Hierarchical Visibility Culling for Spline Models[*]

Subodh Kumar Dinesh Manocha

University of North Carolina
Chapel Hill, NC 27599-3175, USA
Ph: (919) 962-1943. Fax: (919) 962-1799.
Email: {kumar,manocha}@cs.unc.edu
WWW: http://www.cs.unc.edu/~{kumar,manocha}

Abstract

We present hierarchical algorithms for visibility culling of spline models. This includes *back-patch* culling, a generalization of *back-face* culling for polygons to splines. These algorithms are extended to trimmed surfaces as well. We propose different spatial approximations for enclosing the normals of a spline surface and compare them for efficiency and effectiveness on different graphics systems. We extend the culling algorithms using hierarchical techniques to collection of surface patches and combine them with view-frustum culling to formulate a ONE (Object-Normal Exclusion)-tree for a given model. The algorithm traverses the ONE-tree at run time and culls away portions of the model not visible from the current viewpoint. These algorithms have been implemented and applied to a number of large models. In practice, we are able to speed-up the overall spline rendering algorithms by about $20-30\%$ based on back-patch culling only and by more than 50% using ONE-trees.

Keywords: NURBS rendering, Visibility, Back-patch, CAGD, ONE-tree.

1 Introduction

Many large-scaled CAD models like those of automobiles, submarines and airplanes are represented using parametric spline surfaces. Such models are composed of tens of thousands of surfaces and many applications in CAD/CAM, virtual reality, animation and visualization need to render such models at interactive frame rates. Current rendering systems for large spline models on commercial graphics systems are not able to achieve real-time frame rates for applications involving virtual worlds, walkthroughs and immersive design.

Many techniques based on ray-tracing, scan-line conversion, pixel-level subdivision and polygonization have been proposed for rendering parametric spline models [Cat74, NSK90, LCWB80]. However, polygonization based approaches are able to make efficient use of the hardware capabilities of the current graphics systems and are significantly faster than the rest [AES93, Dea89, LC93, SC88, FK90, RHD89, AES91, KML95]. The resulting algorithms use uniform or adaptive subdivision of spline surfaces to compute polygonal approximation. The approximation is a function of the current viewing direction and is typically, re-computed at each frame. The resulting polygons are then rendered using the standard graphics pipeline. However, the best known algorithms based on such approaches are only able to render models consisting of up to $600-700$ patches at interactive frame rates on high-end commercial systems like SGI Reality Engine 2 [RHD89, KML95].

There is considerable literature on visibility preprocessing and on-line culling of polygonal data-sets. Our goal is to extend these techniques to curved and spline models. The techniques for polygonal models include *view-frustum culling*, *obscuration culling* and *back-face culling.*

- **View-frustum culling** methods use spatial data structures like octrees and hierarchical traversals of such structures to cull out portions of the model not visible [FVFH93]. These have been extended to spline models using bounding boxes and convex hulls of control polytopes.
- **Obscuration culling** techniques utilize algorithms for hidden-surface removal and occlusion culling [FVFH93]. However, most of the algorithms for polygonal models are non-trivial to implement and are unable to compute the visi-

*Supported in part by an Alfred P. Sloan Foundation Fellowship, ARPA ISTO Order A410, NSF Grant MIP-9306208, ARO Contract P-34982-MA, NSF Grants CCR-9319957 and CCR-9625217, ONR Contract N00014-94-1-0738, ARPA Contract DABT63-93-C-0048 and NSF/ARPA Center for Computer Graphics and Scientific Visualization. Approved by ARPA for public release - distribution unlimited.

ble surfaces in real-time for large models. The extension of such algorithms to splines is even more difficult. It involves computation of silhouettes and projection curves and is difficult to perform in real-time for even small models.

- **Back-face culling** consists of comparing the normal of a polygon with the viewing direction. If the normal points away, the resulting polygon is not rendered. Most of the high-end graphics system have an implementation of this technique as a part of the graphics pipeline.

In this paper, we extend back-face culling for polygons to **back-patch culling** for splines. The idea of back-patch culling was introduced in [KML95, SAE93]. Our algorithm is more general, more efficient and simpler to implement. It involves efficient computation and representation of bounds on normals of a patch. We present algorithms for exact back-patch culling for perspective projection. The algebraic complexity of exact back-patch culling is high for interactive applications and we present a number of techniques for approximating them using different spatial data structures. We evaluate these spatial approximations using two criteria:

- *Efficiency*: It captures the overhead of visibility computation for each frame. It measures the time spent in visibility tree traversal.
- *Effectiveness*: It measures the number of primitives and surfaces being culled away by the visibility algorithm.

In most cases, there is a trade-off between these two measures, and overall performance can be maximized in different ways for different graphics systems. For example, a graphics pipeline with polygon rasterization as the bottleneck should attempt to increase the effectiveness at the cost of efficiency, shifting some load to the processor performing visibility-tree traversal. Our overall algorithm permits such fine-tuning.

We also combine these algorithms with hierarchical data structures, view-frustum culling and apply them to cull away portions of a large model. In particular we present a new hierarchical data structure, ONE-tree (Object-Normal Exclusion tree), which is used for view-frustum as well as hierarchical back-patch culling. The resulting algorithms have been implemented on different graphics systems and we discuss their performance on a number of large models composed of up to tens of thousands of spline surfaces. Back-patch culling improves the frame rate by $20-30\%$ by itself and the ONE-tree can increase

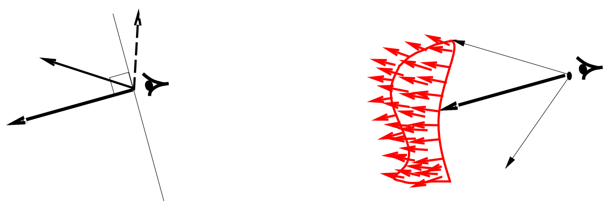


Figure 1: Back Facing Normals

the rendering performance by more than 50%. The algorithms presented are also applicable to algebraic surfaces and general implicit models.

The rest of the paper is organized in the following manner. We review the notion of Gauss maps and visibility for curved surfaces in Section 2 and use it to establish an exact back-patch culling condition for parametric spline surfaces. It is extended to trimmed patches and we consider a number of data structures for spatial approximations. We combine it with hierarchical data structures and outline the construction of ONE-tree in Section 3. We also present an efficient tree traversal algorithm. Finally in Section 4, we discuss the implementation and performance of the algorithms.

2 Visibility Computations

In general, the exact computation of the visible portions of a spline model is a non-trivial problem requiring silhouette computation and projection curves [KM94]. In this section, we show that it is relatively simple to perform an approximate visibility check to find most of the spline surfaces that are completely invisible from the current viewpoint. The algorithms presented are general and applicable to all curved and orientable surfaces with first order continuity. They involve computation of Gauss maps and bounds on Gauss maps. In the rest of the paper, we will demonstrate these algorithms on Bézier surfaces.

A Bézier surface, $\mathbf{F}(u,v)=\left(\frac{X(u,v)}{W(u,v)}, \frac{Y(u,v)}{W(u,v)}, \frac{Z(u,v)}{W(u,v)}\right)$, is specified using control points and is a linear combination of Bernstein functions [Far93]. Moreover, the entire patch is contained in the convex hull of the control points. We denote the convex polytope of a surface as $\mathbf{P}_F$ and its smallest volume axis-aligned bounding box as $\mathbf{B}_F$.

2.1 Back-facing Patches

Given a closed solid model whose boundary is composed of Bézier patches, many of the patches are not visible because their outward normals are facing away from the viewer. In particular, if all the surface normals for a Bézier patch point away from the eye point and the viewing direction we refer to it as a *back-patch* (Fig. 1(b)). The algorithm for back-patch culling needs to compute a bound on the surface normals and we make use of Gauss maps to compute these bounds.

2.2 Gauss Map

The partial derivative vectors of a Bézier surface, $\mathbf{F}(u,v)$, with respect to u and v, respectively $\mathbf{F}_u(u,v)$ and $\mathbf{F}_v(u,v)$, are contained in the tangent-plane at $\mathbf{F}(u,v)$. In the rest of the section, we shall drop the (u,v) suffixes from these vector-valued functions for more concise notation. At any point on the surface $\mathbf{F}(u,v)$, the normal direction is given by $\mathbf{N} = \mathbf{F}_u \times \mathbf{F}_v$. Bézier surfaces belong to the class of surfaces called orientable surfaces such that their normals can be oriented 'inside' or 'outside' the surface [Nei66]. For a given model we can orient all the surfaces by reversing the order of control points such that $\mathbf{N}(u,v)$ points outside for each u,v for each patch. Gauss maps provide a tool to compute $\mathbf{N}$.

> The *Gauss map* $\mathbf{G}$ of a surface, $\mathbf{F}$, is a map $\mathbf{G} : \mathbf{F} \rightarrow S^2$, the 2-Sphere in $\mathcal{R}^3$, which takes the point $\mathbf{F}(u,v)$ into the translation of the vector $\mathbf{U}(u,v)$ to the origin, where $\mathbf{U}(u,v)$ is the unit vector in the direction of $\mathbf{N}(u,v)$ [Nei66].

The function $\mathbf{G}(u,v)$ can be used to compute the unit normal of the surface at the point (u,v). However, this can be relatively expensive to compute and in our application we instead use a pseudo-Gauss map – we translate $\mathbf{N}$ instead of $\mathbf{U}$. The pseudo map can be represented as a Bézier surface itself, and is therefore, defined using a set of control points. If $\mathbf{F}$ is a tensor product $m \times n$ polynomial surface, the pseudo-normal surface is a $(2m-1) \times (2n-1)$ Bézier surface. If $\mathbf{F}$ is a rational surface, the degree of the cross-product of the partial derivative vectors is $4m \times 4n$. However it can be simplified [KML95] to:

$$\mathbf{N} = \frac{\mathbf{f}_\mathrm{u} \times \mathbf{f}_\mathrm{v}\mathrm{W} - \mathbf{f}_\mathrm{u} \times \mathbf{f}\mathrm{W}_\mathrm{v} - \mathbf{f}\mathrm{W}_\mathrm{u} \times \mathbf{f}_\mathrm{v}}{\mathrm{W}^3}, \qquad (1)$$

where $\mathbf{F} = \frac{\mathbf{f}}{W}$. Thus, the pseudo normal surface is a $3m \times 3n$ rational Bézier surface and can be represented by a $(3m+1) \times (3n+1)$ mesh. The control points of the pseudo map are evaluated from the control points of the original surface $\mathbf{F}$ as follows:

1. Compute the control points of $\mathbf{f}_u$ and $\mathbf{f}_v$. $\mathbf{f}_u$ and $\mathbf{f}_v$ are also Bézier surfaces.
2. If $\mathbf{F}$ is polynomial surface, compute the cross product of $\mathbf{f}_u$ and $\mathbf{f}_v$ and return. The cross product is computed by term-wise multiplication and subtraction for each coordinate.

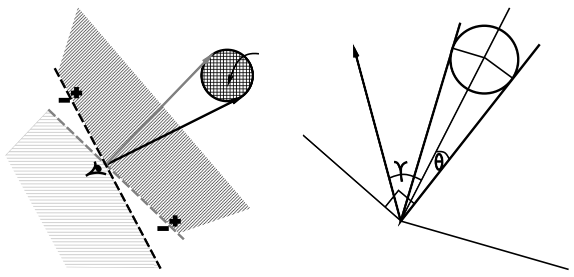


Figure 2: Visibility Computation

3. Compute W_u and W_v, respectively.
4. Piecewise multiply the W terms in (1), compute all cross products and perform the coordinate-wise subtraction.
5. Elevate the degrees along the u and v, respectively, by 1 to compute the control points of the pseudo-normal surface.

Notice that we do not need to divide by the weights of the pseudo-normal patch. This is because the vectors $[X\ Y\ Z]$ and $[WX\ \ WY\ \ WZ]$ are parallel. This means that the open pyramid anchored at the origin that bounds the normal surface remains the same. Hence all the normal directions are still contained in the convex hull or bounding box of $[WX\ \ WY\ \ WZ]$.

2.3 Back-patch Condition

In general a point $\mathbf{p}$ on a surface with a normal $\mathbf{n}$ is back facing if

$$\vec{\mathbf{Ep}} \cdot \vec{\mathbf{n}} > 0,$$

where $\mathbf{E}$ is the eye point (see Fig. 1(a)). In other words, an entire patch, $\mathbf{F}$, is back facing if, $\forall (u,v) \in [0,1] \times [0,1]$, $\mathbf{N}(u,v)$ makes an acute angle with the vector joining the eye to $\mathbf{F}(u,v)$.

If S is a bounding sphere for the patch in the X,Y,Z space, with radius r and center $\mathbf{C}$, we compute a region in space that contains all normal directions for back-patches. This is demonstrated in 2D in Fig. 2(a). The rays $l1$ and $l2$ bound the sphere and the patch, and lines $p1$ and $p2$ are, respectively, perpendicular to these rays. Only the half spaces $p1^-$ and $p2^-$, respectively, may contain normal directions for visible points. That is, there exist normal directions contained in $p1^-$ or $p2^-$ that are obtuse angles with some ray bounded by $l1$ and $l2$. Hence the intersection of half-spaces $p1^+$ and $p2^+$, call it $\mathbf{H}$, is the back-patch region.

For back-patch condition to be satisfied, the angle that $\mathbf{n}$ makes with the vector joining the eye point to $\mathbf{C}$, must be less than the angle $p1$ makes with it (as shown in Fig. 2(b)). Thus direction $\mathbf{n}$ lies in $\mathbf{H}$ if

$$\begin{aligned}
& \cos(\gamma) > \cos(90-\theta) \\
\Rightarrow\quad & \cos(\gamma) > \sin(\theta) \\
\Rightarrow\quad & \frac{\vec{\mathbf{EC}}\cdot\vec{\mathbf{n}}}{|\vec{\mathbf{EC}}||\vec{\mathbf{n}}|} > \frac{r}{|\vec{\mathbf{EC}}|} \\
\Rightarrow\quad & (\mathbf{C}-\mathbf{E})\cdot\vec{\mathbf{n}} > r|\vec{\mathbf{n}}|
\end{aligned}$$

$\mathbf{E}$ is the only point not known till run-time, hence at run-time this test performs one vector difference and one dot product. The problem is, for values of $\mathbf{E}$ close to $\mathbf{C}$, θ becomes large, making the bound loose. This problem is alleviated in the next section.

2.4 Analytic Bound

The back-patch condition outlined in the previous section can be applied to any primitive. For specific cases we can apply that condition in an analytic manner. For example, for Bézier surfaces, the visibility can be tested as follows:

A patch is back facing if, $\forall(u,v)\in[0,1]\times[0,1]$,

$$\begin{aligned}
& \mathbf{EF}(u,v)\cdot\mathbf{N}(u,v) > 0 \\
\Rightarrow\ & (\mathbf{F}(u,v)-\mathbf{E})\cdot\mathbf{N}(u,v) > 0 \\
\Rightarrow\ & \mathbf{F}(u,v)\cdot\mathbf{N}(u,v) - \mathbf{E}\cdot\mathbf{N}(u,v) > 0 \\
\Rightarrow\ & \mathbf{F}(u,v)\cdot\mathbf{N}(u,v) > \mathbf{E}\cdot\mathbf{N}(u,v) \qquad (2)
\end{aligned}$$

Each of the two terms of (2) can be written as Bézier functions. The visibility test reduces to checking if there exists a $(u,v)\in[0,1]\times[0,1]$ such that the bi-variate function $(\mathbf{F}(u,v)-E)\cdot\mathbf{N}(u,v)\le 0$, which, in turn, can be solved by testing if the function has any roots in the domain $[0,1]\times[0,1]$. This is a high degree function, $4m+4n$ for an $m\times n$ Bézier surface, and solving for exact roots leads to efficiency and accuracy problems. In addition, this operation is too expensive to be performed interactively at run-time. Fortunately, we can bound the function by the minimum and maximum valued control points (using Bernstein basis). This results in the following algorithm:

1. Pre-compute the minimum value, m_F, of $\mathbf{F}(u,v)\cdot\mathbf{N}(u,v)$. m_F is the minimum control point of the scalar function $\mathbf{F}\cdot\mathbf{N}$. The control points of $\mathbf{F}\cdot\mathbf{N}$ are computed as follows:
    - Compute the $[X\ \ Y\ \ Z]$ control points of N using the algorithm in section 2.2.
    - Pairwise multiply $[WX\ \ WY\ \ WZ]$ functions of $\mathbf{F}$ and $\mathbf{N}$. This results in the rational control points of $\mathbf{F}\cdot\mathbf{N}$. Compute their pairwise sum.
    - Degree elevate the W coordinate of $\mathbf{F}$ to $4m\times 4n$.
    - Divide each control point computed in step 1 by the corresponding control point of the degree elevated W.

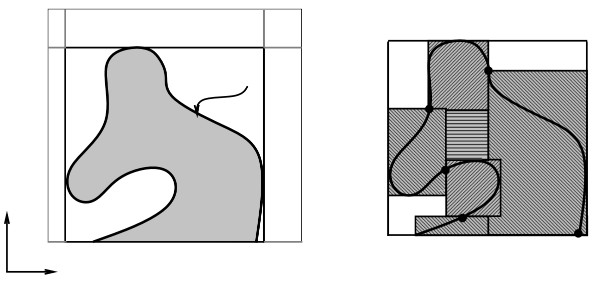


Figure 3: Increased Effectiveness for Trimmed Patches

2. Pre-compute and store, the maxima of each co-ordinate, respectively, of $\mathbf{N}$(u,v). Call this vector of three maxima, $\mathbf{M_N}$.
3. At run-time, if $m_F > \mathbf{M_N}\cdot\mathbf{E}$, the patch is back facing.

2.5 Trimmed Patches

Many real world CAD models consist of trimmed surfaces. Trimmed surfaces have trimming-curves defined on the domain as shown in Fig. 3(a). Such surfaces consist only of the points enclosed by the curves, as opposed to the full domain. Using the full domain for visibility computation can lead to reduced effectiveness — the untrimmed region of the patches may be back-facing while the full patch is not. We define *fullness* of a domain as the ratio of area in the un-trimmed region to the area of the domain. We do not compute exact fullness, but estimate it by tessellating the trimming curves into piecewise linear segments, and calculating the enclosed area. The first step to increase the fullness involves the computation of the tight-fitting domain, D (Fig. 3(a)):

> Compute the minima and maxima of the trim curve in the domain in both u and v coordinates, respectively, u_m, u_M, v_m, and v_M. The new patch is obtained by subdividing the original patch at $u=u_m, u=u_M, v=v_m$ and $v=v_M$ [Far93].

Unfortunately, for many trimmed patches this does not result in a good bound. Such patches are subdivided as follows:

1. Choose points on the curve, that decompose it into 'rectangle like' sub-curves. We have used the inflection points of the curve in our implementation, and they yield good decompositions.
2. Find the closest fitting rectangular box around each component of the curve. $\{\mathbf{R}_0\cdots\mathbf{R}_k\}$ (Fig. 3(b)).

3. Add rectangles to fill the span $D - \cup\{\mathbf{R}_i\}$.
4. Recursively subdivide domains that have $fullness < U_f$. We have found $U_f = \frac{3}{4}$ to work well in our implementation.

This subdivision decomposes a patch $\mathbf{F}$ into a set of possibly overlapping patches, say, $\{\mathbf{F}_1, \mathbf{F}_2 \ldots\}$. One possibility is to treat these patches as separate primitives, and perform visibility culling on each of them. Unfortunately, this method leads to patch-proliferation and inefficiency. In addition, it also complicates the triangulation algorithm. Further, extra processing is required to prevent cracks in the rendered image. Instead, we compute the bounds for each sub-patch, and finally re-merge them by computing union of these bounds. This is more effective than using the full domain. The heuristics that we have found to work well are:

- Do not subdivide for more than 3 levels.
- Always merge sub-patches with any overlap.
- Ensure that patches are not decomposed into more than 3 sub-patches. If the number of unmerged sub-patches is large; merge the closest ones recursively.

3 Hierarchical Visibility

For large databases, the number of primitives is quite large. There is a substantial overhead of performing tests for each primitive per frame. On the other hand we can hierarchically group [Cla76] primitives together and use smaller number of tests to eliminate invisible primitives. A wide variety of space partitioning methods e.g. R-Trees, Quad trees, BSP, Oct trees, etc. for view-frustum culling have been proposed in the literature. Different methods are suited for different applications. Each of these methods can also be applied to back-patch culling in the following manner:

> Instead of constructing the hierarchy on objects, we construct a hierarchy on the gauss maps of the objects. We refer to this as the *normal space* hierarchy, as opposed to *object space* hierarchy, which is used for view-frustum culling.

In our application, we use an approach similar to R-Trees [BKSS90]. In essence, we maintain a hierarchy of bounding boxes. This method greatly simplifies the merging of the normal and object space hierarchies into one structure. Thus we have to store and traverse only one tree. The objects space hierarchy is constructed based on object space adjacency.

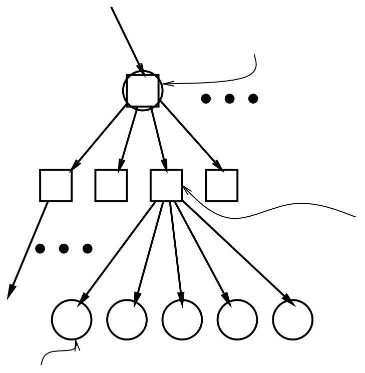


Figure 4: *One*-Tree

In fact, for normal space partitioning also, grouping patches too far apart in object space renders the method ineffective. Thus object space proximity is important for both hierarchies. We exploit this to construct a single hierarchy for both spaces.

3.1 *ONE*-Tree

Naturally, the optimal hierarchies for view-frustum and back-patch culling are different. It is possible to maintain two hierarchies, one for view-frustum culling and one for back-patch culling. But this requires storage and traversal of two trees. Since, proximity is still the basic criterion for both trees, we can actually merge them into one structure. We call this *One*-tree or Object-Normal exclusion tree. It facilitates the exclusion of invisible objects from further processing, and is based on both object and normal spaces.

Since, it is not always prudent to group objects based on a combination of the two hierarchies, we maintain the freedom to group objects based entirely on one hierarchy. A small sphere can never be completely back-facing, but it could still be view-frustum culled. Similarly about half of nearly parallel patches will have a high probability of facing the same way but they may not be close in object space. Hence we allow different criteria for grouping at different nodes of the tree. This leads to three different types of nodes in *ONE*-tree (Fig. 4) —

- *O*-nodes: Perform only view-frustum test.
- *N*-nodes: Perform only back-patch test.
- *ON*-nodes: Perform both tests.

Allowing different node types can potentially increase the tree size, but it allows us to avoid performing potentially ineffective tests. The type of a node is determined at the time of hierarchy construction.

Define the function *looseness*, $\mathcal{L}$, of a pair of bounding boxes B_1 and B_2:

$$\mathcal{L}(B1, B2) = \frac{Volume(B1) + Volume(B2)}{2 \times Volume(B1 \cup B2)}$$

This measures the effectiveness of combining boxes $B1$ and $B2$ at a higher level, and lies between 0 and 1. For sub-trees T_1 and T_2, let the object space bounding boxes be $\mathbf{B}_{F_1}$ and $\mathbf{B}_{F_2}$ and let the normal space bounding boxes be $\mathbf{B}_{N_1}$ and $\mathbf{B}_{N_2}$. To construct the tree, at each level, for each such pair do:

```
if $|\mathcal{L}(\mathbf{B}_{F_1}, \mathbf{B}_{F_2}) - \mathcal{L}(\mathbf{B}_{N_1}, \mathbf{B}_{N_2})| < K_u$
    classify $T_1 \cup T_2$ nodes as potential $ON$-nodes.
    Assign $\mathcal{L}'(\mathbf{B}_1, \mathbf{B}_2) = \mathcal{L}(\mathbf{B}_{F_1}) \times \mathcal{L}(\mathbf{N}_{F_1}, \mathbf{N}_{F_2})$.
else if($\mathcal{L}(\mathbf{B}_{F_1}, \mathbf{B}_{F_2}) < \mathcal{L}(\mathbf{B}_{N_1}, \mathbf{B}_{N_2})$)
    classify $T_1 \cup T_2$ nodes as potential $O$-nodes.
    Assign $\mathcal{L}'(\mathbf{B}_1, \mathbf{B}_2) = (\mathcal{L}(\mathbf{B}_{F_1}, \mathbf{B}_{F_2}))^2$.
otherwise
    classify $T_1 \cup T_2$ nodes as potential $N$-nodes.
    Assign $\mathcal{L}'(\mathbf{B}_1, \mathbf{B}_2) = (\mathcal{L}(\mathbf{N}_{F_1}, \mathbf{N}_{F_2}))^2$.
```

Of all the $<T_i, T_j>$ pairs, choose the one with minimum $\mathcal{L}'(\mathbf{B}_i, \mathbf{B}_j)$. K_u is a user specified tolerance, and 0.5 is a good starting value. The value K_u can be changed to fine-tune the efficiency-effectiveness tradeoff. At $K_u = 1$, all nodes are ON-nodes and should be used for applications with high object-normal space correlation. Similarly $K_u = 0$ implies no nodes are ON-nodes.

In addition to classifying a patch as completely back-facing, we can also determine if a patch is completely facing towards the user. The condition becomes: $\mathbf{F}(u,v) \cdot \mathbf{N}(u,v) < \mathbf{E} \cdot \mathbf{N}(u,v), \ \ \forall(u,v)$. Now the maxima of $\mathbf{F} \cdot \mathbf{N}$, M_F, must be less than the minima of $\mathbf{E} \cdot \mathbf{N}$, $\mathbf{E} \cdot \mathbf{m}_{\mathbf{N}}$ (cf. section 2.4). Analogously for the non-analytic case, the cone in which the gauss-map must lie gets reflected about the eye point as shown in Fig.2(a). This test allows us to trivially accept sub-trees without further traversal down its branches. If all patches in a sub-tree are front facing, we render the full sub-tree. Similarly if a sub-tree is back-facing, we trivially reject the full sub-tree.

3.2 Tree Traversal

We use two flags to guide the traversal, *OnlyNormal* and *OnlyObject*. If *OnlyNormal* is set for a sub-tree, view-frustum tests are not performed even for the O-nodes or ON-nodes. This condition occurs, if a node decides that all its patches lie in the view-frustum but the ones that are back-facing are not yet determined. The function of *OnlyObject* is analogous. The traversal algorithm proceeds from the root as follows:

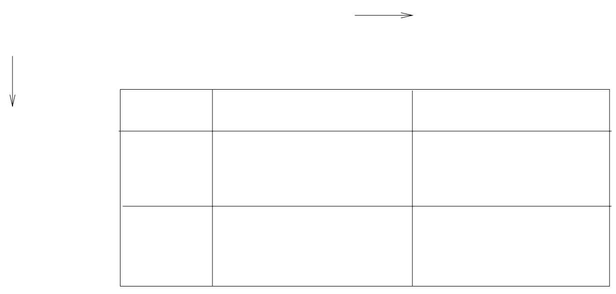


Figure 5: Tree Traversal

1. If flag *OnlyNormal* not set, and an object node, test object visibility.
2. If flag *OnlyObject* not set, and a normal node, test normal visibility.
3. Perform operation according to the table in Fig. 5.

3.3 Coherence

The visibility of models do not change significantly from one frame to the next. We exploit this coherence by starting the search at 'decision points' of the ONE-tree for the last frame. The decision points are those nodes of the tree at which the traversal terminated with either a 'Cull' or a 'Render' decision. In practice, we do not need to move up or down the tree from any start point by more than a level. The traversal proceeds as follows:

- If the decision in the last frame was 'Cull', and that for this frame is not 'Cull', traverse down the subtree.
- If the decision in the last frame was 'Render', and that for this frame is not 'Render', traverse up the subtree.

4 Implementation

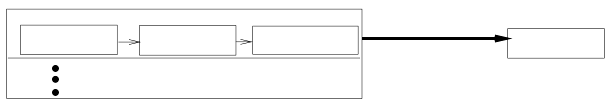


Figure 6: System Organization: Pixelplanes 5

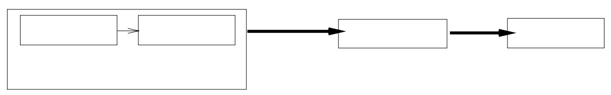


Figure 7: System Organization: SGI Onyx, $RE2$

We implemented the hierarchical visibility algorithm on two machines with quite different architectures – Pixelplanes 5 (PXPL5) and SGI Onyx. The two architectures are shown in Figs. 6 and 7.

Model	Num. Patches	% patches culled	Speedup on PXPL5	Speedup on Onyx
Pencil	570	35%	23%	27%
Dragon	5354	38%	32%	32%
Car	10,012	29%	19%	24%
Sub-room	36,206	42%	29%	NA

Table 1: The effectiveness of back-patch culling

We tested our implementation on a number of models. Scenes from some of these are shown on the color plate. For the performance figures quoted in this section, we recorded thousand frames long view sequences while a user pretended to inspect parts of the model.

We present results for both, simple back-patch culling without any hierarchy, and with the use of ONE-tree. The effectiveness does not change by using the hierarchy. For a given frame the same patches are culled away. The efficiency does improve by using the ONE-tree. About 10-25% fewer back-patch tests needed to be performed by using the tree.

Table 1 shows the effectiveness of back-patch culling using the cuboidal bounding box on the pseudo-normal patches. The actual speedup of the rendering is slightly higher for the Onyx, as CPU was always the bottleneck. Table 2 shows the effect of using different bounding volumes for the pseudo-normal patches. While effectiveness (% culled) of using tighter bounds is higher, the efficiency is lower, as seen by the number of tests needed per patch. Table 3 demonstrates the effectiveness of One-tree (with cuboidal bounds).

4.1 Choice of Bounding Volumes

We compute a minimum-volume, eight vertex, axis-aligned box, $\mathbf{B}_N$, bounding the control points of the pseudo-normal surface, $\mathbf{N}$. Each point on $\mathbf{N}(u,v)$ corresponds to a direction on $\mathbf{F}(u,v)$ and $\mathbf{P}_N$ and $\mathbf{B}_N$ define multiple sided polytopes in which all these directions lie. Testing for visibility reduces to checking whether each of these control points, or just the bounding box $\mathbf{B}_N$, is in the back-patch region $\mathbf{H}$.

Instead of using a sphere to bound the points on the surface, a rectangular box or the convex hull can be used. This increases the effectiveness of the technique but also increases the number of tests needed, thus reducing the efficiency. Similarly, for the pseudo-normal patch, a spherical bounding volume can be used. In general, owing to the higher degree of the normal patches, their control points tend to bound them less tightly, and spherical bounds are the most loose.

Table 2 lists the performance of back-patch culling. Some models have almost half of the patches culled away. Since most patches are relatively flat, we have found that using bounding boxes is good enough. For a rational bicubic patch, the control polytope of $\mathbf{N}(u,v)$ consists of 81 control points, and its convex hull typically has about $20-30$ points. On the other hand the bounding box has only eight points. In fact, instead of a cuboid bounding volume, a spherical bounding volume can give better speedup on some systems. The number of tests per patch reduces to one, while the average culling decreases from 36% to 29%.

For small systems where calculating or keeping extra bounding boxes is prohibitive, the view-frustum can be used as the bounding volume of the patch, specially for applications that use a narrow field of view.

4.2 ONE-Tree

On average, our implementation of ONE-tree culled 45-60% of the model. About 65%-75% of the tree nodes are ON-nodes. To evaluate their effectiveness, let us consider other options — on one end of this spectrum lies a ONE-tree with only object space culling. On average 30–50% of the models were culled using just the object space tree. Performing only normal-space culling, we could cull 20–45% of the models. Using a combination reduces the individual effectiveness of each method, since the sub-tree used in the ONE-tree is not necessarily optimal for either object space or normal space partitioning. In addition, the size of ONE-tree is larger. But overall, we can cull out more of the model, with fewer total number of tests.

In our experiments we found that back-patch culling is more effective when the eye-point is far away from a patch. This, in fact, provides a convenient symbiosis between object space and normal space visibility. When objects are farther from the eye, object space culling is less effective as more of the model lies in the viewing-frustum. On the other hand, for zoomed-up views, when back-patch culling is less effective, object space culling is able to cull out many off-screen objects.

5 Conclusions and Future Work

We presented an algorithm to perform back-patch culling and combined it, in a hierarchical framework, with the more standard view-frustum culling. This combination – the ONE-tree – can eliminate more than half of the model from further processing, but it slightly reduces the effectiveness of each method individually. Maintaining two different trees is an option but it is an open problem to better combine the partitioning schemes without increasing the tree

	Spherical bound		Cuboidal bound		Convex Hull	
Model	% culled	# Tests/patch	% culled	# Tests/patch	% culled	# Tests/patch
Pencil	33%	1	35%	1.44	36%	1.67
Dragon	32%	1	38%	1.80	44%	2.11
Car	22%	1	29%	1.66	33%	1.75
Submarine room	32%	1	42%	1.67	47%	1.81

Table 2: The effect of different bounds on the pseudo-normal patches

size significantly. In our algorithm, the children of a node in a subtree consist of non-overlapping bounding volumes. Extensions to allow such overlap could be useful. In addition, using different bounds on different levels of the tree can increase effectiveness.

We also introduced the idea of coherent tree traversal. We feel that a better exploitation of coherence is possible. For instance, if we could measure the 'degree of visibility' of a patch, we would be able to trivially reject 'highly invisible' patches in the next frame if the viewer position does not change significantly.

For trimmed patches, we presented techniques for partitioning the domains. We feel, our method is not optimum though, and has scope for improvement. Further, our experiments were limited to only a few thousand patches. For larger databases the fraction of invisible primitives is also normally larger, and our method should perform quite well.

	Per patch testing		One-tree	
Model	% culled	# Tests per patch	% culled	# Tests per patch
Pencil	35%	1.44	42%	1.41
Dragon	38%	1.80	48%	1.67
Car	29%	1.66	52%	1.65
Sub-room	42%	1.67	57%	1.59

Table 3: The effect of hierarchy on back-patch visibility

6 Acknowledgements

We thank Jim Boudreaux, Greg Angelini and Ken Fast at Electric boat for the storage and handling system model. David Forsey provided us with the dragon model. Ray Byrnes and Sam Schwartz deserve credit for the Ford car model.

References

[AES91] S.S. Abi-Ezzi and L.A. Shirman. Tessellation of curved surfaces under highly varying transformations. *Proceedings of Eurographics*, pages 385–397, 1991.

[AES93] S.S. Abi-Ezzi and L.A. Shirman. The scaling behavior of viewing transformations. *IEEE Computer Graphics and Applications*, 13(3):48–54, 1993.

[BKSS90] N. Beckmann, P. Kriegel, R. Schneider, and B. Seeger. R* tree: An efficient and robust access method for points and rectangles. In *International Conference on Management of Data*, pages 322–331, 1990.

[Cat74] E. Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, University of Utah, 1974.

[Cla76] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of ACM*, 19(10):547–554, 1976.

[Dea89] T. DeRose and M. Bailey et al. Apex: two architectures for generating parametric curves and surfaces. *The Visual Computer*, 5(5):264–276, 1989.

[Far93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., 1993.

[FK90] D.R. Forsey and V. Klassen. An adaptive subdivision algorithm for crack prevention in the display of parametric surfaces. In *Proceedings of Graphics Interface*, pages 1–8, 1990.

[FVFH93] J. Foley, A. VanDam, S. Feiner, and J. Hughes. *Computer Graphics principles and practice.* Addison Wesley, Menlo Park, California, 1993.

[KM94] S. Krishnan and D. Manocha. Global visibility and hidden surface algorithms for free form surfaces. Technical Report TR94-063, Department of Computer Science, University of North Carolina, 1994.

[KML95] S. Kumar, D. Manocha, and A. Lastra. Interactive display of large scale NURBS models. In *Symposium on Interactive 3D Graphics*, pages 51–58, Monterey, CA, 1995.

[LC93] W.L. Luken and Fuhua Cheng. Rendering trimmed NURB surfaces. Computer science research report 18669(81711), IBM Research Division, 1993.

[LCWB80] J.M. Lane, L.C. Carpenter, J. T. Whitted, and J.F. Blinn. Scan line methods for displaying parametrically defined surfaces. *Communications of ACM*, 23(1):23–34, 1980.

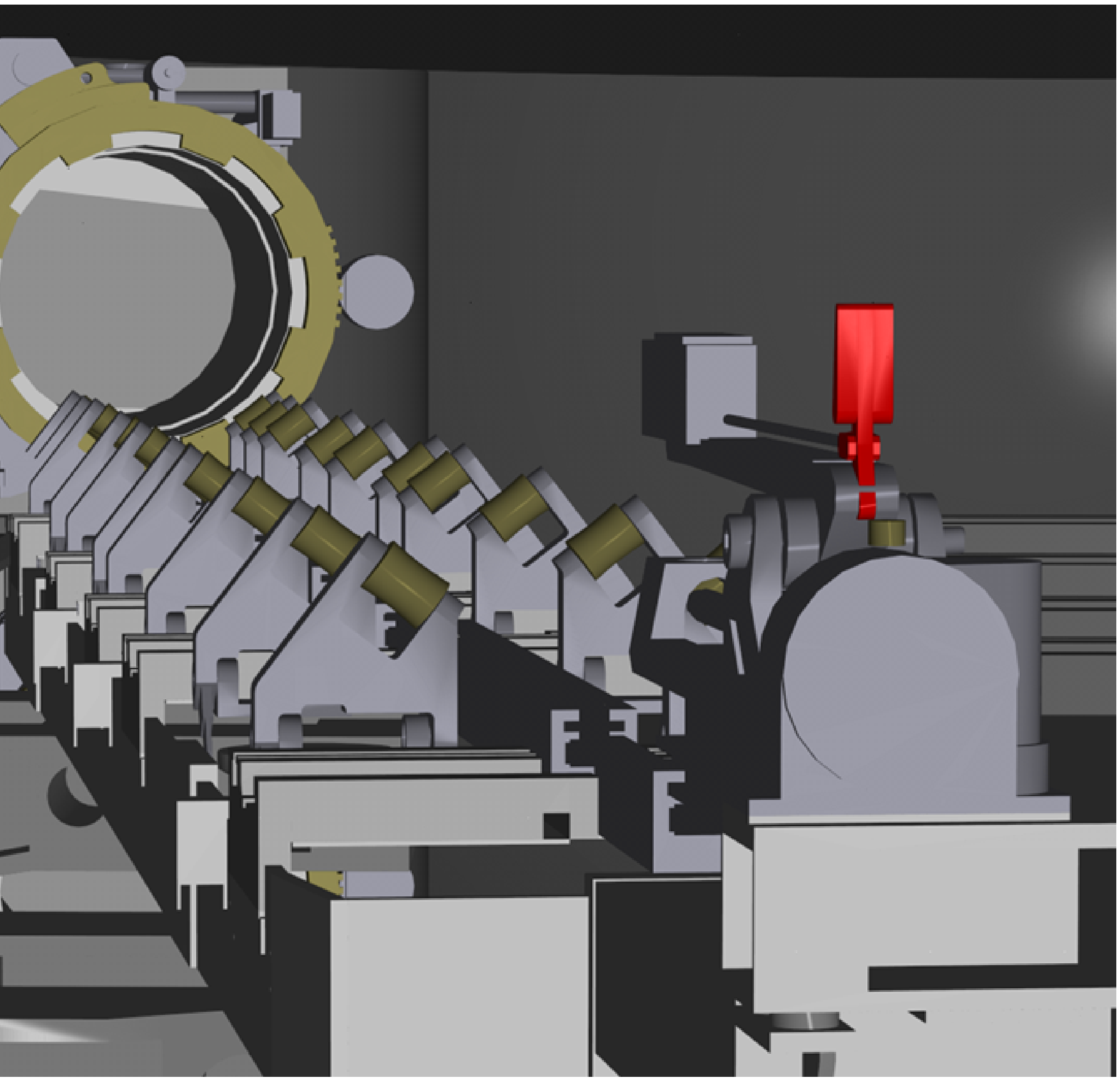
[Nei66] B. O' Neill. *Elementary Differential Geometry.* Academic Press, 1966.

[NSK90] T. Nishita, T.W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *ACM Computer Graphics*, 24(4):337–345, 1990. (SIGGRAPH Proceedings).

[RHD89] A. Rockwood, K. Heaton, and T. Davis. Real-time rendering of trimmed surfaces. *ACM Computer Graphics*, 23(3):107–117, 1989. (SIGGRAPH Proceedings).

[SAE93] L.A. Shirman and S.S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. In *EUROGRAPHICS*, pages 261–272, 1993.

[SC88] M. Shantz and S. Chang. Rendering trimmed NURBS with adaptive forward differencing. *ACM Computer Graphics*, 22(4):189–198, 1988. (SIGGRAPH Proceedings).



c)