Hidden Curve Removal for Free Form Surfaces*

Gershon Elber and Elaine Cohen
UUCS-89-019
Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA
May 1989

Abstract

This paper describes a hidden curve algorithm specifically designed for sculptured surfaces. A technique is described to extract the visible curves for a given scene without the need to approximate the surface by polygons. This algorithm produces higher quality results than polygon based algorithms, as most of the output set has an exact representation. Surface coherence is used to speed up the process. Although designed for sculptured surfaces, this algorithm is also suitable for polygonal data.

CR Categories and Subject Descriptors: I.3.5 **[Computer Graphics]**: Computational Geometry and Object Modeling. I.3.7 **[Computer Graphics]**: Three-Dimensional Graphics and Realism.

Additional Key Words and Phrases: hidden curve removal, curve-curve intersection, visibility propagation.

1 Introduction

Hidden line removal is one of the earliest computer graphics problems; yet new algorithms appear every year [16,1,8,21,4,15,17,18,20,22,23,25,27]. Usually they are developed for polygonal data, so sculptured surfaces must be preprocessed and approximated as large collections of polygons. The result displays the polygonized models accurately, but original model information is lost [4,18,20] (see figure 1).

In [16,1,8,21,27] the idea of *quantitative invisibility* and the use of critical points was developed. Critical points are intersection points between projected polygon primitives. The

*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 author and do not necessarily reflect the views of the sponsoring agencies.

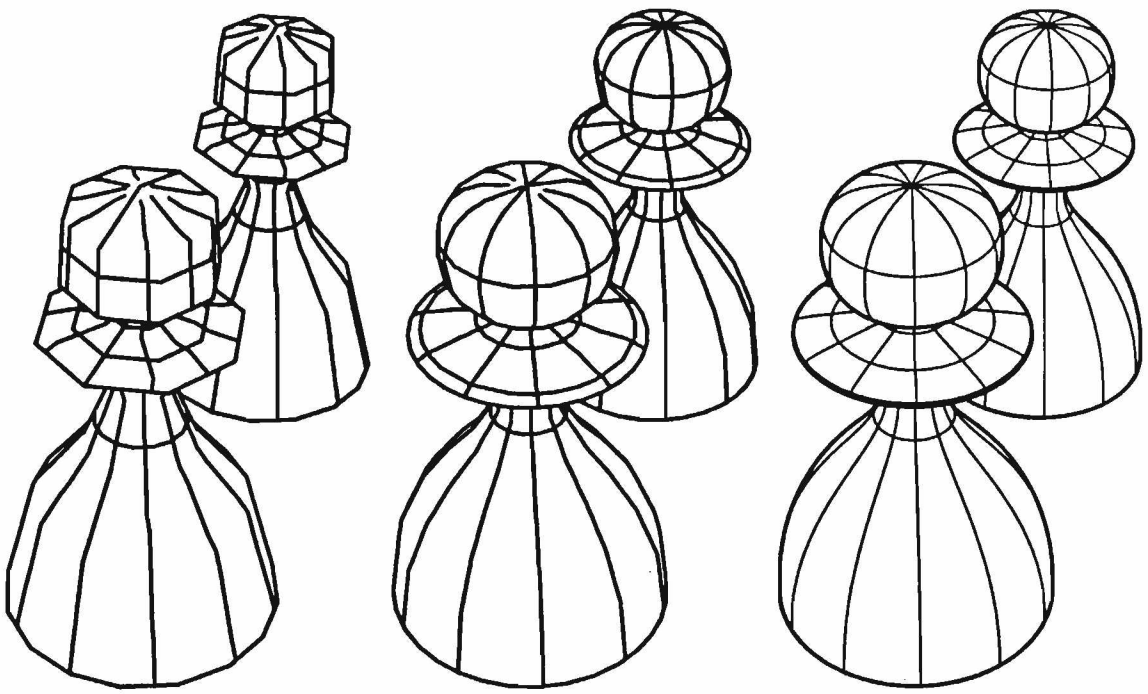

Figure 1: Left: 1930 polygons, middle: 4215 polygons, right: 2 NURBS surfaces (56 patches).

technique was extended to bi-quadratic patches in [12]. The primitives are subdivided at each critical point which guarantees that the interior of each segment has homogeneous visibility. A segment's visibility is then tested by firing a ray from the eye to an interior point of the segment.

While the same basic approach is used here, it is extended to apply to arbitrary nonuniform rational B-spline (NURBS) surfaces (the primitives). By not using polygonal approximations, the algorithm eliminates the vast amount of data resulting from the approximation of surfaces by polygons. The algorithm presented here has several stages (including extracting curves of interest, splitting at critical points, and visibility testing). Surface coherence is used extensively to reduce the number of ray tests one needs to perform to detect curve visibility, in a similar way to that for polygons[10,11].

The trade-off is a reduced number of primitives in exchange for a higher complexity of the operations between them. Driving the output directly from the surface results in high quality images (Figure 1). Only silhouette curves need to be approximated, but they can be approximated at a much higher resolution than with pre-polygonized surfaces. Figure 1 compares the two, after adaptive subdivision has been used to obtain the polygonal approximations.

Section 2 defines the required elements for presenting the algorithm and also introduces the basic hidden curve algorithm. Section 3 explains how the curves of interest are extracted from the surface. Usually four types of curves are useful: the surface boundary curves, curves along C^1 discontinuities in the surface (if any), iso-parametric curves, and silhouette curves. Section 4 addresses the 2D curve-curve intersection problems specific to this algorithm. Sec-

tion 5 presents issues in visibility testing arising from a surface-ray intersection algorithm. Computation of surface-ray intersections is relatively expensive, especially when polygonal approximations to the surfaces have not been used, so methods to speed the computation are described. Section 6 discusses extensions to this algorithm for trimmed surfaces.

All the images/comparison examples shown in this paper, were generated using nonuniform, rational B-spline (NURB) surfaces as implemented in the Alpha_1 system.

2 The Basic Algorithm

Let the *view orientation* be normalized so that the view point V is on the positive z-axis at ∞ and the image is projected onto the plane $z = 0$, which will be referred to as the *projection plane* or the *screen.* All other views can easily be preprocessed into this view.

Definition 2.1

1. *A non-self-intersecting surface is called a* simple surface.
2. *A non-self-intersecting curve is called a* simple curve.
3. *Let P be the map to the projection plane, $P(x, y, z) = (x, y, 0)$. The subscript P will usually denote the mapped entity, e.g., $C_P = P(C)$.*

We consider only simple surfaces whose projected curves are simple curves. Also, intersection between surfaces are restricted to lie along boundaries. If not, the surfaces should be intersected using the appropriate boolean operations, and trimmed surfaces should be created. Figure 16 has surfaces which interpenetrate, and so boundaries between them are unknown and visibility cannot be always correctly determined.

Suppose surface $S = S(u, v) \in C^1$, then $N(u, v) = \frac{\partial \vec{S}}{\partial u} \times \frac{\partial \vec{S}}{\partial v}$ is normal to the surface S. Unless otherwise specified, all surfaces with which we deal are assumed to be C^1 continuous, and their parameterizations are assumed to be such that $N(u, v) \neq 0$. If p is a point on the surface S, then it is the image of a point in parameter space, so we write $p = S(u^p, v^p)$.

Definition 2.2

1. *A* silhouette point *of a surface is a point p on the surface whose normal has a zero z-component, i.e. $N_z(u^p, v^p) = 0$.*
2. *A* silhouette curve *of surface S is an ordered set of silhouette points $Q \subset S$ forming a continuous curve such that $\forall\ p \in Q$, and $\forall\ \epsilon > 0$, $\exists\ q \in S$ such that $N_z(u^q, v^q) \neq 0$, and $\|p - q\| < \epsilon$.*
3. *If $\exists\ D \subset S$ that is the image of an open disk such that $\forall\ d \in D$, $N_z(u^d, v^d) = 0$, we say that S has a* silhouette surface *and the above definition of the silhouette curve is simply its boundary.*

We will not consider surfaces with silhouette sub-surfaces in them. Surfaces that do contain silhouette surfaces may be preprocessed by trimming out the silhouette sub-surfaces (see Section 6).

Lemma 2.1 *A surface S can be decomposed into silhouette curves and disjoint regions whose boundaries are surface boundaries and/or silhouette curves.*

Proof: Consider a point $p \in S$, $N_z(u^p, v^p) \neq 0$, and the set of points which can be reached from p by continuous paths in the surface that do not cross any silhouette curves. Let that set be R^p, and let R^q be a similar set for point $q \in S, q \notin R^p$, if such a point q exists. Clearly $R^p \cap R^q = \emptyset$, for if there exists point $r \in R^p \cap R^q$, then there exists a continuous path from p to r and from r to q, and hence from p to q. But this contradicts the way we picked q. Call the collection of path connected regions $\mathcal{R}$.

For our proofs we require that the projection operation be bijective between each $R \in \mathcal{R}$ and its projected image.

Surprisingly not all surfaces are well behaved in this, although they are fairly rare. Such a surface, as seen in figure 2, must be preprocessed and subdivided. Such surface can be detected by noticing its boundary is self intersection:

Lemma 2.2 *Denote by $Fr(R)$ the boundary of $R \in \mathcal{R}$, if R intersects with some ray $\Re$ more than once, then $P(Fr(R))$ is self intersecting, i.e. not simple.*

Proof: In order to show that the $P(Fr(R))$ is self intersecting, it is enough to show that there exists a point $p^b \in Fr(R)$ and a point $p^i \in Int(R)$ such that $P(p^b) = P(p^i)$. If $\Re$ intersects R in at least two points, called p^l and p^m, find a continuous path ρ in R from p^l to a boundary point p^b in $Fr(R)$. if $P(\rho)$ intersects with $P(Fr(R))$, we are done. Otherwise, a ray through p^b must intersect R in another place, since the projected path $P(\rho)$ never crossed the boundary.

Lemma 2.3 *Let $R \in \mathcal{R}$ and $r_1, r_2 \in R$. Then, $N_z(u_{r_1}, v_{r_1}) N_z(u_{r_2}, v_{r_2}) > 0$, i.e. they have the same sign at both r_1 and r_2.*

Proof: Since there exists a continuous path from r_1 to r_2 totally in R, N_z along that path can have no zeros so N_z has the same sign at both r_1 and r_2.

We define visibility as invisibility count of zero where:

Definition 2.3 *The* invisibility count *of a point p is the number of intersections of the half-open ray $\Re$ from the view point V to the point p has with sculptured surfaces in the scene.*

We will concentrate our discussion on determining visibility only (i.e. where the count is zero), although one may use our algorithm to detect quantitative invisibility [6].

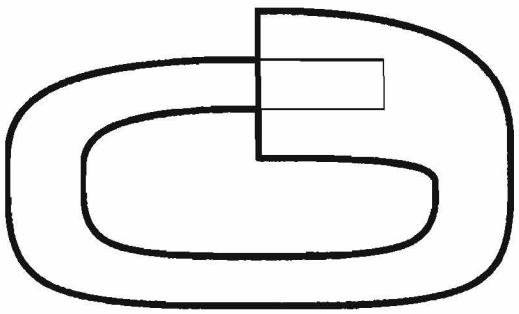

Figure 2: Single region surface (no silhouettes), which is not bijective.

Lemma 2.4 *Let $R \in \mathcal{R}$, and let $\Re$ be an half open ray to some point p.*

1. *The projection of R to $P(R)$ is injective, i.e. every point on R is mapped to a unique point in the $x - y$ plane.*
2. *R can modify the invisibility count of some point p by at most one.*
3. *The invisibility count of a point p will be increased by the region R iff $P(p) \in P(R)$, and $R \cap \Re \neq 0$.*

Proof:

1. Follows from the definition of R and lemma 2.2.
2. Since R must be injective, only one point of it can be mapped to some projection plane point. Specifically, given some point p, at most one point of R can be mapped onto $P(p)$.
3. In order for R to hide p, there must exist a point in R that is mapped onto $P(p)$. But p can be closer to the viewer and therefore R will hide p iff $P(p) \in P(R)$ and $R \cap \Re \neq 0$.

Two types of curves are dealt with:

Definition 2.4

1. *Surface boundary curves and surface silhouette curves are defined to be* active curves.
2. *The* framework ψ^S *of a surface S is the set of all its* active curves. *Set $\psi^S_P = P(\psi^S)$.*
3. *Any curve which is* not *active is defined to be a* passive curve. *For example, an isoparametric curve is passive.*

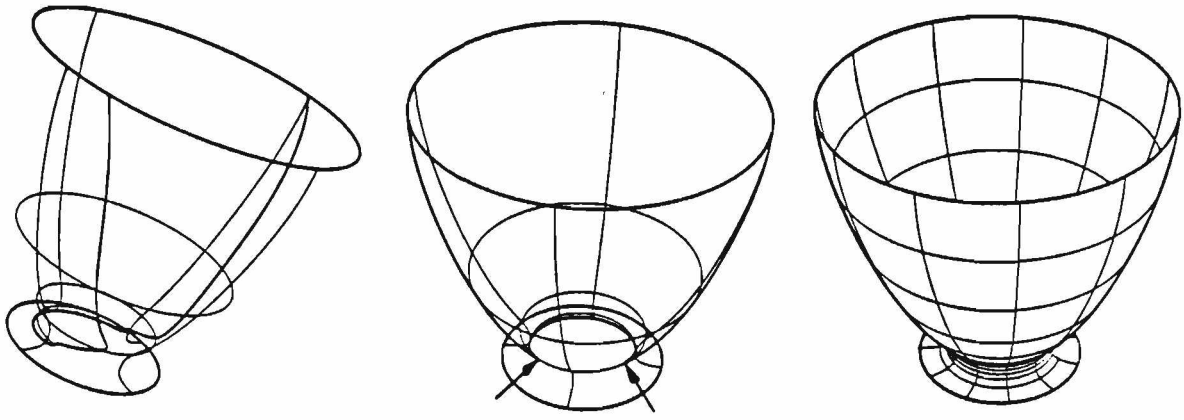

Figure 3: - silhouette might have cusps in its projection which the curve must be split at.

Clearly active curves are those curves at which visibility of all curves can change when crossing them.

Lemma 2.5 *Let C be a curve in the scene. If $C_P \bigcap (\cup_S \psi_P^S) = \phi$, then all points $p \in C$ have the same invisibility count.*

Proof: Since C_P does not intersect with any ψ_P^S, it is totally included in or excluded from any R_P for all regions. Relative to some region R, and based on lemma 2.4, C is invisible iff $C_P \in R_P$ and R is closer to the viewer, and visible otherwise. Since it is true for any R, C visibility must be homogeneous.

Corollary 2.1 *Given a curve C, let $\{C_P^i\}$ denote the collection of sub-curves of C_P resulting from subdividing C_P at each of its intersection points with $\cup_S \psi_P^S$, and let C^i be each pre-image, i.e., $C^i \subset C$. For each C^i, all points $p \in C^i$ have the same invisibility count, except possibly the two end points.*

Corollary 2.1 suggests an algorithmic way of testing visibility of curves in a scene. Obviously many projected curves will intersect with some projected frameworks, but the number of times they do is finite and usually small. For each of the subdivided curves, only one point in its open interval needs to be tested for its invisibility count. As noted in [2,11], silhouette curves can overlap; that is, a silhouette curve can be G^1 continuous and still have cusps in its projection (called a curtain fold [2]). Since the visibility of the silhouette curves may change at curtain folds, they must be split at these points. Fig 3 is an example of such case, in which the single silhouette curve has no G^1 discontinuity (left), while its projection has two cusps (middle - at arrows), where the silhouette must be broke at, so we could get the correct hidden curve result (right).

Using Corollary 2.1, we can define the basic algorithm:

1. Extract all curves of interest from the given surface(s) in the scene. This results in at least the framework of each surface, but can also include passive curves such as isoparametric curves.

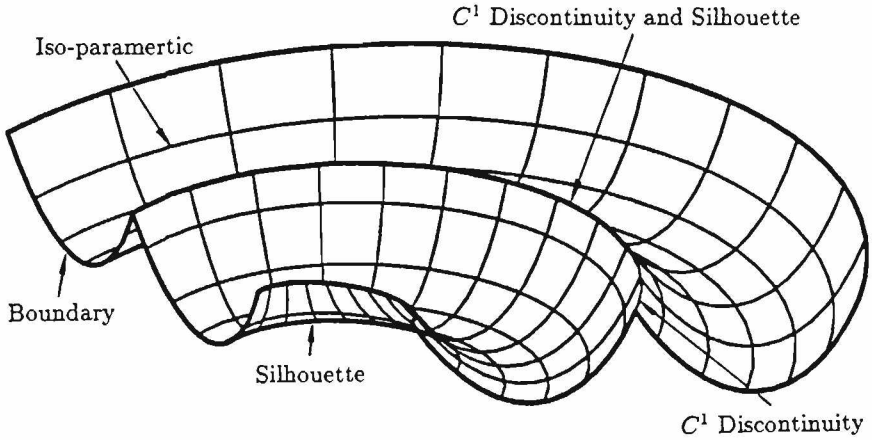

Figure 4: Four curve types in surface display.

2. Split each passive and active curve C at each point where its projection C_P intersects with any ψ_P^S and where ψ^S is closer to the viewer.

3. For each curve that results from the previous stage, fire a ray in the view direction through an arbitrary interior point and find the invisibility count of that point.

4. If the invisibility count of the point is zero, then curve is shown, otherwise it is hidden.

Based on Corollary 2.1, the evaluation of the invisibility count picks an arbitrary point p in the interior of curve C, and determines the visibility of all of C using the invisibility count of p. Curves are always split at tangent intersections, since they could potentially change visibility at those points. A technique is used in [16,1,8,21,27] of splitting a line into segments anytime it intersects with any projected polygon boundary to make the segment visibility homogeneous. In the next sections, we will describe the different stages for arbitrary sculptured surfaces. This algorithm, although conceptually simple, does require accurate computation in non robust numerical situations. We discuss our approaches to simplifying the operations and to reducing the number of times these complex operations must be performed.

3 Curve Extraction

The four types of curves which we display are boundary curves, iso-parametric curves, curves along C^1 discontinuities in the surface, and silhouette curves (see Figure 4). All but the last type are view independent.

Extraction of the first three types of curves is usually extremely simple since these curves are isoparametric. For example, Figure 5 shows the same pawn as in Figure 1. We assumed the surfaces are C^1 continuous, but if some are not, they must be subdivided at the appropriate isoparametric values. This creates a new boundary and two new surfaces on which the algorithm is then correct. This splitting stage is only added when tangent plane discontinuities are present.

The silhouette curves are not isoparametric curves usually (Figure 5), and in general,

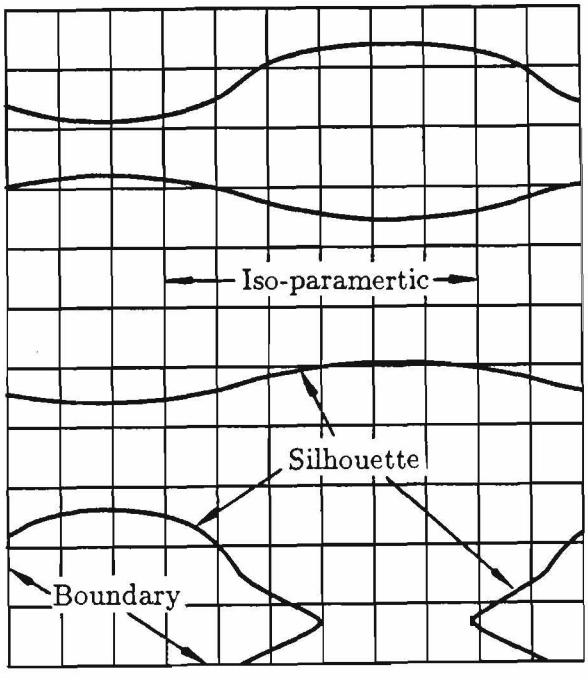

Figure 5: Pawn extracted curves, in $u - v$ space.

there is no computationally feasible way to represent them exactly. An approximation technique must be used in this case, and a piecewise linear curve is the simplest choice.

One method of extracting the silhouette curves of a surface is to subdivide the surface up to a given ϵ tolerance and fetch all the surface pieces with silhouettes on them. We seek methods using subdivision and spline properties to identify regions with silhouettes. To solve this problem we are interested only in the zero set of the z component of $N(u, v)$,

$$N_z(u,v) = \frac{\partial x(u,v)}{\partial u}\frac{\partial y(u,v)}{\partial v} - \frac{\partial y(u,v)}{\partial u}\frac{\partial x(u,v)}{\partial v} \tag{1}$$

Now if the first term on the right side of equation 1 is positive everywhere and the second negative everywhere, N_z is positive everywhere.

Lemma 3.1 *Given a scalar B-spline surface*

$$\sigma(u,v) = \sum_{j=0}^{n-1}\sum_{i=0}^{m-1} a_{i,j} B_{i,k_u,\tau_u}(u) B_{j,l_v,\tau_v}(v),$$

then $\frac{\partial \sigma(u,v)}{\partial u} > 0$ *whenever* $a_{i+1,j} - a_{i,j} > 0$

Proof: Consider partial derivative of σ with respect to u.

$$\frac{\partial \sigma}{\partial u} = \sum_{j=0}^{n-1}(k-1)\sum_{i=0}^{m-2}\frac{a_{i+1,j} - a_{i,j}}{\tau_{i+k} - \tau_{i+1}} B_{i,k-1}(u) B_{j,l}(v).$$

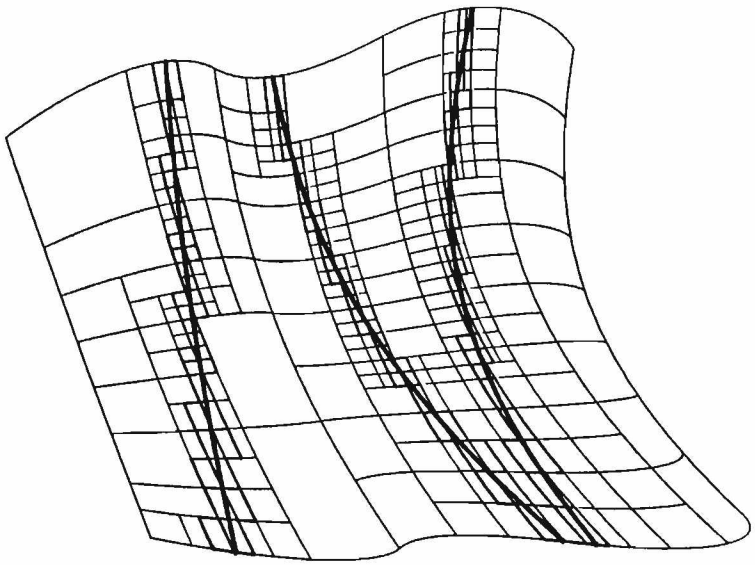

Figure 6: Adaptive silhouette extraction, in R^3.

Object	*Adaptive*	*Non adaptive*
saddle	1	5.7
discont	1	3.6
pawn	1	5.4
queen	1	3.4
rook	1	4.0
knight	1	2.1

Table 1: - adaptive and non adaptive silhouette extraction relative time.

Since the blending functions B are nonnegative, and the difference $\tau_{i+k} - \tau_{i+1} > 0$, if $a_{i+1,j} - a_{i,j} > 0$, for all i and j, $\frac{\partial \sigma}{\partial u}$ will be positive everywhere.

Lemma 3.1, and an analogous version for $\frac{\partial \sigma}{\partial v}$, can be used on the x and y coordinate functions to provide a simple test for ruling out the existence of silhouette curves in a surface. Needing only $O(mn)$ first order differences, it can act as an additional termination condition for the subdivision.

This adaptive subdivision reduces the theoretical number of surfaces resulting from $O(\frac{1}{\epsilon^2})$ in the non adaptive case to $O(\frac{1}{\epsilon})$, when again ϵ is used as flatness tolerance! Figure 6 shows such an adaptive subdivision for a surface with three silhouette curves. Also table 1 compares the relative times adaptive and non adaptive subdivision techniques consumes.

The reverse of lemma 3.1 is not true, that is, a surface may have no silhouette curves, and yet the differences may not all have the same sign. A 2D counterexample in Figure 7 shows a control polygon with 2 silhouette points, but a curve with none.

We can now define the adaptive subdivision to extract silhouette curves. A combined termination condition for this highly recursive algorithm is used. The flatness test of the surface is used with the silhouette existence condition developed above to form a termination test that will be referred as $Terminate(S)$, a boolean predicate.

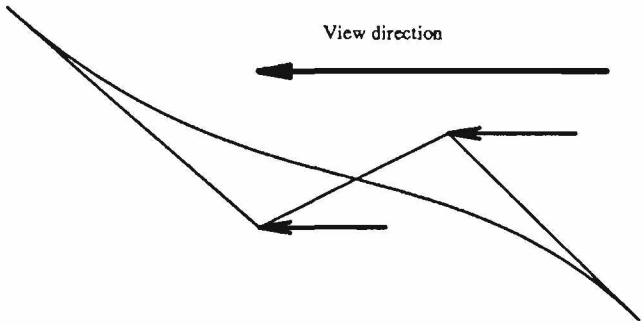

Figure 7: Curve with no silhouette points but with control polygon with two silhouette points.

```
PROCEDURE ExtractSilhouette( Srf )
BEGIN
    Srf1, Srf2;

    IF Srf may have silhouette DO
        IF Terminate(Srf) DO
            add Srf to silhouette surface list;
        END;
        ELSE DO
            Subdivide Srf into two pieces:  Srf1 and Srf2;
            ExtractSilhouette( Srf1 );
            ExtractSilhouette( Srf2 );
        END;
    END;
END;
```

Figure 8 shows the pawn object with the resulting potential silhouette sub-surfaces from another view (left).

Once done, one can trace the silhouette surfaces resulting from the subdivision process to form a piecewise linear approximation of the real continuous silhouette. This solution is frequently only a poor approximation. or else the process is time consuming. To avoid these problems, our method applies an improvement stage. In this numerical stage, an attempt is made to improve each vertex of the piecewise linear polyline approximation so that it is on the real silhouette.

Let p_0 be an approximated silhouette point extracted from the subdivision. Then:

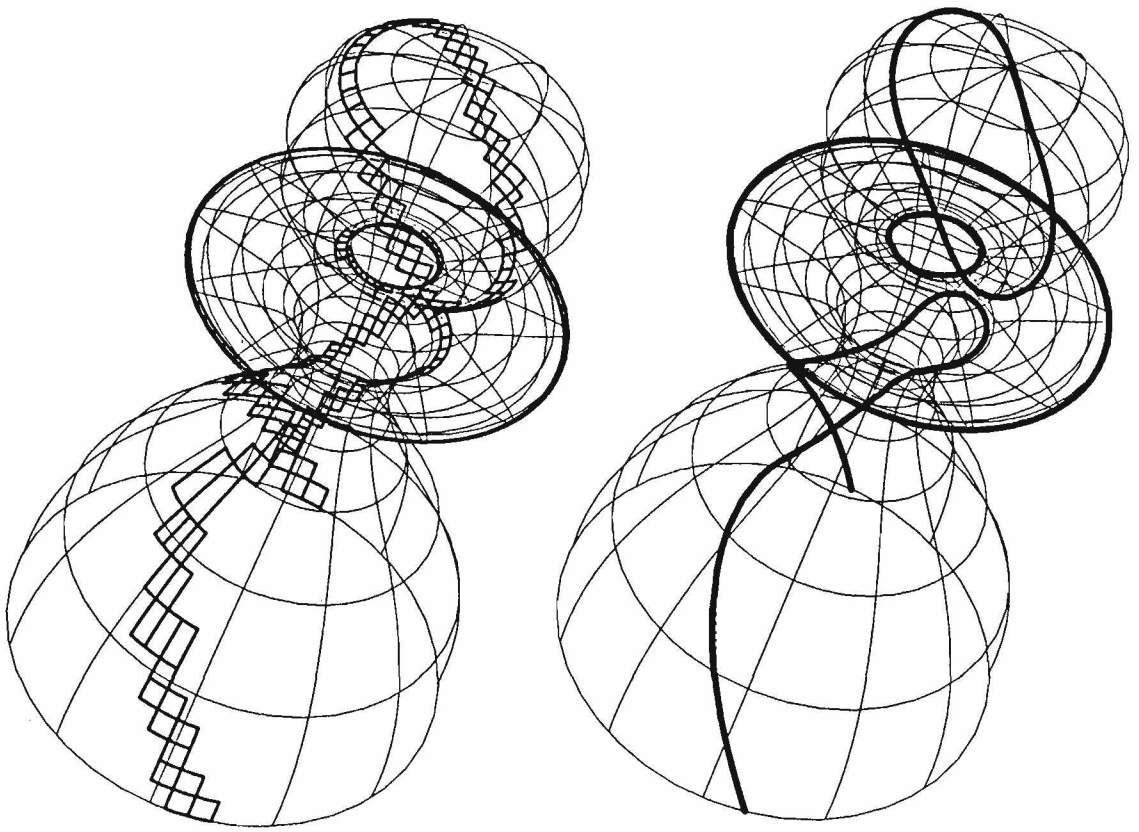

Figure 8: Sub-surfaces, with silhouette, from different view (left), and improved (right).

1. The approximation point p_0, is relatively close to the real silhouette.
2. If $p_0 \rightarrow s$, a point on the silhouette, the exact location of s on the silhouette is not important.
3. A surface silhouette point, s, is a point where $N_z(u^s, v^s) = 0$.
4. Let ϵ be the accuracy required and $|N_z^{p_i}| < \epsilon$.

Armed with that, the idea is quite simple: "walk" on the surface parametric domain in the direction that maximizes the changes in N_z (gradient) to minimize $|N_z|$. The direction is evaluated only once (to speed the process), since (by 1 above), the direction is not going to be changing much, and (by 2 above) the exact location on the silhouette is not important. The direction to "walk", $\vec{\Delta}$ is exactly $\vec{\Delta} = (\frac{\partial N_z(u^{p_0}, v^{p_0})}{\partial u}, \frac{\partial N_z(u^{p_0}, v^{p_0})}{\partial v})$ which is evaluated numerically (see Figure 9). By doing so, we reduced the dimentionality of the problem from a $2D$ problem in the parametric space, to a $1D$ problem along the line with direction $\vec{\Delta}$.

Given surface S and the point p_0, the "walking" algorithm is:

1. Evaluate $\vec{\Delta}$ at p_0, where $\vec{\Delta} = (\frac{\partial N_z(u^{p_0}, v^{p_0})}{\partial u}, \frac{\partial N_z(u^{p_0}, v^{p_0})}{\partial v})$ and set the "walking" direction to $\vec{\Delta}$.

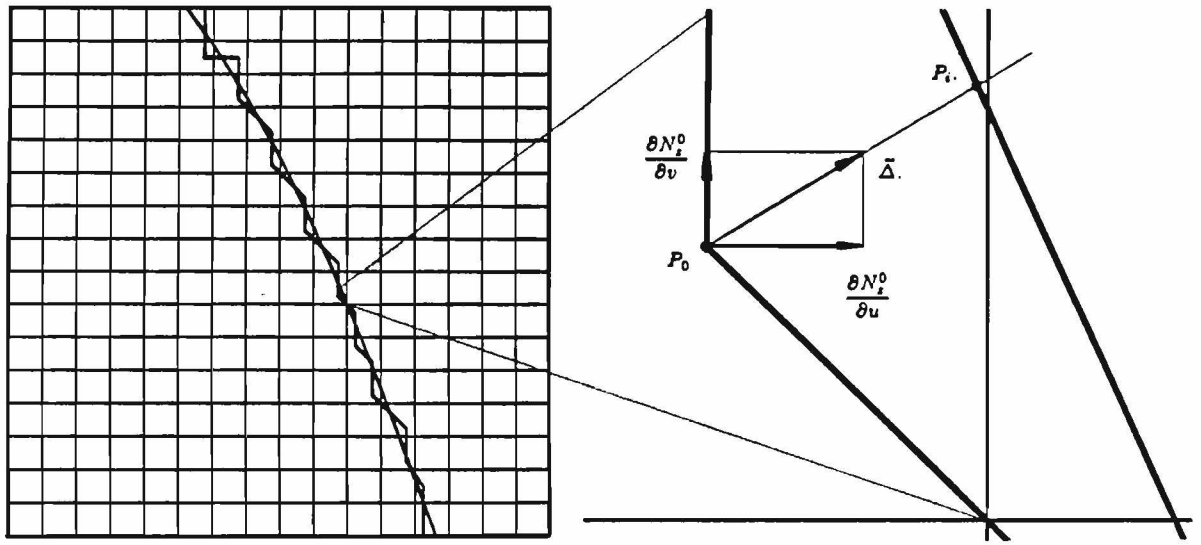

Figure 9: The numerically improved silhouette (of the saddle object) in parametric space.

2. Evaluate δ at p_0, where $\delta = \frac{dN_z(u^{p_0}, v^{p_0})}{dr}$

3. Let

$$p_1 = p_0 + \frac{\vec{\Delta}}{\|\vec{\Delta}\|} \frac{N_z(u^{p_0}, v^{p_0})}{\delta}$$

   Evaluate $N_z(u^{p_1}, v^{p_1})$, at p_1.

4. Let i be 1.

5. If $|N_z(u^{p_i}, v^{p_i})| < \epsilon$ stop: p_i is the improved point.

6. Using $N_z(u^{p_{i-1}}, v^{p_{i-1}})$ and $N_z(u^{p_i}, v^{p_i})$ linearly interpolate (extrapolate) p_{i-1} and p_i to form p_{i+1} such that $N_z = 0$ at p_{i+1}, and evaluate the new $N_z(u^{p_{i+1}}, v^{p_{i+1}})$:

   Let t be the solution for

$$N_z(u^{p_{i-1}}, v^{p_{i-1}})t + N_z(u^{p_i}, v^{p_i})(1-t) = 0$$

   and let

$$p_{i+1} = p_{i-1}t + p_i(1-t)$$

7. $i = i + 1$.

8. Goto 5.

Stages 5 - 8 are exactly the secant method. Because the surface is relatively flat from flatness testing during the subdivisions, the accuracy is improved by almost a magnitude on each iteration. The method usually converges to sufficient accuracy within 3 to 4 iterations. The result is a piecewise linear interpolant to the real silhouette. One can adaptively add and improve points, until the distance between the unimproved and the improved point is less than given tolerance.

Figure 8 also shows the same pawn, this time with the improved silhouette (right).

4 Curve-Curve Intersection

Curve-curve intersection is not a problem specific to hidden curve removal algorithms. The solution to the curve-curve intersection problem is generally not available in analytic form, and some kind of subdivision or iterative technique is used to determine those points [5,19,30]. We do not want to be restricted to a specific curve type or order, but would like as general a solution as possible. While the subdivision process guarantees convergence to all intersections, it is slow, and so effort is invested in finding methods to eliminate unnecessary subdivisions.

Like any other *divide and conquer* method, one needs to define a termination condition for the subdivision algorithm. For example one might use curve flatness tests and approximate it as a line for the intersection purposes. Alternatively curve length smaller than a given tolerance might be used. Again, this termination test will be referred as $Terminate(C)$ boolean predicate. The generic algorithm to solve the problem is therefore:

```
PROCEDURE CurveCurveIntersection( C1, C2 )
BEGIN
    C1^a, C1^b, C2^a, C2^b;

    IF Terminate(C1) and Terminate(C2) DO
        IF C1 bbox intersects C2 bbox DO
            Add intersection point to solution point list.
        END;
    END;
    ELSEIF C1 bbox intersects C2 bbox DO
        Subdivide C1 into two:  C1^a, C1^b;
        Subdivide C2 into two:  C2^a, C2^b;
        CurveCurveIntersection( C1^a, C2^a );
        CurveCurveIntersection( C1^a, C2^b );
        CurveCurveIntersection( C1^b, C2^a );
        CurveCurveIntersection( C1^b, C2^b );
    END;
END;
```

However, this generic algorithm is very expensive. Subdivision is exponential in nature, and the key operation is a fast and tight (see below) test checking whether the two curves intersect. Few ways to achieve that are known, and most of them depend on the fact the Bezier/NURB curves are bounded by the control polygon convex hull. The simplest test bounds each curve by an axes parallel box found using the curve control polygons (bounding box as it called) and uses them for the intersection test. Since the bounding boxes might intersect even when the curves do not, this type of bounding is not tight. A better bound can be achieved, but usually this makes the bounding box intersection comparison more complex and slow. The

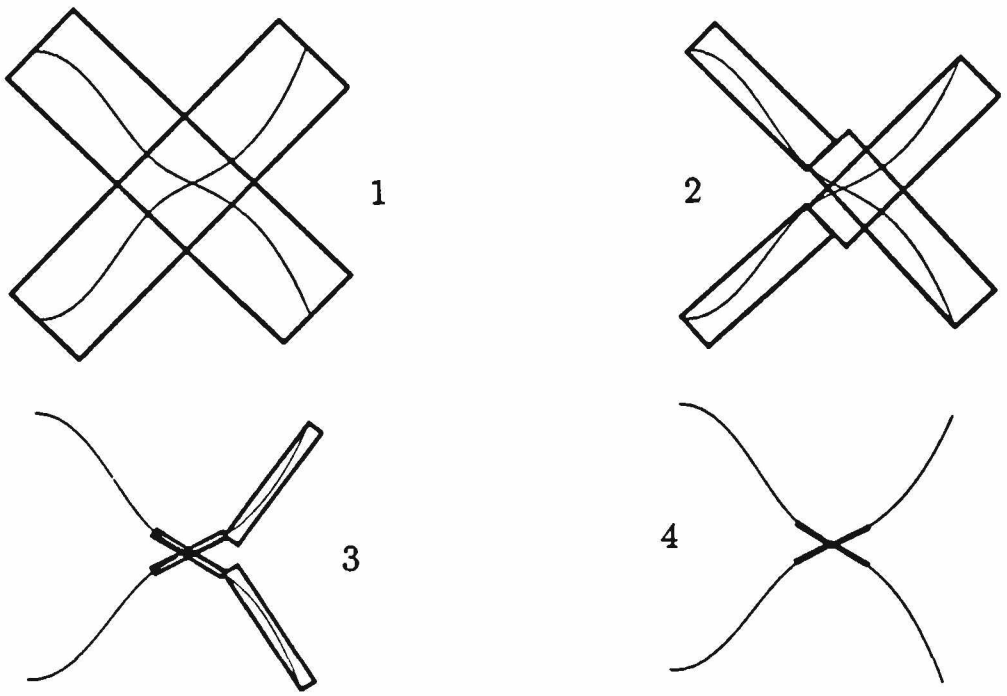

Figure 10: 4 stages of the curve/curve intersection subdivision using strip bounding boxes.

convex hull of the control polygon might be used, but generating and testing with them are neither simple nor fast. Strips, or rotated boxes, allows bounding diagonal curve as tight as horizontal or vertical ones. A better bound (but again more complex) is the fat arc [29]. As Bezier and NURB curves tend to converge to a constant radius curves when they are subdivided, this type will bound them extremely well (for comparison also see [29]). The strip was selected in our case for its simplicity, and yet its much tighter than axes parallel bound. Figure 10 shows few stages of the subdivision using strips.

If it can be shown that two curves have exactly one intersection, and the numerical solution will converge to it, it usually will converge much faster to the root than using only subdivision. Uniqueness of the solution can be determined, using the cone test [28] by comparing the possible tangent directions of the two curves and requiring them not to overlap. Finding a condition to guarantee convergence is harder task, and heuristic approach is used: if the strip width/length ratio is small, the curve is almost a straight line, and 2D Newton Raphson is likely to converge. This numerical method is therefore applied any time the two curves have at most one solution and both strip bounding boxes are elongated enough. If successful, the algorithm can stop, otherwise the subdivision is continued. This numerical stage is initialized with the two subdivided curves, and with their middle parametric point as initial guess.

Figure 11 shows one stage of Newton Raphson process: the initial guesses are P_0^α, P_0^β for curves α and β, respectively. Using the evaluated tangents at these points, linear interpolation is evaluated for both, to find the next point, which is mapped back to the curves parametric domain as P_1^α and P_1^β.

The subdivided curves (and not the original given curves) are used to make sure the

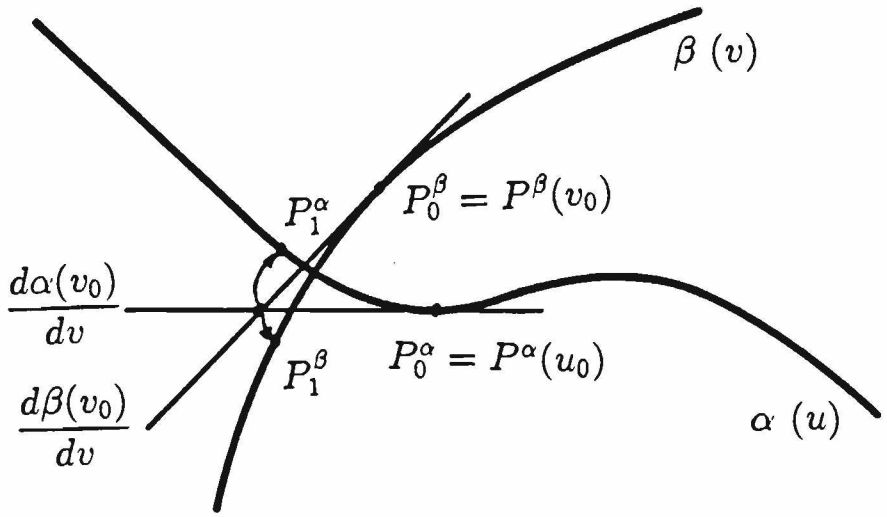

Figure 11: Newton Raphson method is applied whenever its possible.

solution, if found, will be bounded to the right range. The process is iterated until successful or until one of the failing conditions becomes true.

The main problems in this iteration process are that the parameterization is usually nonlinear, and so once the solution of the current iteration is found, it must be mapped back to the parametric space. Fortunately, the curves are the result of subdivision and small, and therefore their parameterization is fairly linear.

The improved curve/curve intersection is therefore:

```
PROCEDURE CurveCurveIntersection( C1, C2 )
BEGIN
    C1a, C1b, C2a, C2b;

    IF Terminate(C1) and Terminate(C2) DO
        IF C1 bbox intersects C2 bbox DO
            Add intersection point to solution point list.
        END;
    END;
    ELSEIF C1 and C2 have at most one intersection,
        and both are ''elongated'' enough DO
        IF C1 and C2 have common end point DO
            Ignore intersection point.
        END;
        ELSEIF Attempt to apply Newton Raphson succeeded DO
            Add point to solution point list.
        END;
```

	Axes Para. BBOX		*Strips BBOX*		*Strips + NR*	
Example	Subdiv	Time	Subdiv	Time	Subdiv	Time
1	138	22.5	104	22.4	0	1
2	98	3.3	75	3.95	4	1
3	1091	2.14	449	1.29	174	1
4	98	7.85	36	4.5	4	1

Table 2: - curve/curve intersection - relative time comparison, and number of subdivisions required.

```
        ELSEIF C1 bbox intersects C2 bbox DO
            Subdivide C1 into two:  C1^a, C1^b;
            Subdivide C2 into two:  C2^a, C2^b;
            CurveCurveIntersection( C1^a, C2^a );
            CurveCurveIntersection( C1^a, C2^b );
            CurveCurveIntersection( C1^b, C2^a );
            CurveCurveIntersection( C1^b, C2^b );
        END;
    END;
    ELSEIF C1 bbox intersects C2 bbox DO
        Subdivide C1 into two:  C1^a, C1^b;
        Subdivide C2 into two:  C2^a, C2^b;
        CurveCurveIntersection( C1^a, C2^a );
        CurveCurveIntersection( C1^a, C2^b );
        CurveCurveIntersection( C1^b, C2^a );
        CurveCurveIntersection( C1^b, C2^b );
    END;
END;
```

In other word, the subdivision technique, and the Newton Raphson method are interleaved until success is reached. The subdivisions presented by the bounding boxes of Figure 10, were the only ones required After four subdivision stages the Newton Raphson method has been applied successfully.

A test is performed, to make sure that the two curves have an interior point as their only intersection point. If not, a trivial comparison of the two end point will detect that and will reduce the cost of applying the numerical method to an end condition where it is likely to fail.

This process is much faster than using only axes parallel bounding boxes, and is also faster than using only strips based subdivision. Table 2 and Figure 12, compare 4 examples, 3 of which are drawn in Figure 12, while the first is simply intersection of X axis with Y axis.

It is required that we find the intersections between curves which are tangent to each

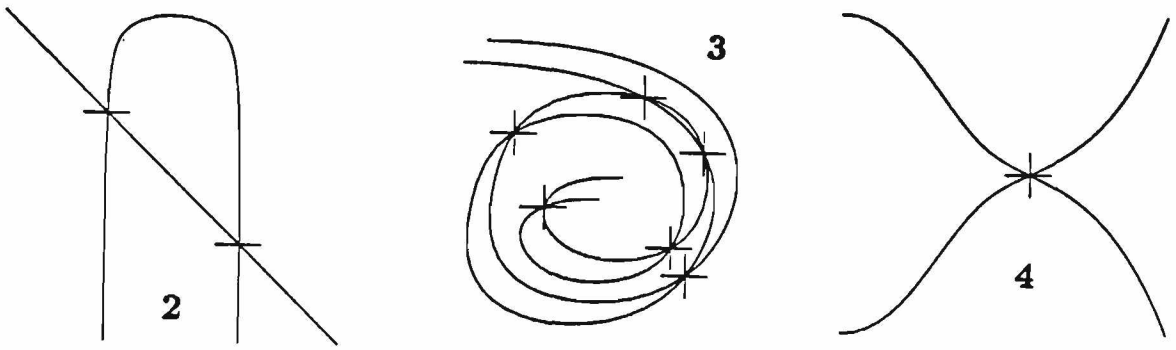

Figure 12: 3 examples of curve/curve intersections (see Table).

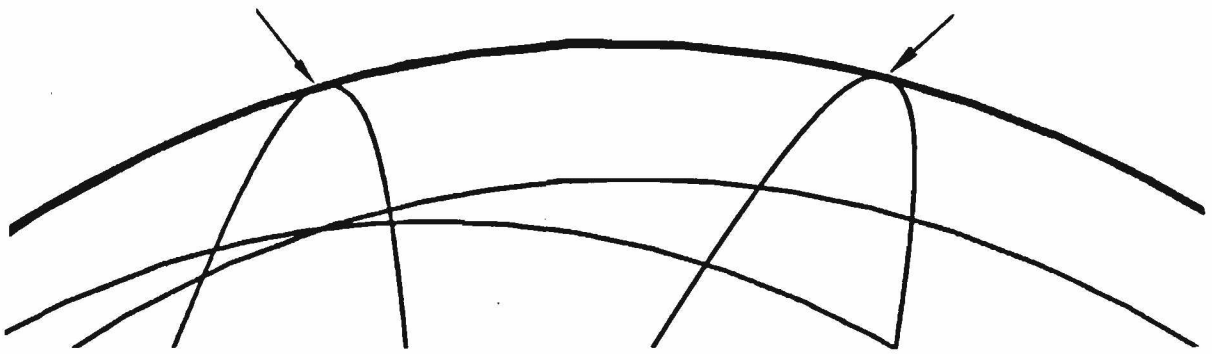

Figure 13: Tangent CCI.

other. Many intersections between iso-parametric curves and silhouettes are of this kind. For example, figure 13 is an enlargement of one such problem region in figure 1. If two curves have the same tangent line at their intersection point, the ϵ selected as the error measure for the computation method affects the results. If it is too small, then the method may miss the intersection; if it is too big, several (approximated) intersection points may result close to each other, where only one actually exists. When surfaces have been approximated by polygons exactly the same problem occurs. However, crossing of active curves in the projection plane must be detected since the visibility may change at such intersections (Lemma 2.5).

In this specific case most of the tangential curve intersection can be considered in a simpler framework. Consider two parametric space curves that map onto curves in R^3 intersecting at simultaneous tangencies. In most cases (see Figure 13) one of the curves is a silhouette curve and the other is isoparametric. Hence considering the intersection in parametric space reduces it to the problem of intersecting a planar curve with a straight line.

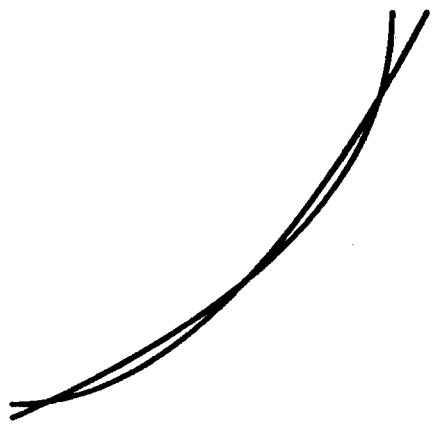

Figure 14: Monotone curves have multiple intersection

Another numerical instability may arise when dealing with curves whose projections are identical or have identical subcurves. If the curves are exactly the same, one can be eliminated. If two distinct curves project onto the same curve in the image space, but have different orders, knot vectors, or parametric continuity, there is currently no stable, computationally feasible method for analytically determining that the two curves are identical. Instead a heuristic threshold on the maximum number of valid intersections is used. This solution causes some overhead which might be reduced by analytical detection of such cases.

Given a curve-curve intersector, finding all the intersections among N curves the straightforward way of intersecting each curves against all the others is $O(N^2)$, which is the worst case possible. Usually one can do much better. Sweep algorithms to sort and improve this order have been developed, mainly for straight line segments [15,9,22,25]. They improve the average case order to $O(NlogN)$, but have the same worst case behavior of $O(N^2)$. Extending this notion to an arbitrary curve type environment is complex since curves, unlike lines, may intersect more than once. In fact, two spline curves can be continuous and have no zeros in their first and second derivatives, but can intersect multiple times, as Figure 14 demonstrates.

Although the current implementation of this algorithm uses a simplified bounding box sweep, research in this area is in progress. More on this can be found in [3].

5 Surface-ray intersection

Finding intersections of rays with surfaces occur frequently in graphics and geometry problems, in particular in the ray-tracing rendering technique[26,33]. Usually freeform surfaces are first approximated into polygons or preprocessed into small simplified pieces. Both strategies require a relatively large amount of memory. Since only a partial invisibility count is needed, we need not solve for the exact intersection point.

Lemma 5.1 *Let S be a path connected continuous surface with no silhouette curves. If $P(\psi^S)$ is a simple curve, then S is visible relative to itself, and therefore, all curves belonging to S are totally visible relative to it.*

Proof: Since S has no silhouette curves, ψ^S (definition 2.4) consists only of S boundary curves, and $\mathcal{R}$ has exactly one element. Since $P(\psi^S)$ is a simple curve, P is bijective on the interior of its whole domain.

By making use of this lemma we can reduce significantly the number of subdivisions required to solve the surface-ray intersection. When resolving the visibility of a given point p often one can use lemma 5.1, and decide on p-visibility much sooner than using only a subdivision technique. This decision can be made even if the exact intersection point of the ray from V to p with the surface S is unknown. Using curve visibility propagation and surface coherence, the number of rays that need to be fired can be reduced.

Definition 5.1 *For a curve β, $Touching(\beta)$ is the set of all curves, other than β, that have one endpoint of their projections lying on β_P.*

If a curve α is split into α_1 and α_2 at the parametric value where $P(\alpha)$ intersected with $P(\beta)$, such that α_1 has a lower parametric range than α_2, then:

1. *α_2 is the* next *curve of α_1, and will be denoted as $\alpha_2 = Next(\alpha_1)$.*
2. *α_1 is the* previous *curve of α_2, and will be denoted as $\alpha_1 = Prev(\alpha_2)$.*
3. *$\alpha_1, \alpha_2 \in Touching(\beta)$.*

To use this coherence, the curve adjacencies of definition 5.1 must be determined and kept during the curve-curve intersection stage. Each time a curve is split, all the information on the splitting and split curves must be kept, in addition to the z level ordering relation between them. Although, this might look time consuming, it is not. Each time a curve is split (a relatively complex operation), a constant number of adjacency pointers is updated. By doing so, one can derive a set of rules by which the visibility is propagated so fewer rays need be fired. The following corollary is only a subset of such rules and which will be sufficient to demonstrate its power:

Corollary 5.1 *If η is an active curve, $\phi \in Touching(\eta)$, and γ is split into γ_1 and γ_2 such that $\gamma_1, \gamma_2 \in Touching(\eta)$, $\gamma_1 = Prev(\gamma_2)$ then (see Figure 15 for all references)*

1. *If η is a silhouette curve of the surface σ, and σ has only one silhouette, then η is visible relative to σ. See h.*
2. *If η is a visible silhouette curve, at least one of γ_1, γ_2 is visible. See a, b and c. If γ is a passive curve or the scene has only* closed *models, then exactly one of γ_1, γ_2 is visible. See c.*
3. *If η is visible but not a silhouette curve, then exactly one of γ_1, γ_2 is visible. See i.*

4. *If ϕ is passive curve and η is a boundary curve (but not a silhouette), and both have the same z at the touching point, then ϕ has the same visibility as η. See f.*
5. *If $\eta \in Touching(\phi)$, both are boundary curves, and both η and ϕ have the same z at their touching point, then η and ϕ have the same visibility. See g.*

Proof: All the above conclusions result directly from lemma 2.5.

Each of these cases is simple by itself, but in combination they form powerful visibility propagation tools. Corollary 5.1- 2 is unique, in that one needs more information so the visibility of the adjacent curve can be resolved. A visible silhouette on closed objects must bound two regions, exactly one of which is visible, so only a single adjacent curve should be tested for visibility to classify the regions. For open objects, if a silhouette curve splits a boundary curve, both subcurves may be visible (see a, Figure 15).

We shall use Figure 15 to demonstrate the propagation ability. Assume the framework curves are given in in the following order: the silhouette curve following by boundary curves 1, 4, 2, 3.

The visibility of each curve in the framework is checked first. Using Corollary 5.1, since only one silhouette curves exists, by result 1, it is visible. As this object is open, no propagation to the boundary curves is allowed. A ray is fired to test the visibility of boundary curve 1, and it is found to be visible. Using result 5.1-5, curves 2.1 (through $\hat{1}$, Figure 15) and 4.1 (through $\hat{2}$) are found to be visible. The next framework curve with unknown visibility is 4 which has been split into two, 4.1 and 4.2. The first half (4.1) visibility is already known, so only one ray for the second half (4.2) is fired, and which find it to be visible as well. Using result 5.1-5 curve 4.2 (through $\hat{3}$) sets curve 3.2 to be visible, which in turn sets 3.1 (through $\hat{4}$), using result 5.1-3, to be invisible. Since 3.1 touches curve 2.2 (at $\hat{5}$), at its end, 2.2 is also set, using result 5.1-5, to be invisible.

Instead of firing 8 rays, for the 8 active curves in the saddle surface (1, 2.1, 2.2, 3.1, 3.2, 4.1, 4.2 boundary curves, and one silhouette), only 2 were fired. The question on the optimal way to order these curves is still open. But the improvement in this specific case in much greater. Using result 5.1-4, since all the iso-parametric curves touch boundary curves at the same z value, they inherit their visibility from that of the boundary. Using result 5.1-3, the visibility of the boundary curves is propagated to the interior iso-parametric pieces, so no ray need to be fired for any iso-parametric curve.

In addition to all the above, if the number of surfaces is small, a cache of the subdivided surfaces can be handled. If a surface was subdivided once for given ray, all but the last few steps will be identical for nearby rays. The main problem with this approach is the amount of memory it may require, and so it may be useful only if the scene consists of a small number of surfaces.

Table 3 compares this propagation in terms of number of rays fired for active and passive curve and relative time to test the visibility of all the curves. Note the relation between the two is not linear mainly because of the cache used.

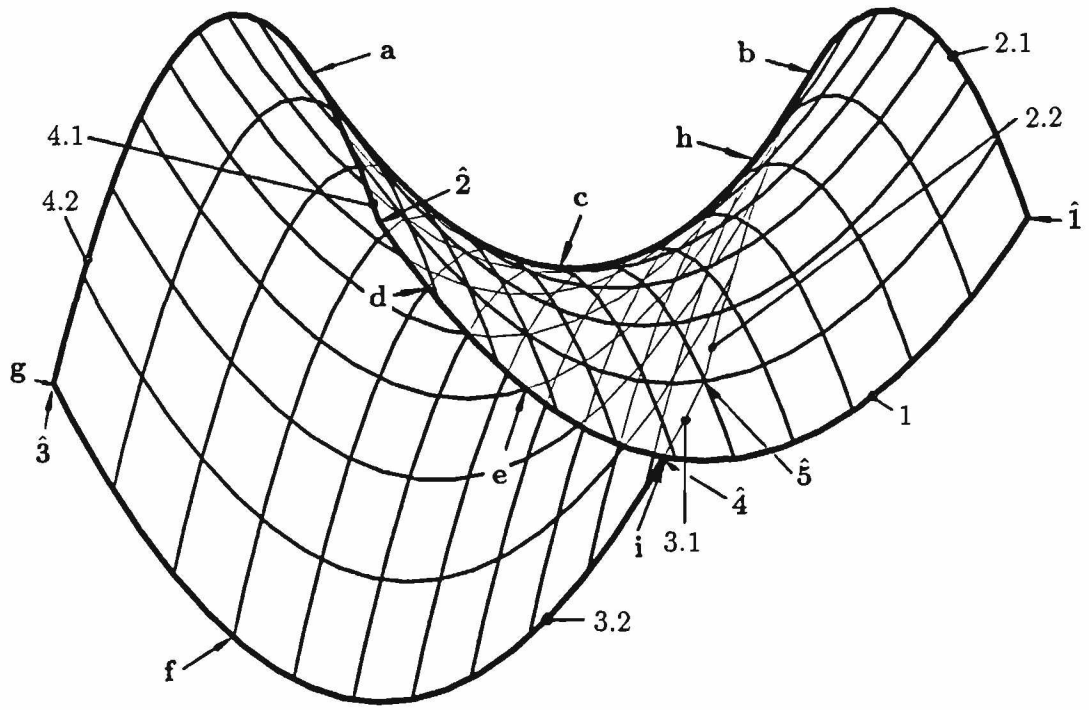

Figure 15: The curves visibility is propagated - firing 2 rays is enough for all curves.

	Propagation		*No propagation*		
	act. rays	pas. rays	rel. time	act. rays	pas. rays
discont	9	14	5.5	19	96
saddle	2	2	8	8	54
wiggle	11	7	4.5	21	95
pawn	12	68	1.9	17	152

Table 3: Visibility propagation comparison.

6 Trimmed Surfaces

An important issue to address is the support of trimmed surfaces. A careful look in the teapot (Figure 16) uncovers a problem with the design near its handle/body joint. The model was originally done by juxtaposing the surface descriptions for the handle and body without trimming the surfaces, so the handle and the body surfaces interpenetrate. Curves which should be hidden at the joint are visible, since this algorithm cannot detect such cases.

Using trimmed surfaces in the model and in the algorithm eliminates this difficulty. The modifications necessary in our algorithm to support trimmed surfaces were small and occurred in two stages of the algorithm.

1. The curve extraction stage needs to trim the extracted curves against the trimming curves in the parametric space.

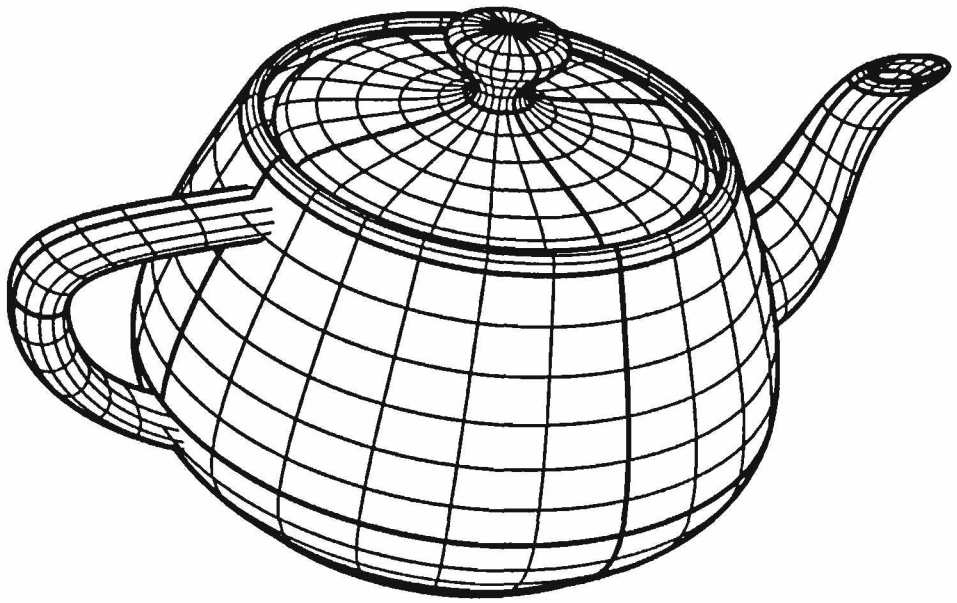

Figure 16: A hidden curve view of the teapot.

2. The surface-ray intersection stage should test if the point on surface S that hides the tested point is in a trimmed part of S. The trimming curves should be propagated through in the subdivision process, and a totally trimmed out surface can be another termination condition for the subdivision.

Figure 17 shows the results of the modified algorithm.

7 Acknowledgements

Particular thanks to the Alpha_1 group for all their help and support in this research.

References

[1] A. Appel. The Notion of quantitative Invisibility and the Machine Rendering of Solids Proceedings ACM National Conference 1967.

[2] J. Blinn. A scan line algorithm for displaying parametricaly defined surfaces. Computer Graphics 12. 3. 1977.

[3] B. Bruderlin, E. Cohen, and G. Elber. A Plane-Sweep Hidden-Surface Algorithm for Curved Surfaces. Technical report No. 90-006, Computer Science, University of Utah.

[4] C. Sequin and P. Wensley. Visible Feature Return at Object Resolution. IEEE Computer Graphics and Application, May 1985, pp 27-50.

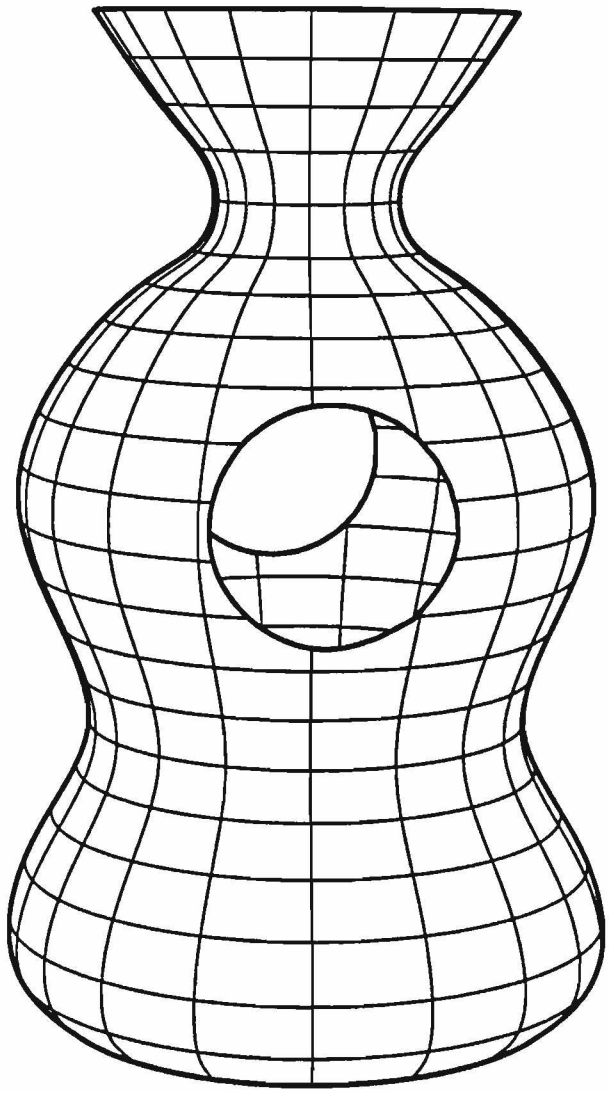

Figure 17: Support of trimmed surfaces.

[5] 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).

[6] D. Griswold and M. Cohen. Automatic Illustration of 3D Geometric Models: Lines. Proceedings of the 1990 Symposium on Interactive 3D Graphics.

[7] V. Fuson. Application of a Hidden Line Algorithm to Surface Visualization. MS thesis, Computer Science, University of Utah, 1984.

[8] R. Galimberti and U. Montanari. An algorithm for Hidden Line Elimination CACM 12, 4, 206, April 1969.

[9] G. Hamlin and C. W. Gear. Raster-scan hidden surface algorithm techniques Computer Graphics, Vol 11, pp 206-213, 1977.

[10] C. Hornung. An Approach to a Calculation-Minimized Hidden Line Algorithm. Computer & Graphics, Vol 6, No 3, pp 121-126, 1982.

[11] C. Hornung. A Method for Solving the Visibility Problem IEEE CG&A July 1984, pp. 26-33.

[12] C. Hornung, W. Lellek, P. Pehwald, and W. Strasser. An Area-Oriented Analytical Visiblity Method for Displaying Parametrically Defined Tensor-Product Surfaces. Computer Aided Geometric Design, 2 (1985) 197-205.

[13] J. Foley and A. Van Dam. Fundamental of Interactive Computer Graphics.

[14] Gerald Farin Curves and Surfaces for Computer Aided Geometric Design. Academic Press, Inc. Harcourt Brace Jovanovich, Publishers.

[15] R. Guting. New Algorithm for Special Cases of the Hidden Line Elimination Problem. Computer Vision, Graphics, and Image Processing 40, 188-204 (1987).

[16] I. Sutherland, R. Sproull, and R. Schumacker A Characterization of ten Hidden-Surface Algorithms. Computer Surveys, Vol. 6, No. 1, Mar. 1974, pp. 1-55.

[17] T. Kamada and S. Kawai. An Enhanced Treatment of Hidden Lines. ACM Transaction on Graphics, Vol 6, No. 4, October 1987, Pages 308-323.

[18] J. Kripac. Classification of edges and its application in determining visibility. Computer Aided Design, Volume 17, Number 1, January/February 1985.

[19] J. Lane and R. Riesenfeld. A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces. IEEE Transaction on pattern analysis and machine intelligence, vol. PAMI-2, No. 1, January 1980.

[20] L. Li. Hidden-line algorithm for curved surfaces. Computer Aided Design, Volume 20, No. 8, October 1988, Pages 466-470.

[21] P. Loutrel. A Solution to the Hidden_line Problem for Computer Drawn Polyhedra IEEE Transactions on Computers, Vol. C-19, No. 3, 205-213, March 1970.

[22] M. Mckenna. Worst-Case Optimal Hidden-Surface Removal. ACM Transaction on Graphics, Vol 6, No. 1, January 1987, Pages 19-28.

[23] C. Montani and M. Re. Vector and Raster Hidden-Surface Removal Using Parallel Connected Stripes. IEEE Computer Graphics and Application, July 1987, pp 14-23.

[24] K. Morken Some Identities for Products and Degree Raising of Splines. UNKNOWN

[25] O. Nurmi. A Fast Line-Sweep Algorithm for Hidden Line Elimination. BIT 25 (1985), pp 466-472.

[26] J. Peterson. PRT - A High Quality Image Systhesis System for B-spline Surfaces. MS thesis, Computer Science Dept., University of Utah, Dec. 1987.

[27] J. Rankin. A Geometric Hidden-Line Processing Algorithm. Comput. & Graphics Vol. 11, No. 1, pp. 11-19. 1987.

[28] T. Sederberg and R. Meyers. Loop Detection in Surface Patch Intersections. Computer Aided Geometric Design 5, pp 161-171, 1988.

[29] Thomas W. Sederberg, Scott C. White and Alan K. Zundel. Fat Arcs: A bounding Region with Cubic Convergence. Technical report No. ECGL-88-1, Engineering Computer Graphics Laboratory, Brigham Young University, Provo, Utah 84602, June 23, 1988.

[30] T. Sederberg and S. Parry. Comparison of Three Curve Intersection Algorithms. Computer Aided Design, Volume 18, Number 1, January/February 1986.

[31] T. Sederberg and Alan K. Zundel Scan Line Display of Algebraic Surfaces. SIGGRAPH 89, pp 147-156.

[32] Steven G. Satterfield and David F. Rogers. A Procedure for Generating Contour Lines From a B-Spline Surface IEEE Computer Graphics and Application, Aprit 1985, pp 71-75.

[33] M. Sweeney. Ray Tracing Free-Form B-Spline Surfaces. IEEE Computer Graphics and Application, Februaury 1986, pp 41-49.