Occlusion Culling Methods

Heinrich Hey Werner Purgathofer

Institute of Computer Graphics and Algorithms
Vienna University of Technology
{hey,wp}@cg.tuwien.ac.at

Abstract

Occlusion culling methods are an important tool for efficient rendering of large scenes. Usually only a small part of such a scene is visible from a specific viewpoint. Therefore occlusion culling methods try to determine which parts of the scene are invisible and can be culled. That way usually only a small part of the scene´s primitives has to be drawn. This is especially important for real-time rendering because large scenes can easily contain much more primitives than available hardware can render in real-time. In this state of the art report we present an overview of the already large number of existing occlusion culling methods and their different approaches and characteristics.

1. Introduction

A large scene, eg. the interior of a building or even a whole city, may contain several millions of polygons, but usually only a small part of such a scene is actually visible. Therefore it would be inefficient to simply draw all the geometry of this scene, because most of the rendering time would be spent into drawing invisible objects.

This is especially a problem in real-time rendering, since available graphics hardware can not render such a large number of primitives at interactive frame-rates. Although graphics hardware is continuously becoming faster, it will probably never be fast enough, because the scenes are also becoming more complex.

The number of rendered primitives can be reduced to some extent by usage of hierarchical view frustum culling and back face culling. View frustum culling [5, 12, 36] determines which parts of the scene are outside of the viewing frustum. These objects are definitely invisible and can therfore be culled.

In large scenes view frustum culling is usually applied on a spatial subdivision structure or on a hierarchy of bounding volumes of the scene, which avoids that every single primitive has to be tested separately.

Back face culling [36] determines which polygons are facing away from the viewer. These polygons are invisible and can be culled. Back face culling can also be done hierarchically [33] so that not every single polygon has to be tested separately.

Nevertheless, only with hierarchical view frustum culling and back face culling in many scenes the number of primitives that would have to be drawn would still be too high.

This problem can be solved by usage of occlusion culling methods which try to determine those parts of the scene

that are invisible due to occlusion by other parts. Only those parts of the scene which are potentially visible have to be rendered. Parts which are identified as occluded are culled so that they do not have to be processed further.

Many occlusion culling methods use a hierarchical representation of the scene. This allows that the occlusion testing and culling can be done for large parts of the scene at once without having to test all their sub-parts individually.

Eg. if we already know that a building in a city is occluded from the current viewpoint then we do not have to test the occlusion of each object inside the building, because these objects are of course also occluded.

Exact global visibility methods try to solve the visibility problem by determining all visibility events in the scene for all possible viewpoints [16, 17, 40], but due to their complexity they are impractical for large scenes.

The high expense of exact global visibility calculations can be avoided by overestimating the set of visible objects. Most occlusion culling methods do not solve exact visibility. They return objects which are potentially visible, which includes not only those primitives that are completely visible, but also objects that are only partially visible and maybe even some objects that are completely invisible.

The exact visibility of these potentially visible objects is then calculated with an additional visibility technique. In most cases the z-buffer of conventional graphics hardware is used for the exact visibility determination.

Nowadays exist a large number of solutions that realize occlusion culling in different ways. These occlusion culling methods can be categorized according to several different criteria. In the following sections we describe the most important of these characteristics to give an overview of occlusion culling methods for real-time rendering.

2. Visibility from region or from viewpoint

Occlusion culling methods have to determine the set of objects that are potentially visible (potentially visible set (PVS)) from the current viewpoint. This can be done for this single viewpoint alone (see figure 1), but it can also be done for a entire region (cell) in space [3] (see figure 2).

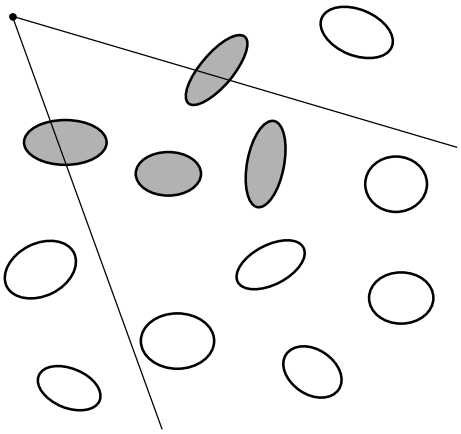

Figure 1: *From-point visibility: the gray objects are potentially visible from the current viewpoint, the white objects are occluded.*

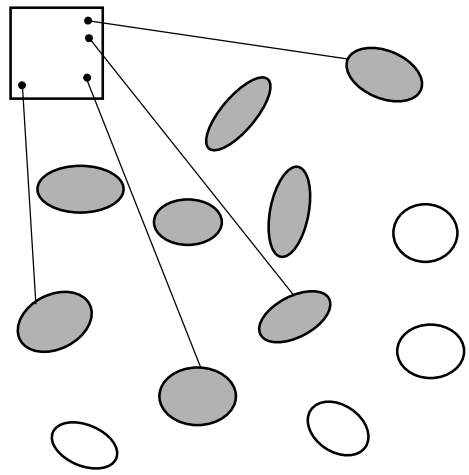

Figure 2: *From-region visibility: the gray objects are potentially visible from at least one viewpoint within the square cell, the white objects are occluded.*

In the latter case the generated potentially visible set consists of all those objects which are completely or partially visible from at least one viewpoint in the region. From-region visibility methods use the coherence of visibility of the viewpoints in a region. They also distribute the computational expense of the visibility calculations over all the viewpoints in the region.

On the other hand the potentially visible set that is returned from a from-point visibility method can be

noticeably smaller than the one from a from-region visibility method, because the visibility has to be calculated only for a single viewpoint.

3. Visibility calculations in a preprocessing step or on the fly

Occlusion culling methods can do their visibility calculations in a preprocessing step that precedes the rendering of the images, or it can be done on-the-fly during rendering.

Methods like eg. the technique by Law et al. [34], the visibility octree method [41], or the technique by Wang et al. [51], which use a preprocessing step for their visibility calculations, subdivide the static scene into cells. In the preprocessing step these methods calculate the potentially visible set of each of the cells.

These methods are therefore from-region visibility methods, and the potentially visible set of a cell consists of all those objects which are completely or partially visible from at least one viewpoint in the cell.

During the rendering-phase these methods only have to render the objects from the potentially visible set of the cell in which the current viewpoint is located. Therefore these methods have the advantage that their rendering-phase is usually very fast because the potentially visible objects can be rendered without any further occlusion culling overhead.

But the visibility preprocessing also has some disadvantages:

- The visibility precomputation usually requires between several minutes and several hours depending on the complexity of the scene. This makes it impossible to immediately render a scene after it has been modified.

- The memory requirements for the precomputed visibility information can easily become very large. Compression techniques [50] can be used to minimize this memory consumtion.

- Only static objects can be used as occluders for the generation of the potentially visible sets in the visibility preprocessing step. Occlusion by dynamic objects is not taken into consideration in the precomputed visibility information.

Visibility preprocessing methods are often used in games [1], because the frame-rate of the released product is the major criterion, and the cost of the time-expensive visibility precomputation after modeling is only secondary.

The typical scenes of these games consist of a large static environment and several dynamic objects which are relatively small. Usually the static environment can be designed in a way that only a very small portion of the scene is visible from each possible viewpoint. Due to their small size in the image the dynamic objects usually do not have to be considered as important occluders. Therefore the visibility preprocessing works quite well.

Most occlusion culling methods that do their visibility calculations on the fly during rendering have the following advatages:

- No time-expensive visibility precomputation is needed. This makes these methods suitable for applications where the scene has to be instantly displayed after it has been modified, eg. during modeling or for scenes that can be interactively manipulated by the user.

- Dynamic objects can be used as occluders. Apart from using temporal coherence (see section 13) on-the-fly occlusion culling methods do not have to distinguish between static and dynamic objects, because visibility is computed for the entire actual scene at the given point in time of the image.

Exceptions are occlusion culling techniques like virtual occluders [32] or directional discretized occluders [8] which do a part of their visibility calculations on the fly during rendering, but which additionally also use a preprocessing step to generate an intermediate visibility information.

Hierarchical occlusion maps [58] also use a preprocessing step. They need it to generate a database of potential occluders. This occluder precomputation can be avoided with incremental occluder selection as it is used for incremental occlusion maps [2], or by combining hierarchical occlusion maps with frustum slicing [10].

The disadvantage of all occlusion culling methods which are done on the fly is that their visibility calculations produce some overhead during rendering.

4. Visibility calculations in object space or image space

Occlusion culling methods can do their visibility calculations in object space or in image space. Image space methods do their visibility calculations typically on-the-fly during rendering.

Several image-based occlusion culling techniques use a hierarchical representation of their occlusion information in the image [22, 23, 24, 26, 27, 58]. In figure 3 this is shown for a hierarchical occlusion map [58] as an example. The hierarchical occlusion map consists of a mipmap pyramid of grayscale images. The intensity of a pixel in this pyramid represents the percentage of underlying pixels that are occluded in the full resolution image.

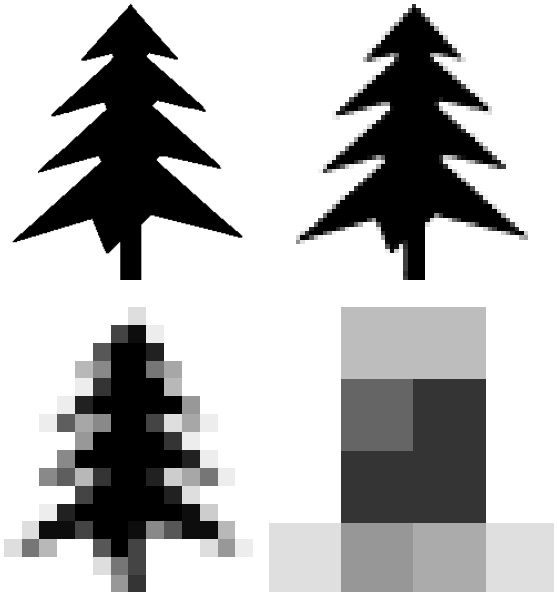

Figure 3: *The hierarchical occlusion map represents the occlusion in the image hierarchically at several resolutions.*

These hierarchical representations allow efficient culling in large occluded image areas, because occluded objects can be culled with a few accesses to entries in the high levels of the occlusion information hierarchy instead of having to access a much larger number of low level entries.

5. Continuous or point sampled visibility

Continuous visibility methods determine the visibility in all view directions that pass through the image, which is an infinitely large set of view directions. In contrast to that are point sampled visibility methods which determine the visibility only for a limited set of view directions, eg. for the centers of all pixels in the image. Point sampling can also be used for object space occlusion culling [21].

6. Conservatism of visibility

Most occlusion culling methods return conservative visibility information. This means that their returned set of potentially visible objects contains at least all objects that are completely or partially visible.

Several methods support non-conservative occlusion culling, which means that they do not guarantee that their returned potentially visible set contains all objects that are completely or partially visible. This increases the performance, but of course it causes artifacts in the image.

Non-conservative occlusion culling can be done by rendering only those objects which are most likely to be visible [30], by using stochastic point sampled visibility in object space [21], by testing only a few of the pixels of a bounding volume [7], or by culling objects which are visible only in a few pixels [58].

7. Hardware acceleration

Especially for image space occlusion culling methods it is very important whether they support some kind of hardware acceleration to enhance the performance of their visibility calculations.

Hierarchical occlusion maps [58] read the frame buffer from the graphics hardware and use it to generate a pyramid of occlusion maps. They also use the mipmap functionality of the graphics hardware to generate the downsampled versions of the occlusion map.

An opacity map [27] can be used instead of a hierarchical occlusion map. Whereas the the hierarchical occlusion map resembles a pyramid of mipmaped textures, the opacity map corresponds to a summed area table which

allows to perform an overlap test in constant time. The generation of the summed area table has to be done in software.

The hierarchical z-buffer [22] requires a specialized graphics hardware for full efficiency. A variant of the hierarchical z-buffer for parallel architectures has been implemented for Pixel-Planes 5 [20]. An optimized version of the hierarchical z-buffer has been proposed [24] that allows to integrate a hierarchical z-buffer stage into the rendering pipeline of conventional graphics hardware.

Adaptive hierarchical visibility [55] is a simplified one layer version of the hierarchical z-buffer where bucket sorted polygon bins are rendered and occlusion tested. It is simpler to implement in graphics hardware than the hierarchical z-buffer.

Hierarchical coverage masks [23] are very efficient in comparison to a software implementation of a conventional scanline rasterizer, but unfortunately hardware acceleration is very limited because conventional graphics hardware can only be used for texturing and shading.

The lazy occlusion grid [26] uses currently available graphics hardware to update the occlusion information that is stored in this grid. Due to a lazy update scheme of this occlusion information the number of accesses to the hardware z-buffer is usually even lower than the number of pixels in the image. Nevertheless it is a conservative method.

Occlusion queries are currently implemented in graphics hardware on a few systems, eg. on Hewlett-Packard's Visualize fx series of graphics accelerators [28, 44, 45], on Silicon Graphics Visual Workstations [11], and on Nvidia's GeForce3 [39].

These occlusion queries test the occlusion of a given bounding volume by rasterizing it without modifying any buffer. The occlusion query then returns whether the bounding volume passed the z-test (is visible) in any of its pixels. The relative cost of such an occlusion query in comparison to the cost of drawing triangles varies between different hardware [45].

Such an occlusion query is used eg. in the conservative prioritized-layered projection algorithm [31] to cull occluded cells from the set of cells that are candidates for projection. Hardware accelerated occlusion queries could also be extended to return additional occlusion information and to work in parallel [6].

A similar occlusion query can also be implemented on systems where such an occlusion query is not implemented in the graphics hardware. This can be done by usage of the stencil buffer [7]. The write access to the color buffer and the z-buffer is disabled, and then the bounding volume is drawn. For each pixel a bit in the stencil buffer is set to 0 or 1 corresponding to whether the bounding volume passed the z-test in that pixel or not.

After that the pixels in the bounding volume's image area must be read from the stencil buffer, and it must be tested in software if their stencil bits are 0 or 1. Reading the stencil buffer and testing it in software is of course a significant performance bottleneck.

8. Selection of occluders

Occlusion culling methods can use all objects as occluders, or they can select a set of objects and use only these objects as occluders. Using all objects as occluders has the advantage that it maximizes the occlusion, but several occlusion culling methods require to use a selected set of occluders.

Such methods [9, 14, 15, 29, 58] select the occluders heuristically based on the assumption that these objects occlude large parts of the scene. In cases where these heuristics do not work will large parts of the scene remain unoccluded.

Additionally simplified representations of the occluders can be synthesized, eg. by using sets of boxes that are enclosed by the original geometry of the occluders [4]. This allows to increase the performance of visibility tests.

9. Occluder fusion

Not only the number and distribution of the occluders is important, but also the ability of the occlusion culling method to support occluder fusion. This means that several (small) occluders together can occlude parts of the scene that the single occluders alone would not occlude if they were used independently of each other, which is illustrated in figure 4 and 5.

Occluder fusion is supported eg. by the directional discretized occluders method [8], by the extended projections method [18], by the conservative volumetric visibility method [42], or in the visibility preprocessing method for urban walkthroughs by Wonka et al. [54].

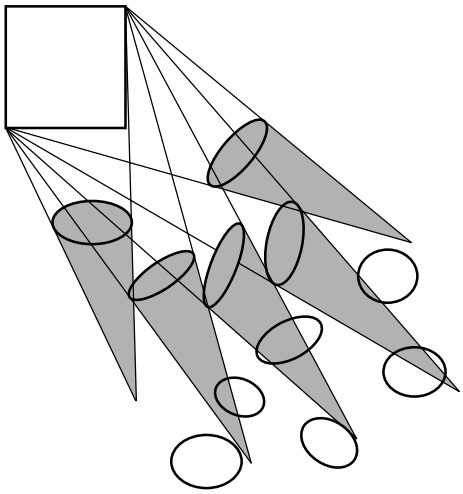

Figure 4: *The single occluders (gray) occlude only a small region (gray) of the scene if they are used independently of each other.*

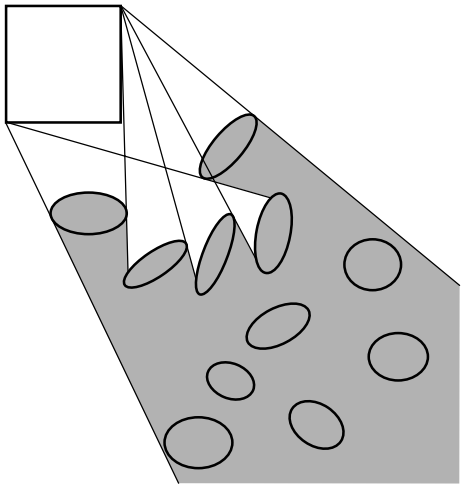

Figure 5: *Together the same set of occluders as in figure 4 occludes a considerably larger region (gray) of the scene.*

Point sampled image space occlusion culling methods implicitly support occluder fusion [22, 23, 26, 27, 44, 58], because in their image space occlusion information they do not distinguish between different occluders. Therefore the occluders are automatically combined without having to do additional computations for the occluder fusion.

Occluder fusion is very important for occluders like trees, because each single leaf of a tree usually occludes only very few objects, if any, behind it. But all the leafs of the tree together can represent an important occluder that occludes many objects behind it.

Occlusion culling methods which do not support occluder fusion can usually only be used efficiently in restricted scenes which contain objects that are large enough to represent strong occluders [13].

10. Supported scenes

Although it is desirable to support general scenes, many occlusion culling methods are nevertheless restricted to certain types of scenes. Visibility precomputation methods are restricted to mainly static scenes. Occlusion culling methods which use portals for their visibility calculations [3, 35, 49] usually require architectural environments.

Several methods are restricted to terrains [46, 57] which are usually based on a height field, and several other methods are restricted to walkthroughs in urban environments [52] or to 2½D scenes [43, 53] which are modeled on a ground plan.

But of course also the (in)ability of a method to use all objects as occluders and to support occluder fusion decides whether the method is suitable for general scenes or not.

11. Traversal of the scene

Many occlusion culling methods require that the scene is traversed in a front to back order to make efficient occlusion culling possible. This means that objects which are nearer to the viewpoint are processed before objects that are farther away, so that the nearer objects can occlude the objects behind them.

It is also important to distinguish whether the method requires a certain front to back traversal of the scene, or if the scene can be traversed in any approximative front to back order.

12. Supported bounding volumes/spatial subdivision structure

Several occlusion culling methods require that a certain kind of bounding volumes or spatial subdivision structure is used, eg. a regular spatial grid [56]. Several methods use a BSP-tree [19] as spatial subdivision structure in object space [9, 15, 38], or also to subdivide the image [37].

The hierarchical z-buffer [22] has been proposed in combination with an object space octree, but in principle it can also be used with other space subdivision structures or with hierarchies of bounding volumes.

13. Temporal coherence

The successive frames of a walkthrough or an animation usually have a high temporal coherence. In the context of occlusion culling this means that it is likely that objects which have been visible/hidden in the previous frame will still be visible/hidden in the current frame.

Some occlusion culling methods utilize temporal coherence [25] to enhance their efficiency. This can be done eg. by caching the objects' occlusion relations of the previous frame and reusing them in the current frame [14].

The hierarchical z-buffer [22] can be initialized by rendering all visible objects of the previous frame before the hierarchical z-buffer is used for visibility testing. After this initialization the hierarchical z-buffer will usually contain most of the visible objects of the current frame.

Temporal coherence can also be utilized by usage of temporal bounding volumes [47, 48]. A temporal bounding volume encloses a dynamic object not only at a single point in time. Instead it encloses the object at every position that the object has during a time interval.

14. Conclusion

In this state of the art report we have presented an overview of existing occlusion culling methods for real time rendering. We have described the major characteristics that allow to distinguish these methods, and that allow to select appropriate methods for a given application.

It is likely that hardware support for occlusion culling methods will increase and will become more popular on graphics hardware in the near future. Of course this will make occlusion culling even more interesting for real-time applications.

The widespread availability of fast hardware accelerated occlusion queries will possibly also make the majority of other occlusion culling methods obsolete, similar as the z-buffer has superseded most of the other visibility techniques.

Acknowledgments

This work has been supported by the Austrian Science Fund (FWF) project no. P13600-INF.

References

1. Michael Abrash. Inside Quake: Visible Surface Determination. Ramblings in Real Time. Dr. Dobb's Sourcebook January/February 1996 pp. 41-45

2. Timo Aila and Ville Miettinen. dPVS Reference Manual. Hybrid Holding, Ltd. www.renderware.com/dpvs.html, 2001

3. John M. Airey, John H. Rohlf and Frederick P. Brooks Jr. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. Symposium on Interactive 3D Graphics 1990 pp. 41-50

4. Carlos Andújar, Carlos Sanoa-Vázquez and Isabel Navazo. LOD Visibility Culling and Occluder Synthesis. Computer-Aided Design vol. 32 no. 13 2000 pp. 773- 783

5. Ulf Assarsson and Tomas Möller. Optimized View Frustum Culling Algorithms for Bounding Boxes. Journal of Graphics Tools vol. 5 no. 1 pp. 9-22, 2000

6. Dirk Bartz, Michael Meißner and Tobias Hüttner. Extending Graphics Hardware For Occlusion Queries In OpenGL. Proceedings of Eurographics/SIGGRAPH workshop on graphics hardware 1998 pp. 97-103

7. Dirk Bartz, Michael Meißner and Tobias Hüttner. OpenGL-assisted Occlusion Culling for Large Polygonal Models. Computers & Graphics vol. 23 no. 5 1999 pp. 667-679

8. Fausto Bernardini, James T. Klosowski and Jihad El-Sana. Directional Discretized Occluders for Accelerated Occlusion Culling. Proceedings of EUROGRAPHICS 2000

9. Jiří Bittner, Vlastimil Havran and Pavel Slavík. Hierarchical Visibility Culling with Occlusion Trees. Computer Graphics International 1998 pp. 207-219

10. Karsten Bormann. An Adaptive Occlusion Culling Algorithm for use in Large VEs. IEEE Virtual Reality 2000 p. 290

11. Allen Bourgoyne. Renée Bornstein and David Yu. Silicon Graphics Visual Workstation OpenGL Programming Guide For Windows NT. Document number 007-3876-001, Silicon Graphics, 1999

12. James H. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. Communications of the ACM vol. 19 no. 10 1976 pp. 547-554

13. Daniel Cohen-Or, Gadi Fibich, Dan Halperin and Eyal Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. Proceedings of EUROGRAPHICS 1998 pp. 243-253

14. Satyan Coorg and Seth Teller. Real-Time Occlusion Culling for Models with Large Occluders. ACM symposium on interactive 3D graphics 1997 pp. 83-90

15. Pavan K. Desikan, T. M. Murali and Pankaj K. Agarwal. Occlusion Culling Using Exact Shadow Computations and Ray-Shooting Queries. Preliminary draft, Duke University, 2000

16. Frédo Durand, George Drettakis and Claude Puech. The 3D visibility complex: a new approach to the problems of accurate visibility. Proceedings of Eurographics Workshop on Rendering 1996 pp. 245-256

17. Frédo Durand, George Drettakis and Claude Puech. The Visibility Skeleton: A Powerful And Efficient Multi-Purpose Global Visibility Tool. Proceedings of SIGGRAPH 97 pp. 89-100

18. Frédo Durand, George Drettakis, Joëlle Thollot and Claude Puech. Conservative Visibility Preprocessing using Extended Projections. Proceedings of SIGGRAPH 2000 pp. 239-248

19. Henry Fuchs, Zvi M. Kedem and Bruce F. Naylor. On visible surface generation by a priori tree structures. Proceedings of SIGGRAPH 80 pp. 124-133

20. Chris Georges. Obscuration Culling on Parallel Graphics Architecures. Technical Report TR95-017, UNC-Chapel Hill, 1995

21. Craig Gotsman, Oded Sudarsky and Jeffrey A. Fayman. Optimized occlusion culling using five-dimensional subdivision. Computers & Graphics 23 1999 pp. 645-654

22. Ned Greene, Michael Kass and Gavin Miller. Hierarchical Z-Buffer Visibility. Proceedings of SIGGRAPH 93 pp. 231-238

23. Ned Greene. Hierarchical Polygon Tiling with Coverage Masks. Proceedings of SIGGRAPH 96 pp. 65-74

24. Ned Greene. Occlusion Culling with Optimized Hierarchical Buffering. SIGGRAPH 99 Sketches & Applications p. 261

25. Eduard Gröller and Werner Purgathofer. Coherence in Computer Graphics. Technical report TR-186-2-95-04, Vienna University of Technology, 1995

26. Heinrich Hey, Robert F. Tobler and Werner Purgathofer. Real-Time Occlusion Culling With A Lazy Occlusion Grid. Proceedings of Eurographics Workshop on Rendering 2001 pp. 215-220

27. Poon Chun Ho and Wenping Wang. Occlusion Culling Using Minimum Occluder Set and Opacity Map. Proceedings of IEEE International Conference on Information Visualization 1999 pp. 292-300

28. Hewlett-Packard. OpenGL Implementation Guide. www.hp.com/workstations/support/documentation/manuals/user_guides/graphics/opengl/ImpGuide/01_Overview.html#OcclusionExtension, 2000

29. T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff and H. Zhang. Accelerated Occlusion Culling using Shadow Frusta. ACM Symposium on Computational Geometry (SCG) 1997

30. James T. Klosowski and Cláudio T. Silva. The Prioritized-Layered Projection Algorithm for Visible Set Estimation. IEEE transactions on visualization and computer graphics vol. 6 no. 2 pp. 108-123, 2000

31. James T. Klosowski and Cláudio T. Silva. Efficient Conservative Visibility Culling Using The Prioritized-Layered Projection Algorithm. SIGGRAPH 2000 Course Notes 4

32. Vladlen Koltun, Yiorgos Chrysanthou and Daniel Cohen-Or. Virtual Occluders: An Efficient Intermediate PVS representation. Proceedings of Eurographics Workshop on Rendering 2000 pp. 59-70

33. Subodh Kumar, Dinesh Manocha, William Garrett and Ming Lin. Hierarchical Back-Face Computation. Proceedings of Eurographics Workshop on Rendering 1996 pp. 235-244

34. Fei-Ah Law and Tiow-Seng Tan. Preprocessing Occlusion For Real-Time Selective Refinement. Symposium on Interactive 3D Graphics 1999 pp. 47-53

35. David Luebke and Chris Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. Symposium on Interactive 3D Graphics 1995 pp. 105-106

36. Tomas Möller and Eric Haines. Real-Time Rendering pp. 192-200, 1999

37. Bruce F. Naylor. Partitioning Tree Image Representation and Generation from 3D Geometric Models. Graphics Interface ´92 pp. 201-212

38. Bruce F. Naylor. Interactive Playing with Large Synthetic Environments. Symposium on Interactive 3D Graphics 1995 pp.107-108

39. NVIDIA. GeForce3: Lightspeed Memory Architecture. Technical Brief, 2001

40. Harry Plantinga and Charles R. Dyer. Visibility, Occlusion and the Aspect Graph. International Journal of Computer Vision 5(2) 1990 pp. 137-160

41. C. Saona-Vázquez, I. Navazo and P. Brunet. The Visibility Octree. A Data Structure for 3D Navigation. Computers & Graphics 23 pp. 635-643, 1999

42. Gernot Schaufler, Julie Dorsey, Xavier Decoret and François X. Sillion. Conservative Volumetric Visibility with Occluder Fusion. Proceedings of SIGGRAPH 2000 pp. 229-238

43. Dieter Schmalstieg and Robert F. Tobler. Exploiting coherence in 2½D visibility computation. Computers & Graphics vol. 21 no. 1 p. 121, 1997

44. Noel D. Scott, Daniel M. Olsen and Ethan W. Gannett. An Overview of the VISUALIZE fx Graphics Accelerator Hardware. Hewlett-Packard Journal May 1998 pp. 28-34

45. Ken Severson. VISUALIZE Workstation Graphics for Windows NT. Hewlett-Packard product literature, 1999

46. A. James Stewart. Hierarchical Visibility in Terrains. Proceedings of Eurographics Workshop on Rendering 1997 pp. 217-228

47. Oded Sudarsky and Craig Gotsman. Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality. Proceedings of EUROGRAPHICS 1996 pp. 249-258

48. Oded Sudarsky and Craig Gotsman. Dynamic Scene Occlusion Culling. IEEE transactions on visualization & computer graphics vol. 5 no. 1 pp. 217-223, 1999

49. Seth J. Teller and Carlo H. Séquin. Visibility Preprocessing For Interactive Walkthroughs. Proceedings of SIGGRAPH 91 pp. 61-69

50. Michiel van de Panne and A. James Stewart. Effective Compression Techniques for Precomputed Visibility. Proceedings of Eurographics Workshop on Rendering 1999 pp. 313-324

51. Yigang Wang, Hujun Bao and Qunsheng Peng. Accelerated Walkthroughs of Virtual Environments Based on Visibility Preprocessing and Simplification. Proceedings of EUROGRAPHICS 1998 pp. 187-194

52. Michael Wimmer, Markus Giegl and Dieter Schmalstieg. Fast Walkthroughs with Image Caches and Ray Casting. Proceedings of Eurographics workshop on virtual environments 1999 pp. 73-84

53. Peter Wonka and Dieter Schmalstieg. Occluder Shadows for Fast Walkthroughs of Urban Environments. Proceedings of EUROGRAPHICS 1999 pp. 51-60

54. Peter Wonka, Michael Wimmer and Dieter Schmalstieg. Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs. Proceedings of Eurographics Workshop on Rendering 2000 pp. 71-82

55. Feng Xie and Michael Shantz. Adaptive Hierarchical Visibility in a Tiled Architecture. Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware 1999 pp. 75-84

56. Roni Yagel and William Ray. Visibility Computation for Efficient Walkthrough of Complex Environments. Presence vol. 5 no. 1 pp. 45-60, 1996

57. Brian Zaugg and Parris K. Egbert. Voxel Column Culling: Occlusion Culling for Large Terrain Models. Proceedings of the Joint Eurographics-IEEE TCVG Symposium on Visualization 2001 pp. 85-93

58. Hansong Zhang, Dinesh Manocha, Tom Hudson and Kenneth E. Hoff III. Visibility Culling using Hierarchical Occlusion Maps. Proceedings of SIGGRAPH 97 pp. 77-88