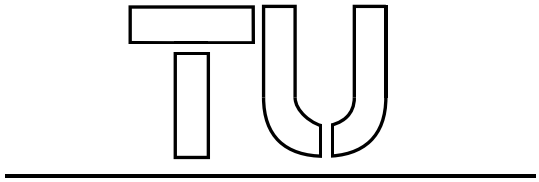


TECHNISCHE UNIVERSITÄT WIEN

Dissertation

Occlusion Culling for Real-Time Rendering of Urban Environments

ausgeführt
zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Michael Gervautz,
Institut 186 für Computergraphik und Algorithmen,
und unter Mitwirkung von
Univ.-Ass. Dipl.-Ing. Dr.techn. Dieter Schmalstieg

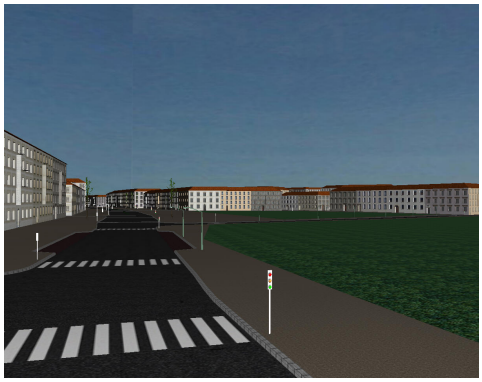
eingereicht
an der Technischen Universität Wien,
Fakultät für Technische Naturwissenschaften und Informatik

von
Dipl.-Ing. Peter Wonka,
Matrikelnummer 9325666,
Hölzlgasse 3,
A-3400 Klosterneuburg, Österreich,
geboren am 10. August 1975 in Wien.

Wien, im Juni 2001.

Peter Wonka (PhD Thesis)

Occlusion Culling for Real-Time Rendering of Urban Environments



http://www.cg.tuwien.ac.at/research/vr/urbanviz/
mailto:wonka@cg.tuwien.ac.at

reviewers:
Dieter Schmalstieg
Michael Gervautz
François X. Sillion

Abstract

This thesis makes a contribution to the field of three-dimensional visualization of urban environments, for applications like urban planning and driving simulation. In a simulation system a user interactively navigates through virtual cities in real-time. To give the impression of fluid motion, high frame rates are necessary, and new images have to be rendered several times each second. To cope with the large amounts of data, acceleration algorithms have to be used to sustain high frame rates. The contributions of this thesis are new algorithms for occlusion culling, that is to quickly identify parts of the scene that are occluded by other objects. These parts are invisible and therefore do not need to be sent to the graphics hardware. This thesis contains three new algorithms:

The first algorithm calculates occlusion culling online for each frame of a walkthrough. The algorithm computes a tight superset of objects that are visible from the current viewpoint. Graphics hardware is exploited to be able to combine a large number of occluders. In contrast to other approaches, an occlusion map is calculated using an *orthographic* projection to obtain smaller algorithm calculation times.

The second algorithm precalculates visibility. The scene is discretized into view cells for which cell-to-object visibility is precomputed, making online overhead negligible. This requires the calculation of occlusion from a region in space. The occlusion boundaries from a region are complex, and analytical computation methods are not very robust and do not scale to the large scenes we envision. We demonstrate how to approximate those complex occlusion boundaries using simple point sampling and occluder shrinking. This approximation is conservative and accurate in finding all significant occlusion and occluder interactions.

The third algorithm calculates occlusion in parallel to the rendering pipeline. We show how to use point visibility algorithms to quickly calculate a tight potentially visible set which is valid for several frames, by shrinking the occluders by an adequate amount. These visibility calculations can be performed on a visibility server, possibly a distinct computer communicating with the display host over a local network. The resulting system essentially combines the advantages of online visibility processing and region-based visibility calculations, in that it is based on a simple online visibility algorithm, while calculating a visibility solution that remains valid for a sufficiently large region of space.

Kurzfassung

Diese Dissertation beschäftigt sich mit der Visualisierung von dreidimensionalen Stadtmodellen für Anwendungen wie Stadtplanung und Fahrsimulation. Für eine Simulation ist es wichtig, daß ein Benutzer interaktiv in der Stadt navigieren kann. Dabei versucht man, wie bei einem Film, den Eindruck einer flüssigen Bewegung zu erzeugen. Um dieses Ziel zu erreichen, ist es notwendig, mehrere Bilder pro Sekunde auf einem graphischen Ausgabegerät, wie einem Monitor, darzustellen. Die großen Datenmengen, die für die Visualisierung eines Stadtmodells verarbeitet werden müssen, machen Beschleunigungsverfahren notwendig, um genügend Bilder pro Sekunde anzeigen zu können. Der Beitrag dieser Dissertation sind neue Verfahren der Sichtbarkeitsberechnung, um schnell diejenigen Teile des Stadtmodells zu finden, die von weiter vorne liegenden Teilen verdeckt werden. Die verdeckten Teile liefern keinen Beitrag zu dem erstellten Bild und müssen deshalb nicht von der Graphikhardware weiterverarbeitet werden. Die erzielten Geschwindigkeitsverbesserungen der Darstellung sind in der Größenordnung von zehn bis hundertfacher Beschleunigung für die verwendeten Testszenen. Es werden drei neue Algorithmen vorgestellt:

Der erste Algorithmus berechnet die Sichtbarkeit für jedes neue Bild in einer Simulation. Der Algorithmus berechnet eine konservative Abschätzung der Objekte, die vom aktuellen Blickpunkt sichtbar sind, ohne dabei sichtbare Objekte als unsichtbar zu klassifizieren. Einige wenige unsichtbare Objekte können allerdings als sichtbar klassifiziert werden. Für die Sichtbarkeitsberechnung wird mit Unterstützung von Graphikhardware ein Bild gezeichnet, das die Sichtbarkeitsinformation codiert. Mit Hilfe dieses Bildes kann man für Szenenteile schnell feststellen, ob sie unsichtbar sind.

Der zweite Algorithmus verwendet Vorberechnungen für die Sichtbarkeit. Die Szene wird in viele Regionen unterteilt, für die jeweils eine Liste von sichtbaren Objekten bestimmt wird. Im Gegensatz zum ersten Algorithmus wird aber die Sichtbarkeit von einer Region und nicht von einem Punkt ausgerechnet. Bekannte analytische Verfahren für dieses Problem sind instabil und zu zeitaufwendig für größere Szenen. Das vorgeschlagene Verfahren zeigt, wie man diese komplexen Berechnungen mit Hilfe von Graphikhardware effizient approximieren kann.

Der dritte Algorithmus ist die Grundlage für ein System, bei dem die Sichtbarkeitsberechnungen paralell zur Darstellung durchgeführt werden können. Diese können somit auf einem anderen Rechner, einem Server, ausgeführt werden, der das Ergebnis der Sichtbarkeitsberechnung dann über das Netzwerk kommuniziert. Dieses paralelle Verfahren erlaubt es, ohne Vorberechnungen schnelle Simulationen zu erstellen.

Acknowledgements

During my work on this thesis, I was employed at three different universities. I would like to thank all the people in Vienna, Rennes and Grenoble who supported my work. I would especially like to thank Michael Wimmer, Dieter Schmalstieg, Michael Gervautz and François X. Sillion for their discussions about my work and their help in getting the results published.

The modeling of the final test scenes was very time consuming and many people contributed to this work. I would like to thank Gerald Hummel, for working on the streets and the terrain, Paul Schroffenegger, for his work on the roofs, Gregor Lehniger and Christian Petzer, who implemented a procedural tree modeler, Marc Pont, for his work on the textures, and Xavier Decoret, for his implementation of an automatic street graph generation program.

Special thanks also to my parents for their support in every way.

This research was supported by the Austrian Science Fund (FWF) contract no. P13867-INF and by the EU Training and Mobility of Researchers network (TMR FMRX-CT96-0036) ”Platform for Animation and Virtual Reality”.

Contents

Chapter 1

Introduction

1.1 Urban visualization

The global scope of this work is urban visualization—modeling and rendering of existing or planned urban environments. In the planning process, it is useful to simulate the environment before changes in the real city are made. In this context, three-dimensional computer simulation gained immense popularity, not only because it produces appealing graphics, but also because it is a more adequate representation for a three-dimensional environment and easier to understand than conventional 2D plans. A common method is to create a three-dimensional model with a commercial modeler and to produce short video clips for presentations.

The real challenge, however, is the visualization in real-time, so that an observer can interactively navigate in the environment with high frame rates. A system like this is not only useful for planning purposes. It can be used as a basis for applications like traffic, driving and architectural simulations, three-dimensional geographic information systems, information visualization and location-based entertainment.

1.2 Urban visualization in real-time

For real-time rendering we need a database of the geometric model of the environment. A user can navigate in this environment using an input device such as a mouse, a keyboard or a car simulator. The user input modifies the location of the viewpoint in the virtual environment, so that new images have to be rendered several times each second.

An important goal of real-time visualization systems is maintaining a sufficiently high frame rate to give the impression of fluid motion. However, for larger environments a simple approach is not able to handle the large amounts of data so that algorithms are important to do the following:

- Calculate simplified versions of the model. Since the screen resolution is only finite, distant objects should be replaced by simpler versions that are faster to render, without losing visual quality.
- Calculate occlusion. From any viewpoint only a portion of the scene is visible. An *occlusion culling* algorithm identifies parts of the scene that are definitely invisible. These parts do not need to be processed by the graphics hardware.
- Prepare and organize the model for real-time rendering. The model has to be organized in data structures to achieve the following two goals: spatial arrangement that allows good occlusion culling, and fast processing on the graphics hardware.

- Handle memory management. Large models do not necessarily fit into the main memory, and it is often impractical to load the whole scene at once. It might be better to load data from the hard disk or from a server on demand.

The contribution of this thesis lies in the area of occlusion culling. For many views in an urban environment one observes that only a few buildings are visible and that big parts of the environment are occluded. To understand why the calculation of occlusion is still a problem requires a quick look at modern hardware architecture.

1.3 Visibility in real-time rendering

In the beginning of computer graphics, visibility was an essential problem. To obtain a correct image it was necessary to guarantee that surfaces in the back do not incorrectly occlude surfaces that are closer to the viewer. On current hardware this problem is resolved with the help of a z-buffer that stores depth information for each pixel of the image. With the use of the z-buffer, visibility can be calculated in the last stage of the real-time rendering pipeline that is described as a three-step process:

1. *Traversal:* The database containing the geometrical model of the environment has to be traversed. The geometry that is selected to be rendered is sent to the graphics hardware (A common method is to select only objects that intersect the viewing frustum).
2. *Transformation:* The geometry is transformed from the world coordinate system to the screen coordinate system.
3. *Rasterization:* After transformation the geometry is rasterized and written into the frame buffer. For each covered pixel in the frame buffer, a fragment is generated. Each fragment has a z-value. This z-value is compared to the z-value in the z-buffer to test the fragment's visibility. Apart from a few discretization errors, this visibility test always creates the desired results.

The visibility test is done very late in the pipeline and is done once for each generated fragment. Although the correctness of the visibility calculation is sufficient, it is very inefficient for large scenes. If we consider an urban walkthrough, one building might occlude hundreds of others. All those occluded buildings would have to undergo the three steps of the rendering pipeline unnecessarily which is too time consuming. In this thesis algorithms will be described that can quickly identify occluded scene parts in the *traversal*-stage of the pipeline. The occluded parts will no longer have to undergo the other two steps of the pipeline, which results in large time savings. As will be shown in the results in chapter 5, even with occlusion culling alone, speed-ups of a factor of one-hundred are realistic for a medium-sized city model. Furthermore, it is important to calculate visibility before the problem of model simplification or data prefetching can be handled efficiently. The next section briefly describes our main contributions to solve this occlusion culling problem.

1.4 Main contributions

The results of this thesis have been partially published by the author [Wonk99, Wonk00, Wonk01]. The main contributions are:

- Development of an online visibility algorithm for urban environments. Previous approaches rendered occluders into an occlusion map using perspective projection. We propose the use of *orthographic* projection for the creation of occlusion maps:

  Peter Wonka and Dieter Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum (Proc. Eurographics '99)*, 18(3):51–60, September 1999.

- Development of a visibility preprocessing algorithm. For preprocessing, visibility has to be calculated for a region in space rather than a point in space. This calculation is inherently complex. We propose a method to approximate complex region visibility using simple point sampling:

  Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. *Rendering Techniques 2000 (Proceedings of the Eurographics Workshop 2000)*, Eurographics, pages 71–82. Springer-Verlag Wien New York, June 2000.

- Construction of a hybrid system that calculates visibility in parallel to the rendering pipeline. This system combines several advantages of point and region visibility:

  Peter Wonka, Michael Wimmer, and François Sillion. Instant visibility. *Computer Graphics Forum (Proc. Eurographics 2001)*, 20(3), September 2001.

Furthermore, the author participated in the following project closely related to this thesis [Wimm01]:

- Development of *Point-based Impostors*. This representation can be used to simplify distant scene parts to avoid aliasing artifacts and to accelerate rendering:

  Michael Wimmer, Peter Wonka, and François Sillion. Point-Based Impostors for Real-Time Visualization. In Karol Myszkowski and Steven J. Gortler, editors, *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 2001)*. Eurographics. Springer-Verlag Wien New York, June 2001.

1.5 Structure of the thesis

The thesis is organized as follows: Chapter 2 describes important basics of our work that are necessary to understand the discussion of the related work and the proposed algorithms. This includes a definition of terms, a motivation from human factors, and an informal study of visibility in urban environments. Chapter 3 reviews related work and discusses it in the light of its suitability for visibility calculation in urban environments. Chapter 4 introduces an algorithm to calculate visibility from a point and shows how this algorithm was used to build an online occlusion culling system. Chapter 5 describes an algorithm to calculate visibility from a region of space and is therefore useful for preprocessing. Chapter 6 shows how to build an online occlusion culling system that is a hybrid between point and region visibility. Finally, Chapter 8 presents the conclusions of this thesis.

Chapter 2

Basics of Real-time Rendering and Visibility

This section describes important basics for the discussion of the related work and the research conducted for this thesis. The first section explains frequently used terms. The second section describes human factors that became a driving motivation for our work and are essential for all real-time rendering systems. The third section presents an informal study of visibility in urban environments. This study is not geared towards a deeper understanding of the technical aspects of visibility problems but gives an intuitive impression of the nature of visibility in urban environments.

2.1 Description of terms

Real-time rendering: In this thesis, the word is used in the global sense, describing rendering at interactive frame rates. We can identify two types of real-time rendering: the term *hard real-time* is used for a system that gives a guarantee for a constant frame rate, while *soft real-time* describes a system without such a guarantee. Often this distinction is not necessary and therefore we use the general term *real-time* to refer to both types of real-time rendering. A good introduction to the topic of real-time rendering can be found in [Möll99].

Scene-graph: A scene-graph is a graph that is used to describe a computer graphics model (Figure 2.1). This graph consists of nodes and edges. Each interior node of the scene-graph is the root-node of a sub-graph. The actual geometry of the scene is located in the leaf-nodes and consists mainly of triangles. Triangles are typically defined through their vertices, normal vectors at these vertices and texture coordinates. The inner-nodes are used to define transformations and to group several sub-graphs.

Object: An object is an object in a city like a building, a car or a tree. In a computer graphics model, such an object is represented by a scene-graph. The terms *scene-graph* and *object* are used on different levels: an object can be broken down into several smaller objects, e.g. a building can be broken down into a roof, windows, doors, walls ... that are themselves represented by a scene-graph (sub-graphs of the scene-graph describing the whole object). Similarly, all objects in the scene are organized in one scene-graph, describing the whole city model.

Occluder: An occluder is either a single polygon (*occluder polygon*) or a volume described by a subset of $\mathbf{R}^3$ (*volumetric occluder*). Note that an occluder is used to calculate occlusion of parts of the scene-graph, while the occluder itself does not need to be a part of the scene-graph.

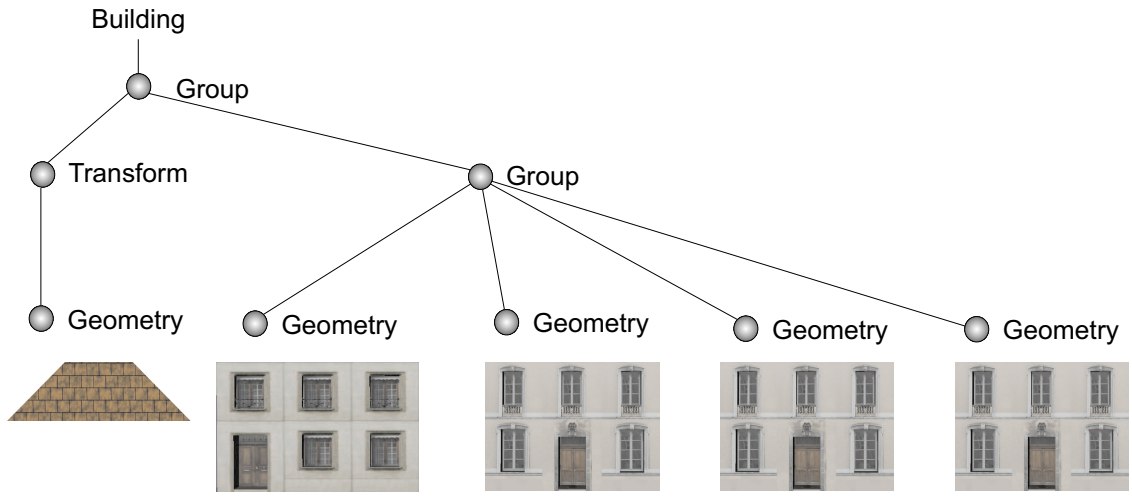


Figure 2.1: This figure shows a simple scene-graph of a building consisting of a roof and four facades. The scene-graph has different group-nodes, geometry-nodes and one transformation-node.

Bounding Volume: A bounding volume is a volume (subset of $\mathbf{R}^3$) that encloses the geometry described by a scene-graph or a sub-graph of the scene-graph (Usually such a bounding volume is stored with a node in the scene-graph). A *bounding box* is a bounding volume defined by a box.

Occludee: An occludee is a sub-graph of the scene-graph that is tested for occlusion.

View Cell: A view cell is a region in space where the viewpoint can be located. In urban environments, a view cell is typically an extruded polygon.

PVS: The PVS is a set of objects that are potentially visible from a viewpoint or a view cell. PVS is the abbreviation for "potentially visible set".

2.5D: We call an occluder 2.5 dimensional if it can be described by a function $z = f(x, y)$. Basically this function defines a height field.

2.2 Human factors and technological constraints

In walkthrough applications for urban environments, a user navigates through a city as a pedestrian or vehicle driver. To give the impression of fluid motion, several images per second need to be displayed. The rendering speed is measured in frames per second (fps) or Hertz (Hz). The general strategy of displaying images is to use *double buffering*: The new image is drawn in the back buffer (off-screen) and swapped into the front buffer (visible on screen) when it is finished.

To set useful goals in real-time rendering, it is important to understand the human factors that determine the perception of interactive graphics and their relation to technological constraints. We are especially interested in answering the following question: *What frame rate should be used in real-time rendering?*

The maximum frame rate is determined by the hardware setup. For video games running on a TV set, the refresh frequency is limited to 59.94 Hz (NTSC) or 50 Hz (PAL). Higher frame rates cannot be displayed on the screen. Similarly, on PCs the frame rate is limited by the monitor refresh rate, which is typically between 60 and 120 Hz on current hardware. The frequency at which modulation is no longer perceptible varies with the monitor, depending on attributes such as field of view, resolution and brightness (Padmos [Helm94] reports that large, bright displays need refresh rates above 85 Hz). For field interleaved stereo images the frame rate is at least 120 Hz, as one image is needed for each eye. A real-time rendering

application should guarantee refresh rates equal to the refresh rate of the output device to avoid ghosting artifacts, where objects appear to be split into multiple copies along the direction of motion [Helm94].

A *constant frame rate* is very important and is required for most visual simulators. A varying frame rate distracts the user, complicates the accurate perception of velocities, and causes temporal inaccuracies resulting in jerky motion, because several frames would not appear at the time for which they are scheduled [Helm94].

These high guaranteed frame rates, however, are out of reach for many applications. Several authors (e.g. [Sowi94, Mano00, Möll99, Helm94]) usually recommend frame rates between 10 and 30 fps as a reasonable goal for real-time rendering. The human eye sees guaranteed 20 fps as fluid motion—this is probably the second best reasonable goal for real-time rendering.

To sum up, real-time rendering should be done with a guaranteed 60 Hz or more. The next reasonable goal is sustained 20 Hz. The last resort is to render each frame as fast as possible. This usually results in greatly varying frame rates and is not beneficial for many applications.

2.3 Visibility analysis of urban environments

In this section, we present an analysis of visibility in urban environments to sketch the problem. This analysis results in a few key observations that motivate the algorithms described in this thesis and help to evaluate the related work. We are mainly interested in walkthrough applications, where the viewpoint is located near the ground. In these scenarios, occlusion culling is very effective. There is also a significant amount of occlusion in flyovers, but we will not analyze these configurations. The analysis is done using real photographs rather than computer graphics models. We have to consider that current models are greatly simplified compared to the complexity of the real world, though we want to end up with models that closely resemble reality. We will state different observations and illustrate each observation with photographic examples.

Buildings are the main source of occlusion in urban environments. In several parts of a city a few close buildings occlude the rest of the model (Figure 2.2). The visible portion of the model is small compared to the whole city.

Figure 2.2: Two images from the city of Vienna. Note that buildings are the main source of occlusion.

It is also important to note that visibility in urban environments is, in practice, 2.5 dimensional. Buildings are connected to the ground and it is generally impossible to see through them. Therefore, occlusion is defined by the roof edges of buildings. If we consider an image from a certain viewpoint, we can observe that all objects behind a building that cannot be seen over the roof are occluded (Figure 2.3). Of course there are several cases where this assumption does not hold. Objects like bridges or buildings with large pathways through them would require more complicated treatment.

Figure 2.3: Occlusion in urban environments is essentially 2.5 dimensional. The occlusion of a building can be described by its roof edges, shown in red.

In general, the attributes of visibility will strongly vary in different cities and different areas within a city. The centers of European cities have developed over centuries and have complex layout plans, whereas many American cities contain long, straight streets. The optimal cases for occlusion culling are dense environments with high buildings that are connected to building blocks and stand on flat ground. Streets should be short and change direction frequently to prevent long open views. This is a typical setup for the center of many European cities. Due to the flat terrain and the high buildings, the visibility problem is almost reduced to two dimensions. Since the buildings are connected to building blocks one cannot see between them. Although these scenes occur quite frequently, it is important to be aware that we cannot rely on the fact that only a few objects close to the viewpoint are visible. First of all, even in the center of such occlusion-friendly cities, not all parts of the environment conform to these ideal circumstances. Furthermore, it is important to consider all types of urban scenes for the design of an occlusion culling algorithm. We will now discuss the main factors that determine the complexity of visibility.

First, it is important to note that cities contain many viewpoints where a viewer can see quite far. Many cities, such as London, Vienna and Paris, were built near a river. Rivers usually run through a city near the center so that even in the inner city an observer can see for a distance of several hundred meters (Figure 2.4). Other possibilities for long views include long, straight streets, large places and train stations.

Figure 2.4: These photographs were taken near the center of Vienna. Note that the view in the right image is several kilometers.

In the center of large cities the building structure is very dense, to efficiently use the existing space. In areas further away from the center, in suburbs or in smaller cities, people have more space so that buildings

become smaller and are not organized in building blocks to make room for private gardens or parks. One implication is that the height structure becomes more important due to smaller buildings (Figure 2.5), so that it occurs more frequently that buildings are visible over the roof edges of closer buildings. If buildings are smaller and not connected, more objects are visible in general (Figure 2.6).

Figure 2.5: The image on the left shows a view in the inner city of Vienna with high buildings. On the right side, the view has a much more complicated height structure due to smaller buildings.

Figure 2.6: Two images from a small city. Note that buildings are not always connected and are smaller than buildings in the center of a larg city.

An important factor for visibility is the terrain. If a city is built on flat terrain, the height structure is less important, since only occasionally can one building be seen behind another. However, in the presence of greater height differences, many objects can become visible (Figure 2.7). In the near field, buildings still occlude many objects. In the far field, however, we made the observation that in these cases the visibility is mainly determined by the terrain itself, and buildings no longer contribute much to the occlusion. A huge number of objects may be visible in these types of scenes. These views are the transition from urban walkthroughs to terrain rendering or urban flyovers.

The assumption of the 2.5D nature of visibility in urban environments breaks down when visibility is determined by vegetation (Figure 2.8). These kinds of scenes cannot be handled by the algorithms proposed in this thesis. One reason why we did not consider vegetation in our work is that the modeling and real-time rendering of these scenes is an open problem. Trees are highly complex entities, and when we consider a continuous image of many trees, this image will contain very high frequencies. One pixel can contain several leaves and branches, and the contribution to a single pixel can come from multiple trees placed several hundred meters apart. Rendering of such a scene calculates a sampled version of the continuous

Figure 2.7: These images show the effect of greater height differences in the terrain. Note the large number of visible objects that can be seen from one viewpoint.

image in the frame buffer. Prefiltering is necessary, because otherwise we would see strong aliasing artifacts due to the high frequencies. Usually trees are modeled with several alpha-textured quadrilaterals. Such a texture corresponds to a prefiltered version of one tree from one viewpoint. Although reasonable results can be achieved when only a few dense trees are placed in the model, the strategy using billboards is only a very crude approximation whose limitations become apparent if scenes like those in figure 2.8 are modeled.

Figure 2.8: Vegetation is a real challenge for computer graphics algorithms. The complexity of these scenes is so high that it can be hardly captured by the human eye. Note that even these images fall short of reproducing this complexity.

Another interesting observation is that visibility in computer graphics models differs quite strongly from visibility in the real-world. Usually the views in computer graphics models are more open, due to missing objects like trees, bushes, cars or details in the terrain (Figure 2.9).

Main Observations

Based on the previous observations we want to point out the following conclusions:

- Visibility in urban environments is basically 2.5 dimensional.
- An observer can only see a few close buildings in many parts of an inner-city walkthrough. However, there are many different types of views with a wide range of complexity. One can never count on the fact that only a few objects are visible.

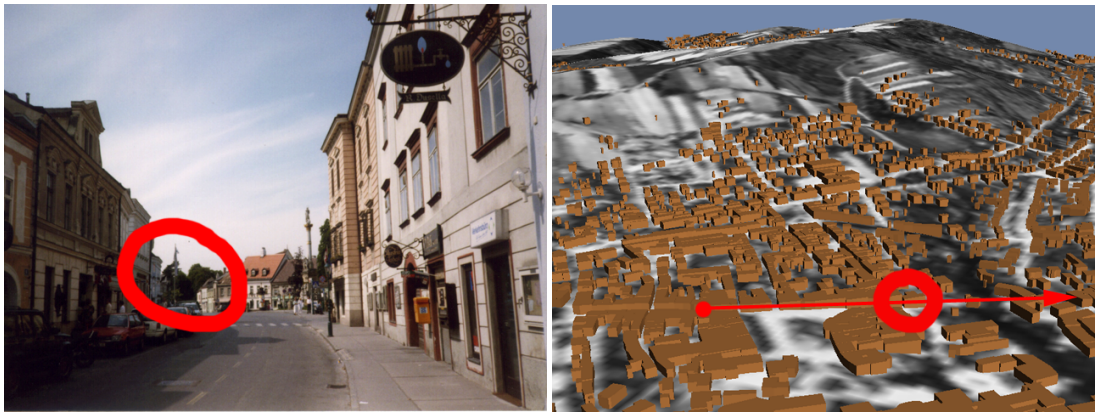


Figure 2.9: These images demonstrate the difference between the real-world and a computer graphics model. In the left image the observer can see about 200 meters. The corresponding view in the computer graphics model is much farther because of the missing trees. (The viewpoint, view direction and the location of the trees is shown in red).

- If a city contains great height differences in the terrain, many objects will be visible from most viewpoints.
- In the same view an observer can see a close object next to a distant one.
- Visibility in real cities differs from visibility in current computer-graphics models.

Chapter 3

Related Work

The structure of this chapter is as follows: First, we will review different approaches to urban modeling to understand how a model looks and what data is typically available. Then we will address different acceleration methods that can be used in addition to or instead of occlusion culling. In the last section we will discuss the related work dealing with visibility that is relevant for this thesis.

3.1 Urban modeling

To create a smaller model consisting of a few buildings, commercial software like AutoCAD [Wils99], ArchiCAD [Grap01] or Multigen [Inc.01] can be used.

A common approach to urban modeling is to replace geometric detail with texture maps obtained from videos or photographs. Only basic geometry including building footprints, building heights, streets and property borders is used for the geometric model. If this information is not already available from other projects, it can be obtained by digitizing aerial photographs [Jeps96, Jeps95]. Building heights are either measured, calculated by the number of floors in a building, or estimated from photographs [Hild95].

As a second step, photographs or videos are taken as a basis for texture mapping and geometry reconstruction. For a simple model, these images are mapped on extruded building footprints. For higher model quality or complex buildings, the images can also be used to reconstruct additional geometric detail (see for example [Debe96, Debe98]).

Although such models look nice and realistic, the acquisition and post-processing of images is a time-consuming task. The images have to be color corrected, perspective corrected or warped, and undesirable elements such as telephone poles and cars have to be removed (often done with Adobe Photoshop [Bout00]). Reported times for post-processing range from several minutes to two hours for more covered images [Jeps96].

This approach is used with minor variations in many cities including Los Angeles [Jeps96], Tübingen [Veen97], Frankfurt [Hild95] and Philadelphia [Maho97].

Procedural and parametric techniques are a good method for database amplification [Smit84] and provide scalable means for large-scale modeling. This is a common technique for the design of natural phenomena like vegetation [Prus91, Gerv96, Webe95, Měch96, Deus98]. For urban environments, the solutions proposed for modeling are rather simple compared to the successful work in plant modeling. The main idea is to automatically extrude building footprints and use random textures for the facades. Prominent buildings can be separately modeled with higher accuracy to allow recognition and orientation in the city. This approach is used for the city of Rennes using the modeler VUEMS [Doni97b, Doni97a, Thom00] and software tools from the company IWI [IVT01]. Another procedurally generated model is part of the

computer game Midtown Madness [Micr01] that includes the cities of Chicago and San Francisco. The important landmarks, like the Loop, the Field Museum and the Navy Pier are modeled in sufficient exactness so that the recognition effect is high.

In the future automatic acquisition of urban data using one or more cameras, videos or aerial photographs, will become more important. Examples include the recording of building facades using a CCD camera [Mare97] and the City Scanning Project at MIT [Tell01] which aims at the fully automated reconstruction of CAD data for a city. However, these reconstruction methods are not yet advanced enough to create large scale models.

3.2 Acceleration of real-time rendering

3.2.1 The real-time rendering pipeline

To render a model, it is first loaded into main memory. During runtime the scene-graph is traversed top down and the geometry is sent to the graphics hardware. This process is organized in a pipeline called the *Rendering Pipeline*. The pipeline is organized in three stages, the *Traversal*, the *Transform* and the *Rasterization* stage and was already briefly described in section 1.3. To optimize rendering, it is important to understand this pipeline to identify useful optimization strategies. We want to point out two observations:

- First, the geometry and pixel processing in the *Transform* and *Rasterization* stage is usually done by the graphics hardware. Results of the calculations cannot be accessed easily. Reading values from the frame buffer, for example, involves very high overhead on current hardware. Optimizations in these pipeline stages are usually reduced to efficient programming of the rendering hardware.
- Second, there is usually one stage in the pipeline that constitutes a bottleneck. Optimization should concentrate on this stage. However, this bottleneck can change within a frame and can strongly depend on the hardware and the display drivers. Therefore, it is often useful to make all stages as fast as possible.

In the following we will discuss algorithms for fast real-time rendering. We will review standard optimizations that are important when programming the graphics hardware, geometric and image-based simplifications, and system issues. Finally, we will address related work in visibility.

3.2.2 Programming real-time rendering hardware

The rendering hardware is accessible through an application programming interface (API). Lower level programming is possible but not recommended on current hardware. The two most popular APIs are OpenGL [Woo99] and Direct3D [Kova00]. The straightforward rendering algorithm traverses the scene-graph top down. When a geometry node is encountered the geometry is passed via an API call to the graphics hardware. For an efficient scene-graph layout we have to consider the following three properties of rendering hardware:

- State changes are relatively expensive. Switching different vertex shaders, textures or materials is a time consuming operation. Therefore, rendering packages like Performer [Rohl94, Ecke00], OpenGL Optimizer [Ecke98] and Open Inventor [Wern94] have auxiliary procedures that rearrange the geometry to minimize state changes during rendering. At the same time the scene-graph can be flattened to reduce unnecessary nodes in the scene-graph hierarchy.
- The result of a vertex transformation can be reused. If a surface is represented by a triangle mesh, two neighboring triangles usually share two vertices. This can be exploited using triangle strips which is a popular data format used to pass data to the graphics hardware. Vertices of neighboring

triangles are sent only once to the graphics hardware and are transformed only once. Triangle strips are created in a preprocess [Evan96, Xian99]. Newer graphics hardware includes a larger vertex cache for previously transformed vertices. A preprocessing algorithm that exploits the vertex cache was proposed by Hoppe [Hopp99].

- The transfer of data to the hardware is a critical operation in some architectures. On modern consumer hardware it is necessary to avoid the extensive use of API calls. Therefore, it is necessary to send many triangles to the graphics hardware with a single API call. However, it is not possible to send an arbitrary group of triangles with one API call, because the triangles have to share the same texture, material ....

Backface culling is an optimization typically implemented in the graphics hardware. Back-facing polygons can be quickly rejected based on the vertex order. This test mainly helps in the presence of large polygons where the pixel fill rate is the bottleneck.

Other optimizations include the use of precompiled rendering commands (e.g. OpenGL display lists), optimized use of the cache hierarchy, multi processing, and optimized texture layout [Möll99, Rohl94].

In general the programming of graphics hardware is surprisingly frustrating and often gives unexpected rendering times. The optimal way to program the graphics hardware changes quite frequently and depends on many different factors that cannot be easily identified. Therefore, it is not possible to decide in advance which API calls or data format to pass data to the graphics hardware will be the fastest. The only promising strategy is to implement many different API calls and optimization strategies and time the results to see what works best.

3.2.3 Levels of detail

For certain objects, the full geometric information in a given frame cannot be perceived, e.g. because the object is far away from the viewpoint. To better use the effort put into rendering such features, an object should be represented at multiple levels of detail (LODs). The main problems that need to be addressed are the calculation of different levels of details, the selection of different levels of details during runtime [Funk93, Ecke00], and the switching between different levels of details.

Different LODs of one object can be created manually or automatically. Several algorithms exist for the automatic creation of LODs. They make use of vertex clustering [Ross93], edge collapses [Hopp96], octree based simplification [Schm97a], and quadric error metrics [Garl97], to name just a few. Newer algorithms also take into account texture and color information, which is crucial for most computer graphics models (e.g. [Garl98, Cohe98a]).

Continuous (or smooth, or progressive) LOD representations (e.g. [Hopp96, Schm97b]) permit the extraction of a model with an arbitrary triangle count at runtime, whereas discrete representations provide a small number of pre-created LODs with fixed triangle counts. Some continuous algorithms allow the level of detail to vary over the model depending on viewing parameters [Hopp97, Xia96, Lueb97].

Yet, for urban simulation projects [Jeps95], hard switching between a few pre-created LODs is very popular because it produces almost no overhead during runtime. In general it is also difficult to simplify trees and buildings because of their special geometric structure (Buildings have a very regular structure and the leaves of trees consist of many disconnected triangles).

3.2.4 Image-based rendering

The idea of image-based rendering is to synthesize new views based on given images. Ideally, we would like to replace distant geometry with one image, an alpha texture map, because it is very fast to render [Maci95]. Schaufler et al. [Scha96] and Shade et al. [Shad96] used this idea to build a hierarchical image cache for an online system with relaxed constraints for image quality. However, precalculating

many alpha texture maps for several viewpoints in the view cell and blending between them depending on the viewing position requires too much memory if high image quality is desired. A more efficient solution is a 4D parameterization of the plenoptic function, the light field [Levo96], which can be seen as a collection of images taken from a regular grid on a single plane. At runtime it is possible to synthesize images for new viewpoints not only on this plane, but also for all viewpoints within a certain view cell behind the plane. Gortler et al. [Gort96] independently developed a similar parameterization. They additionally use depth information for better reconstruction. Depth information can also be added as a triangle mesh to create surface light fields [Wood00, Mill98]. Chai et al. [Chai00] studied the relationship between depth and spectral support of the light field in more detail and added depth information to the light field in layers. Although a light field can be rendered interactively, memory consumptions and calculation times make it hard to use for real-time rendering applications.

Starting from the texture map idea, depth information can be added to make an image usable for a larger number of viewpoints, for example by using layers [Scha98, Meye98]. These layers can be rendered quickly on existing hardware but they contain a strong directional bias which can lead to image artifacts, especially in complex scenes. Several authors added depth information to images using triangles [Sill97, Deco99, Dars97, Mark97]. While a (layered) depth mesh can be calculated and simplified for one viewpoint, the representation is undersampled for other viewpoints, leading to disocclusion artifacts or blurring effects. The calculation of an equally sampled high quality triangle mesh remains an open problem. Finally, depth can be added per point sample. In particular, layered depth images (LDIs) [Shad98, Max96] provide greater flexibility by allowing several depth values per image sample. However, warping the information seen from a view cell into the image of a single viewpoint again leads to a sampling bias. To overcome this problem, several LDIs have to be used [Lisc98, Chan99].

As an alternative, point-based rendering algorithms were introduced by Levoy et al. [Levo85], Grossman et al. [Gros98] and Pfister et al. [Pfis00]. These algorithms are currently not implemented in hardware and hole filling in particular is a challenging problem. For faster rendering, warping can be replaced by hardware transformation together with a splatting operation [Rusi00]. In the context of complex geometry, points can no longer be seen as point samples of a surface [Pfis00, Rusi00] and a more sophisticated treatment of filtering is necessary to construct a high quality representation.

Although image-based rendering is a promising approach to replace distant geometry in urban walkthroughs, all of the current solutions have their limitations. The creation of high quality representations is still an open research problem.

3.2.5 Database management and system integration

The integration of several rendering techniques for an interactive walkthrough system was done by the research group at Berkeley [Funk96] for architectural scenes. The system uses a cell decomposition for the calculation of potentially visible sets, precalculated levels of detail and asynchronous data prefetching for memory management.

The problem of management and visualization of large GIS databases has been investigated by Kofler et al. [Kofl98b, Kofl98a]. They propose the use of R-LOD-trees for fast retrieval and visualization of 3D GIS data. Furthermore, they observed that conventional database systems provide inadequate performance for the demands of 3D GIS applications and analyzed the applicability of object-oriented database systems.

Researchers at Georgia Tech [Lind97] are working on a GIS system for visual simulation. The main problems to be tackled are LOD management for height fields and texturing of large terrains.

At the University of North Carolina [Alia99a], several acceleration techniques are being combined to enable an interactive walkthrough in a massive CAD model - a powerplant with about 15 million polygons. Besides LODs and database management, the system uses occlusion culling and textured depth meshes, which are created in a preprocessing step. The placement of LDIs to obtain a guaranteed frame rate was investigated by Aliaga et al. [Alia99b]. However, these methods are only plausible for high performance workstations and take huge amounts of memory and preprocessing time. The involved assumptions do not hold for consumer hardware.

3.3 Visibility

There is a large amount of visibility literature. A very recent survey of visibility was written by Durand [Dura99], which also incorporated visibility in related fields, like robotics and computer vision. In this section we will give an overview of the work that is especially important for real-time rendering.

3.3.1 General idea

Calculating visibility for three-dimensional scenes is intrinsically complicated. Concepts like the aspect graph [Egge92], the visibility complex [Pocc93, Dura96] or its simpler version, the visibility skeleton [Dura97]), are important to the study of three-dimensional visibility and can be applied to scenes with a few polygons. Unfortunately these concepts cannot be easily used for applications we envision. The scalability issues and numerical robustness problems involved make room for a large amount of specialized algorithms. Additionally, for real-time rendering systems the time constraints are an important factor, as such a system cannot allot much time for visibility calculations. To cope with the complexity of visibility, most algorithms use rather strong assumptions and simplifications that meet the demands of only a few applications. Therefore, a careful choice of assumptions and simplifications is a crucial part in the design of a visibility algorithm.

The idea of a modern visibility culling algorithm is to calculate a fast estimation of those parts of the scene that are definitely invisible. Final hidden surface removal is done with the support of hardware, usually a z-buffer. The visibility estimation should be conservative, i.e., the algorithm never classifies a visible object as invisible. However, an invisible object can still be classified as visible because the z-buffer would detect it as invisible in the last stage of the *Rendering Pipeline*. For large models it is also common to calculate visibility on a per-object level rather than on a polygon level, because

- this saves a considerable amount of time when performing the occlusion tests, which is important for online calculations.
- the memory consumptions would be a problem for offline calculations.
- triangles are usually sent to the graphics hardware in triangle strips (see section 3.2.2). A per-polygon visibility classification would involve a costly rearrangement of data-structures at runtime.

3.3.2 View-frustum and backface culling

View-frustum culling [Clar76] is a simple and general culling method, which is applicable to almost any model. Each node in the scene-graph has a bounding volume, such as an axis aligned bounding box or a bounding sphere. When the scene-graph is traversed, the bounding volume of these nodes is compared against the viewing frustum. If the bounding volume is completely outside the viewing frustum, the node and its children are invisible. This method is essential and is implemented in almost all real-time rendering systems.

Another method is backface culling. Usually surfaces are oriented and have only one visible side. Back-facing surfaces can therefore be discarded. This test is usually done in hardware. The first version calculates the normal vector of the projected polygon in screen space based on the orientation of the polygon. The second version calculates the angle between the polygon normal and a vector from the viewpoint to one of the vertices. If the angle is larger than $\pi/2$, the polygon can be discarded. Using the first kind of backface culling makes it necessary to use the same vertex order for all polygons of an object, or better for the whole scene. This demands careful modeling.

Backface culling for a single polygon can be extended to calculate a group of back-facing polygons. Polygons with similar normals are grouped to clusters. At runtime a whole group can be determined to be back-facing and discarded [Kuma96b, Joha98]. A variation of this algorithm for spline surfaces

was also investigated [Kuma96a]. These methods work well for smooth surfaces, but in the context of urban environments we have to consider that many polygons are orthogonal which results in large normal variations. It is also important to note that back-facing polygons are eliminated because they must be occluded by other polygons. Therefore, the effect of clustered backface culling will be reduced when it is used in combination with occlusion culling.

3.3.3 Point visibility

Point based visibility is the visibility calculated from a point in space. The calculations are used to identify visible objects for every new frame in a walkthrough. We can identify geometric, image-based and hardware-based point visibility algorithms.

Geometric point visibility

Several algorithms were proposed for occlusion culling in object space using a few large convex occluders. Similar algorithms were proposed by Coorg and Teller [Coor97, Coor99], Bittner et al. [Bitt98] and Hudson et al. [Huds97]. The main idea in these algorithms is to select a small set of occluders likely to occlude a large part of the model. The occluders define a shadow frustum against which objects in the scene can be tested. At startup the scene-graph is organized in a hierarchical data structure like a k-d tree or bounding-box tree. For each frame of the walkthrough several occluders are selected and used to build a shadow data structure. Then the scene-graph is traversed hierarchically and the bounding volumes of the scene-graph are tested for occlusion. For the occluder selection a heuristic is used to calculate the occlusion potential of an occluder polygon, based on attributes like the distance to the viewpoint, the area of the occluder and the angle between polygon normal and the viewing direction. Bittner et al. [Bitt98] use a variation of the shadow volume BSP tree to merge occluder shadows. This algorithm calculates better fusion of occluders than the other algorithms.

These algorithms work in full 3D and demonstrate their results on urban walkthroughs. However, these algorithms have certain properties that are not desirable for urban walkthroughs:

- Only a small number of occluders is considered.
- The occluders are selected according to a heuristic. For scenes of medium depth complexity a heuristic can achieve good results. For larger scenes, much better occlusion calculation is necessary because through every small gap between two occluders hundreds of buildings can and probably will be classified as visible. Good occluder selection should be based on visibility.
- The calculation of full three-dimensional occlusion is an overhead for urban scenes. If only a few occluders are considered all occluders will be buildings. For buildings 2.5D calculations should be enough.

An algorithm that improves these shortcomings was proposed only recently by Downs et al. [Down01]. Like the algorithm proposed in this thesis, they exploit the fact that urban environments can be seen as height fields and calculate occlusion in 2.5D. Occlusion information is stored as an occlusion horizon. This horizon is a conservative approximation of a cross section through the shadow defined through a plane P. This plane P is swept front-to-back. During the sweep, scene-graph nodes can be tested hierarchically against the occlusion horizon. New occluders are either found to be invisible or are inserted into the occlusion horizon. The horizon is organized in a hierarchical tree and contains piecewise constant functions (Figure 3.1). The big advantage of this algorithm is that, in contrast to many other approaches, the occluder selection is based on visibility. Therefore, the algorithm does not degenerate as easily as other methods (e.g. [Coor97, Bitt98, Huds97, Zhan97].

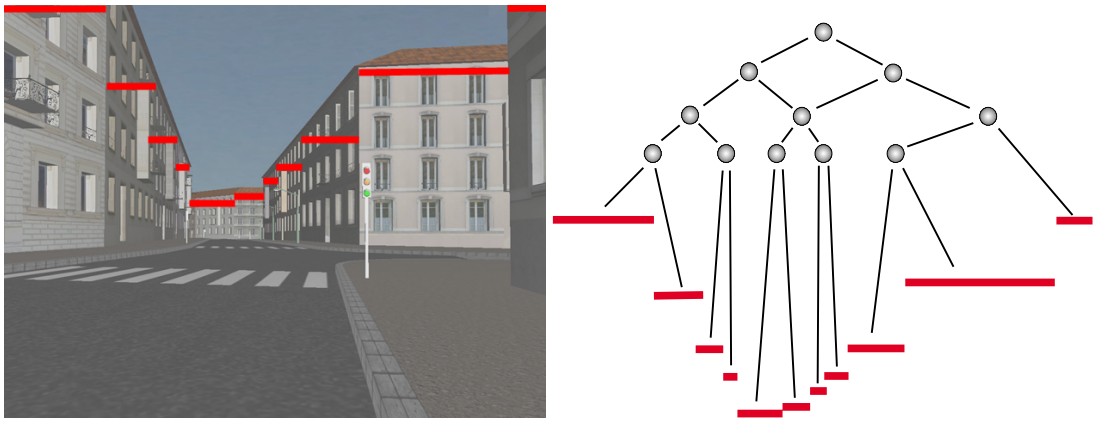


Figure 3.1: These figures demonstrate the idea of the occlusion horizon. (Left) The occlusion horizon is shown in red. (Right) The figure shows the occlusion horizon stored in a binary tree. Each interior node stores the minimum and maximum height of its children.

Image-based point visibility

General algorithms for occlusion culling were proposed that calculate occlusion in image space. The basic idea is to create or maintain an occlusion map with the same or lower resolution as the generated image. The bounding volumes of the scene-graph nodes can be tested against the occlusion map to determine if they are occluded.

Greene proposed the hierarchical z-buffer [Gree93, Gree96] as an extension to the regular z-buffer. The scene is organized in an octree and the z-buffer is organized in a z-pyramid (similar to a texture pyramid for mip-mapping). The lowest level of the pyramid corresponds to a regular z-buffer. The higher levels are always half the resolution of the level below.

The spatial arrangement of the scene cannot be easily done dynamically and should be done in advance. During runtime a bounding box of a scene-graph node can be tested for occlusion by rendering the bounding box. The classification is done based on the z-values using the z-pyramid for accelerated comparison. If all generated fragments that are created by the bounding box are behind the fragments already in the z-buffer, the bounding box is occluded.

Greene proposes to organize the scene in an octree, but the system would also work for other spatial data-structures. If a bounding box is invisible, it can be skipped. If it is visible, then either the bounding boxes of the children can be tested or the geometry in the bounding box will be rendered. The scene-graph should be traversed in front-to-back order because scene-graph nodes are only occluded by objects closer to the viewpoint. When the geometry is rendered the z-buffer is updated, which usually involves an update of all levels of the pyramid.

This algorithm builds on two major concepts:

1. A z-buffer pyramid to accelerate z-tests of large screen space areas. This is especially useful to test bounding box nodes because they normally cover large screen space areas.

2. A way to communicate the results of the z-test back to the application.

These concepts are only partially implemented in hardware (see next section), so that a full implementation only runs in software. This is usually too slow for real-time rendering. Several variations were proposed to obtain a working solution on current hardware. In his PhD thesis Zhang studies several versions and aspects of the problem of occlusion culling for general scenes using a variation of the hierarchical z-buffer [Zhan97, Zhan98]. This occlusion culling algorithm was implemented and tested on existing graphics hardware.

In general the major obstacle for hardware-accelerated occlusion culling is the fact that the application cannot access the frame buffer directly but needs to copy the data to main memory first. A progressive occlusion culling algorithm that updates the occlusion hierarchy each time an object is rendered is not feasible on current hardware. Zhang et al. start with a multi-pass occlusion culling algorithm that updates the occlusion representation fewer times. The algorithm consists of several iterations of the following three steps:

- render selected occluders into an occlusion map (typically an off-screen rendering area).
- update the occlusion representation. Usually this includes reading back the frame buffer into the main memory if hardware support is involved.
- traverse scene-graph nodes and add visible occluders to the selected occluder set.

Reading back the frame buffer is very expensive, even on high end machines. To obtain reasonable calculation times Zhang mainly works with a one pass solution. Potentially useful occluders are selected based on a heuristic and the occlusion representation is only built once.

A big difference to the hierarchical z-buffer is that such an algorithm can decouple the occlusion map from the actual rendered image. If the occlusion map has a lower resolution this results in occlusion culling which is faster but not strictly conservative. Furthermore it is easier to use occluders that are not polygons of the original scene. The disadvantage is that rendering occluders is an overhead. Therefore Zhang et al. select only large polygons for occlusion culling based on a heuristic.

In contrast to the hierarchical z-buffer they use one occlusion map using only 1-bit occlusion information and a separate buffer to estimate the depth values. For each frame they select occluders near the viewpoint and render them into an occlusion map. Then they read back the occlusion map into main memory and calculate a hierarchical opacity map. They propose to use mip-mapping hardware to create the lower levels of the occlusion maps and the main processor to calculate the higher levels, where the maps are smaller. The opacity information at higher levels allows for approximate occlusion culling, using a user selected threshold. If an object is found to be occluded by the opacity test, it has to be tested against a depth-estimation buffer to guarantee that the object lies behind the occluders. The depth information buffer is another occlusion map that stores for each pixel the farthest value of an occluder in this pixel. The resolution of this map should be small compared to the screen resolution. Zhang used a 64x64 map for his results.

In summary the hierarchical occlusion maps are a good study of image-based occlusion culling on current hardware. The occlusion map can fuse an arbitrary number of occluders, without any geometrical restrictions. However, there are a few concerns regarding the design choices of the algorithm:

- The one pass solution might make the algorithm much more practical but it is not very systematic. This results in a degeneration of the algorithm for complex views.
- Even a one pass solution that reads the frame-buffer only once is rather slow.

Ho and Wang observe that the occlusion map could be stored as a summed-area table[Ho] and Bormann [Borm00] proposes layered hierarchical occlusion maps. However, these two methods do not address the main shortcomings of the hierarchical occlusion map algorithm.

A more systematic algorithm was proposed by Hey et al. [Hey01]. They propose to use a lower resolution occlusion map in addition to the z-buffer. In contrast to the hierarchical occlusion maps, the occlusion grid is updated more frequently. The main idea is the introduction of lazy updates: the occlusion map is not updated every time the z-buffer changes but only when the information is needed to decide the outcome of an occlusion query. Although the approach is more systematic, the algorithm computation time is even higher.

The best available image-based occlusion culling system is probably the Umbra system by Surrender 3D [Aila01]. The system creates occlusion maps in software and does not rely on read operations on

the frame-buffer. Although the system is faster than other algorithms, the overhead introduced through occlusion culling is still high.

In general we can observe the following problem: point-based occlusion culling algorithms are too slow. They introduce an overhead of several milliseconds so that the occlusion culling algorithm alone often takes longer than the time that can be allotted to each frame in a 60 Hz walkthrough.

In principle, image-based occlusion culling fits well in the concept of the rendering-pipeline. However, although frame buffer access is getting faster, the concept of reading back the frame buffer is not supported in modern graphics architectures and a read-operation from the frame-buffer will involve a high performance penalty. Therefore, several authors propose hardware extensions to solve the problem of occlusion culling (e.g. [Bart98, Bart99] or [Gree99b, Gree99a]).

Hardware-based point visibility

The main idea of hardware-based visibility queries is to render bounding box polygons of a scene-graph node without changing the frame buffer. The hardware extension can set a flag that indicated whether a fragment passed the z-test or even report the number of visible fragments. If a fragment passes the z-test the scene-graph node is potentially visible. Otherwise the bounding box is occluded and therefore the scene-graph node is occluded. As with the hierarchical z-buffer the scene-graph should be traversed in approximate front-to-back order.

HP has implemented occlusion queries in the *visualize-fx* graphics hardware [Scot98]. At the time of writing of this thesis the visualize-fx graphics system is not very widespread and other vendors have far greater importance on the graphics market.

SGI implemented a very similar occlusion query in a previous workstation line that is no longer supported in current workstations.

Newer graphics cards on the consumer market (ATI's Radeon [More00] and Nvidia's GeForce3) have a version of the z-pyramid, to accelerate rasterization. As per-pixel calculations are becoming more and more complicated, unnecessary calculations can be avoided by doing a z-test early in the pipeline. Improvements are memory savings by z-buffer compression, fast z-buffer clears and (hierarchical) fragment tests early in the pipeline. The details of ATI's and Nvidia's implementation are not publicly available.

The available information makes it hard to judge the near future of these hardware extensions. In particular the occlusion culling queries for bounding-boxes are essential for occlusion culling of general scenes. In the long run we would estimate that occlusion culling queries will be supported by most graphics cards.

3.3.4 Region visibility

In general point visibility algorithms have to be executed for every frame during runtime, which poses two problems:

- The visibility calculations are time consuming and use time that should rather be reserved for other tasks.
- Current online visibility calculations do not contribute to the problem of constant frame time. On the contrary: scenes with many visible objects often have higher algorithm calculation times.

Therefore, it is better for many applications to calculate visibility in advance for a region in space. Many algorithms work according to this framework: the viewspace is split into view cells (regions) and for each view cell visibility is calculated. The output of the visibility calculation is a potentially visible set (PVS) of objects per view cell. This PVS is stored on the hard disk. During runtime only the PVS of the current view cell (the view cell that contains the viewpoint) is rendered.

Cells and portals

For building interiors, most visibility algorithms partition the model into cells connected by portals. Cells roughly correspond to rooms and portals to doors connecting the rooms. Another cell is only seen through a sequence of portals. A viewer located in a cell A sees another cell X, if there exists a sightline that stabs the sequence of portals between cell A and cell X. The decomposition into cells can be made by hand or can be done automatically [Mene98]. Teller et al. [Tell91, Tell92b, Tell93] build an adjacency graph, where the nodes correspond to cells and the arcs to portals. To determine which cells are visible from a cell A, the graph is traversed in depth-first order. The path to a cell X corresponds to a portal sequence that can be tested for visibility:

- For simple 2D scenes, visibility can be calculated using linear programming.
- For rectangular axis-aligned portals in 3D, the problem can be solved by projecting it in 2D along the three axis directions.
- For general 3D a conservative solution can be found using separating and supporting planes.

If the cell X is visible the cell can be added to the list of visible cells for the cell A, and the traversal continues with the neighbors of the cell X. If the cell X is invisible, the traversal can terminate. Cell-to-cell visibility can be refined to cell-to-object visibility for better results and during runtime also to eye to object visibility.

Airey [Aire90] proposes a visibility test through ray casting. A 2D algorithm is proposed by Yagel and Ray [Yage96], that uses a rasterization of the whole scene into a regular grid and calculates occlusion due to opaque cells.

Luebke's algorithm [Lueb95] completely eliminates the precalculation of the PVS and precomputes only the cell and portal decomposition. During runtime the portals are projected to the screen. This results in an axis aligned screen space rectangle. This rectangle is intersected with the projection of the portals prior in the portal sequence. If the intersection is empty the recursion can stop. If the intersection is not empty the cell is visible and the rectangle can be used to quickly reject invisible objects.

Urban environments were stated as a possible application for cell and portal algorithms, but no implementation was reported. It might be hard to find a good partition into cells and portals of the model. Furthermore, the height structure is more complicated. For example distant objects (e.g. a tower) can be seen through a potentially very long sequence of portals.

Region visibility using large convex occluders

If a cell and portal decomposition is not available, occlusion can still be calculated for arbitrary view cells. However, the visibility in 3D from a view cell is rather complicated. To avoid involved calculations, the problem can be greatly simplified by considering only large convex occluders and ignoring the effects of occluder fusion.

Cohen-Or et al. [Cohe98b, Nadl99] use ray casting to test if an object is occluded by a single convex occluder. For each object they test occluders to check if the object is hidden. They cast rays from the vertices of the view cell to the vertices of the bounding box of the tested object. Because occludees have to be testing against multiple occluders the test is quite expensive. Cohen-Or et al. discuss several methods to reduce some of the high costs.

The visibility octree [Saon99] is a similar method that computes occlusion due to single large convex occluders for each node of an octree.

Region visibility with large convex occluders can detect some occlusion but the question is whether it is sufficient. Durand [Dura00] showed in an informal comparison of his method to the algorithm by Cohen-Or et al. [Cohe98b] that the amount of occlusion due to the combined effect of multiple occluders (occluder fusion) can be significant.

Approximate region visibility using occluder fusion

In this section we will discuss methods for occlusion culling that consider occluder fusion. These methods deal with the same problems as our algorithm described in chapter 5. All those methods, including ours, were developed independently and concurrently and a more detailed discussion of the algorithms in comparison to our method is presented after a description of our work in chapter 5.

Durand et al. [Dura00] propose extended projections that can be seen as an extension of image space visibility, like the hierarchical occlusion maps, to area visibility. They place six projection planes around the view cell. The algorithm proceeds for each of the six planes separately. First they project selected occluders onto a plane. Then they read back the result from the frame buffer and build a hierarchical z-pyramid. To detect occlusion, bounding boxes of scene-graph nodes are projected onto the plane. As in other image-based occlusion culling methods, a bounding box is occluded if its depth values are behind the depth values in the occlusion map. The key part is how to calculate the projection of occluders and occludees, to guarantee a conservative result. They define the *extended projection* of occluders onto a plane to be the intersection of all views in the view cell, and the *extended projection* of an occludee as the union of all views in the view cell. To assign depth values to the projection they define the *extended depth* that is the maximum depth of the occluders (or the minimum depth for the occludees). The practical implementation is quite complicated in the general case, therefore Durand et al. discuss several optimizations and simplifications for special configurations. One feature of this projection is that the position of the projection plane should be near the occluders. Therefore an extension is proposed that calculates an occlusion sweep. The occlusion aggregated on a plane can be reprojected on a plane behind. A similar version of this algorithm was proposed already in 1991 by Lim [Lim92].

Schaufler et at. [Scha00] use volumetric occluders for occluder fusion. They insert the scene into an octree. Their insertion relies on the fact that occluders are ”water tight”. Therefore, they can mark certain octree cells as opaque and use them as occluders. For occluder fusion they combine neighboring octree cells that are opaque or that are already found to be occluded. To determine occlusion due to a volumetric blocker, an axis aligned box, they calculate the shadow and mark it in the octree as occluded. To calculate a volumetric blocker they select an octree cell as occluder and try to expand it as far as possible. They build on the following observation: An occluder can be expanded to neighboring cells, if they are opaque or most importantly if they are already found to be occluded. They propose heuristic occluder extension using a greedy algorithm. With these building blocks they traverse the scene front-to-back and calculate the occlusion due to opaque octree cells. Before occlusion calculations they use blocker extension. This algorithm is easy to understand and should be simple to implement.

Very recently another interesting algorithm was introduced by Bittner [Bitt01a]. His algorithm works in line space and is essentially two-dimensional. In line space an occluder can be seen as a polygon that describes which rays are occluded. To calculate the combined effect of multiple occluders, Bittner merges the polygons using BSP trees. The interesting part of this algorithm is an extension to 2.5D that works very well for urban environments. At the time of writing of this thesis this work is not yet finished, but preliminary results indicate fast calculation times and good scalability [Bitt01b].

Ideally, a visibility algorithm should be very simple and work for general scenes. A robust method is point sampling from several locations within the view cell. The point sampling can be done using graphics hardware or ray casting. This method is easy to implement and is a good reference solution for a comparison to test the quality of a conservative algorithm (e.g. [Deco99, Gots99, Aire90]). However, the possible danger is to miss small objects or cracks between two occluders, so that the visibility solution is not conservative.

3.3.5 Parallel visibility

On a multiprocessor system occlusion culling can be calculated in parallel for the cost of latency. While a frame n is rendered occlusion can be calculated for the frame $n+1$ on another processor. This implies that the viewpoint for the next frame has to be known in advance, which is only possible if latency is introduced to the pipeline.

This approach is used in the Performer toolkit [Ecke00] and the UNC massive model walkthrough system [Alia99a]. The UNC system uses hierarchical occlusion maps [Zhan97], which poses the problem that the graphics hardware has to be used to render the occluders.

3.3.6 Related problems

Occluder selection and synthesis

One common problem for all occlusion culling algorithms is the selection and synthesis of good occluders. Many algorithms use ad hoc methods for their results which use prior knowledge about the scene structure.

The straightforward method is the selection of large scene polygons. However, in many scenes such polygons do not exist. Modern graphics cards can render several millions of polygons per second so that interesting scenes are highly tesselated. Therefore occluders have to be synthesized from the input scene [Adúj00a, Bern00].

Koltun et al. [Kolt00] propose a 2D algorithm for urban environments. They aggregate occlusion from several smaller buildings to obtain a large occluders for a view cell. The algorithm is done for several 2D cross sections parallel to the ground plane so that the height structure can be captured. These occluders are valid for only one view cell but these view cells can be several hundred meters long.

Occluder levels of detail

To accelerate occlusion culling, occluders can be used in several levels of detail. According to a heuristic, more important occluders can be used in a high level-of-detail and less important occluders can be used in a lower level to save computation time. For the level-of-detail generation it is important to note that lower levels of detail should not occlude more than higher levels. Therefore lower levels of detail should be completely contained in the original model.

Zhang [Zhan98] uses a modified version of the simplification envelopes [Cohe96]. Law et al. [Law99] show how to add additional constraints to a level-of-detail algorithm and use a modified version of quadric error meshes [Garl97] for their results.

Stress culling

When high guaranteed frame rates for arbitrary scenes are an important goal, several objects have to be omitted or can only be rendered in a lower level of detail. To select which objects should be rendered in a lower level-of detail, visibility calculations can be used. The basic idea is that objects that are likely to be occluded can be rendered with a lower level of detail [Adúj00b, Klos99, Klos00]. This heuristic selection of levels of detail results in popping artifacts that can be strongly noticeable. Although these approaches are beneficial for some applications, the image quality is not sufficient for most applications of urban simulation.

3.4 Summary

Visibility is a complex subject. Therefore most algorithms were designed with certain applications in mind and use several assumptions and tradeoffs to obtain a practical solution. Tradeoffs include limitations to convex occluders (no concave occluders), discretization of occluders, simplified handling of occluder fusion, heuristic selection of occluders, tight approximation instead of conservative classification, and the use of bounding boxes instead of actual geometry. These tradeoffs result in algorithms that are by nature more suitable for a certain type of scene and less suitable to others. If we consider that the visibility problems are usually well stated (calculate visibility from a point and calculate visibility from a region),

there is a comparatively large number of algorithms that deal with these problems. We attribute that to the fact that most algorithms rely on rather strong assumptions, and are therefore only suitable for a small number of scenes or applications.

Another aspect is that visibility algorithms are often used as basis for other algorithms. Examples are database prefetching in client-server walkthrough systems, image-based rendering, global illumination and level-of-detail generation. This leads to a constant demand for new algorithms.

Therefore, it is likely that visibility will continue to be an open research topic, with room for new ideas. In this sense the work in this thesis can hardly solve visibility problems for a broad range of applications but is especially designed to build a real-time simulation system of urban environments.

Chapter 4

Visibility from a Point

This section introduces a new data structure to store occlusion information for 2.5D occluders. The representation is an occlusion map that contains the *orthographic* projection of occluder shadows. We will show how to exploit graphics hardware by rendering and automatically combining a relatively large set of occluders to calculate visibility from a point. This occlusion map can be used to create an online occlusion culling system for urban environments.

4.1 Introduction

The original motivation to create a new occlusion culling algorithm for urban environments was the high overhead of previous algorithms. First of all, a good occlusion culling algorithm needs low calculation times to be usable. The calculation times should be low in comparison to the rendering time. We assumed that an algorithm that takes up 50%–80% of the frame time is inefficient and would not be able to compensate for its calculation for many frames, where occlusion is low. Our original goal was to achieve frame rates of 20 Hz and an algorithm calculation time of 20% of the frame time. Precalculation times should be as low as possible.

Another problem of previous algorithms was the use of heuristics to select occluders. A good occlusion algorithm should use all important occluders if possible because any holes reduce the performance of the algorithm (see section 2.3). Therefore, the second goal of this algorithm was to allow handling a large number of occluders.

The last goal was to have all forms of occluder fusion, which is typical for image-based occlusion culling. To sum up, the algorithm should

- be fast to calculate.
- handle a large number of occluders.
- calculate occluder fusion.

In this chapter, we introduce a new hybrid image-based and geometrical online culling algorithm for urban environments exhibiting the properties discussed above. We compute *occluder shadows* (Figure 4.1) to determine which parts of the environment are invisible from a given viewpoint. Occluder shadows are *shadow frusta* cast from selected occluders such as building fronts. The 2.5D property of an urban environment allows us to generate and combine occluder shadows in a 2D bitmap using graphics hardware. Our algorithm has the following properties:

- It uses a relatively large set of occluders (up to 500) that are automatically combined by the graphics hardware.

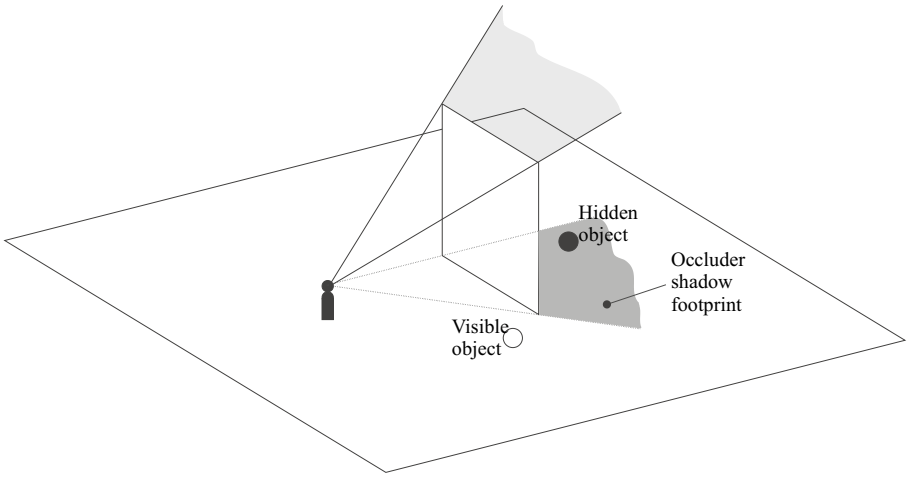


Figure 4.1: An occluder shadow is used for occlusion culling. Note how the roof top of the shown building facade allows to determine whether an object is hidden from its occluder shadow footprint in the ground plane.

- The algorithm is reasonably fast to calculate (about 13 ms on a 1998-mid-range workstation for our environment of 4 km^2) and therefore also useful for scenes of moderate complexity (Figure 4.7) and walkthroughs with over 20 fps.
- We calculate the occlusion dynamically. However, useful occluders might need to be extracted in a preprocess.
- The algorithm is simple to understand and implement.

The structure of this chapter is as follows: We give an overview of our algorithm and an explanation of the various stages of our acceleration method. Next we will describe our implementation and give results of different tests made with our example environment. Then we will discuss the applicability of our algorithm and compare it to the other existing solutions. Finally, we will present ideas to achieve further optimizations.

4.2 Overview of the approach

4.2.1 Occluder shadows

The concept of occluder shadows is based on the following observation: Given a viewpoint O, an occluder polygon P casts an occluder shadow that occludes an area of the scene that lies behind the occluder, as seen from the viewpoint. This area can be seen as a shadow frustum that is determined by the occluder and the viewpoint. Objects fully in this shadow frustum are invisible.

A more formal definition of an occluder shadow (Figure 4.2) is given in the following (Please note that the definition is also valid if the urban environment is modeled on top of a height field but this complicates the definition and is omitted here for brevity):

- The environment is required to have a ground plane π with a normal vector N (up direction).
- An occluder polygon P is a (not necessarily convex) polygon with its normal vector parallel to the ground plane (i.e., P is standing upright). P is constructed from a set of vertices $V = \{v_1, \ldots, v_k\}$, connected by a set of edges $E = \{e_1, \ldots, e_k\}$. An edge e from va to vb is written as $e = (va, vb)$. At least one edge of P must lie in the ground plane.

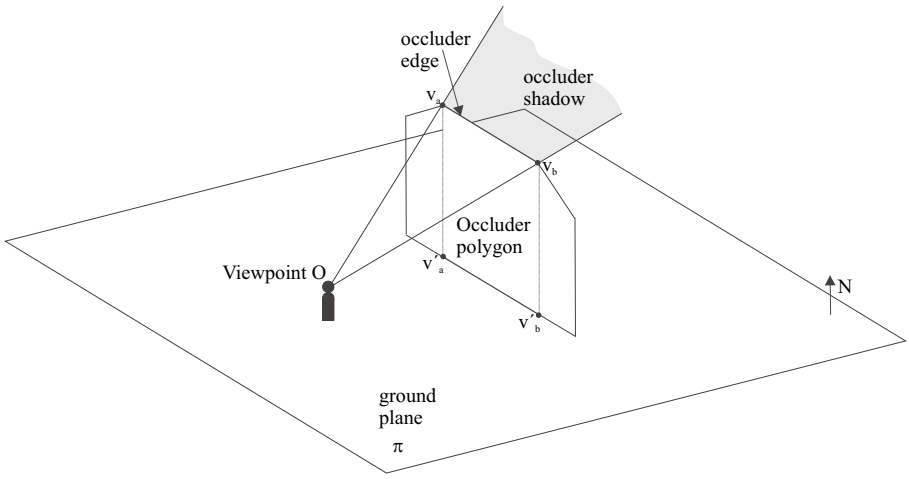


Figure 4.2: This sketch shows the geometric relationship used for the occlusion culling. An occluder shadow is constructed from a given viewpoint and an occluder edge of an occluder polygon P.

- An occluder edge is an edge $e \in E$ with $e = (va, vb)$ of P that does not lie in the ground plane. Let va' and vb' denote the normal projection of va and vb onto π, respectively. If the polygon defined through the vertices va, va', vb', vb is completely inside P the edge e is a valid occluder edge.
- An occluder shadow is a quadrilateral with one edge at infinity, constructed from an occluder edge $e = (va, vb)$ and two rays: $va + \lambda(va - O)$ and $vb + \mu(vb - O)$. When looking in $-N$ direction, the occluder shadow covers all scene objects that are hidden by the associated occluder polygon when viewed from O.

We exploit this observation by rendering the occluder shadow with orthogonal projection into a bitmap coincident with the ground plane. The height information (z-buffer) from this rendering allows the determination of whether a point in 3D space is hidden by the occluder polygon or not.

4.2.2 Algorithm outline

This section will explain how the concept of occluder shadows is used for rendering acceleration. We assume an urban model as described in the introduction. We rely on no special geometric features concerning the model. We need no expensive preprocessing and all needed data-structures are built on the fly at startup. For a completely arbitrary input scene, two tasks have to be solved that are fully addressed in the explanation of this algorithm:

- The scene has to be decomposed into objects for the occlusion culling algorithm, which can be a semantic decomposition into buildings, trees and road segments, or a plain geometric decomposition into polygon sets.
- Useful occluders - mainly building fronts - must be identified. A more detailed discussion about these problems is given in chapter 7.

The scene is structured in a regular 2D grid coincident with the ground plane, and all objects are entered into all grid cells with which they intersect. During runtime, we dynamically select a set of occluders for which 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 ground plane of the scene (see section 4.3.2). Visibility can be determined for each cell (or object) of the grid according to the values 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 (see section 4.3.3).

When we render an occluder shadow with the graphics hardware, we obtain z-values and color-buffer entries that correspond to a sample in the center of the pixel. This is not satisfying for a conservative occlusion calculation. For a correct solution, we have to guarantee that (1) only fully occluded cells are marked as invisible, and (2) that the z-values in the z-buffer correspond to the lowest z-value of the occluder shadow in the grid cell and not a sample in the center. How this is achieved is detailed in section 4.3.4.

To summarize, during runtime the following calculations have to be performed for each frame of the walkthrough:

- *Occluder selection*: For each frame a certain number of occluders has to be selected, that have a high chance to occlude the most parts of the invisible portion of the scene. The input to this selection is the viewpoint and the viewing direction.
- *Draw occluder shadows*: The selected occluders are used to calculate occluder shadows which are rendered into the cull map using graphics hardware.
- *Visibility calculation*: The cull map is traversed to collect all the potentially visible objects in the scene. Objects that are definitely not visible are omitted in the final rendering traversal.
- *Hidden surface removal*: The selected objects are passed to a hidden surface removal algorithm (z-buffer hardware).

4.3 Culling algorithm

4.3.1 Preprocessing

At startup, we have to build two auxiliary data-structures: the scene grid and the occluder grid. To construct the scene grid we have to organize all objects in a regular grid covering the whole environment. All objects are entered into all grid cells with which they collide, so that we store a list of objects for each grid cell. Usually objects span more than one cell. As occlusion is computed on a cell basis, the complexity and size of the environment influence the choice of the grid size.

Occluders have to be identified. In our implementation, occluders are created by the modeling system and read from a file at startup (see chapter 7). The synthesis of good occluders from arbitrary scenes is an open research problem. In our current implementation we use extruded building footprints as volumetric occluders. Each edge of the building footprint corresponds to a façade of the building. An edge together with the building height defines an occluder polygon and we store connectivity information between occluder polygons for more efficient rasterization (see section 4.3.4).

These tasks can be done very fast and require an amount of time comparable to the loading of the scene.

4.3.2 Cull map creation

Once an occluder polygon is selected, the associated occluder shadow quadrilateral is rendered into the cull map. Since a quadrilateral with points at infinity cannot be rendered, we simply assume very large values for the λ and μ parameters from section 4.2.1, so that the edge in question lies fully outside the cull map. When the quadrilateral is rasterized, 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 (Figure 4.3 and figure 4.8).

When occluder shadows intersect, the z-buffer automatically stores the z-value, which provides the best occlusion.

Previous approaches selected a set of occluders according to a heuristic depending on the distance from the viewpoint, the area and the angle between the viewing direction and the occluder-normal [Bitt98, Coor97]. This is a necessary step if only a small number of occluders can be considered, but it has the disadvantage that much occlusion can be missed. In our approach, the number of occluders is large enough, so we use only the distance from the viewpoint as a criterion for dynamic selection during runtime (Figure 4.9).

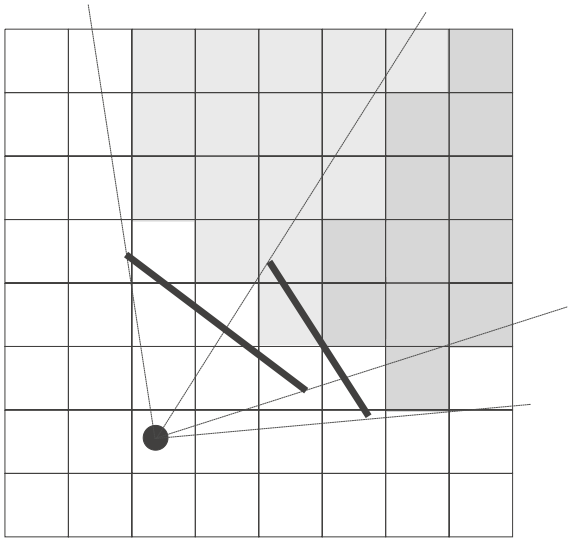


Figure 4.3: The cull map is created by drawing occluder shadows as polygons using graphics hardware. Overlapping occluders are correctly fused by standard rasterization and z-buffer comparison.

This is a good approach that guarantees systematic occlusion culling up to a certain distance. Beyond that distance it basically depends on luck, like all other heuristics. See section 4.6 for a discussion of the occluder selection.

We perform a simple backface culling test on all occluder candidates. Building fronts we consider are generally closed loops of polygons, and it is sufficient to consider only front facing polygons. For each of the occluders we render the occluder shadow in the cull map. The resulting cull map is passed to the next step, the actual visibility calculation.

4.3.3 Visibility calculation

To determine the visible objects that should be passed to the final rendering traversal, we have two options: We can either traverse the cull map or we can traverse the scene-graph to determine which objects are visible. While traversing the scene-graph would make a hierarchical traversal possible, we found that a fast scan through the cull map is not only simpler, but also faster.

Our algorithm therefore visits each cell that intersects the viewing frustum and checks the z-value in the cull map against the height of the objects stored in that cell. We use a two-level bounding box hierarchy to quickly perform that test: first, the z-value from the cull map is tested against the z-value of the bounding box enclosing all objects associated with the cell to quickly reject cells where all objects are occluded. If this test fails, the bounding boxes of all individual objects are tested against the cull map. Only those objects that pass the second test must be considered for rendering. In pseudo code, the algorithm looks like this:

```
for each cell C(i,j) in the grid do
  if C(i,j).z > cullmap(i,j)
    for each object O(k) in C(i,j) do
      if O(k).z > cullmap(i,j).z
      and O(k) not already rendered
        render O(k)
```

4.3.4 Cull map sampling correction

As stated in the previous section, the simple rendering of the occluder shadows generally produces a non-conservative estimation of occlusion in the cull map because of undersampling. For (1) correct z-values in the cull map and (2) to avoid the rendering of occluder shadows over pixels that are only partially occluded, we have to shrink the occluder shadow depending on the grid size and the rasterization rules of the graphics hardware. Our method to overcome these sampling problems requires information about the exact position used for sampling within one pixel. Current image generators usually take a sample in the middle of a pixel. For example, the OpenGL [Sega99] specification requires that (1) the computed z-value corresponds to the height in the middle of a pixel, (2) a pixel fragment is generated when the middle of a pixel is covered by the polygon and (3) for two triangles that share a common edge (defined by the same endpoints), that crosses a pixel center, exactly one has to generate the fragment.

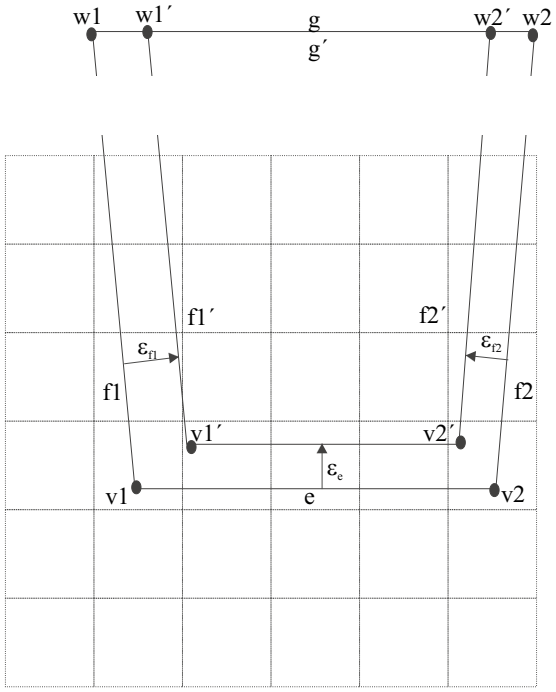


Figure 4.4: Sampling correction is performed by moving three edges of the occluder shadow inwards so that no invalid pixels can accidentally be covered.

Given these sampling constraints, a conservative solution for both stated problems is to shrink the occluder shadow geometrically (Figure 4.4). Consider an occluder shadow quadrilateral constructed from the vertices v_1, v_2, w_1, and w_2, defining the occluder edge $e = (v_1, v_2)$ and three other edges $f_1 = (v_1, w_2)$, $f_2 = (v_2, w_2)$ and $g = (w_1, w_2)$. By moving e, f_1, and f_2 towards the interior of the polygon along the normal vector by distances ε_e, ε_{f1}, and ε_{f2}, respectively, we obtain a new, smaller quadrilateral with edges e', f_1', f_2' and g', that does not suffer from the mentioned sampling problems. The edge g is outside the viewing frustum and need not be corrected, it is only shortened.

To avoid sampling problems in x-y-direction, the correction terms ε_e, ε_{f1}, and ε_{f2} have to depend on the orientation of e, f_1, and f_2, respectively. An upper bound for them is $l\sqrt{2}$, where l is half the length of a grid cell. However, it is sufficient to take $l(|n_x| + |n_y|)$, where n is the normalized normal vector on an edge that should be corrected. This term describes the farthest distance from the pixel middle to a line with normal vector n that passes through a corner of the pixel.

If the occluder or the angle between the occluder and the viewing direction is small, f_1' and f_2' may intersect on the opposite side of e as seen from e'. In this case, the occluder shadow quadrilateral degenerates into a triangle, but this degeneration does not affect the validity of the occlusion.

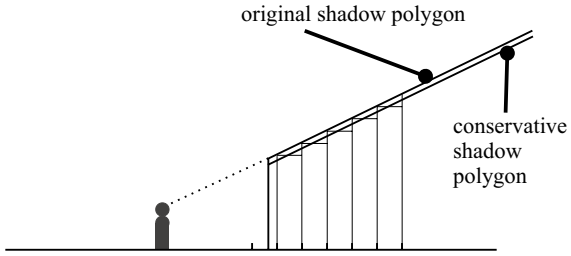


Figure 4.5: This figure shows the sampling correction problem for z-values. We have to render a *conservative shadow polygon* to guarantee that the discretized shadow volume remains conservative.

To avoid sampling problems in z-direction we have to guarantee that the discretized shadow volume is conservative (Figure 4.5). We render the new, smaller quadrilateral with a slope in z that is equal to or smaller than the slope of the original quadrilateral. In a case where the occluder edge e is parallel to the ground plane, the gradient (i.e., direction of steepest slope in z) of the occluder shadow is identical to the normal vector n of the occluder edge. Thus, the sampling problem for z-values is solved implicitly. If the occluder edge is not parallel to the ground plane, we have to consider an additional correction term that is dependent on the gradient of the occluder shadow. Note that only in this case is the z-value of the vertices v_1' and v_2' different from that of v_1 and v_2.

4.4 Implementation

Our implementation uses input scenes produced by a procedural modeling system. We use building footprints and building heights as input and generate geometry by extruding the footprints. Occluder information and other semantic structures are created by the modeling system and written into a separate file. We reuse this information instead of extracting occluder polygons at startup.

We implemented our algorithm on an SGI platform using Open Inventor(OIV) [Wern94] as high-level toolkit for the navigation in and rendering of the urban environment. The cull map handling is done with native OpenGL (which also guarantees conservative occlusion) in an off-screen buffer (pbuffer). OIV allows good control over rendering and navigation and is available on different platforms. In our cull map implementation, we use only simple OpenGL calls, which should be well supported and optimized on almost any hardware platform. The most crucial hardware operation is fast copying of the frame buffer and the rasterization of large triangles with z-buffer comparison. Hardware accelerated geometry transformation is not an important factor for our algorithm.

We tested and analyzed our implementation on two different hardware architectures: an SGI Indigo2 Maximum Impact, representing medium range workstations, and an O2 as a low end machine. Whereas the implementation of most tasks met our estimated time frames, the copying of the cull map showed significantly different behavior on various hardware platforms and for different pixel transfer paths. Where the time to copy the red channel of a 250x250 pixel wide frame buffer area on the Maximum Impact takes about 3 ms, this step takes over 20 times as long on the O2, where only the copying of the whole frame buffer (red, green, blue and alpha channels) is fast. These differences have to be considered in

the implementation. Furthermore, copying the z-buffer values on an O2 is more time-consuming and not efficient enough for our real-time algorithm.

Due to the fact that fast copying of the z-buffer is not possible on an O2, we had to resort to a variation of the algorithm that only needs to copy the frame buffer:

1. At the beginning of each frame, each value of the z-buffer is initialized with the maximum z-value from the objects in the corresponding grid cell. z-buffer writing is disabled, but not z-comparison. Next the color buffer is cleared and the viewing frustum is rendered with a key color meaning visible. The cells not containing any objects are initialized to have very high z-values, so that they are not marked visible by rendering the viewing frustum.

2. Each occluder shadow is then written with a key color meaning invisible overwriting all pixels (= grid cells) that are fully occluded.

3. Finally, the resulting frame buffer is copied to memory for inspection by the occlusion algorithm.

The sampling correction for the occluder shadows makes it necessary to keep information of directly connected occluders. An occluder polygon is represented by its top edge and all connected occluders are stored as closed poly-lines that correspond to the building footprints. We can shrink the building footprints in a preprocess (see chapter 7). After shrinking of the building footprints, the occluder polygons are moved towards the interior of the building, but they are still connected. The shrinking operation during runtime uses the original and the shrunk occluder to obtain conservative shadow polygons, without producing cracks between two occluder shadows stemming from the same building.

Furthermore, all occluder polygons have an oriented normal-vector that is used for backface culling. This helps to reduce the number of occluders by about 50%.

4.5 Results

To evaluate the performance of our algorithm we performed several tests using a model of the city of Vienna. We started with the basic data of building footprints and building heights and used a procedural modeling program to create a three-dimensional model of the environment. This made it possible to control the size and scene complexity of the test scene. We modeled each building with a few hundred polygons. For the first test, we created a smaller model with 227,355 polygons, which covers an area of about a square kilometer.

We recorded a camera path through the environment where we split the path in two parts. In the first part (until frame number 250) the camera moves along closed streets and places. This is a scenario typically seen by a car driver navigating through the city. In the second part the viewer moves in a wide-open area (in the real city of Vienna there is a river crossing the city).

For our walkthroughs, we configured our system with a grid size of 8 meters and we selected all occluders up to 800 meters (which is on the safe side). In most frames, we select between 400 and 1,000 occluders and drop about half of them through backface culling. The construction of the auxiliary data structures and occluder preprocessing does not take longer than 10 seconds, even for larger scenes, while loading of the geometry takes sometimes over a minute. The cull map size for this test is 128x128.

Figure 4.6 (left) shows the frame times for the walkthroughs on the Indigo2 in ms. The curve “frustum culling” shows the frame times for simple view-frustum culling. The second curve “occlusion culling” shows the frame times for our algorithm. We see that we have good occlusion and that the algorithm is fast to compute. We have a speedup of 3.7 for the Indigo2. Furthermore, the frame rate stayed over 20 fps. The frame rates in the second part are also high because the model is structured in a way so that little additional geometry comes into sight.

For the second test, we used the same scene embedded in a larger environment to test (1) the robustness of the occlusion and (2) the behavior of the algorithm when the viewpoint is located on wide-open places.

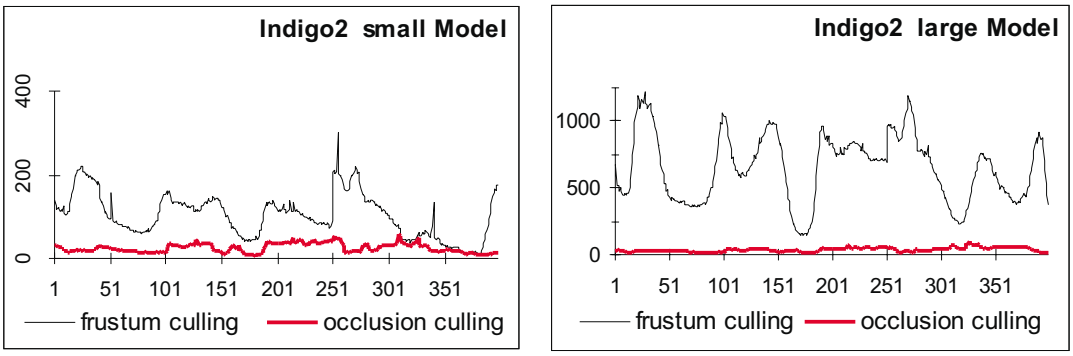


Figure 4.6: (Left) The frame times for the walkthrough of a model consisting of 227,355 polygons on an Indigo2 (frame times on y-axis in ms). Right: The frame times for the walkthrough of a model consisting of 1,300,000 polygons on an Indigo2 (frame times on y-axis in ms).

Indigo2	Model 1	Model 2
Frustum culling	103 ms	653 ms
Occlusion culling	25 ms	37 ms
Algorithm time	7 ms	13 ms
Part 1	26 ms	32 ms
Part 2	24 ms	44 ms
Speedup	4.2	17.8
O2	Model 1	Model 2
Frustum culling	312 ms	2029 ms
Occlusion culling	68 ms	108 ms
Algorithm time	13 ms	25 ms
Part 1	68 ms	84 ms
Part 2	67 ms	149 ms
Speedup	4.6	18.8

Table 4.1: Summary of all measurements for the two walkthrough sequences. Speedup factors between 4 and 18 were be obtained. Frustum culling: average frame time using view-frustum culling. Occlusion culling: average frame time using our occlusion culling algorithm. Algorithm time: average calculation time of our algorithm. Part1 (Part2): average frame time of the walkthrough in Part1 (Part2). Speedup: speedup factor compared to view-frustum culling.

The new city has about 1,300,000 polygons and covers a size of 4 km^2 (Figure 4.7). The cull map size for this test is 256x256. To be able to compare the results, we used the same camera path in the larger environment. The results for the Indigo2 are shown in figure 4.6 (right) and a summary of all results is given in Table 4.1.

It can be seen that the frame rate in the first part is almost the same as in the smaller model. Almost exactly the same set of objects was reported visible for both city models. Longer frame times are only due to higher algorithm computation time and overhead from data structure management. In the second part of the walkthrough, the field of view is not fully occluded for several hundred meters (sometimes up to 700) and a large number of buildings becomes visible. Still our algorithm works efficiently and selects only few invisible objects.

This evaluation demonstrates that the occlusion is robust and that our algorithm generally does not leave unoccluded portions in the field of view if suitable occluders are present (see Part 1 of the walkthrough). Even the low-end O2 workstation was able to sustain a frame time of 108 ms.

In the second part of the walkthrough we demonstrated the algorithm behavior in a more challenging

Indigo2				
Cell size	8m	6m	4m	2m
Cull map size	128x128	174x174	256x256	512x512
Render shadows	5.3	5.3	5.3	8
Copy cull map	1.3	1.7	2.3	6.3
Traverse cull map	0.5	0.7	0.9	2.2
Render buildings	18.3	17.4	17	15.8
complete frame time	25.4	25.0	25.5	32.3

Table 4.2: This table shows the results of the walkthrough in the smaller model on the Indigo 2 for different cell sizes (cull map sizes). All occluders in the viewing frustum were selected for this test (time values are given in ms).

environment, where dense occlusion is no longer given. The performance drops as expected, but nevertheless the frame time of the Indigo2 does not exceed 85 ms (the maximum frame time) and the average frame time for the second part also stays below 50 ms. The overall improvement for the whole path (compared to simple view-frustum culling) equals a factor of about 18. However, results also indicate that performance depends on the presence of a large amount of occlusion. To unconditionally sustain a high frame rate, a combination with other acceleration methods (see proposed future work) is desirable.

A third experiment was conducted to examine the theoretical performance limits of the algorithm. The goal was to assess how the amount of detected occlusion is related to grid size. Completely accurate occlusion can be computed if all possible occluders are rendered into a cull map with infinitely small grid elements. We therefore performed a walkthrough for model 1 with cell sizes of 8, 6, 4, and 2 meters. All occluders in the viewing frustum were rendered into the cull map unconditionally. A breakdown of the times for individual parts of the algorithm is given in Table 4.2.

The times reported for actual rendering of buildings indicate that a standard cell size of 8m has only about 20% overhead compared to the much smaller 2m cells. This overhead is mainly due to the fact that whole buildings are treated as one object. Despite its discrete nature, the algorithm estimates true visibility very accurately. We expect that better results can be achieved with more careful scene structuring while keeping cell size constant.

We also observed that the rasterization of occluder shadows and copying of the cull map becomes the bottleneck of the system for larger environments. Cull map traversal is really fast for a typical setup (cull map $<$ 256x256), so that it is not necessary to find optimizations through hierarchical traversals.

4.6 Discussion

Algorithm calculation times: For the applicability of an occlusion culling algorithm we found that fast calculation times are an important feature. If we assume the goal of 20 fps for a fluid walkthrough, we have only 50 ms per frame. If 30 ms are spent on the occlusion algorithm, little is left for other tasks like rendering, LOD selection or image-based simplifications. Such an algorithm may accelerate from 2 to 10 frames per second, but not from 4 to 20 fps, which is a fundamental difference. It was our goal to have an algorithm that uses only 20% of the frame time for its calculations. However, although our algorithm is faster than previously published methods, it suffers from the same kind of problem on current consumer level hardware. It might be useful to accelerate from 4 to 20 fps but is too slow for 60 Hz walkthroughs on current consumer level hardware.

An expensive algorithm depends on strongly occluded scenarios to compensate for its calculation time, whereas a fast algorithm can also result in speedups for slightly occluded environments. Consider HOMs [Zhan97] used for the UNC walkthrough system [Alia99a], for which the calculation times of the occlusion algorithm itself on mid-range machines are relatively high. The pure construction time of the occlusion map hierarchy given the basic occlusion map on an SGI Indigo2 Maximum Impact is about 9 ms.

This is about the time for one complete frame of our algorithm on the same machine. However, it must be stressed that HOMs provide a framework that is suitable for far more general scenes which cannot be handled by our algorithm.

Occluder selection: The occluder selection is a weak part in our algorithm (see section 4.3.2): The selection of occluders in a radius around the viewer only works well if the viewer cannot see farther than this radius. However, we designed this algorithm with the idea of combining it with an online image-based rendering algorithm [Wimm99] that does not depend on further visibility calculations. Other options are to omit all distant objects (maybe in combination with the use of fog), or to use a more sophisticated heuristic similar to previous approaches [Coor97, Huds97].

For the area visibility algorithm (see chapter 5) we developed a hierarchical extension that uses systematic occluder selection. This extension would also work for point visibility, but the calculation times could vary greatly, a feature that is not very beneficial for the goal of constant frame rates.

Information: An advantage of our method is that it generates a discretized representation of the shadow volume. This feature allows us to use this algorithm as a basis for other calculations, like area visibility (see next chapter).

Current Hardware: Previously published geometrical (e.g. [Coor97, Huds97, Bitt98]) and image-based algorithms [Zhan97] calculate full three-dimensional occlusion, but will probably not be able to compete with the proposed algorithm in urban environments. Competitive algorithms were introduced only recently by Downs et al. [Down01] and Aila and Miettinen [Aila01]. Both algorithms use visibility driven occluder selection and are therefore more systematic than other methods. Additionally, the algorithms are implemented in software and do not use graphics hardware. Current hardware architectures for consumer level hardware do not encourage read access to the frame buffer. Although read-back times became much faster, efficient programming of current hardware results in a high latency, that can easily be over one frame (i.e., the actual rendering on the graphics hardware is over one frame behind). It is still possible to read back the frame buffer, but this results in an inefficient use of the graphics hardware.

4.7 Conclusions and future work

We have presented a new algorithm for fast walkthroughs of urban environments based on occluder shadows. The algorithm has proven to be fast, robust, and useful even for scenes of medium complexity and low-end graphics workstations. It is capable of accelerating up to one order of magnitude, depending mostly on support for fast frame buffer copying. This support is now also available on low-cost hardware for the consumer market.

Working on the combination of online occlusion culling and ray-casting, it became clear that

- online occlusion culling can hardly contribute to the goal of constant frame rates.
- online occlusion culling is too expensive on current hardware for walkthroughs at 60Hz or higher.

These problems are inherent to online occlusion culling and therefore it seems to be necessary to precalculate visibility. Our next chapter will describe an efficient algorithm for urban environments.

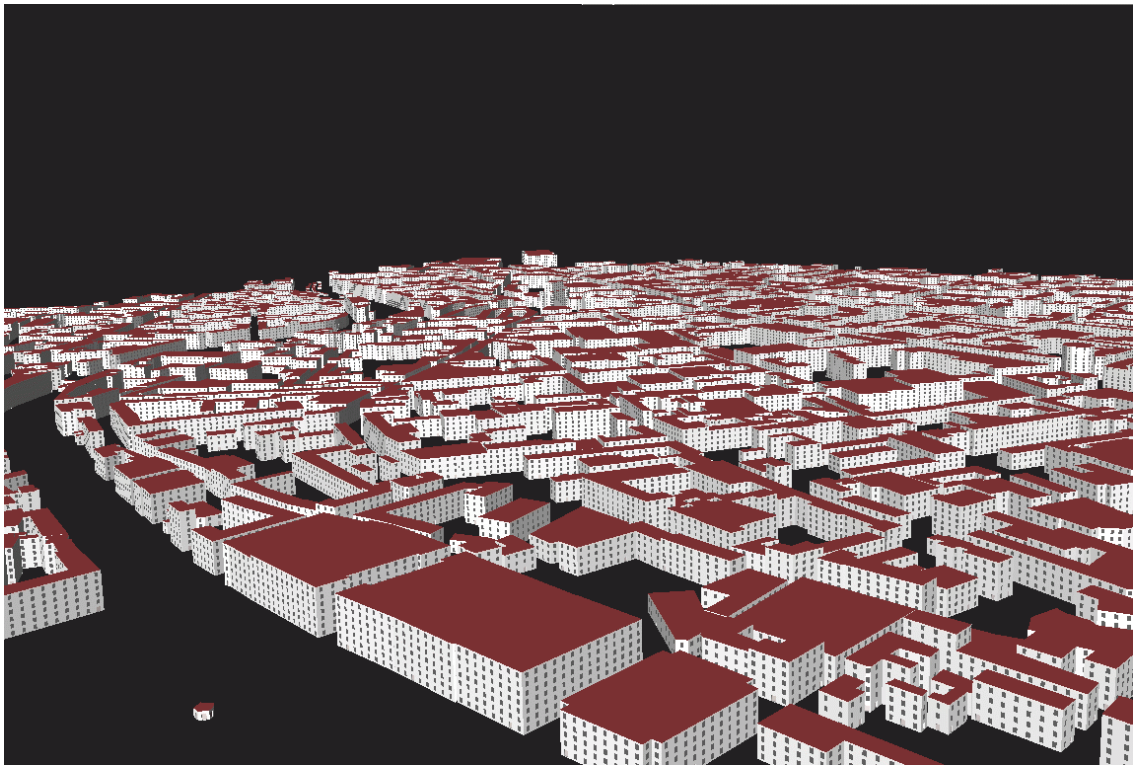


Figure 4.7: This model of the city of Vienna with approximately 1.3M polygons was used for our experiments.

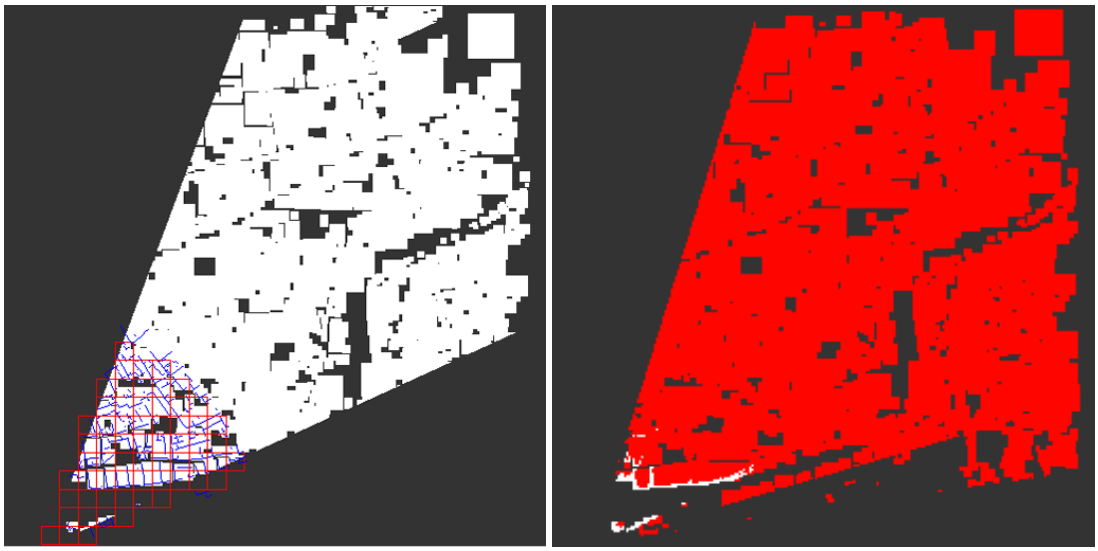


Figure 4.8: Two views of the cull map used for occlusion culling. The left view shows the grid cells inspected for suitable occluders (in red) and selected occluders near the viewpoint (in blue). The right view shows the culled portion of the model (in red) and the remaining cells after culling (in white).

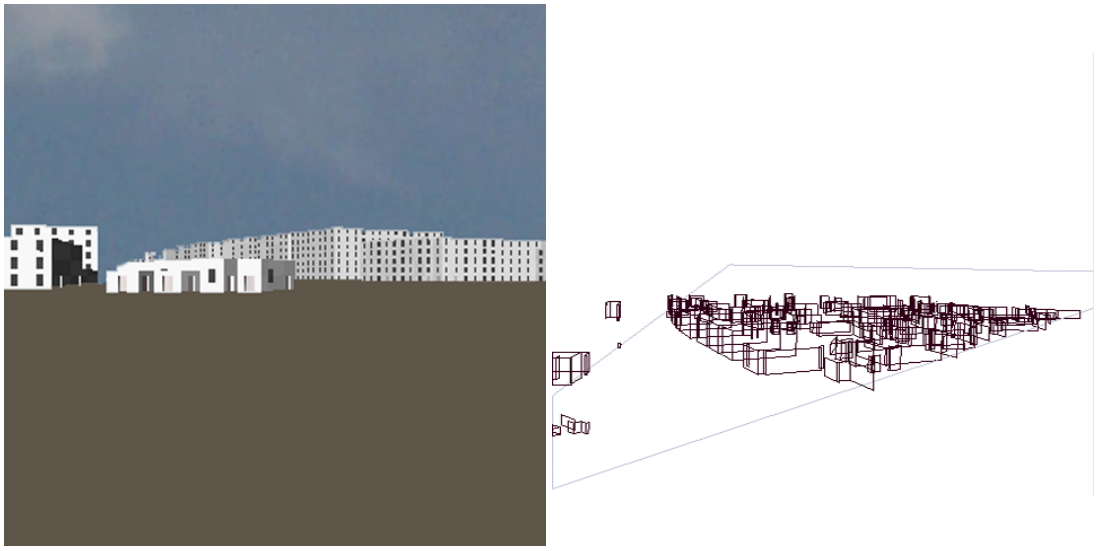


Figure 4.9: For the viewpoint used in figure 4.8, the resulting image is given on the left. The right view shows a wireframe rendering of the occluders to give an impression of occluder density.

Chapter 5

Visibility from a Region

This chapter presents an efficient algorithm to precompute visibility in urban environments. The algorithm 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. 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.

5.1 Introduction

As described in the previous chapter, visibility calculations for a single viewpoint have to be carried out online and infer significant runtime overhead. We observed that the proposed online algorithm cannot be used to achieve the two main goals of real-time rendering (see section 2.2): 1) the frame rates are lower than the monitor refresh rate (60Hz or more) and 2) a constant frame rate cannot be guaranteed.

These problems can be solved more easily, if occlusion is calculated in a preprocess. The issue of high frame rates could be addressed by improving the algorithm calculation time of the online algorithm, but the guarantee for a frame time is more involved. Even an exact visibility algorithm that uses no calculation time does not solve the problem because too many objects can be visible. For several frames in a walkthrough, it will be necessary to simplify distant geometry to accelerate rendering and to reduce aliasing. These simplifications often rely on visibility preprocessing (e.g. [Sill97, Deco99]). Therefore, it is useful to calculate visibility for a region of space (*view cell*) in a preprocessing step.

In this chapter we introduce an algorithm that calculates visibility for a view cell. The visibility algorithm is based on point-sampling and builds on the method described in the previous chapter: we use the concept of *occluder shadows* to combine visibility information of several point samples in a *cull map*. Our results demonstrate that this method is useful for walkthroughs with high frame rates (i.e., 60Hz or more).

5.1.1 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 (or volume) 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:

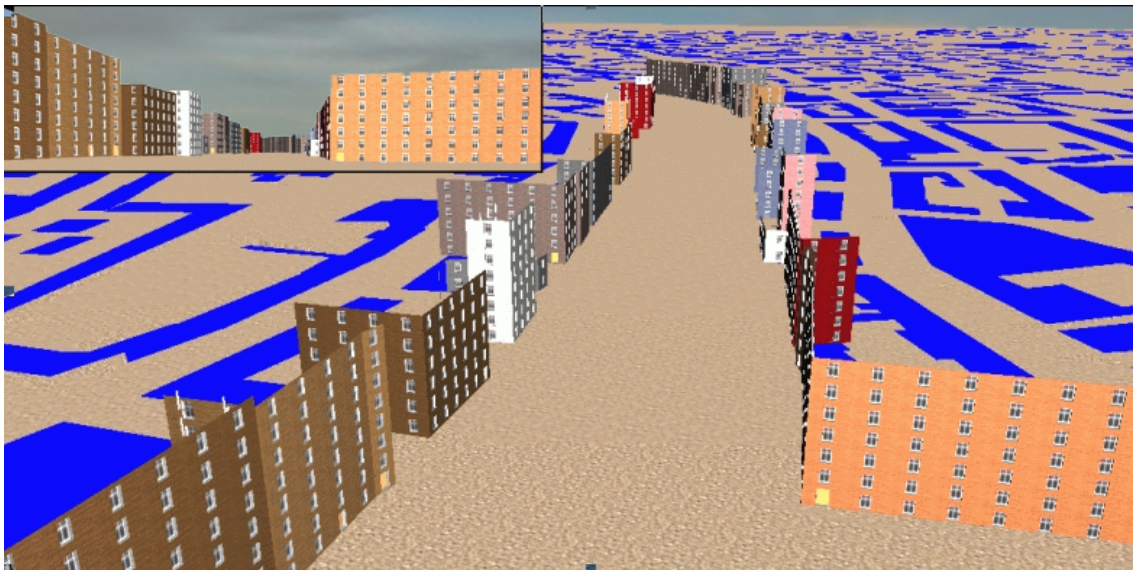


Figure 5.1: Views from the 8 million polygon model of the city of Vienna used in our walkthrough application. The inset in the upper left corner shows a typical wide open view, while the large image shows the portion of the scene rendered after occlusion culling. Note how occlusion fusion from about 200 visible occluders allows the pruning of over 99% of the scene.

- 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 [Tell92a, Dura99]. Such reguli are difficult to store, intersect etc.
- The complete set of viewing regions with topologically distinct views is captured by the aspect graph [Gigu91, Plan90], which is costly to compute (O(n^9) time). For applications such as radiosity, a more practical approach is the visibility skeleton [Dura97]. 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. (The umbra with respect to a view cell is the region of space from which no point of the view cell can be seen, whereas the penumbra is the region of space from which some, but not all points of the view cell can be seen.) 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 (Figure 5.2). 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 (Figure 5.2, letter a, b and c respectively).

This chapter 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

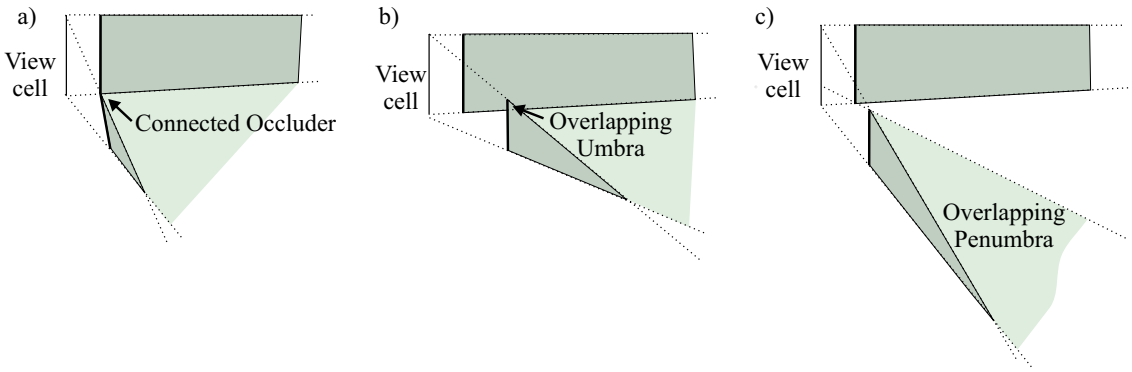


Figure 5.2: Different types of occluder interactions. Individual umbrae (shaded dark) cover only a small area, while the occluders jointly cover a large unbounded area (shaded light).

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 and even penumbrae is implicitly performed. This includes the hard case where individual umbrae of two occluders are disjoint (Figure 5.2, letter c). We also discretize umbra boundaries, which are typically made up of reguli (curved surfaces).

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.

5.1.2 Organization of the chapter

The remainder of the chapter is organized as follows. Section 5.2 introduces the main idea of how to calculate occluder fusion for the area visibility problem. Section 5.3 describes how the scene is structured into view cells. A short overview of occluder shrinking is described in section 5.4 (a detailed version can be found in chapter 7). Section 5.5 shows how to use graphics hardware to accelerate our algorithm, while section 5.6 presents a hierarchical extension to the main algorithm. Results are shown in section 5.7 and section 5.8 contains a discussion of our algorithm in comparison with other methods. Section 5.9 concludes the chapter and shows avenues of future research.

5.2 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.

5.2.1 Occluder fusion by occluder shrinking and point sampling

As can be concluded from the examples in figure 5.2, 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 which 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 (Figure 5.3).

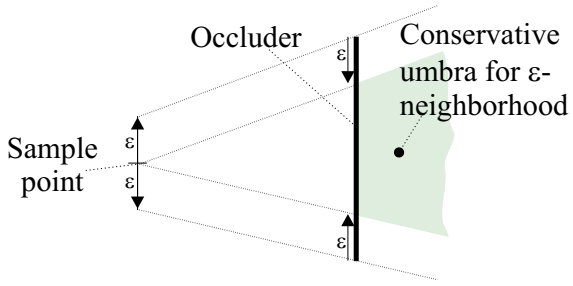


Figure 5.3: Occluder shrinking: By considering an occluder after shrinking it by ε, the umbra from a point sample provides a good conservative approximation of the umbra of the original occluder in a neighborhood of radius ε of the sample point.

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).

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 (Figure 5.4 and figure 5.5). 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.

5.2.2 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 the previous chapter and thus operates on city-like scenes (2.5D). While this type of environment may

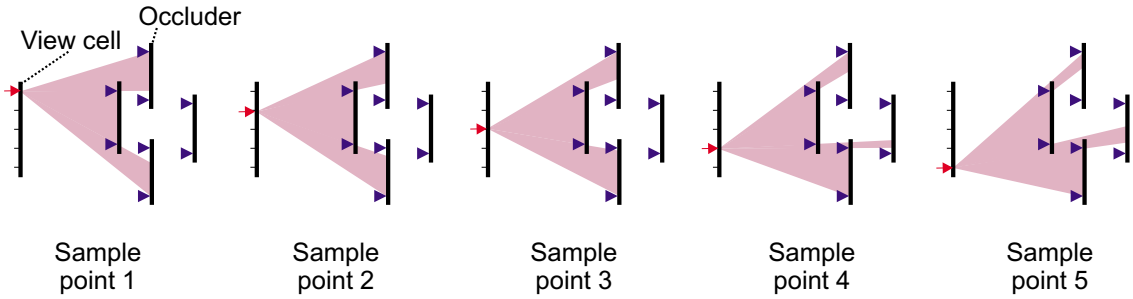


Figure 5.4: When performing point sampling for occlusion after occluders have been shrunk (as indicated by small triangles), all four occluders can be considered simultaneously, cooperatively blocking the view (indicated by the shaded area) from all sample points.

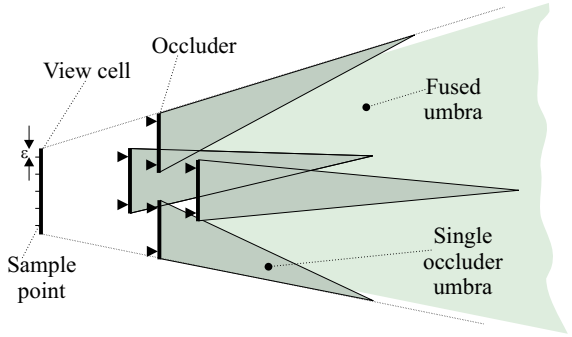


Figure 5.5: The fused umbra from the 5 point samples shown in figure 5.4 is the intersection of the individual umbrae (shaded light). It is here compared to the union of umbrae from the original view cell (shaded dark). As can be seen, point sampling computes superior occlusion and only slightly underestimates exact occlusion.

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 (Figure 5.3). 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 $\varepsilon \geq \frac{k\sqrt{2}}{2}$ (see chapter 4), rendering shadow polygons need not take into account rasterization errors. For a formal proof of this fact, see chapter 7. In practice, ε (along with the cull map grid size) is chosen to trade off accuracy of occluder shadow calculation against total preprocessing time.

5.2.3 Implications of occluder shrinking

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

1. It makes point sampling of 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 5.2), 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.

5.2.4 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 5.8.2).
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 5.3, 5.4, and 5.5 will explain the steps of this algorithm in detail, while section 5.6 presents a hierarchical extension of the algorithm which optimizes processing time.

5.3 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.

5.4 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. It is important to note that occluder shrinking is related to the problem of occluder selection and synthesis. Buildings are usually highly tesselated so that scene polygons cannot be used as occluders. Furthermore, the algorithm works primarily with volumetric occluders. For our implementation we chose to use the occluders that are created by our modeling system. Since buildings are procedurally created starting from building footprints, we use these extruded footprints as volumetric occluders and shrink them at startup (see chapter 7).

5.5 Rendering and merging occluder shadows

This section explains how to extend the frame buffer algorithm described in chapter 4 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 section 4.3.2 and figure 4.3). 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 chapter 4. 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.

5.6 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. The idea is to calculate

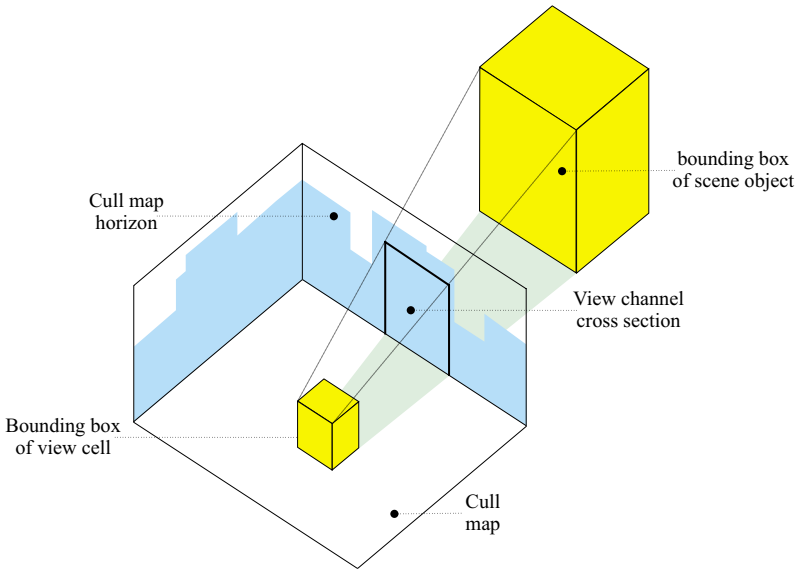


Figure 5.6: After computing an initial cull map, arbitrary parts of the scene can be quickly rejected by projecting their axis-aligned bounding boxes against the horizon of the cull map.

visibility for a smaller area around the viewpoint and use the result to determine occlusion of larger parts of the scene, especially of other occluders.

Starting from an initial size, e.g. 100x100 pixels, 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 (Figure 5.6). 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.

5.7 Implementation and results

A walkthrough system using the techniques outlined in this chapter 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.

5.7.1 Preprocessing

A model of the city of Vienna with 7,928,519 polygons was used throughout testing (Figure 5.9). 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. One building block consists of one up to ten connected buildings. Note that each building block was modeled with several thousand polygons (3,900 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 1 m, which led to the use of 6-8 sample points per view cell edge on average. An initial cull map size of 100x100 pixel (with a grid cell size of 2 m for the final walkthrough) was used, with a maximum cull map size of 1,500x1,200 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.

5.7.2 Quality

We experimented with different settings for sample spacing (ε), cull map grid size and cull map dimensions. Figure 5.7 shows the results for various grid sizes. As can be seen, larger grid sizes also yield acceptable occlusion, allowing the handling of 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 number 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 5.8.2 for a discussion of this fact.

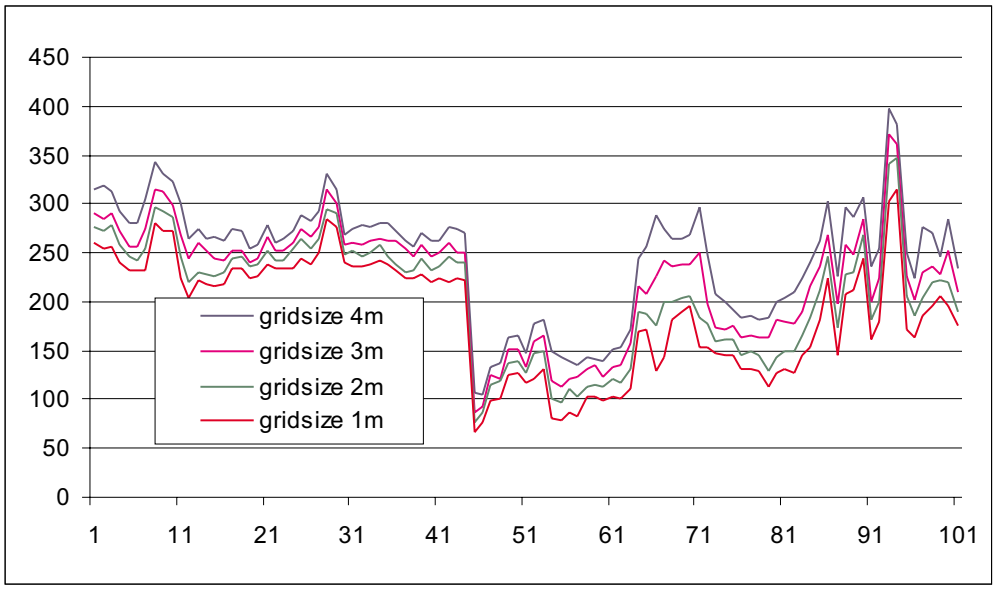


Figure 5.7: Influence of various grid sizes of the cull map on occlusion quality. Plotted are the number of visible objects (y-axis) against the view cells visited in the test walkthrough.

5.7.3 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 (including auxiliary rendering data-structures) and was fully loaded into main memory before the beginning of the walkthrough. Occlusion information (object IDs of potentially visible objects for each view cell) consumed 55 MB and was also fully loaded.

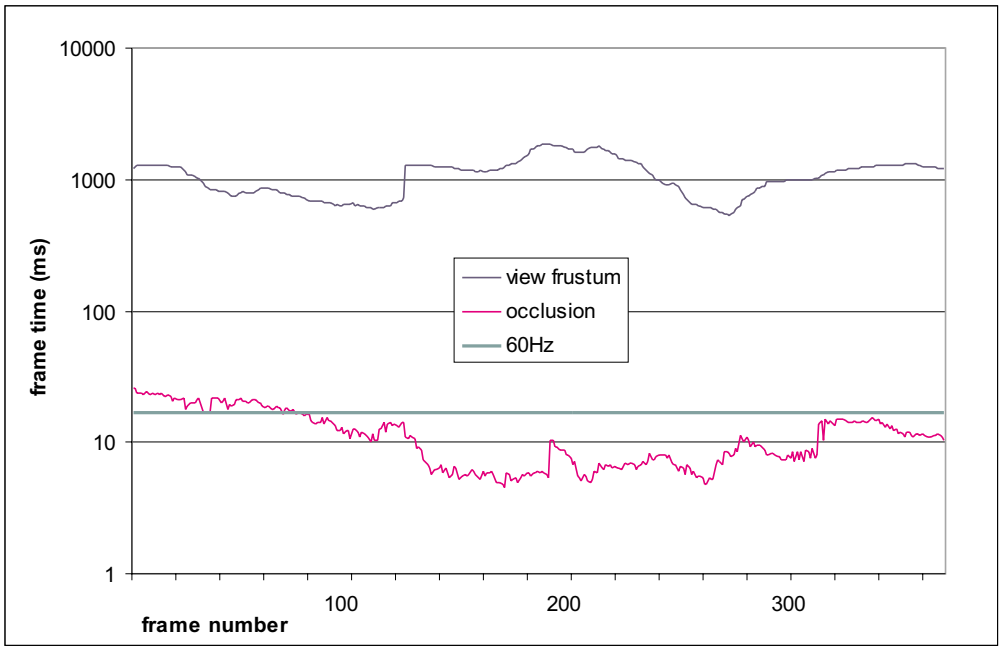


Figure 5.8: Frame times of occlusion culling vs. view-frustum culling for the walkthrough. Note that frame times are plotted on a log scale to fit two orders of magnitude into the diagram.

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 5.8 shows the frame times from our walkthrough for occlusion culling vs. view-frustum culling (logarithmic scale). Note that the desired 60 Hz are achieved for the second part of the walkthrough (rather dense occlusion), while the first part (wide open view) would require further optimizations (e. g. the use of simplified representations for distant objects).

5.8 Discussion

5.8.1 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. [Scha00] extend blockers into areas already found occluded, while Durand et al. [Dura00] 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 5.2, 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 already arise in simple cases. Even in 2.5D, a simple occluder with non-convex footprint gives rise to so-called 'EEE event surfaces' [Dura97], 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 discretization 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 5.5 is not straightforward, which is the main limitation of our algorithm.

5.8.2 Selection of occluders

Several algorithms achieve good results with a heuristic which selects a set of occluders that is most likely to occlude a large part of the scene [Coor97, Huds97]. However, we have found that the number of occluders that contribute to occlusion is typically quite large (as indicated by results presented in section 5.7.2). 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.

5.8.3 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 sole 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 disadvantage of our method is that it requires read access to the frame buffer. In current consumer hardware this access is reasonably fast[1], but still too slow to allow more efficient variations of the proposed algorithm. Ideally, we would like to test each occluder for occlusion just before it is rendered.

5.9 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) leads 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

[1] Note that the timings in this chapter were done using an older GeForce driver. Newer drivers, released after spring 2000, are an order of magnitude faster.

closely below or above the desired 60 Hz, show that no significant time is available for on-line occlusion calculation, and that precomputed occlusion is necessary for true real-time performance.

To sum up, we are approaching our goal of hard real-time rendering. We demonstrated that walk-throughs with high frame rates are feasible. The problem of constant frame rates, however, cannot be addressed by visibility calculations alone. Although our current research focuses on constructing suitable image-based representations for areas of the city where visibility preprocessing is not sufficient [Wimm01], the description of these methods is beyond the scope of this thesis. Instead, we will address another visibility-related problem in the next chapter—preprocessing times.

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 large data sets. However, precomputation times of several hours are still inconvenient. In the next chapter, we propose a method that eliminates preprocessing and calculates visibility for small view cells at runtime.

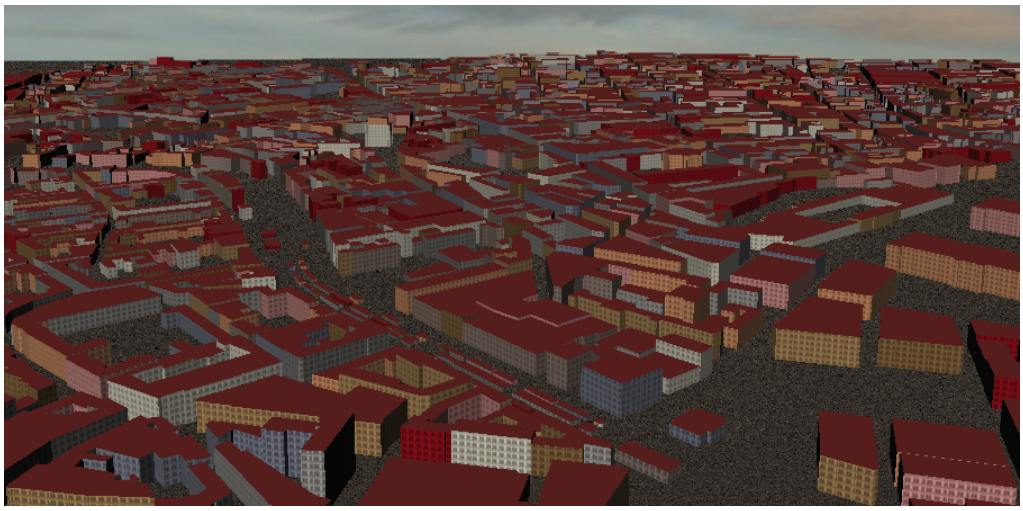


Figure 5.9: Overview of the 8 million polygon model of the city of Vienna used as a test scene. Note that the city is modeled in very high detail - every window, for example, is modeled separately with 26 polygons.

Chapter 6

Instant Visibility

We present an online occlusion culling system which computes visibility in parallel to the rendering pipeline. We show how to use point visibility algorithms to quickly calculate a tight potentially visible set (PVS) which is valid for several frames by shrinking the occluders used in visibility calculations by an adequate amount. These visibility calculations can be performed on a visibility server, possibly a distinct computer communicating with the display host over a local network. The resulting system essentially combines the advantages of online visibility processing and region-based visibility calculations, allowing asynchronous processing of visibility and display operations. We analyze two different types of hardware-based point visibility algorithms and address the problem of bounded calculation time which is the basis for true real-time behavior. Our results show reliable, sustained 60 Hz performance in a walkthrough with an urban environment of nearly 2 million polygons, and a terrain flyover.

6.1 Introduction

In chapter 4 we introduced a point visibility algorithm for urban environments. The general problem of point visibility algorithms is that they have to be executed for each frame and the renderer cannot proceed until a PVS is available. Their relatively long computation time significantly reduces the time available to render geometry, if not reducing the achievable frame rates below limits acceptable for real-time rendering applications.

Therefore, we concluded that hard real-time rendering can better be achieved with visibility preprocessing (see chapter 5). Precalculating visibility for a region of space (view cell) reduces almost all runtime overhead. However, there is a tradeoff between the quality of the PVS estimation on the one hand and memory consumption and precalculation time on the other hand. Smaller view cells reduce the number of potentially visible objects and therefore improve rendering time. However, smaller view cells also increase the number of view cells that need to be precomputed, which can result in prohibitively large storage requirements and precalculation times for all PVSs. Another problem is that many runtime modifications of the scene cannot be handled, and even small offline changes to the model might entail several hours of recomputation. This makes region visibility a viable choice for certain models (as, for example, in a computer game), but impractical for dynamic systems where changes to the model occur frequently (as, for example, in an urban modeling scenario).

In this chapter we address the aforementioned problems. We show how to achieve a large improvement over previous systems by adding new hardware to the system in the form of an additional machine in the network which is used as a visibility server. We calculate visibility at *runtime*, avoiding memory problems because the PVS need not be stored, but for a *region*, allowing it to be calculated in parallel to the rendering pipeline so that it imposes virtually no overhead on the rendering system. This results in *Instant Visibility*, a system which calculates a tight PVS with very little preprocessing and practically no runtime overhead on the graphics workstation.

The remainder of the chapter is organized as follows: after reviewing some relevant work, section 6.2 gives an overview of the system, and section 6.3 gives a detailed description of the various algorithms involved. We discuss integration of existing point visibility algorithms into our system in section 6.4, and present results in section 6.5. Section 6.6 gives a detailed discussion of some important aspects of the system.

6.2 Overview

We introduce a new visibility system that allows calculating visibility in parallel to the traditional rendering pipeline. The idea is to calculate a visibility solution (PVS) which is valid for several frames, depending on a maximum observer movement speed. This is achieved by using a point visibility algorithm and shrinking the occluders so as to make the resulting visibility solution valid for a certain ε-neighborhood around the viewpoint from which the visibility solution is calculated.

The algorithm consists of a short preprocessing phase and an online phase. The following parameters have to be determined in advance:

- Choose a point visibility algorithm.
- Decide how much time to allot for the visibility solution.
- Set a maximum observer movement speed.

In the preprocessing phase, occluders are generated for the scene and shrunk by an amount determined through the maximum movement speed and the time allotted for the visibility solution. For the online phase, two computing resources are needed:

- one resource to render and display frames with the current PVS
- a second resource to calculate the PVS for the next set of frames

6.3 Instant visibility

6.3.1 The traditional pipeline

The traditional rendering pipeline consists of several steps, where each step depends on the outcome of the previous step. Roughly, we identify two important steps of the pipeline for our discussion:

- $Vis(P)$ determines which objects are visible from an observer position P.
- $Draw(P)$ draws (traverses, transforms, rasterizes) the objects identified as visible as seen from the observer at position P.

These two steps communicate via the

- $PVS(P)$, potentially visible set, i.e., the set of objects determined to be potentially visible for a position P.

Efforts to parallelize $Vis(P)$ and $Draw(P)$ (for example, in a manner consistent with the multiprocessing mode of a popular rendering toolkit [Ecke00]) suffer from latency, since $Draw(P)$ requires the result of $Vis(P)$ to be able to operate (Figure 6.1 left).

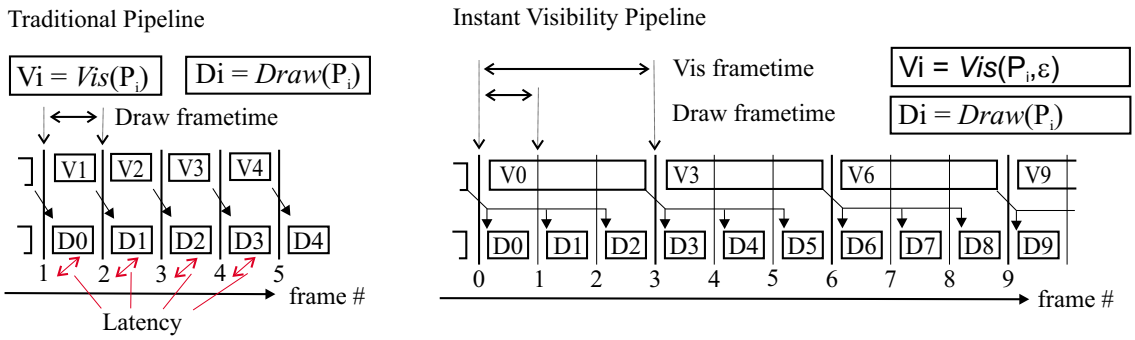


Figure 6.1: (Left) Parallelization in the traditional pipeline introduces latency and limits the time available for visibility. The figure shows where additional latency is introduced. (Right) The new *Instant Visibility* pipeline. Vis can take several frame times, the arrows show for which frames the resulting PVS is used.

6.3.2 Extending PVS validity

In chapter 5 we showed that the result of any point visibility algorithm can be made valid for an ε-neighborhood around the viewpoint by occluder shrinking. If all occluders in the scene are shrunk by an amount of ε, the pipeline step $Vis(P)$ actually computes

$$PVS_\varepsilon(P)$$

the set of objects potentially visible from either the point P or any point Q with $\|P - Q\| < \varepsilon$. $PVS_\varepsilon(P)$ can also be characterized by

$$PVS(P) \subseteq PVS_\varepsilon(P) \; \forall Q : \|P - Q\| < \varepsilon$$

We observe that the result of $Vis(P_0)$ is still valid during the computation of $Vis(P_1)$, as long as the observer does not leave an ε-neighborhood around P_0.

6.3.3 Parallel execution

We exploit the above observation to remove Vis from the pipeline and instead execute it in parallel to the pipeline (Figure 6.1 right). $Vis(P)$ might even take longer than a typical frame to execute - as long as the observer doesn't move too far away from P. More precisely, it is easy to show the following

Lemma 1 *Assume a frame time of t_{frame}. Assume also that $Vis(P)$ takes at most a time of t_{vis} to compute, where t_{vis} is a multiple of t_{frame}, and $Vis(P)$ always starts at a frame boundary. Then the time t_ε for which the visibility solution $PVS_\varepsilon(P)$ computed by $Vis(P)$ has to be valid so as to allow parallel execution of Vis and $Draw$ can be calculated as*

$$t_\varepsilon = 2t_{vis} - t_{frame}$$

Proof 2 *$Vis(P_i)$ takes t_{vis} to calculate. The result has to be valid till the result from $Vis(P_{i+1})$ is available, which takes again t_{vis}. But, the last frame where $PVS_\varepsilon(P_i)$ is valid displays an image for a viewpoint at the start of the frame. During the time period needed to render this last frame, no visibility solution is actually needed. So, we have $t_\varepsilon = t_{vis} + t_{vis} - t_{frame}$.*

Given a maximum observer speed v_{max}, the amount ε by which to shrink occluders can now be readily calculated as

$$\varepsilon = t_\varepsilon v_{max}$$

If the visibility solution does not take longer to compute than a typical frame (i.e., $t_{vis} = t_{frame}$), this means that $t_\varepsilon = t_{frame}$ and ε identifies the amount of space the observer can cover in one frame.

The algorithm described here effectively allows near-asynchronous execution of Vis and $Draw$. This makes it possible to achieve high frame rates even if the used visibility algorithm is not fast enough to complete in one frame time. In a typical scenario, the point visibility algorithm can provide results at a rate of at least 20 Hz, and the screen update rate is 60 Hz. Then PVS_ε has to be valid for the distance ε the observer can go in 5 frames. Assuming a maximum observer speed of 130 km/h, ε would be 3 m.

6.3.4 Networking

Executing visibility in parallel to rendering requires an additional computing resource. If the point visibility algorithm does not need access to graphics hardware itself, it can run on a separate processor. In case the point visibility algorithm does need access to graphics hardware, multichannel architectures allow the system to run on one machine.

The real strength of the method, however, lies in its inherent networking ability. The low network latency of today's local area networks allows a second machine to be used as a visibility server. At the start of a visibility frame, the current viewpoint is passed to the visibility server. After visibility calculations, the PVS is transmitted back to the graphics workstation.

A PVS typically contains object identifiers of a spatial scene data structure, so the size of a PVS depends on the granularity of this data structure and the amount of occlusion present. It should be noted that PVS-data is well compressible; even standard entropy coding achieves a compression ratio of up to 3:1. To give a practical example, passing a PVS containing 4,000 32bit object identifiers on a 100MBit-network takes about 1ms after compression.

Running the point visibility algorithm on a second machine also has the advantage that access to graphics hardware is automatically available, provided the visibility server is equipped with a good graphics card.

6.3.5 Synchronization

Running the visibility step in parallel to the rendering step requires synchronizing the two. In particular, it is crucial to deal with situations where visibility takes longer than t_{vis} to calculate, because the rendering step cannot continue without a potentially visible set. We list several ways to cope with this problem and discuss their applicability.

The preferred strategy depends strongly on the point visibility algorithm chosen. Many such algorithms consist of two steps: creating an occlusion data structure using a set of occluders, and testing the scene against this occlusion data structure. We assume that the number of occluders to draw determines the running time of visibility, and that the time necessary to test the scene against the occlusion data structure can be determined in advance.

guaranteed visibility Use a point visibility algorithm that has an inherent bound on its running time.

abort visibility Consider only so many occluders that visibility can execute in t_{vis}.

predictive occluder scheduling Determine in advance which occluders to use so that visibility can execute in t_{vis} and best possible occlusion is achieved. If occluder levels of detail are available, they can be incorporated in a similar fashion to Funkhouser's predictive level of detail scheduling [Funk93].

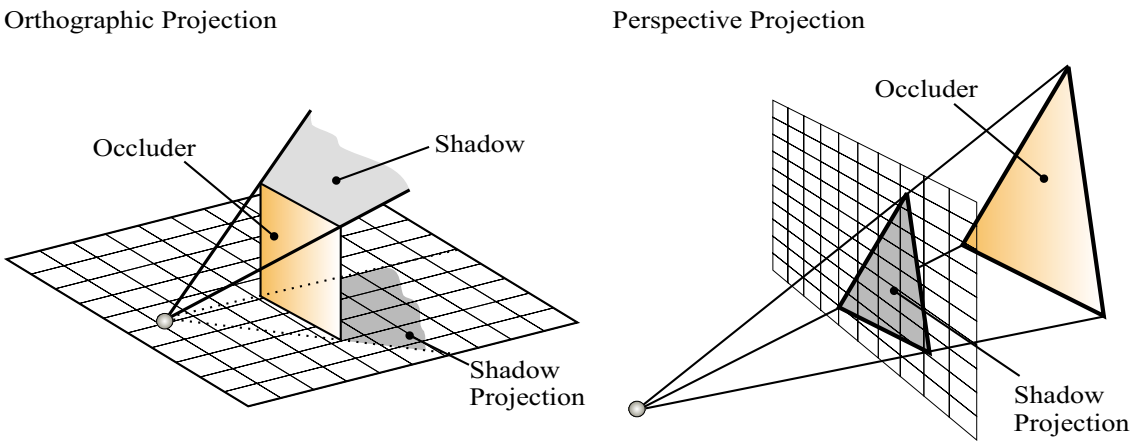


Figure 6.2: (Left) An orthographic occlusion map. The shadow cast by an occluder is projected orthographically into the cull map, which stores the depth values of the highest shadow. (Right) A perspective occlusion map. Occluders are projected onto a plane defined by the current camera parameters.

The next two possibilities are intended as fallback-strategies rather than solutions of their own, in case the other strategies fail, or small errors are not an issue and the chosen visibility algorithm executes fast enough in the majority of cases. They are implemented in $Draw$ instead of Vis.

stall Stall movement to prevent the observer from leaving the ε-neighborhood as long as there is no visibility solution available. This always guarantees correct images.

incomplete visibility Let the observer leave the ε-neighborhood, but still use $PVS_{\varepsilon}(P)$. Errors in visibility might occur, but observer speed is continuous and unhampered.

6.4 Integration with current occlusion culling algorithms

In this section we discuss important issues when integrating a point occlusion algorithm into the system.

6.4.1 Choice of projection

Image-based occlusion culling algorithms rely on projecting occluders and scene objects to a common projection plane.

For the application in urban environments, we propose rendering the shadows cast by occluders into an *orthographic* top-down view of the scene, the cull map (Figure 6.2 left and chapter 4). Although the approach is limited to 2.5D scenes, it has two advantages:

- Shrinking the occluders also guarantees conservative rasterization of occluder shadows into the cull map.
- It is very easy to calculate occlusion for a whole $360°$ panorama.

Other common approaches like the hierarchical z-buffer or the hierarchical occlusion maps are based on a *perspective* projection of the occluders and the scene into the current viewing frustum (Figure 6.2 right). The advantage is that arbitrary 3D scenes can be processed. However, visibility is only calculated for the current viewing frustum. Therefore, the viewing frustum (a pyramid) for occlusion culling has to be carefully chosen so as to include all parts of the scene that could be viewed within a time of t_{ε}.

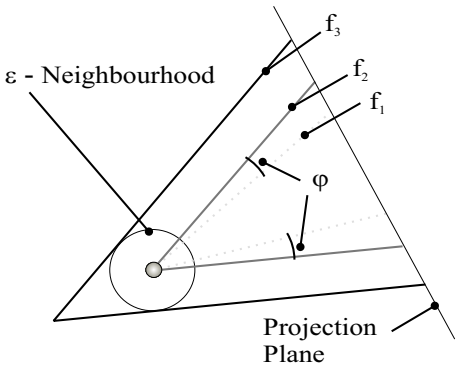


Figure 6.3: Adjusting the frustum for perspective projections. The original frustum f_1 is made wider by an angle of φ. The resulting frustum f_2 is then moved back to yield a frustum f_3 tight on the ε-neighborhood, to accommodate rotation and movement during the time t_ε.

To make this at all possible, a maximum turning speed ω_ε has to be imposed. The viewing pyramid is then made wider at all sides by the angle φ the observer can turn in t_ε, given by $\varphi = t_\varepsilon \omega_\varepsilon$. Finally, the pyramid is moved backwards until it fits tightly on an ε-sphere centered on the viewpoint, to account for backward and sideward movement (Figure 6.3).

As a simple example, assume visibility and screen update rates of 20 Hz and 60 Hz respectively, as above. If $\omega_\varepsilon = 180\frac{°}{s}$ (e.g., relatively fast head movement), the pyramid would have to be made wider by $15°$ on each side. This would enlarge a typical viewing frustum of $60°$ to $90°$.

6.4.2 Occluder selection policies

The execution speed of most current occlusion culling algorithms is mainly determined by the number of occluders they consider. It is thus desirable not to render all occluders for every viewpoint. Typically, a heuristic based on the projected area is used to select a number of relevant occluders. Note that since only visible occluders contribute to occlusion, finding all relevant occluders boils down to solving the visibility problem itself - it is therefore not possible to find exactly all relevant occluders.

We present and discuss 3 different approaches to occluder selection. In the discussion, an occlusion data structure is a perspective or orthographic projection of occluders.

conservative Always render all occluders. Although slow, this approach has the advantage that the time required for calculating visibility can be estimated or even bounded, which is important when determining the number of frames to allot for visibility calculation.

temporal coherence Use the set of occluders visible in the previous visibility step. Since new important occluders can appear even when moving only a small distance, occlusion will not be perfect. This approach is useful in scenarios where visibility takes quite long to compute, but the rendering step is not saturated. It essentially moves load from Vis to $Draw$.

2-pass In a first pass, create an occlusion data structure with the occluders visible in the previous step as above. But instead of the scene, test the occluders against this data structure for visibility. Occluders found visible are used to create a new occlusion data structure in a second pass. The scene is then tested against this new occlusion data structure for visibility.

Like the conservative approach, this approach computes the correct PVS, and like the temporally coherent approach, it reduces the number of occluders to be used, at the expense of having to build an occlusion data structure twice. It is best used when rendering all occluders is expensive compared to testing objects against the occlusion data structure.

6.4.3 Occluder shrinking

Theoretically, *Instant Visibility* works with any point visibility algorithm. In practice, there are restrictions on the type of occluders that can be used. In order to shrink occluders in 3D, they must be of volumetric nature. Furthermore, typical occluding objects should be large enough to ensure that shrunk occluders provide sufficient occlusion. See chapter 7 for details about occluder shrinking.

6.5 Implementation and results

We have implemented an *Instant Visibility* system and show its applicability on two different test scenes.

The first scene (Figure 6.4) demonstrates a walkthrough of an urban environment (2 km x 2 km) consisting of about 1.9 million polygons. An orthographic projection was used for the point visibility algorithm. For each visibility calculation we considered all occluders in the scene. 1,483 occluders (Figure 6.6) were generated from the building footprints and shrunk with the 2D algorithm described in chapter 7. Shrinking the occluder takes no more than 4 seconds.

The second scene (Figure 6.5) shows a flyover of a mountain range (4 km x 4 km) populated with trees. The size of the complete database is 872,000 polygons. Orthographic projection was used for point visibility, and 4,470 occluders (Figure 6.7) were generated from the terrain grid.

We recorded a path through each environment and compared the frame times for rendering without occlusion, rendering using region visibility, rendering using *Instant Visibility* and the pure visibility calculation times (Tab. 6.1). For *Instant Visibility*, we set the rendering frame time to 16 ms and the visibility frame time to 32 ms (= two rendering frames) for both experiments. Head rotation was not restricted, so we calculated occlusion for a full 360° panorama. Viewer movement was restricted to 108 km/h for the first scene (which gives $\varepsilon = 1.5$ m) and 720 km/h for the second scene (which gives $\varepsilon = 10$ m).

urban walkthrough				
method	avg	min	max	std dev.
VFC	207.3	57.6	733.4	131.9
region visibility	8.7	3.1	15.7	2.8
Instant Visibility	7.4	2.7	12.9	2.1
visibility time	19.0	17.6	20.9	0.4
terrain flyover				
method	avg	min	max	std dev.
VFC	10.8	2.6	23.0	5.2
region visibility	6.2	1.0	14.2	3.3
Instant Visibility	5.5	1.0	13.6	3.3
visibility time	19.1	23.5	17.6	0.9

Table 6.1: The table shows the measured times in milliseconds for the two test scenes. We measured rendering times for view-frustum culling (VFC), region visibility and *Instant Visibility*. The last row shows the time required for the visibility calculations for *Instant Visibility*. Note that the rendering times for *Instant Visibility* always stay below 16 ms. This is necessary for a 60 Hz simulation.

We want to point out the following observations:

- *Instant Visibility* is slightly better than region visibility.
- The frame rate is always below 16 ms for *Instant Visibility*. To give a real guarantee for the frame rate, LOD switching [Funk93] would have to be implemented. For our results, we only used distance based LODs for the trees in the terrain flyover.

Figure 6.4: This figure shows an orthophoto of the urban environment. The area is located near the center of Vienna and is about 2 km x 2 km wide.

- The times for visibility are well below 33 ms. Although the standard deviation is very small, a real guarantee would require a very careful analysis of the code. The variation is mainly due to the code that tests the scene against the occlusion map.

We additionally measured the latency and throughput of our 100 MB/s network by sending PVS data from the server to the client. We measured 0.741 ms on average for the latency (min: 0.605 / max: 0.898) and 1.186 ms for a PVS of 8 KB (min: 0.946 / max: 1.694). For our examples we did not need more than 4 KB of PVS data.

6.6 Discussion

We have shown that the *Instant Visibility* system is able to handle different types of scenes while maintaining high frame rates. An important aspect of a walkthrough system is to make reasonable assumptions about observer movement. Although it might be true that walking speed is limited to several km/h in the real world, it is doubtful that imposing such speed limits would benefit the behavior of a typical user of a walkthrough system. Mouse navigation in particular allows the user to change location quickly and to rapidly explore different sections of the environment. We have observed peak speeds of about 300 km/h in the urban scene, and up to 3,000 km/h in the terrain scene. The actual limits to be chosen depend strongly on the type of application (100 km/h might be a reasonable limit for an urban car simulation, for example) and user behavior.

Another point to note is that the *Instant Visibility* system solves the visibility problem by providing a PVS for each frame, but this PVS might still be too complex to be rendered interactively on the graphics workstation. Level-of-detail approaches and image-based rendering methods should be used to further reduce the rendering load. Funkhouser's predictive level-of-detail scheduling [Funk93] provides a way to maintain the desired frame rate during a walkthrough (in this method, levels of detail are selected before

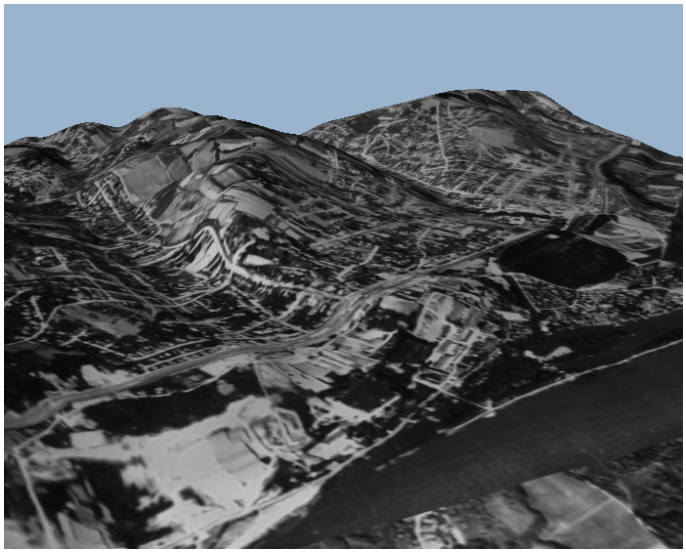


Figure 6.5: This figure shows an overview of the terrain. The model covers 4 km x 4 km of the city of Klosterneuburg, a smaller city in the north of Vienna.

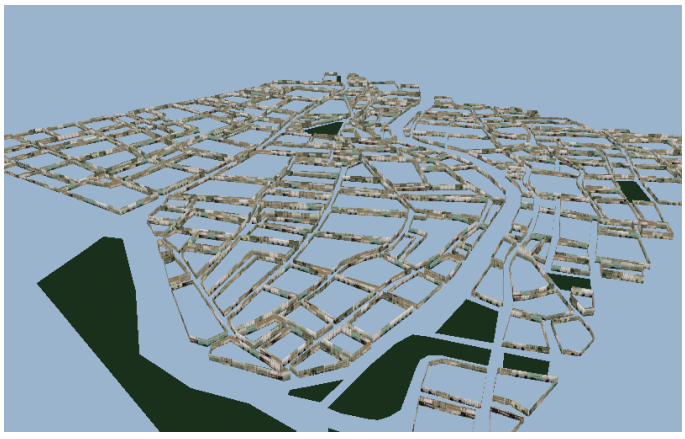


Figure 6.6: The figure shows the building fronts that are used as occluders in the urban walkthrough. The parks are shown as green textured polygons and are not used as occluders.

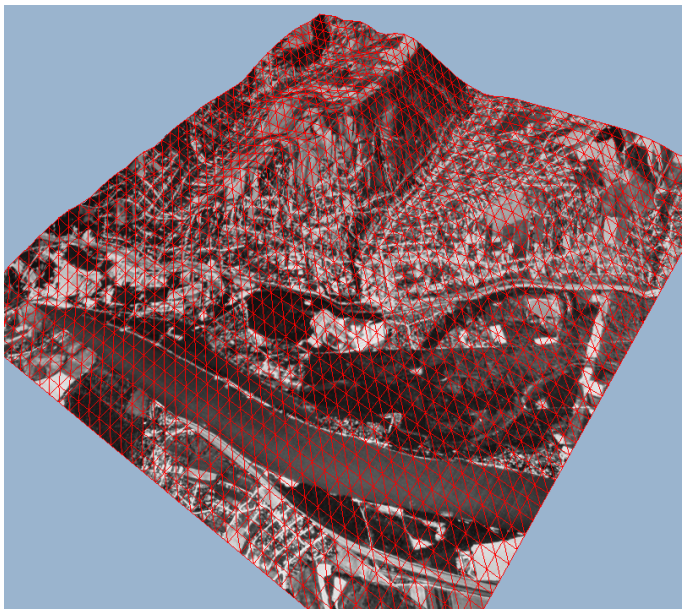


Figure 6.7: The occluders used for the terrain flyover are shown in red.

Figure 6.8: The figure shows a frame of the urban walkthrough. Note that the geometric complexity is high, as all windows are modeled as geometry.

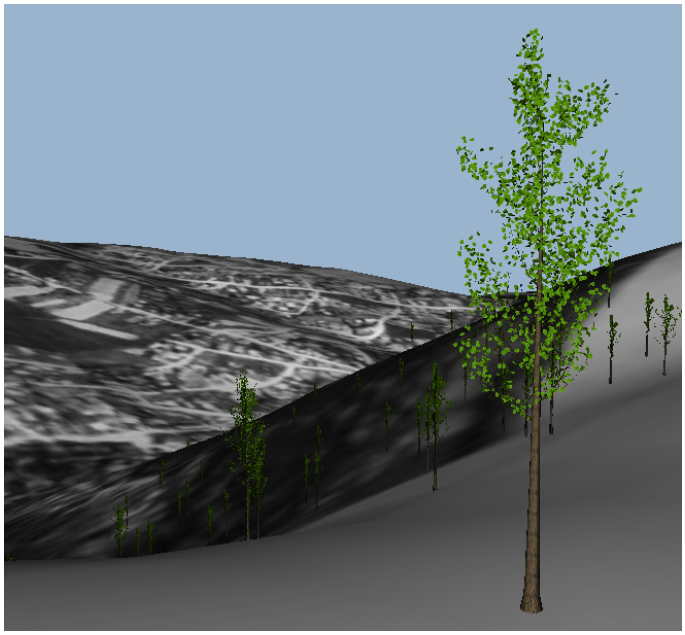


Figure 6.9: A typical frame of the terrain flyover. Note that a large portion of the scene is visible.

a frame is rendered so as to maximize the visual quality while bounding the rendering cost). Terrain rendering should benefit from a multiresolution algorithm.

We would also like to briefly skirt the problem of hard real-time systems. Although the *Instant Visibility* system tries to maintain a given frame rate, it is not a hard real-time system. It is in general hard to give accurate running times and upper bounds for any algorithm, and especially so for rendering and visibility algorithms which depend on graphics hardware and drivers. Therefore, the system can occasionally miss a frame, which we deem reasonable in view of the high costs involved in assuring hard real-time behavior.

The *Instant Visibility* system works very well if overall smooth system behavior and high frame rates are desired. The advantage over region visibility is that the time required for preprocessing is negligible, so that it is even possible to modify the scene during runtime. If occluders are shrunk separately, rigid transformations do not require any recalculation at all; in all other cases, calculations remain local to the area changed. This was one of the motivations that lead to the creation of the *Instant Visibility* system—we found that we rarely ever used region visibility because the needed PVS dataset was never available, and if it was, the scene had already changed, making the PVS unusable.

Another advantage over region visibility is that the view cells defined by an ε-neighborhood are usually smaller than typical view cells for region visibility, providing for better occlusion.

However, if its significant storage overhead and precalculation time are acceptable, region visibility offers the advantage that difficult visibility situations can be treated with special care. If the PVS is large, the visibility solution can be refined, and alternative representations for objects can be precalculated on a per-view cell basis. This is advantageous for shipping systems where the scene is not going to change, and which should not require more resources than a single machine.

Finally, we discuss the issue of latency and synchronization. The advantage of *Instant Visibility* over traditional pipeline systems is near-asynchronous execution of visibility and rendering, which is tightly coupled with a reduction in latency. In a traditional pipeline architecture [Ecke00, Alia99a], visibility and rendering have to be synchronized, so the rendering frame rate is tied to the time required for visibility (Figure 6.1). The latency from user input to the display of an image on the screen is therefore at least twice the time required for visibility (an image is always displayed at the end of a frame interval). In *Instant Visibility*, on the other hand, the time required for visibility only influences the maximum observer speed.

The graphics hardware can render at the highest possible frame rate, and the latency is always one frame interval. At the same time, visibility can be calculated more accurately because there is more time available for visibility and thus more occluders can be considered.

6.7 Conclusions

We have introduced *Instant Visibility*, a system to calculate visibility in real-time rendering applications. *Instant Visibility* combines the advantages of point and region visibility, in that it is based on a simple online visibility algorithm, while producing a PVS that remains valid for a sufficiently large region of space. We have demonstrated the behavior of the system in two real-time simulation applications with sustained refresh rates of 60 Hz. We believe that *Instant Visibility* is a valuable alternative to visibility preprocessing (see chapter 5) and we would strongly recommend the use of *Instant Visibility* whenever a second computer is available as a visibility server or in a multiprocessor system like the Onyx2.

Chapter 7

Occluder Synthesis and Occluder Shrinking

7.1 Occluder synthesis

Among the objects in the scene, candidates for good occluders have to be identified. An occluder should have good potential for occlusion - it should be fully opaque and large enough to possibly occlude other parts of the scene. Objects like trees, bushes, vehicles, or roads violate one or more of these requirements, which leaves mainly buildings and the terrain as useful occluders.

It is important to note that the algorithms described in this thesis basically rely on volumetric occluders. This restriction is not necessary but simplifies the calculations greatly. The proof presented in this chapter is also based on volumetric occluders. Even for the point visibility algorithm we shrink volumetric occluders, which allows all shrinking operations to be done at startup.

The second important fact is that good models are highly tesselated, which prevents us from using the geometric information directly. Occluders have to be synthesized from the input scene. This occluder synthesis is an open research problem that is not addressed in this thesis. Instead we make use of a priori knowledge about the scene structure. The input scenes for our tests are generated by a procedural modeling system (Figure 7.1). The input to the modeling system is a file describing the building footprints and the building heights. We simply use the extruded building footprints as volumetric occluders. We do not need to consider irregular features of the façade like doors, windows or balconies as long as we can guarantee that occluders are 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 [Laur94]). The footprint of the occluder may be (and usually is) concave.

An extruded building footprint is a volumetric occluder. However, the calculation of occluder shadows relies on polygonal occluders. Each edge of a building footprint defines such a polygonal occluder that corresponds to a façade in the real city (Figure 7.2).

Our current implementation handles arbitrary 2D-occluders and height fields stored as a regular grid (each grid point has a height value). However, the occluders from the terrain data were only used for the terrain flyover, because the city model we use is sufficiently dense so that the terrain occluders would not add additional occlusion. Although our implementation only handles the 2D and the 2.5D case, the *Occluder Shrinking* principle can be applied to full 3D, and the proof in the next section is valid for three-dimensional volumetric occluders.

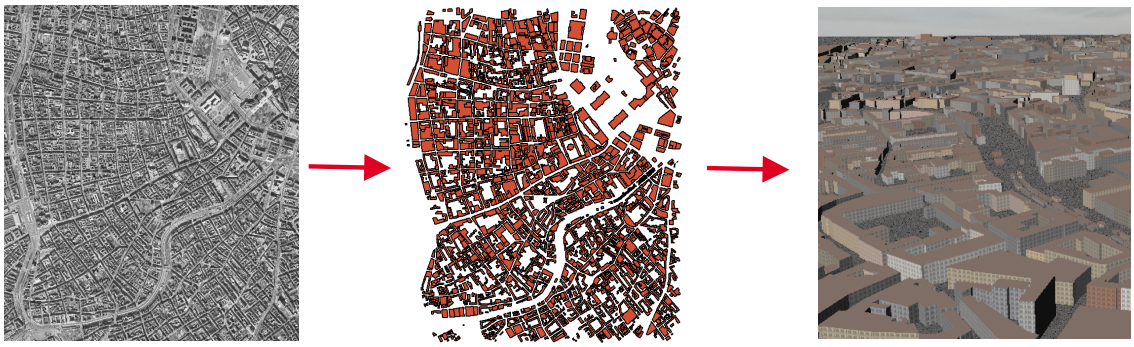


Figure 7.1: A simple overview of our modeling system. Starting from a satellite image, building footprints are digitized and heights are generated randomly. As a second step the building footprints are extruded automatically so that façade detail is added to the buildings.

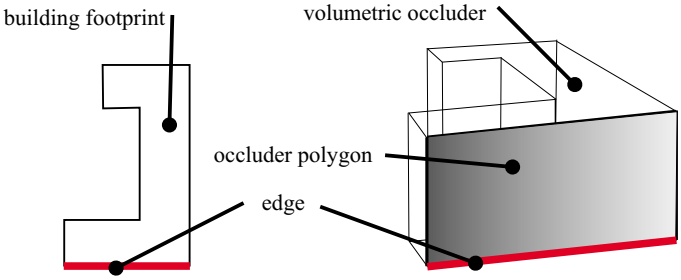


Figure 7.2: This figure shows the relation between volumetric and polygonal occluders. (Left) A building footprint. (Right) The extruded building footprint defines a volumetric occluder. One edge of the building footprint corresponds to an occluder polygon.

7.2 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 $A \in O' : |B - A| \leq \varepsilon \rightarrow B \in O$.

Theorem 3 *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 7.3).*

Proof 4 *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 $|Q' - Q| \leq e$. 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.*

Corollary 5 *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 $\varepsilon \geq \frac{k\sqrt{2}}{2}$, where k is the length of one cell of the cull map, as $\frac{k\sqrt{2}}{2}$ is exactly the maximum distance of two parallel lines through the same pixel where one of them passes through the pixel center (Figure 7.4).*

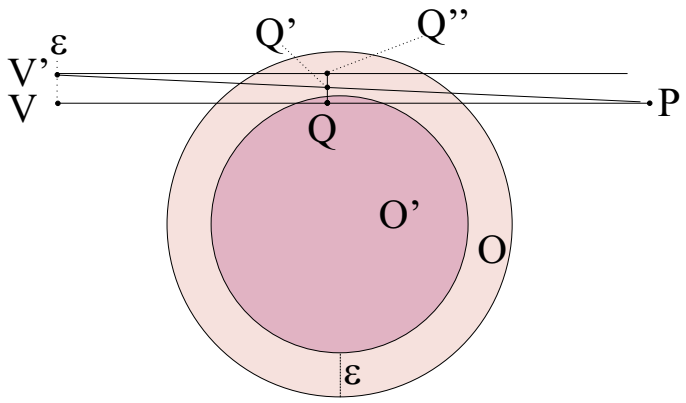


Figure 7.3: Any point P hidden by a shrunk occluder O' from V is also hidden by the original occluder O from any point V' in an ε-neighborhood of V.

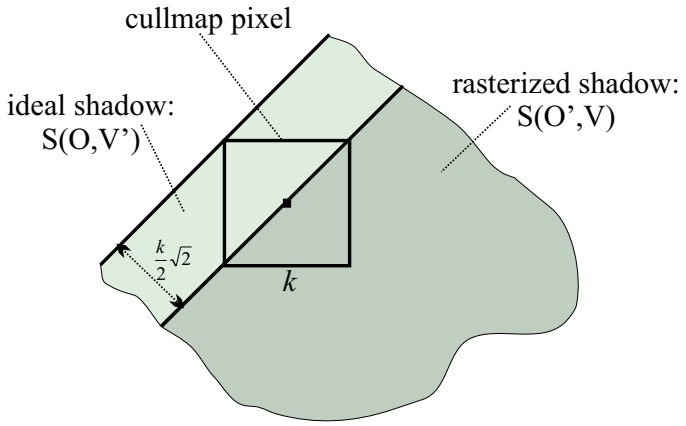


Figure 7.4: A pixel together with an actually rendered shadow polygon $S(O', V)$ (shaded dark) rendered from sample point V for a shrunk occluder O', and the additional part of an ideal shadow polygon $S(O, V')$ (shaded light) as it would be rendered by infinitely precise rendering hardware from some 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 $\varepsilon \geq \frac{k\sqrt{2}}{2}$, all of the pixel is actually covered by $S(O, V')$ and rasterization is correct.

Note that it can be shown that conservative results can already be had if $\varepsilon \geq \frac{k}{2}$, but a proof for this is more involved.

7.3 Implementation of occluder shrinking

7.3.1 Occluder shrinking in 3D

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, 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.

If volumetric objects are given as polyhedra we can shrink occluders as follows: Calculate a tetrahedal-

ization for the exterior of the occluder, enlarge each tetrahedra by ε, and intersect the occluder with the union of all enlarged tetrahedra. Polyhedral set operations have been explored for CSG by applying the operators directly [Laid86] or via BSP-subdivision [Nayl90].

Purely geometrical approaches are sometimes prone to numerical robustness and efficiency issues. In an alternative approach based on discretization, Schaufler et al. [Scha00] have shown how to represent the opaque portion of a general scene via an octree, provided the model is *water-tight*. The advantage of using an octree to represent occluders is that the set operations required for shrinking are trivial to implement. After shrinking the octree, it is converted to polyhedral form. To speed up occluder rendering, it is advisable to simplify the occluder mesh using a conservative level of detail algorithm [Law99].

7.3.2 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 (Terrain is usually given in a triangulated form). 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.

7.3.3 Occluders shrinking in 2D

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.

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 (Figure 7.5).

7.4 Advanced occluder shrinking

This section describes a restricted version of occluder shrinking that is mainly useful for *Instant Visibility*. Up to this point we have used occluder shrinking based on ε-neighborhoods. It stands to reason, however, that the observer is usually limited more in some directions than in others, and therefore an isotropic ε-neighborhood is not flexible enough. In most applications, movement in the up/down-direction will occur less frequently and with less speed than movement in the plane.

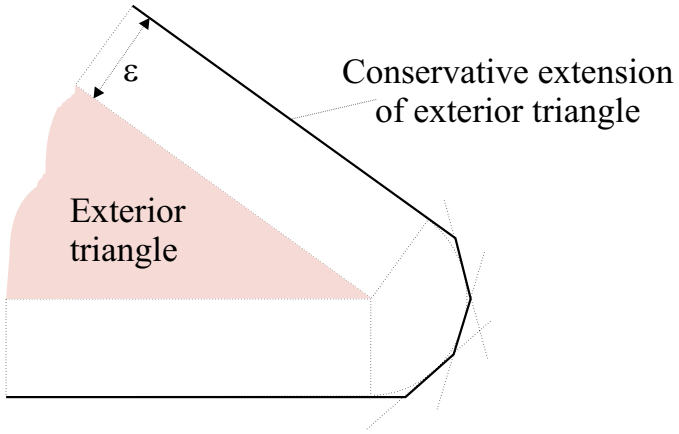


Figure 7.5: To extend a triangle by ε, the arc at the corner is approximated by 3 tangential line segments.

We therefore introduce a more formal definition of occluder shrinking based on Minkowski-operators, which allows handling anisotropic neighborhoods. The Minkowski-subtraction of two sets A and B is defined as

$$A \ominus B := \bigcap_{b \in B} \{a + b | a \in A\}$$

The *erosion* of a set A by a set B is defined as

$$E(A, B) := A \ominus (-B)$$

Intuitively, erosion can be interpreted as flipping the shape B around its origin and using this to carve out A, with the origin of B following the surface of A. We make use of erosion for occlusion via the following

Lemma 6 *Let an occluder O be an arbitrary subset of $\mathbf{R}^3$, and let V be the set of possible movement vectors from the current viewpoint vp (a not necessarily bounded subset of the vector space $\mathbf{R}^3$). Define a shrunk occluder O' via $O' := E(O, V)$. Then any point p occluded by O' seen from vp is also occluded by O when seen from any viewpoint $vp' \in VP$ (where $VP := vp + v, v \in V$).*

Proof 7 *Interpret occlusion as a shadow problem. Then the space of possible viewpoints VP can be interpreted as a volumetric light source. The question whether p is occluded as seen from all points of VP translates to the question whether p is in the umbra region of the light source VP. Erosion has previously [Luka98] been shown to provide an answer to this problem (the formulation in the previous chapter is based on Minkowski addition but can easily be shown to be equivalent to erosion).*

The practical implication is that occluder shrinking can be adapted to anisotropic observer movement. If, for example, movement is restricted to the ground plane, then objects only have to be shrunk in the two dimensions of the plane. An important optimization results for 2.5D environments: the region of possible observer movements can be approximated by a cylinder with radius ε and slightly elevated top. The elevation of the cylinder top is defined by how far the observer can move up and down in the time t_ε.

Implementation of advanced occluder shrinking can proceed exactly as in section 7.3. The new formulation immediately validates the 2D algorithm shown above for 2.5D urban environments, if occlusion is always calculated from the highest possible observer point. For the more general 3D algorithm, the only change is that exterior cells should be expanded by the vectors present in $-V$ instead of a constant ε in all directions.

Chapter 8

Conclusions and Future Work

The driving motivation for this research is real-time simulation of urban environments. For real-time simulation, high frame rates are crucial to give the impression of fluid motion. For large environments consisting of several million polygons a brute-force approach is no longer sufficient for fast rendering and therefore acceleration algorithms are necessary to handle the large amounts of geometric data. If we consider an observer who navigates through a city near ground level, it will be rapidly apparent that only a few buildings of the environment are actually visible. For our 8 km^2 test scene of Vienna, for example, we observed that the ratio of visible to invisible objects is typically smaller than $1 : 100$. Therefore, it is essential to identify objects that are definitely invisible because all rendering time spent on invisible objects is wasted. Invisible objects cannot contribute to the final image and hence do not need to be sent down the rendering-pipeline.

8.1 Synopsis

The research described in this thesis investigates methods of calculating occlusion culling for urban environments. General three-dimensional visibility is a complex problem. Therefore, we use the observation that visibility in urban environments is mainly determined by buildings so that the visibility problem can be simplified to 2.5D: occluders are height fields, i.e., functions $z = f(x, y)$. We propose three related algorithms to calculate occluded scene parts:

- The first algorithm calculates visibility from a viewpoint for every frame in a walkthrough.
- The second algorithm calculates visibility for a region and is used for visibility preprocessing.
- The third algorithm calculates visibility for a small ε-neighborhood. This visibility solution is valid for a few frames in a walkthrough and combines some of the advantages of preprocessing and online calculations.

For the design of the occlusion culling algorithm it is necessary to evaluate the algorithms in the context of a real-time rendering application. In the following we want to discuss how well our algorithms fulfill the demands of typical applications and how our algorithms relate to other aspects in the design of a complete system.

8.2 Frame rate control

We discovered that the importance of high frame rates is often underestimated in real-time rendering. Ideally, an urban walkthrough system should guarantee frame rates equal to the monitor refresh rate, which

is typically over 60 Hz. If the frame rate is lower than the monitor refresh rate, we observed that the resulting ghosting artifacts are strongly noticeable in our test models.

These high frame rates leave only a few milliseconds time for a single frame. Besides occlusion culling, many different tasks like database-traversal, simulation, user-input, and networking have to be done in that time span. On current hardware, this leaves too little time for our online visibility calculations. Therefore, visibility preprocessing is a better option for the applications we envision. Our results for the visibility preprocessing algorithm showed that these high frame rates are realistic even when consumer hardware is used. We also showed that it is possible to sustain high frame rates using the *Instant Visibility* system.

The important part of the frame rate control is the guarantee for a constant frame rate. For a carefully designed test scene, it is possible that the visible geometry can be rendered in the given time frame for all viewpoints. In a general scene, however, there will be several viewpoints where too much geometry is visible and further simplification is needed. The proposed *Instant Visibility* system could handle these frames with a level-of-detail selection strategy that gives a guarantee for the frame time [Funk93]. However, such a selection strategy does not ensure a high image quality and strongly noticeable artifacts can occur. Furthermore, the selection calculations themselves take additional time, the most precious resource in real-time simulation. With the use of the visibility preprocessing algorithm a more efficient simplification strategy can be used. Visibility preprocessing allows the identification of areas with many visible objects before runtime and simplification calculations can concentrate on the areas where they are really needed. In the existing system we use point-based representations, called *Point-based Impostors*, to simplify distant geometry [Wimm01]. However, an algorithm that can automatically place these *Point-based Impostors* and give a guarantee for a given frame rate and a certain image-quality is subject of ongoing research.

8.3 Visibility and modeling

We found that occlusion culling is related to modeling and scene structuring. For example, a “large” scene that consists of several million polygons can span several hundred square kilometers, but each building might only be modeled with a few large textured polygons. For each viewpoint, only a few polygons will be visible and the graphics hardware will not be pushed to its limit. Although such large models with little complexity might be interesting, we believe that it is important to work with models of higher visual complexity. Therefore, we tried to evaluate our algorithms using highly tesselated models, such that the potentially visible set for most view cells can be rendered at over 60Hz but not much faster. Figure 8.1 shows images of our current model that consists of over 10 million triangles for a 8 km^2 area of Vienna.

For such highly tesselated models it is not practical to store visibility on a per-triangle basis. First, the storage requirements for the potentially visible sets of all view cells would be prohibitively high. Secondly, efficient real-time rendering requires that the triangles are sent to the graphics hardware in groups using only a single API call. The triangles in a group have to share the same texture, material ...(see section 3.2.2). These groups cannot be formed efficiently during runtime and should thus be calculated in a preprocess.

In our system we used the following solution: we split the scene into objects that are subject to visibility classification. Each object consists of several triangles and is passed to an optimization program that performs mode sorting, triangle stripping and scene-graph flattening. For this process there is a clear tradeoff involved. On the one hand, larger groups of triangles allow for more efficient mode sorting and can be rendered with fewer API calls. On the other hand, smaller objects allow for more efficient visibility classification. We experimented with a few different object sizes for the splitting operation, but our implementation is too limited to present systematic results. A more detailed investigation of the relationship between scene structuring and visibility is a topic for future work.

8.4 Visibility and navigation

The main topic of this thesis was to investigate occlusion culling for walkthrough navigation. For this type of navigation the viewpoint is located close to the ground and we simulate the environment as it would be seen by a pedestrian or a car driver. For several applications it is also interesting to view the scene from above. Through flyover navigation we observed two things:

- The visibility calculations still work well. In several cases the calculations even become faster because the shadows cast by occluders become shorter so that we do not need to rasterize large polygons anymore.
- Many objects become visible. As discussed in the previous section the model is constructed with a geometric complexity so that the PVS can be rendered for a viewpoint near ground level. For our test scenes the PVS for an elevated position is easily 10–50 times larger than a typical PVS near ground level.

This means that the visibility calculations would still work but they do not solve the main problem any more. For walkthrough navigation we demonstrated that the design of a real-time rendering system is closely related to visibility. A system for flyover navigation would require fundamentally different design choices.

8.5 Visibility and dynamic objects

In a typical urban environment we can observe several moving objects, such as cars, buses and pedestrians. Many applications of urban visualization will also require including these objects in the simulation.

We did not consider moving objects in our results, but they can be handled by the proposed algorithms. The more important case is where only occludees are allowed to move and occluders have to remain static. We mainly consider buildings and terrain as occluders so that this is an adequate restriction for a typical application. The main part of the occlusion calculations does not need to be altered. For online visibility we have to introduce an additional step where we test moving objects against the cull map. For the visibility preprocessing algorithm, we recommend using the extension described by Durand et al. [Dura00, Suda96]: A hierarchy of bounding volumes is constructed in the regions of space in which dynamic objects can move. During preprocess, these bounding volumes are also tested for occlusion. During runtime a dynamic object is displayed, if the bounding box containing the dynamic object is in the current PVS.

However, for an urban modeler, for example, it is also necessary to deal with changing and moving occluders. For such an application it is only possible to use the online visibility or the *Instant Visibility* algorithm. A moving occluder cannot be handled by the proposed visibility preprocessing algorithm.

8.6 Visibility and shading

For many scenes the visibility problem can be stated as a geometric problem by treating the scene-triangles as static opaque surfaces. However, the occlusion effect of scene objects is closely related to rendering. Sometimes the occlusion properties of scene-objects can be changed at runtime through advanced shading calculations or rendering techniques. In the following we will discuss three examples where shading can affect the occlusion properties of the geometry:

First, newer graphic cards have the capability of vertex shading, that allows the altering of the positions of vertices when they are processed by the hardware. These changes correspond to a simple form of animation.

Secondly, pixels can be eliminated in the shading process. The most important example is alpha-texturing. A balcony or a fence, for example, can be modeled with a few large textured polygons. The

detailed geometry can be represented by a texture map, where several pixels are marked as transparent in the alpha channel of the texture. During rendering these pixels would be eliminated by the alpha test and more distant objects could be seen “through” the polygons. This problem can be even more involved when the elimination of pixels is obscured by a complex pixel shader.

Thirdly, shading can be used to achieve transparency effects by combining multiple triangles via alpha blending. A transparent surface like a window can therefore not be considered as an occluder.

Using our procedural modeling system we have full control over the model and therefore we can use the knowledge of those special cases for the occluder calculation process or avoid the generation of tricky configurations. However, the treatment of these special cases in arbitrary test scenes is more involved and leads to the problem of automatic occluder synthesis, a topic for future work.

8.7 Indirect visibility

Another different type of problem occurs with the use of rendering techniques, where objects can contribute to the final image that are not directly visible. Examples include the calculation of reflections in the windows of a building or a car and the casting of shadows. The incorporation of such rendering techniques and their effect on visibility calculations is an interesting topic for future research.

8.8 Visibility and simplification

As discussed above, a guaranteed high frame rate can only be achieved with the use of simplification algorithms, like levels of detail. For visibility preprocessing and occluder synthesis it is easier to use simplification strategies that do not change the occlusion properties of objects that are used as occluders. This should not be a strong limitation because for buildings most simplification algorithms would try to simplify the façade and the part of the visual hull which is used to form occluders would remain unchanged. However, if this cannot be guaranteed, the solution would strongly depend on the simplification that is used. For levels of detail, for example, it would be possible to use different versions of occluders that each corresponds to a range of levels of detail. A simpler solution would be to use only one occluder that is contained in the visual hull of all levels of detail. The problem of simplification leads again to the problem of occluder synthesis.

8.9 Occluder synthesis

In the presence of highly tesselated models it is not possible that scene polygons can be used as occluders any more. Since each building consists of several hundred polygons we cannot simply extract large polygons from the scene. Therefore, occluders have to be synthesized from the input scene. This problem was not treated systematically in this thesis, but we believe that the occluder synthesis during modeling is a useful method that could be applied in many existing systems. However, the automatic synthesis of occluders from general input scenes is probably the most important topic for future work.

Additionally, the occluder shrinking principle relies on volumetric occluders. Therefore, an occluder synthesis algorithm for volumetric occluders is needed.

8.10 Extension to 3D

Some ideas in this thesis are applicable to full 3D, but most parts are only useful for 2.5D visibility solutions:

- The occluder shrinking principle is applicable to full 3D. Therefore it is possible to calculate *Instant Visibility* for 3D scenes using a variation of the hierarchical z-buffer [Gree93] or the hierarchical occlusion maps [Zhan97].
- Visibility preprocessing using point sampling could also be combined with a 3D point visibility algorithm to calculate a full 3D solution. However, we expect a straightforward solution to be prohibitively slow.
- Occlusion culling using cull maps is inherently 2.5D and cannot be extended to three dimensions.

For current urban models we believe that the restriction to 2.5D is a useful simplification of the otherwise complex visibility problem. However, as we showed in our analysis of urban environments in chapter 2, visibility can be very complicated for many parts of a real city. For example, if vegetation plays a more important role in urban models this will require the use of more powerful visibility algorithms. It could be useful to use the proposed 2.5D algorithms as a first pass and try to eliminate more objects in a second pass with a full 3D algorithm.

8.11 The future

At the time of the start of this thesis visibility calculation was a hot research topic that needed to be solved to be able to build an urban simulation system. Due to the intense research in visibility for real-time rendering in the last three years, we believe that visibility is not the *main* problem any more, because the existing solutions are suitable for most existing urban models. However, if the scenes that can be modeled and rendered are becoming larger and more detailed, new interesting visibility questions will arise.

Personally, I believe that visibility will stay an interesting and important research problem in the near future and so I would like to finish with Calvin's words [Watt96]:

> *It's a magical world, Hobbes, ol' buddy . . . let's go exploring.*

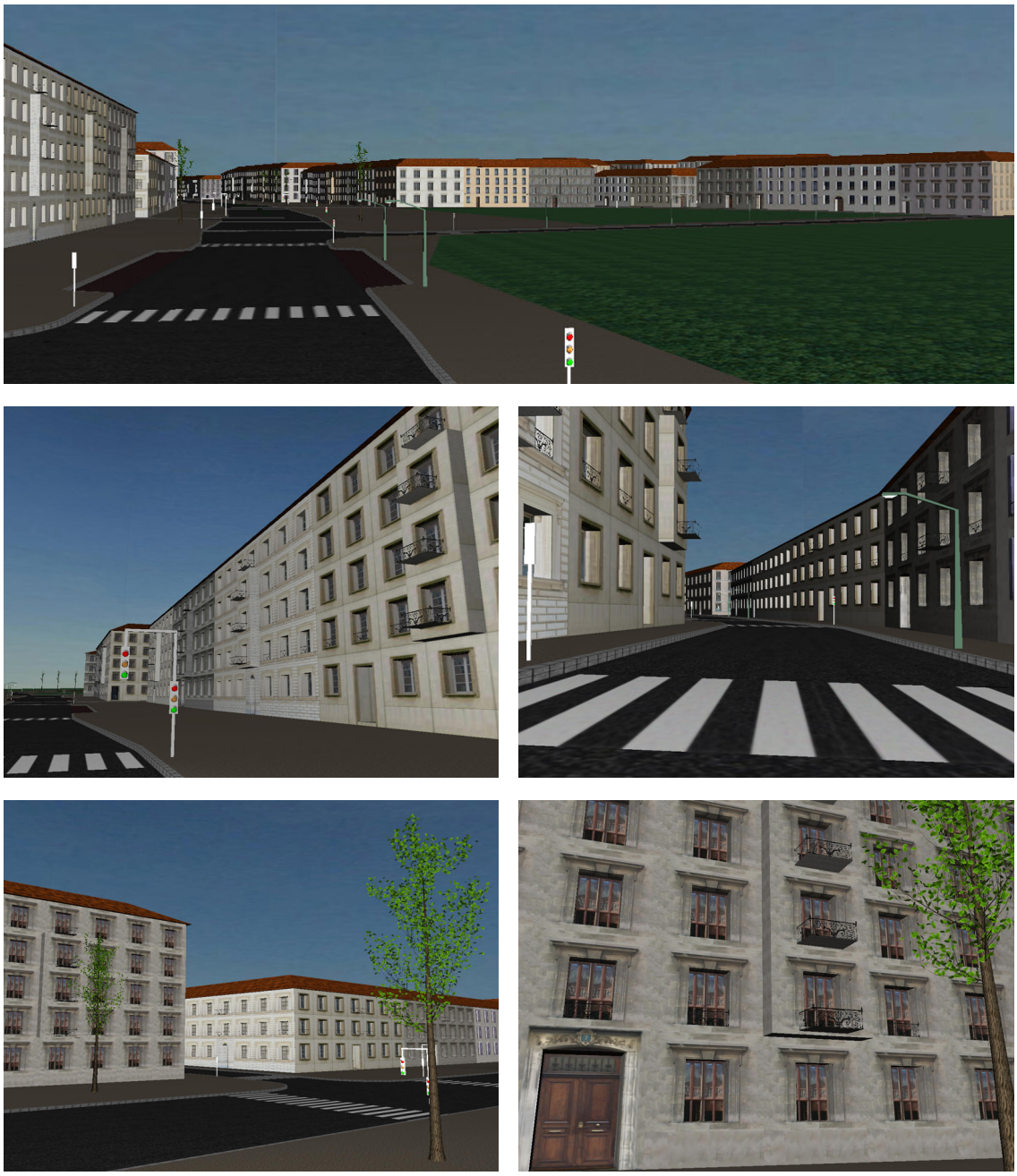


Figure 8.1: These images show the current city model.

Bibliography

[Adúj00a] C. Adújar, C. Saona-Vázquez, and I. Navazo. Lod visibility culling and occluder synthesis. *Computer-Aided Design*, 32(13):773–783, 2000. Cited on page 30.

[Adúj00b] C. Adújar, C. Saona-Vázquez, I. Navazo, and P. Brunet. Integrating Occlusion Culling with Levels of Detail through Hardly-Visible Sets. *Computer Graphics Forum (Proc. Eurographics '96)*, 19(3):499–506, August 2000. Cited on page 30.

[Aila01] Timo Aila and Ville Miettinen. SurRender Umbra – Reference Manual. http://www.surrender3d.com/umbra/index.html, 2001. Cited on pages 26 and 42.

[Aire90] John M. Airey, John H. Rohlf, and Frederick P. Brooks, Jr. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 41–50, March 1990. Cited on pages 28 and 29.

[Alia99a] Daniel Aliaga, Jon Cohen, Andrew Wilson, Eric Baker, Hansong Zhang, Carl Erikson, Keny Hoff, Tom Hudson, Wolfgang Stürzlinger, Rui Bastos, Mary Whitton, Fred Brooks, and Dinesh Manoclia. MMR: An Interactive Massive Model Rendering System Using Geometric and Image-Based Acceleration (Color Plate S. 237). In Stephen N. Spencer, editor, *1999 Symposium on interactive 3D Graphics*, pages 199–206. ACM SIGGRAPH, ACM Press, April 1999. Cited on pages 22, 30, 41, and 67.

[Alia99b] Daniel G. Aliaga and Anselmo Lastra. Automatic Image Placement to Provide a Guaranteed Frame Rate. In Alyn Rockwood, editor, *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 307–316. ACM SIGGRAPH, Addison Wesley, August 1999. Cited on page 22.

[Bart98] Dirk Bartz, Michael Meißner, and Tobias Hüttner. Extending Graphics Hardware for Occlusion Queries in OpenGL. In Stephen N. Spencer, editor, *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 97–104. ACM Press, August 31–September 1 1998. Cited on page 27.

[Bart99] Dirk Bartz, Michael Meißner, and Tobias Hüttner. OpenGL-assisted occlusion culling for large polygonal models. *Computers and Graphics*, 23(5):667–679, October 1999. Cited on page 27.

[Bern00] Fausto Bernardini, James T. Klosowski, and Jihad El-Sana. Directional Discretized Occluders for Accelerated Occlusion Culling. *Computer Graphics Forum (Proc. Eurographics '00)*, 19(3):507–516, August 2000. Cited on page 30.

[Bitt98] Jiri Bittner, Vlastimil Havran, and Pavel Slavík. Hierarchical Visibility Culling with Occlusion Trees. In Franz-Erich Wolter and Nicholas M. Patrikalakis, editors, *Proceedings of the Conference on Computer Graphics International 1998 (CGI-98)*, pages 207–219, Los Alamitos, California, June 22–26 1998. IEEE Computer Society. ISBN 0-8186-8445-3. Cited on pages 24, 36, and 42.

[Bitt01a] J. Bittner and J. Prikryl. Exact Regional Visibility using Line Space Partitioning. Technical Report TR-186-2-01-06, Institute of Computer Graphics and Algorithms, Vienna University of Technology, March 2001. Cited on page 29.

[Bitt01b] J. Bittner, P. Wonka, and M. Wimmer. Visibility Preprocessing for Urban Scenes using Line Space Subdivision. Submitted to Pacific Graphics 2001, 2001. Cited on page 29.

[Borm00] Karsten Bormann. An adaptive occlusion culling algorithm for use in large ves. In S. Feiner and D. Thalmann, editors, *IEEE Virtual Reality 2000 Proceedings*, page 290. IEEE Computer Society, March 18–22 2000. Cited on page 26.

[Bout00] Gary D. Bouton, Gary Kubicek, Barbara M. Bouton, and Mara Z. Nathanson. *Inside Adobe Photoshop 6*. 2000. Cited on page 19.

[Chai00] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic Sampling. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 307–318. ACM SIGGRAPH, Addison Wesley, 2000. Cited on page 22.

[Chan99] Chun-Fa Chang, Gary Bishop, and Anselmo Lastra. LDI Tree: A Hierarchical Representation for Image-Based Rendering. In Alyn Rockwood, editor, *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 291–298. ACM SIGGRAPH, Addison Wesley, August 1999. Cited on page 22.

[Clar76] James H. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19(10):547–554, October 1976. ISSN 0001-0782. Cited on page 23.

[Cohe96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification Envelopes. (Annual Conference Series):119–128, August 1996. Cited on page 30.

[Cohe98a] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-Preserving Simplification. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 115–122. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8. Cited on page 21.

[Cohe98b] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. *Computer Graphics Forum (Proc. Eurographics '98)*, 17(3):243–254, September 1998. ISSN 1067-7055. Cited on page 28.

[Coor97] Satyan Coorg and Seth Teller. Real-Time Occlusion Culling for Models with Large Occluders. In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 83–90. ACM SIGGRAPH, ACM Press, April 1997. Cited on pages 24, 36, 42, and 55.

[Coor99] Satyan Coorg and Seth Teller. Temporally Coherent Conservative Visibility. *Computational Geometry: Theory and Applications*, 12(1-2):105–124, February 1999. Cited on page 24.

[Dars97] Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney. Navigating Static Environments Using Image-Space Simplification and Morphing. In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 25–34. ACM SIGGRAPH, ACM Press, April 1997. ISBN 0-89791-884-3. Cited on page 22.

[Debe96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996. Cited on page 19.

[Debe98] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient View-Dependent Image-Based Rendering With Projective Texture-Mapping. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98 (Proceedings of the Eurographics Workshop 98)*, pages 105–116. Springer-Verlag Wien New York, June 1998. Cited on page 19.

[Deco99] Xavier Decoret, François Sillion, Gernot Schaufler, and Julie Dorsey. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum (Proc. Eurographics '99)*, 18(3):61–73, September 1999. ISSN 1067-7055. Cited on pages 22, 29, and 45.

[Deus98] Oliver Deussen, Patrick Hanrahan, Matt Pharr, Bernd Lintermann, Radomír Měch, and Przemyslaw Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 275–286. ACM SIGGRAPH, Addison Wesley, July 1998. Cited on page 19.

[Doni97a] Stéphane Donikian. Multilevel Modelling of Virtual Urban Environments for Behavioural Animation. In *Animation 97 Conference Proceedings*, pages 127–133, June 1997. Cited on page 19.

[Doni97b] Stéphane Donikian. VUEMS: A Virtual Urban Environment Modeling System. In *Proceedings of the Conference on Computer Graphics International 1997 (CGI-97)*, pages 84–92, 1997. Cited on page 19.

[Down01] L. Downs, T. Moeller, and C. H. Séquin. Occlusion Horizons for Driving through Urban Scenery. In *2001 Symposium on Interactive 3D Graphics*. ACM SIGGRAPH, ACM Press, 2001. Cited on pages 24 and 42.

[Dura96] Frédo Durand, George Drettakis, and Claude Puech. The 3D Visibility Complex: A New Approach to the Problems of Accurate Visibility. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96 (Proceedings of the Eurographics Workshop 96)*, pages 245–256. Eurographics, Springer-Verlag Wien New York, June 1996. Cited on page 23.

[Dura97] Frédo Durand, George Drettakis, and Claude Puech. The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 89–100. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. Cited on pages 23, 46, and 55.

[Dura99] Frédo Durand. *3D Visibility: Analytical Study and Applications*. PhD thesis, Université Joseph Fourier, Grenoble, France, July 1999. Cited on pages 23 and 46.

[Dura00] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative Visibility Preprocessing Using Extended Projections. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 239–248. ACM SIGGRAPH, Addison Wesley, July 2000. Cited on pages 28, 29, 54, and 76.

[Ecke98] G. Eckel, R. Kempf, and L. Wennerberg. *OpenGL Optimizer Programmer's Guide: An Open API for Large-Model Visualization*. SGI techpubs library, 1998. Cited on page 20.

[Ecke00] G. Eckel and K. Jones. *OpenGL Performer Programmer's Guide*. SGI techpubs library, 2000. Cited on pages 20, 21, 30, 58, and 67.

[Egge92] David W. Eggert, Kevin W. Bowyer, and Charles R. Dyer. Aspect Graphs: State-of-the-Art and Applications in Digital Photogrammetry. In *Proc. ISPRS 17th Cong.: International Archives Photogrammetry Remote Sensing*, pages 633–645, August 1992. Cited on page 23.

[Evan96] Francine Evans, Steven S. Skiena, and Amitabh Varshney. Optimizing Triangle Strips for Fast Rendering. In Roni Yagel and Gregory M. Nielson, editors, *Proceedings of the conference on Visualization '96*, pages 319–326. IEEE, October 1996. Cited on page 21.

[Funk93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. In James T. Kajiya, editor, *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 247–254. ACM SIGGRAPH, Addison Wesley, August 1993. Cited on pages 21, 60, 63, 64, and 75.

[Funk96] Thomas Funkhouser, Seth Teller, Carlo Sequin, and Delnaz Khorramabadi. The UC Berkeley System for Interactive Visualization of Large Architectural Models. *Presence, the Journal of Virtual Reality and Teleoperators*, 5(1):13–44, 1996. Cited on page 22.

[Garl97] Michael Garland and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. Cited on pages 21 and 30.

[Garl98] Michael Garland and Paul S. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of the conference on Visualization '98*, pages 263–270. IEEE, October 1998. ISBN 1-58113-106-2. Cited on page 21.

[Gerv96] Michael Gervautz and Christoph Traxler. Representation and realistic rendering of natural phenomena with cyclic CSG graphs. *The Visual Computer*, 12(2):62–71, 1996. ISSN 0178-2789. Cited on page 19.

[Gigu91] Ziv Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE PAMI*, 13(6):542–551, 1991. Cited on page 46.

[Gort96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The Lumigraph. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996. Cited on page 22.

[Gots99] Craig Gotsman, Oded Sudarsky, and Jeffrey A. Fayman. Optimized occlusion culling using five-dimensional subdivision. *Computers and Graphics*, 23(5):645–654, October 1999. Cited on page 29.

[Grap01] Graphisoft. Archicad product information, 2001. http://www.graphisoft.com/products. Cited on page 19.

[Gree93] Ned Greene and Michael Kass. Hierarchical Z-Buffer Visibility. In James T. Kajiya, editor, *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 231–238. ACM SIGGRAPH, Addison Wesley, August 1993. ISBN 0-201-51585-7. Cited on pages 25 and 78.

[Gree96] Ned Greene. Hierarchical Polygon Tiling with Coverage Masks. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 65–74. ACM SIGGRAPH, Addison Wesley, August 1996. Cited on page 25.

[Gree99a] Ned Greene. Efficient Occlusion Culling for Z-Buffer Systems. In Bob Werner, editor, *Proceedings of the Conference on Computer Graphics International 1999 (CGI-99)*, pages 78–79, Los Alamitos, California, June 7–11 1999. IEEE Computer Society. Cited on page 27.

[Gree99b] Ned Greene. Occlusion culling with optimized hierarchical buffering. In ACM, editor, *SIGGRAPH 99 Conference Proceedings. Conference abstracts and applications*, Annual Conference Series, pages 261–261. ACM SIGGRAPH, ACM Press, 1999. Cited on page 27.

[Gros98] J. P. Grossman and William J. Dally. Point Sample Rendering. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98 (Proceedings of the Eurographics Workshop 98)*, pages 181–192. Eurographics, Springer-Verlag Wien New York, June 1998. Cited on page 22.

[Helm94] James L. Helman. Architecture and Performance of Entertainment Systems, Appendix A. *ACM SIGGRAPH 94 Course Notes - Designing Real-Time Graphics for Entertainment*, 23:1.19–1.32, July 1994. Cited on pages 13 and 14.

[Hey01] Heinrich Hey, Robert Tobler, and Werner Purgathofer. Real-Time Occlusion Culling With A Lazy Occlusion Grid. Technical Report TR-186-2-01-02, Institute of Computer Graphics and Algorithms, Vienna University of Technology, January 2001. Cited on page 26.

[Hild95] A. Hildebrand, S. Müller, and R. Ziegler. REALISE: Computer-Vision basierte Modellierung für Virtual Reality. In D. W. Fellner, editor, *Modeling - Virtual Worlds - Distributed Graphics (Beiträge zum Internationalen Workshop MVD, 1995)*, pages 159–167, November 1995. Cited on page 19.

[Ho] Poon Chun Ho and Wenping Wang. Occlusion culling using minimum occluder set and opacity map. In *IEEE International Conference on Information Visualization, 1999*, pages 292–300, July 14–16. Cited on page 26.

[Hopp96] Hugues Hoppe. Progressive Meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. Cited on page 21.

[Hopp97] Hugues Hoppe. View-Dependent Refinement of Progressive Meshes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. Cited on page 21.

[Hopp99] Hugues Hoppe. Optimization of mesh locality for transparent vertex caching. In Alyn Rockwood, editor, *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 269–276. ACM SIGGRAPH, Addison Wesley, August 1999. Cited on page 21.

[Huds97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated Occlusion Culling using Shadow Frusta. In *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, pages 1–10. ACM Press, 4–6 June 1997. ISBN 0-89791-878-9. Cited on pages 24, 42, and 55.

[Inc.01] MultiGen Inc. MultiGen product information, 2001. http://www.multigen.com/. Cited on page 19.

[IVT01] IVT. IVT company website, 2001. http://www.ivt.fr/. Cited on page 19.

[Jeps95] William Jepson, Robin Liggett, and Scott Friedman. An Environment for Real-time Urban Simulation. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 165–166. ACM SIGGRAPH, April 1995. Cited on pages 19 and 21.

[Jeps96] William Jepson, Robin Liggett, and Scott Friedman. Virtual Modeling of Urban Environments. *PRESENCE*, 5(1):72–86, 1996. Cited on page 19.

[Joha98] Andreas Johannsen and Michael B. Carter. Clustered Backface Culling. *Journal of Graphics Tools: JGT*, 3(1):1–14, 1998. Cited on page 23.

[Klos99] James T. Klosowski and Claudio T. Silva. Rendering on a Budget: A Framework for Time-Critical Rendering. In David Ebert, Markus Gross, and Bernd Hamann, editors, *Proceedings of the 1999 Conference on Visualization '99*, pages 115–122, N.Y., October 1999. IEEE, ACM Press. Cited on page 30.

[Klos00] J. T. Klosowski and C. T. Silva. The Prioritized-Layered Projection Algorithm for Visible Set Estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, April/June 2000. Cited on page 30.

[Kofl98a] Michael Kofler. *R-trees for Visualizing and Organizing Large 3D GIS Databases*. PhD thesis, Graz University of Technology, Graz, Austria, July 1998. Cited on page 22.

[Kofl98b] Michael Kofler, Michael Gervautz, and Michael Gruber. The Styria Flyover: LoD Management for Huge Textured Terrain Models. In Franz-Erich Wolter and Nicholas M. Patrikalakis, editors, *Proceedings of the Conference on Computer Graphics International 1998 (CGI-98)*, pages 444–454. IEEE Computer Society, June 1998. Cited on page 22.

[Kolt00] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. Virtual Occluders: An efficient Intermediate PVS representation. In Bernard Péroche and Holly Rushmeier, editors, *Rendering Techniques 2000 (Proceedings of the Eurographics Workshop 2000)*, Eurographics, pages 59–70. Springer-Verlag Wien New York, June 2000. Cited on page 30.

[Kova00] P. J. Kovach. *Inside Direct3d*. Microsoft Press, 2000. Cited on page 20.

[Kuma96a] Subodh Kumar and Dinesh Manocha. Hierarcical Visibility Culling for Spline Models. In Wayne A. Davis and Richard Bartels, editors, *Proceedings of Graphics Interface '96*, pages 142–150. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1996. Cited on page 24.

[Kuma96b] Subodh Kumar, Dinesh Manocha, William Garrett, and Ming Lin. Hierarchical Back-Face Computation. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96 (Proceedings of the Eurographics Workshop 96)*, pages 235–244. Eurographics, Springer-Verlag Wien New York, June 1996. Cited on page 23.

[Laid86] David H. Laidlaw, W. Benjamin Trumbore, and John F. Hughes. Constructive Solid Geometry for Polyhedral Objects. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH 86 Proceedings)*, volume 20, pages 161–170. ACM SIGGRAPH, ACM Press, August 1986. Cited on page 72.

[Laur94] A. Laurentini. The Visual hull Concept for Silhouette-Based Image Understanding. *IEEE PAMI*, 16(2):150–162, February 1994. Cited on page 69.

[Law99] Fei-Ah Law and Tiow-Seng Tan. Preprocessing Occlusion For Real-Time Selective Refinement (Color Plate S. 221). In Stephen N. Spencer, editor, *1999 Symposium on interactive 3D Graphics*, pages 47–54, New York, April 26–28 1999. ACM SIGGRAPH, ACM Press. Cited on pages 30 and 72.

[Levo85] Marc Levoy and Turner Whitted. The Use of Points as Display Primitives. Technical Report TR85-022, Department of Computer Science, University of North Carolina - Chapel Hill, October 1 1985. Cited on page 22.

[Levo96] Marc Levoy and Pat Hanrahan. Light Field Rendering. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. Cited on page 22.

[Lim92] H. L. Lim. Toward a Fuzzy Hidden Surface Algorithm. In *Proceedings of the Conference on Computer Graphics International 1992 (CGI-92)*, pages 621–635, 1992. Cited on page 29.

[Lind97] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, and N. Faust. An Integrated Global GIS and Visual Simulation System. Technical Report Report GIT-GVU-97-07, GVU Center Georgia Institute of Technology, 1997. Cited on page 22.

[Lisc98] Dani Lischinski and Ari Rappoport. Image-Based Rendering for Non-Diffuse Synthetic Scenes. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98 (Proceedings of the Eurographics Workshop 98)*, pages 301–314. Eurographics, Springer-Verlag Wien New York, June 1998. Cited on page 22.

[Lueb95] David P. Luebke and Chris Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, ACM Press, April 1995. ISBN 0-89791-736-7. Cited on page 28.

[Lueb97] David Luebke and Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 199–208. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. Cited on page 21.

[Luka98] A. Lukaszewski and A. Formella. Fast Penumbra Calculation in Ray Tracing. In Vaclav Skala, editor, *Proceedings of WSCG'98, the 6th International Conference in Central Europe on Computer Graphics and Visualization '98*, pages 238–245. University of West Bohemia, 1998. Cited on page 73.

[Maci95] Paulo W. C. Maciel and Peter Shirley. Visual Navigation of Large Environments Using Textured Clusters. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 95–102. ACM SIGGRAPH, ACM Press, April 1995. ISBN 0-89791-736-7. Cited on page 21.

[Maho97] Diana Phillips Mahoney. Philadelphia 2000. *Computer Graphics World*, 20(6):30–32, june 1997. Cited on page 19.

[Mano00] Dinesh Manocha. Introduction. *ACM SIGGRAPH '00 Course Notes - Interactive Walkthroughs of Large Geometric Datasets*, 18:1–25, July 2000. Cited on page 14.

[Mare97] M. Maresch. *Linear CCD array based recording of buildings for digital models*. PhD thesis, Graz University of Technology, Graz, Austria, 1997. Cited on page 20.

[Mark97] William R. Mark, Leonard McMillan, and Gary Bishop. Post-Rendering 3D Warping. In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM SIGGRAPH, ACM Press, April 1997. ISBN 0-89791-884-3. Cited on page 22.

[Max96] Nelson Max. Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-Buffers. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96 (Proceedings of the Eurographics Workshop 96)*, pages 165–174. Eurographics, Springer-Verlag Wien New York, June 1996. ISBN 3-211-82883-4. Cited on page 22.

[Měch96] Radomír Měch and Przemyslaw Prusinkiewicz. Visual Models of Plants Interacting With Their Environment. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 397–410. ACM SIGGRAPH, Addison Wesley, August 1996. Cited on page 19.

[Mene98] D. Meneveaux, K. Boauatouch, E. Maisel, and R. Delmont. A New Partitioning Method for Architectural Environments. *Journal of Visualization and Computer Animation*, 9(4):195–213, October/November 1998. Cited on page 28.

[Meye98] Alexandre Meyer and Fabrice Neyret. Interactive Volumetric Textures. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98 (Proceedings of the Eurographics Workshop 98)*, pages 157–168. Eurographics, Springer-Verlag Wien New York, June 1998. Cited on page 22.

[Micr01] Microsoft. Midtown Madness2 product information, 2001. http://www.microsoft.com/games/midtown2/. Cited on page 20.

[Mill98] Gavin Miller, Steven Rubin, and Dulce Ponceleon. Lazy Decompression of Surface Light Fields For Precomputed Global Illumination. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98 (Proceedings of the Eurographics Workshop 98)*, pages 281–292. Eurographics, Springer-Verlag Wien New York, June 1998. Cited on page 22.

[Möll99] Tomas Möller and Eric Haines. *Real-Time Rendering*. A. K. Peters Limited, 1999. Cited on pages 12, 14, and 21.

[More00] S. Morein. ATI Radeon Hyper-Z Technology, 2000. Commercial presentation at SIGGRAPH / Eurographics Workshop on Graphics Hardware. Cited on page 27.

[Nadl99] Boaz Nadler, Gadi Fibich, Shuly Lev-Yehudi, and Daniel Cohen-Or. A qualitative and quantitative visibility analysis in urban scenes. *Computers and Graphics*, 23(5):655–666, October 1999. Cited on page 28.

[Nayl90] Bruce Naylor, John Amanatides, and William Thibault. Merging BSP Trees Yields Polyhedral Set Operations. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH 92 Proceedings)*, volume 24, pages 115–124. ACM SIGGRAPH, ACM Press, August 1990. Cited on page 72.

[Pfis00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface Elements as Rendering Primitives. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 335–342. ACM SIGGRAPH, Addison Wesley, 2000. Cited on page 22.

[Plan90] Harry Plantinga and Charles R. Dyer. Visibility, Occlusion, and the Aspect Graph. *International Journal of Computer Vision*, 5(2):137–160, November 1990. ISSN 0920-5691. Cited on page 46.

[Pocc93] M. Pocchiola and G. Vegter. The visibility complex. In ACM-SIGACT ACM-SIGGRAPH, editor, *Proceedings of the 9th Annual ACM Symposium on Computational Geometry*, pages 328–337. ACM Press, May 1993. Cited on page 23.

[Prus91] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, 1991. Cited on page 19.

[Rohl94] John Rohlf and James Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real–Time 3D Graphics. In Andrew Glassner, editor, *SIGGRAPH 94 Conference Proceedings*, Computer Graphics Proceedings, Annual Conference Series, pages 381–395. ACM SIGGRAPH, ACM Press, July 1994. Cited on pages 20 and 21.

[Ross93] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications (Proc. Second Conference on Geometric Modelling in Computer Graphics*, pages 455–465, Berlin, June 1993. Springer-Verlag. Cited on page 21.

[Rusi00] Szymon Rusinkiewicz and Marc Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 343–352. ACM SIGGRAPH, Addison Wesley, July 2000. Cited on page 22.

[Saon99] C. Saona-Vázquez, I. Navazo, and P. Brunet. The visibility octree: a data structure for 3D navigation. *Computers and Graphics*, 23(5):635–643, October 1999. Cited on page 28.

[Scha96] Gernot Schaufler and Wolfgang Sturzlinger. A Three-Dimensional Image Cache for Virtual Reality. *Computer Graphics Forum (Proc. Eurographics '96)*, 15(3):227–235, September 1996. ISSN 0167-7055. Cited on page 21.

[Scha98] Gernot Schaufler. Per-Object Image Warping with Layered Impostors. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98 (Proceedings of the Eurographics Workshop 98)*, pages 145–156. Eurographics, Springer-Verlag Wien New York, June 1998. Cited on page 22.

[Scha00] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François X. Sillion. Conservative Volumetric Visibility with Occluder Fusion. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 229–238. ACM SIGGRAPH, Addison Wesley, July 2000. Cited on pages 29, 54, and 72.

[Schm97a] Dieter Schmalstieg. Lodestar: An Octree-Based Level Of Detail Generator for VRML. In Rikk Carey and Paul Strauss, editors, *VRML 97: Second Symposium on the Virtual Reality Modeling Language*. ACM SIGGRAPH / ACM SIGCOMM, ACM Press, February 1997. Cited on page 21.

[Schm97b] Dieter Schmalstieg and Gernot Schaufler. Smooth Levels of Detail. In *Proceedings of VRAIS'97*, pages 12–19, 1997. Cited on page 21.

[Scot98] N. D. Scott, D. M. Olsen, and E. W. Gannett. An Overview of the VISUALIZE fx Graphics Accelerator Hardware. *The Hewlett-Packard Journal*, pages 28–34, May 1998. Cited on page 27.

[Sega99] Mark Segal and Kurt Akeley. The OpenGL Graphics System: A Specification (Version 1.2.1). Technical Report, Silicon Graphics Computer Systems, Mountain View, CA, USA, 1999. Cited on page 37.

[Shad96] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996. Cited on page 21.

[Shad98] Jonathan W. Shade, Steven J. Gortler, Li-wei He, and Richard Szeliski. Layered Depth Images. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 231–242. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8. Cited on page 22.

[Sill97] François Sillion, G. Drettakis, and B. Bodelet. Efficient Impostor Manipulationfor Real-Time Visualization of Urban Scenery. *Computer Graphics Forum (Proc. Eurographics '97)*, 16(3):207–218, August 1997. ISSN 1067-7055. Cited on pages 22 and 45.

[Smit84] Alvy Ray Smith. Plants, Fractals and Formal Languages. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH 84 Proceedings)*, volume 18, pages 1–10, July 1984. Cited on page 19.

[Sowi94] Henry Sowizral. Optimizing Graphics for Virtual Reality. *ACM SIGGRAPH 94 Course Notes - Developing Advanced Virtual Reality Applications*, 2:3.1–3.12, July 1994. Cited on page 14.

[Suda96] Oded Sudarsky and Craig Gotsman. Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality. *Computer Graphics Forum (Proc. Eurographics '96)*, 15(3):249–258, August 1996. ISSN 0167-7055. Cited on page 76.

[Tell91] Seth J. Teller and Carlo H. Séquin. Visibility Preprocessing for Interactive Walkthroughs. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 61–69. ACM SIGGRAPH, ACM Press, July 1991. Cited on page 28.

[Tell92a] Seth J. Teller. Computing the antipenumbra of an area light source. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH 92 Proceedings)*, volume 26, pages 139–148. ACM SIGGRAPH, ACM Press, July 1992. ISSN 0097-8930. Cited on page 46.

[Tell92b] Seth Jared Teller. Visibility Computations in Densely Occluded Polyhedral Environments. Technical Report CSD-92-708, University of California, Berkeley, California, U.S.A., 1992. Cited on page 28.

[Tell93] Seth Teller and Pat Hanrahan. Global Visibility Algorithms for Illumination Computations. pages 239–246, August 1993. Cited on page 28.

[Tell01] Seth Teller. MIT City Scanning Project: Fully Automated Model Acquisition in Urban Areas, 2001. http://city.lcs.mit.edu//city.html. Cited on page 20.

[Thom00] Gwenola Thomas and Stéphane Donikian. Modelling virtual cities dedicated to behavioural animation. *Computer Graphics Forum (Proc. Eurographics '00)*, 19(3):71–80, August 2000. Cited on page 19.

[Veen97] Hendrik A.H.C. van Veen, Hartwig K. Distler, Stephan J. Braun, and Heinrich H. Bülthoff. Navigating through a Virtual City, 1997. http://www.mpik-tueb.mpg.de/bu/projects/vrtueb/index.html. Cited on page 19.

[Watt96] Bill Watterson. *It's a Magical World*. Andrews McMeel Publishing, 1996. Cited on page 78.

[Webe95] Jason Weber and Joseph Penn. Creation and Rendering of Realistic Trees. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 119–128. ACM SIGGRAPH, Addison Wesley, August 1995. Cited on page 19.

[Wern94] J. Wernecke. *The Inventor Mentor: Programming Object-Oriented 3d Graphics With Open Inventor*. Addison Wesley, 1994. Cited on pages 20 and 38.

[Wils99] John E. Wilson and Arnie Williams. *3D Modeling in AutoCAD: Creating and Using 3D Models in AutoCAD 2000*. CMP Books, 1999. Cited on page 19.

[Wimm99] Michael Wimmer, Markus Giegl, and Dieter Schmalstieg. Fast walkthroughs with image caches and ray casting. *Computers and Graphics*, 23(6):831–838, December 1999. Cited on page 42.

[Wimm01] Michael Wimmer, Peter Wonka, and François Sillion. Point-Based Impostors for Real-Time Visualization. In Karol Myszkowski and Steven J. Gortler, editors, *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 2001)*. Eurographics, Springer-Verlag Wien New York, June 2001. Cited on pages 11, 56, and 75.

[Wonk99] Peter Wonka and Dieter Schmalstieg. Occluder Shadows for Fast Walkthroughs of Urban Environments. *Computer Graphics Forum (Proc. Eurographics '99)*, 18(3):51–60, September 1999. ISSN 1067-7055. Cited on page 10.

[Wonk00] Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs. In Bernard Péroche and Holly Rushmeier, editors, *Rendering Techniques 2000 (Proceedings of the Eurographics Workshop on Rendering 2000)*, pages 71–82. Eurographics, Springer-Verlag Wien New York, June 2000. ISBN 3-211-83535-0. Cited on page 10.

[Wonk01] Peter Wonka, Michael Wimmer, and François Sillion. Instant Visibility. *Computer Graphics Forum (Proc. Eurographics 2001)*, 20(3), September 2001. Cited on page 10.

[Woo99] M. Woo, J. Neider, T. Davis, D. Shreiner, and OpenGL Architecture Review Board. *OpenGL Programming Guide*. Addison Wesley, 1999. Cited on page 20.

[Wood00] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface Light Fields for 3D Photography. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 287–296. ACM SIGGRAPH, Addison Wesley, 2000. Cited on page 22.

[Xia96] Julie C. Xia and Amitabh Varshney. Dynamic View-Dependent Simplification for Polygonal Models. In Roni Yagel and Gregory M. Nielson, editors, *Proceedings of the conference on Visualization '96*, pages 327–334. IEEE, October 1996. ISBN 0-7803-3673-9. Cited on page 21.

[Xian99] Xinyu Xiang, Martin Held, and Joseph S. B. Mitchell. Fast and Effective Stripification of Polygonal Surface Models. In Stephen N. Spencer, editor, *1999 Symposium on interactive 3D Graphics*, pages 71–78. ACM SIGGRAPH, ACM Press, April 1999. Cited on page 21.

[Yage96] R. Yagel and W. Ray. Visibility Computation for Efficient Walkthrough Complex Environments. *PRESENCE*, 5(1):1–16, 1996. Cited on page 28.

[Zhan97] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility Culling Using Hierarchical Occlusion Maps. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 77–88. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. Cited on pages 24, 25, 30, 41, 42, and 78.

[Zhan98] Hansong Zhang. Effective Occlusion Culling for the Interactive Display of Arbitrary Models. Technical Report TR99-027, University of North Carolina, Chapel Hill, February 1998. Cited on pages 25 and 30.

Curriculum Vitae

Peter Wonka
Hölzlgasse 3
3400 Klosterneuburg

Mai 2001

Sprachen: Deutsch, Englisch, Französisch

10.8.1975:	geboren in Wien, Österreich
September 1981 - Juni 1985:	Besuch der Volksschule in Klosterneuburg
September 1985 - Juni 1993:	Besuch des Bundesrealgymnasiums in Klosterneuburg
Oktober 1993 - Juni 1997:	Studium der Informatik an der Technischen Universität Wien
Juni 1997:	Abschluß des Informatikstudiums als Diplomingenieur der Informatik
Oktober 1997 - Juni 2001:	wissenschaftlicher Mitarbeiter am Institut für Computergraphik
September 1998 - Jänner 1999:	wissenschaftlicher Mitarbeiter an der Universität Rennes II (F)
Juni 2000 - Jänner 2001:	wissenschaftlicher Mitarbeiter an der Universität Joseph Fourier (F)