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

June 27, 1996

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, 2, 15, 17, 22, 23, 25, 26] presented methods that render surfaces as sequences of isocurves.

These methods each have deficiencies in their ability to guarantee a complete coverage of the rendered surface, in their ability to prevent processing the same pixel multiple times, or in their ability to produce an optimal surface coverage under some prescribed norm.

In this paper, an algorithm is introduced that alleviates the difficulties in all of these areas. This algorithm can be combined with a fast curve rendering method to make surface rendering without polygonal approximation practical.

Keywords: scan conversion, NURBs, surface coverage, direct freeform surface rendering

1 Introduction

Most surface rendering systems render a set of polygons that 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 frequently aliasing can occur. Intensity (Gouraud) and normal (Phong) interpolation schemes [5] were developed to overcome the visual effects caused by C^1 discontinuities across boundaries between polygons. The faceted appearance of the boundary and silhouette edges can be alleviated by increasing, as necessary, the number of polygons along the silhouettes, making each one smaller, globally or adaptively [14]. It is an even more difficult problem to subdivide trimmed surfaces into polygons for rendering [24, 27]. On the other hand, rendering the surface as a set of isocurves is appealing since the representation of each curve is exact, eliminating some of the need for the anti-aliasing techniques developed for rendering of polygonal approximations. Furthermore, rendering surfaces as isocurves reduces the algorithmic complexity necessary to support trimmed surfaces, as will be demonstrated, as well as reducing the complexity of the algorithms involved in texture mapping computations. Several methods have been published in recent years to render

*This work was supported in part by DARPA (N00014-92-J-4113) and the NSF and DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219). 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.

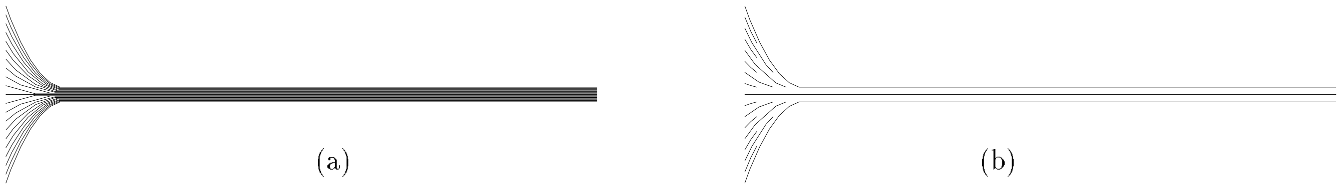

Figure 1: Scan converting complete isocurves can lead to an unbounded amount of redundancy when surfaces are scan converted using isoparametric curves (a). Adaptive extraction of isocurves can introduce much more optimal results (b).

surfaces using isocurves [1, 2, 15, 17, 22, 23, 25, 26]. However, many of the previous techniques assumed the existence of the set of isocurves that covers the surface and do not automatically extract an optimal or almost optimal set of isocurves from a surface S. Hence, rendering S with these techniques cannot be guaranteed to include every pixel representing the surface in image space, nor can it guarantee optimality under some prescribed norm.

In [1], the special case of direct rendering of sweep surfaces is considered. To partially address this problem for arbitrary surfaces, a heuristic subdivision based approach was used in [22]. Since this method subdivides the surface each time the spacing of the isocurves varies more than a specified tolerance, this approach can lead to processing a large number of small patches. The adaptive forward differencing algorithm in [25, 26] has a fixed initialization cost per isocurve so rendering a large number of small patches means drawing an even larger number of isocurves, which would greatly increase the total rendering cost of the complete surface. Isocurves at equally spaced intervals, $u = n\delta u$, are used in [25]. The use of fixed spacing intervals for isocurves can result in pixels being missed, leaving holes in the image of the surface. Alternatively, the rendering algorithm might end up processing identical pixels numerous times, reducing the rendering efficiency. We refer to this last phenomena as redundancy in the coverage, as is demonstrated in the middle of Figure 2 (a) and in the center of Figure 3 (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 spans the entire u domain of the surface. Therefore, the redundancy demonstrated in Figures 2 (a) and 3 (a) would continue to reduce optimality in [26]. A similar approach to that of [25] is taken in [23], which applies the mean value theorem. A bound on the Euclidean distance resulting from a small h step in the parametric space of a Bézier curve is computed as $\|C(u+h)-C(u)\| \leq n\ h\ max\|P_{i+1}-P_i\|$, where n is the degree of $C(u)$ and $\{P_i\}$ are its control points. In [25] and [23] complete isoparametric curves are scan converted, were a complete isoparametric curve is one that spans the entire parametric domain of the surface. There is no way to establish an upper bound on the number of times the same pixel is actually drawn and hence on the amount of redundancy. See for example Figure 1.

This paper presents an algorithm that colors all required pixels, yet the algorithm provides a bound on the amount of pixel redundancy in the established coverage. The new algorithm *adaptively* extracts partial isocurves and covers the entire surface in an almost optimal way. The polygon primitive is replaced by a finite thickness isocurve.

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 approximately

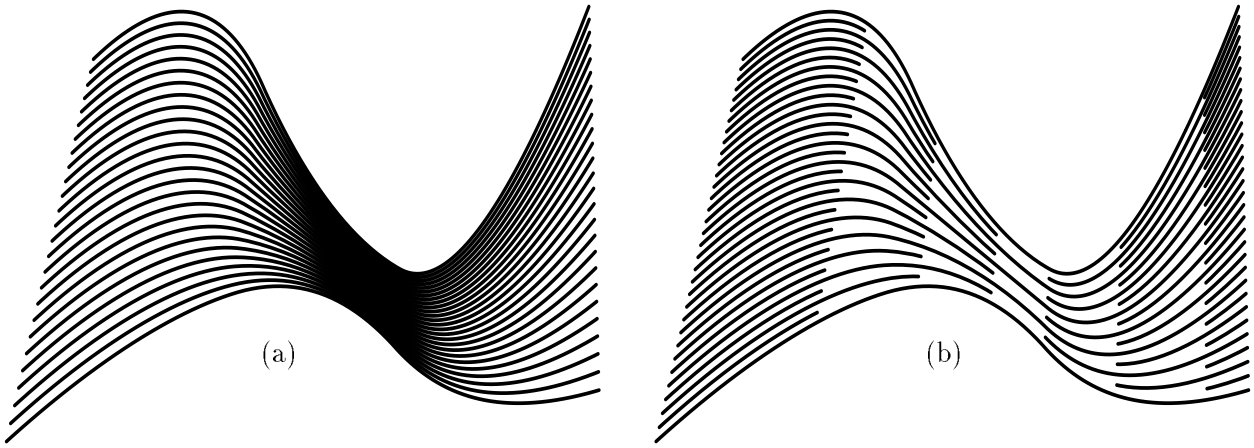

Figure 2: Isocurves are obviously not an optimal solution as a valid coverage for this surface (a). Adaptive isocurves are more optimal and their coverage is valid as well (b).

half of the image pixel size. All pixels representing S in the image are then guaranteed to be covered by at least one isocurve. In the ensuing discussion S will be assumed to be represented in the viewing space. A surface in viewing space has its x and y coordinates aligned with the image plane coordinates i_x, and i_y. That is, $i_x = x$ and $i_y = y$. However, the z coordinate of the surface is still accessible. The viewing space automatically accounts for distant and small surfaces that require less effort to render, since the perspective transformation has already been applied. In many cases, it is sufficient to compute the iso-distance using only the x and y surface components, since coverage of the image plane is the concern. However, ignoring z may result in missed pixels when the surface is partially hidden. We will discuss this issue further later.

Definition 2 *A coverage for a given surface is considered* optimal *if it is valid and the accumulated pixel drawing cost function is minimal over all valid coverages.*

A *pixel drawing cost function* should weigh the initialization cost of drawing a curve amortized over the length of the curve plus the actual cost of drawing each pixel.

If one could compute the parametric spacing required for a valid coverage of 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 2 (a) and the center of the surface in Figure 3 (a). Denote the isoparametric curves of surface $S(u, v)$ by $C_i(u)$. Because $\frac{\partial S}{\partial v}$ is not a constant value across the parametric domain of the surface, local dynamic change of the parameter spacing is required as seen in Figures 2 (b) and 3 (b), to improve the optimality of the coverage.

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 $[u_3^s, u_3^e] \cap \mathcal{U} \ne \emptyset$.*

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 $\|C_1(u) - C_2(u)\|_2$.*

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.

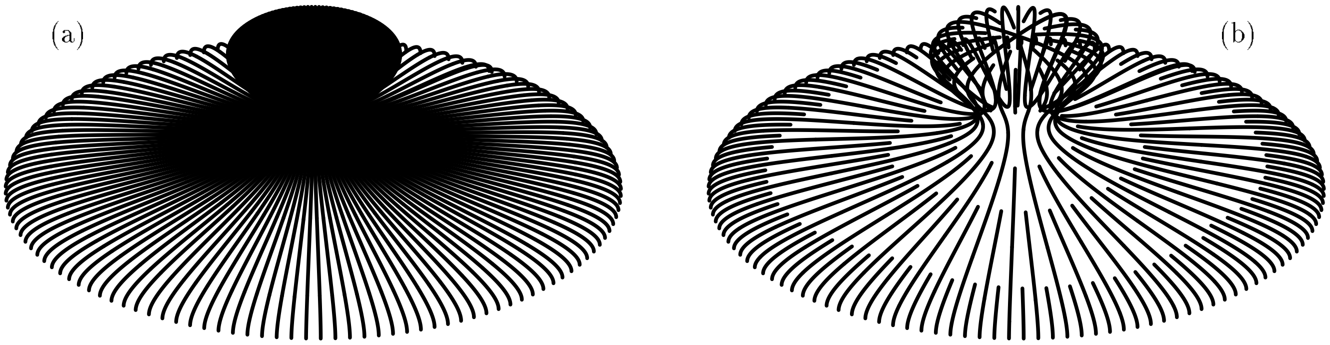

Figure 3: Utah teapot lid. Constant parameter spacing causes redundancy in coverage (a) mostly eliminated by adaptive extraction of isocurves (b). Both provide a valid coverage with respect to same δ.

2 Background

Lemma 1 *Let $C_1(u) = S(u, v_1)$ and $C_2(u) = S(u, v_2)$ and let $\hat{R}(u, v) = C_1(u) * v + C_2(u) * (1 - v)$, $v \in [0, 1]$. Then, 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: Let p be an arbitrary point in $\hat{R}$, $p = \hat{R}(u^*, v^*)$. p is on the line connecting $C_1(u^*)$ 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, $\hat{R} = \hat{R}(u, v)$, 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 [9], curvature analysis of R [7], or alternatively analysis of the speed variance of R [8]. 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 one pixel wide, usually a more accurate approximation than the polygons used for rendering. Hence, unless subpixel results are required (in which case δ can be made smaller), it is unnecessary to further bound 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 minimal distance between the curves. One might expect the penalty for this assumption to be quite high. In practice, it was found that most surfaces are well behaved and the Hausdorff distance between two adjacent isocurves is not much smaller than $\Delta_{12}(u)$. Section 4 demonstrates many common cases where highly non-isometric mappings are infrequent and isolated in small regions of the domain. Even at locations with highly non-isometric mappings the proposed algorithm provides competitive results compared to previous methods that employ 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:

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

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 [3, 4, 10, 19, 16], for finding the form of the sum and product in the Bézier and NURBs representation as well as its zero set. 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:

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

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 $(\Delta_{12}^2(u) - \delta^2)$. In practice, $(\Delta_{12}^2(u) - \delta^2)$ should only be bounded from above. That is, the exact zero set of $(\Delta_{12}^2(u) - \delta^2)$ is unnecessary as long as $(\Delta_{12}^2(u) - \delta^2) < 0$ for all adjacent isocurves forming the valid coverage. 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 computed with this algorithmic process forms a valid coverage. The variation of the speed in the v direction (ruled direction of $\hat{R}$), $\frac{\partial S}{\partial v}$, between two adjacent isocurves, $C_1(u)$ and $C_2(u)$, decreases as δ becomes smaller and the coverage becomes more dense. Hence, an isocurve introduced in the middle of the parametric domain of R, will be almost half way between $C_1(u)$ and $C_2(u)$.

In the limit, as δ approaches zero, and since an isocurve is introduced between $C_1(t)$ and $C_2(t)$ only if

Algorithm 1

```
Input:
    $S(u,v)$, input surface.
    $\delta$, maximum distance between isocurves.

Output:
    S, the set of constant $v$ isocurves of $S(u,v)$ adjacent within $\delta$, covering $S$.

Algorithm:
    adapIsoCrvs( S, $\delta$ )
    begin
        $C_1(u)$, $C_2(u)$ $\Leftarrow$ isocurves of $S$ in $u$ direction at $VMin$, $VMax$.
        return
            { $C_1(u)$ } $\cup$
            adapIsoCrvsAux( $S$, $\delta$, $u$, $VMin$, $VMax$, $C_1(u)$, $C_2(u)$ ) $\cup$
            { $C_2(u)$ }.
        end
    end

    adapIsoCrvsAux( $S$, $\delta$, $VMin$, $VMax$, $C_1(u)$, $C_2(u)$ )
    begin
        $UMax$, $UMin$ $\Leftarrow$ $C_1(u)$, $C_2(u)$ common $u$ domain.
(1)     $\Delta_{12}^2(u)$ $\Leftarrow$ squared iso-distance between $C_1(u)$ and $C_2(u)$.
        $\mathcal{Z}$ $\Leftarrow$ zero set of $(\Delta_{12}^2(u) - \delta^2)$.
        if $\mathcal{Z}$ empty then
            $R$ $\Leftarrow$ $S$ subsurface between $C_1(u)$ and $C_2(u)$.
            $\hat{R}$ $\Leftarrow$ $C_1(u) * v + C_2(u) * (1-v)$, $v \in (0,1)$.
(2)         if $\Delta_{12}^2((UMax + UMin)/2) < \delta^2$ and $(\hat{R} \rightsquigarrow R)$ valid then
(3)             return $\phi$.
            else
                $VMid$ $\Leftarrow$ $(VMin + VMax)/2$.
                $C_{12}(u)$ $\Leftarrow$ isocurve of $S$ at $VMid$ from $UMin$ to $UMax$.
                return
(4)                 adapIsoCrvsAux( $S$, $\delta$, $VMin$, $VMid$, $C_1(u)$, $C_{12}(u)$ ) $\cup$
                    { $C_{12}(u)$ } $\cup$
(5)                 adapIsoCrvsAux( $S$, $\delta$, $VMid$, $VMax$, $C_{12}(u)$, $C_2(u)$ ).
            end
        else
(6)         Subdivide $C_1(u)$, $C_2(u)$ at all $u^i \in \mathcal{Z}$ into $\{C_1^i(u)$, $C_2^i(u)\}$ pairs.
(7)         return $\bigcup_i$ adapIsoCrvsAux( $S$, $\delta$, $VMin$, $VMax$, $C_1^i(u)$, $C_2^i(u)$ ).
        end
    end
```

$\Delta_{12} > \delta$ no two isocurves will have iso-distance less than $\frac{\delta}{2}$. In practice, δ is in the order of a pixel size and is much smaller than the surface size. The vast majority of the isocurves in the computed coverages were found to be within $\Delta_{12} > \frac{\delta}{2}$ of their neighbors. Therefore, the redundancy in the computed coverage is bounded.

There are some subtleties that have not yet been considered. 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 the prevention of such cases is to insure the surface is silhouette free from the rendering direction (See [6, 11] for silhouette detection). A surface is silhouette free if its normal is never perpendicular to the viewing direction. An alternative may be to use a heuristic that 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 [10], 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 [2, 15, 17, 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 [10, 12, 13].

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 xy image plane is silhouette free or locally one to one, then only x and y need to be consulted. By 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 4 (a)), while distant apart in the viewing or object space (as seen in Figure 4 (b)), violating the one to one mapping requirement. For example, a planar surface, almost perpendicular to the image plane should be drawn with many fewer isocurves if only x and y coefficients are considered. If a surface has silhouette curves in the image plane, using only x and y in the computation of $\Delta_{12}^2(u)$ for two isocurves could result in invalid coverage, as would be the case in Figure 4, if $\Delta_{12}^2(u)$ were computed without using z. Therefore, one can easily determine if the iso-distance should be computed using z or without it by determining if the surface is silhouette free.

4 Results

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

Figure 5 presents the well known Utah teapot model and a chess set rendered using the adaptive isocurve extraction algorithm.

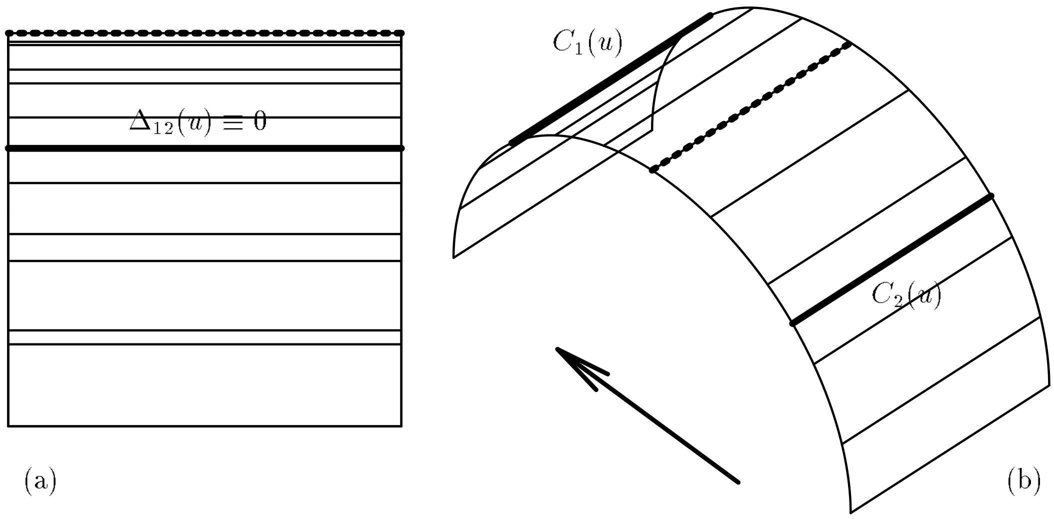

Figure 4: If S has silhouettes (dotted) in image plane, and only x and y are consulted in $\Delta_{12}(u)$ computation, $\Delta_{12}(u)$ may be found by Algorithm 1 to be wrongly zero, terminating prematurely.

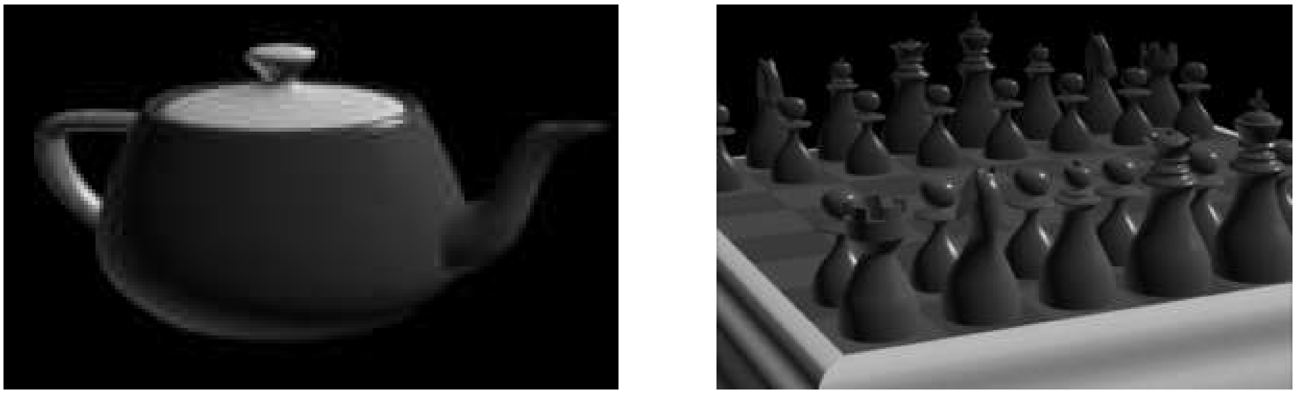

Figure 5: Utah teapot and a chess set adaptive isocurves rendered images.

Most techniques developed for enhancing image rendering quality can be applied to curve rendering. For example, Figure 6 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 5 and 6 show a complex chess scene rendered using this new algorithm.

Figures 7 and 8 demonstrates the use of solid texture to define a virtual planet and a camouflaged plane, using techniques presented in [20, 21], rendered using this isocurve based renderer.

The use of polygons for displaying sculptured surfaces requires the generation of a large number of small polygons from compact surface forms. This intermediate polygonal data set is used for the sole purpose of displaying the surface. In contrast, the extraction of isoparametric curves from a freeform surface is a simple task. It is hoped that the intermediate polygonal representation could be eliminated by providing dedicated real time isocurve extraction capabilities, perhaps in hardware. One could extract the isoparametric curves only at parameter values of the form $\frac{i}{2^n}$, $0 \leq i \leq 2^n$ (assuming a v-domain from zero to one). Then, the coefficients that blend the control mesh of the surface, and are used to extract

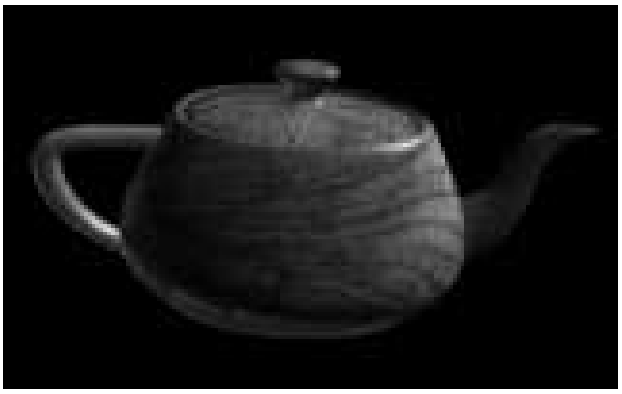


Figure 6: Wood texture mapped on teapot model. Marble texture mapped chess set scene.

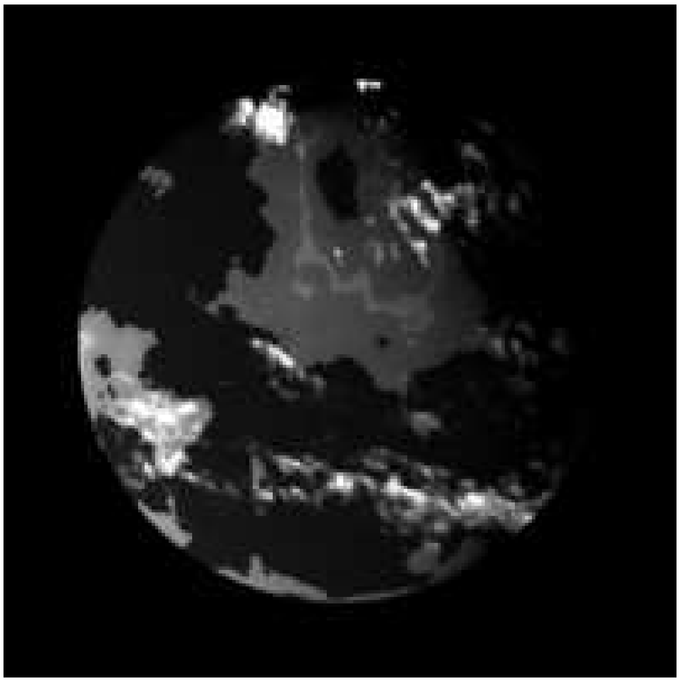

Figure 7: A virtual planet texture image using isocurves rendering.

the $2^n + 1$ isoparametric curves, could be precomputed into a table. Therefore, isoparametric curves could be extracted on the fly and scan-converted in real time. Given a surface S, approximating S into a set of polygons for rendering is not only difficult but is also a time consuming process. Moreover, further impediments to rapid rendering of trimmed surfaces are posed because of the necessity to clip the polygons against the trimming curves [18, 24, 27]. However, since the algorithm presented herein produces only isocurves, the clipping process is significantly simplified. The isoparametric curve is a horizontal or a vertical line in the parametric space and hence the clipping problem is reduced to finding the intersection between an axis parallel line and a planar trimming curve.

A model consisting of trimmed surfaces is rendered in Figure 9. A piston bridge model from a Diesel engine consisting of 27 trimmed NURBs surfaces was rendered while using wood texture mapping.

One might also consider rendering the surface adaptively 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 10 shows six steps

Figure 8: A camouflage texture using isocurves rendering of an F16 model.

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 $2N - 2$. Each iso-distance curve computation of a polynomial curve exploits three scalar curve subtractions, three scalar curve products and two scalar curve additions (equation (2)), using the Bézier or the B-spline representation. Hence, an $O(N)$ output sensitive complexity is expected. 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, while curves were rendered as piecewise linear polylines in our implementation, the computed highlights and shading have more realistic appearance, as is demonstrated by the images throughout this paper, due to using accurate surface normals for each individual pixel that is delivered by this algorithm. This, in contrast to the Gouraud and Phong methods that interpolate colors and/or normals within polygons. Usage of (adaptive) forward differencing might improve the overall algorithm performance, and further remove aliasing introduced by the employed piecewise linear approximation of the isocurves.

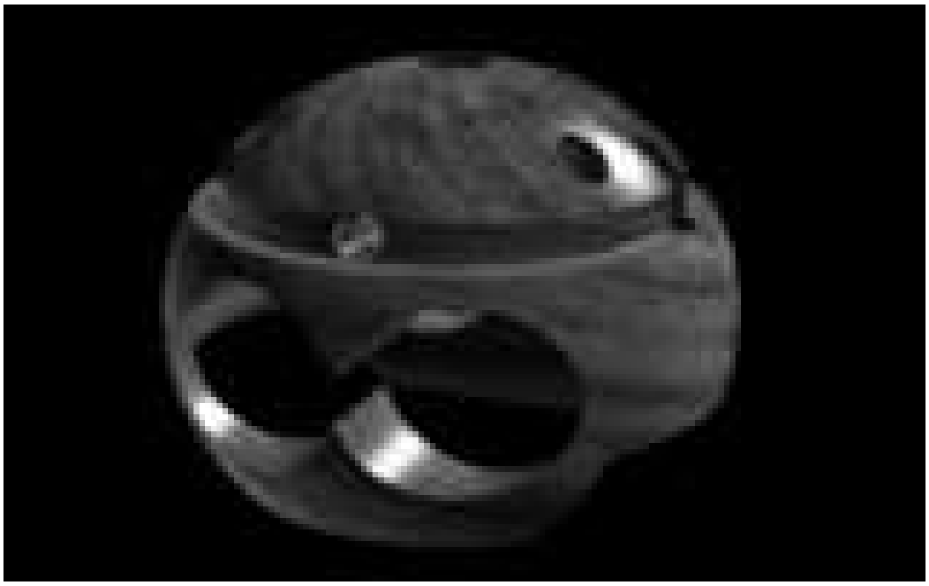

Figure 9: Trimmed NURBs model using adap. isocurves.

5 Conclusion

A new algorithm is presented that automatically computes an adaptive valid coverage of a surface using isocurves. The existing capability 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] W. F. Bronsvoort A Surface-Scanning Algorithm for Displaying Generalized Cylinders. The Visual Computer, Vol 8, pp 162-170, 1992.

[2] S. Chang, M. Shantz and R. Rocchetti. Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing. Computer Graphics, Vol. 23, No. 3, pp. 157-166, Siggraph Jul. 1989.

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

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

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

[6] G. Elber and E. Cohen. Hidden Curve Removal for Free Form Surfaces. Computer Graphics, Vol. 24, No. 4, pp. 95-104, Siggraph Aug. 1990.

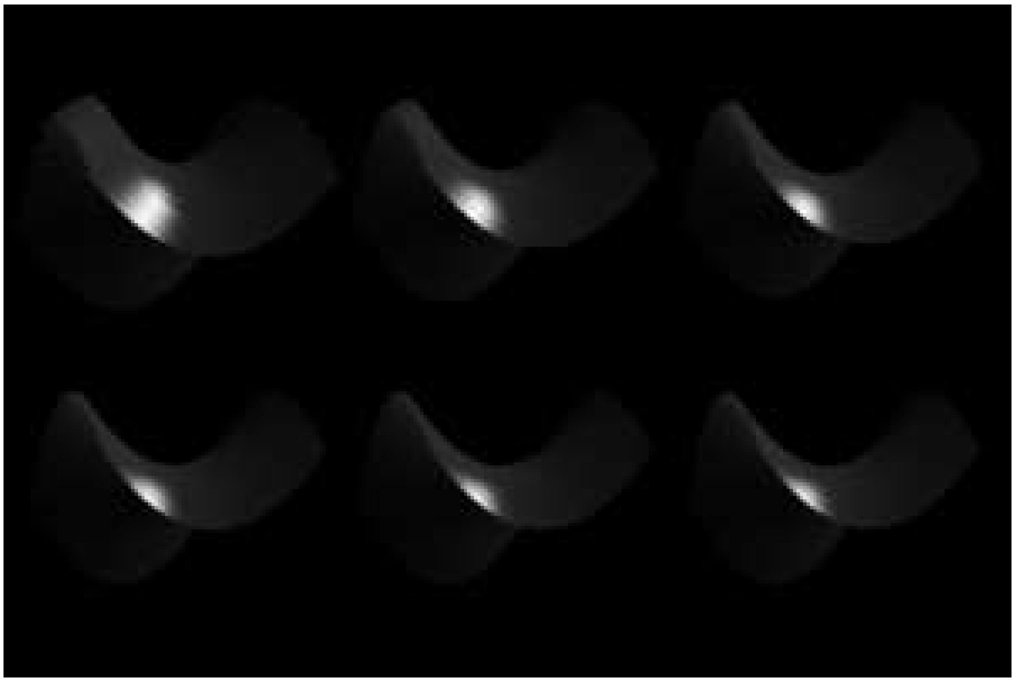

Figure 10: Six steps in coarse to fine rendering using adaptive isocurves.

[7] G. Elber and E. Cohen. Second Order Surface Analysis Using Hybrid Symbolic and Numeric Operators. Transaction on Graphics, Vol. 12, No. 2, pp 160-178, April 1993.

[8] G. Elber and E. Cohen. Hybrid Symbolic and Numeric Operators as Tools for Analysis of Freeform Surfaces. Modeling in Computer Graphics, B. Falcidieno and T. L. Kunii (Eds.), Working Conference on Geometric Modeling in Computer Graphics (IFIP TC 5/WG 5.10), pp 275-286, Genova, June-July 1993. Also technical report UUCS-92-023, University of Utah.

[9] G. Elber. Model Fabrication using Surface Layout Projection. CAD, Vol. 27, No. 4, pp 283-291, April 1995.

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

[11] G. Heflin and G.Elber. Shadow Volume Generation from Free Form Surfaces. Communicating with Virtual Worlds, Nadia Magnenat Thalmann and Daniel Thalmann (Eds.), Computer Graphics International 1993 (CGI 93), pp 115-126, Lausanne, Switzerland, June 1993.

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

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

[14] B. V. Herzen and A. H. Barr. Accurate Triangulations of Deformed, Intersecting Surfaces. Computer Graphics, Vol. 21, No. 4, pp. 103-110, Siggraph Jul. 1987.

[15] R. V. Klassen. Integer Forward Differencing of Cubic Polynomials: Analysis and Algorithms. ACM Transaction on Graphics, Vol. 10, No. 2, pp 152-181, Apr. 1991.

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

[17] S. Lien, M. Shantz, and V. Pratt. Adaptive Forward Differencing for Rendering Curves and Surfaces. Computer Graphics, Vol. 21, No. 4, pp. 111-118, Siggraph Jul. 1987.

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

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

[20] D. R. Peachey. Solid texturing of Complex Surfaces. Computer Graphics, Vol. 19, No. 3, pp. 279-286, Siggraph Jul. 1985.

[21] K. Perlin. An Image Synthesizer. Computer Graphics, Vol. 19, No. 3, pp. 287-296, Siggraph Jul. 1985.

[22] A. Rappoport. Rendering Curves and Surfaces with Hybrid Subdivision and Forward Differencing. ACM Transaction on Graphics, Vol. 10, No. 4, pp. 323-341, Oct. 1991.

[23] A. Rockwood. A Generalized Scanning Technique for Display of Parametrically Defined Surfaces. IEEE Computer Graphics and Applications, Vol. 7, No. 8, pp 15-26, Aug. 1987.

[24] A. Rockwood, K. Heaton, and T. Davis. Real-Time Rendering of Trimmed Surfaces. Computer Graphics, Vol. 23, No. 3, pp. 107-117, Siggraph Jul. 1989.

[25] M. Shantz and S. L. Lien. Shading Bicubic Patches. Computer Graphics, Vol. 21, No. 4, pp. 189-196, Siggraph Jul. 1987.

[26] M. Shantz and S. Chang. Rendering Trimmed NURBS with Adaptive Forward Differencing. Computer Graphics, Vol. 22, No. 4, pp. 189-198, Siggraph Aug. 1988.

[27] X. Sheng and B. E. Hirsh. Triangulation of Trimmed Surfaces in Parametric Space. Computer Aided Design, Vol. 24, No. 8, pp 437-444, August 1992.