed [24, 19]. However, since this algorithm produces only isocurves, the clipping process is significantly simplified. A piston bridge model from a Diesel engine consisting of 27 trimmed NURBs surfaces was rendered in Figure 8 using wood texture.

One might also consider adaptively rendering the surface using variable width curves. Starting with very few but widely drawn isocurves, one could immediately provide a coarse shape of the surface which could be refined hierarchically into a more accurate image using more isocurves. Figure 9 shows six steps of such a process.

Consider the computational complexity of Algorithm 1. Let the number of isocurves in the output be N. For each isocurve in the output set, Algorithm 1 computes an iso-distance curve in line (1), for each of his neighbors when recursion occurs in lines (4) and (5). Since all but the first boundary curves have two neighbors, the number of iso-distance computations between curves is equal to $2N - 2$. Using the Bézier and or the B-spline representation, each iso-distance curve computation of a polynomial curve involves with 3 scalar curve subtractions, 3 scalar curve products and 2 scalar curve additions (equation (2)). All in all, the number of curve operations is linear in the number of curves in the output. The number of addition, subtraction and products for rational curves is somewhat higher because of the more complex addition and subtraction required, but is still linear in the output size.

Timing comparisons are difficult since they strongly depend on the complexity of the images and the realism that is attempted. All images throughout this paper have been created using a simple curve rendering technique that renders piecewise linear approximations to each curve. Without any special optimization, our implementation was time competitive with a regular adaptive polygonal based renderer which is part of the Alpha_1 solid modeler and produced equal quality imagery in approximately the same time. Furthermore, even though curves where rendered as piecewise linear polylines in our implementation, the computed highlights and rendering have more realistic appearance, as is demonstrated by the images throughout this paper, due to a better surface normal approximation. Usage of (adaptive) forward differencing might improve the overall algorithm performance, and further remove aliasing introduced by the isocurve piecewise linear approximation used. It is the authors' view that the software based computation per pixel (the shading model used), performed far more times than any other operation, has far higher cost than any other high level computation such as polygonal approximation or isocurve extraction.

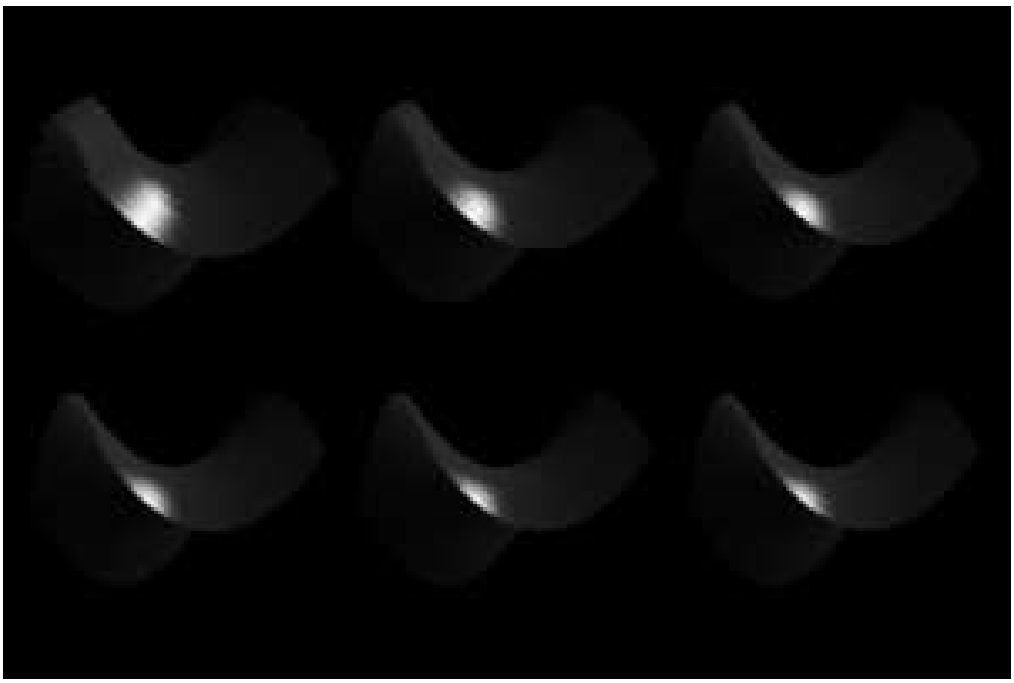

Figure 9: Six steps in coarse to fine rendering using adaptive isocurves.

5 Conclusion

A new algorithm is presented that automatically computes an adaptive valid coverage of a surface using isocurves. The existing ability to efficiently render curves combined with the almost optimal isocurves extraction method presented here makes isocurves rendering of surfaces a feasible alternative to polygon based rendering of surfaces. The simplicity of the algorithm, compared to the complexity involved in polygonal approximation of surfaces and especially, trimmed surfaces, and the need to deal with two dimensional polygonal entities during the scan conversion process, could make the isocurve rendering approach even more attractive in hardware based systems. The algorithm presented here efficiently reduces the problem of surface rendering to a simpler problem of curve rendering.

References

[1] S. Chang, M. Shantz and R. Rocchetti. Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing. Computer Graphics, Vol. 23, Num. 3, pp. 157-166, Siggraph Jul. 1989.

[2] E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics. Computer Graphics and Image Processing, 14, 87-111 (1980).

[3] E. Cohen, T. Lyche, and L. Schumaker. Degree Raising for Splines. Journal of Approximation Theory, Vol 46, Feb. 1986.

[4] E. Cohen, T. Lyche, and L. Schumaker. Algorithms for Degree Raising for Splines. ACM Transactions on Graphics, Vol 4, No 3, pp. 171-181, Jul. 1986.

[5] G. Farin. Curves and Surfaces for Computer Aided Geometric Design Academic Press, Inc. Second Edition 1990.

[6] R. T. Farouki and V. T. Rajan. Algorithms For Polynomials In Bernstein Form. Computer Aided Geometric Design 5, pp 1-26, 1988.

[7] J. D. Foley and A. Van Dam. Computer Graphics, Principles and Practice, Second Edition. Addison-Wesley Systems Programming Series, Jul. 1990.

[8] G. Elber and E. Cohen. Hidden Curve Removal for Free Form Surfaces. Computer Graphics, Vol. 24, Num. 4, pp. 95-104, Siggraph Aug. 1990.

[9] G. Elber and E. Cohen. Second Order Surface Analysis Using Hybrid Symbolic and Numeric Operators Submitted for publication.

[10] G. Elber and E. Cohen. Hybrid Symbolic and Numeric Operators as Tools for Analysis of Freeform Surfaces. Technical Report UUCS-92-023, University of Utah.

[11] G. Elber and E. Cohen. Model Fabrication using Surface Layout Projection. To be published.

[12] G. Elber. Free Form Surface Analysis using a Hybrid of Symbolic and Numeric Computation. Ph.D. thesis, University of Utah, Computer Science Department, 1992.

[13] J. Hoschek. Approximate Conversion of Spline Curves. Computer Aided Geometric Design 4, pp 59-66, 1987.

[14] J. Hoschek, F. J. Schneider, and P. Wassum. Optimal approximate conversion of spline surfaces. Computer Aided Geometric Design 6, pp 293-306, 1989.

[15] B. V. Herzen and A. H. Barr. Accurate Triangulations of Deformed, Intersecting Surfaces. Computer Graphics, Vol. 21, Num. 4, pp. 103-110, Siggraph Jul. 1987.

[16] R. V. Klassen. Integer Forward Differencing of Cubic Polynomials: Analysis and Algorithms. ACM Transaction on Graphics, Vol. 10, Num. 2, pp 152-181, Apr. 1991.

[17] J. M. Lane and R. F. Riesenfeld. Bounds on a Polynomial BIT 21 (1981), 112-117.

[18] S. Lien, M. Shantz, and V. Pratt. Adaptive Forward Differencing for Rendering Curves and Surfaces. Computer Graphics, Vol. 21, Num. 4, pp. 111-118, Siggraph Jul. 1987.

[19] T. McCollough. Support for Trimmed Surfaces. M.S. thesis, University of Utah, Computer Science Department, 1988.

[20] K. Morken. Some Identities for Products and Degree Raising of Splines. To appear in the journal of Constructive Approximation.

[21] D. R. Peachey. Solid texturing of Complex Surfaces. Computer Graphics, Vol. 19, Num. 3, pp. 333-342, Siggraph Jul. 1985.

[22] K. Perlin. An Image Synthesizer. Computer Graphics, Vol. 19, Num. 3, pp. 333-342, Siggraph Jul. 1985.

[23] A. Rappoport. Rendering Curves and Surfaces with Hybrid Subdivision and Forward Differencing. ACM Transaction on Graphics, Vol. 10, Num. 4, pp.323-341, Oct. 1991.

[24] A. Rockwood, K. Heaton, and T. Davis. Real-Time Rendering of Trimmed Surfaces. Computer Graphics, Vol. 23, Num. 3, Siggraph Jul. 1989.

[25] M. Shantz and S. L. Lien. Shading Bicubic Patches. Computer Graphics, Vol. 21, Num. 4, pp. 189-196, Siggraph Jul. 1987.

[26] M. Shantz and S. Chang. Rendering Trimmed NURBS with Adaptive Forward Differencing. Computer Graphics, Vol. 22, Num. 4, pp. 189-198, Siggraph Aug. 1988.