ENTKERNER: A SYSTEM FOR REMOVAL OF GLOBALLY INVISIBLE TRIANGLES FROM LARGE MESHES

Manfred Ernst[1] Frank Firsching[1] Roberto Grosso[2]

[1] *Bytes+Lights GmbH, Erlangen, Germany.*
[2] *Computer Graphics Group, University of Erlangen*
{*manfred.ernst, frank.firsching*}*@bytes-lights.de, grosso@informatik.uni-erlangen.de*

ABSTRACT

We present a method that computes a global potentially visible set for the complete region outside the convex hull of an object. The technique is used to remove invisible parts (triangles) from complex tessellated CAD models. Such optimizations are required to achieve interactive frame rates for the visual exploration of huge data sets on graphics workstations. Our algorithm is subdivided into three stages. At the first step the the tesselated object is rendered with OpenGL from various camera positions to detect visible triangles very fast. A hardware-accelerated hemicube test is applied as the second stage, marking all triangles that can directly exchange energy with an infinitely distant environment. Finally, a Monte Carlo ray tracing pass is applied to each remaining triangle, sampling its visibility with arbitrary accuracy. All invisible triangles are completely removed from the mesh. It is therefore not necessary to store visibility information, allowing the reduced mesh to be processed and displayed by any software.

Keywords: Mesh simplification, visibility testing, industrial application, complex meshes, CAD

1. INTRODUCTION

Triangulated CAD models can consist of several millions of triangles. Such large data sets still cannot be handled in real-time by modern graphics cards. Nevertheless interactive frame rates are crucial for visual quality assurance. Most often a model is inspected only from the outside of the convex hull or any point reachable from there without penetrating the surface. The interior of the object is irrelevant in this case. The elimination of the invisible parts (triangles) leads to significant improvements for real-time applications, because the display performance is limited by the geometric complexity of the meshes.

Traditionally, users have selected and removed the invisible parts of the models by hand. This process is very time consuming, in particular when a large number of data sets must be prepared for visualization. More recently, fully automatic removal methods were integrated into commercial mesh processing software. A common technique uses OpenGL and graphics hardware to render the object from several camera positions to identify visible triangles. Yet, the limited precision of this approach causes severe problems for complex geometries. A more robust method, that computes visibility with arbitrary precision was therefore requested by the industry. We have developed an efficient three stage algorithm and integrated it into our commercial mesh processing tool. It is successfully used by industrial customers.

2. PREVIOUS WORK

Visibility computations can be categorized according to the criteria: from-point visibility vs. from-region visibility, object-precision vs. image precision and cell-and-portal vs. generic scenes. Extensive surveys about this research area can be found in [1] and [2].

From-region visibility for cell-and-portal scenes, that can be divided into rooms connected by doors or windows, was first addressed by Airey [3], Teller and Sequin [4] and by Luebke et al. [5]. Their methods target interactive walk-throughs of architectural interiors and similar data sets. For each cell, the potentially visible set (PVS) of scene entities is precomputed and queried during rendering. Their methods are *conservative* in that the PVS contain at least all visible objects.

A Point-based object-precision algorithm for scenes with large occluders was presented by Coorg and Teller in [6] and [7]. Hudson et al. have developed a technique to compute from-point visibility by culling objects at the frusta of potential occluders [8]. The algorithm was improved by Bittner et al., using a BSP-tree [9].

Image-precision from-point visibility is similar to the well known hidden surface removal problem. The difference is that it is sufficient for occlusion culling to determine whether any part of an object is visible. Ray tracing is an example for an image synthesis algorithm, that has efficient occlusion culling built right into it. Implementations with extraordinary performance were presented by Wald et al. [10], [11] and [12]. The hierarchical Z-buffer [13] and the hierarchical occlusion map [14] are intended for use in the classic rasterization pipeline. More recently, support for image-precision occlusion culling was integrated into graphics hardware. In OpenGL 1.5, occlusion queries return the number of pixels drawn onto the screen. This can be used to test if the bounding box of an object is visible, before the complete geometry is sent down the pipeline.

Generic from-region visibility can be computed by sampling the region with from-point visibility tests. Gotsman et al. use a 5D-tree to partition the five-dimensional viewing parameter space and store PVS in the leafs of the tree [15]. Conservative methods, that do not miss any potentially visible objects were discussed by Cohen-Or et al. [16] and by Saona-Vazquez et al. [17]. Their algorithms were not able to perform occluder fusion, which is very important for effective from-region visibility culling. Schaufler et al. voxelized the occluders and applied a fusion method to this discretized representation [18]. Durand et al. introduced extended projections, an extension to point-based image-precision algorithms [19].

3. OVERVIEW

Our method could be classified as an aggressive from-region visibility algorithm using image-precision techniques to samples the region. Aggressive means that the algorithm never misses any invisible triangles, but it may remove some visible geometry if not properly configured. This is an important feature for the massive data reduction required. Additionally, the visibility is not only computed for a small region, but globally for the complete area outside the convex hull of the object. Visibility information is not stored in a complex data structure. Instead, the invisible parts are completely removed from the scene description.

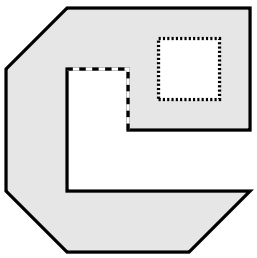


Figure 1: Visibility of edges in 2D. Solid lines are directly visible. The dashed lines are indirectly visible and the dotted lines are completely invisible.

We introduce the following nomenclature (depicted in figure 1 for visibility:

> A triangle t is *directly visible*, if there exists a ray from infinity, whose first intersection with the scene is in t.

> A triangle t is *indirectly visible*, if an arbitrary path from infinity exists, that ends in t and has no other intersection with the scene.

The algorithm is separated into three stages, that are always executed in this order:

Multi Test This test renders the scene from several camera positions using OpenGL. A triangle is marked as visible if it appears in any of the images.

Single Test Up to six images, covering all incoming directions, are rendered from the centroid of each triangle. If the background is visible (tested with occlusion query) in any image, the triangle is visible. In this way, direct visibility can be computed with high accuracy, except for partially occluded large triangles.

Ray Tracing Test A set of rays is adaptively cast from the surface of each triangle into all directions. If any ray leaves the scene, the triangle can be seen from outside the convex hull. With reflections enabled, this test can also compute indirect visibility. Ray tracing is the most reliable method with the highest accuracy.

Each test marks those triangles, that it has detected to be visible and passes the remaining triangles to the next stage. All triangles that are not marked after the last test, are eventually removed from the scene. The tests are used together to combine their advantages and improve performance as described in the following sections.

4. MULTI TEST

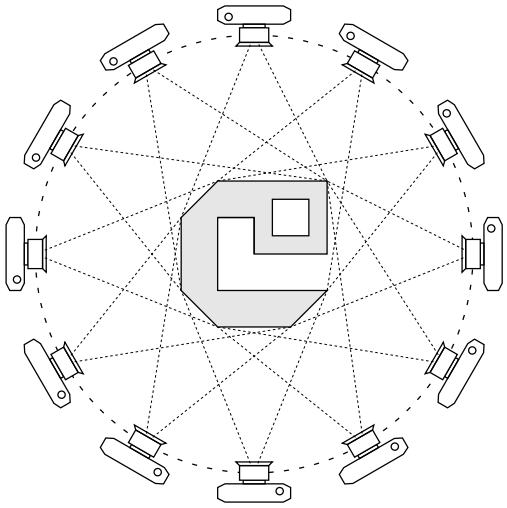


Figure 2: Principle of the multi test.

The multi test is based on the principle, to keep only those triangles visible from a set of multiple camera positions. Figure 2 illustrates this process. Therefor we render the model using OpenGL into a PBuffer. Each triangle is drawn with a unique color. Since we use a standard RGBA color format with eight bits per component, we are theoretically limited to models with at most $2^{32} - 1 = 4,294,967,295$ triangles. If such huge models will become usable in the future, we could easily change the PBuffer to a 128 bit floating point format. Afterwards we read back the image and mark every triangle, that is represented in the image as visible.

In order to achieve good results, we place the cameras on a sphere, which is 1.5 times bigger than the largest extent of the object bounding box. The viewing frustum is always headed towards the center of the object and the six frustum planes are fitted tightly around the bounding box. By this means we make the best use of the limited PBuffer resolution. Although this implies a fairly distorted frustum, it improves the identification rate, significantly.

Computing camera positions based on a simple spherical parameterization leads to lumping of the samples at the poles. This waste of computational resources can be avoided by transforming a 2D Larcher Pillichshammer random quasi Monte Carlo sequence from the $[0,1] \times [0,1]$ unit square to the sphere. The following mapping is applied:

$$\begin{aligned} x &= c_x + 2r \cdot \sin\left(\sqrt{u - u^2}\right) \cdot \cos(2\pi v) \\ y &= c_y + 2r \cdot \sin\left(\sqrt{u - u^2}\right) \cdot \sin(2\pi v) \\ z &= c_z + r(1 - 2u) \end{aligned}$$

where u and v are the coordinates of the sample point in the unit square and r is the radius of the sphere. The values c_x, c_y and c_z contain the center of the sphere and x, y and z are the resulting coordinates.

The key advantage of this method is to quickly detect a large number of directly visible triangles. For most simple models, it is sufficient to evaluate visibility from a few tens of camera positions. However, the method fails for high resolution triangle meshes, typically used in industrial applications. These models usually consist of triangles, whose projection measures less than a pixel in screen space. As a consequence, many triangles are overlooked, because they never appear in the frame-buffer. In fact, this is aliasing, due to under-sampling of the image. Unfortunately, the PBuffer resolution is limited by the hardware. Image 9 depicts an example for the failure of the multi test. A workaround for this restriction is tiled rendering. The image plane is divided into smaller rectangular regions that fit into the PBuffer. In this way, quality is increased whereas rendering time increases linearly with the number of tiles.

5. SINGLE TEST

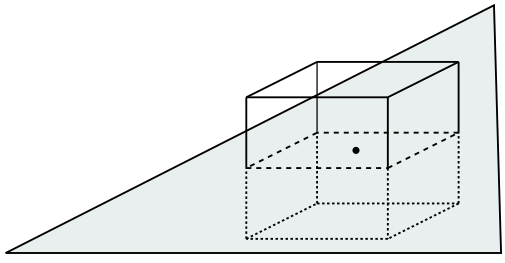


Figure 3: Placement of the virtual cube in the single visibility test.

The second test actually reverses the method of the multi test. While the multi test considers the whole model in each step, this is a triangle centric approach. It iterates over all triangles, that are not yet marked as

visible and applies a technique similar to the hemicube method [20] for radiosity computations.

Basically we place a virtual cube in the center of the triangle and render six images onto each face. We need to use the entire cube, because most real world data sets, do not have consistently oriented normal vectors. Depending on the capabilities of the graphics hardware we either use occlusion queries or frame-buffer readback for compatibility to OpenGL 1.2.

The occlusion method renders the model after setting up the camera transformation to look through one of the cube's faces. Subsequently, the model view matrix is reset to the identity. A screen filled quad is now drawn at the far plane with occlusion test enabled. If the number of visible fragments is greater than zero, a straight path to the triangle originating from infinity, exists. This means, that the triangle is visible outside the convex hull of the object and should be kept. Following tests with further directions of the virtual cube can be skipped.

The plain OpenGL approach running on older graphics hardware, that does not support occlusion queries, renders the model into the depth buffer just like the advanced method. The screen sized quad is now drawn using a special color. The rendered image is read back into main memory and inspected for pixels containing this color.

Both methods render the geometry without updating the color buffer. Only the depth values are computed. This technique saves memory bandwidth and exploits the capability of modern graphics chips to compute depth-only images in half the time.

The single test fails for large triangles, that are partially occluded, because the visibility is only tested for the centroid. On the other hand, the direct visibility is computed with very high accuracy for that point. Rendering six sides of the cube with a typical resolution of 64×64 pixels each, corresponds to 24,576 camera positions in the multi test. However, it is required to render up to six images for each triangle, leading to very long computation times.

6. RAY TRACING TEST

The global visibility problem is analog to the global illumination problem. Consider an infinitely large sphere surrounding the geometry, radiating inwards. A triangle is globally visible if any amount of energy from the light source can reach it, including multiple reflections on the way to the triangle. The important difference to global illumination is, that the exact amount of energy transferred need not be computed.

A large number algorithms to solve global light transport problems were developed in the past. Ray tracing

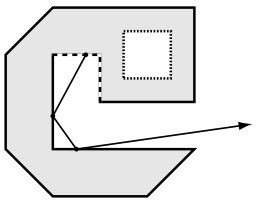


Figure 4: Path of a ray for an indirectly visible triangle.

is most suitable for a visibility test, because it is naturally adaptive, can handle arbitrarily complex scenes and allows for highly efficient implementations, including thread and host parallelism. Efficiency and good scalability for large scenes are clearly necessary to cope with typical scenes consisting of many millions of triangles. Adaptivity, on the other hand, is the key to exploit the difference to classic global illumination: It is sufficient to know if *any* ray can reach the triangle, possibly including an arbitrary number of reflections at the geometry.

Performance of ray tracing based algorithms is directly related to the chosen sampling strategy and the degree of coherence in the rays. Sampling is substantially different to image rendering. For the visibility test, a set of rays is shot from the triangle surface into different directions. Two sets of samples must be computed, one for the ray origins, located on the triangle, and one for the ray directions, covering a full sphere. We use two connected randomized quasi Monte Carlo (RQMC) sequences for this purpose. In this way, locations and directions cover their domains evenly, while each ray has a unique origin and direction.

Locations on the triangle are computed by transforming a Sobol sequence from the unit square to the triangle surface. Very efficient code for the computation of Sobol's low discrepancy point set is presented in [21]. The mapping to the triangle surface is computed according to Turk's formula [22]:

$$\begin{aligned} \alpha &= 1 - \sqrt{u} \\ \beta &= (1 - v)\sqrt{u} \\ \gamma &= v\sqrt{u} \end{aligned}$$

where u and v are the coordinates of the point in the unit square and α, β and γ are the barycentric coordinates of the transformed point in the triangle. Figure 5 shows an example for this mapping.

For the directional samples, we do not use a single Monte Carlo sequence. Instead, the directional space is subdivided into a set of sectors, each of which is sampled by a short Larcher-Pillichshammer sequence as described in section 4. In this way, directions can be tested adaptively, while coherence of the rays is increased.

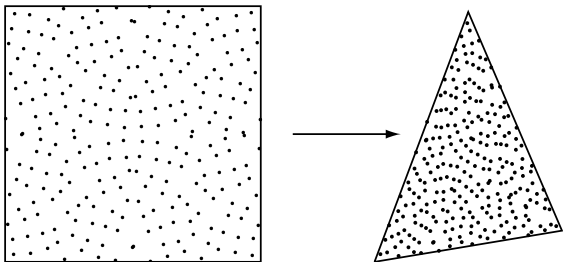


Figure 5: Sampling of a triangle with 256 transformed Sobol samples.

An important advantage of ray tracing over all other tests, is its ability to determine not only direct visibility, but also indirect visibility. This is achieved by switching to the path tracing algorithm, bouncing rays up to n times after they hit an occluder. Rays are scattered with equal probability into all directions of the incoming hemisphere. This corresponds to a perfectly diffuse reflection model without importance sampling. When all triangles in the scene are consistently oriented, it would also be possible to propagate visibility along the rays. A ray that hits a visible triangle on its front side can propagate the visibility to all triangles that its path hits.

A kd-tree is used for ray shooting acceleration. This data structure is well suited for very large data sets and can handle scenes with detail at different scales, very efficiently. We apply various optimization techniques to reduce the memory footprint and increase traversal speed. A good survey of such methods was presented by Havran in [23]. The Moeller-Trumbore ray-triangle test [24] was chosen for intersection calculations, because it does not require the plane equation to be stored with the triangles.

In contrast to the other visibility tests, the ray tracing method has no inherent limitations and can compute direct or indirect visibility with arbitrary precision.

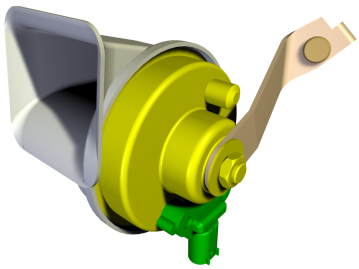


Figure 6: The horn dataset is a smaller model consisting of 36,621 triangles.

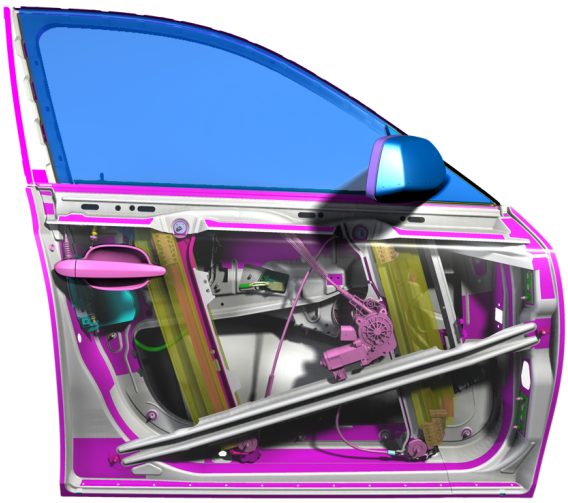


Figure 7: The car door scene consisting of 782,018 triangles. The top most surface was removed to show the interior of the door.

Two parameters are controlling accuracy: the number of rays and the trace depth for each ray. Both can be chosen by the user so that an acceptable trade-off between computation time and precision is achieved.

7. RESULTS

The `entkerner` is already in use by our customers in the automotive industry. We have chosen two typical examples from this application area to evaluate the three test stages and demonstrate the capabilities of our system. The first example is the door of a car including all mechanical components consisting of 782,018 triangles. Our second example is the complete engine of the new BMW 5 series. The model contains 3,741,833 triangles. Additionally we applied the `entkerner` to a smaller dataset, the horn. It has 36,621 triangles. The single test was analyzed using this model.

We have tested various parameter sets on a dual Xeon 2.4 GHz with 2 GB of Rambus memory and an NVIDIA GeForce FX 5800 Ultra graphics card. The results are listed in tables 1, 2 and 3. The following abbreviations are used:

MR PBuffer Resolution for the multi test.

MV Number of views used by the multi test.

SR PBuffer Resolution for the single test.

TR Maximal number of rays shot by the ray tracing test per triangle.

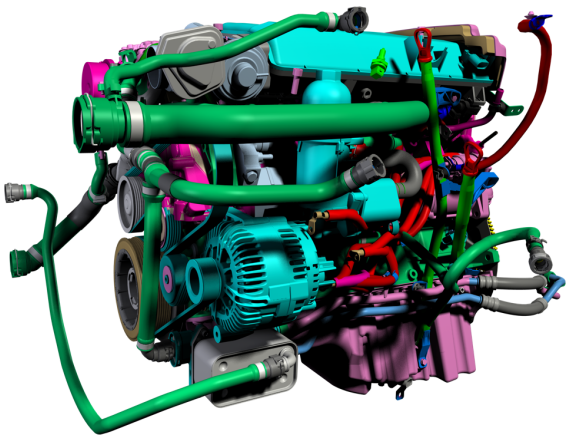


Figure 8: The engine of a BMW 5 with 3,741,833 triangles. Most of the geometry is never visible when the camera is outside the convex hull of the engine.

Time Total computation time.

Tri Number of remaining triangles after all tests have passed.

Vis Percentage of initial triangles classified as visible.

MR	MV	TR	Time	Tri	Vis
-	-	-	-	3,741,833	100 %
256	128	-	3m:18s	341,740	9 %
512	128	-	3m:17s	556,752	15 %
1024	128	-	3m:20s	771,270	21 %
2048	128	-	3m:31s	942,651	25 %
256	512	-	13m:10s	554,864	15 %
512	512	-	13m:10s	776,893	21 %
1024	512	-	13m:13s	966,731	26 %
2048	512	-	14m:02s	1,109,857	30 %
2048	128	64	19m:39s	1,115,762	29 %
2048	128	256	1h:02m	1,217,817	33 %
2048	128	1024	3h:43m	1,311,566	35 %

Table 1: Results for various parameter sets tested with the engine scene.

Obviously, the PBuffer resolution for the multi test has only little influence on the rendering time, because performance is limited by the geometry, not the fill rate. The recognition rate for visible triangles, however, increases dramatically. It is therefore recommended to chose the highest resolution possible. Figure 9 shows a series of images with different PBuffer resolutions. Only if the resulting model is viewed at a fixed resolution without zooming in, is a lower PBuffer resolution reasonable. In this case, the holes in the geometry will never become visible.

MR	MV	TR	Time	Tri	Vis
-	-	-	-	782,018	100 %
256	128	-	42s	64,895	8 %
512	128	-	42s	87,904	11 %
1024	128	-	45s	111,012	14 %
2048	128	-	56s	132,139	16 %
256	512	-	2m:45s	88,735	11 %
512	512	-	2m:48s	115,128	14 %
1024	512	-	2m:59s	143,422	18 %
2048	512	-	3m:45s	169,526	21 %
2048	128	64	4m:25s	147,554	18 %
2048	128	256	13m:36s	167,283	21 %
2048	128	1024	48m:28s	199,605	25 %

Table 2: Results for various parameter sets tested with the door scene.

MR	SR	TR	Time	Tri	Vis
-	-	-	-	36,621	100 %
2048	-	-	16s	22,577	62 %
2048	-	1024	1m:28s	29,165	80 %
-	64	-	3m:49s	29,027	79 %
2048	64	-	2m:38s	29,347	80 %
2048	64	1024	3m:48s	29,706	81 %

Table 3: Results for various parameter sets tested with the horn scene. The multi test used 128 camera positions.

A low frame-buffer resolution can be alleviated to some degree by increasing the number of camera positions. This method comes at a significantly higher cost. In fact, the complexity increases linearly with the number of views.

Ray tracing with a large number of rays detects visible triangles most reliably. The computation time grows linearly with the number of rays and the number of reflections. Most of the time is spent testing invisible triangles, because the maximum number of rays is always shot for them. We apply the multi test in advance, to detect many directly visible triangles very fast. Reflections were not required for the test scenes.

Tessellations of CAD data are usually not crack free. Many triangles are therefore marked visible, mistakenly. This is a problem of all tests, because a single pixel or a single ray is sufficient to mark a triangle. This issue could be solved by defining a threshold value greater than one.

8. CONCLUSION

We have presented a fast and robust technique to remove globally invisible triangles from large data sets.

No assumptions are made about the geometry and its topology. Two hardware accelerated tests and one ray tracing based method were developed. The three tests can be executed consecutively to obtain the best merits of each algorithm. A commercially available implementation of the method is in use by our industrial customers.

In a future release we are planning to integrate distributed ray tracing on PC clusters for very complex geometries. Another feature will be tiled rendering for the multi-test, to circumvent the limited PBuffer size. For the single test, we want to implement a hemispherical projection using vertex programs. This would reduce the number of render passes per triangle from six to two.

References

[1] Durand F. *3D Visibility: Analytical study and Applications.* Ph.D. thesis, Universite Joseph Fourier, Grenoble, France, July 1999

[2] Cohen-Or D., Chrysanthou Y., Silva C.T., Durand F. "A Survey of Visibility for Walkthrough Applications." *IEEE Transaction on Visualization and Computer Graphics*, 2002

[3] Airey J. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations.* Ph.D. thesis, University of North Carolina, Chappel Hill, 1991

[4] Teller S., Séquin C. "Visibility Preprocessing for Interactive Walkthroughs." *Computer Graphics (Proceedings of SIGGRAPH '91)*, pp. 61–69, 1991

[5] Luebke D., Georges C. "Portals and mirrors: simple, fast evaluation of potentially visible sets." *Proceedings of the 1995 Symposium on Interactive 3D graphics.* ACM Press, 1995

[6] Coorg S., Teller S. "Temporally Coherent Conservative Visibility." *Proceedings of the 13th ACM Symposium on Computational Geometry*, pp. 78–87. 1996

[7] Coorg S.R., Teller S.J. "Real-Time Occlusion Culling for Models with Large Occluders." *Symposium on Interactive 3D Graphics*, pp. 83–90, 189. 1997

[8] Hudson T., Manocha D., Cohen J.D., Lin M.C., III K.E.H., Zhang H. "Accelerated Occlusion Culling using Shadow Frusta." *Symposium on Computational Geometry*, pp. 1–10. 1997

[9] Bittner J., Havran V., Slavík P. "Hierarchical Visibility Culling with Occlusion Trees." *Proceedings of Computer Graphics International '98 (CGI'98)*, pp. 207–219. IEEE, 1998

[10] Wald I., Slusallek P., Benthin C., Wagner M. "Interactive Rendering with Coherent Ray Tracing." *Computer Graphics Forum*, vol. 20, no. 3, 153–164, 2001

[11] Wald I., Kollig T., Benthin C., Keller A., Slusallek P. "Interactive Global Illumination using Fast Ray Tracing." *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, pp. 15–24. Jun. 2002

[12] Benthin C., Wald I., Slusallek P. "A Scalable Approach to Interactive Global Illumination." *Eurographics 2003.* Sep. 2003

[13] Greene N., Kass M., Miller G. "Hierarchical z-buffer visibility." *Computer Graphics*, vol. 27, no. Annual Conference Series, 231–238, 1993

[14] Zhang H., Manocha D., Hudson T., Hoff III K.E. "Visibility Culling Using Hierarchical Occlusion Maps." *Computer Graphics*, vol. 31, no. Annual Conference Series, 77–88, 1997

[15] Gotsman C., Sudarsky O., Fayman J. "Optimized Occlusion Culling Using Five-Dimensional Subdivision." *Computers and Graphics*, vol. 23, no. 5, 645–654, 1999

[16] Cohen-Or D., Fibich G., Halperin D., Zadicario E. "Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes." *Computer Graphics Forum*, vol. 17, no. 3, 1998

[17] Saona-Vazquez C., Navazo I., Brunet P. "The visibility octree: A data structure for 3D navigation." *Computers and Graphics*, vol. 23, no. 5, 635–644, 1999

[18] Schaufler G., Dorsey J., Decoret X., Sillion F.X. "Conservative Volumetric Visibility with Occluder Fusion." K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pp. 229–238. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000

[19] Durand F., Drettakis G., Thollot J., Puech C. "Conservative Visibility Preprocessing Using Extended Projections." K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pp. 239–248. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000

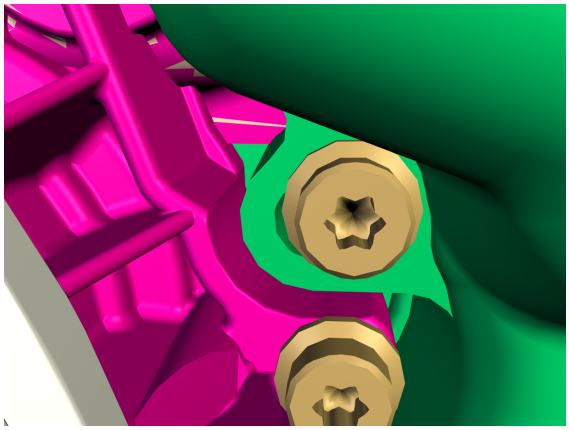
[20] Cohen M.F., Greenberg D.P. "The Hemi-Cube: A Radiosity Solution for Complex Environments." *Computer Graphics (Proceedings of SIGGRAPH 85)*, vol. 19, pp. 31–40. Aug. 1985

[21] Kollig T., Keller A. "Efficient Multidimensional Sampling." *Computer Graphics Forum*, vol. 21, no. 3, 557–564, 2002

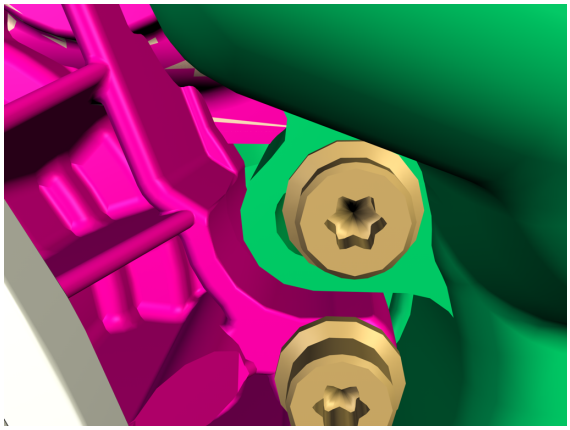
[22] Turk G. "Generating Random Points in Triangles." A.S. Glassner, editor, *Graphics Gems*, p. 24. Academic Press, San Diego, 1990

[23] Havran V. *Heuristic Ray Shooting Algorithms*. Ph.D. thesis, Czech Technical University, Prague, November 2000

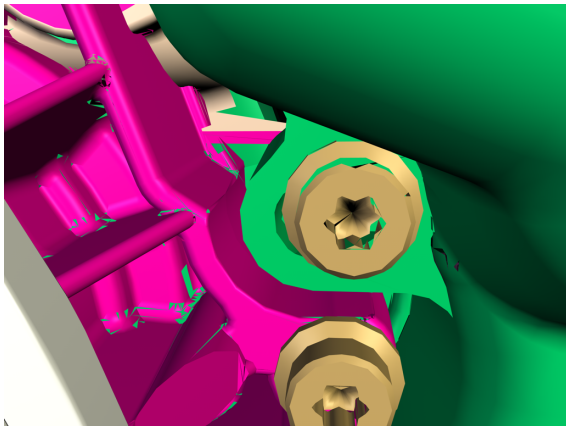
[24] Möller T., Trumbore B. "Fast, Minimum Storage Ray-Triangle Intersection." *Journal of Graphics Tools*, vol. 2, no. 1, 21–28, 1997



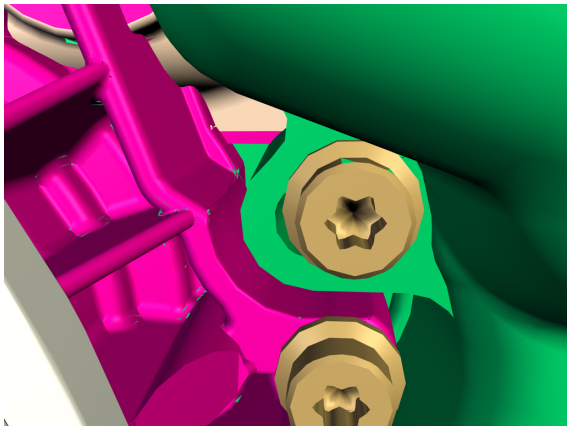
(a) The original engine dataset.



(b) The combined multi and ray tracing tests lead to an artifact free result. (MR 2048, MV 128, TR 1024)



(c) The multi test alone results in unsatisfying output. (MR 2048, MV 128)



(d) Raising the number of camera positions to 512 still leaves some holes in the geometry. (MR 2048, MV 512)

Figure 9: A detail view of the engine dataset showing some artifacts of the multi test.