GEOMETRIC SIMPLIFICATION FOR EFFICIENT OCCLUSION CULLING IN URBAN SCENES

Rick Germs & Frederik W. Jansen
Computer Graphics & CAD/CAM Group
Faculty of Information Technology & Systems
Delft University of Technology, The Netherlands
e-mail: {h.m.l.germs|f.w.jansen}@its.tudelft.nl

ABSTRACT

Most occlusion culling algorithms select a subset of suitable occluder planes or geometries to exclude invisible objects from further visualization processing. Preferably these occluders are large and simple. For complex scenes it is worthwhile to generate virtual occluders to replace complex occluder geometries by simple polygonal structures. In urban scenes, the facade of buildings comprises most of the occlusion potential. In this paper we present an algorithm to extract simplified facades from complex building models, exploiting the fact that, in most cases, buildings have a 2.5D structure.

Keywords: walkthrough visualization, occlusion culling, virtual occluders, geometric simplification.

1. INTRODUCTION

Interactive visualization of large 3D polygonal models is a task that is still beyond the capacity of today's computer systems. Even though computers will be developed that are faster and more efficient, there will always be the need to visualize larger 3D models. Successful ways to speed up visualization aim at lowering the polygon-count at each frame, thereby reducing graphics hardware load. Viewing frustum culling, backface culling and level of detail switching are techniques that are widely used to achieve this. Viewing frustum culling is the removal of polygons that are not in the field of view. With backface culling, all polygons that are on the back of an object, and therefore cannot be seen, are removed. Level of detail switching replaces a complex, detailed object with a model consisting of less polygons when it is expected that details cannot be seen in the final image (e.g. when the object is far away from the viewpoint). All these techniques address the problem of removing polygons that do not contribute to the final image. Another approach, which recently attracts a lot of attention, is *occlusion culling*. Occlusion culling is the technique of identifying and removing polygons that are hidden behind ("occluded by") other objects in a scene. It is, in fact, a hidden surface removal algorithm that pre-computes visibility of polygons before rendering, to reduce the load on the image space hidden surface removal done in graphics hardware. Although in theory occlusion culling can remove a large number of invisible polygons, the computation cost is often high. This can be explained by the fact that it is not trivial to efficiently perform the *visibility test* needed to determine which surfaces are completely occluded by other surfaces. Most existing algorithms select a limited number of surfaces, which are suitable as *occluders* for the other surfaces in the scene. Preferably, these occluders are large, so they have the potential of occluding a large part of the scene. Unfortunately, most scenes do not contain enough large occluders to efficiently perform occlusion culling. Therefore, it may be advantageous to find *virtual occluders* [Law99], e.g. by simplifying the geometry of occluding objects, and to replace sets of small occluders with one large polygon. In particular for urban scenes, the front faces of houses and rows of buildings are crucial for efficient culling of the city parts that are hidden behind them.

This paper presents an algorithm to generate simplified, occlusion preserving, virtual occluders in urban scenes. These virtual occluders are generated using *footprints*, which represent levels in a 3D model. Using several footprints for a given model, a simplified representation can be generated, which we call a *facade*. Facades can also be used as a low-detail model in level of detail switching.

The paper is organized as follows. In chapter 2, we take a look at some previous work in the fields of occlusion culling and occlusion preserving geometric simplification. We present our facade

algorithm in chapter 3. Implementation and results are discussed in chapter 4, and chapter 5 concludes the paper.

2. PREVIOUS WORK

In the last decade, a lot of work has been done in the field of occlusion culling and geometric simplification. Much less research has been focussed on the construction of occlusion preserving geometric simplification. In this chapter, we first discuss some of the recently developed occlusion culling algorithms, and show how occlusion preserving geometric simplification can be used to improve their efficiency. Next, we look at the occlusion preserving simplification method proposed by [Law99].

2.1 Occlusion Culling

In 3D scene rendering, for example in urban walk-through visualization, many objects (e.g. buildings) are hidden behind other objects for a given viewing position. Although these objects are invisible, the rendering algorithm still has to process them, which can be seen as a waste of computation effort. *Occlusion Culling* is a technique that tries to identify hidden objects or *occludees*. The occludees can then be excluded from the rendering process. In occlusion culling, two basic approaches can be distinguished; *image space* and *object space* methods, indicating the space in which the occlusion culling calculations take place. Both approaches use a set of nearby, visible objects called *occluders* to perform their calculations. Image space methods usually make use of graphics hardware to render an occluder. Occludees are identified through image space overlap and Z-buffer tests. With object space methods, the edges of an occluder (e.g. a polygon) are used to generate a number of planes, which indicate the volume that is occluded. Potential occludees that are completely contained by the volume can be culled away.
In the rest of this section, we will discuss some of the occlusion culling algorithms available today.

Greene's *hierarchical Z-buffer* algorithm [Greene93] is based on an image space quadtree (*z-pyramid*), in which the farthest z-value for a corresponding rectangular part of the image is stored at each node. Objects in the scene are grouped in an octree. The z-pyramid is generated by rendering objects in octree cells that are likely to be visible or were tagged visible in the previous animation frame. The visibility of the other objects is tested by rendering the forward facing planes of the octree cells bounding the objects. Note that the algorithm does not require selection of suitable (i.e. large) occluders.

The *hierarchical occlusion map* (HOM) introduced in [Zhang97] also performs its visibility test in image space. Potentially visible surfaces are rendered in an image map, from which a hierarchy of occlusion maps (image pyramids) is generated. The hierarchical occlusion maps are used as occluders for the surfaces that are likely to be hidden (occludees). For each possible occludee, a bounding rectangle overlap test is performed with the pixels of the occlusion maps. This, together with a simple occluder/occludee depth comparison, comprises the run-time visibility test. The number of occluders that can be used each frame is limited. Replacing occluders with a simplified, occlusion preserving geometry would allow more occluders to be taken into account each frame, possibly resulting in more objects being culled.

Coorg and Teller [Coorg97] propose an object space algorithm that selects a small number of convex polygons as occluders each frame. The occluder selection mechanism selects polygons that are thought to have a high *occlusion potential*, indicated by the polygon's solid angle from the viewpoint. The scene is subdivided using a kD-tree, where the nodes of the tree are candidate occludees. As the tree is traversed, each node is subject to the visibility test, which is performed in object space. Visibility is determined by the use of *supporting* and *separating* planes, which are generated for each edge of an occluder polygon. A possible occludee has to be tested to each of the planes to determine its visibility. So, the occluder/occludee visibility test will be more efficient, if scene objects are simplified into large occlusion polygons with a low edge count.

The algorithm developed by Wonka [Wonka99] is aimed at walk-through visualization in 2.5D environments, such as urban scenes. It uses two regular (2D) grids, a scene grid and an occluder grid, which can be of different resolution. The occluder grid is used to store the *occluder polygons*, which are selected in a pre-processing step. Effectively, suitable occluder polygons are large, convex and must be perpendicular to the ground plane. For every top edge of an occluder polygon, a so-called *occluder shadow* is rendered in a hardware *cull map*. The occluder shadow is a plane through the viewpoint and an occluder edge, covering all possible occludees behind the occluder polygon, as seen from the viewpoint. Each pixel in the cull map corresponds with a cell in the scene grid. The visibility test consists of a comparison of the z-values of the objects in a scene grid cell and the z-value stored in the cull map. For this test, standard graphics hardware can be used, which allows run-time occlusion culling for real-time walkthrough visualization. For real-time performance, only a limited number of suitable occluder polygons can be selected during the pre-processing step. Here, a

geometric simplification algorithm optimizing occlusion culling should generate large, convex polygons, which are perpendicular to the ground plane and have a small number of occluder edges.
The occlusion culling methods described above perform the visibility test in run-time, but there also exist algorithms that pre-compute visibility for every cell in an object space subdivision data structure. For example, in [Saona99], pre-processed lists of visible bounding boxes (containing groups of objects) are stored at the leaf nodes of an octree. The run-time algorithm consists of selecting the correct list and rendering the objects in the list. In [Zhang97], visibility pre-processing can optionally be used to improve the dynamic occluder selection algorithm. The run-time algorithm proposed in [Wonka99] can also be applied in a visibility pre-processing step [Wonka00], where point sampling is used to determine visibility for parts of the scene.

2.2 Occlusion Preserving Geometric Simplification

In [Law99], Law & Tan replace complex occluder geometry by simplified *virtual occluders*. Virtual occluders are created from the lowest level node in the level-of-detail hierarchy of an occluder object. To ensure that the virtual occluder does not occlude other objects than the original model, an algorithm called *edge error correction* is applied. Edge error correction moves all edges of a simplified model inside the boundary of the original model individually. Although this does not always result in a valid geometry (the edges of a polygon are not always coplanar), it can be very well be used as an occluder (e.g. with the visibility tests proposed in [Coorg97]). The edges are moved individually, to preserve most of the original model's occlusion potential. Next to this virtual occluder algorithm, Law & Tan also implement an occlusion culling scheme, where lists of occludees for each cell of an object space subdivision are created in a pre-processing step.

Looking only at the occluder creation part of the algorithm, one of the advantages is that it can use several existing (and extensively tested) geometric simplification algorithms for creating occluders with few, but large polygons for any (convex or non-convex) geometry. Second, with edge error correction, the virtual occluders can maintain a very high occlusion potential. But, because a virtual occluder may be an invalid geometry, it can only be used for visibility processing. So, it must always exist next to a LOD hierarchy, instead of being part of the hierarchy. Although the virtual occluders are highly occlusion preserving, there is no control over the size and orientation of individual polygons (unless the simplification algorithm used can provide this). This makes it less suitable for application in run-time occlusion culling algorithms such as [Wonka99].

3. FACADE: A GEOMETRIC SIMPLIFICATION FOR EFFICIENT OCCLUSION CULLING

This section introduces a new approach for optimizing 3D polygonal models into fast occluder geometries. Generally speaking, most object space and some image space occlusion culling algorithms will benefit from simplification of complex occluders into simple geometries consisting of large, convex polygons with a low vertex count. At the same time, we have to preserve the contours, and therefore the occlusion potential, of the original model. Apart from that, it may be advantageous to restrict the orientation and shape of the polygons generated by the simplification algorithm to suit a specific occlusion culling algorithm or application. It may therefore be more efficient to use a special purpose simplification, aimed specifically at urban scenes.

The *facade* approach presented here is aimed at simplifying geometric models of buildings for occlusion culling in walkthrough visualization of urban scenes.

3.1 Overview

Looking at an architectural model in urban walk-through visualization, we can say that its "facade" and generates most of the model's occlusion potential. Unfortunately, the geometry of an architectural model, including its facade, is usually very complex. We propose a geometric simplification algorithm that generates an occlusion preserving model consisting of only the simplified facade of a building. We do this by exploiting the fact that most parts of 3D architectural models have a 2.5D type of structure. Most buildings can be seen as a stacked number of blocks. A block can be described by a 2D contour (a footprint) with an associated height.

To identify the blocks that compose a building, we must find parts of the geometry that have more or less the same height. Looking along the vertical (Z) axis of a building's geometry, a number of levels can be found, which we will refer to by the name *Z-levels*. For each of the isolated Z-levels, we can create a so-called *footprint*. A footprint can be defined as the 2D projection of a 3D model on the plane that represents the model's ground surface (Fig. 1b). In conventional geographic information

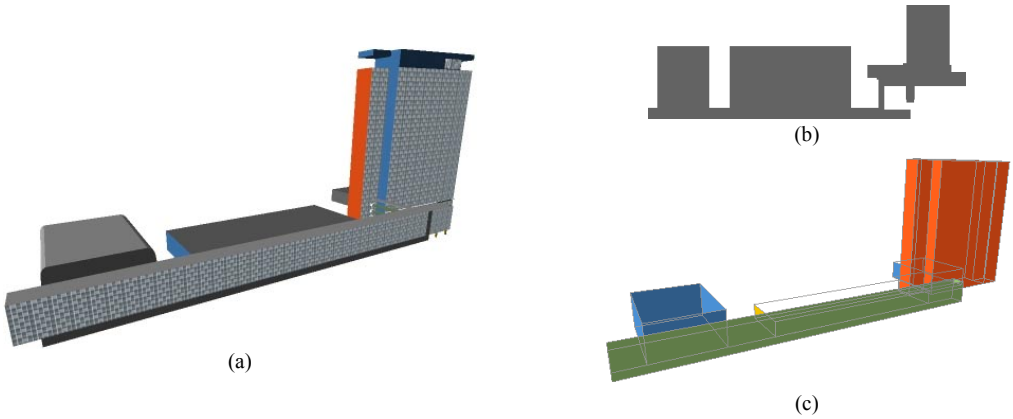

Figure 1: A 3D model of a building (a), its footprint (b) and facade (c).

systems (GIS), which usually work with 2D data (maps), footprints of 3D objects (e.g. newly built buildings) are regularly used to represent the contour of the object on a map. Once we have generated the footprints for each of the Z-levels of a building, we can generate a 2.5D geometry, which we will call a *facade* (Fig. 1c). The facade has to be completely contained by the original model, to ensure that it will occlude only the set of objects occluded by the original model.
So, the procedure of generating facades for occlusion culling is:

1. find the Z-levels of a building (Z-level identification),
2. create a footprint for each of the Z-levels,
3. generate a facade from the set of footprints and Z-levels.

We will elaborate the identification of Z-levels in section 3.2. In section 3.3, we look at the process of creating a 2D footprint. Last, in section 3.4, the creation of a facade using a set of footprints is explained.

3.2 Z-level Identification

Z-level identification is the process of extracting the most significant Z components of the basic 2.5D structure of a 3D architectural model. This (basic 2.5D) structure is mainly characterized by the non-vertical sections of the model's boundary (e.g. roofs, balconies). In this paper, we will look at the case of identifying Z-levels in building models that are completely 2.5D (i.e. having no overhanging parts, like a balcony). The identification of the Z-levels of a model globally consists of two steps:

1. Creating the set of polygons that form the non-vertical sections of the model's boundary (i.e. with a face normal pointing upwards). All vertically oriented polygons are discarded.
2. Clustering this set into Z-levels, where all polygons in a Z-level are at the same height within a tolerance T_Z.

The number of Z-levels that is generated can be controlled indirectly, using the tolerance T_Z. Using a

low T_Z, the number of levels will increase, and so will the occlusion potential of the resulting facade. But, using more Z-levels also means that the facade will consist of more (and smaller) polygons. By increasing T_Z, we can lower the number of Z-levels, but then we might lose too much occlusion potential. An adequate solution to the problem would have to find the optimal balance between facade polygon count and occlusion potential. Computing such an optimal solution would require a sophisticated algorithm. For the moment, we intend to use a sub-optimal, but quicker method. The main idea of this method is that some Z-levels add more to the occlusion potential of the facade than others will. For example, a Z-level consisting of only a few small polygons will not influence the occlusion potential of the facade much, while still adding to the facade's polygon count. The method works as follows:

1. Cluster the set of polygons into Z-levels using a relatively low T_Z, such that extra Z-levels are generated.
2. Calculate the *occlusion benefit* for a Z-level.
3. Merge Z-levels, or delete them on the basis of their occlusion benefit.

The occlusion benefit of a Z-level can be described as a measure for the occlusion potential that the level adds to the occlusion potential of a subsequent (lower) level. If the level does not generate much additional occlusion potential, it can be deleted. Otherwise, the algorithm can either decide to keep the two separate levels, or merge them on the basis of their mutual occlusion benefit and distance ΔZ.

(a) (b)

Figure 2: The metric used for the occlusion benefit of a Z-level L_2, compared to L_1 (a), is based on the surface area of the non-intersecting parts of the facade geometries defined by the Z-levels (b).

If both occlusion benefit and ΔZ are high, both levels are kept. If only the occlusion benefit is high and ΔZ is considered small, the Z-levels are merged. The occlusion benefit of one Z-level compared to another Z-level can be calculated from the non-intersecting parts of the facade geometries generated from the levels (Fig. 2). This can easily be seen from the fact that these parts may occlude different objects, whereas the intersecting parts will occlude the same objects. So, a metric indicating the occlusion benefit of one Z-level compared to another would have to be related to the non-intersecting parts.

Finding a simple, but suitable *metric* for the occlusion benefit (or occlusion potential) is not straightforward, because a 3D object will have a different occlusion effect from different viewpoints. The solid angle of an object is a good metric for its occlusion potential [Coorg97], but it is bound to the viewpoint. Here, we want to find a metric that is independent of the viewpoint. A simple, but adequate metric would be the area of an object's visible surface[1]. For convex objects, we can state that a larger visible surface area generally generates a higher *overall occlusion potential*. With the overall occlusion potential, we want to indicate the quality of an object as an occluder, independent of the viewpoint.

Now, we apply this metric to refine the choice of Z-levels. For two Z-levels $\mathbf{L_1}$ and $\mathbf{L_2}$ (Fig. 3), this is done as follows:

1. Intersect $\mathbf{L_1}$ with $\mathbf{L_2}$, resulting in intersecting and non-intersecting parts for $\mathbf{L_2}$ (Fig. 3b)
2. Determine the (2D) convex hull for the intersecting and non-intersecting parts of $\mathbf{L_2}$, and the convex hull of $\mathbf{L_1}$ (Fig. 3c). As explained before, this is done because concave parts enlarge the visible surface area, but do not contribute to the occlusion potential of the Z-level.
3. Calculate the perimeter $\mathbf{P_{1\cap 2}}$ of the intersecting part(s), the perimeter $\mathbf{P_{2\text{-}1}}$ of the non-intersecting part(s) of the convex hull of $\mathbf{L_2}$ and the perimeter $\mathbf{P_1}$ of the convex hull of $\mathbf{L_1}$ (Fig. 3c).
4. If $\mathbf{Z_1}$ is the height of $\mathbf{L_1}$, and $\mathbf{Z_2}$ is the height of $\mathbf{L_2}$, we can calculate the surface area (or occlusion benefit $\mathbf{B_{2,1}}$) that $\mathbf{L_2}$ adds to the facade defined by $\mathbf{L_1}$ (Fig. 3e):

   $$\mathbf{B_{2,1} = Z_2 \cdot P_{2\text{-}1} + (Z_2 - Z_1) \cdot P_{1\cap 2}}$$

5. To decide whether to keep both levels, merge them or delete the higher level $\mathbf{L_2}$, we compare the occlusion benefit $\mathbf{B_{2,1}}$ with the surface area of the facade defined by (the convex hull of) $\mathbf{L_1}$ using a threshold value $\mathbf{T_B}$. Now, Z-level $\mathbf{L_2}$ is said to add significantly to the occlusion potential of Z-level $\mathbf{L_1}$, if:

   $$\mathbf{B_{2,1} > T_B \cdot A_1}, \qquad \text{where } \mathbf{A_1 = P_1 \cdot Z_1}$$

We can extend the basic method, e.g. by accounting for the number of facade polygons generated by the individual levels or by introducing weights of importance to preserve lower levels. The first will improve the overall occlusion potential of the facade against the cost of some extra polygons. By preserving lower levels, we can exploit the fact that these levels occlude more objects in urban walkthrough visualization.

3.3 Generating a footprint

In this section, we explain the method we used to create a footprint of a 3D model for use with occlusion culling. Since the input required by the footprint algorithm is just a set of polygons, we can apply the algorithm to the polygons grouped in a Z-level without any adjustments.

If we assume that we want the footprint to be a projection onto the XY-plane, then we first select all polygons of the model that have a normal $\mathbf{N}$ with $|\mathbf{N_z}| > \varepsilon$. We then have a set of upward or downward oriented polygons. The edges of these polygons are then inserted into a quadtree. The quadtree is used to localize intersection calculations. The footprint algorithm starts by selecting the edge closest to one

[1] We must use the area of the convex hull of the object to be exact. This is because concave parts will enlarge the surface area of the object, but they do not contribute to its occlusion potential. The convex hull of a façade can be calculated by extruding the convex hull of the Z-level footprints that define the façade.

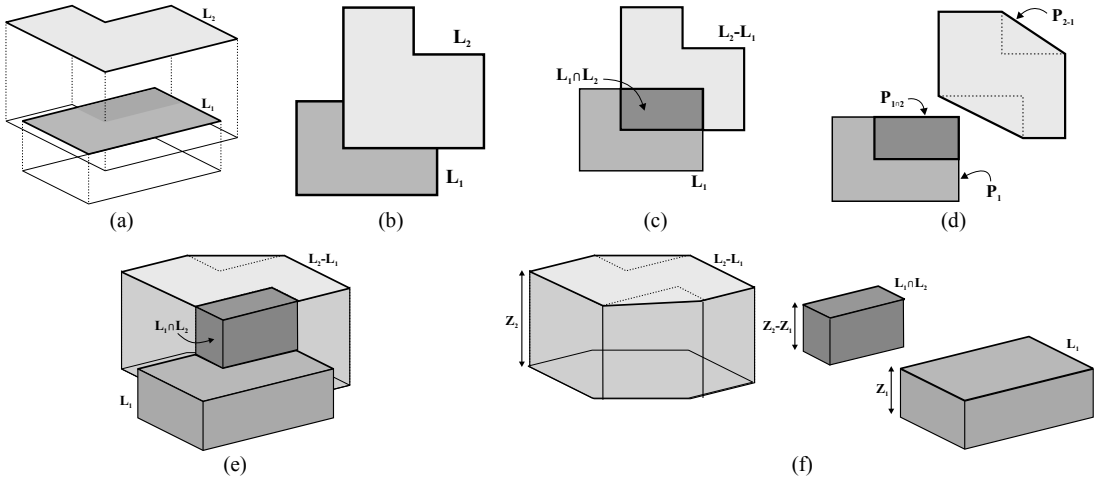

Figure 3: Occlusion benefit calculation. The two Z-levels $\mathbf{L_1}$ and $\mathbf{L_2}$ (a), (b) are intersected in 2D (c), and then the perimeter of the convex hull of the resulting parts is determined (d). These parts are extruded (e), (f) showing the surface area (vertical faces only) that defines the occlusion benefit. **Note:** in (d) and (f), an exploded view is used for clarity.

of the corner points of the quadtree border. It then tracks the contour of the projected 3D model by intersecting edges stored in the quadtree cells, traversing the tree along the contour. The algorithm stops when there are no edges left outside the contour, which we now call a footprint. A footprint is allowed to have multiple (non-intersecting) parts.

Footprints are generated to create a facade, which has to be an efficient occluder (i.e. consisting of few but large polygons). Each vertex in a footprint accounts for two triangles (or one quad) in a facade (section 3.4), and thus, *footprint simplification* will produce more efficient occluders. There are, however, two basic rules that must be followed in the simplification process:

1) the simplified footprint must be completely contained by the original footprint. Only then, we can assume that the facade generated from the footprint does not occlude other objects than the original model.

2) the occlusion potential of the facade generated from the footprint must be preserved as much as possible.

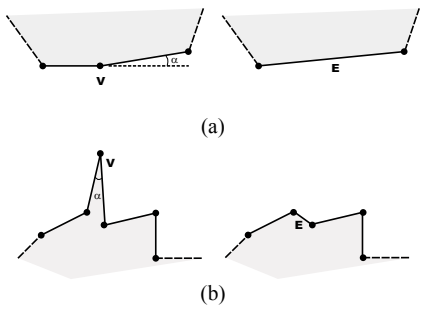

Figure 4: Footprint simplification: redundant vertex (a) and edge removal (b).

Footprints are simplified by removing more or less "redundant" vertices and edges (Fig. 4). A vertex **V** and its connected edges are removed if the lines through these edges are at an angle $\alpha < \varepsilon$ (Fig. 4a, 4b). This satisfied rule no. 1 for both redundant vertex and edge removal. As long as α is small, removing redundant vertices does not effect the occlusion potential much (satisfying rule no. 2). But, for redundant edge removal, we must use a constraint to preserve occlusion potential. The constraint is that we only remove an edge if the distance between **V** and the center-point of the edge **E** is below some threshold distance. (Note: this constraint can also be applied on redundant vertex removal. This is not needed however, because using a small α already guarantees a small loss in occlusion potential.)

3.4 Generating a Facade for Optimized Occlusion Culling

We have now shown how we extract a number of Z-levels from a 3D model of a building, and how we create a 2D footprint from the polygons that comprise a Z-level. The last step is to create a facade geometry from the (simplified) footprints.

For the moment, we intend to support only 2.5D objects (buildings without overhanging parts). A facade is created by extruding each of the input footprints from their Z-level to the groundplane of the model (Fig. 5a, 5b).

There are two basic facade types, the *open* and the *closed* facade. An *open* facade is a set of 4-vertex polygons (called *walls*), that are all perpendicular to the groundplane (Fig. 5b). This type of facade can be used as an occluder, or, when textured, as a low-level of detail model in (urban) walkthrough visualization. A *closed* facade consists of slightly more polygons,

because it also has a "roof", consisting of triangles (Fig 5c). When textured, it is suitable as a low level-of-detail representation in a fly-over visualization of an urban scene.

For occlusion culling purposes, the open, *wall-oriented* facade will probably perform best (Fig. 5d). For this type of facade, each of the walls of the facade can have its own height. This of course increases the occlusion potential of the facade, against the cost of storing a Z-coordinate with each wall. To insure that the object does not occlude other objects than the original model, the height of the wall is set to the minimum Z found along the edge defining the wall (which is always equal to, or higher than the Z of the entire footprint).

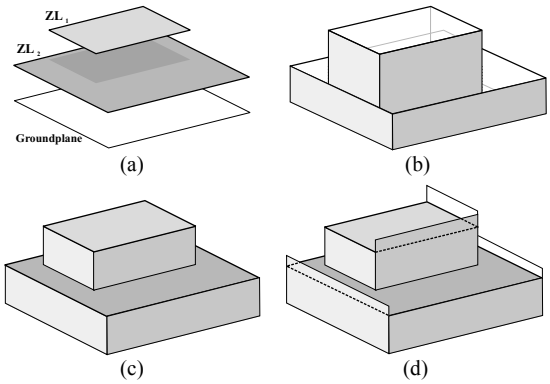

Figure 5: Facade creation. The Z-level footprints (a) can be used to create an open (b), closed (c) or "wall-oriented" facade (d).

4. IMPLEMENTATION AND RESULTS

We tested the algorithm on two building models. For the purpose of these tests, the complexity of the models was low, and the models did not have any overhanging parts, or holes. Furthermore, no footprint simplification was applied, and we only tested "open" facades. The test results shown in Table 1 should therefore be regarded as just an indication of what can be expected from the algorithm. The reason for this is, that at this time, the implementation is not robust enough to handle every input model.

A facade can be tested by determining its quality as an occluder. This quality is reflected by the following properties:

1. the occlusion preserving quality of the facade compared to the original model,
2. the efficiency with which the facade can be used as an occluder, compared to the original model.

4.1 Occlusion preserving quality

Since the solid angle is a suitable metric for the occlusion potential [Coorg97], we can measure the occlusion preserving quality of a facade by sampling its solid angle from a number of viewpoints, and then repeat the procedure with the original model. The average solid angle of both samples can then be compared. The final occlusion preserving indicator can then be a percentage showing the overall occlusion potential of the facade compared to the original model. We can do this, because the test models did not have any overhanging parts or holes, and therefore, a facade will always be completely contained by this model.

In the actual tests, we rendered a series of binary images of the original model and the facade, showing a black object on a white background. By counting the number of black pixels, we got an indication of the solid angle, or rather the occlusion potential of the object for each of the chosen viewpoints. The viewpoints were uniformly distributed along the edge of a horizontal circle, located at half the height of the original model. The radius of the circle was set to a fixed value, such that the entire model would be visible in each of the images. We can justify this approach, because of the fact that the use of facades as an occluder is mainly in walkthrough visualization, and the models we considered did not have any overhanging parts or holes. To measure the occlusion preserving quality of the facade, we need to get an idea of the overall occlusion potential of both the original model $\boldsymbol{O_{model}}$, and the facade $\boldsymbol{O_{facade}}$, and then compare these values. For this, we first find the maximum occlusion potential of the original model as a reference value (we do not want to determine the quality of the model as an occluder, we want to measure the occlusion preserving quality of the facade). We find $\boldsymbol{O_{model}}$ and $\boldsymbol{O_{facade}}$ by comparing the sampled occlusion potential values with this maximum value. Finally, the occlusion preserving quality is given as a percentage, by calculating $\boldsymbol{O_{facade}/O_{model}}$ * 100%. In both test cases, the overall occlusion preserving quality of the facade (which is an open facade) is between 80% and 90%, which is acceptable, considering the increase in occluder efficiency (Table 1).

4.2 Occluder efficiency

The efficiency with which a facade can be used as an occluder is related to the number of polygons of the facade, and the average area of these polygons. It is clear that an object with many, small polygons will be a less efficient occluder than an equally sized object consisting of only a few, very large polygons. In an ideal test case, the original model is part of the first category of objects, and the facade is part of the second group. It is clear from Table 1 and Fig. 6, that the facade is a much more efficient occluder geometry in both test cases.

	Original model		Facade		
Name	*Polygon count*	*Average poly area*	*Polygon count*	*Average poly area*	*Occlusion Preserving Quality*
House	714	36.0	22	2918.5	89.2%
Station	1730	2.02	28	461.4	84.4%

Table 1: Results showing the polygon count, the gain in average polygon area and the occlusion preserving quality for the two test cases. The average polygon area and polygon count values give an indication of the efficiency of the facade as an occluder.

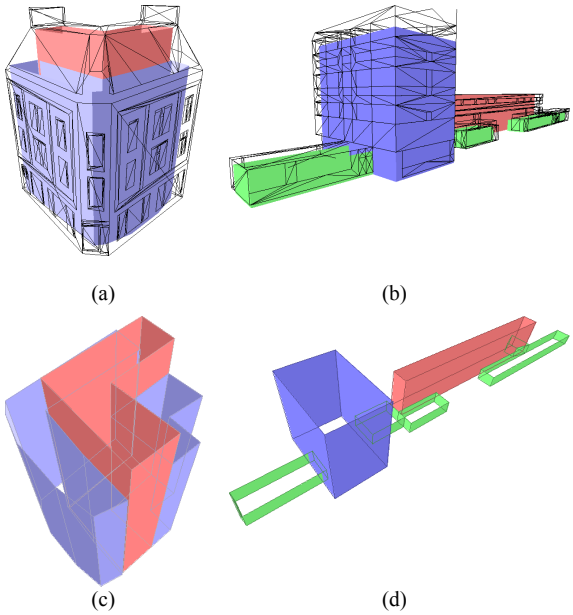

Figure 6: The facade generated for the "House" (a) and (c), and the "Station" model (b) and (d). In (a) and (b), a wireframe rendering of the original models is shown.

5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

Although the presented test results are preliminary, we can draw some conclusions on the potential of the facade approach. Our main goal was to create a simplified geometry that would enable more efficient occlusion culling. This means that a facade has to be a geometry consisting of much less polygons with a relatively large area, compared to the original model. Also, the facade has to be a conservative occluder, preserving most of the occlusion potential of the original model. Looking at the (preliminary) results presented in this paper, we conclude that the facade approach will produce promising results, especially when we incorporate footprint simplification and more occlusion preserving features (e.g. by using the wall-oriented facade) into the algorithm.

5.2 Future Work

The first topic we will look at is the robustness of the footprint tracking algorithm. The problems in that area are probably due to floating point inaccuracies. Next, the algorithm must be extended to handle building models with overhanging parts. This will be quite a complex task, since we aim to be able to support any input model, without any knowledge about the model structure.

Finally, we will test the run-time efficiency of the facade representation, within an occlusion culling algorithm. Since the facade algorithm is aimed at simplifying building models, we will use an occlusion culling scheme particularly aimed at urban walkthrough visualization, which is based on the urban environment occlusion culling algorithm proposed by [Wonka99]. Since this algorithm is specifically designed to exploit the 2.5D character of urban scenes, we expect that it will show the full potential of facade simplification.

REFERENCES

[Coorg97] Coorg, S. and Teller, S., *Real-Time Occlusion Culling for Models with Large Occluders*, Symposium on Interactive 3D Graphics, pp. 83-90, 1997, ACM SIGGRAPH, ISBN 0-89791-884-3.

[Gotsman99] Gotsman, C. and Sudarsky O. and Fayman, J., *Optimized occlusion culling using five-dimensional subdivision*, Computers & Graphics, 23(5), pp. 645-654, 1999, Pergamon Press. ISSN 0097-8493.

[Greene93] Greene, N. and Kass, M., *Hierarchical Z-Buffer Visibility*, Proceedings of SIGGRAPH 93, Computer Graphics Proceedings, Annual Conference Series, pp. 231-240, 1993, ISBN 0-201-58889-7.

[Law99] Law, F. and Tan, T., *Preprocessing Occlusion for Real-Time Selective Refinement*, Symposium on Interactive 3D Graphics, pp. 47-54, 1999, ACM SIGGRAPH, ISBN 1-58113-082-1.

[Saona99] Saona-Vásquez, C. and Navazo, I. and Brunet, P., *The visibility octree: a data structure for 3D navigation*, Computers & Graphics, 23(5), pp.635-643, 1999, Pergamon Press, ISSN 0097-8493.

[Wonka99] Wonka, P. and Schmalstieg, D., *Occluder Shadows for Fast Walkthroughs of Urban Environments*, Computer Graphics Forum, 18(3), pp. 51-60, 1999, Blackwell Publishers, ISSN 1067-7055.

[Wonka00] Wonka, P. and Wimmer, M. and Schmalstieg, D., *Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs*, Rendering Techniques 2000, pp. 71-82, Eurographics, ISBN 3-211-83535-0.

[Zhang97] Zhang, H. and Manocha, D. and Hudson, T. and Hoff, K.. Visibility Culling Using Hierarchical Occlusion Maps, Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series, pp. 77-88, 1997, Addison Wesley, ISBN 0-89791-896-7.