Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs

Peter Wonka, Michael Wimmer, Dieter Schmalstieg
Vienna University of Technology

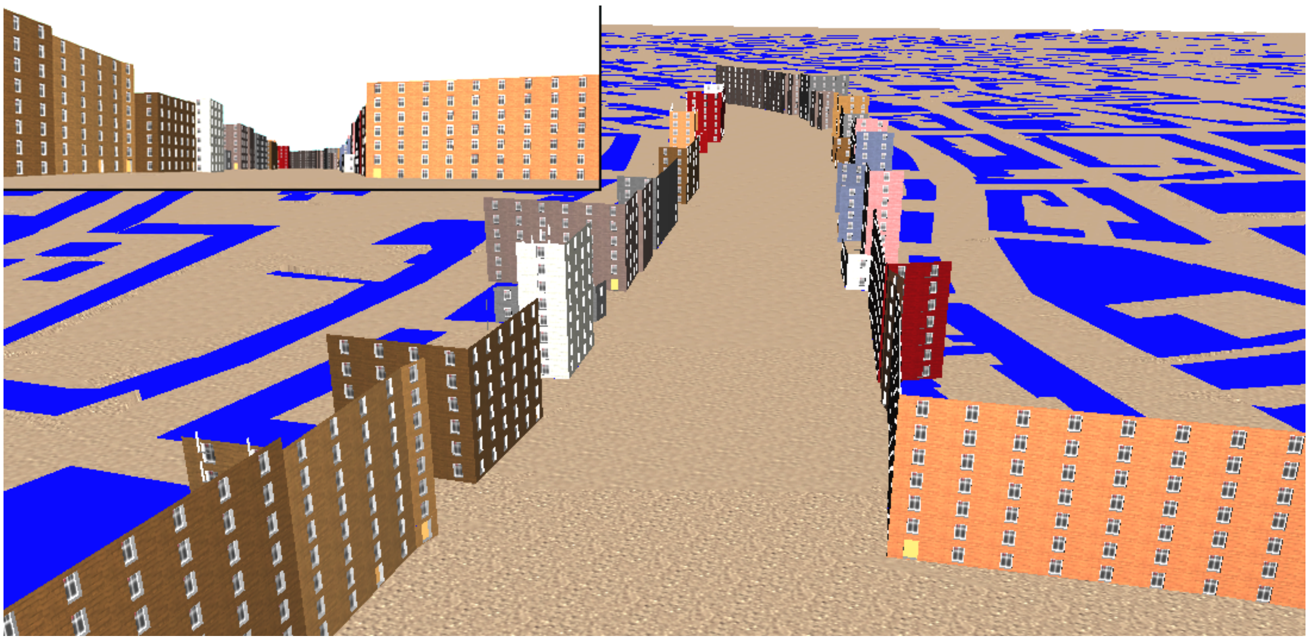

Figure 1: Views from the 8 million polygon model of the city of Vienna used in our walkthrough application. The inset in the upper left corner shows a typical wide open view, while the large image shows the portion of the scene rendered after occlusion culling. Note how occlusion fusion from about 200 visible occluders allows to prune over 99% of the scene.

ABSTRACT

This paper presents an efficient algorithm for occlusion culling of urban environments. It is conservative and accurate in finding all significant occlusion. It discretizes the scene into view cells, for which cell-to-object visibility is precomputed, making on-line overhead negligible. Unlike other precomputation methods for view cells, it is able to conservatively compute all forms of occluder interaction for an arbitrary number of occluders. To speed up preprocessing, standard graphics hardware is exploited and occluder occlusion is considered. A walkthrough application running an 8 million polygon model of the city of Vienna on consumer-level hardware illustrates our results.

Keywords

Visibility determination, occlusion culling, occluder fusion, occluder occlusion.

1. INTRODUCTION

Applications of visual simulation, such as architectural walkthroughs, driving simulators, computer games, or virtual reality require sufficiently high update rates for interactive feedback. While update rates starting from 10 frames per second (fps) are often called interactive, they are rarely considered satisfactory. To avoid temporal aliasing, true real-time behavior requires update rates equivalent to the monitor refresh rate, i. e. 60 Hz or more.

For very large databases, this kind of performance cannot be achieved with hardware acceleration alone, but must be assisted by algorithms that reduce the geometry to be rendered to a tolerable amount. A popular means is occlusion culling, which quickly prunes large portions of the scene. Visibility calculations for a *single viewpoint* have to be carried out online and infer significant runtime overhead. Moreover, in a typical single-CPU system, online processing time must be shared with other tasks, such as rendering and collision detection. Therefore it is useful to calculate visibility for a region of space (*view cell*) in a preprocessing step.

1.1 Motivation

Occlusion culling shares many aspects with shadow rendering. Occlusion culling from a view cell is equivalent to finding those objects which are completely contained in the umbra (shadow volume) with respect to a given area light source. In contrast to occlusion from a point, exact occlusion culling for regions in the general case (or its equivalent, shadow computation for area light sources) is not a fully solved problem. Two main problems impede a practical closed-form solution:

- The umbra with respect to a polygonal area light source is not only bounded by planes, but also by reguli, i. e. ruled quadratic surfaces of negative Gaussian curvature [11][30]. Such reguli are difficult to store, intersect etc.
- The maximum number of topologically distinct viewing regions constitutes the so called *aspect graph* [14][22], which is costly to compute ($O(n^9)$ time [14]). For

applications such as radiosity, a more practical approach is the *visibility skeleton* [12]. However, the algorithmic complexity and robustness problems of analytic visibility methods impede their practical use for large scenes.

For visibility from a point, the joint umbra of many occluders is the union of the umbrae of the individual occluders, which is simple to compute. In contrast, for view cells, the union of umbrae is only contained in the exact umbra, which also depends to a great extent on contributions of merged penumbrae (see Figure 2). While umbra information for each occluder can be described by a simple volume in space ('in/out'-classification for each point), penumbra information also has to encode the visible part of the light source, making it hard to find a practical spatial representation. Therefore a general union operation for penumbrae is not easily defined.

Although an approximation of the umbra from multiple occluders by a union of individual umbrae is conservative, it is not sufficiently accurate. There are frequent cases where a significant portion of occlusion coming from occluder fusion is missed, making the solution useless for practical purposes. In particular, we distinguish between the cases of connected occluders, overlapping umbrae, and overlapping penumbrae (see Figure 2, letter a, b and c respectively).

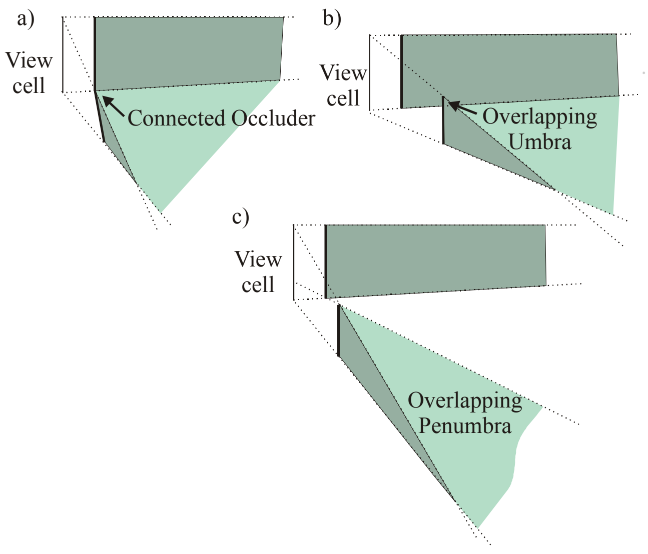

Figure 2: Different types of occluder interactions. Individual umbrae (shaded dark) cover only a small area, while the occluders jointly cover a large unbounded area (shaded light).

This paper describes a fast, conservative occlusion culling algorithm for urban environments (i.e., with 2.5D occluders – objects represented by functions z = f(x,y)) that solves the aforementioned problems. It is based on the idea that conservative visibility for a region of space can be calculated by *shrinking* occluding objects and *sampling* visibility from an interior point of the region. Given a partition of space into view cells, we exploit graphics hardware to calculate and merge such conservative visibility samples on the boundary of each cell. Thus, conservative visibility for each cell can be computed as a preprocess, and visibility determination consumes no time during actual walkthroughs. This approach has several essential advantages:

1. Culling Efficiency. Through the use of graphics hardware, our technique is able to consider a very large number of occluders. This is especially important in situations where a large portion of the scene is actually visible. Techniques that consider only a small number of occluders (e. g., 50-100) and rely on heuristics to select them may fail to capture significant occlusion, especially under worst-case conditions.

2. Occluder Fusion: Our technique places no inherent restriction on the type of occluder interaction. Fusion of occluder umbrae[1] and even penumbrae is implicitly performed. This includes the hard case where individual umbrae of two occluders are disjoint (see Figure 2, letter c). We also discretize umbra boundaries, which are typically made up of reguli (curved surfaces). Other published algorithms do not currently handle such cases of occluder interactions.

3. Conservativity. Our method never reports visible objects as invisible. While some authors argue that non-conservative (approximate) visibility *can* be beneficial, it is not tolerable for all applications.

Note that while occlusion culling can typically speed up rendering by orders of magnitude, if the complexity of visible objects is too high, no visibility algorithm alone can guarantee sufficiently high frame rates. In such cases, occlusion culling has to lay a good foundation for further simplification techniques like level of detail and image-based rendering.

1.2 Related Work

Several methods were proposed to speed up the rendering of interactive walkthrough applications. General optimizations are implemented by rendering toolkits like Performer [23] that aim for an optimal usage of hardware resources. Level of detail (LOD) algorithms [16] are very popular in urban simulation [18], because they don't need a lot of calculation during runtime. With image-based simplification, coherent parts of the scene are replaced by impostors (textured quadriliterals [25][27], textured depth meshes [2][10][28] or layered depth images [26]).

Occlusion culling algorithms calculate a conservative estimation of those parts of the scene that are definitely invisible. Final hidden surface removal is usually done with a z buffer. A simple and general culling method is view frustum culling [4], which is applicable for almost any model. One class of occlusion algorithms calculate occlusion in image space. The hierarchical z buffer from Greene et al. [15] combines object-space and image space hierarchies, but requires unconventional hardware. Hierarchical occlusion maps proposed by Zhang et al. [33] adapt this idea to be used on standard graphics hardware.

A general method to accelerate visibility is to break down the view space into cells and precompute for each cell a set of objects that are potentially visible (potentially visible sets, PVS). For general scenes, visibility precomputation can become quite costly with respect to time and memory, but for certain scenes, like terrains, it is possible to use the a priori knowledge about the scene structure for visibility calculation [7][29]. Cohen-Or et al. [6] show an algorithm for densely occluded scenes where most objects are occluded by a single convex occluder near the viewpoint. The resulting PVS computation is demonstrated with an urban environment.

For building interiors, most visibility algorithms partition the model into cells connected by portals [1][31] and compute inter-cell visibility. The algorithm by Luebke and Georges [21]

[1] The umbra with respect to a view cell is the region of space from where no point of the view cell can be seen, whereas the penumbra is the region of space from where some, but not all points of the view cell can be seen.

eliminates most precomputations and calculates the PVS during runtime. Urban environments were stated as a possible application, but no examples are shown.

Occlusion culling with a few large occluders is a popular approach for urban environments. Similar algorithms have been proposed by Coorg and Teller [8][9], Bittner et al. [3] and Hudson et al. [17]. They all select for each frame a small set of occluders (about 5-30) which are likely to occlude a big part of the model. Hudson et al. [17] organize the scene in a hierarchical data structure like a k-d tree or bounding-box tree and clip it against the occluded region. Bittner et al. use a variation of the shadow volume BSP tree to merge occluder shadows. Coorg and Teller calculate visibility events to make additional use of temporal coherence.

Several view cell visibility methods have only recently been proposed. Durand et al. [13] position six planes around a view cell and projects occluders on those planes. To calculate conservative occlusion he defines an extended projection for occluders and occludees. Schaufler et al. [24] perform occluder fusion by extending occluders in an octree into octree nodes already found occluded. Koltun et al. [19] calculate large cross sections through the shadow volume from a view cell and use them as virtual occluders during online occlusion culling.

1.3 Cull Maps

The algorithm presented in this paper makes use of graphics hardware to calculate occlusion. We use the concept of *cull maps*, which have recently been proposed by Wonka and Schmalstieg [32] for online occlusion culling of city-like scenes. Occluders in urban environments can be seen as height fields, i.e. they can be modeled by a function $z = f(x,y)$. Occlusion caused by an object occurs inside a shadow volume cast from the viewpoint. We exploit the fact that this shadow volume is also a height field, and can be described by its upper bounding polygon (shadow polygon): an object is hidden if it is below the shadow polygons of the buildings (see Figure 3). Graphics hardware can be used to efficiently compute this test.

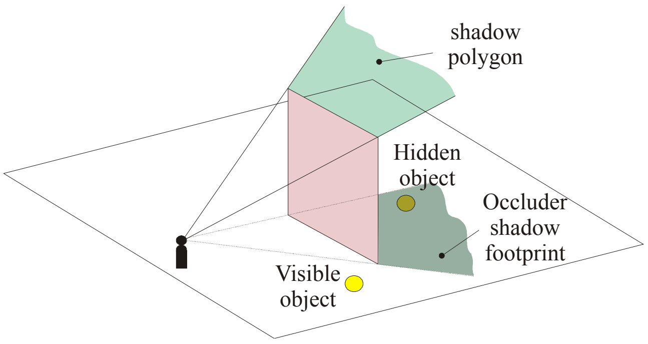

Figure 3: Occluder shadows are an efficient tool for occlusion culling in urban environments.

The scene is structured in a regular 2D grid coincident with the (x,y)-plane and all objects are entered into all grid cells with which they intersect. Occluder shadows are rendered into an auxiliary buffer - the *cull map* - using polygonal rendering hardware. Each pixel in the cull map (image space) corresponds to a cell of the scene-grid (object space). Therefore, the cull map is an image plane parallel to the (x,y)-plane of the scene.

The z-values of the covered pixels in the cull map describe the minimal visible height of the corresponding cell in the scene-grid. The graphics hardware automatically fuses multiple occluder polygons rendered in sequence (see Figure 4): when occluder shadows intersect, the z-buffer automatically stores the z-value which provides the best occlusion.

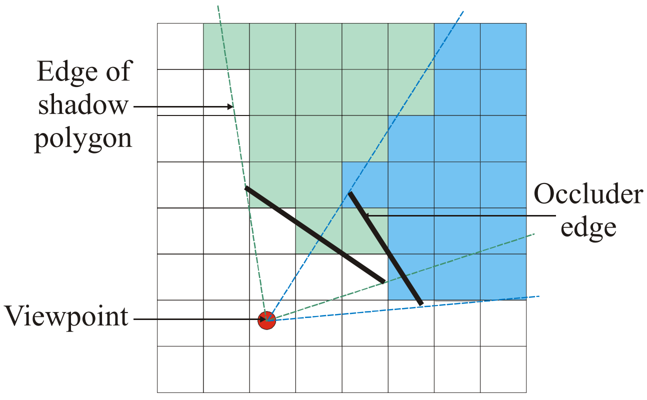

Figure 4: A cull map from a single viewpoint is created by drawing occluder shadows as polygons using graphics hardware. Overlapping occluders are correctly fused by standard rasterization and z buffer comparison.

Visibility can be determined for each cell (or object) of the grid according to the z-value in the cull map. To calculate the occluded parts of the scene, we can compare for each cell the z-value in the cull map to the z-value of the bounding boxes of the objects entered into the scene-grid at this cell.

When we render a shadow with the graphics hardware, we obtain z-values that more or less correspond to a sample in the center of the pixel. For a conservative solution, we have to guarantee that only fully occluded cells are marked as invisible and that the z-values in the z-buffer correspond to the lowest z-value of the occluder shadow in the grid cell. Therefore, shadow polygons need to be *shrunk* by a term that depends on the maximum length of a cull map cell and the slope of the boundary lines, and *moved* along its gradient by a similar amount to correct z-values (see [32] for details).

Cull maps are a simple and fast way to calculate visibility from a single point. The occlusion algorithm presented in this paper exploits their efficiency to solve the significantly more complex problem of calculating visibility for all points within a region in space.

1.4 Organization of the paper

The remainder of the paper is organized as follows. Section 2 introduces the main idea of how to calculate occluder fusion for the area visibility problem. Section 3 describes how the scene is structured into view cells. An algorithm for shrinking occluders is described in section 4. Section 5 shows how to use graphics hardware to accelerate our algorithm, while section 6 presents a hierarchical extension to the main algorithm. Results are shown in section 7 and section 8 contains a discussion of our algorithm in comparison with other methods. Section 9 concludes the paper and shows avenues of future research.

2. OCCLUDER FUSION

This section explains the main idea of our algorithm: visibility can be calculated fast and efficiently for point samples. We use such point samples to calculate visibility for a region of space. To obtain a conservative solution, occluders have to be shrunk by an amount determined by the density of point samples.

2.1 Occluder fusion by occluder shrinking and point sampling

As can be concluded from the examples in Figure 2, occluder fusion is essential for good occlusion culling, but difficult to compute for view cells directly. To incorporate the effects of occluder fusion, we present a much simpler operation, which can be constructed from point sampling (for clarity, our figures will show a simple 2D-case only).

Our method is based on the observation that it is possible to compute a conservative approximation of the umbra for a view cell from a set of discrete point samples placed on the view cell's boundary. An approximation of actual visibility can be obtained by computing the intersection of all sample points' umbrae. This approach might falsely classify objects as occluded because there may be viewing positions between the sample points from where the considered object is visible.

However, *shrinking* an occluder by ε provides a smaller umbra with a unique property: An object classified as occluded by the shrunk occluder will remain occluded with respect to the original larger occluder when moving the viewpoint no more than ε from its original position (see Figure 5).

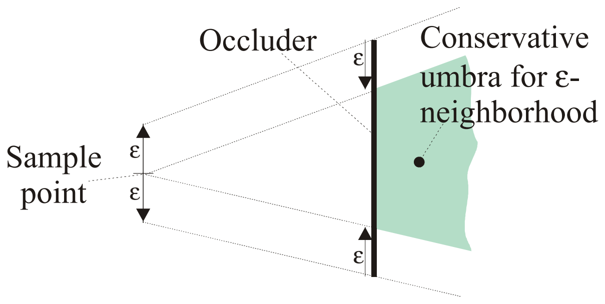

Figure 5: Occluder shrinking: By considering an occluder after shrinking it by ε, the umbra from a point sample provides a good conservative approximation of the umbra of the original occluder in a neighborhood of radius ε of the sample point.

Consequently, a point sample used together with a shrunk occluder is a conservative approximation for a small area with radius ε centered at the sample point. If the original view cell is covered with sample points so that every point on the boundary is contained in an ε-neighborhood of at least one sample point, an object lying in the intersection of the umbrae from all sample points is therefore occluded for the original view cell (including its interior).

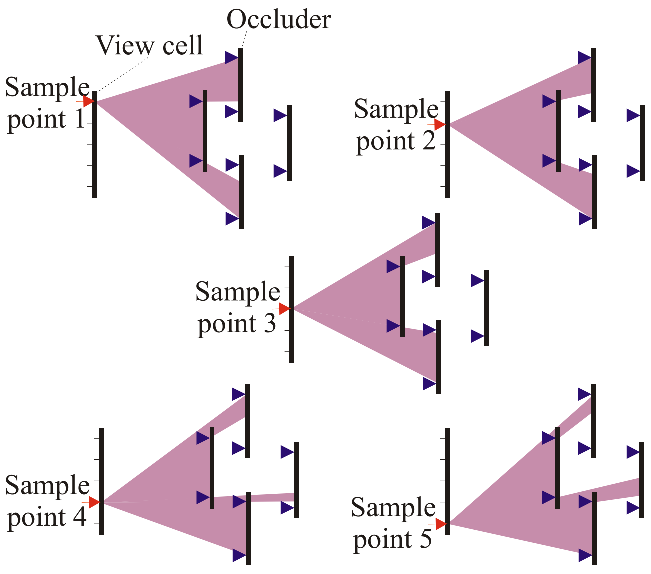

Figure 6: When performing point sampling for occlusion after occluders have been shrunk (as indicated by small triangles), all four occluders can be considered simultaneously, cooperatively blocking the view (indicated by the shaded area) from all sample points.

Using this idea, multiple occluders can be considered simultaneously. If the object is occluded by the joint umbra of the shrunk occluders for every sample point of the view cell, it is occluded for the whole view cell. In that way, occluder fusion for an arbitrary number of occluders is implicitly performed (see Figure 6 and Figure 7). While the method is conservative and not exact in that it underestimates occlusion, the fact that an arbitrary number of occluders can cooperate to provide occlusion helps to find relevant occlusion as long as ε is small in relation to a typical occluder.

In general, the exact amount by which a planar occluder has to be shrunk in each direction is dependent on the relative positions of sample point and occluder. If we consider a *volumetric* occluder, however, it can be shown that shrinking this volumetric occluder by ε provides a correct solution for an arbitrary volumetric view cell (see the appendix for a formal proof of this fact). Therefore, the occlusion culling problem can be solved using occluder shrinking and point sampling.

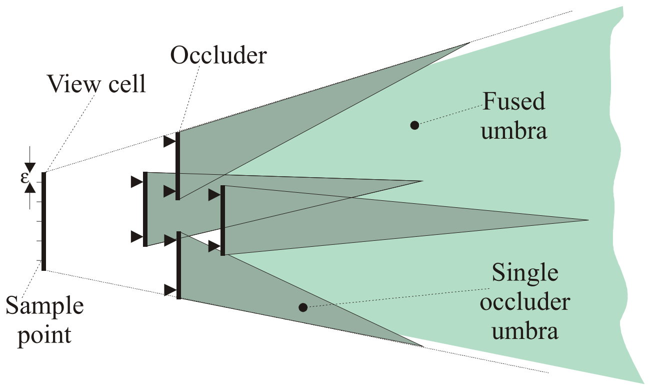

Figure 7: The fused umbra from the 5 point samples shown in Figure 6 is the intersection of the individual umbrae (shaded light). It is here compared to the union of umbrae from the original view cell (shaded dark). As can be seen, point sampling computes superior occlusion and only slightly underestimates exact occlusion.

2.2 Hardware rasterization of shrunk occluders

Using the occluder shrinking principle explained in the last section requires a great amount of point sampling. To speed up computation, the algorithm described here exploits standard graphics hardware for accelerated computation of view cell-to-object visibility. It builds on the occluder shadow work explained in section 1.3 and thus operates on city-like scenes (2.5D). While this type of environment may appear restrictive, it is suitable for the applications we have in mind (urban walkthroughs, driving simulation, games) and typically provides a large amount of occlusion.

Note that the umbrae obtained by shrinking occluders by ε are overly conservative: although the *shape* of the conservative umbra corresponds to the shape of the real umbra, its *size* is too small, i.e., shrinking also moves the shadow boundaries into the interior of the actual umbra (see Figure 6). Luckily, this is exactly the operation needed to provide for conservative rasterization!! It follows that if ε and the length of one cull map grid cell (k) are coupled via the relation $\varepsilon \geq k/2*\sqrt{2}$ [32], rendering shadow polygons need not take into account rasterization errors. For a formal proof of this fact, see the corollary in the appendix.

In practice, ε (along with the cull map grid size) is chosen to trade off accuracy of occluder shadow calculation against total preprocessing time.

2.3 Implications of occluder shrinking

In summary, occluder shrinking solves two important problems in one step:

1. It makes point sampling the umbra conservative for an ε–neighborhood.
2. It makes hardware shadow polygon rasterization conservative.

Since occluder shrinking need only be applied once during preprocessing for each occluder, the actual visibility computation is very efficient for each view cell.

Note that while the algorithm apparently discretizes the view cell, it actually discretizes the complicated occluder interactions (Figure 2), i.e., including merged penumbrae and umbra boundaries defined by reguli. This principle makes it possible to compute occlusion from simple and fast point sampling.

2.4 Algorithm Overview

In the following, we outline the basic preprocessing algorithm:

1. Subdivide environment into convex view cells, choose ε.
2. Shrink occluders to yield conservative visibility with respect to ε. This step also compensates for rasterization errors.
3. Choose a view cell.
4. Determine a sufficient number of sample points for that view cell.
5. Determine a (potentially very large) number of occluders for that view cell (a straightforward implementation would simply select all occluders – for a more sophisticated approach, see section 6).
6. For each sample point, rasterize occluder shadows into a *cull map* using graphics hardware (the rendered occluder shadows from individual sample points are combined in graphics hardware).
7. Traverse cull map to collect visible objects, then proceed to next view cell.

The following sections 3, 4, and 5 will explain the steps of this algorithm in detail, while section 6 presents a hierarchical extension of the algorithm which optimizes processing time.

3. SUBDIVISION INTO VIEW CELLS

The occlusion preprocessing procedure relies on a subdivision of the space of possible viewing locations into cells. For these view cells, per-cell occlusion is later computed.

For our system we chose to use a constrained Delaunay triangulation of free space, modified to guarantee that the maximum length of view cell edges does not exceed a user specified threshold. The actual view cells are found by erecting a prism above each triangle of the triangulation. Note that because of the 2.5D property of occluders, any object found visible from one particular point is also visible from all locations above this point. This implies that sample points need only be placed on the 3 edges bounding the top triangle of the view cell.

As free space is per definition not occupied by occluder volumes (buildings), the inside of occluders need not be considered and occluders cannot intersect view cells. This way we can avoid special treatment of occluders that intersect a view cell and we need not process areas inside buildings. However, the algorithm could also use any other subdivision into view cells.

4. OCCLUDER SHRINKING

The goal of occluder shrinking is to compute a set of occluders that can be used with point sampling to yield conservative occlusion from a view cell.

4.1 Occluder selection

Typical models of urban environments consist of a large number of small triangles. To obtain the volumetric occluders required by our algorithm, our modeling system automatically extracts and stores building facades as occluders, a feature that cannot be assumed to be available in a general application. Occluders obtained this way correspond to a subset of the visual hull (i.e., the maximal object that has the same silhouettes and therefore the same occlusion characteristics as the original object, as viewed from all view cells [20]). Therefore, we do not need to consider irregular features of the facade like doors or windows. Our system handles arbitrary 2.5D-occluders, i.e., heightfields. The footprint of the occluder may be (and usually is) concave.

4.2 General Occluder shrinking

A general polyhedral occluder must be shrunk so that any point of the surface of the shrunk occluder has at least a distance of ε to any point on the surface of the original occluder[2], so that any object occluded by the shrunk occluder from a sample point will also be occluded with respect to the original larger occluder for any point in an ε-neighborhood of the sample point. In practice, rather than directly shrinking the original occluder's geometry, it is easier to take an indirect approach by enlarging the exterior of the occluder and intersecting the enlarged exterior with the occluder.

We describe the algorithm for a 2.5D occluder and show an improved algorithm for the special case of occluders with constant height.

4.3 Occluder shrinking in 2.5D

In 2.5D, occluders can be arbitrary discontinuous functions $z = f(x,y)$ with compact support. Occluders must be given as a triangular irregular network, with heights assigned to the vertices of the triangles and possibly with discontinuities. This is sufficient for city structures; however, building footprints must be triangulated first. The following algorithm computes shrunk occluders:

1. Enclose the occluder in a box larger than the occluder itself and triangulate the whole box area (if not already done)
2. Above each triangle (no matter whether it belongs to the occluder or the ground), erect a triangular prism. The union of all these prisms is equal to the exterior of the occluder with respect to the enclosing box.
3. Enlarge each prism by ε and subtract the enlarged prism from the occluder using standard CSG or BSP Boolean set operations. Note that enlarging a triangular prism is trivial as it is a convex volume.

4.4 Occluders with constant height

Occluders with constant height (buildings with flat roofs) can be represented by arbitrary 2D polygons with a single height value. For such occluders, occluder shrinking can be performed purely in 2D:

1. Enclose the polygon in a box larger than the polygon itself.
2. Triangulate the exterior of the polygon.
3. Enlarge each triangle of the exterior by ε and clip the polygon against the enlarged triangle.

[2] This is in contrast to simplification envelopes [5], where a maximum distance to the original object has to be guaranteed.

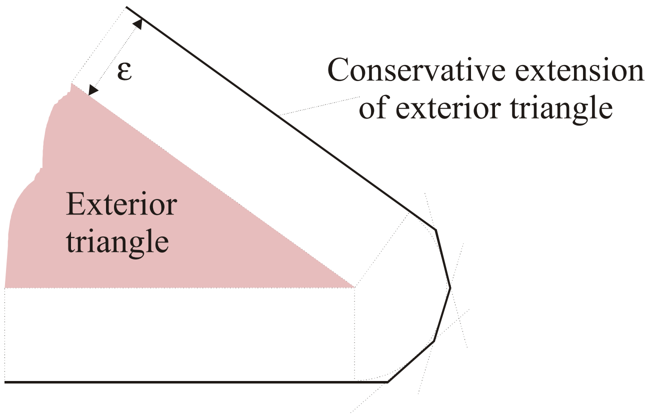

Figure 8: To extend a triangle by ε, the arc at the corner is approximated by 3 tangential line segments.

At the corner points, the ε-neighborhood of such an enlarged exterior triangle consists of arc segments. For a good approximation, these arc segments are approximated by tangent lines. Using 3 tangent lines usually suffices for a good approximation without making the resulting enlarged triangle too complicated.

5. RENDERING AND MERGING OCCLUDER SHADOWS

This section explains how to extend the frame buffer algorithm described in section 1.3 to calculate the umbra from a view cell instead of one sample point only. The idea is to merge umbrae from sample points already in graphics hardware.

The input to this algorithm is a set of occluders OS (represented by occluder *edges* in 2.5D) and a number of point sample locations PS for the chosen view cell. The output is the cull map, an encoding of the umbra stored in the z buffer. A pixel in the z buffer corresponds to a square cell in the scene and the z value of that pixel encodes the height of the umbra for that cell (see Figure 4). The cull map is created by rendering *shadow polygons* computed by the viewpoint and an occluder edge into the cull map.

In 2.5D, the umbra is a function $z = f(x,y)$. Merging umbrae f_1 and f_2 from two occluders $O_1, O_2 \in OS$ for the same viewpoint can be done by calculating the *union*

$$z = \max(f_1(x,y), f_2(x,y))$$

for all (x,y), since for a point in space to be in shadow, it suffices if it is in shadow with respect to one occluder. We use the z buffer of commonly available graphics hardware to represent umbrae by rendering them in an orthographic projection of the scene.

While the umbra from a single point sample can be computed in that way, the umbra of a view cell requires the *intersection* of the umbrae from all sample points $P \in PS$, i. e.

$$z = \min \{ \max \{ f_{O,P}(x,y) \text{ for all } O \in OS \} \text{ for all } P \in PS \}$$

Incremental rendering of the union of occluder shadows is easily performed with a z buffer by rasterizing one occluder shadow polygon after the other. Incremental rendering of the intersection of the union of occluder shadows is only conceptually simple, but not trivial to implement. A solution requires a two-pass rendering effort using a combination of z buffer and stencil buffer. For the first sample point, all shadow polygons are rasterized just like in section 1.3. Then, for each additional sample point, after clearing the stencil value to zero, the following two passes are carried out:

- The first pass identifies pixels where the previously accumulated umbra is already correct. This is, when any pixel from a new shadow polygon is higher than a pixel from the previous umbra. This is done by rendering all polygons from the new sample point and setting the stencil value to 1 if a greater z-value is encountered (z is not written in this pass).
- In the second pass, only pixels that are not yet correct (i.e., pixels that have not been identified in the first pass) are considered (by testing the stencil value for zero). Those pixels are initialized to a zero depth value and updated by simply rendering all shadow polygons from the current sample point again (this time, z values are written).

After all sample points have been considered, the cull map is copied to main memory and potentially visible objects are determined by comparing their maximum z-values against the z-values stored in the cull map.

6. HIERARCHICAL OCCLUSION TEST

For rapid preprocessing of large scenes, rasterizing *all* occluders to obtain a close approximation of exact visibility is not feasible. Quick rejection of large portions of the scene including the contained *occluded occluders* is essential. To achieve this goal, we employ a hierarchical approach.

Starting from an initial size, e.g. 100mx100m, we construct successively larger cull maps (e.g., twice as large in each dimension). At the borders of each cull map, the z buffer values define a cross section through the umbra, the *cull map horizon*. In each iteration, this information is used for the following tests:

1. If all scene objects are found to be invisible with respect to the cull map horizon, the algorithm terminates, and the visible objects can be collected from the cull map ("early break").
2. Occluders for the next iteration which are *inside* the current cull map are tested against the z-values of the current cull map and are rejected if they are found invisible.
3. Occluders for the next iteration which are *outside* the current cull map are tested against the cull map horizon and rejected if found invisible.

To test an object (or occluder) outside the cull map against the cull map horizon for visibility, it is sufficient to guarantee that no sight line exists between the object and the view cell. To quickly evaluate this criterion, we consider the bounding box of an object (or an occluder) and the bounding box of the view cell (see Figure 9). For those two boxes we calculate the two supporting planes on the side and one supporting plane at the top from a bounding box edge of the object to a bounding box edge of the view cell. These three planes define a conservative view channel from the view cell to the object. This view channel is intersected with the cull map horizon. An object is occluded if the computed view channel is completely contained in the horizon.

We use a quadtree of axis aligned bounding boxes for accelerating the bounding box tests, so that together they sum up to less than 5% of the total algorithm computation time.

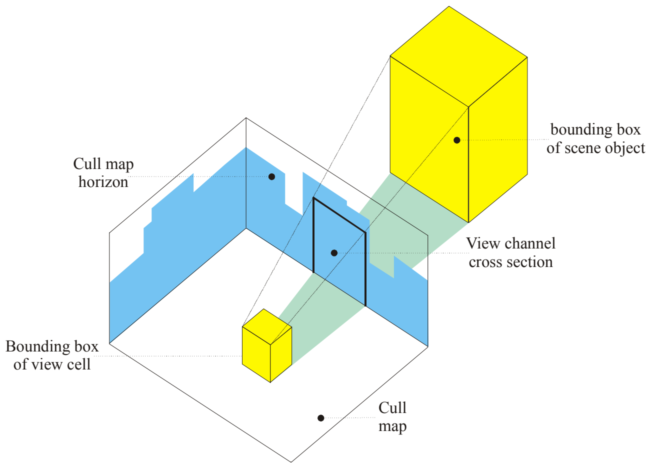

Figure 9: After computing an initial cull map, arbitrary parts of the scene can be quickly rejected by projecting their axis-aligned bounding boxes against the horizon of the cull map.

7. IMPLEMENTATION AND RESULTS

A walkthrough system using the techniques outlined in this paper was implemented in C++ and OpenGL. All tests reported here were run under Windows NT on a Pentium-III 650MHz with 1GB RAM and a GeForce 256 graphics accelerator.

7.1 Preprocessing

A model of the city of Vienna with 7,928,519 polygons was used throughout testing. It was created by extruding about 2,000 building block footprints from an elevation map of Vienna, and procedurally adding façade details such as windows and doors (Figure 14). One building block consists of one up to ten connected buildings. Note that each building block was modeled with several thousand polygons (3900 on average). Building blocks were further broken down into smaller objects (each consisting of a few hundred polygons) which were used as primitives for occlusion culling.

82,300 view cells were considered. The sample spacing constant ε was set to 1m, which led to the use of 6-8 sample points per view cell edge on average. An initial cull map size of 100×100 pixel (with a grid size of 2m in the final walkthrough) was used, with a maximum cull map size of 1500×1200 pixel (tiled) for the hierarchical algorithm.

Preprocessing took 523 minutes. 54 % of the preprocessing time were spent on reading back cull maps from the frame buffer.

7.2 Quality

We experimented with different settings for sample spacing (ε), cull map grid size and cull map dimensions. Figure 10 shows the results for various grid sizes. As can be seen, larger grid sizes also yield acceptable occlusion, allowing to handle larger models. On average, our algorithm identifies 99.34% (~1:150) of the scene as occluded. While these numbers are representative for a real-world data set, arbitrary occlusion ratios can be generated by increasing the depth complexity of the model. More interesting is the fact that the absolute numbers of occluders (in our case, occluder edges) that contribute to occlusion is about 200 on average for our test model, which is quite high. See section 8.2 for a discussion of this fact. Figure 1 demonstrates the effect of occluder fusion.

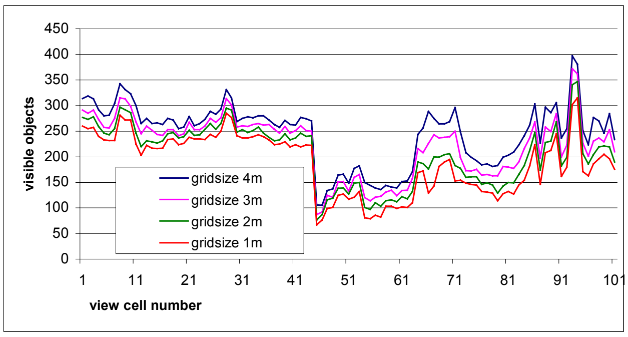

Figure 10: Influence of various grid sizes of the cull map on occlusion quality. Plotted are the number of visible objects (y-axis) against the view cells visited in the test walkthrough.

7.3 Real-time rendering

To assess real-time rendering performance, precomputed occlusion was used to speed up rendering of a prerecorded walkthrough of the model, which was 372 frames long and visited 204 view cells. The model consumed approximately 700 MB and was fully loaded into main memory before the begin of the walkthrough. Occlusion information (object IDs of potentially visible objects for each view cell) consumed 55 MB and was also fully loaded.

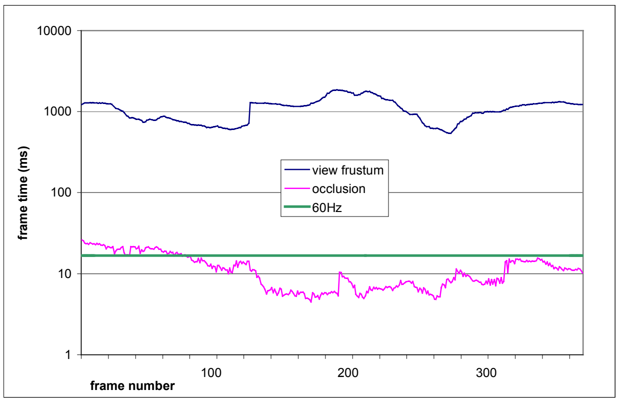

Figure 11: Frame times of occlusion culling vs. view frustum culling for the walkthrough. Note that frame times are plotted on a log scale to fit two orders of magnitude into the diagram.

We measured frame times for two variants

- Full occlusion culling
- View frustum culling as a reference measurement

Occlusion culling provided an average speed-up of 93.78 over view frustum culling. Figure 11 shows the frame times from our walkthrough for occlusion culling vs. view frustum culling (logarithmic scale). Note that the desired 60Hz are achieved for the second part of the walkthrough (dense occlusion), while the first part (wide open view) would require further optimizations (e. g. impostors).

8. DISCUSSION

8.1 Comparison

Competitive methods for view cell visibility have only recently been investigated independently by other authors. We are aware of two approaches: both require that an occluder intersects the accumulated umbra of previously considered occluders for occluder fusion to occur. Schaufler et al. [24] extend blockers into areas already found occluded, while Durand et al. [13] project occluders and occludees onto planes with new extended projections from volumetric view cells.

While these methods work in full 3D, they only consider a subset of occluder interactions handled by our method. Since their spatial data structure only represent *umbra* information, they cannot handle cases such as Figure 2, letter c, for example, where the *penumbrae* of two occluders can be merged, even though there is no joint umbra. In most cases, this will cause a larger number of objects to appear in a PVS than necessary.

We found that complex shadow boundaries arise already in simple cases. Even in 2.5D, a simple occluder with non-convex footprint gives rise to so-called 'EEE event surfaces' [12], ruled quadric surfaces incident on three edges in the scene. Those event surfaces bound the umbra due to the occluder, but are usually not considered in other methods. Our approach discretizes such boundaries through point sampling, which gives a good approximation of the real umbra.

We believe that discretizaton is a reasonable trade-off between correctness and efficiency: our method implicitly handles all types of occluder interactions, and if we decrease ε together with the cull map grid size, our method converges to a correct solution.

Although the idea of occluder shrinking and point sampling is applicable also in 3D, an extension of the hardware-accelerated methods presented in section 5 is not straightforward, which is the main limitation of our algorithm.

8.2 Selection of occluders

Several algorithms achieve good results with a heuristic to select a set of occluders that is most likely to occlude a big part of the scene [8][17]. However, we have found that the number of occluders that contribute to occlusion is typically quite large (as indicated by results presented in section 7.1). Even missing small occluders can create holes that make additional objects visible. For an accurate representation of occlusion, only those occluders that would not contribute to a more exact solution can be discarded. In our case, these are the occluders inside the calculated umbra.

8.3 Use of graphics hardware

Using a large number of occluders requires significant computational effort. However, our preprocessing times are reasonable even for large scenes because the only geometric operation, occluder shrinking, needs to be performed only once during preprocessing. Per view cell operations almost fully leverage the speed of current rasterization hardware, so we can calculate and render up to several hundred thousand occluder shadow polygons per second.

A general problem with hybrid image-based rendering that requires access to the frame buffer is that copying of the frame buffer to memory is slow, especially on low end graphics hardware. While this is less of a problem on high end workstations, we expect a trend towards faster frame buffer copy times even on consumer hardware.

9. CONCLUSIONS AND FUTURE WORK

Visibility preprocessing with occluder fusion is a new method for accelerated real-time rendering of very large urban environments. While it cannot compute exact visibility, no simplifying assumptions on the interaction between occluders or heuristics for occluder selection are necessary. Through point sampling, the proposed algorithm approximates actual umbra boundaries due to multiple occluders more exactly than previous methods, leading to better occlusion.

The measured number of occluders that contribute to occlusion (~200 on average) lead us to believe that simultaneous consideration of a large number of occluders is indeed crucial for achieving significant culling in hard cases where large open spaces are visible. The frame rates from our walkthrough, which are closely below or above the desired 60Hz, show that no significant time is available for on-line occlusion calculation, and that precomputed occlusion is necessary for true real-time performance.

While computation time of a preprocessing step is not strictly critical, it is still an issue in practice. The use of graphics hardware and hierarchical cull map generation makes our algorithm practical even for very large data sets. However, precomputation times of several hours are still inconvenient, which should be addressed in future research.

Future work will also focus on constructing suitable image-based representations for areas of the city where visibility preprocessing alone is not sufficient to guarantee a framerate above 60 Hz. The restriction to volumetric occluders can also be resolved with the introduction of view-dependent occluder simplification.

ACKNOWLEDGEMENTS

This research is supported by the Austrian Science Foundation (FWF) contract no. p-13867-INF. The authors would like to thank Fredo Durand and Gernot Schaufler from MIT for useful discussion, Martin Held and Xinyu Xiang from Stony Brooks for their triangle stripper and triangulation code, Alan Murta from the University of Manchester for a polygon clipper, and Gerald Hummel from TU Vienna for support with the city model.

REFERENCES

[1] J. Airey, J. Rohlf, F. Brooks, Jr. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. Computer Graphics (1990 Symposium on Interactive 3D Graphics), 24(2), pp. 41-50, 1990.

[2] D. Aliaga, J. Cohen, A. Wilson, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stürzlinger, E. Baker, R. Bastos, M. Whitton, F. Brooks, D. Manocha. A Framework for the Real-Time Walkthrough of Massive Models. SIGGRAPH Symposium on Interactive 3D Graphics, pp. 199-206, 1999.

[3] J. Bittner, V. Havran, P. Slavík. Hierarchical Visibility Culling with Occlusion Trees. Computer Graphics International 1998 Proceedings, pp. 207-219, 1998.

[4] J. Clark. Hierarchical geometric models for visible surface algorithms. Communications of the ACM, 19(10), pp. 547-554, 1976.

[5] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright. Simplification Envelopes. SIGGRAPH 96 Conference Proceedings, pp. 119-128, 1996.

[6] D. Cohen-Or, G. Fibich, D. Haperin, E. Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. Computer Graphics Forum (Proceedings of EUROGRAPHICS'98), 17(3), pp. 243-253, 1998.

[7] D. Cohen-Or, A. Shaked. Visibility and Dead-Zones in Digital Terrain Maps. Computer Graphics Forum (Proceedings of EUROGRAPHICS'95), 14(3), pp. 171-180, 1995.

[8] S. Coorg, S. Teller. Real-Time Occlusion Culling for Models with Large Occluders. Proceedings of the Symposium on Interactive 3D Graphics, pp. 83-90, 1997.

[9] S. Coorg, S. Teller. Temporally Coherent Conservative Visibility. Proceedings of the Twelfth Annual Symposium on Computational Geometry 96, pp. 78-87, 1996.

[10] X. Decoret, G. Schaufler, F. Sillion, J. Dorsey. Multi-Layered Impostors for Accelerated Rendering. Computer Graphics Forum (Proceedings of EUROGRAPHICS'99), 18(3), pp. 61-73, 1999.

[11] F. Durand. 3D Visibility. Analytical Study and Applications. PhD thesis, IMAGIS-GRAVIR/IMAG-INRIA, Grenoble, France, 1999.

[12] F. Durand, G. Drettakis, C. Puech. The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. SIGGRAPH 97 Conference Proceedings, pp. 89-100, 1997.

[13] F. Durand, G. Drettakis, J. Thollot, C. Puech. Conservative Visibility Preprocessing using Extended Projections. To appear in SIGGRAPH 2000 Conference Proceedings.

[14] Z. Gigus, J. Canny, R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(6), pp. 542-551, 1991.

[15] N. Greene, M. Kass. Hierarchical Z-Buffer Visibility, SIGGRAPH 93 Conference Proceedings, pp. 231-240, 1993.

[16] P. Heckbert, M. Garland. Survey of Polygonal Surface Simplification Algorithms. Multiresolution Surface Modeling Course, SIGGRAPH, 1997.

[17] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, H. Zhang. Accelerated Occlusion Culling using Shadow Frusta. 13th International Annual Symposium on Computational Geometry (SCG-97), pp. 1-10, 1997.

[18] W. Jepson, R. Liggett, S. Friedman. An Environment for Real-time Urban Simulation. Proceedings of the Symposium on Interactive 3D Graphics, pp. 165-166, 1995.

[19] V. Koltun, Y. Chrysanthou, D. Cohen-Or. Virtual Occluders: An Efficient Intermediate PVS Representation. Proceedings of EUROGRAPHICS'2000, 2000.

[20] A. Laurentini. The visual hull concept for silhouette-based image understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(2), pp. 150-162, 1994.

[21] D. Luebke, C Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. Proceedings of the Symposium on Interactive 3D Graphics, pp. 105-106, 1995.

[22] H. Plantinga, C. R. Dyer. Visibility, Occlusion, and the Aspect Graph, IJCV(5), No. 2, pp. 137-160, 1990.

[23] J. Rohlf, J. Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. SIGGRAPH 94 Conference Proceedings, pp. 381-395, 1994.

[24] G. Schaufler, X. Decoret, J. Dorsey, F. Sillion. Conservative Volumetric Visibility with Occluder Fusion. To appear in SIGGRAPH 2000 Conference Proceedings.

[25] G. Schaufler, W. Stürzlinger. A Three-Dimensional Image Cache for Virtual Reality. Computer Graphics Forum (Proceedings of EUROGRAPHICS'96), 15(3), pp. 227-235, 1996.

[26] J. Shade, S. Gortler, L. He, R. Szeliski. Layered Depth Images. SIGGRAPH 98 Conference Proceedings, pp. 231-242, 1998.

[27] J. Shade, D. Lischinski, D. Salesin, T. DeRose, J. Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. SIGGRAPH 96 Conference Proceedings, pp. 75-82, 1996.

[28] F. Sillion, G. Drettakis, B. Bodelet. Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery. Computer Graphics Forum (Proceedings of EUROGRAPHICS'97), 16(3), pp. 207-218, 1997.

[29] A. Stewart. Hierarchical Visibility in Terrains. Eurographics Rendering Workshop 1997, pp. 217-228, 1997.

[30] S. Teller. Computing the antipenumbra of an area light source. Computer Graphics (SIGGRAPH 92 Conference Proceedings), vol. 26, pp. 139-148, 1992.

[31] S. Teller, C. Sequin. Visibility preprocessing for interactive walkthroughs. Computer Graphics (SIGGRAPH 91 Conference Proceedings), vol. 25, pp. 61-69, 1991.

[32] P. Wonka, D. Schmalstieg. Occluder Shadows for Fast Walkthroughs of Urban Environments. Computer Graphics Forum (Proceedings of EUROGRAPHICS'99), pp. 51-60, 1999.

[33] H. Zhang, D. Manocha, T. Hudson, K. E. Hoff. Visibility Culling Using Hierarchical Occlusion Maps. SIGGRAPH 97 Conference Proceedings, pp. 77-88, 1997.

Appendix: A proof for conservative point sampling

Let an occluder O be an arbitrary connected subset of R^3. A *shrunk occluder* O' is a subset of O so that for all points $A \in O'$: $|B-A| \leq \varepsilon \rightarrow B \in O$.

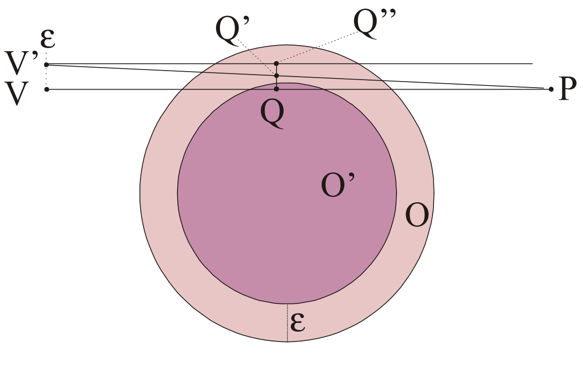

Figure 12: Any point P hidden by a shrunk occluder O' from V is also hidden by the original occluder O from any point V' in an ε-neighborhood of V

Theorem: Any point P that is occluded by O' seen from a sample point V is also occluded by O from any sample point V' with $|V'-V| \leq \varepsilon$ (Figure 12).

Proof: Assume there is a V' for which P is not occluded by O. Then any point along the line segment V'P must not be contained in O. Since P is occluded, there is at least one point $Q \in O'$ along the line segment VP. VV'P form a triangle and $|V'-V| \leq \varepsilon$, so there must be point Q' along V'P with $|Q'-Q| \leq \varepsilon$. By definition, all points within an ε-neighborhood of Q are contained in O, so Q' is contained in O. Therefore P cannot be visible from V'. q.e.d.

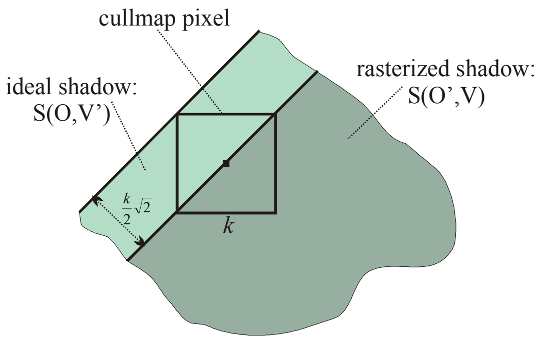

Figure 13: A pixel together with an actually rendered shadow polygon S(O',V) (shaded dark) rendered from sample point V for a shrunk occluder O', and the additional part of an ideal shadow polygon S(O,V') (shaded light) as it would be rendered by infinitely precise rendering hardware from some point V' with $|V'-V| \leq \varepsilon$ for the original occluder O. The pixel shown is filled by hardware because S(O',V) covers the pixel center. If $\varepsilon \geq k/2*\sqrt{2}$, all of the pixel is actually covered by S(O,V') and rasterization is correct.

Corollary: A viewing ray starting in V' which is *parallel* to VP must also intersect O, as there must also be a point Q'' along the parallel ray with $|Q''-Q| \leq \varepsilon$, which again implies $Q'' \in O$. It follows that if a rasterized shadow polygon line goes through the center of a pixel (causing the pixel to be shaded by rasterization hardware), all parts of the pixel with a distance to the line $\leq \varepsilon$ are guaranteed to belong to the ideal shadow polygon which would have been cast by V' through O. Therefore O' also yields conservative results with respect to rasterization error if $\varepsilon \geq k/2*\sqrt{2}$, where k is the length of one cell of the cull map, as $k/2*\sqrt{2}$ is exactly the maximum distance of two parallel lines through the same pixel where one of them passes through the pixel center (see Figure 13).

Note that it can be shown that conservative results can already be had if $\varepsilon \geq k/2$, but a proof for this is more involved and cannot be represented in such confined space.

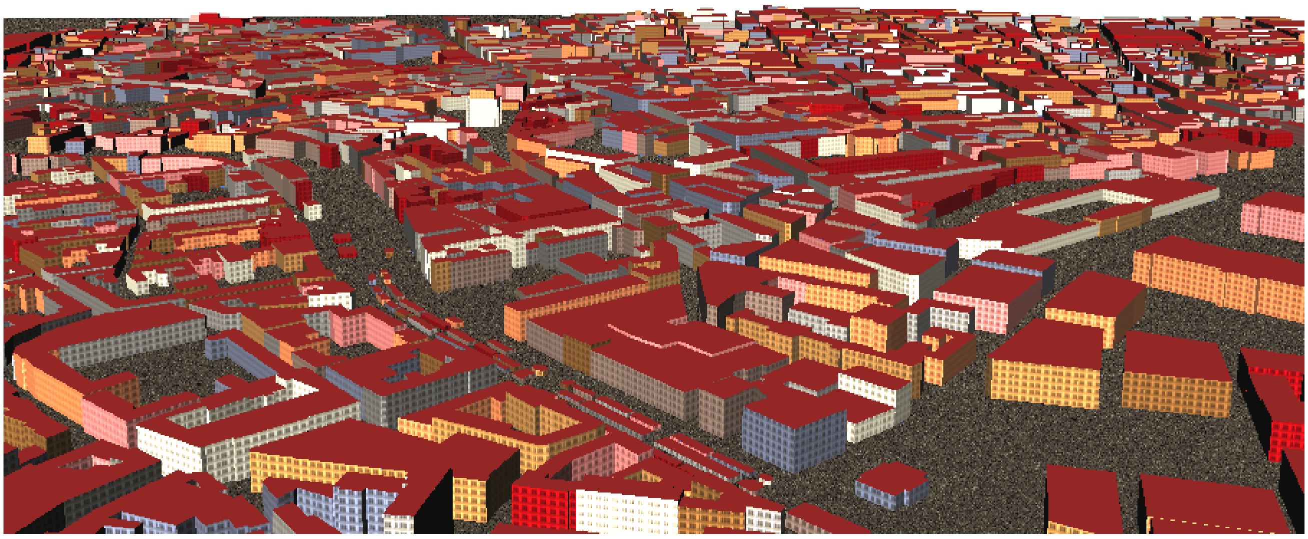

Figure 14: Overview of the 8 million polygon model of the city of Vienna used as a test scene. Note that the city is modeled in very high detail – every window, for example, is modeled separately with 26 polygons