Fast Backface Culling Using Normal Masks

Hansong Zhang* Kenneth E. Hoff III*

Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

This paper presents a novel method for fast and efficient backface culling: we reduce the backface test to one logical operation per polygon while requiring only two bytes extra storage per polygon. The *normal mask* is introduced, where each bit is associated with a cluster of normals in a normal-space partitioning. A polygon's normal is approximated by the cluster of normals in which it falls; the cluster's normal mask is stored with the polygon in a preprocessing step. Although conceptually the normal masks require as many bits as the number of clusters, we observe that only two bytes are actually necessary. For each frame (and for each viewing volume), we calculate the *backface mask* by ORing the normals masks of all normal clusters that are backfacing. The backface test finally reduces to *a single logical AND operation* between the polygon's normal mask and the backface mask.
CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Methodology and Techniques - Graphics Data Structures and Data Types.
Additional Keywords: visibility, backface culling

1 INTRODUCTION

In the rendering of polygonal models we have traditionally attempted to quickly remove or cull away portions of the model that are invisible with respect to a particular viewpoint. By removing these polygons early in the rendering process, we can substantially reduce the required workload. For a solid (closed), opaque model, polygons with normals facing away from the viewer must be invisible and can thus be omitted from further graphics processing. Each polygon is subjected to this backface test and culled as necessary, forming the well-known process called *backface culling*.

* CB #3175, Sitterson Hall, UNC-CH, Chapel Hill, NC 27514.
Email: {zhangh, hoff}@cs.unc.edu

Typically, the backface test involves calculating the dot product between the polygon's normal and the vector formed from the viewing point to any point on the polygon. If the result is negative then the polygon is facing towards the viewer; if the result is positive then the polygon is facing away from the viewer and is considered to be backfacing.

[KUMA96] has proposed a backface culling method that, in theory, can achieve sublinear performance. This method groups polygons and then uses frame-to-frame coherence to identify, track, and cull entire backfacing patches. This method achieves sublinear performance since it culls away polygons by groups. In practice, however, the application of this method is limited due to its model partitioning requirements and complexity (further explained in Section 4).

This paper introduces an efficient and lightweight approach to backface culling which requires very little effort to integrate into existing graphics systems. The key idea is a bitmask encoding scheme for the polygon normals and the normal space partitioning so that the backface culling operation is reduced to a single logical AND operation per polygon, with some inexpensive per-frame computational overhead. As a result, we are able to cull over 40 percent of the polygons and achieve a 60-80 percent speedup over hardware-only backface culling.

The rest of the paper is organized as follows: Section 2 describes the polygon normal bitmask, called the *normal mask*, and its encoding scheme; in Section 3 we explain how the bitmask encodings are used in backface culling; in Section 4 we show experimental results; Section 5 includes a comparison of our method with existing or possible alternatives; and then we conclude with section 6.

2 NORMAL MASKS

The core idea of this paper is to use a bitmask (*the normal mask*) to represent groups of normals in the normal space partitioning; this forms the basis for performing the backface culling test with only a single logical operation. This section briefly describes our normal clustering scheme and then focuses on the method by which these clusters are encoded with normal masks.

2.1 Normal Clustering

Grouping normals into clusters is not a new idea. Previously, normal clustering has been used for various purposes, including hierarchical backface culling [KUMA96] and backface culling of curved surfaces [SHIR93]. In these papers, the authors clustered the normal directions in a variety of ways; for example, Shirman partitioned the normal space into "normal cones". Our method is a simple alternative.

We partition the normal space by subdividing the surface of the unit sphere (illustrated in 2D in figure 1 below). This can be simplified by surrounding the sphere with an axis-aligned bounding cube and subdividing the six faces into two-dimensional grids of NxN rectangular cells. Each cell on the surface of the cube is then projected onto the sphere's surface by normalizing the vector formed from the sphere's center to points in the cell (in practice, of course, we only project the four corner vertices of the cell as an approximation). This forms a convenient surface subdivision of resolution NxNx6.

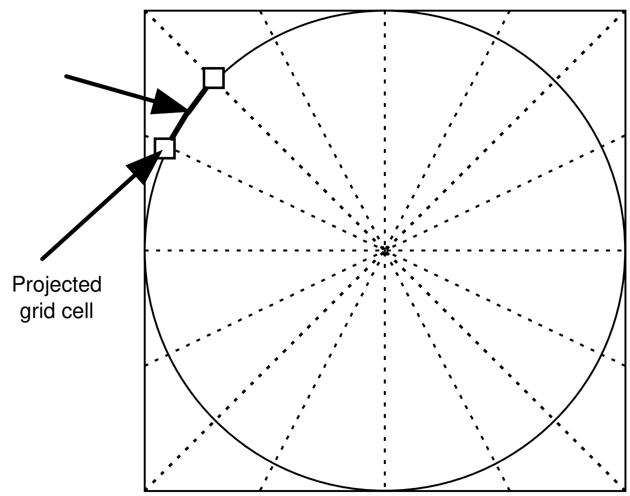


Figure 1: Normal space partitioning in 2D. The axis-aligned bounding box of the unit circle is partitioned by projecting each box side grid cell (line segment in 2D) onto the surface of the sphere. Each cell's projection forms a normal clustering.

Clearly this does not result in a uniform subdivision of the normal directions; however, it does offer the advantage of efficient classification of a normal into a normal cluster by requiring only a simple intersection test between a ray (defined by the normal) and a face of the cube.

2.2 Normal Encoding

The key component of our method is encoding into bitmasks the normal clusters in the normal space subdivision. Conceptually, imagine the bitmask as a long sequence of bits that has one bit for each normal cluster. The bit corresponding to the cluster occupied by a given normal will be set. We approximate a polygon's normal by the normal cluster in which it falls, and this approximation (i.e. membership information) can be stored with the polygon by storing the cluster's bitmask. Although conceptually this bitmask can have as many bits as the number of clusters, we do not require such an excessive representation since only one bit is set for a given normal mask (since a normal belongs to only one normal cluster). So we can represent a normal mask in a much more compact form, e.g. by treating the bitmask as a byte sequence and storing only the non-zero byte and its offset into the virtual byte array. We find it enough to use one byte for the offset, which means we can represent at most 256 * 8 or 2048 bits, i.e. the maximum number of clusters is 2048. In practice, we usually subdivide each face of the bounding cube (see the previous section) into a 16x16 grid which leads to 16x16x6 or 1536 normal clusters. In short, we have a two-byte representation for the normal of each polygon:

```
typedef struct {
    Byte byteOffset, bitMask;
} PolygonNormalMask;
```

3 BACKFACE CULLING

We now have a normal encoding scheme that allows for a very fast and efficient backface culling test. For each frame, we first find all of the normal clusters that are backfacing with respect to the current viewing direction. A normal cluster is considered backfacing only when all of the normals it contains are backfacing. In practice, we approximate this requirement by testing only the four corner normals of the cluster. If they are all backfacing, then the cluster is marked as backfacing. The backfacing clusters are recorded in a *backface mask,* which has as many physical bits as the number of clusters, by setting their corresponding bits to one. In other words, the backface mask is the OR of all the backfacing clusters' normal masks. Clearly, if a polygon's normal belongs to a backfacing normal cluster, the polygon is backfacing. Let the backface mask be a byte array `BackMask[]` and let the two-byte normal representation for a polygon be (`byteOffset`, `bitMask`), and then the backface test is simply a logical AND operation:

```
BackMask[byteOffset] & bitMask
```

3.1 Determination of Backfacing Normals

First we discuss how to decide whether a normal points away from the viewer regardless of the viewer's position. In the parallel projection case, this does not present a problem since we can easily cull away the backfacing 180 degrees; however, in the perspective case, the amount we can cull is very dependent on both direction and position. In fact, in the perspective case, we can often cull away more than 180 degrees. To be able to find backfacing normals without positional information, it is necessary to allow ourselves to be somewhat conservative in our culling; we must find backfacing normals that will remain backfacing regardless of viewer position (or polygon translation). We will explain this in two dimensions, but this extends to the three dimensional generalization in a similar manner.

Again in a parallel projection, the normal and the viewing direction are sufficient to cull away polygons with the backfacing 180 degrees of normals. For perspective projection, the conclusion is that if the field-of-view is FOV degrees, we can safely identify 180-FOV degrees around the viewing direction to be backfacing, *regardless of the polygon's spatial position*. To see this is the case, it helps to think of a perspective projection as a "cone" of parallel projections whose directions are distributed throughout the field-of-view. A normal is backfacing for the perspective projection only when it is backfacing for all of its "corresponding" parallel projections.

As a summary, backfacing normal determination is illustrated in the following figure:

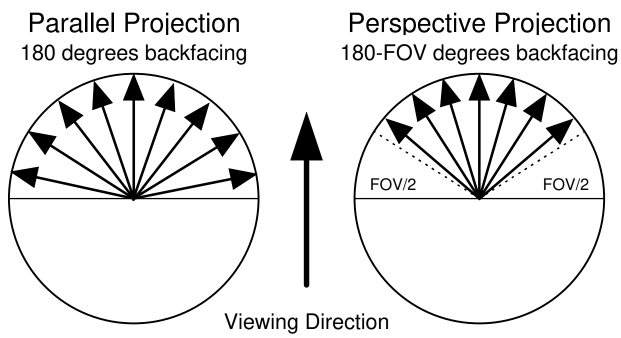


Figure 2: The areas marked with arrows indicate the backfacing normal direction region with respect to a given viewing direction. If a normal cluster is completely inside the backfacing region, then it is marked as a backfacing cluster.

3.2 Improving the Rate of Culling

From the discussion above, clearly the amount we can cull depends largely on the field-of-view. Simply put, a larger field-of-view results in a lower culling rate. So, in order to improve the culling rate, we can subdivide the model spatially into smaller groups and form a viewing volume to each group which subtends a much smaller angle than the total field-of-view. In so doing, we make the culling rate and the total field-of-view (as one would set in a OpenGL call) independent. We can do this rather conveniently by utilizing any existing bounding-volume hierarchies,

Now for each frame, we form a viewing volume (a viewing cone actually) to each object group and calculate its field-of-view, compute a backface mask for each, and cull the corresponding polygons. The normal masks are calculated exactly as before, but now we have a much tighter range of possible projection directions, thereby increasing the culling in each group. Clearly there is a tradeoff between the number of viewing volumes and the percentage of culling since we must now compute backface masks for each group, and there will be a point beyond which the expense of finding backfacing clusters exceeds the advantage of increased culling.

4 IMPLEMENTATION AND RESULTS

The implementation requirements are quite modest; we implemented the algorithm in around 80 lines of C code. The interface need only consist of two functions and one macro: a function for encoding a normal (`Encode(N)`), a function for determining backfacing clusters (`BuildBackfaceMask (VDIR,FOV)`), and a macro for the fast backface test (`ISBACKFACING(NMASK)`). Given a polygon's normal, Encode simply returns a corresponding normal mask to be stored with the polygon. `BuildBackfaceMask` fills the bits in the `BackMask` byte array corresponding to the current backfacing clusters, given the viewing direction and a field-of-view. Finally, the `ISBACKFACING` macro performs the previously described backface test consisting of a logical AND operation between the normal mask and the backface mask.

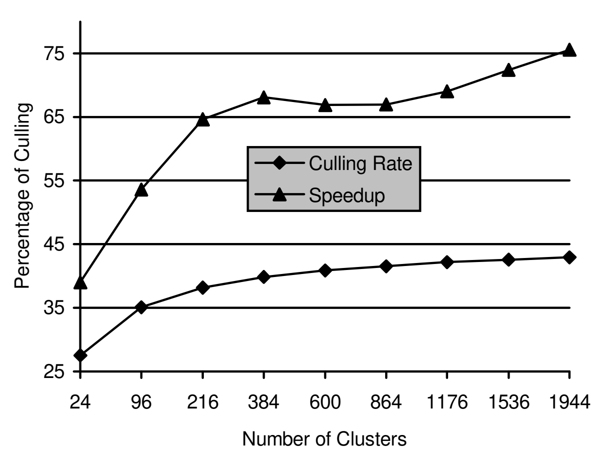
We tested the algorithm for a scene consisting of two copies of the 70,000 polygon bunny model. To increase the culling rate, we used the bounding spheres of the bunnies to form the viewing cones. The results are encouraging: we generally obtain over 40% culling and 80% speedup on an SGI Indigo2 Maximum Impact. Hardware backface culling is used in all of the performance tests.

We did performance tests for different resolutions of the normal space subdivision. The scene is made up of two copies of a bunny model, each of which has a bounding sphere. We compute the angles subtended by these spheres as field-of-views for backface culling. The results are summarized below:

Table 1: Performance Statistics

# Normal Clusters	# Front-facing Polys	# Back-facing Polys	% Polys Culled	Frame Time (ms/Frame)
24	100749	38152	27.5	469.5
96	90137	48764	35.1	425.1
216	85867	53034	38.2	396.3
384	83462	55439	39.9	388.3
600	82073	56828	40.9	391.3
864	81129	57772	41.6	393.8
1176	80299	58602	42.2	386.2
1536	79665	59236	42.6	378.6
1944	79135	59766	43.0	371.9

Graph 1: Percentage of Culling and Speedup



5 DISCUSSION

[KUMA96] achieves sublinear speedup by discarding groups of polygons as a whole instead of visiting each polygon. An important assumption for this type of method is that polygons can be reorganized into groups according to their normals — i.e., one can reshuffle the database of polygons — without any negative effect on rendering. In practice, this is often not the case because there are other conflicting criteria for polygon grouping. For example, polygons sharing a texture are often grouped together because setting the current texture for each polygon individually can be an expensive operation. Also, visual simulation applications require polygons to be grouped into a spatial hierarchy. One can always reconcile two grouping criteria into one application, but data structures become quite complex and sublinear performance can be lost.

Our method is easily integrable into existing rendering systems since it performs culling operations per polygon, and thus does not impose structural requirements on the database. The strength of this method comes from the fact that even if the backface face test is done per polygon, the cost is still trivial.

Since our backface culling method is currently performed on the host CPU, we are limited to immediate mode graphics; however, it is readily applicable for use in graphics hardware. An implementation of this method has been planned for the geometry processors of PixelFlow [MOLN92]; so, when in retained mode, the hardware will perform our backface test when rendering a display list, while requiring only two bytes of extra storage per polygon in the display list.

Currently, our method does not support dynamic scenes in which an object's normals are allowed to change throughout the course of interaction. This is a common problem for any static database preprocessing method, and we do not address it here further.

6 CONCLUSION

We have devised a faster method for backface culling which reduces the backface test to one logical AND operation per polygon, while requiring only two bytes extra storage per polygon. The *normal mask* is introduced as an efficient bitmask encoding scheme for the polygon's normals and for the clusters in the normal space partitioning. We have shown that although conceptually the bitmasks may require a great deal of storage, they actually only require the two bytes per polygon. The preprocessing stage simply involves normal cluster classification of all polygons by their normals. The per-frame overhead in finding backfacing clusters and building the backface mask is very low. Our technique requires little or no modification to existing data structures and does not affect existing object hierarchies. In short, our system is elegant, simple, and easy to integrate into existing complex, high-performance graphics systems and hardware.

Acknowledgments

Special thanks go to Sarah Hoff for extensive editing support. This work is supported in part by ARO Contract P34982-MA, NSF Grant CCR-9319957, NSF Grant CCR-9625217, ONR Contract N00014-94-1-0738, ARPA Contract DABT63-93-C-0048 and NSF/ARPA Center for Computer Graphics and Scientific Visualization

References

[KUMA96] S. Kumar, D. Manocha, B. Garrett, and M. Lin. Hierarchical Back-face Culling. In 7th Eurographics Workshop on Rendering. pages 231-240, 1996.

[MOLN92] S. Molnar, J. Eyles, and J. Poulton. PixelFlow: High Speed Rendering Using Image Composition. In SIGGRAPH 92, pages 231-240, 1992.

[SHIR93] 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.