Integrating Occlusion Culling with View-Dependent Rendering

Jihad El-Sana[*] Ben-Gurion University
Neta Sokolovsky[†] Ben-Gurion University
Cláudio T. Silva[‡] AT&T

Abstract

We present a novel approach that integrates occlusion culling within the view-dependent rendering framework. View-dependent rendering provides the ability to change level of detail over the surface seamlessly and smoothly in real-time. The exclusive use of view-parameters to perform level-of-detail selection causes even occluded regions to be rendered in high level of detail. To overcome this serious drawback we have integrated occlusion culling into the level selection mechanism. Because computing exact visibility is expensive and it is currently not possible to perform this computation in real time, we use a visibility estimation technique instead. Our approach reduces dramatically the resolution at occluded regions.

1 Introduction

Interactive rendering of large datasets is a fundamental issue for various applications in computer graphics. Although graphics-processing power is increasing every day, its performance has not been able to keep up with the rapid increase in data set complexity. To address this shortcoming, techniques, such as occlusion culling and level-of-detail rendering, are being developed to reduce the amount of geometry that is required to be rendered, while still preserving image accuracy.

The rendering time of a given polygonal dataset is mainly determined by the number of polygons sent to the graphics hardware. Hence, rendering-acceleration algorithms have tried to reduce the number of rendered polygons. These algorithms include geometric simplification and occlusion culling, which achieve their speedup through the decrease of the set of polygons sent to the graphics hardware. Geometric simplification reduces the number of the rendered triangles by reducing the resolution (number of triangles per area unit) of the dataset, while occlusion culling manages to reduce the size of the rendered-polygons set by culling invisible polygons.

In a certain way geometric simplification and occlusion culling achieve their respective speedups through orthogonal operations. Occlusion culling algorithms cull occluded triangles, but these techniques still render small triangles, which might not contribute significantly to the final image. Geometric simplifications, on the other hand, manage to reduce the number of triangles that occupy small regions, but they still render invisible triangles. The combination of both approaches can potentially lead to great reduction on the number of rendered triangles, while not compromising image quality. To our knowledge, the only other work that combines level-of-detail selection with visibility determination is the recent paper by Andujar *et al.* [2] (see section 2).

View-dependent rendering provides different level-of-detail over different regions of the same surface. The selection of appropriate level of detail is based on view-parameters such as illumination and view position. To combine view-dependent rendering with occlusion culling we add visibility as another parameter in selecting the appropriate level of detail. The major technical challenge lies in computing the visibility parameter fast enough as not to harm the frame rate. Performing exact visibility computations is expensive, and hard to achieve in real-time. In practice, we may not need complete accuracy, it would be sufficient to approximate visibility. At each frame and for each node we estimate the probability of being visible. The probability of a node to be visible is inversely proportional to the density of the region between the viewpoint and the node. To estimate the density of the region between a node and the viewpoint we impose a grid over the given dataset. Then, similar to Klosowski and Silva [30, 31], we assign a solidity value to each cell, which is directly proportional to the number of polygons contained within it. The probability that a given cell is invisible is a function that depends on the solidity of the cells which intersect the segment that connects the center of the cell and the viewpoint.

2 Previous Work

We build on previous work in the area of view-dependent simplification and occlusion culling, which we now review.

2.1 View-Dependent Simplification

The vast majority of previous work on geometric simplification for level-of-detail-based rendering has concentrated on computing a fixed set of view-independent levels of detail. At runtime an appropriate level of detail is selected based on viewing parameters. We refer the interested reader to the recent survey and comparison by Cignoni *et al.* [6].

Recent advances on generating multiresolution hierarchies take advantage of the temporal coherence to adaptively refine or simplify the polygonal datasets from one frame to the next in a view-dependent manner. In particular, adaptive levels of detail have been used in terrains by Gross *et al.* [22] and Lindstrom *et al.* [32]. Gross *et al.* define wavelet space filters that allow changes to the quality of the surface approximations in locally-defined regions. Lindstrom *et al.* define a quadtree-based block data structure that provides a continuous level of detail representation, and Duchaineau *et al.* [11] developed real-time optimally adapting meshes (ROAMing) for terrains. However, these approaches provide elegant solutions for terrains and other datasets that are defined on a grid.

Progressive meshes have been introduced by Hoppe [25] to provide a continuous resolution representation of polygonal meshes. Progressive meshes are based upon two fundamental operators – edge collapse and its dual, the vertex split, as shown in Figure 1. A polygonal mesh $\hat{M} = M^k$ is simplified into successively coarser meshes M^i by applying a sequence of edge collapses. One can retrieve the successively higher detail meshes from the simplest mesh M^0 by using a sequence of vertex-split transformations. The sequence $(M^0, \{split_0, split_1, \ldots, split_{k-1}\})$ is referred to as a *progressive mesh* representation.

Xia *et al.* [42] developed the *Merge Trees* as a data structure built upon progressive meshes to enable real-time view-dependent ren-

[*]Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, el-sana@cs.bgu.ac.il.

[†]Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, netaso@cs.bgu.ac.il.

[‡]AT&T Labs-Research, 180 Park Ave., Room D265, Florham Park, NJ 07932, csilva@research.att.com.

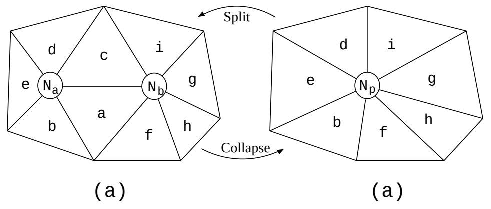


Figure 1: Edge collapse and vertex split.

dering of an object. These trees encode the vertex splits and edge collapses for an object in a hierarchical manner. Hoppe [26] has independently developed a view-dependent simplification algorithm that works with progressive meshes. This algorithm uses the screen-space projection and orientation of the polygons to guide the run-time view-dependent simplifications. Luebke and Erikson [33] define a *tight octree* over the vertices of the given model to generate hierarchical view-dependent simplifications. If the screen-space projection of a given cell of an octree is too small, all the vertices in that cell are collapsed to one vertex.

De Floriani *et al.* [10] have introduced multi-triangulation (MT). Decimation and refinement in MT are achieved through a set of local operators that affect fragments of the mesh. The dependencies between these fragments are then used to construct a directed acyclic graph (DAG) of these fragments. Gueziec *et al.* [23] demonstrate a surface partition scheme for a progressive encoding scheme for surfaces in the form of a DAG. Klein *et al.* [29] have developed an illumination-dependent refinement algorithm for multiresolution meshes. Schilling and Klein [36] have introduced a refinement algorithm that is texture dependent. El-Sana *et al.* [14] have developed Skip Strip, a data-structure that efficiently maintains triangle strips during view-dependent rendering. View-dependent rendering schemes require the existence of the entire dataset in main memory. To overcome the memory size drawback El-Sana and Chiang [5] have developed an external memory view-dependent simplification. Shamir *et al.* [38] have developed a view-dependent approach that handles dynamic environments with arbitrary internal deformation. They have also introduced the T-DAG data structure to support their approach.

2.2 View-Dependence Trees

The *View-Dependence Tree* was introduced by El-Sana and Varshney [15, 16] as a compact multiresolution hierarchical data-structure that supports view-dependent rendering. In fact, for a given input dataset the view-dependence tree construction often leads to a forest (set of trees) since not all the nodes can be merged together to form one tree. View-dependence trees are constructed bottom-up by recursively applying the vertex-pair collapse operation (see Figure 1). The order of the collapses is determined by the simplification metric. This data structure (tree) differs from other previous work [26, 42] in that it enables topology simplification, does not store explicit dependencies, and handles non-manifold cases. At run-time the view-dependence tree is used to guide the selection of the appropriate level of detail based on view-parameters such as view position and illumination.

To enable topology simplification, pair of vertices that are not connected by an edge are allowed to collapse. This allows merging of unconnected components. Such a vertex pair is said to be connected by a *virtual edge*, while the original model edges are referred to as *real edges*. To handle non-manifold cases, a more general scheme is used so that when a vertex split occurs, more than two new adjacent triangles can be added that share the newly created edge (in the case of a manifold each edge is shared by no more than two triangles).

2.3 Visibility Culling Algorithms

Occlusion culling is a technique whose goal is to determine which geometry is hidden from the viewer by other geometry. Such occluded geometry need not be processed by the graphics hardware since it will not contribute to the final image produced on the screen. Occlusion culling, also known as visibility culling, is especially effective for scenes with high depth complexity, due to the large amount of occlusion that occurs. In such situations, much geometry can often be eliminated from the rendering process. Starting with the pioneering work of Airey *et al.* [1] and Teller and Séquin [41] on precomputing lists of potentially visible geometry from cells, visibility culling grew into an important area of research in computer graphics. We refer the interested reader to the recent surveys by Cohen-Or *et al.* [7] and Durand [12].

Roughly speaking, there are several classes of occlusion algorithms (for a more complete classification, see [7]):

– image-precision algorithms, which are generally classified as those which perform visibility computations with image precision, i.e. using depth values [3, 21, 37] or occlusion maps [44];

– object-precision algorithms, which perform visibility computations with object precision, usually by finding a set of large occluders which can be used to determine if other objects (or a hierarchy of objects) in a scene are visible [9, 28];

– and from-region algorithms, which perform visibility computations not for a single viewpoint, but instead compute the set of potentially visible cells for a whole region. These techniques have been the source of considerable research recently [8, 13, 35].

Occlusion culling techniques are usually conservative, producing images that are identical to those that would result from rendering all of the geometry. However, they can also be approximate techniques [20, 30, 31, 44] that produce images that are mostly correct, in exchange for even greater levels of interactivity. The approximate approaches are more effective when only a few pixels are rendered incorrectly, limiting any artifacts that are perceivable to the viewer.

Along the same line as our work, Andujar *et al.* [2] propose a hybrid technique which combines level-of-detail rendering with visibility. In their work, they define the term “hardly visible sets” (HVS), which are subsets of the potentially visible cells that contributes only a small number of pixels to the overall picture. Then, they propose a stratified rendering framework which uses a user-defined error bound to choose from a fixed set of level-of-detail representations based on the HSV error estimates.

Another approach to approximate visibility is based on using a volumetric representation, that is, instead of performing geometric visibility computations, one can compute a volume which has intrinsic properties related to the “density” of geometry in the environment, and approximate the visibility between regions by computing the volume opacity between regions. This approach was pioneered by Sillion [39] in the context of speeding up visibility computations for a radiosity system. It is extended in [40] into a multi-resolution framework. Volumetric visibility was independently developed by Klosowski and Silva [30,31] in their PLP system (see below), where it is used to roughly estimate the order of projection of the geometry.

2.4 Time-Critical Algorithms

Several authors have addressed the problem of achieving constant frame processing of very large datasets for different graphics and geometry algorithms [4, 17, 19, 27, 34, 43].

Funkhouser and Séquin [18] propose an adaptive display algorithm for rendering complex virtual environments. Their rendering algorithm achieves a near constant frame rate by adapting the level of detail of each object by using a constrained optimization algorithm, which tries to generate the "best" image possible within the allowable time budget. In their technique they use the occlusion culling algorithm of Teller and Séquin [41] to avoid rendering geometry which is not visible through the portals. At a high-level our technique can be seen as a finer grain version of their work. There are two main differences:

- we employ view-dependent level-of-detail algorithms which generate smooth transitions, and essentially provides continuous models;
- the occlusion-culling technique we use is only approximate, but it can account for a larger class of occlusion types. In particular the models do not need to be composed of only cells and portals.

The Prioritized-Layered Projection (PLP) algorithm, introduced by Klosowski and Silva [30,31], is an approximate occlusion culling technique. Rather than performing (expensive) conservative visibility determinations, PLP is an aggressive culling algorithm that estimates the visible primitives for a given viewpoint, and only renders those primitives that it determines to be most likely visible, up to a user-specified budget. Consequently, PLP is suitable for generating partially correct images for use in a time-critical rendering system. PLP works by initially creating a partition of the space occupied by the geometric primitives. Each cell in the partition is then assigned, during the rendering loop, a probabilistic value indicating how likely it is that the cell is visible, given the current viewpoint, view direction, and geometry in the neighboring cells. The intuitive idea behind the algorithm is that a cell containing much geometry is likely to occlude the cells behind it. At each point of the algorithm, PLP maintains a priority queue, also called the *front*, which determines which cell is most likely to be visible and therefore projected next by the algorithm. As cells are projected, the geometry associated with those cells is rendered, until the algorithm runs out of time or reaches its limit of rendered primitives. At the same time, the neighboring cells of the rendered cell are inserted into the front with appropriate probabilistic values. It is by scheduling the projection of cells as they are inserted in the front that PLP is able to perform effective visibility estimation.

3 Our Approach

Rendering very large polygonal datasets at interactive rates is one of the fundamental problems of computer graphics. The time spent in rendering a polygonal dataset is (mostly) proportional to the number of triangles sent to the graphics hardware.[*] Therefore, reducing the number of triangles sent to the graphics hardware accelerates the rendering of polygonal datasets. Such reduction should be achieved without noticeable loss of fidelity. Several solutions have been developed to address this problem such as geometric simplification, occlusion culling and image-based rendering.

Level-of-detail rendering and occlusion culling techniques have successfully reduced the number of triangles sent to the graphics hardware while preserving the visual appearance of the rendered 3D polygonal scenes. It is important to note that these two approaches have reduced the number of triangles in two orthogonal ways (refer to section 1).

Our approach is based on integrating these two fundamental approaches – level-of-detail rendering and occlusion culling – into one algorithm in a simple and intuitive fashion. We have developed an algorithm that successfully integrates view-dependent rendering with occlusion culling. Our occlusion culling algorithm is an approximate occlusion culling technique. Instead of performing expensive conservative occlusion culling, our algorithm estimate occlusion probability by using an approximate occlusion culling technique similar to the prioritized-layered projection (PLP) algorithm (refer to section 2). Since the occlusion culling algorithm is integrated in the view-dependent framework, it achieves additional reduction that contributed by low level-of-detail for far-from-viewer regions. Furthermore, the use of view-dependent rendering avoids the generation of "black" regions in PLP resulting of a shortage in triangle budget. Instead of removing hidden triangles, as in typical occlusion culling algorithms, our algorithm reduces the level of detail in different regions proportional to the occlusion probability of these regions.

[*]In some cases, the rasterization phase can also be the bottleneck (e.g., volume rendering based on texture mapping), and if a number of very large polygons are sent to the graphics hardware the rasterization cost can potentially limit the frame rate.

4 Estimating Visibility

Our approach is based on estimating visibility of a region from a given viewpoint. Such approximation should be as accurate as possible with the time limit between two consecutive frames at interactive rates. It is important to note that we have chosen to approximate visibility because it is quite hard (and at this time computationally intractable) to compute exact visibility within the given limited time.

For a given viewpoint p and a region r we compute an occlusion factor between 0.0 and 1.0 that determines the degree of occlusion of the region r with respect to the viewpoint p. To determine the occlusion factor we first impose a three-dimensional uniform grid over the bounding box of the given scene. The dimensions of the grid depend on the depth complexity of the scene. Then we compute visibility of a cubic cell from a given viewpoint in way similar to computing opacity of cell in volumetric representation. Therefore, for each cell, which includes triangles, we need to define an attribute equivalent to opacity in volumetric representation. Similar to Klosowski and Silva [31] we shall refer to this attribute as *solidity of a cell*.

4.1 Solidity of a Cell

We would like to define a solidity of a cell as an indicator for how this cell occludes other cells of the imposed 3D grid. It is obvious that the more primitives contained within the cell the more it may occlude other cells. Therefore, we use clutter as the basic factor in determining solidity of a cell.

Let us consider shooting a ray r through the cell c_i. Since we are considering opaque polygons a ray r may or may not leave the cell c_i. If a ray r that enters the cell c_i manages to leave it we say that r *passes* c_i. Determining whether a ray *passes* a cell could be very expensive and may not be feasible within a duration of one frame for large datasets. Therefore, we compute a probability that a ray passes a cell instead of exact expensive determination.

The *solidity* of a cell is inversely proportional to the probability that any ray passes that cell. Since our approach deals with static scenes we have chosen to compute solidity in a preprocessing stage. Therefore, we do not have the time limits constraints as in real-time interaction.

A ray r passes a cell c_i if it does not hit any polygon inside the cell c_i. Using this principle we have developed two approaches to compute the solidity of a cell. One is based on projecting the polygons inside the cell on its sides; and the other is based on shooting ray across the cell. Next we shall describe these two approaches:

- *Face projection.* This technique is based on projecting polygons on the six faces of a cell. We refer to the six images created by projecting a polygon on the six faces of a cell as

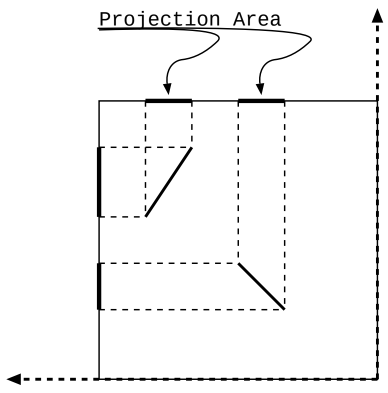


Figure 2: Computing solidity using orthogonal projection: two faces of a 2D cell.

the *projection area* of a polygon. Then we compute the solidity of a cell as the ratio between the union of the projection areas of all polygons inside a cell and the total area of the six faces of the cell. Since we are using parallel projection, it is enough to project each polygon on three orthogonal faces and then compute the solidity with respect to the selected faces. If $poly(c_i)$ is the set of polygon inside a cell c_i, $face(c_i)$ is the set of the three orthogonal faces of c_i, and $proj\_area(t)$ is the *projection area* of a polygon t then

$$solidity(c_i) = \frac{\bigcup_{t \in poly(c_i)} proj\_area(t)}{\sum_{f \in face(c_i)} area(f)} \qquad (1)$$

– *Ray shooting.* In this technique we shoot a number of rays into a cell and compute the number of rays that pass the cell. Then we compute the solidity as the ratio between the number of rays that do pass the cell and the total number of rays (the "shot rays") as in Equation 2. To provide a meaningful solidity value for a cell we would like to shoot rays in a uniform fashion. Therefore we bound the cell we intend to compute its solidity value with a sphere, virtually subdivide the surface of the sphere into uniform regions, and then shoot one random ray through each region. As result of the symmetry of spheres, it is enough to shoot rays for half of the bounding sphere.

$$solidity(c_i) = \frac{Shot\ Rays - Passed\ Rays}{Shot\ Rays} \qquad (2)$$

$$= 1.0 - \frac{Passed\ Rays}{Shot\ Rays} \qquad (3)$$

4.2 Visibility of a Cell from Viewpoint

We would like to estimate visibility from a viewpoint. Therefore, we need to estimate the probability for a vertex v_i to be visible from a given viewpoint p. Our estimation is based on the solidity of the cells between the viewpoint p and the vertex v_i. In the first step of our algorithm we determine the set of cells s_c that contribute to computing the visibility of the vertex v_i from the viewpoint p. We define the set s_c as the cells that intersect the line segment that connects the viewpoint p with the vertex v_i. Computing the visibility of a cell from a viewpoint was inspired by rendering volume data.

The rendering of a volume data can be done by traversing a ray from viewpoint into the volume dataset in a front-to-back or back-to-front fashion. In back-to-front rendering, each voxel occludes

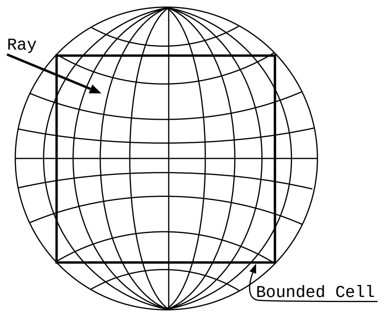


Figure 3: Computing solidity by shooting ray through bounding sphere.

the preceding one in proportion to its color and opacity. Opaque voxels contribute more to the final pixel than the transparent ones. Front-to-back traversal uses the same basic process but traverses the voxels from the viewpoint to the voxels. The benefit of the front-to-back traversal is that once the maximum opacity for that pixel is reached, the traversal process can stop without traversing the rest of the dataset.

In our approach we treat solidity as opacity value and use front-to-back traversal to compute the visibility (occlusion) of a cell. We initialize the occlusion probability to 0.0, then we traverse the cells of the set s_c in order from the viewpoint $p(= v_0)$ to the vertex v_i. For each visited cell we computed the accumulating solidity along the segment $\overline{pv_i}$ (as shown in Equation 4). The contribution of a visited cell to the solidity along a segment depends on the following two factors:

(1) The solidity ρ of the visited cell: Cells with high solidity have higher probability to occlude cells behind it.

(2) The portion of the segment $\overline{pv_i}$ that lies within that cell: cells that include longer portions contribute more than cells that include smaller fractions of the segment. Each segment $\overline{v_{i-1}v_i}$ contributes a certain "transparency", which is used for attenuation:

$$t_i = (1 - \rho(c_i))e^{-\triangle \overline{v_k v_{k+1}}},$$

where $e^{-\triangle \overline{v_k v_{k+1}}}$ is normalized by the maximum cell diagonal, so $t_i \le 1$. (Note that if a cell has solidity equal to one, its transparency is zero.)

$$occlusion\_probability(v_i) = \sum_{j=0}^{i} \rho(c_j) \prod_{k=0}^{j-1} t_k \qquad (4)$$

5 Adaptive Level of Detail

Once the view-dependence trees have been constructed off-line, it is possible to construct an adaptive level-of-detail mesh representation at run-time. Real-time adaptive mesh reconstruction requires the determination of a list of vertices and the list of triangulation amongst them. We refer to the vertices selected for display at a given frame as *display vertices list* and triangles for display as *display triangles list*. The determination of the display lists is controlled by the selection of the *active nodes* of the view-dependence trees. The list of *active nodes* is a subset of nodes of the view-dependence trees that forms a breadth cut of the view-dependence trees as shown in figure 4. In the active nodes list, low resolutions (in term of triangles) are associated with nodes that are close to the top of the tree and high resolutions are associated with the nodes that are close to the bottom of the trees.

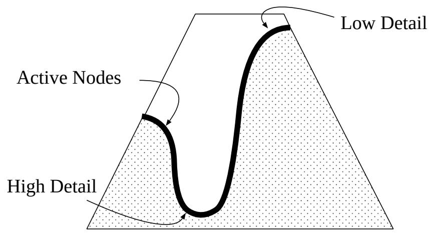


Figure 4: Active nodes list in view-dependence trees.

The selection of active nodes is based on view-parameters and occlusion culling which are mainly determined by the viewpoint. On each change of the view-parameters we scan the active nodes list. For each visited node we determine whether a node requires an increase, reduction, or no change on its resolution. We increase the resolution for a node by performing *split* operation, and reduce the resolution by performing *merge* operation. To simplify the explanation, we shall refer to the no-change operation as *nop*. The two adapt operations *split/merge* change the *active nodes* list that influences the *display vertices* and *display triangles* lists.

The split operation involves removing the node from the active nodes list and inserting its two children into the active nodes list. In addition, we need to update the active triangles list, by inserting the newly created adjacent triangles due to this split. The added triangles are obtained by looking at the adjacent triangle lists stored in the two children nodes (refer to [15]). The merge operation is the dual operation, and it also requires an update of the active nodes list accordingly.

5.1 View-Parameter Influence

In view-dependent rendering, we would like to represent objects with respect to their visual importance in the final rendered image. Except for silhouettes, we would like to represent regions in a resolution (in terms of polygons count) on the 3D space proportional to their contribution in the final rendered image. Hence, regions that are close-to-viewer and visible would enjoy high level of detail while occluded or far-from-viewer regions would be represented in low resolution.

The decision on the next adapt operation (split, merge, or nop) for a node depends on the current resolution of the node, the view-parameters, and occlusion probability of the node with respect to the current viewpoint. We compute the contribution of view-parameters by considering the current viewpoint and the set of current light sources. Each factor (viewpoint or light source) determines appropriate level of detail that is computed by taking into account distance from the node and the difference between its direction (view-direction, or light direction) and the normal of the inspected view-dependence node.

The target level of detail for a node is the lowest level of detail among the various level of detail computed by viewpoint and light sources. In our level-of-detail determination scheme, regions that are far from the viewpoint have lower level of detail, and close regions have higher level-of-detail. The difference between the view-direction (or light direction) and the node normal results in high resolution for regions facing the viewer and coarse level of detail for back-facing regions. Silhouette regions are handled separately.

5.2 Level of Detail Selection

The selection of level of detail that is based on view-parameters assigns high resolution to close nodes that are not visible from the current viewpoint. To avoid such serious drawback we incorporate occlusion culling in the above scheme. The occlusion-culling factor cannot increase resolution; it can only reduce it. This is an immediate result of the fact that occluded-close regions are represented in more triangles than their contribution to the final rendered image. Therefore, we determine the level of detail at a node by computing its view-parameters contribution and estimate its occlusion probability. For occlusion probability 0.0, the selected level of detail is the one determined by the view-parameters, and for occlusion probability 1.0 we would like to select the lowest level of detail possible. Hence, the probability value can reduce the level of detail at a node from the level of detail determined by the view-parameters to the lowest possible one as expressed in Equation 5.

Note that in Equation 5 LOD_{final} is the final level of detail, LOD_{view} is the level of detail determined by view parameters, LOD_{lowest} is the lowest possible level of detail, and OP is the occlusion probability at the inspected node.

$$LOD_{final} = (1 - OP) * LOD_{view} + OP * LOD_{lowest} \quad (5)$$

We have found out that even though our occlusion culling is neither exact nor conservative, we have achieved very high fidelity while greatly reducing the number of triangles. This is a result of the use of dependences among nodes on the view-dependence trees. These dependences (implicit and explicit) were introduced to prevent fold-overs in run-time level-of-detail selection by preventing the split or merge of nodes before others. Hence, these dependencies often restrict the refinement of nodes, which might have otherwise refined to comply with the visual fidelity or error metric. Such behavior seems to improve our occlusion probability approach as result of:

- Dependencies prevent one node that inaccurately assigned high occlusion probability to select very low resolution.
- We give high priority for high resolution to overcome inaccuracy in computing the occlusion probability.

6 Implementation Details

We have implemented our algorithm in C/C++ on Unix (and GNU/Linux) operating systems. In our current implementation we have carefully selected efficient data structures, but have not fully optimized our code.

Our approach is aimed at rendering large three-dimensional datasets. Therefore, we reduce the memory requirements of view-dependence trees by assigning the geometry of a parent node to one of its children geometry. Such scheme of the view-dependence trees stores geometry information such as vertex position, normal, and color on the leaves of the tree, and internal nodes keep pointers to appropriate addresses where their geometry is stored. These changes reduce the overhead of storing the view-dependence tree to only $2/3$ times more than the 3D representation of the original mesh.

We compute the occlusion probability by traversing the line segments that connect the viewpoint and the inspected node. We have implemented an algorithm similar to Bresenham's algorithm, which is based on integer increments. The algorithm starts the traversal from the viewpoint, and proceeds to the inspected node. It stops when the occlusion probability value exceeds 1.0 or it has visited all the cells intersecting the traversed segment. In our current implementation, we compute the occlusion probability along the segment that connects the center of the cell that includes the viewpoint and the center of the cell that include the inspected node. In addition, we assign the same occlusion probability for the view-dependence nodes that belong to the same cell. We have also managed to reduce the time spent on estimating occlusion probability by avoiding unnecessary re-computation. We achieve this by using a one-byte counter for each cell. We update this counter to be equal to the current frame counter after each time we compute the occlusion probability for a cell. We recompute occlusion probability for a cell only if its one-byte counter is different from the current frame counter.

7 Results

We have tested our unoptimized implementation on various datasets and have received encouraging results. In this section we report and analyze some of these results.

View-dependent rendering algorithms are known for their ability to take advantage of coherence between consecutive frames. Hence, it makes more sense to compare its performance over sequences of frames and not over one isolated frame. Therefore, all the results we report were obtained using a sequence of frames.

The results were achieved using a Pentium-III machine with 512 MB of memory and an Nvidia GeForce graphics card. In Table 1 we report the average number of split/merge operations, number of triangles at each frame, and the average time for a frame. As can be seen, we have achieved interactive frame rates even for large datasets. To evaluate the improvement of integrating occlusion culling with view-dependent rendering we have performed tests that compare the performance of our current algorithm (view-dependent rendering that supports occlusion culling) with previous view-dependent rendering algorithm [15]. In these tests we consider two factors – image quality and frame rates.

For each dataset we record a path which includes the camera parameters and illumination. Then, we compare the performance of the two algorithms over the same path for each dataset. For each dataset, we compare the average frame rates and the average quality of the two algorithms. It is easy to compute the frame rate after measuring the run time of the entire sequence. Measuring the image quality is not that simple. For each frame, the image quality is determined by the set of rendered triangles which are constructed from the selected level-of-detail. For the same simplification metric, the camera parameters and illumination determine the selected level-of-detail. Comparing the image quality could be done by either comparing the two images pixel-to-pixel or by comparing the set or triangles that contribute to the rendered images. We chose to use the second approach because comparing the images pixel-to-pixel may not be accurate as result of the floating-point calculation during the rendering process. In the last three columns of Table 1 we report results of comparing the average frame time and image quality for the two algorithms using different datasets. The column titled VDR includes the average time per frame for view-dependent rendering (without occlusion culling), and the column VDR+OC includes the average time per frame for view-dependent rendering using occlusion culling to reduce the number of rendered triangles. For these results we have used the first approach – *Face projection* – to approximate the solidity of the cells. The time to estimate the occlusion probability (in real-time) depends primarily on the size of the imposed 3D grid.

Figure 8 shows three images (a, b, and c) of a four spheres dataset. Figure 8a shows the four-spheres dataset, Figure 8b shows the solidity values of each cell represented by the size of the *blue* points, and Figure 8c depicts the occlusion probability along the segment $\overline{AB}$. The saturation of the *red* color indicates the occlusion probability value. We have assigned *red* color for occlusion probability 0.0 and *black* for value 1.0. As can be seen in these images cluttered cells were assigned large solidity values (larger blue points in Figure 8b) and occluded cells were assigned large occlusion probability values (black segment color in Figure 8c).

Figure 6 depicts the influence of occlusion probability on the selected level of detail. Figure 6 shows two images of nested spheres. Figure 6a shows the final images and Figure 6b shows the outer sphere in wire-frame in order to show the inner sphere. In Figure 6b we depict the reduction of the resolution of the occluded regions.

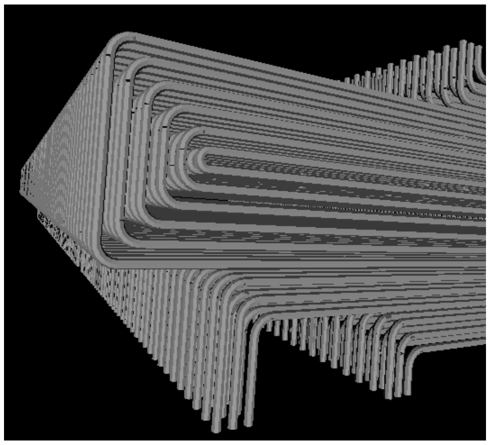


Figure 5: A close view of the model Sect01A (Power Plant).

The images in the Figures 7, 9, and 10, and compare the performance of view-dependent rendering, in terms of resolution, with and without the use of occlusion culling. The top row in Figure 9 shows three images of the Car Engine model from the same viewpoint: (a) full resolution, (b) view-dependent rendering, and (c) view-dependent rendering with occlusion culling. In the bottom row we show a side view of each instance in wire-frame mode to depict the triangle reduction. These images show that we gain a considerable reduction in the number of triangles by integrating occlusion culling with view-dependent. Note that the two top images Figure 9b and 9c differ in less that 0.05% of the visible triangles. Figure 10 shows images of the Section01A model (of the Power Plant dataset [24]). These images were generated in the same approach that we use to produce the images in Figure 9. The difference between the top images in Figure 10b and 10c is less that 0.1% of the visible triangles. Since the images in Figure 10 do not depict well the complexity of the Section01A model, we show it in a better detail in Figure 5. Figure 7 shows full resolution, view-dependent rendering, and view-dependent rendering with occlusion culling of Section16 of the Power Plant model.

8 Conclusion and Future Work

We have presented a novel approach for incorporating occlusion culling within the framework of view-dependent rendering. Our approach manages to reduce the number of triangles sent to the graphics hardware without compromising image quality. Our idea is based on estimating occlusion probability rather than performing expensive conservative visibility determinations. To approximate visibility we impose a 3D uniform grid over the bounding box of the scene. Then we assign a solidity value for each grid-cell based on the polygons included in that cell. We compute the occlusion probability for a node by traversing the cells along the segment that connects the node with the viewpoint. The computed occlusion probability and the view-parameters are used to determine the appropriate level of detail for each frame.

We see the scope for future work in designing occlusion culling algorithms that can take advantage of coherence between consecutive frames. Such algorithms can provide better performance when incorporated into view-dependent rendering framework.

Our current techniques for approximating visibility are quite simple, and are likely not optimal ones. Also, there are several approximations that we are performing in the calculations which do not

Dataset	Original		VDR+OC Average/frame		Average Time(ms)/frame		Image Quality
	Vertices	Triangles	Triangles	Split/Merge (K)	VDR	VRD+OC	Δ Triangles(%)
Hiding Bunny	38 K	75 K	30 K	1.4 K	55	39	0.0
Car Engine	70 K	140 K	52 K	2.1 K	93	51	0.23
Geometry Objs	95 K	190 K	63 K	2.4 K	91	54	0.25
Section16	597 K	366 K	69 K	2.9 K	96	52	0.5
Terrain	262 K	522 K	71 K	3.1 K	110	61	0.01
DragonTeam	320 K	650 K	74 K	4.8 K	115	64	0.5
Section01A	507 K	748 K	81 K	5.1 K	120	75	0.3

Table 1: Run time over a sequence of frames for various datasets by different view-dependent rendering algorithms.

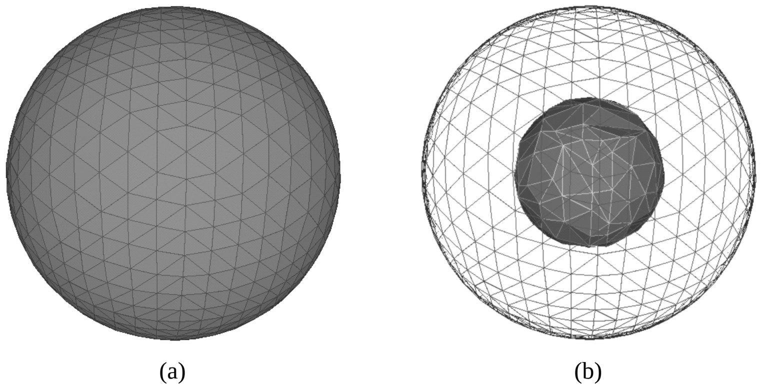


Figure 6: Occlusion probability contribute to the lower resolution of the inner sphere.

necessarily capture the correct physical phenomena. An interesting avenue for future work is to develop more accurate techniques for approximating visibility.

Acknowledgements

Thanks to Wagner Corrêa for useful criticisms. We are grateful to the Walkthru Group at the University of North Carolina for making the Power Plant Model available.

References

[1] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, No. 2, pages 41–50, March 1990.

[2] C. Andújar, C. Saona-Vázquez, I. Navazo, and P. Brunet. Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum*, 19(3):499–506, August 2000.

[3] D. Bartz, M. Meiner, and T. Huttner. Extending graphics hardware for occlusion queries in opengl. In *1998 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 97–104, August 1998.

[4] S. T. Bryson and S. Johan. Time management, simultaneity and time-critical computation in interactive unsteady visualization environments. In *IEEE Visualization '96*, pages 255–262, October 1996.

[5] J. El-Sana and Y.-J. Chiang. External memory view-dependent simplification. In *Computer Graphics Forum*, volume 19, pages C139–C150, 2000.

[6] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, February 1998.

[7] D. Cohen-Or, Y. Chrysanthou, C. Silva, and G. Drettakis. Visibility, problems, techniques and applications. Siggraph 2000 course notes, ACM, 2000.

[8] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):C243–C253, 1998.

[9] Y. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *1997 Symposium on Interactive 3D Graphics*, pages 83–90, April 1997.

[10] L. De Floriani, P. Magillo, and E. Puppo. Efficient implementation of multi-triangulation. In *IEEE Visualization '98*, pages 43–50, October 1998.

[11] M. Duchaineau, M. Wolinsky, D. Sigeti, M. Miller, C. Aldrich, and M. Mineev-Weinstein. ROAMing Terrain: real-time optimally adapting meshes. In *IEEE Visualization '97*, pages 81–88, 1997.

[12] F. Durand. A multidisciplinary survey of visibility (included in [7]). Technical report, 2000.

[13] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000.

[14] J. El-Sana, E. Azanli, and A. Varshney. Skip strips: Maintaining triangle strips for view-dependent rendering. In *IEEE Visualization '99*, 1999.

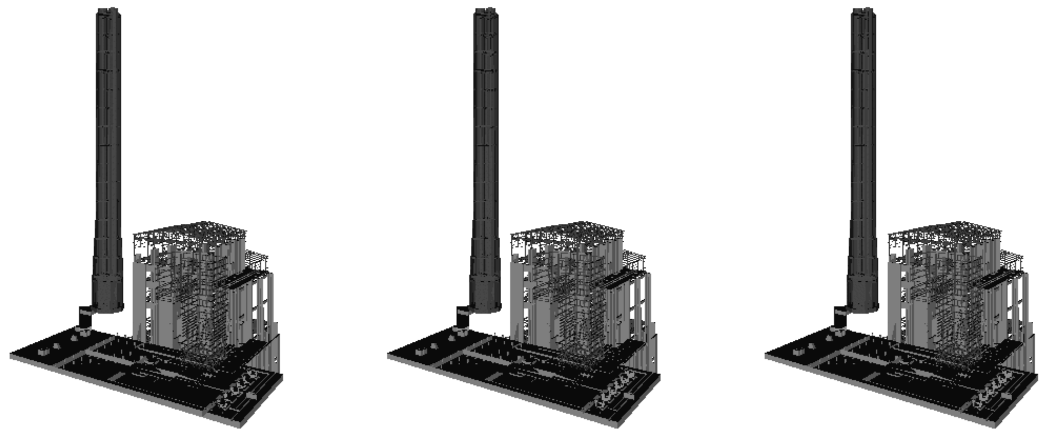
[15] J. El-Sana and A. Varshney. Generalized view-dependent simplification. In *Computer Graphics Forum*, volume 18, pages C83–C94, 1999.

[16] J. El-Sana and A. Varshney. Parallel construction and navigation of view-dependent virtual environments. In *Conference on Visual Data Exploration and Analysis*, pages 62–70. SPIE Press, January 1999.

[17] T. Funkhouser, P. Min, and I. Carlbom. Real-time acoustic modeling for distributed virtual environments. In *Proceedings of SIGGRAPH 1999*, pages 365–374, August 1999.

[18] T. Funkhouser and C. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of SIGGRAPH 1993*, pages 247–254, August 1993.

[19] E. Gobbetti and E. Bouvier. Time-critical multiresolution rendering of large complex models. *Computer-Aided Design*, 32(13):785–803, 2000.



(a) 366K triangles (b) 296K triangles (c) 162K triangles

Figure 7: Section01A of Power Plant: a) Full resolution model, b) View-dependent rendering, and c) View-dependent rendering with occluding culling support.

[20] N. Greene. A quality knob for non-conservative culling with hierarchical z-buffering. Technical Report, 2001.

[21] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993.

[22] M. H. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing. In *IEEE Visualization '95*, pages 135–142, 1995.

[23] A. Guéziec, F. Lazarus, G. Taubin, and W. Horn. Surface partitions for progressive transmission and display, and dynamic simplification of polygonal surfaces. In *Proceedings VRML 98, Monterey, California, February 16–19*, pages 25–32, 1998.

[24] UNC Chapel Hill. Power plant dataset, 2001. http://www.cs.unc.edu/˜geom/Powerplant/.

[25] H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH '96*, pages 99 – 108. ACM SIGGRAPH, ACM Press, August 1996.

[26] H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH '97*, pages 189 – 197. ACM Press, August 1997.

[27] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[28] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *In Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 1997.

[29] R. Klein, A. Schilling, and W. Straßer. Illumination dependent refinement of multiresolution meshes. In *Computer Graphics Intl*, pages 680–687, June 1998.

[30] J. T. Klosowski and C. T. Silva. Rendering on a budget: A framework for time-critical rendering. In *IEEE Visualization 99*, pages 115–122, 1999.

[31] J. T. Klosowski and C. T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, 2000.

[32] P. Lindstrom, D. Koller, W. Ribarsky, L. Hughes, N. Faust, and G. Turner. Real-Time, continuous level of detail rendering of height fields. In *SIGGRAPH 96 Conference Proceedings*, pages 109–118, August 1996.

[33] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of SIGGRAPH '97*, pages 198 – 208. ACM SIGGRAPH, ACM Press, August 1997.

[34] A. Reisman, C. Gotsmann, and A. Schuster. Parallel progressive rendering of animation sequences at interactive rates on distributed-memory machines. In *IEEE Parallel Rendering Symposium*, pages 39–48, November 1997.

[35] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. In *Proceedings of SIGGRAPH 2000*, pages 229–238, July 2000.

[36] A. Schilling and R. Klein. Graphics in/for digital libraries — rendering of multiresolution models with texture. *Computers and Graphics*, 22(6):667–679, 1998.

[37] N. Scott, D. Olsen, and E. Gannet. An overview of the visualize fx graphics accelerator hardware. *The HewlettPackard Journal*, pages 28–34, May 1998.

[38] A. Shamir, V. Pascucci, and C. Bajaj. Multi-resolution dynamic meshes with arbitrary deformation. In *IEEE Visualization '2000*, pages 423–430, 2000.

[39] F. X. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):240–254, September 1995.

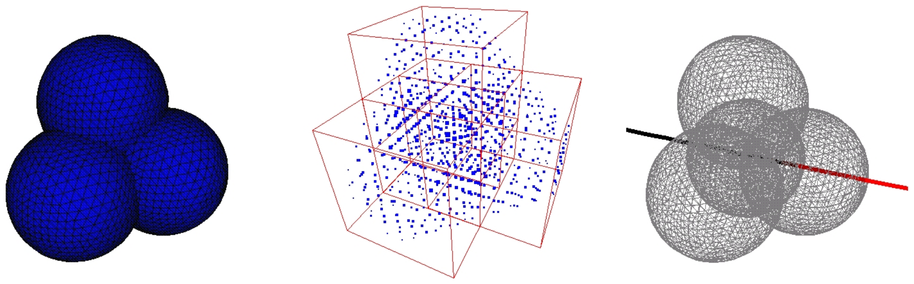
[40] F. X. Sillion and G. Drettakis. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. *Proceedings of SIGGRAPH 95*, pages 145–152, 1995.

[41] S. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics: Proceedings of SIGGRAPH'91*, 25, No. 4:61–69, 1991.

[42] J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, pages 171 – 183, June 1997.

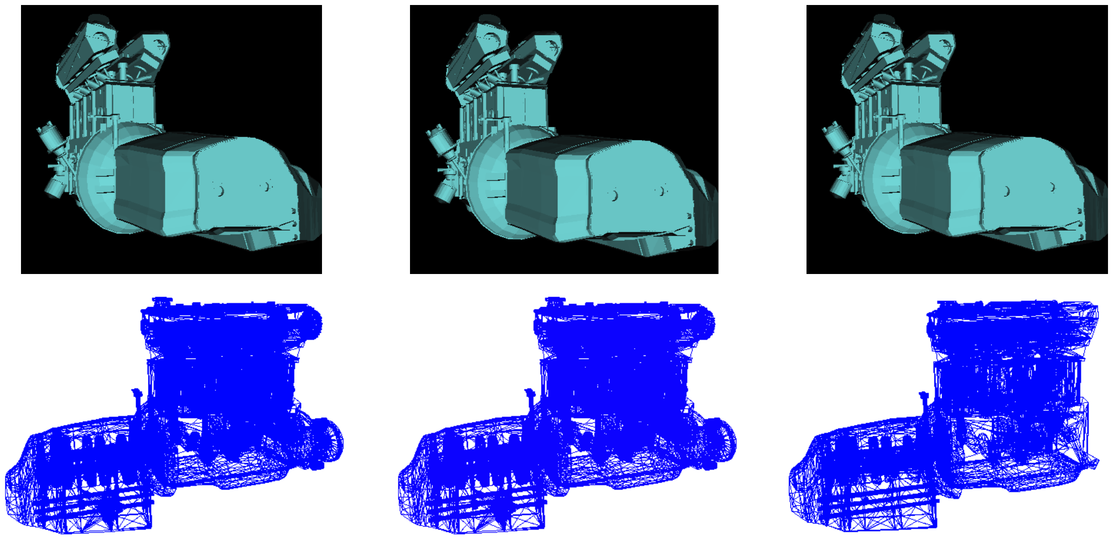
[43] P. Yuan and H. Sun. P-buffer: a hidden-line algorithm in image-space. *Computers & Graphics*, 21(3):359–366, 1997.

[44] H. Zhang, D. Manocha, T. Hudson, and K. Hoff III. Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH 1997*, pages 77–88, August 1997.



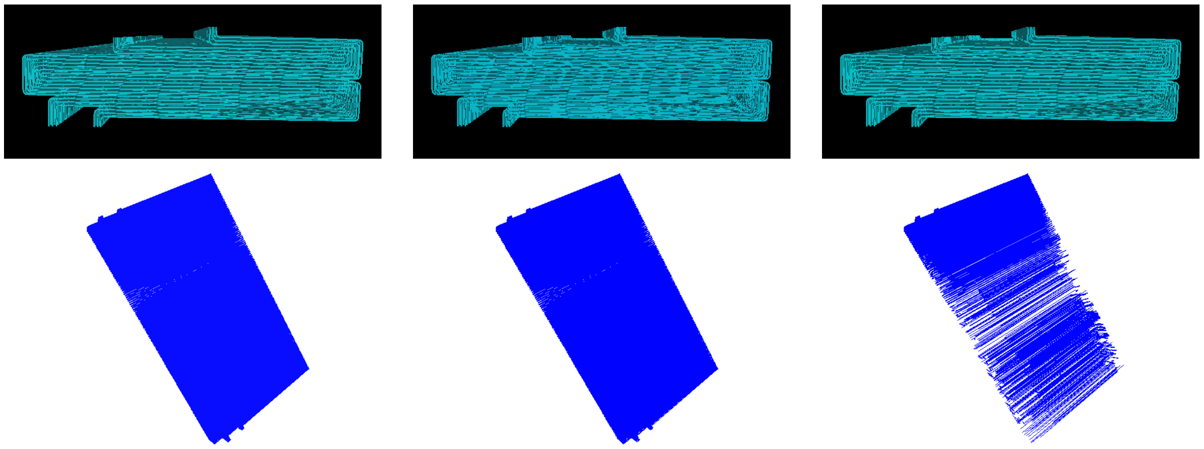
(a) (b) (c)

Figure 8: Occlusion probability along a line that passes through four spheres scene.



a) 140K triangles (b) 86K triangles (c) 24K triangles

Figure 9: Car Engine: a) Full resolution model, b) View-dependent rendering, and c) View-dependent rendering with occluding culling support.



(a) 748K triangles (b) 474K triangles (c) 66K triangles

Figure 10: Section01A of Power Plant: a) Full resolution model, b) View-dependent rendering, and c) View-dependent rendering with occluding culling support.