Volume xx (200y), Number z, pp. 1–10

Occluder Simplification using Planar Sections

Ari Silvennoinen[†1,2,4], Hannu Saransaari[1], Samuli Laine[3] and Jaakko Lehtinen[2,3]

[1]Umbra Software Ltd. [2]Aalto University [3]NVIDIA Research [4]Remedy Entertainment Ltd.

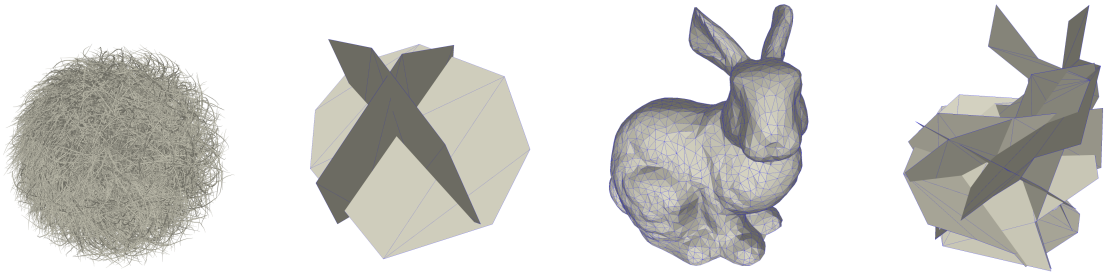


Figure 1: Left: HAIRBALL and simplified occluder with 16 triangles (0.0005% of original). Right: BUNNY and simplified occluder with 64 triangles (1.3% of original).

Abstract

We present a method for extreme occluder simplification. We take a triangle soup as input, and produce a small set of polygons with closely matching occlusion properties. In contrast to methods that optimize the original geometry, our algorithm has very few requirements for the input—specifically, the input does not need to be a watertight, two-manifold mesh. This robustness is achieved by working on a well-behaved, discretized representation of the input instead of the original, potentially badly structured geometry. We first formulate the algorithm for individual occluders, and further introduce a hierarchy for handling large, complex scenes.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

Keywords: visibility, occlusion, geometry simplification, rendering

1. Introduction

Real-time rendering of modern, massive scenes requires sophisticated occlusion culling methods for avoiding unnecessary rendering of geometry that does not affect the final image. This is typically achieved through a combination of various techniques.

For static environments, precomputed potentially visible sets (PVS) [ARB90] and portals [LG95] can be used for determining objects and regions that are not visible to the camera. A PVS may efficiently model visibility relationships between large regions of the scene, but in addition to being restricted to static occluders, it is generally overly conservative compared to exact from-point visibility. Portals are a practical solution for indoor scenes only, where different regions are visible to each other through narrow openings.

Modern, dynamic from-point visibility algorithms, e.g., [AM04, BWPP04], operate on perspective-projected geometry in image space, and are thus able to capture the complex and unpredictable occlusion caused by objects of any scale. Recent algorithms are based on the

† ari.silvennoinen@aalto.fi, hannu@umbrasoftware.com, slaine@nvidia.com, jlehtinen@nvidia.com

observation that although output resolutions are high and geometries complex, it often suffices to perform occlusion computations at a coarse resolution and using approximate geometry [Val11, Per12]. This brings up the need to perform visibility-aware geometry simplification.

The number of potentially falsely occluded or unoccluded pixels depends on the projected size of the simplified occluder. Therefore, it is desirable to use a different simplification depending on the distance between the occluder and the camera—a coarse approximation with few triangles is generally sufficient for distant occluders. In addition, the apparent topology of the occluder may vary with distance, given that our goal is to bound the error in pixels. For example, consider a box with a small hole in one of the faces. When viewed up close, the hole needs to remain unoccluded, but at distance we may consider the box to be solid as the size of the hole shrinks below error threshold. This may allow the simplification to retain the occlusion characteristics with fewer triangles. Our method naturally takes advantage of this phenomenon through the discretization step, where voxel resolution determines the allowable error.

To constrain the amount of potential overocclusion, we want the simplification method to be error-bounded in the sense that the maximum pointwise distance between the simplified model and the original model can be controlled. Previous error-sensitive simplification methods (e.g., [CVM*96, Hop96, GH97]) are constrained to operate on the mesh surface and are therefore not well suited for extreme simplification. In our method, the maximal simplification error is governed by the discretization resolution.

Our simplification algorithm is based on the following two observations. First, every 2D interior slice of a solid object is a conservatively correct occluder in the sense that the rays occluded by the slice are a subset of the rays occluded by the original model. Second, we can form an error-bounded solid representation even for non-solid input geometry through discretization. This gives us considerable freedom in generating extremely simplified occluders by analyzing slices of the input geometry without sacrificing the property of error-boundedness. Due to the relaxed requirements for the input geometry, the method can be extended in a straightforward fashion to hierarchical occluder generation, where multiple objects are simplified together to produce a common occluder. This decouples the algorithm from the object partitioning used in the scene.

2. Related Work

Error-sensitive simplification. Cohen et al. [CVM*96] present an error-bounded simplification algorithm which applies iterative mesh simplification operators restricted to an error-bounding *simplification envelope* around the original mesh surface. Law and Tan [LT99] generate virtual occluders by taking the output of a mesh simplification algorithm such as simplification envelopes and moving edges which would cause false occlusion as a post-process. However, both methods require mesh connectivity and they are not suitable for extreme simplification due to the envelope constraints.

Quadric error metric based simplification by Garland and Heckbert [GH97] and its variants [LT00, ZT02], as well as progressive meshes by Hoppe [Hop96], operate similarly on the input mesh, iteratively applying simplifications that minimize the introduced error in each step. However, the occlusion properties of the simplified mesh may deviate considerably from the input, especially for extremely small triangle budgets. For example, these methods are eager to cover the central hole of a fat torus before applying other simplifications that would make more sense in the occlusion context.

Decoret et al. [DDSD03] build a simplified model by fitting a set of tangent planes to the original mesh and using texture maps to capture fine geometric and shading detail. The key difference to our method is that the tangent planes in their method are restricted to lie near the surface of the input geometry, whereas we consider all possible planes that intersect the input geometry.

Virtual occluders. Coorg and Teller [CT97] and Wonka and Schmalstieg [WS99] use a subset of the input polygons as occluders. The occluder polygons are chosen based on a combination of surface area and optionally angle and hence the occluder polygons are sensitive to the input geometry.

Germs and Jansen [GJ01] build simplified occluders in 2.5D scenes by detecting 2D floor plans and extruding them vertically. Koltun et al. [KCCo00] use a geometric construction based on supporting and separating planes to build virtual occluders. However, both of these methods are difficult to generalise to a 3D setting.

Voxel-based simplification. Nooruddin and Turk [NT03] use an intermediate voxel representation to perform simplification operations such as erosion and dilation, potentially introducing changes in topology, before converting back to polygonal domain. Similar to us, they take advantage of the voxel representation to build simplified models with different levels of detail. Because the voxel representation is converted back to polygonal domain, they have to rely on a polygonal simplification method to produce the final output mesh. The method therefore still operates fundamentally on the surface of the input mesh, and shares the disadvantages brought by final mesh simplification step.

The Oxel system [Dar11] voxelizes the scene into an octree and heuristically classifies each octree node as either inside or outside. The method then generates occluders by inserting seed boxes into the inside voxels and iteratively expands them in axis-aligned directions.

Visibility analysis. Dachsbacher [Dac11] considered the structure of visible and occluded regions across object surfaces under different viewing configurations whereas our analysis is not restricted to object surfaces.

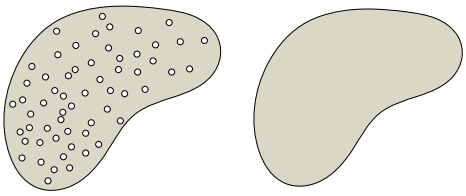


Figure 2: The two objects have equal surface area but different occlusion charasteristics.

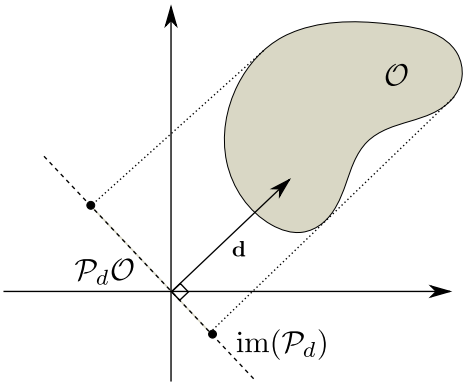


Figure 3: The orthogonal projection operator $\mathcal{P}_d : R^3 \mapsto \mathbb{R}^2$ projects the object $\mathcal{O}$ to the image plane $\mathrm{im}(\mathcal{P}_d)$ in direction $\mathbf{d}$.

3. Measuring Occlusion

In order to produce efficient occluders through optimization, we need to quantify occlusion power in a principled manner. While it is well known that surface area directly measures the power of an object to block sets of individual rays, we argue that it does not encode all aspects of a good occluder. Intuitively, the surface area neglects coherence — *clearly, a good occluder blocks large, contiguous sets of rays*. An extreme example is given by cutting holes in a solid surface: in the limit, where the holes become small and densely packed, the surface becomes effectively semi-transparent and loses all its occlusion power, even though the surface area will have shrunk by some constant factor only. Such a mesh blocks many individual rays, but almost no beams of finite radius, and is a useless occluder (cf. Figure 2).

We describe a novel occlusion measure that generalizes the surface area to account for how effectively the object blocks *beams of rays with varying radii*. To maintain scale-independence, we consider beams of all possible sizes. Efficient computation is enabled by the use of a fast Euclidean distance transform. Our measure directly corresponds to the above intuition: comparing two occluders, we favor the one that has, overall, a more "compact" shape, less holes, and larger local feature size.

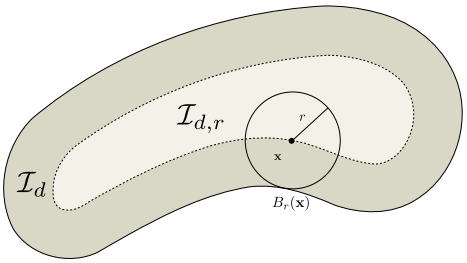


Figure 4: The set $\mathcal{I}_{d,r} \subset \mathcal{I}_d$ is obtained by erosion with a ball B_r of radius r.

3.1. Total Occlusion Measure

We begin by considering occlusion caused by the object for a group of parallel rays of direction $\mathbf{d}$. Let $\mathcal{P}_d : \mathbb{R}^3 \mapsto \mathbb{R}^2$ be the orthogonal projection operator in direction $\mathbf{d}$ and $\mathcal{I}_d = \mathcal{P}_d\mathcal{O} \subset \mathrm{im}(\mathcal{P}_d)$ be the orthogonal projection of the object $\mathcal{O}$ (Figure 3), i.e., a binary image that encodes the projected shape of the object as seen from direction $\mathbf{d}$. The area of the projection $\mathcal{I}_d$ corresponds directly to the number of parallel rays blocked by the object.

To extend the above to larger beams, let $B_r(\mathbf{x})$ be a 2D disk of radius $r \geq 0$ and center $\mathbf{x} \in \mathrm{im}(\mathcal{P}_d)$. Clearly, the disk directly corresponds to a cylinder of rays: $\mathcal{P}_d^{-1}B_r(\mathbf{x})$ is the set of 3D lines that project to the disk, and the beam is blocked by the object if and only if the disk $B_r(\mathbf{x})$ is fully contained in $\mathcal{I}_d$.

We observe that the number of blocked beams of radius r is given by the surface area of the set $\mathcal{I}_{d,r}$ that remains after binary erosion of $\mathcal{I}_d$ by B_r (cf. Figure 4). Formally, $\mathcal{I}_{d,r} = \{\mathbf{x} \in \mathcal{I}_d : B_r(\mathbf{x}) \subset \mathcal{I}_d\}$, and we denote its area by $A(\mathcal{I}_{d,r})$.

The larger the area $A(\mathcal{I}_{d,r})$, the more beams are blocked; further, even infinitesimal holes in the original projection $\mathcal{I}_d$ eat away a disk of radius r under erosion, quickly diminishing the occlusion power of a surface with holes and narrow extremities. Intuitively, assuming a parallel projection, the surface area measures the how many points the object can occlude in a given direction, whereas the area $A(\mathcal{I}_{d,r})$ measures how many balls with radius r the object has potential to occlude. Note that usual surface area coincides with $A(\mathcal{I}_{d,0})$.

To measure the compound occlusion power over beams of all sizes, we define the *directional occlusion measure* $M(\mathbf{d})$ as the integral of the area of the erosions over all beam radii:

$$M(\mathbf{d}) = \int_0^\infty A(\mathcal{I}_{d,r})\mathrm{d}r. \tag{1}$$

Note that this integral never diverges, as the area $\mathcal{I}_{d,r}$ is guaranteed to decrease with increasing radius. Finally, in order to account for different directions, we define the *total occlusion measure* M for the object $\mathcal{O}$ as the integral over all directions

$$M = \int_\Omega M(\mathbf{d})\mathrm{d}\omega. \tag{2}$$

3.2. Fast Occlusion Measure Computation

The directional occlusion measure M_d is closely connected to the Euclidean distance transform [RP66]. The Euclidean distance transform $D(\mathbf{x})$ at a point $\mathbf{x} \in \mathcal{I}_d$ gives the radius r of the maximal disk $B_r(\mathbf{x})$ such that $B_r(\mathbf{x}) \subset \mathcal{I}_d$, i.e., it returns the distance to the closest point on the boundary. Now, we can write the directional occlusion measure $M(\mathbf{d})$ in terms of the Euclidean distance transform D:

$$M(\mathbf{d}) = \int_0^\infty A(\mathcal{I}_{d,r})\mathrm{d}r = \int_{\mathcal{I}_d} D(\mathbf{x})\mathrm{d}A, \tag{3}$$

where $\mathrm{d}A$ is the area measure on the projection plane. This can be seen by considering the 3D space $\mathrm{im}(\mathcal{P}_d) \times \mathbb{R}$ and rewriting Equation (1) as

$$M(\mathbf{d}) = \int_{\mathrm{im}(\mathcal{P}_d)} \int_{r=0}^\infty \mathbb{1}_{\mathcal{I}_{d,r}} \,\mathrm{d}A\,\mathrm{d}r, \tag{4}$$

where we've merely expanded the definition of the surface area of the set $\mathcal{I}_{d,r}$ by writing it as the area integral over its indicator function. For each $\mathbf{x}$ in the original projection $\mathcal{I}_d$, the indicator function remains non-zero with increasing r until the point is "eaten away" by erosion. Clearly, this happens when $r = D(\mathbf{x})$, i.e., when the erosion frontier reaches the point from the closest boundary point. The right-hand side of Equation (3) immediately follows.

Equation (3) is key to efficient evaluation of the directional occlusion measure since it allows us to compute $M(\mathbf{d})$ in a single pass instead of repeatedly evaluating $A(\mathcal{I}_{d,r})$ for different radii. In practice, we compute its value by rasterizing the projection $\mathcal{I}_d$ onto a discrete grid. We then compute the discrete Euclidean distance transform of the resulting binary image using a parallel jump flood fill algorithm [RT06], and approximate the integral using a Riemann sum. The error introduced by the discrete pixel grid is controlled by rasterization resolution. The outer integral over directions (Equation 2) that yields the total occlusion measure M is evaluated by repeating the process for N uniformly distributed directions.

4. Algorithm

In this section, we describe the operation of our occluder simplification algorithm. When bounding the pixel error caused by using a simplified occluder at various rendering distances, we need multiple simplified versions with different Euclidean error bounds. Each of these is generated separately, and in the remainder of this section we discuss the construction of a single level of detail.

Overview. We start by voxelizing the input mesh using a voxel resolution appropriate for the desired error bound (Section 4.1). The binary voxel data is then immediately converted to an intermediate *voxel mesh* that is always closed and manifold. This step is crucial, because voxelization merges the input geometry, regardless of its connectivity, into a topologically simple representation that is amenable for further processing. Specifically, the voxel mesh is a well-behaved, error-bounded approximation of the original geometry: the grid resolution determines a strict bound on the maximal (Hausdorff) distance between the input geometry and the voxel mesh, meaning that the occlusion properties of the voxel mesh are guaranteed to resemble that of the input within the discretization error. Furthermore, the discretization resolution determines a natural "feature size" such that smaller holes are automatically filled, allowing simpler occluders while remaining error-bounded. Figure 5 illustrates the effect of resolution-dependent discretization and the resulting interpretation of the input geometry.

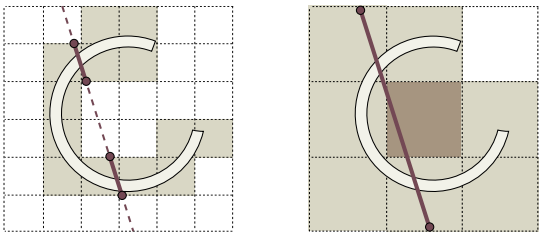


Figure 5: Scale-sensitive discretization illustrated in 2D. Left: In fine scale, corresponding to using the occluder near the camera, the input object is treated as open, allowing visibility between its interior and exterior. A candidate slice for the simplified occluder (dashed line) correctly contains only the boundary of the object. Right: On a coarser scale, applicable for using the occluder at a farther distance, flood fill marks the interior voxel as inside, permitting the production of an occluder candidate slice that spans the entire object.

After constructing the voxel mesh, we then generate a set of 2D slices which serve as an approximation of the original model (Section 4.2). The slices are then converted into a simplified polygon soup (Sections 4.3 and 4.4) and used as input to a greedy optimization algorithm to generate the final occluder (Section 4.5).

4.1. Voxelization and Remeshing

We begin by transforming the input triangle soup into a voxel representation, using a voxel resolution that is appropriate for the allowed Euclidean error in the simplification. Initially, all voxels are labeled OUTSIDE. For each input triangle, we mark all voxels touched by the triangle as INSIDE using an exact triangle vs. AABB test. The input geometry is not used after this step.

After processing the input triangles, we partition the remaining OUTSIDE voxels into 6-connected components using flood fills. Voxels in the components that do not connect to the boundary of the voxel grid are marked INSIDE, turning closed boundaries—as interpreted in the given voxel resolution—into solid objects.

The final step is to generate the voxel mesh by iterating

over all voxels, and adding two triangles for each face between an INSIDE and an OUTSIDE voxel, orienting the triangles consistently towards the OUTSIDE voxel. This results in a closed, 2-manifold triangle mesh.

4.2. Interior Sampling

To accelerate the construction of occluder polygon candidates, we employ hardware rasterization. Given a direction **d**, we sample the depth interval of the object AABB with a set of evenly distributed planes. For each depth slice, we use GPU to rasterize the voxel mesh using an orthogonal projection where the near clip plane is set at the slicing depth, and the far clip plane is behind the voxel mesh. By setting the stencil buffer to increment by one for back faces and decrement by one for front faces of the near-clipped mesh, we obtain an image of the slice in the stencil buffer (Figure 6a). The rasterization resolution is chosen based on the voxel resolution, so that each voxel in the discretized input will cover at least one pixel.

In order to generate a set of 2D slices capturing the occlusion properties of the original mesh as closely as possible, we discretize the search space by dividing the sample directions and depths into bins. We then generate a single representative 2D slice for each bin by exhaustively sampling the corresponding direction and depth ranges on the GPU and pick the slice with maximal surface area.

4.3. Edge Loop Extraction and Simplification

From the slice bitmap image (Figure 6a), we construct edge loops out of edges placed at the boundaries between covered and uncovered pixels. Finding the edge loops can be done simply by stepping along the edges and closing the loops when encountering the starting point, until all edges have been processed (Figure 6b). The resulting loops are non-intersecting but may meet at pixel corners, which has to be taken into account in the subsequent processing steps.

In order to convert the set of edge loops into simple polygons, we connect each loop that represents a hole in the slice to the enclosing boundary loop (Figure 6c), thereby converting it into a part of the boundary. Note that even after all hole loops have been eliminated, a single slice may still produce multiple polygons as a result if the cross-section of the input contains disconnected parts.

As the edge loops were constructed from a raster image, they typically contain many redundant edges. As a final step, we perform error-bounded edge loop simplification using the Ramer-Douglas-Peucker algorithm [Ram72, DP11] for each edge loop (Figure 6d). This algorithm takes the maximum allowed deviation of the simplified edge loop as an input, and we set this to match the voxel resolution.

4.4. Polygon Simplification Chain Generation

The number of triangles T_i in each simple polygon P_i produced by the previous step can easily exceed the triangle budget, ruling out the simplest optimization algorithm where we would incrementally add new candidate polygons to the occluder as long as we have room in the triangle budget. To solve this issue, we build a progressive simplification chain for each candidate polygon and optimize over the simplified versions instead of the entire polygons. This approach allows for extreme simplification and produces balanced occluder models even with very low triangle count.

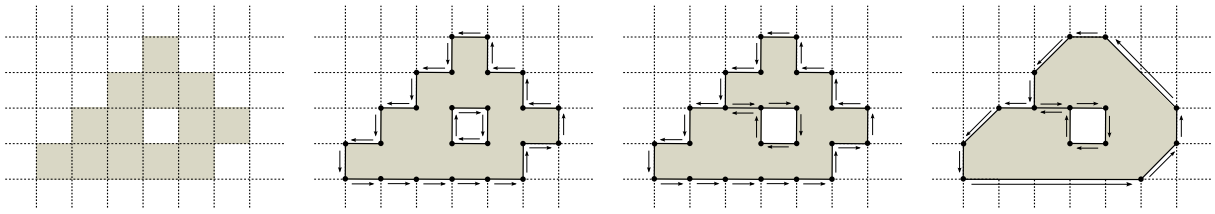
For each candidate polygon P_i with T_i triangles, we build a progressive simplification chain $(P_i^{T_i}, P_i^{T_i-1}, ..., P_i^1)$, where the superscripts denotes the number of triangles, using a greedy vertex collapse algorithm. We start with the complete polygon $P_i^{T_i}$ and iteratively collapse vertices until we have a polygon with only a single triangle P_i^1. During each iteration, we remove the vertex v_k such that the area of the triangle v_{k-1}, v_k, v_{k+i} is minimal and the resulting polygon is fully contained in the interior of the original candidate polygon. Finally, each resulting polygon P_i^j, where $j = 1, ..., T_i$ is triangulated.

4.5. Occluder Optimization

The remaining task in generating the simplified occluder is choosing how many triangles of each candidate polygon to include in the result. Formally, given n candidate polygons P_i, $i = 1, ...n$ and a budget of B triangles, our goal is to find a cut across the simplification chains $C = (c_1, c_2, ..., c_n)$, where $0 \leq c_i \leq T_i$, $i = 1, ..., n$ such that the set of polygons $P = \{P_1^{c_1}, P_2^{c_1}, ..., P_n^{c_n}\}$ maximizes occlusion subject to $\sum_{i=r}^{n} c_i \leq B$.

An exhaustive search would be prohibitively expensive, so we use a greedy approach instead. We start with an empty cut $C_0 = (0, 0, ..., 0)$ and continue by adding one polygon $P_i^{T_i}$ at a time until the cut is α-*complete*, i.e., adding new polygons would increase the occlusion measure by less than α percent. During each step K we evaluate all possible cuts C_{k+1} obtained from C_k by adding a new polygon $P_i^{T_i}$ and choose the one which maximizes the total occlusion measure for the polygon soup corresponding to the cut C_{k+1}.

The resulting occluder is complete but it might still exceed the triangle budget. As a final step, we simplify the cut until the budget constraint is satisfied. We start with the cut obtained by the greedy algorithm and proceed by removing one triangle at a time until the budget is met. During each step k, we evaluate all possible cuts C_{k+1} obtained by traversing a single step in the simplification chain of every polygon, i.e., we consider cuts of the form $C_k - (1, 0, ..., 0), C_k - (0, 1, 0, ..., 0), ..., C_k - (0, ..., 0, 1)$ and choose the one which maximizes the surface area of C_{k+1}.



(a) Rasterized depth slice (b) Extracted edge loops (c) Simplified topology (d) Final slice polygon

Figure 6: Occluder candidate generation. (a) Result of interior sampling is a bitmap of the rasterized voxel mesh at one depth slice. (b) Edge loops are extracted by connecting boundary edges of the rasterized pixels. (c) Topology is simplified by connecting the edge loops of holes to the enclosing boundary loops. (d) The remaining edge loops are then simplified using Ramer-Douglas-Peucker algorithm [Ram72, DP11] with user-specified error tolerance.

5. Hierarchical Occluders

For the occluder generation algorithm to be useful in complex settings, we generalize it from individual objects to hierarchies. Fortunately, this turns out to be straightforward. For a given scene, we build a bounding volume hierarchy (BVH) and apply our algorithm independently for each node. In order to smoothen out the transitions between different levels of the hierarchy, we apply a single *dilation step*, i.e., marking every OUTSIDE voxel as INSIDE if it has any INSIDE neighbors during the voxelization. This has the effect of gluing neighbors together. Note that the error in this step remains bounded.

We use the hierarchy at runtime to select a suitable set of occluder nodes based on the camera position and viewing parameters. These hierarchical occluders are subsequently used to perform occlusion culling from the camera (Figure 10), using axis-aligned bounding box of each object as a testing primitive. The testing primitives are then expanded based on the screen space error bounds and taking perspective projection into account in order to guarantee conservative culling results when possible, i.e., in cases where the discretization step has not changed the apparent topology.

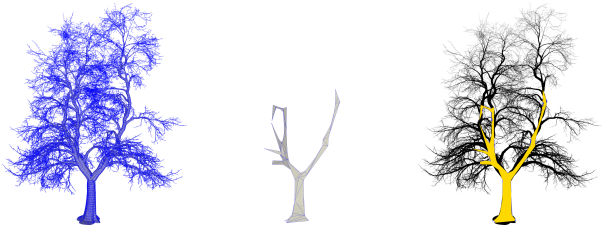
6. Results

Figure 7 shows the results of our algorithm from multiple different angles in five different test scenes. Each occluder was computed with a budget of 64 triangles. The images are color-coded in order to demonstrate the difference between the original mesh and the generated occluder.

We show an example of a particularly difficult input scene in Figure 8.

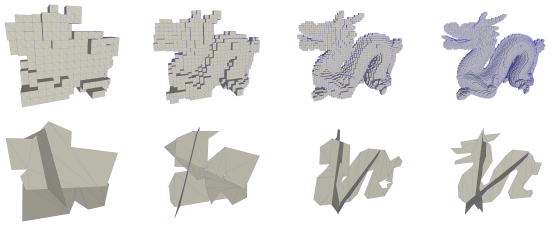
To demonstrate the effect of the voxel grid resolution on the resulting occluder, Figure 9 shows a sequence of occluders built from the DRAGON scene.

Figure 10 and the accompanying video demonstrate the hierarchical variant of the algorithm. The upper left corner in the video displays the percentage of occluder triangles used in comparison to the amount of actual display geometry.



Input Occluder with 64 triangles Comparison

Figure 8: An example of particularly difficult input, e.g., a complex tree without leaves. The occluder was computed using a voxel resolution of 256^3.



$16x16x16$ 22 triangles | $32x32x32$ 41 triangles | $64x64x64$ 64 triangles | $128x128x128$ 97 triangles

Figure 9: The effect of scale-sensitive discretization on the resulting occluder models in the DRAGON-scene.

6.1. Occlusion Accuracy

We numerically evaluate the quality of our occluders by sampling their occlusion performance over a set of directions using random occludees. For each direction, we rasterize both the occluder and 32 000 axis aligned quads chosen at random over the screen bounding box of the object. We classify the pixels of each quad as follows: 1) When all pixels in a quad are covered by both the simplified occluder and the original mesh, the pixels are counted as true negatives (N_t). 2) If the quad is completely covered by the original

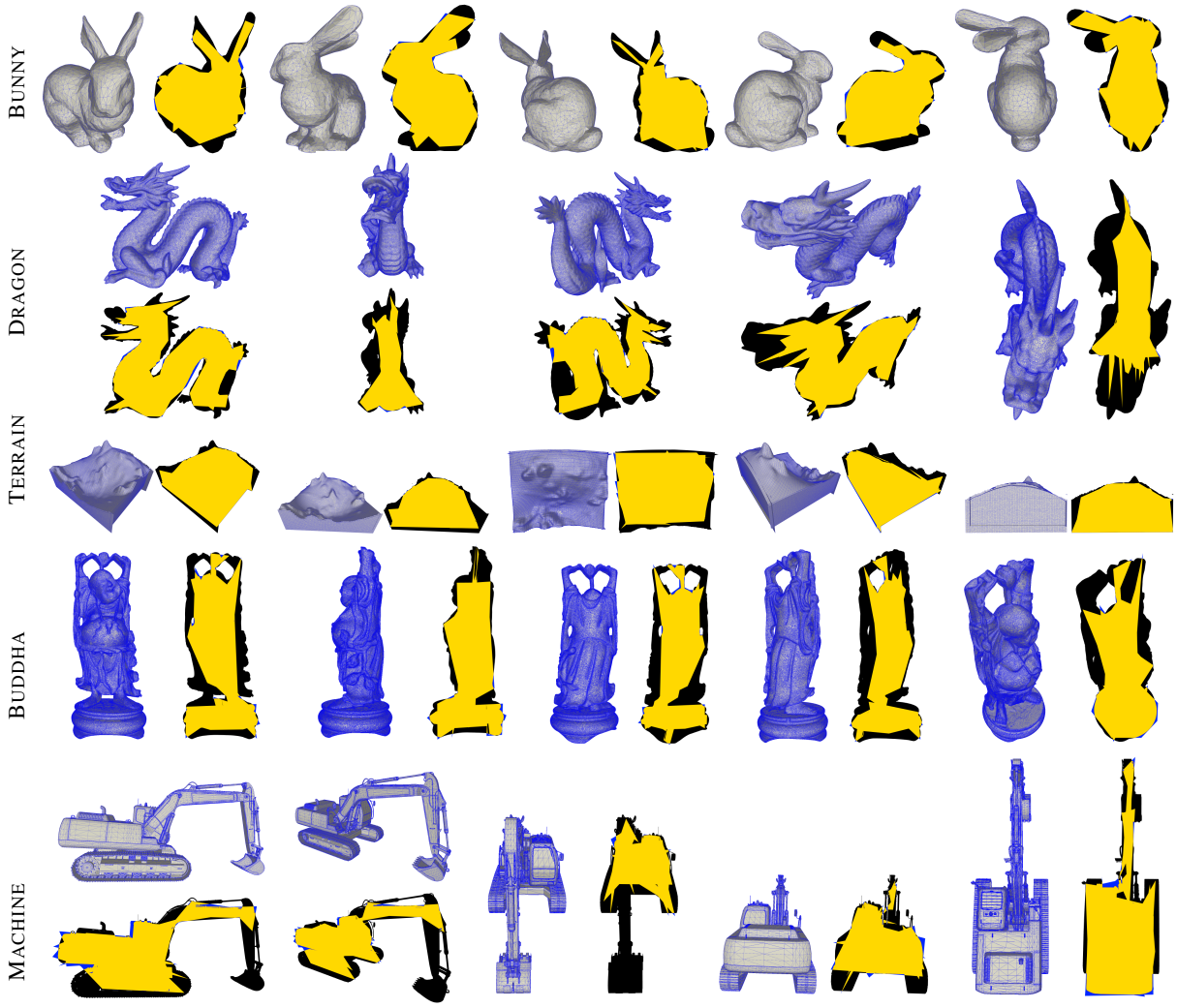


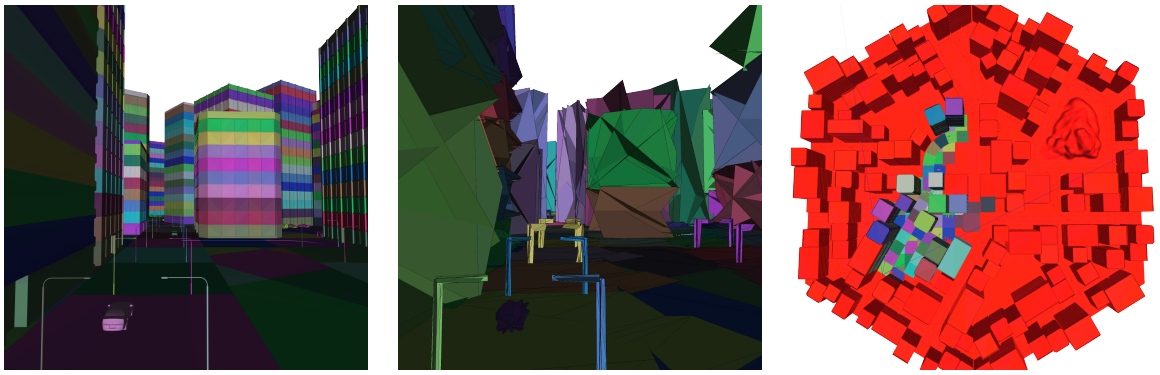
Figure 7: Results from our test scenes. The silhouette images are colour-coded for visual comparison. The original geometry is rendered in black, the intersection of the original geometry and the generated occluder is yellow. Over-occluded parts are coded in blue.

model but some pixels are not covered by the occluder, all pixels are classified as false positives (P_f). 3) Finally, if the quad is entirely covered by the occluder, but some pixels are not covered by the original mesh, we classify the erroneous pixels as false negatives (N_f). Note that false positives cause additional work but no image artifacts, and false negatives cause erroneous renderings.

We can now compute the precision $P = N_t/(N_t + N_f)$ and recall $R = N_t/(N_t + P_f)$. High precision means that the amount of visual artifacts due to false occlusion is low, and high recall means that the simplified occluder occludes almost all the same quads as the original mesh, i.e., the amount of unnecessary rendering work due to lost occlusion is small.

To verify that our total occlusion measure leads to improved results over using the surface area, Figure 11 compares recall distributions over five scenes and over 256 randomly chosen view directions. In both cases, occluders are built using our algorithm; the only difference is the measure used. Occluders built using surface area have distributions that are clearly heavier at low recall values in all scenes, meaning they generate larger number of false positives. This conforms to intuition: surface area does not penalize holes, which leads to poor occluders (Figure 12). The graphs demonstrate that the proposed total occlusion measure behaves consistently better.

The precision distributions (not shown) are almost iden-



(a) Original scene (b) Hierarchical occluders (d) A bird's eye view of the scene

Figure 10: Hierarchical occluders. (a) Camera view. The objects in the scene are color coded to show object granularity. (b) At runtime we choose a tree cut of occluder hierarchy based on viewing parameters. (c) The occluders are used to perform occlusion culling. Red objects are hidden from the camera's point of view.

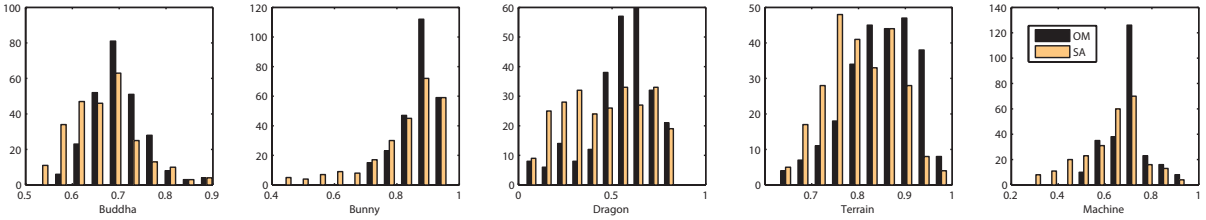


Figure 11: Distribution of recall over 256 randomly oriented views, comparing occluders built using the total occlusion measure (OM) and surface area (SA). The recall value for each view is computed by computing occlusion over 32 000 random screen-aligned quads. Cf. Section 6.1.

tical between surface area and the proposed occlusion measure. The values range between $97-99\%$, meaning the occluders do not significantly over-occlude in comparison to the original model. This is to be expected, since the discretization error is bounded.

6.2. Overdraw and Rasterization Performance

Our algorithm optimizes for low triangle count, but by its nature, generates occluders with potentially significant overdraw. We analyze the significance of this design choice by computing the rasterization cost of both the original mesh and the simplified occluders using a state-of-the-art hierarchical software rasterizer [Umb13]. We find an almost perfect linear correlation between rasterization time and triangle count, i.e., the speedup from using our occluders is directly predicted by the ratio of the triangle count to the original model. This demonstrates that overdraw does not cause bottlenecks.

Depending on the scene, the rasterization of the original, detailed meshes takes 5-150ms. This is unacceptable for a real-time occlusion culling system. Our simplified occluders are faster to render in direct proportion to triangle count, one to two orders of magnitude in our test cases. This brings rasterization cost down to the submillisecond range, which is necessary for real-time applications.

6.3. Comparison to Other Methods

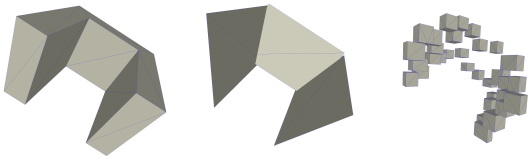
We compare our method against Oxel [Dar11] and the Autodesk® 3ds Max ProOptimizer ® (Max) in five scenes (Figure 14). Each method is given the same triangle budget and we compute the precision and recall of the resulting occluder models (Table 1). We use a voxel resolution of $64 \times 64 \times 64$ for both Oxel and our method.

Our method compares favourably against Oxel and Max. Oxel demonstrates good precision but suffers from overall poor recall. Max performs inconsistently, having a slightly better recall compared to our method in BUNNY and DRAGON scenes, albeit at the cost of lower precision, but fails to produce usable results in MACHINE and BUDDHA scenes.

Oxel can produce good results in the special case of axis-aligned input but has difficulties in more general cases (Figure 13). Furthermore, no guarantees are given that the re-

Input SA OM

Figure 12: The difference between surface area (SA) and total occlusion measure (OM). Surface area can lead to sub-optimal occluders with visible holes (Middle) compared to total occlusion measure (Right).



Input model 28 triangles | Our method 6 triangles | Oxel 509 triangles

Figure 13: The Oxel method [Dar11] has difficulties with non-axis aligned input geometry. In comparison, our method does not make any assumptions on the orientation of the input geometry.

sulting occluder has fewer triangles than the input triangle soup.

7. Conclusions

We have described a technique for error-bounded occluder simplification with few requirements on input geometry. Our algorithm is capable of extreme simplification due to its volumetric nature, and is driven by expected screen-space error. In addition, we extended the occluder generation algorithm for large scenes using a hierarchy. In particular, the camera is free to move *inside* the simplified hierarchical model. This would be impossible using a single-level algorithm.

We believe that our resilience to bad topology and support for large scenes (as opposed to single objects) are the keys for real-world applicability.

Scene	Triangles	Budget	Time	Precision			Recall		
				Our	Oxel	Max	Our	Oxel	Max
BUNNY	4968	53	18s	99.9%	100%	99.7%	83.5%	50.5%	86.6%
DRAGON	871306	36	11s	99.9%	99.9%	97.2%	47.8%	16.7%	48.5%
TERRAIN	60160	23	11s	99.9%	100%	99.8%	82.3%	29.3%	73.6%
MACHINE	394452	36	8s	99.8%	100%	100%	56.4%	12.1%	0.5%
BUDDHA	1087474	30	10s	99.9%	100%	100%	60.2%	17.4%	3.8%

Table 1: Details and color-coded precision/recall measurements from our test scenes. Time corresponds to the total computation time for our method. Highest precision/recall values are indicated in green and lowest are indicated in red. Cf. Section 6.1.

In the future, we are interested in investigating weighted occlusion measures that take the object distribution in the scene as a prior. In addition, we would like to apply the occlusion measure to other problem domains, such as hierarchy construction in ray tracing.

References

[AM04] AILA T., MIETTINEN V.: dPVS: An occlusion culling system for massive dynamic environments. *IEEE Comput. Graph. Appl. 24*, 2 (2004), 86–97. 1

[ARB90] AIREY J. M., ROHLF J. H., BROOKS JR. F. P.: Towards image realism with interactive update rates in complex virtual building environments. In *Proceedings of the 1990 symposium on Interactive 3D graphics* (New York, NY, USA, 1990), I3D '90, ACM, pp. 41–50. 1

[BWPP04] BITTNER J., WIMMER M., PIRINGER H., PURGATHOFER W.: Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum 23*, 3 (Sept. 2004), 615–624. Proceedings EUROGRAPHICS 2004. 1

[CT97] COORG S., TELLER S.: Real-time occlusion culling for models with large occluders. In *Proceedings of the 1997 symposium on Interactive 3D graphics* (New York, NY, USA, 1997), I3D '97, ACM, p. 83. 2

[CVM*96] COHEN J., VARSHNEY A., MANOCHA D., TURK G., WEBER H., AGARWAL P., BROOKS F., WRIGHT W.: Simplification envelopes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 119–128. 2

[Dac11] DACHSBACHER C.: Analyzing visibility configurations. *IEEE Transactions on Visualization and Computer Graphics 17*, 4 (Apr. 2011), 475âĂŞ486. 2

[Dar11] DARNELL N.: Automated occluders for GPU culling. *Game Developer Magazine* (September 2011). 2, 8, 9

[DDSD03] DÉCORET X., DURAND F., SILLION F. X., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Trans. Graph. 22*, 3 (July 2003), 689–696. 2

[DP11] DOUGLAS D. H., PEUCKER T. K.: *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature*. John Wiley & Sons, Ltd, 2011, pp. 15–28. 5, 6

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 209–216. 2

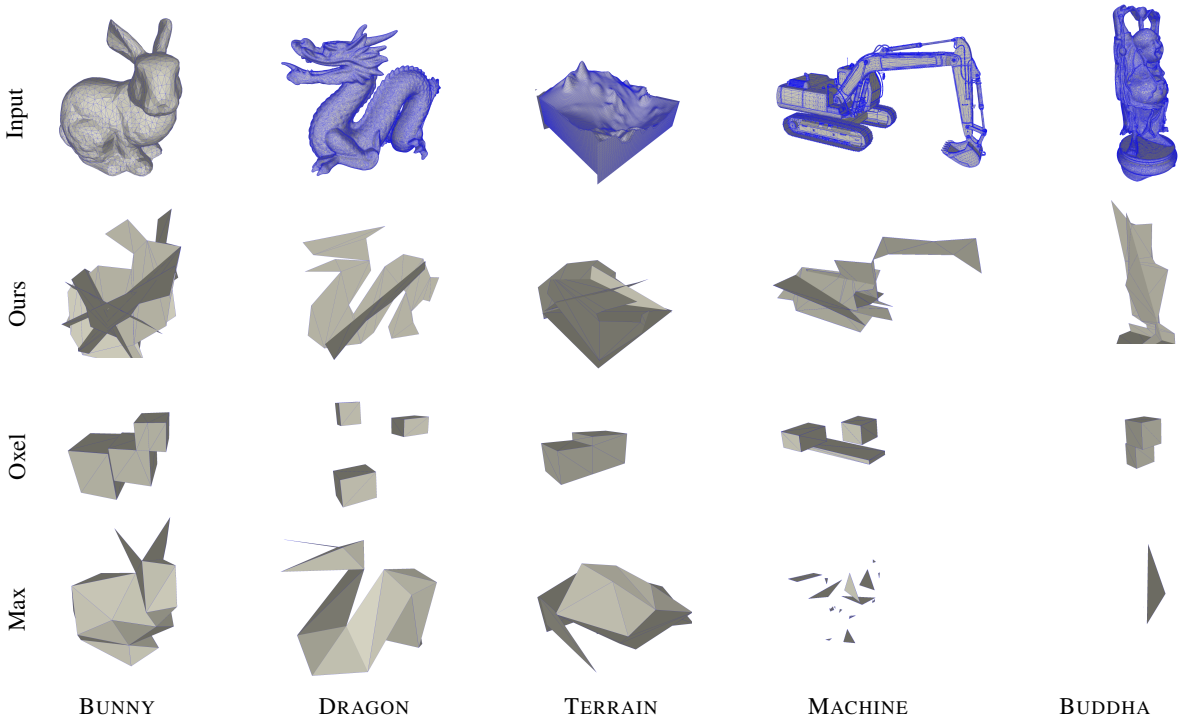


Figure 14: Comparison to alternative methods. Cf. Section 6.3.

[GJ01] GERMS R., JANSEN F. W.: Geometric simplification for efficient occlusion culling in urban scenes. In *Proc. of WSCG 2001* (2001), pp. 291–298. 2

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 99–108. 2

[KCCo00] KOLTUN V., CHRYSANTHOU Y., COHEN-OR D.: Virtual occluders: An efficient intermediate PVS representation. In *11th Eurographics Workshop on Rendering* (2000), pp. 59–70. 2

[LG95] LUEBKE D., GEORGES C.: Portals and mirrors: simple, fast evaluation of potentially visible sets. In *Proceedings of the 1995 symposium on Interactive 3D graphics* (New York, NY, USA, 1995), I3D '95, ACM, pp. 105–ff. 1

[LT99] LAW F.-A., TAN T.-S.: Preprocessing occlusion for real-time selective refinement. In *Proceedings of the 1999 symposium on Interactive 3D graphics* (New York, NY, USA, 1999), I3D '99, ACM, pp. 47–53. 2

[LT00] LINDSTROM P., TURK G.: Image-driven simplification. *ACM Trans. Graph. 19*, 3 (July 2000), 204–241. 2

[NT03] NOORUDDIN F. S., TURK G.: Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (Apr. 2003), 191–205. 2

[Per12] PERSSON E.: Creating vast game worlds: experiences from Avalanche Studios. In *ACM SIGGRAPH 2012 Talks* (Los Angeles, CA, USA, 2012), SIGGRAPH '12, ACM. 2

[Ram72] RAMER U.: An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing 1*, 3 (1972), 244 – 256. 5, 6

[RP66] ROSENFELD A., PFALTZ J. L.: Sequential operations in digital picture processing. *J. ACM 13*, 4 (Oct. 1966), 471–494. 4

[RT06] RONG G., TAN T.-S.: Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006), I3D '06, ACM, pp. 109–116. 4

[Umb13] UMBRA SOFTWARE: *Umbra3*. 2013. 8

[Val11] VALIENT M.: Practical occlusion culling in KILLZONE 3. In *ACM SIGGRAPH 2011 Talks* (Vancouver, BC, Canada, 2011), SIGGRAPH '11, ACM. 2

[WS99] WONKA P., SCHMALSTIEG D.: Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum 18*, 3 (1999), 51–60. 2

[ZT02] ZHANG E., TURK G.: Visibility-guided simplification. In *Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 267–274. 2