Visibility-Based Simplification of Objects in 3D Scenes

Jérôme Grasset
3IL Ecole d'Ingénieurs
43 rue Ste Anne
87000 Limoges, France
grasset@3il.fr

Dimitri Pléménos
Laboratoire MSI, Université de Limoges
83 rue d'Isle
87000 Limoges, France
plemenos@unilim.fr

ABSTRACT

We present a method to perform occlusion culling on moving rigid objects in a 3D scene. This method computes potentially visible sets of polygons (PVS) from cells that are based on a bounding box of the object and have the particularity to be overlapping. The visibility is pre-processed from several renderings of the object. The implementation is easy, with a data structure that does not require additional memory compared to the original data used to describe the object. It provides interesting acceleration and smooth animation even when the list of polygons sent to the display device has to be updated. Since it is not an exact method, we discuss the errors that may occur and the ways to fix them.

Keywords
Visibility culling, Occlusion culling, Simplification, Bounding Box.

1. Introduction

The visualization of 3D scenes made of polygons uses "hidden surface removal" algorithms, like the Z-Buffer method that is integrated in common graphics cards. But, even if these hardware Z-Buffers are very fast, it is still interesting to decrease the number of polygons they have to process by determining that some are "obviously" invisible: that is the point of visibility culling. It is composed of three phases :

- view-frustum culling, to reject the objects which are outside the view frustum
- back-face culling to remove the polygons that do not face the camera , for closed opaque objects
- occlusion culling to suppress polygons that are hidden behind others.

Short papers proceedings, ISBN 80-903100-9-5
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

The first two steps are very easy to implement, the last one is much more difficult. The approach is usually to do some preprocessing to determine which polygons are potentially visible from different subspaces of the viewpoints space. Then, when the camera is in a subspace, only the potentially visible polygons associated with this subspace are sent to the display device. Subspaces are called "cells" and polygons are grouped into "Potentially Visible Sets" (PVS), a PVS being associated with each cell.

Various classifications of visibility culling methods are described in the survey by [Coh03a]. We will use the following terminology, which is slightly different but more simple and sufficient for our needs:

- **exact** method : a PVS contains all the visible elements and none of the invisible ones
- **conservative** method : a PVS contains all the visible elements, but maybe some invisible ones too
- **aggressive** method : a PVS may lack some visible elements, but it does not include any invisible one
- **approximate** method : a PVS may lack some visible elements and may include invisible ones.

2. Previous Works

One can refer to [Coh03a] for a detailed survey on occlusion culling, or [Dur00a] for the more general scope of visibility computation.

Theory – Exact Methods

The exact methods use the concept of "visual events". This is the base of aspect graphs ([Egg92a]), visibility skeleton ([Dur02a]), or the method proposed by [Nir02a].
Exact methods are, in theory, the ideal. Using their implementation experience, authors that proposed such methods explain that they are difficult to implement and require long computation time and large data structures ([Nir02a], [Dur02a]).

Non-Exact Methods

The non-exact methods come from the ideal frame and trade some precision against computation speed and implementation easiness. The usual approach here is to define the cells first and then to compute the visibility from each one. The visibility is not constant in each cell: the problem is to calculate all the polygons that can be seen from at least one viewpoint of the cell.
The first algorithms ([Tel91a], [Lue95a]), were made for architectural scenes. More recent works address general scenes. A major direction of research is the detection of good "occluders", that are polygons or sets of polygons that hide numerous other polygons (see for example [Sch00a], [Won00a]).

Application Fields

The occlusion culling methods are mainly dedicated to virtual walkthrough applications. Some properties of occlusion culling methods are to be mentioned to help situate the differences with our work:

- the global scene is static, moving objects - like cars in the streets - are not simplified
- objects that are inside a cell are entirely in the PVS of the cell
- the scene is very complex but it also has to be "highly occluded" : from the common viewpoints most of the polygons can not be seen.

3. Visibility Through Bounding Box

Differences From Existing Methods

Our method is not dedicated to simplify large complex models like a whole town, and it does not fit for such a goal. We want to simplify moving objects (for example cars), or "small" objects around which the virtual walker can move. The simplification is pre-processed for each object individually and is stored with it.
Another difference is the viewpoints space: our 3D objects are seen from the outside while the scenes of previous works are seen from their interiors.
The objects that we consider are not "highly occluded" so the simplification ratio is very different.

Visibility Pre-processing

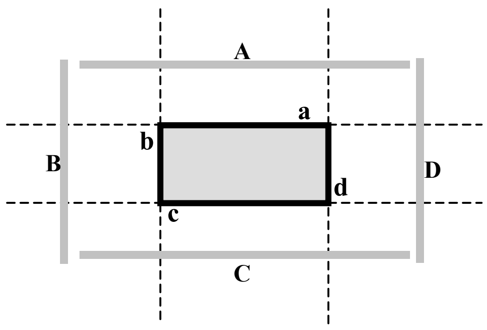

Figure 1. 2D schema of 4 half spaces (A,B,C,D) around a bounding rectangle

The viewpoints space is the whole space minus the interior of the bounding box. The cells are 6 infinite half spaces defined by the plane of each face of a bounding box of the object (see figure 1 for a 2D schema). Considering a mathematical approach, these cells do not define a partition since they overlap.
For each cell the bounding box is supposed closed except for its face lying in the plane defining the cell (see figure 2).

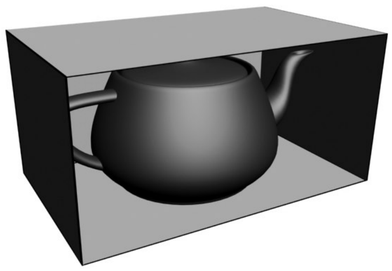

Figure 2. Bounding box with a unique open face

The first step is to compute which polygons of the object are visible through each face of the bounding box. This defines our PVS. Then, to display a scene we will only consider the polygons of the PVS associated with every face of the box that are visible from the current viewpoint. With figure 1, if the observer can see the faces 'a' and 'd' then the union of the PVS associated with areas A and B have to be displayed.

The viewpoints space is sampled. That is why our method is “aggressive”: the PVS do not contain any invisible polygons, but they may miss visible ones.

For each face of the bounding box, the sampling viewpoints are disposed on the surface of a half ellipsoid. If all the viewpoints of this surface could be taken into account the PVS would be exact: there is no need to move the camera further or closer.

To pre-process the visibility for each face we use the hardware Z-Buffer. Every polygon is given a unique colour and an image is rendered from each sample viewpoint: the colours in the image denote the visible polygons. The PVS of a half space is made of the polygons detected in at least one image.

Data Structure and Display

Since the cells overlap, if we just sent “raw” PVS to the display device there would be redundancies for all the polygons visible through several faces. To avoid this we define an intermediate data structure that insures that a polygon can only be sent once to the display device for a given frame.

The idea is to store for each polygon the list of PVS it belongs to, instead of storing for each PVS the list of polygons it includes. There are 6 half spaces, so 6 PVS. Using a binary code we can associate each polygon with a 6 bits number, where a bit indicates if a polygon belongs to a PVS (see table 1). Every polygon is given a code, and there are 2^6 different codes: an easy way to store this information is to create a set of polygons for each of these codes. Then a polygon belongs to one an only one set. The storage of the pre-processed visibility is only a classification of the polygons: no extra data is needed.

Faces through which a polygon is visible	Label of the set it belongs to (6 bits)
6 faces	
0,1,2,3,4,5	111111
5 faces	
0,1,2,3,4	011111
0,1,2,3,5	101111
….	
No visible face	
/	000000

Table 1. Classification of the polygons

To display the object from any viewpoint, we first create the bit mask of the visible faces of the bounding box, using the same coding as for the PVS. Then the obtained mask is applied with a logical “and” to the label of every set of polygons: if the test succeeds the set has to be displayed.

We implemented this technique with the OpenGL display lists to represent the sets of polygons: the storage cost is reduced to the use of 64 display lists. The same has been implemented with Direct3D vertex buffers, with the same negligible cost.

4. Results

Overall Performance and Motion Fluidity

The time needed to compute which faces of the bounding box of an object are visible is definitely negligible (6 tests on the normals of these faces), so the gain in frames per second (FPS) is the same as the simplification ratio in polygons count. That is why in the following table we present only the FPS values.

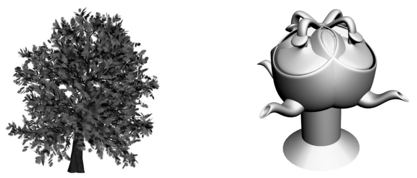

Figure 3. Two objects : a tree and a “tea fountain”

Since the set of polygons sent to the display device has to be changed when the observer crosses a half-space border one could worry about potential freezing in animation. With the proposed data structure and implementation there is no delay at all, the motion is smooth. Of course if a much higher number of polygons has to be displayed the motion may slow down, but there is no extra latency induced by the change of active display lists or vertex buffers.

Pre-processing Time

In our implementation, the precomputation time depends on the size of the images rendered to pre-process the visibility because most of the time is spent reading every pixel to know which polygons have been seen. It varies from about 1 min 15 s when using 9 viewpoints by face and a resolution of 800 x 600 to about 14 min with 25 viewpoints in 1280 x 1024.

The test material is a Pentium IV 2.8Ghz with an ATI Radeon 9600 Pro.

Culling and Errors

For the results presented here we use a resolution of 1280x1024 for visibility pre-processing and 25 viewpoints for each face of the bounding box. The display resolution is 1024x768.

4.3.1 Culling results

The table 2 gives the the worst FPS, the best one, and the one obtained when the observer is in front of the object. The percentages indicate the gain in FPS and therefore the amount of polygons that were discarded.

	without culling		with culling FPS / gain	
Object	**Faces**	**FPS**	**Worst**	**Best**
Tree	49534	72	85 - *15.2%*	117 - *38.5%*
Cup	11290	273	290 - *5.8%*	520 - *47.5%*
Ship	31216	112	145 - *22.7%*	200 - *44%*
Character	18984	175	222 - *21.1%*	455 - *61.5%*
Fountain	78984	47	77 - *39%*	300 - *84.3%*

Table 2. Culling results

Back face culling is always off, since it can not be applied to most of our objects (incompatible for example with the leafs of the tree or with the teapots).

The culling ratio directly depends on the number of faces of the bounding box that are visible. As a consequence the bounding box should be oriented to show a number of faces as small as possible from the most common viewpoints. For instance, for an object usually on the floor the bounding box should have a face on the floor.

4.3.2 Errors

Since our method is aggressive (see section 1), it is important to study the errors it makes.

First of all, with the parameters given above, most of the objects do not exhibit any error.

The first cause of errors is the sampling of the viewpoints space. When some polygons can only be seen from a very restricted area ("critical area") they are missed if the viewpoints are not in this area.

The second sampling is made when the visibility is preprocessed: we render images to detect visible polygons. If several polygons are in the same pixel only one is rendered, and thus only one is detected.

These two types of errors can be significantly reduced or suppressed, by increasing the appropriate sampling rate.

5. Conclusion and Further Works

We presented a method to perform occlusion culling on rigid 3D objects moving in a scene. Existing methods mainly focus on static scenes and do not deal with objects inside them. Our method, based on a preprocessing of visibility through a bounding box using Z-Buffer techniques, is easy to implement. It does not increase the amount of memory used by the objects since the visibility data are stored as a classification of the polygons in various display lists. The obtained culling is quite efficient. This method can be applied in any application that needs a 3D scene visualization. It can also be a complement of a method of occlusion culling of the whole scene.

Further works should aim at reducing preprocessing time and maintain the culling rate as close as possible as the best one, obtained when only one face of the bounding box is visible. Furthermore, some other particular problems should be addressed, like objects with transparent parts.

6. References

[Coh03a] Cohen-Or, D.; Chrysanthou, Y.; Silva, C.; Durand, F. A survey of visibility for walkthrough applications. IEEE Transactions on Visualization and Computer Graphics 2003

[Dur00a] Durand, F. A multidisciplinary survey of visibility. ACM Siggraph course notes Visibility, Problems, Techniques, and Applications 2000,

[Dur02a] Durand, F.; Drettakis, G.; Puech, C. The 3D visibility complex. ACM Trans. Graph. 2002, 21, 176-206

[Egg92a] Eggert, D. W. ; Bowyer D.W.; Dyer C.R Aspect graphs: State-of-the-art and applications in digital photogrammetry. In Proc.SPRS 17th Cong.: Int. Archives Photogrammetry Remote Sensing, pp 633–645, 1992.

[Lue95a] Luebke, D.; Georges, C. Portals and mirrors: simple, fast evaluation of potentially visible sets. Proceedings of the 1995 symposium on Interactive 3D graphics 1995, 105-ff.

[Nir02a] Nirenstein, S.; Blake, E.; Gain, J. Exact from-region visibility culling. Proceedings of the 13th Eurographics workshop on Rendering 2002, 191-202

[Sch00a] Schaufler, G.; Dorsey, J.; Decoret, X.; Sillion, F. Conservative volumetric visibility with occluder fusion; ACM Press/Addison-Wesley Publishing Co.: 2000; pp 229-238

[Tel91a] Teller, S. J.; Séquin, C. H. Visibility preprocessing for interactive walkthroughs. Proceedings of the 18th annual conference on Computer graphics and interactive techniques 1991, 61-70

[Wil03a] Williams, N.; Luebke, D.; Cohen, J. D.; Kelley, M.; Schubert, B. Perceptually guided simplification of lit, textured meshes; ACM Press: 2003; pp 113-121

[Won00a] Wonka P; Wimmer M; Schmalstieg D Visibility preprocessing with occluder fusion for urban walkthroughs. Rendering Techniques 2000: 11th Eurographics Workshop on Rendering, pages 71–82, June 2000.