Occlusion Culling for Virtual Environments based on the 3D-Sectorgraph

Dipl.-Inform. Jan Klein
janklein@uni-paderborn.de
Theoretical Computer Science
University of Paderborn

Dipl.-Inform. Matthias Fischer
mafi@uni-paderborn.de
Theoretical Computer Science
University of Paderborn

Prof. Dr. Friedhelm Meyer auf der Heide
Diploma Thesis Jan Klein
GI-Department 4

Abstract

We present a new approximate occlusion-culling algorithm that in contrast to other algorithms, manages the objects of the scene in a 3D-sectorgraph. For generating a frame, as far as possible only the visible objects are rendered that can be found quickly by an edge of the graph. The algorithm allows a real-time navigation with over 20 frames per second in complex scenes consisting of over 10 millions of polygons. Moreover, approximation errors are very low.

1. Introduction

A basic problem in computer graphics is the real-time navigation (walkthrough) in complex virtual environments. The virtual environment – also called scene – is usually modelled by surfaces that are described by polygons. To generate an image of a scene, the well-known z-buffer algorithm can be used that renders all the polyons. Its disadvantage is its running time of $\Theta(n + a)$ [6] which is linear in the number n of polygons and the area a of the projected, clipped polygons (without consideration of any covering): Consequently, a real-time navigation with about 20 frames per second is not possible in large scenes.

Occlusion-culling algorithms can reduce this problem by rendering only the polygons which are visible to the viewer. Objects (occludees) that are covered by other objects (occluders) should be excluded from rendering. In the following, a new occlusion-culling algorithm is presented that in contrast to other algorithms manages the objects of the scene in a network data-structure, namely a 3D-sectorgraph [10]. The objects of the scene – e.g., houses, trees, cars, etc. – are represented by nodes in the graph. To produce an image of the scene, the graph is traversed by a "restricted" breadthfirst search (BFS), i.e., the BFS traverses as far as possible only those areas that are visible to the viewer (see Fig. 1).

During BFS, the visible objects corresponding to the nodes found are immediately rendered and serve for occluding other objects. Necessary visibility tests are done with the aid of the hardware z-buffer. Theses tests as well as restricted BFS are performed approximatively, i.e., the generated images may contain little errors. The OCS algorithm (Occlusion Culling based on Sectorgraphs) described in Section 3 allows to navigate with over 20 frames per second in complex scenes consisting of over 10 millions of polygons on a low-end PC.

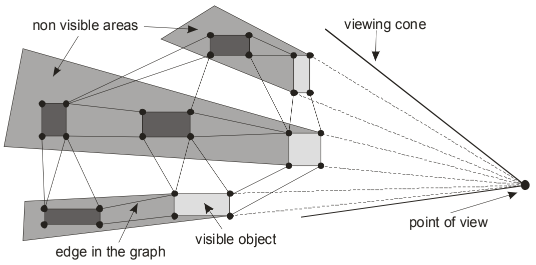

Fig. 1: Only the bright objects are visible to the viewer.

2. Related Work

Known occlusion-culling algorithms, e.g., see [1, 5, 12], manage the polygons in volume-separating data structures, as, e.g., in an octree, BSP-tree, kD-tree, etc.: All polygons in a certain 3D-volume bounded by a box are attached with it. If such a bounding box is not visible, all attached polygons are also not visible. The OCS algorithm, however, manages the objects of the scene with a 3D-sectorgraph. As in [11] occluded or visible cells with sightlines are determined, the OCS algorithm finds, as far as possible, the visible objects by the edges of the graph. The visibility tests of the OCS algorithm however are done during the navigation.

Image-space occlusion-culling algorithms: These algorithms test the visibility of a box with its projection onto the viewing plane. Indeed most of them have the disadvantage that for that purpose, they have to read many values from the z-buffer, e.g., like [5], or from a similar buffer of the graphics card [1], or from an additional software-buffer [8]. In practice, reading the values appears to be quite expensive, especially on PC-architectures. Finally, these algorithms are only faster than the z-buffer algorithm if the scenes are appropriately large: Reading the z-values and the computations of the occlusion-culling algorithms have to be cheaper than rendering the non-visible polygons. Generally, the scenes are so large that again a real-time navigation is not possible. The OCS algorithm avoids this problem: By managing the objects of the scene in a 3D-sectorgraph, the number of z-buffer accesses can clearly be reduced by approximations while approximation errors are very low. The algorithm is faster than the z-buffer algorithm for relatively small scenes as well as for very complex scenes.

Object-space occlusion-culling algorithms: These algorithms, e.g., see [3, 7], need no expensive accesses to any buffer, but they often have the disadvantage that they depend on occluders that are large or well chosen in the preprocessing. Furthermore, they obtain only poor results in scenes which consist of many single non-coherent polygons. For this reason, the developed OCS algorithm is an image-space algorithm, in order to achieve good results in more general scenes.

PVS algorithms (Potentially-Visible-Sets algorithms) are based on the idea of cells and portals, e.g., see [11]. Of course there exist some algorithms, e.g., see [11], which allow a real-time navigation in complex scenes, but they often have the disadvantage that they only fit for office rooms or other similar architectural scenes that have a volume-separating structure. The OCS algorithm can also be used for outdoor scenes that do not have such a structure. A more precise overview on occlusion-culling algorithms can be found in [2] as well as in [4, p. 137 ff.].

3. Occlusion Culling based on the 3D-Sectorgraph

Preprocessing: Before navigation, for every single object in the scene, a bounding box is determined, i.e., a box that encloses as close as possible all polygons of the object. Subsequently, the 3D-sectorgraph is built from the corner points of all bounding boxes of the scene; thus, the points correspond to the nodes of the graph.

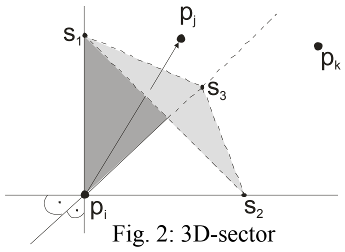

Fig. 2: 3D-sector

In order to construct this graph, the space around every node or point p_i is subdivided into 3D pyramidal sectors. The number of sectors for a scene is variable, but fixed. In every sector S, point p_i gets an edge to the point in S that has the least distance to p_i, if such a point exists. In Fig. 2 the space around p_i is subdivided into 8 sectors, from which only one is drawn: This one is bounded by three halflines through the additionally drawn points s_1, s_2 and s_3, starting in p_i. The graph for a scene of n objects can be constructed in time $O(n \cdot \log^2 n)$.

Idea: One can show that BFS on a sectorgraph still finds nearly all visible points if the search is aborted at every node which is visible to the viewer [9, pp. 73-75]. This property is based on the fact that in a sectorgraph a path from one point p_1 to any other point p_2 exists, that runs in sector S outgoing from p_1 and including p_2 or in the neighbour sectors of S [9, pp. 48-53].

OCS algorithm: Based on this idea, the walkthrough algorithm is developed. To render a frame, the graph is traversed by a restricted BFS. The search starts at the visible node with the least distance to the viewer. If the BFS finds a node or a point which is not visible to the viewer, this point is marked and no longer considered when the search continues. So it can be ensured that the search traverses as far as possible only the visible areas. The visibility test of a point can easily be realised by the z-buffer of the graphics card. As only one pixel has to be read from the buffer, this test turns out to be cheap. However, if a found point is visible, the corresponding object is immediately rendered, if this has not already been done. During this, the z-values are updated automatically and thus further occluded objects can be discovered.

To avoid approximation errors which are possibly made by the restriction of the BFS as well as by the visibility test (explanation in section 4), the algorithm offers an additional option, namely circular-range search (CRS): The CRS guarantees that all nodes within a fixed radius r around the viewer are found; the corresponding objects are rendered. Then only nodes outside the radius r are excluded from further BFS if they are not visible to the viewer.

4. Analysis

The analysis investigates the execution time of the algorithm as well as two kinds of approximation errors.

Benchmark: To analyse the execution time of the algorithm empirically, a town scene was modelled which consists of 12,745 objects (10,132,144 polygons) like houses, trees and cars (see Fig. 3). The objects are randomly distributed on a rectangular surface. Thus the scene is not especially so constructed that only a few objects are visible from each point of view. In order to analyse the algorithm's execution time, the framerates, etc., a path of 260 steps is fixed for the walkthrough. This path is directed in a way that as many polygons as possible are in the viewing cone. For the measurements an AMD Athlon 1.2 GHz with 256 MB RAM and a GeForce 256 graphics chip were used.

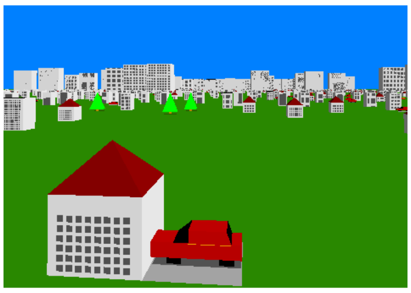
Fig. 3: Scene consisting of over 10 mio. polygons

Execution time: Fig. 4 shows the produced framerates. Without CRS the average number of generated frames amounts to 20.5 frames per second, whereas the z-buffer algorithm produces only 0.3 frames per second. Furthermore, the measurements show that as wanted, the algorithm excludes most polygons from the rendering and renders as far as possible only the visible polygons: In the average case, the algorithm renders only 11,584 of 10,132,144 polygons, whereby at least about 2,000 polygons have to be rendered for a correct image.

Measuring errors: To measure the approximation errors the number of incorrect pixels is compared to the number of all 500·500 pixels of the image.

Approximation errors due to inaccurate BFS: The rendered image can have some incorrect pixels because it cannot be guaranteed that all visible objects are found, if all non-visible points are not considered during the BFS. Strictly speaking, a few non-visible points still have to be considered (see [9, pp. 73-75]), in order to find all visible objects. In practice, it is much easier and faster (but still inaccurate) if the number of outgoing sectors per point is increased up to about 70 to find visible nodes along additional paths, that would not have been found otherwise. Tests show that then only about 0.04 % of all pixels are incorrect, in the average case.

Approximation errors due to inaccurate visibility tests: Moreover it can cause errors to test only the visibility of the corner points of the bounding boxes: It might happen that all corner points are not visible to the viewer although parts of the corresponding object are visible. As every object is assigned to an own bounding box and as a non-hierarchical datastructure is used, it is probable that rather than a whole cluster of objects, just some single objects are missed during rendering. Tests show that in the average case only about 0.21 % of all pixels are incorrect, but some single images can consist of about 10 % incorrect pixels. It is obvious that scenes can be constructed which increase these errors as much as one likes. Especially not rendered visible objects near to the viewer contribute to these errors because these objects had to be projected on relatively large parts of the viewing plane because of the used central projection.

These errors can be easily reduced by an additional circular-range search (CRS) described in Section 3. Then the average number of generated frames still amounts to about 15.5 frames per second (see Fig. 4). Fig. 5 shows the number of incorrect pixels (in %) per frame with CRS: One can see that this number – apart from very few runaways – is clearly below 0.2 %. Besides, many images are generated without any errors. In the average case only 0.03 % of all pixels are incorrect.

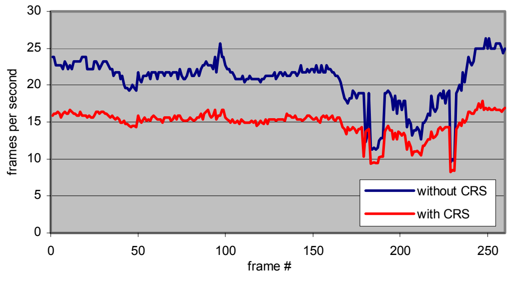

Fig. 4: Framerates (CRS: Circular-Range-Search)

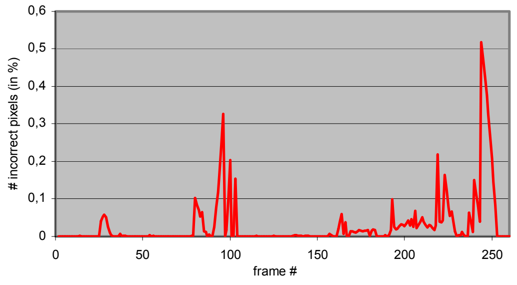

Fig. 5: Approximation errors with CRS

References

[1] Bartz, Dirk; Meißner, Michael; Hüttner, Tobias: OpenGL-assisted Occlusion Culling for Large Polygonal Models; Computer and Graphics Journal, Vol. 23, No. 5, pp. 667-679, 1999

[2] Cohen-Or, Daniel; Chrysanthou, Yiorgos; Silva, Cláudio T.: A Survey of Visibility for Walkthrough Applications; SIGGRAPH ‘00 course notes, 2000

[3] Coorg, S.; Teller, S.: Real-Time Occlusion Culling for Models with Large Occluders; ACM Symposium on Interactive 3D Graphics Proc., pp. 83-90, 1997

[4] Durand, F.: 3D Visibility: Analytical study and Applications; Ph.D. thesis, Université Joseph Fourier, Grenoble, France, 1999

[5] Greene, N.; Kass, M.; Miller, G.: Hierarchical Z-Buffer Visibility; ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH ’93 Proceedings), pp. 231-238, 1993

[6] Heckbert, P.; Garland, M.: Multiresolution Modeling for Fast Rendering, Graphics Interface ’94, pp. 43-50, 1994

[7] Hudson, T.; Manocha, D.; Cohen, J.; Lin, M.; Hoff, K.; Zhang, H.: Accelerated Occlusion Culling using Shadow Frusta; Proc. of ACM Symp. on Comput. Geometry, pp. 1-10, 1997

[8] Hey, Heinrich; Tobler, Robert F.; Purgathofer, Werner: Real-Time Occlusion Culling With A Lazy Occlusion Grid; Tech. Report TR-186-2-01-02, Vienna University of Technology, 2001

[9] Klein, J.: Occlusion-Culling in virtuellen Umgebungen mittels 3D-Sektorengraphen; Diploma Thesis in Computer Science, Paderborn, 2001

[10] Ruppert, J.; Seidel, R.: Approximating the d-dimensional complete Euclidean graph; Proc. of the 3rd Canadian Conference on Computational Geometry (CCCG '91), pp. 207-210, 1991

[11] Teller, S. J.; Séquin, C. H.: Visibility preprocessing for interactive walkthroughs; ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH ’91 Proceedings), pp. 61-69, 1991

[12] Zhang, H.; Manocha, D.; Hudson, T.; Hoff, K. E.: Visibility Culling using Hierarchical Occlusion Maps; ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH ’97 Proceedings), pp. 77-88, 1997