DISSERTATION

Photorealistic and Hardware Accelerated Rendering of Complex Scenes

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

Universitätsprofessor Dr. Werner Purgathofer
Institut 186 für Computergraphik und Algorithmen

eingereicht an der Technischen Universität Wien
Technisch-Naturwissenschaftliche Fakultät

von

Dipl.-Ing. Heinrich Hey
9225223
Schwaigergasse 19/3/34
A-1210 Wien

Wien, am 11.5.2002

Kurzfassung

Diese Arbeit präsentiert neue Methoden zur effizienten fotorealistischen und Hardware-beschleunigten Bildgenerierung von Szenen die komplexe globale Beleuchtung aufweisen, und zusätzlich auch groß sein können. Das beinhaltet

- eine Photon Map-basierte Radiance Abschätzungs Methode die die Qualität der globalen Beleuchtungs Lösung in der Photon Map-globalen Beleuchtungs Simulation verbessert.
- eine Particle Map-basierte Importance Sampling Technik die die Leistung von stochastischer Ray Tracing-basierter Bildgenerierung und globaler Beleuchtungs Simulation verbessert.
- eine Hardware-beschleunigte Bildgenerierungs Methode die das interaktive Durchschreiten global beleuchteter glänzender Szenen ermöglicht.
- eine Occlusion Culling Technik die das interaktive Durchschreiten auch in großen Szenen ermöglicht.

Es hat sich erwiesen daß die Photon Map-globale Beleuchtungs Simulation eine leistungsvolle Methode zur Ray Tracing-basierten Bildgenerierung von global beleuchteten Szenen mit allgemeinen bidirektionalen Streuungs Verteilungs Funktionen, und allen dadurch möglichen Beleuchtungseffekten ist. Dennoch, eine der Schwächen dieser Methode ist bisher gewesen daß sie eine sehr grobe Radiance-Abschätzung verwendet, die Beleuchtungs-Artefakte in der Nähe von Kanten und Ecken von Objekten, und auf Oberflächen mit unterschiedlich orientierten kleinen geometrischen Details verursachen kann. Unsere neue Photon Map-basierte Radiance-Abschätzungs Methode vermeidet diese Artefakte. Das wird gemacht indem die tatsächliche Geometrie der beleuchteten Oberflächen berücksichtigt wird.

In stochastischem Ray Tracing-basierten Bildgenerierungs und globalen Beleuchtungs Techniken, z.B. in Photon Map-globale Beleuchtungs Simulation, muß eine sehr große Anzahl an Strahlen in die Szene geschossen werden um die globale Beleuchtung und/oder das endgültige Bild zu berechnen. Die Leistung dieser Techniken kann daher wesentlich verbessert werden indem die Strahlen vorzugsweise in Richtungen geschossen werden wo sie einen hohen Beitrag liefern. Importance Sampling Techniken versuchen dies zu tun, aber das Problem dabei ist daß der Beitrag geschätzt werden muß, und das muß freilich effizient getan werden.

Unsere neue Importance Sampling Technik löst dieses Problem unter Verwendung einer Particle Map. Die Wahrscheinlichkeits-Dichte Funktion anhand derer die Schußrichtung eines von einem Punkt ausgehenden Strahls

gewählt wird ist aus adaptiven Abdrücken zusammengesetzt die die nähesten Nachbar Partikel auf der Hemisphäre über dem Punkt machen. Die Strahlen können daher präzise in Richtungen mit hohem Beitrag geschossen werden.

Interaktive Durchschreitungen in global beleuchteten statischen Szenen können realisiert werden indem die berechnungsintensive globale Beleuchtungs Simulation in einem Vorverarbeitungsschritt getan wird. Das Resultat dieses Schritts sollte eine Repräsentation der globalen Beleuchtung sein die in einer folgenden interaktiven Durchschreitung effizient dargestellt werden kann, die mit Grafik-Hardware dargestellt wird. Ein wesentliches Problem dabei ist die räumlich und richtungsmäßig variierende globale Beleuchtung auf glänzenden Oberflächen zu handhaben. Unsere neue Methode für interaktive Durchschreitungen von leicht glänzenden Szenen lößt dieses Problem mit richtungsabhängigen Light Maps, die effizient mit konventioneller Grafik-Hardware dargestellt werden können.

In großen Szenen, z.B. in einem Gebäude, in denen von jedem möglichen Betrachtungspunkt aus nur ein kleiner Teil sichtbar ist, wäre es ineffizient all jene Objekte zu zeichnen die von anderen Teilen der Szene verdeckt sind. Um eine Echtzeit-Bildwiederholrate für interaktive Durchschreitungen zu erreichen ist es notwendig effizient zu ermitteln welche Objekte verdeckt sind, damit sie weggelassen werden können. Unsere neue konservative Bildraum-Occlusion Culling Methode erreicht das unter Verwendung eines Lazy Occlusion Grids das effizient mit konventioneller Grafik-Hardware funktioniert.

Abstract

This work presents new methods for the efficient photorealistic and hardware accelerated rendering of scenes which exhibit complex global illumination, and which additionally also may be large. This includes

- a photon map-based radiance estimation method that improves the quality of the global illumination solution in photon map global illumination simulation.
- a particle map-based importance sampling technique which improves the performance of stochastic ray tracing-based rendering and global illumination simulation.
- a hardware accelerated rendering method which allows to do interactive walkthroughs in globally illuminated glossy scenes.
- an occlusion culling technique which allows to do interactive walkthroughs also in large scenes.

Photon map global illumination simulation has proven to be a powerful method for ray tracing-based photorealistic rendering of globally illuminated scenes with general bidirectional scattering distribution functions, and all illumination effects that are possible thereby. Nevertheless, one of the weaknesses of this method has been that it uses a very coarse radiance estimation which may cause illumination artifacts in the vicinity of edges or corners of objects, and on surfaces with differently oriented small geometric details. Our new photon map-based radiance estimation method avoids these illumination artifacts. This is done by taking the actual geometry of the illuminated surfaces into consideration.

In stochastic ray tracing-based rendering and global illumination techniques, eg. in photon map global illumination simulation, a very large number of rays have to be shot into the scene to compute the global illumination solution and/or the final image. The performance of these techniques can therefore be considerably improved by shooting the rays preferably into directions where their contribution is high. Importance sampling techniques try to do this, but the problem herein is that the contribution has to be estimated, and this of course has to be done efficiently.

Our new importance sampling technique solves this problem by utilization of a particle map. The probability density function according to which the shooting direction of a ray from a point is selected is composed of adaptive footprints that the nearest neighbor particles make onto the hemisphere above the point. The rays can therefore be precisely shot into directions with high contribution.

Interactive walkthroughs in a globally illuminated static scene can be realized by doing the computationally expensive global illumination simulation in a preprocessing step. The result of this step should be a representation of the global illumination that can be efficiently displayed during a following interactive walkthrough, which is rendered with graphics hardware. A major problem herein is to handle the spatially and directionally variant global illumination on glossy surfaces. Our new method for interactive walkthroughs for soft glossy scenes solves this problem with directional light maps, which are efficiently displayed with conventional graphics hardware.

In large scenes, eg. in a building, where only a small part is visible from each possible viewpoint, it would be inefficient to draw all those objects that are occluded by other parts of the scene. To achieve a real-time frame-rate for interactive walkthroughs it is necessary to determine efficiently which objects are occluded, so that they can be culled. Our new conservative image-space occlusion culling method achieves this by utilization of a lazy occlusion grid that works efficiently with conventional graphics hardware.

Contents

Chapter 1

Introduction

Photorealistic rendering is needed in all application areas where the accurate simulation of indirect illumination is important to achieve realistic images, for example in:

- architecture
- lighting design
- stage design
- film

The scenes that are used in such applications often exhibit complex global illumination on its surfaces. This global illumination is caused by light that is reflected and refracted by other objects in the scene. These scenes may contain surfaces with general reflection properties, and therefore all kinds of illumination effects that are possible thereby.

For several applications it is also important that interactive walkthroughs can be done in these globally illuminated scenes. An example application would be to give customers a realistic impression of their planed new house by walking around in the virtual model. To achieve this realistic impression it is necessary to display the scenes with accurate indirect illumination. This also includes that glossy surfaces have to be supported, because many real materials are non-diffuse. Without glossy materials these surfaces would look very synthetic, because highlights would be missing, which are very important for the realistic perception of the scenes.

Often the scenes in these applications are additionally also large, for example a whole building with several rooms. The interactive walkthroughs should nevertheless also be possible in these large globally illuminated scenes. An

important property of such large scenes is that usually only a small part of the scene is visible from each possible viewpoint, because large parts of the scene are occluded by other parts in front of them. For example a viewer inside a room will usually be able to see only into few other rooms.

In this work we present new methods for the efficient rendering of photorealistic images and interactive walkthroughs in such complex scenes. In each chapter we also give an overview of existing related methods.

The first new method, which is explained in chapter 2, is a photon map-based radiance estimation for photon map global illumination simulation. The later has proven to be a powerful technique for ray tracing-based photorealistic rendering of globally illuminated scenes. It supports surfaces with general bidirectional scattering distribution functions, and all illumination effects that are possible thereby. Our new radiance estimation method improves the quality of the resulting images by avoiding illumination artifacts of the existing method in the vicinity of edges or corners of objects, and on surfaces with differently oriented small geometric details. Our new method does this by taking the actual geometry of the illuminated surfaces into consideration.

Next, we present a new importance sampling method in chapter 3, which improves the performance of stochastic ray tracing-based rendering and global illumination techniques, in particular photon map global illumination simulation. Our new importance sampling method utilizes a particle map to select the direction into which a path is scattered at a surface. The global illumination computations are thereby concentrated to those parts of the scene with the highest contribution.

After that we describe a new technique in chapter 4, which allows to do interactive walkthroughs in globally illuminated soft glossy scenes. It uses directional light maps which represent the spatially and directionally variant global illumination in form of textures at the surfaces. The directional light maps are generated in a photon tracing preprocessing step. Afterwards during the interactive walkthrough these directional light maps are efficiently displayed with conventional graphics hardware.

Finally we show how the interactive walkthroughs can be done in large globally illuminated scenes. This is achieved with a new conservative image-space occlusion culling method, which is explained in chapter 5. It is based upon a lazy occlusion grid, and it works efficiently with conventional graphics hardware. It determines which parts of the scene are occluded by other parts, so that the occluded parts can be culled, and that only the potentially visible parts have to be displayed.

Chapter 2

Global illumination simulation with photon maps

In this chapter we discuss how photon maps can be used to simulate global illumination. After an overview of existing methods in chapter 2.1, we present our new method for geometry based radiance estimation by means of the photon map [HP02b], which improves the quality of the photon map global illumination simulation.

2.1 Existing methods

Existing photon map global illumination simulation [Jen96b, JCS01] supports scenes with general bidirectional scattering distribution functions (BSDFs). It allows to simulate all light transport paths (L(D|S)*E[†]) and all illumination effects that are possible thereby [Chr97].

A photon map stores information about the directionally variant indirect illumination in the scene in form of photons. These photons are distributed into the scene in a photon tracing pass. During image generation in a following ray tracing pass this illumination information in the photon map is used in a radiance estimation to compute the indirect illumination at the displayed surface points. These steps are described in the following sub-chapters.

In the following we concentrate on global illumination simulation on surfaces. Nevertheless, photon map global illumination simulation can also be extended to support participating media [JC98].

[†] Heckbert's notation of light transport paths [Hec90]. In our context S means a specular or strong glossy surface, and D means a diffuse or soft glossy surface.

2.1.1 Photon tracing pass

A photon map is generated in a photon tracing pass. The lightsources distribute their energy into the scene by shooting stochastically distributed light paths. At each point after the first bounce where a light path hits an object, information about the incoming indirect light is stored in form of a photon. The photon stores the hit position, the incoming direction of the light path and the incoming light power. The photons are organized in a spatial structure, eg. a kd-tree, which represents the photon map [Jen96a].

Usually a separate caustic photon map is used which contains all photons with LS+D paths. These photons represent caustics. All other photons are stored in a second photon map, the global photon map, which represents soft indirect illumination.

Density control [SW00] can be used to achieve a more uniform photon distribution, so that a smaller number of photons is sufficient.

2.1.2 Ray tracing pass

After the photon tracing pass the illumination information in the photon map is used to render the scene in a ray tracing pass. View paths that sample the image are shot from the camera into the scene. If a view path hits a specular or strong glossy surface, the view path is continued by shooting a ray from the hit point. The outgoing direction of the ray is stochastically distributed according to the BSDF of the surface. Alternatively importance sampling by means of the photon map [HP02a, Jen95, PP98] can be used to select the outgoing direction. Otherwise, if the hit surface is diffuse or soft glossy, the view path ends at this point x, and the radiance at x into the incoming direction of the view path is computed.

The direct illumination part (LD sub-paths) is conventionally computed by casting shadow rays from x to the lightsources. The indirect illumination part that represents caustics (LS+D sub-paths) is computed by using the radiance estimate at x of the caustic photon map, as described in chapter 2.1.3. Due to this direct visualization of the radiance estimate, its quality is very important for the quality of the resulting caustics.

If the surface has a low contribution to the image then the soft indirect illumination part (LS*D(D|S)*D sub-paths) is computed by using the radiance estimate at x of the global photon map. Otherwise final gathering is used to compute the soft indirect illumination. Final gathering shoots additional rays

from x which gather radiance estimates from the scene. These radiance estimates are averaged so that the effect of a few wrong estimates is minimized [Dri00]. The final gathering operation can be accelerated by using irradiance gradients [WH92], and by precalculating irradiance estimates at diffuse surfaces, so that each of these irradiance estimates has to be calculated only once [Chr99].

2.1.3 Radiance estimation

Existing photon map radiance estimation [Jen96a] expands a sphere around the given point x until it contains n_{max} photons, or until the radius r of the sphere is equal to r_{max}. This expansion of the sphere corresponds to searching for the n_{max} nearest photons to x, up to the maximum distance r_{max}. n_{max} and r_{max} are user defined constants which control the variance and blurring of the resulting illumination. More nearest neighbor photons mean less variance, but at the same photon density it also means more blurring due to their larger distances to x.

This radiance estimation assumes that the nearest neighbor photons lie in the same plane as x, and that they distribute their power over the circular area $A_c=r^2\pi$ around x. A_c is the intersection area of the sphere and x's plane. The radiance L at x into direction Ψ_v is therefore estimated as

$$L \approx \sum_{p \in P_n} \frac{\Phi_p}{A_c} f(x, \Psi_p, \Psi_v) = \frac{1}{r^2 \pi} \sum_{p \in P_n} \Phi_p f(x, \Psi_p, \Psi_v) . \qquad (2.1)$$

P_n is the set of photons inside the sphere, Φ_p is the flux that the nearest neighbor photon p carries, Ψ_p is its incoming direction, and f is the BSDF at x from Ψ_p to Ψ_v.

Optionally an ellipsoid can be used instead of the sphere. The ellipsoid is oriented along the plane of x. This minimizes that photons which lie in different planes are used in the radiance estimate, thereby reducing light leakage between different surfaces.

This radiance estimation is incorrect in the vicinity of edges (and corners) of objects, because those photons which contribute to the illumination of a point in this region are distributed only at one side of the edge. Therefore the circular area is an overestimation of the actual area over which the nearest neighbor photons distribute their energy. The resulting illumination artifact is a dark region near the edge (see figure 2.1a). If the surface is large and quasi-planar then adaptive density estimation [Mys97] can be used to avoid this kind of artifact.

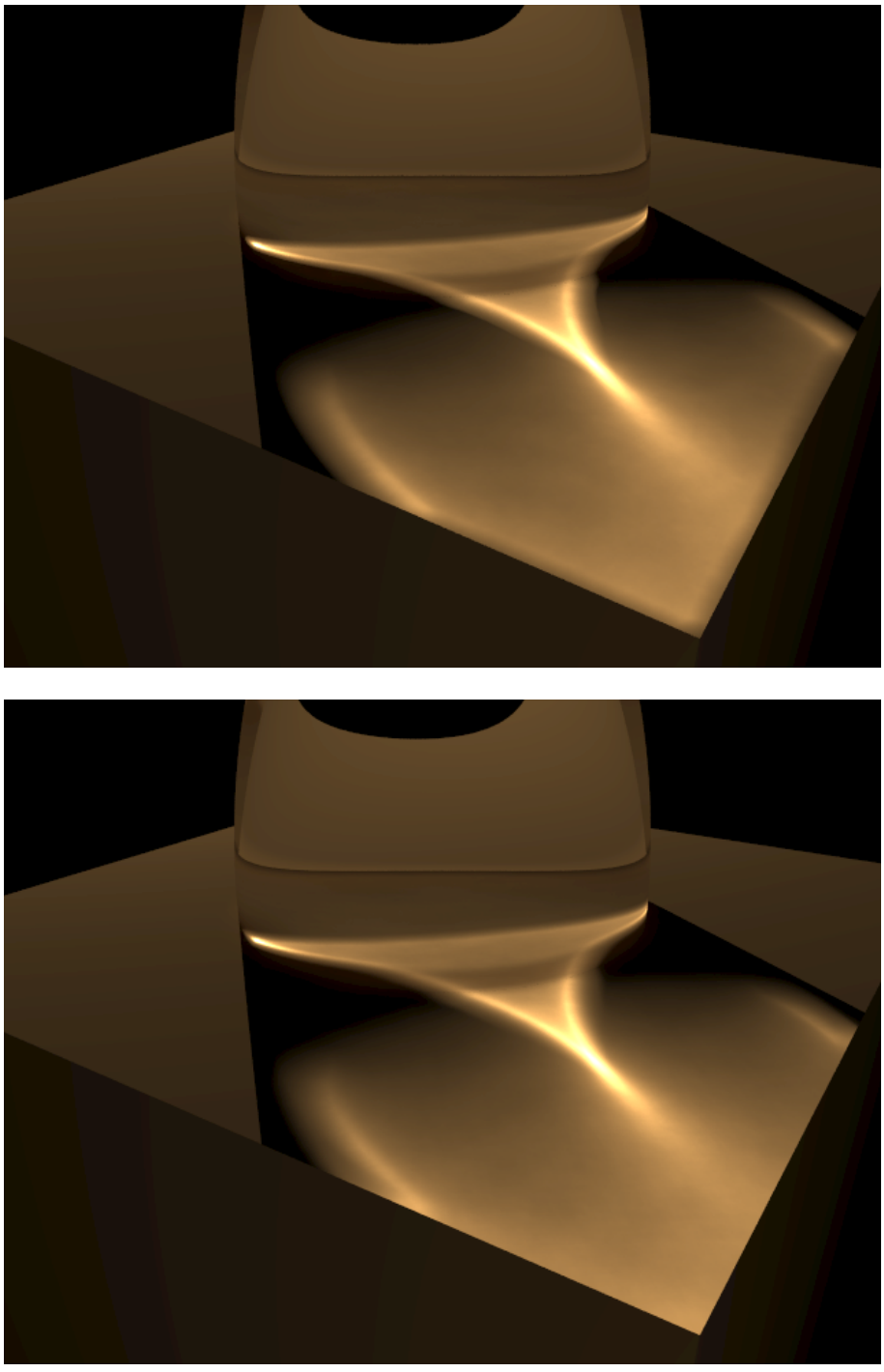

Figure 2.1a (top)**:** Existing radiance estimation causes illumination artifacts (dark regions) in the vicinity of the object edges. **2.1b** (bottom)**:** Our new geometry based radiance estimation avoids these illumination artifacts for approximately the same rendering time.

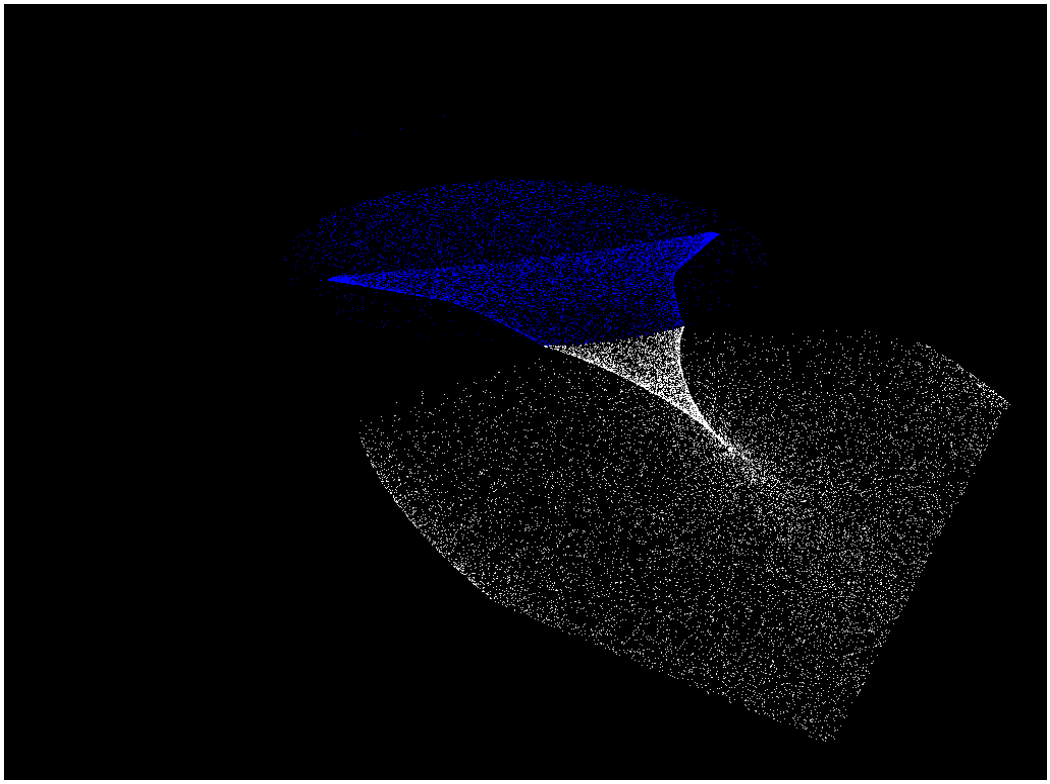

Figure 2.1c (bottom)**:** One of the photon maps which were used for both radiance estimation methods in figure 2.1a and 2.1b. Hidden photons at the bottom of the glass egg are shown in blue.

Differential checking [JC95] avoids excessive blurring of caustics. This is done by including photons into the neighborhood sphere (or ellipsoid) only as long as it does not significantly increase or decrease the radiance estimate. In general differential checking does not avoid the illumination artifact in the vicinity of an edge, because if all nearest neighbor photons lie on one side of the edge then including them into the sphere does not significantly change the underestimated radiance.

The radiance estimate is also incorrect on surfaces with differently oriented small geometric details, as can be seen in figure 2.2a. The neighborhood sphere is larger than the illuminated small surface detail, therefore many nearest neighbor photons do not lie in the same plane as x. Using differential checking, or using an ellipsoid instead of the sphere would not solve this problem, because on such a small surface detail too few photons lie in x's plane for a reliable radiance estimate.

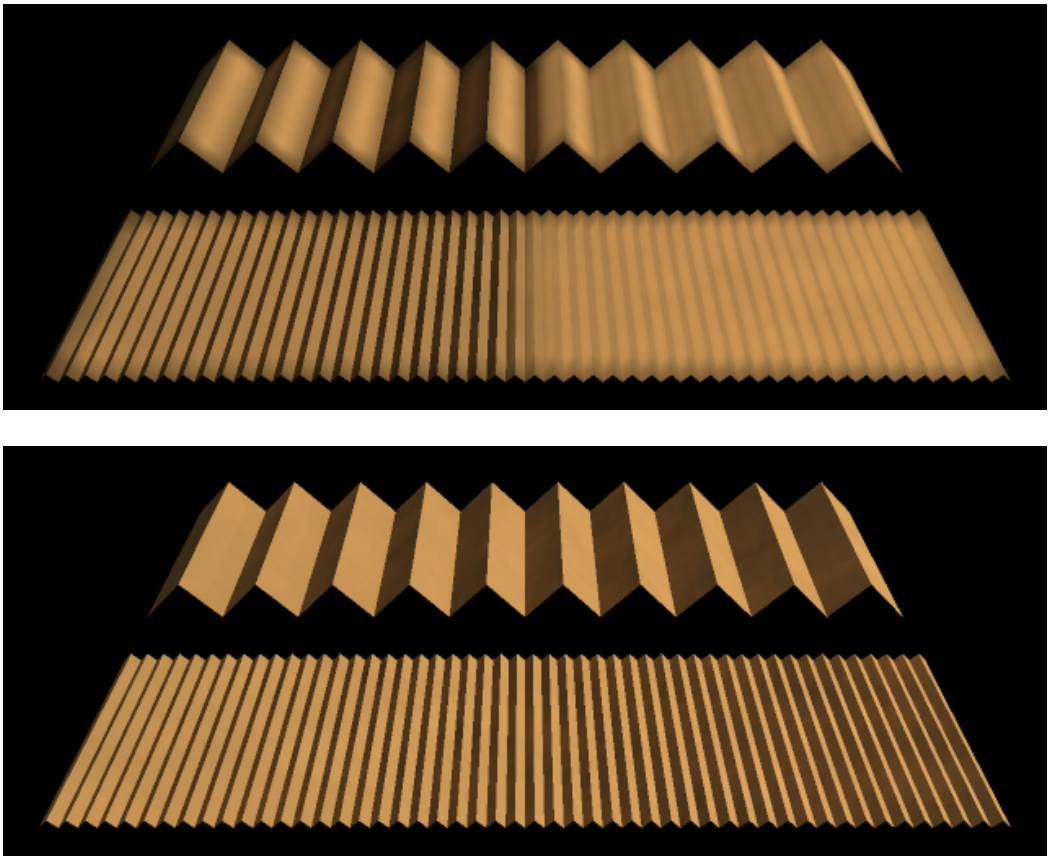

Figure 2.2: Surfaces with geometric details that are larger and smaller than a neighborhood sphere/cube. Most indirect illumination comes from approximately 45° from top right. **2.2a** (top)**:** Existing radiance estimation incorrectly estimates radiance on the small surfaces. On the left side of the image, where the incoming direction of the photons is a little bit below 45°, the radiance on the small surfaces is underestimated. On the right side of the image, where the incoming direction of the photons is a little bit above 45°, the radiance on the small surfaces is overestimated. **2.2b** (bottom)**:** Our new geometry based radiance estimation avoids these illumination artifacts for approximately the same rendering time.

2.2 Geometry based radiance estimation

As we have seen in chapter 2.1.3, existing photon map radiance estimation [Jen96a] assumes that the photons in the neighborhood of a requested surface point x lie in the same plane as x, and that they are distributed in a circular area around x. Therefore illumination is incorrectly estimated in regions where these assumptions are not true. In particular this is the case in the vicinity of edges or corners of objects, and on surfaces with differently oriented small geometric details.

Our new radiance estimation method is used as replacement for the existing radiance estimation in photon map global illumination simulation, as described in chapter 2.1. The most important difference to existing radiance estimation is that our new method uses the actual geometry in the neighborhood of x to determine the area over which the nearest neighbor photons distribute their power. It does not require the previous assumptions about the location and distribution of the photons in x's neighborhood. Therefore it gives accurate illumination also in those regions where these assumptions would not be true. This results in higher image quality for approximately the same rendering time, as shown in figure 2.1b and 2.2b.

A high quality radiance estimate is especially important for photon map based rendering of caustics (LS+DS*E paths), which is done by direct visualization of the photon map. In particular if a caustic is bright, artifacts due to the photon map radiance estimation are visible.

Our method uses a mesh-representation of the scene for its geometrical computations, but note that it does not store any illumination information in this mesh. The part of the mesh that potentially intersects x's neigborhood could therefore be generated on demand, which could also be combined with an adaptive triangulation of non-polygonal geometry. This could be efficient for large scenes where only a part of the geometry is visible.

In this description of our method we use an octree to organize the scene geometry. This allows us to efficiently find the geometry in the neighborhood of x. Note that other spatial subdivision structures (eg. kd-trees or hierarchical grids) or a hierarchy of bounding volumes could be used instead.

2.2.1 Generation of the octree of polygons

Before the photon tracing pass a octree of polygons is generated from the geometry in the scene. This octree is later on used in the radiance estimation during the ray tracing pass to efficiently find the geometry in the neighborhood of x.

The octree encloses the whole scene. Each leaf node of the octree stores references to all surface polygons that intersect this node or that lie completely inside this node. Each polygon may therefore be referenced by several leaves. The octree is generated by recursively subdividing its nodes. The subdivision starts at the root of the octree which contains the whole scene. A node is subdivided if it contains more than a user defined number of polygons n_P, but only if the side lengths of the node are larger than a user defined threshold s. s avoids infinite subdivisions at vertices or edges where more than n_P polygons meet. For each subnode of a subdivided node it is determined which polygons the subnode contains. This is done by testing which of those polygons that are contained in the subdivided node intersect or lie completely inside the subnode [GH95, Voo92]. Only leaves have to store their polygon references. The intermediate nodes of the finished octree only store references to their subnodes.

2.2.2 Radiance estimation

Our radiance estimation during the ray tracing pass expands a cube, which is centered at the given point x, and which is axis aligned with the coordinate system of the photon map and the octree of polygons, until it contains n_{max} photons, or until the half side length l of the cube is equal to l_{max}. This expansion of the cube corresponds to using the max-distance $max(|x|,|y|,|z|)$ instead of the euclidean distance when searching for the n_{max} nearest photons to x, up to the maximum max-distance l_{max}. n_{max} and l_{max} are user defined constants which control the variance and blurring of the resulting illumination, similar as in existing radiance estimation.

Next, we continue expanding the cube as long as the difference of the max-distance of the next nearest neighbor photon and the current l is less than a user defined ε. After that we add $\varepsilon/2$ to l to get the final value of l. These steps guarantee that a surface which is (nearly) coplanar to the walls of the cube is in the cube if the photons on the surface are in the cube, and vice versa (due to numerical inaccuracies the photons may not lie exactly on the surface).

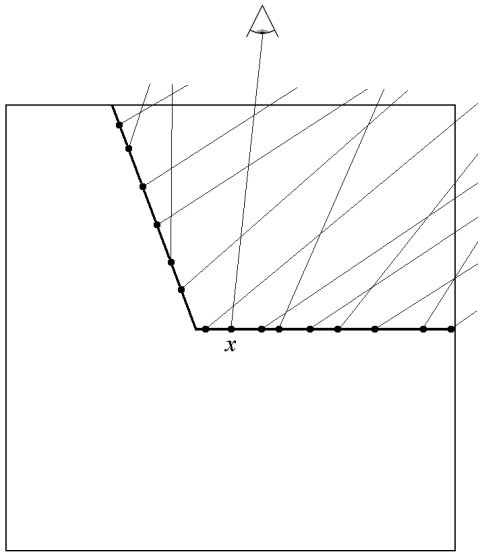

Figure 2.3: Neighbor polygons and nearest neighbor photons (with incoming directions) in the neighborhood cube around the requested point *x*.

In the next step we determine the area of each polygon inside the cube. The polygons in the cube can be efficiently found by searching for the octree leaves that intersect the cube. The polygons that are contained in these leaves are potentially inside the cube. All other surfaces in the scene are completely outside of the cube. Each potentially-inside polygon is clipped against the cube to determine the inside part of the polygon. Each potentially-inside polygon has a flag that shows if it already has been clipped to avoid that a polygon is processed several times if it is contained in several leaves. In the following we will call the polygon parts inside the cube neighbor polygons. The area A_n of a neighbor polygon is calculated as sum of the areas of the triangles of its triangulation. The area of a triangle is calculated as

$$A_{tri} = \frac{1}{2}\left|N_n \cdot \left(\left(P_2 - P_1\right) \times \left(P_3 - P_1\right)\right)\right|. \tag{2.2}$$

N_n is the polygon's normal, and P_i are the vertices of the triangle.

Each nearest neighbor photon distributes its power Φ_p over all neighbor polygons that are frontfacing ($0<\Psi_p N_n$) into the photon's incoming direction Ψ_p[†]. The power that a frontfacing neighbor polygon receives from the nearest neighbor photon is proportional to the neighbor polygon's projected area A_{np} into the photon´s incoming direction.

$$A_{np} = A_n \Psi_p N_n \tag{2.3}$$

This yields the photon's flux area density u_p on the projected areas of the frontfacing neighbor polygons.

$$u_p = \frac{\Phi_p}{\sum_n A_{np}} = \frac{\Phi_p}{\sum_n A_n \Psi_p N_n} \tag{2.4}$$

The photon's irradiance at x, which has the surface normal N_x, is

$$E_p = u_p \Psi_p N_x . \tag{2.5}$$

This finally gives us the estimate for the radiance L at x into direction Ψ_v. f is the BSDF at x from Ψ_p to Ψ_v.

$$L \approx \sum_p E_p f(x, \Psi_p, \Psi_v) \tag{2.6}$$

2.3 Results

We have compared existing nearest neighbor area estimation [Jen96a] and our new geometry based radiance estimation in a parallel implementation of photon map global illumination simulation, which ran on a cluster of 10 PCs with dual 1 GHz Pentium3s in a 100 MBit Ethernet network. In this implementation each CPU generates an individual photon map, and then uses it to render an image with 1 view path per pixel. The images from all CPUs are then accumulated to achieve the final image. Using several individual photon maps has the advantage

[†] Due to the distribution of the nearest neighbor photons' power over the surfaces in the neighborhood it is possible in any radiance estimation method that light leakage occurs in the extent of the neighborhood. This may happen even if an ellipsoid and differential checking is used. Light leakage could be avoided by determining which photons can contribute to which surfaces. This would require efficient visibility tests between x, the photons and the geometry.

that the variance in the illumination in the final image can be made arbitrarily low without requiring very large photon maps. The same set of photon maps has been used for both methods. In the geometry based radiance estimation implementation is a random rotation used to define the common coordinate system of the octree of polygons, the photon map, and the neighborhood cubes. We do this to avoid directionally non-uniform blurring due to the shape of the neighborhood cubes.

Figure 2.1 shows the quality of the radiance estimate, which is directly visualized to render the caustic, in the vicinity of object edges and corners. 20 photon maps have been used. Each map contains 28,585 photons on average. 100 nearest neighbor photons have been used per radiance estimate. This corresponds to 2,000 photon contributions per final pixel in the caustic. The total rendering time was 8.7 minutes. In figure 2.1, as well as in figure 2.2, the differences between the rendering times of both methods were lower than the differences between several runs of the same method.

Figure 2.2 shows the quality of the radiance estimate at a surface with geometric details that are smaller than the size of a neighborhood sphere/cube. 20 photon maps with an average of 13,741 photons per map, and 100 nearest neighbor photons per radiance estimate have been used, which corresponds to 2,000 photon contributions per final pixel. The total rendering time was 4.6 minutes.

Chapter 3

Importance sampling with particle maps

In this chapter we discuss how importance sampling in stochastic ray tracing-based rendering and global illumination techniques can be done by means of a particle map (photon map or importance map). After an overview of existing methods in chapter 3.1, we present our new importance sampling method which is based on hemispherical particle footprints [HP02a], and compare it with existing importance sampling techniques in photon map global illumination simulation.

3.1 Existing methods

In scenes that are large, or that contain complex illumination settings efficiency demands to concentrate the computational effort during the global illumination simulation and rendering to those parts of the scene that contribute most to the image.

For stochastic ray tracing-based rendering and global illumination methods this means that paths shall be shot preferably into directions where their effect is high. Light paths shall be shot preferably into parts of the scene that are visible, so that as little work as possible is spent into unnecessarily illuminating invisible parts of the scene. View paths shall be shot preferably into directions where much light comes from, so that they contribute most to the image.

This problem can be solved with importance sampling, where the outgoing direction of a ray of a path is selected stochastically distributed according to a probability density function (PDF) which approximates the contribution of the path into the outgoing direction.

A simple way to do importance sampling is to select the direction solely by means of the bidirectional scattering distribution function (BSDF) at the scattering point [BLS94, DW94, LF97, Lan91, TN+98]. It does not require to estimate the incoming light or visibility, but this of course makes it less likely that the PDF corresponds to the actual contribution.

Several methods use meshing to store an approximation of the contribution in the scene [DW95, NN+96, SCP99, UT97]. Another solution is to generate the PDFs from incoming radiance which is stored in a 5D tree [LW95], or to store the illumination in the scene in a neural gas structure [Bus97]. Alternatively outgoing directions can also be generated in an evolutionary manner [Bus97, LB94] instead of stochastically, or they can be generated by mutating already existing paths with high contributions [VG97].

Photon map based importance sampling [Jen95] uses a photon map, which is generated in a particle tracing pass, as approximation of the illumination to select the scattering directions in a subsequent path tracing pass. The number of photons in the photon map can be controlled with density control [SW00], or with importance driven photon deposition [KW00]. Photon map based importance sampling is also used in photon map global illumination simulation [Jen96b, JCS01] to select the shooting directions of the final gathering rays.

This kind of importance sampling can also be used for selecting scattering directions of light paths by usage of an importance map [PP98], which is generated in a preceding pass that distributes importance [PM93, VG94] into the scene. The importance map is the analogue of a photon map (an importon is the analogue of a photon).

In these methods, a PDF is generated for a given scattering point in the scene by inserting the contribution of the k_p (typically 50) nearest particles from the photon map or importance map into a grid that is mapped onto the hemisphere above the point (see figure 3.1). A grid cell is selected by means of the accumulated contributions of the cells, and the outgoing direction is selected randomly within this cell [Shi92]. The targeting precision into important directions is therefore limited by the fixed grid resolution, which is limited by k_p.

Directional importance information can also be represented in a hierarchical data structure, eg. a kd-tree or a hierarchy of spherical triangles, instead of a grid [TN+98]. A hierarchical data structure is useful if the PDF that it represents is estimated by means of a large number of samples, eg. for generating a PDF at a lightsource by means of the contribution of already shot light paths [DW94]. If only a small number of nearest neighbor particles is available to estimate the

PDF, as it is the case in particle map based importance sampling, then inserting these particles into a hierarchical data structure results in unnecessary blurring of the borders of highly contributing small regions, as shown in figure 3.2. An optimal PDF should be able to represent such important directions precisely to allow precise targeting, as shown in figure 3.3.

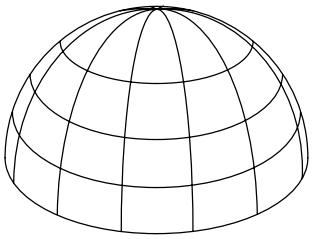

Figure 3.1: Grid on hemisphere that is used for the PDF in existing particle map based importance sampling.

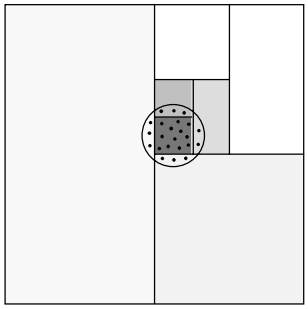

Figure 3.2: Several particles lie in a small solid angle and represent a region of high contribution (here: the circular area), eg. bright light that comes through a small opening or from a small reflector. If these particles are inserted into a hierarchical data structure, eg. a kd-tree, then several particles from this dense region lie in large low density nodes of the data structure. Therefore the region´s border is unnecessarily blurred and not the whole highly contributing region can be precisely targeted.

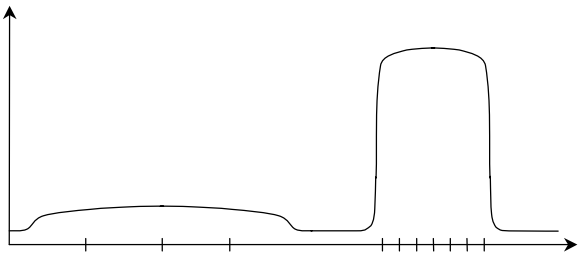

Figure 3.3: An optimal PDF (here sketched in 1 dimension of the hemisphere) should be tightly fitting in directions with directionally dense particles, to allow precise targeting, because the particles provide detailed illumination information in these directions. On the other hand it should be loosely fitting in sparse directions, which means that particles in sparse directions should distribute their contribution to the PDF over a wider solid angle, because the particles provide only coarse illumination information in these directions.

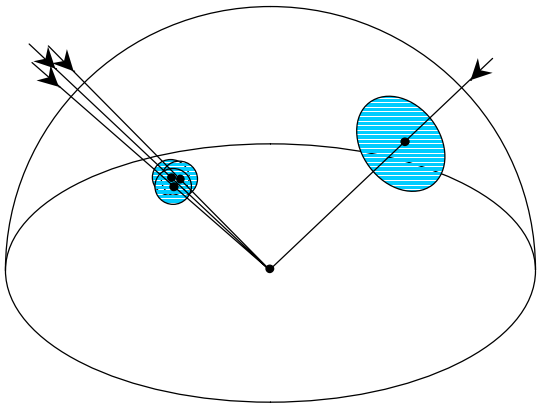

Figure 3.4: Footprints of a few nearest neighbor particles on the hemisphere. Each footprint has an adaptive radius that corresponds to the directional particle density (how many particles come from a nearby direction) at its particle´s incoming direction. We realize a PDF with the characteristics from figure 3.3 as sum of these footprints plus a small BSDF based value (not shown here) to avoid bias in directions without footprints.

3.2 Hemispherical particle footprint importance sampling

We present a new importance sampling method that uses a particle map to generate the PDF. It supports surfaces with general BSDFs, and needs no meshing of the scene. The major advantage of our new method is that it features the desired targeting characteristic from figure 3.3 without increasing the required number of nearest neighbor particles.

For importance sampling of view paths a photon map [Jen95] is used which is generated in a preceding photon tracing pass. For importance sampling of light paths an importance map [PP98] is used which is generated in a preceding importon tracing pass.

Given a point in the scene, for which an outgoing direction shall be selected, a PDF is realized by making footprints of the nearest neighbor particles onto the hemisphere above the point, as shown in figure 3.4. By selecting the radii of the footprints adaptively according to the directional density of the particles, rays can be shot precisely into highly contributing regions where several particles come from a small solid angle.

The contribution of a nearest neighbor particle to the PDF corresponds to the light power of the photon, or importance of the importon, and the BSDF. This PDF-contribution of a particle is uniformly distributed in its footprint's area. The footprint's center is located at the incoming direction of the particle. The footprint's radius corresponds to how many other particles come from a nearby direction.

The total PDF, according to which the outgoing direction is selected, is the sum of these footprints plus a small BSDF based value to avoid bias in directions without footprints. In comparison to existing particle map based importance sampling, this has the advantage that, due to the adaptive radii of the footprints, an outgoing ray can be shot more precisely into highly contributing regions where several particles come from a small solid angle.

The selection of an importance sampled outgoing direction at the given point is done in the following sequence:

- Get the nearest neighbor particles of the point from the particle map.
- Make a fast and rough estimation of the directional density of the nearest neighbor particles.
- Select the outgoing direction with an one-sample model [VG95]. This is done
  - by selecting one of the nearest neighbor particles, selecting its footprint radius according to the directional particle density, and selecting the outgoing direction in this particle´s footprint,
  - or by selecting the outgoing direction solely by means of the BSDF, to avoid bias in directions that are not covered by footprints.

  The decision which of these two methods is used is done stochastically. p_{BSDF} is the user defined probability of selecting the outgoing direction solely by means of the BSDF.
- Weight the outgoing direction according to the value of the PDF in this direction. The PDF value is calculated by means of the footprints of the nearest neighbor particles and the BSDF.

Note that the PDF value has to be calculated only for this single generated outgoing direction. Therefore we do not need to calculate the PDF values for the whole hemisphere (this could be represented eg. with spherical wavelets [SS95]).

In the following sub-chapters we describe these steps in more detail.

3.2.1 Nearest neighbor particles

To perform importance sampling at a given scattering point x, we search for the k_p (user defined, typically 50) nearest neighbor particles to x [Jen95] whose contribution c_q to the given path with the incoming direction Ψ_i at x is not 0. c_q can be 0 eg. if x lies at the frontside of an opaque surface, and the particle q lies at the backside. If no k_p such particles can be found within a user defined maximum distance to x, then there is not enough information available for importance sampling at x with the particle map. In this case we have to fall back on importance sampling solely by means of the BSDF.

In the case of a photon, c_q is equivalent to the photon´s reflected flux. In the case of an importon, c_q is equivalent to the importon´s reflected importance.

$$c_q = \Phi_q f\left(x, \Psi_q, \Psi_i\right) \tag{3.1}$$

$$c_q = W_q f\left(x, \Psi_q, \Psi_i\right) \tag{3.2}$$

Φ_q is the flux that the photon q carries, W_q is the importance that the importon q carries, Ψ_q is the particle´s incoming direction, and f is the BSDF at x from Ψ_q to Ψ_i. According to Peter and Pietrek [PP98] an importon does not store its W_q, because it is assumed to be equal for all importons, therefore W_q can be set to 1 in equation 3.2. Extending this definition of an importon so that it stores its W_q for each color channel can nevertheless be useful in many scenes, eg. if parts of the scene are seen through a colored glass.

3.1.2 Directional particle density estimation

Next, we perform a fast and rough estimation of the directional particle density at the hemisphere above x. This estimate is necessary for the selection of the footprints´ radii in the following steps. We estimate the directional particle density by splatting the incoming directions of the nearest neighbor particles onto a grid at the ground plane of the hemisphere (see figure 3.5). The ground plane coincides with the tangential plane of x.

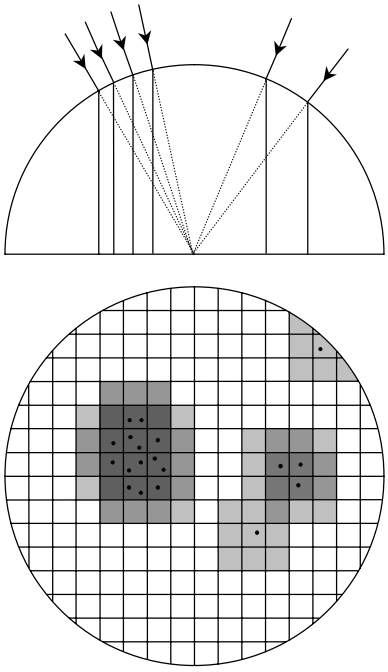

Figure 3.5: The incoming directions of the nearest neighbor particles are projected onto the ground plane of the hemisphere (top) where they make splats (here: 3x3 cells per splat) into a low resolution grid (bottom) to estimate the directional particle density, which is used to select the radii of the footprints.

The incoming direction of a particle is projected onto the ground plane of the hemisphere, where it falls into a cell of the grid, and makes a splat that is centered at this cell. Each splat increases the value g of each cell in its extent by 1, independently of c_q. A splat is more than 1 pixel wide to ensure that the directional particle density is not underestimated in cells at the border of a highly contributing region, because the cells at the border may contain much fewer particles than the cells inside the highly contributing region.

For k_p=50 we use a grid with k_c=32x32 cells, and splats that are 3x3 cells wide. These values have been experimentally found to give the best overall quality per computation time in most cases. A higher resolution grid needs more nearest neighbor particles or larger splats.

This directional particle density estimation method requires that we only use the particles from the positive hemisphere, or only the particles from the negative hemisphere, because otherwise the directions from both hemispheres would be mixed up at the ground plane. Therefore we stochastically select one of both hemispheres. The probabilities $p_{\Omega+}$ and $p_{\Omega-}$ of selecting the positive hemisphere Ω_+ or negative hemisphere Ω_- are

$$p_{\Omega_+} = \frac{\sum_{i \in \Omega_+} c_i}{\sum_{i \in \Omega_+ \cup \Omega_-} c_i} \tag{3.3}$$

$$p_{\Omega_-} = 1 - p_{\Omega_+} . \tag{3.4}$$

After all nearest neighbor particles from the selected hemisphere Ω have been splatted into the grid, the directional particle density estimate δ for a direction Ψ is calculated as

$$\delta(\Psi) = \frac{g_{z_\Psi}}{\omega_{z_\Psi}} . \tag{3.5}$$

z_Ψ is the cell that corresponds to Ψ, and ω_z is the solid angle of cell z projected onto the hemisphere. ω_z is precomputed for each cell z of the grid as

$$\omega_z \approx \frac{4}{k_c \Psi_z \cdot N} . \tag{3.6}$$

k_c is the number of cells in the grid, Ψ_z is the direction that corresponds to the center of z, and N is the surface normal at x.

Note that δ does not need to be very exact, because it is only used to select the radii of the footprints. Even if the footprint radii would be selected arbitrarily, the resulting PDF would still be correct, but of course it would not feature the desired characteristic from figure 3.3. Note also that we do not use this grid as PDF, because it would not have this desired characteristic of fitting tightly in dense directions, and fitting loosely in sparse directions.

3.1.3 Footprints

Each nearest neighbor particle of the selected hemisphere distributes its contribution to the PDF uniformly in a directional footprint on the hemisphere (see figure 3.4). We select the footprint radius (see figure 3.6) of a particle q with incoming direction Ψ_q as

$$r_q = \sqrt{\frac{k_r}{\delta(\Psi_q)}} \tag{3.7}$$

with a user defined scaling factor k_r. To achieve a valid PDF we have to ensure that all generated footprints lie completely in the selected hemisphere. For a particle with an incoming direction at a low angle the r that we get from equation 3.7 results in a footprint that lies partly in the other hemisphere if

$$r_q > r_{max,q} \tag{3.8}$$

with

$$r_{max,q} = \Psi_q \cdot N\,. \tag{3.9}$$

In such a case we have to resize the footprint so that it fits into the selected hemisphere by setting r_q=$r_{max,q}$, as shown in figure 3.7. The footprint´s solid angle is [Bar89]

$$\omega_q = 2\pi h_q\,. \tag{3.10}$$

Herein the footprint's height, which is shown in figure 3.6, is

$$h_q = 1 - \sqrt{1 - r_q^{\,2}}\,. \tag{3.11}$$

A direction Ψ is inside the footprint if

$$\Psi \cdot \Psi_q > 1 - h_q . \tag{3.12}$$

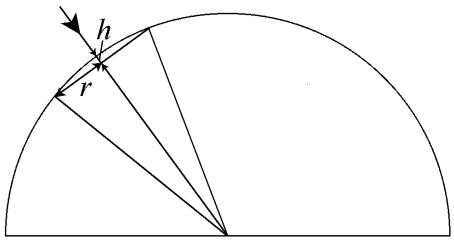

Figure 3.6: Footprint radius r and height h.

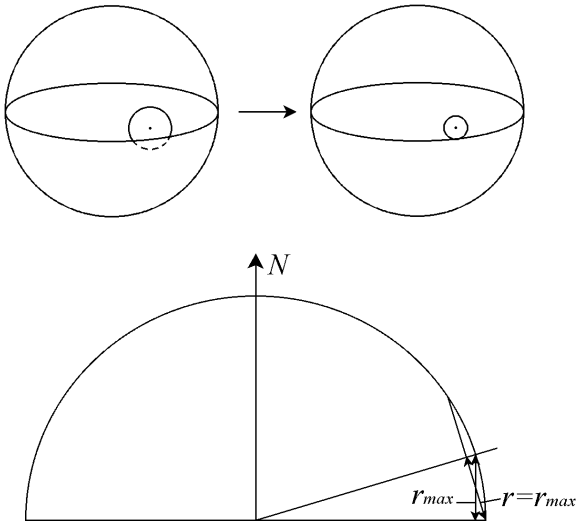

Figure 3.7: Resizing the radius of a footprint so that it fits into the selected hemisphere.

3.1.4 Generation of an importance sampled direction

After the directional particle density estimation we decide stochastically according to p_{BSDF} whether we select the outgoing direction Ψ solely by means of the BSDF, or with the footprints. If we decide to select it with the footprints then we stochastically choose one of the nearest neighbor particles of the selected hemisphere Ω. The probability of selecting particle q is

$$p_q = \frac{c_q}{\sum_{i \in \Omega} c_i}. \tag{3.13}$$

Next, an uniformly distributed direction Ψ is selected in this particle´s footprint. Let u,v be random numbers $u,v \in [0,1)$, then [Shi92]

$$\Psi = (\theta, \phi) = (\arccos(1 - h_q u), 2\pi v). \tag{3.14}$$

(θ,ϕ) is given in a local coordinate system relative to Ψ_q. From equations 3.3, 3.4, 3.13, and from the selection with p_{BSDF} follows that the total probability of selecting Ψ with q's footprint is

$$p_{tot,q} = (1 - p_{BSDF}) p_\Omega p_q = (1 - p_{BSDF}) \frac{c_q}{\sum_{i \in \Omega_+ \cup \Omega_-} c_i}. \tag{3.15}$$

From equation 3.15 and the uniform distribution of Ψ in q's footprint follows that the footprint's contribution to the PDF in direction Ψ is

$$\begin{aligned} p_{f,q}(\Psi) &= p_{tot,q} \frac{4\pi}{\omega_q} && \text{if } \Psi \text{ is inside the footprint, and} \\ p_{f,q}(\Psi) &= 0 && \text{if } \Psi \text{ is outside the footprint.} \end{aligned} \tag{3.16}$$

Let p_b be the PDF for selecting Ψ solely by means of the BSDF, then the total PDF is consequently

$$p(\Psi) = p_{BSDF} p_b(\Psi) + \sum_{i \in \Omega_+ \cup \Omega_-} p_{f,i}(\Psi). \tag{3.17}$$

But due to the fact that $p_{f,q}(\Psi)=0$ for all particles q from the other hemisphere, this is equal to

$$p(\Psi) = p_{BSDF} p_b(\Psi) + \sum_{i \in \Omega} p_{f,i}(\Psi). \tag{3.18}$$

After we have generated the outgoing direction Ψ of a ray by means of the footprints, or solely by means of the BSDF, we finally have to weight the contribution of the ray with $1/p(\Psi)$ to avoid bias.

3.3 Results

We compare our footprint importance sampling technique with classic photon map based importance sampling [Jen95], and with importance sampling solely by means of the BSDF. We have applied all 3 methods in photon map global illumination simulation [Jen96b, JCS01].

Herein importance sampling is used to select the shooting direction of the final gathering rays. Final gathering rays are shot from a surface point to gather illumination from the scene to calculate the soft indirect illumination at the surface point. In our implementation this is done for each point where a view path hits a surface. Irradiance gradients [WH92] could be used to enhance performance [Jen96b] by doing the final gathering operation for a reduced set of points, and interpolating the indirect illumination from these points for other surface points.

We have done our tests on a cluster of 11 PCs with dual 1 GHz Pentium3s in a 100 MBit Ethernet network. Each PC has a copy of the photon map, and each CPU renders a part of the final image. Footprint importance sampling and classic photon map importance sampling use the photon map which is also used for photon map global illumination.

We have used k_p=50 for all 3 importance sampling methods and for the radiance estimation in the photon map global illumination simulation. We have used p_{BSDF}=0.3, k_c=32x32, 3x3 cells wide splats, and k_r=7. These values have been experimentally found to give the best overall quality per computation time. However, the resulting quality is not very sensitive to these parameters, and usually their values can be reused.

The scene in figure 3.8 contains many glossy surfaces, and most parts of the scene receive only indirect illumination. The photon map that has been used for figure 3.8a-c, and which is shown in figure 3.8d, contains 862,880 photons. A more uniform photon distribution, and therefore a smaller number of photons could be achieved by using density control [SW00].

40 view paths per pixel have been used for figure 3.8a-c. 10 final gathering rays per surface point have been used in figure 3.8a. In figure 3.8b 11 final gathering rays per surface point have been used, and in figure 3.8c 17 final gathering rays per surface point have been used to achieve the same rendering time as for figure 3.8a.

The generation of the photon map took 27 seconds, and the rendering pass with importance sampling took 12 minutes. Note that the rendering pass can be

accelerated by using irradiance gradients [WH92], and by precalculating irradiance estimates at diffuse surfaces [Chr99].

As can be seen in figure 3.8a-c, footprint importance sampling results in considerably reduced noise in the same rendering time. We have compared figure 3.8a-c with a high quality solution of the BSDF-only importance sampling method which used 40 view paths per pixel, and 1000 final gathering rays per surface point. The resulting mean square error of figure 3.8a (footprint importance sampling) has been 1.6 times lower than the mean square error of figure 3.8b (classic photon map based importance sampling), and 2.4 times lower than the mean square error of figure 3.8c (BSDF-only importance sampling).

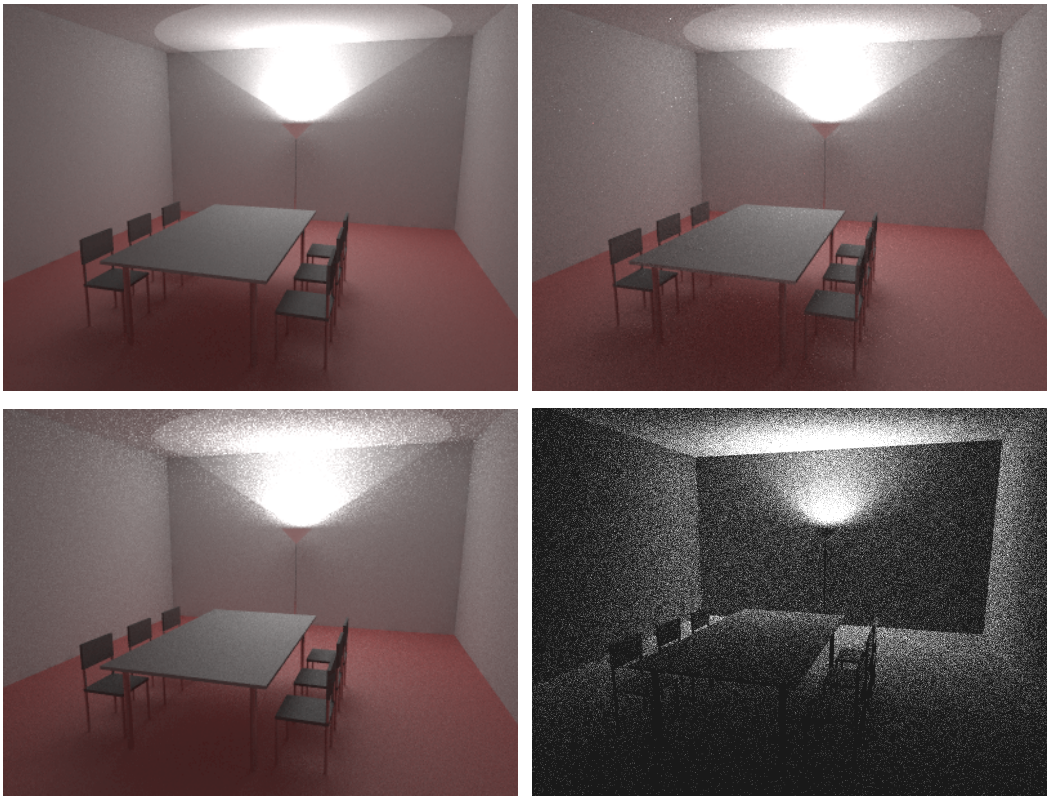

Figure 3.8: Importance sampled photon map global illumination simulation in a scene with many glossy surfaces. Figure 3.8a-c took the same rendering time. **3.8a** (top left)**:** Hemispherical photon footprint importance sampling. **3.8b** (top right)**:** Classic photon map based importance sampling. **3.8c** (bottom left)**:** Importance sampling solely by means of the BSDF. **3.8d** (bottom right)**:** The photon map that has been used for importance sampling and for photon map global illumination in figure 3.8a-c.

Chapter 4

Interactive walkthroughs in globally illuminated glossy scenes

In this chapter we discuss how walkthroughs in scenes with globally illuminated glossy surfaces can be rendered at interactive frame-rates. After an overview of existing methods in chapter 4.1, we present our new method for hardware accelerated real-time rendering of globally illuminated soft glossy surfaces with directional light maps [HP02c].

4.1 Existing methods

The computationally most expensive part of rendering a globally illuminated scene is the generation of the global illumination solution. If a walkthrough in a static scene shall be rendered then it is not necessary to perform the expensive global illumination simulation for each frame separately. Instead, the global illumination can be computed in a preprocessing step which stores the global illumination solution in some representation that allows to efficiently render the illuminated scene later on during the walkthrough.

The illumination on a diffuse surface can be represented with an illumination map [Arv86], which is a texture map that stores the spatially varying irradiance on the surface. This information, which is generated in the global illumination simulation, is enough to correctly display diffuse surfaces from arbitrary view points, because the outgoing radiance of diffuse surfaces is independent of the viewing direction.

If globally illuminated glossy surfaces shall be displayed from arbitrary view points then information about the spatial distribution of the illumination on the surface is not enough. Here also information about the directional distribution of

the illumination is required. This can be represented in form of the incoming light that hits the surface, or in form of the outgoing light that is reflected from the surface.

Light fields [LH96] and lumigraphs [GG+96] store the outgoing radiance of an object as 4-dimensional function on an image plane (light field), or on a cube that encloses the object (lumigraph). The outgoing radiance may also be stored directly at the surfaces of the object with surface light fields [MRP98, WA+00], with wavelets [SS+00], or with eigen-textures [NSI99].

Graphics hardware light sources may be used to represent the outgoing radiance by fitting a small number of virtual light sources (usually 8 hardware light sources are available) for each object individually, so that the resulting phong lobes represent the glossy highlights on the object as best as possible [WA+97]. Virtual light sources may also be used to display a radiosity solution by placing them at the positions of the most contributing sending patches to illuminate a receiving glossy patch [SSS95]. Here the hardware light sources also have to be set for each glossy patch individually.

The incoming radiance from far away objects may be stored in an environment map [Hei99, Hei01], which may be prefiltered for the rendering of reflections on glossy surfaces. Glossy reflections may also be rendered with an on-the-fly convolution of images of pure specular reflections [BH+99]. The incoming light may also be stored in a directional irradiance mesh [Stü98], or in a photon map which can be rendered at nearly interactive frame-rates by drawing splats of the photons on the surfaces using graphics hardware [SB97].

4.2 Directional light maps

Directional light maps support the representation of spatially and directionally variant illumination on a soft glossy surface by storing the incoming light at the surface for several incoming light directions. Each directional light map is a texture that represents the spatially varying incoming light at a surface from one of these directions. The directional light maps are generated in a global illumination simulation, eg. photon tracing, in a preprocessing step. Afterwards in an interactive walkthrough these directional light maps are used for hardware accelerated rendering of the soft glossy surfaces including their view dependent global illumination.

By using a global set of incoming light directions for the directional light maps of all surfaces in the scene, the hardware accelerated rendering can be efficiently

done with only few state switches, which avoids expensive stalls of the hardware rendering pipeline.

A directional light map $m_{s,\Psi}$ is a texture on a surface s that stores the spatially varying incoming light that s receives from an incoming light direction Ψ. The texels' values correspond to the irradiance of this light on a plane perpendicular to Ψ. For each soft glossy surface s in the scene with surface normal N_s, and for each direction $\Psi \in \Omega$ which is frontfacing to s ($0 < N_s \cdot \Psi$), a directional light map $m_{s,\Psi}$ is stored. Ω is the predefined global set of light directions which is used for all surfaces. Note that for each surface directional light maps are generated and processed during rendering for only 50% of the directions of Ω (the frontfacing ones).

For the efficiency of hardware accelerated rendering, as explained in chapter 4.2.2, it is essential that all surfaces use the same set Ω of light directions. This global set of light directions also avoids illumination discontinuities at the borders of adjacent surfaces. Such discontinuities would arise if adjacent surfaces would be illuminated from different directions, as it would be the case if each surface would have its individual set of light directions.

4.1.1 Generation of directional light maps

The directional light maps are generated in a preprocessing step. First of all, the set Ω of light directions has to be defined. This is done by selecting n_Ω uniformly distributed directions on the unit sphere [Shi92]. n_Ω is a user defined value, and determines the directional accuracy of the illumination resulting from the generated directional light maps. A larger n_Ω allows directionally more precise illumination, but requires more texture memory and rendering time to store and render the larger number of directional light maps.

Next, the directional light maps are generated with photon tracing. A large number (typically several millions) of light paths are stochastically shot from the light sources into the scene. At each hit point x of a light path at a surface s, the light path's incoming power is splatted into that directional light map of s which is directionally nearest to the light path's incoming direction Ψ_l. This is the directional light map $m_{s,\Psi}$ where $\Psi_l \cdot \Psi$ is maximum. The splat is centered at that texel of $m_{s,\Psi}$ which maps to x. The splats' size and shape are user defined, and determine the resulting spatial blurring and noise in the directional light maps. In our implementation we have used pyramidal splats which have been 1.5 texels wide. Alternatively, a more advanced photon density estimation, eg. local linear density estimation [WH+97], could be used for each directional light map

instead of the splatting. The texture resolution of $m_{s,\Psi}$ is selected proportionally to the area of the projection of s onto a plane perpendicular to Ψ.

Graphics hardware supports only texel values in the range [0,1], therefore the irradiance values have to be mapped to texel values. Let k be the directional hardware light source intensity that corresponds to the texel's irradiance on a plane perpendicular to Ψ, and let k_{max} be the directional hardware light source intensity that corresponds to the user defined maximum representable irradiance. The texel value is then

$$t = \min\left(\frac{k}{k_{max}}, 1\right). \tag{4.1}$$

The directional light maps are packed together into large textures. Usually many (small) directional light maps fit into one of these textures, thereby only needing few textures. Directional light maps with the same Ψ are preferably put into the same texture, because the directional light maps are used in the order of their Ψ during rendering.

4.1.2 Interactive rendering

During an interactive walkthrough the directional light maps of a surface s are used for hardware accelerated rendering of the view dependent global illumination on s. Each directional light map $m_{s,\Psi}$ illuminates s by modulating a directional hardware light source which shines from direction Ψ with intensity k_{max}. The view dependent contributions of the directional light maps on s are accumulated together in the image.

The whole scene is rendered in the following steps:

- Clear the z-buffer and color buffer.
- Draw the scene with correct visibility into the z-buffer, without modifying the color buffer.
- For each $\Psi \in \Omega$:
  - Set a single directional light source, with its direction=Ψ, and with its intensity=k_{max}.
  - For each surface s for which a directional light map $m_{s,\Psi}$ exists (the $m_{s,\Psi}$ of Ψ are stored in a list for that):
    - Set the texture that contains $m_{s,\Psi}$ as current drawing texture if it is not currently set.
    - Set the material of s as current drawing material if it is not currently set.
    - Draw s only in those pixels where it is visible according to the z-buffer, illuminated by the directional light source, and modulated by $m_{s,\Psi}$, and add the resulting fragments to the color buffer.

Note that the number of state switches is minimized, because usually many directional light maps are drawn before the direction of the light source, the current texture, or the current material has to be changed. This is important for the performance, because state switches cause expensive stalls of the hardware rendering pipeline. If each surface would have its individual set of light directions, then the light source direction would have to be changed for each directional light map, which would cause very many stalls.

4.2 Results

Figure 4.1 shows a snapshot of an interactive walkthrough that has been rendered with directional light maps. A large part of the scene consists of soft glossy surfaces with specular exponent 10 (floor, table, chairs). Only a small part of the scene (ceiling, back wall, pillars) receives direct illumination, the rest of the scene is completely indirectly illuminated.

We have used n_Ω=14 and k_{max}=2. The directional light maps have required 9 MB texture memory. In total 77,000,000 light paths (3,500,000 light paths per CPU) have been shot in a parallel implementation of the photon tracing preprocessing step, which took 9.6 minutes on a cluster of 11 PCs with dual 1 GHz Pentium3s in a 100 MBit Ethernet network. In this parallel implementation each CPU generates directional light maps for the whole scene. The directional light maps from all CPUs are then accumulated to achieve the final directional light maps which are used in the interactive walkthrough.

During the interactive walkthrough the scene has been rendered at a frame-rate of 41-53 Hz on a PC with a 900 MHz Thunderbird CPU and a GeForce2 GTS graphics card with 32 MB frame buffer and texture memory under OpenGl.

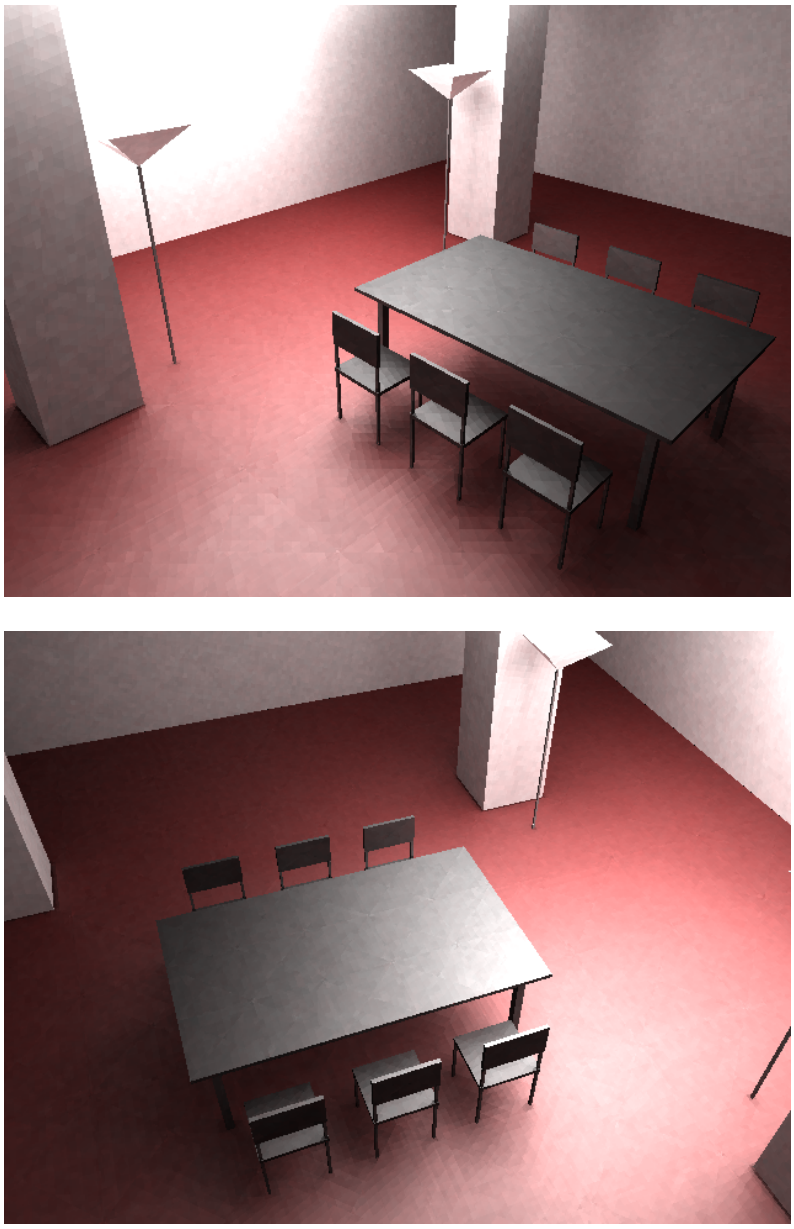

Figure 4.1: Snapshots of a walkthrough in a globally illuminated soft glossy scene that has been rendered at interactive frame-rates with directional light maps.

Chapter 5

Occlusion culling for interactive walkthroughs

In this chapter we discuss how walkthroughs can be rendered at interactive frame-rates in large scenes, for example in a building, where only a small part is visible from each possible viewpoint due to occlusion. In chapter 5.1 we review existing techniques for occlusion culling. After that, we present our new occlusion culling method with a lazy occlusion grid [HTP01], and we explain how occlusion culling can be combined with hardware accelerated rendering of directional light maps. Together these methods allow to render walkthroughs in large globally illuminated soft glossy scenes at interactive frame-rates.

5.1 Existing methods

In a large scene, for example the interior of a building, or even a whole city, usually only a small part of the scene is actually visible, as shown in figure 5.1. Therefore it would be inefficient to simply draw all the geometry of this scene, because most of the rendering time would be spent into drawing invisible objects.

This is especially a problem in real-time rendering, because available graphics hardware usually can not render the whole geometry of the scene at interactive frame-rates. Although graphics hardware is continuously becoming faster, it will probably never be fast enough, because the scenes are also becoming more complex.

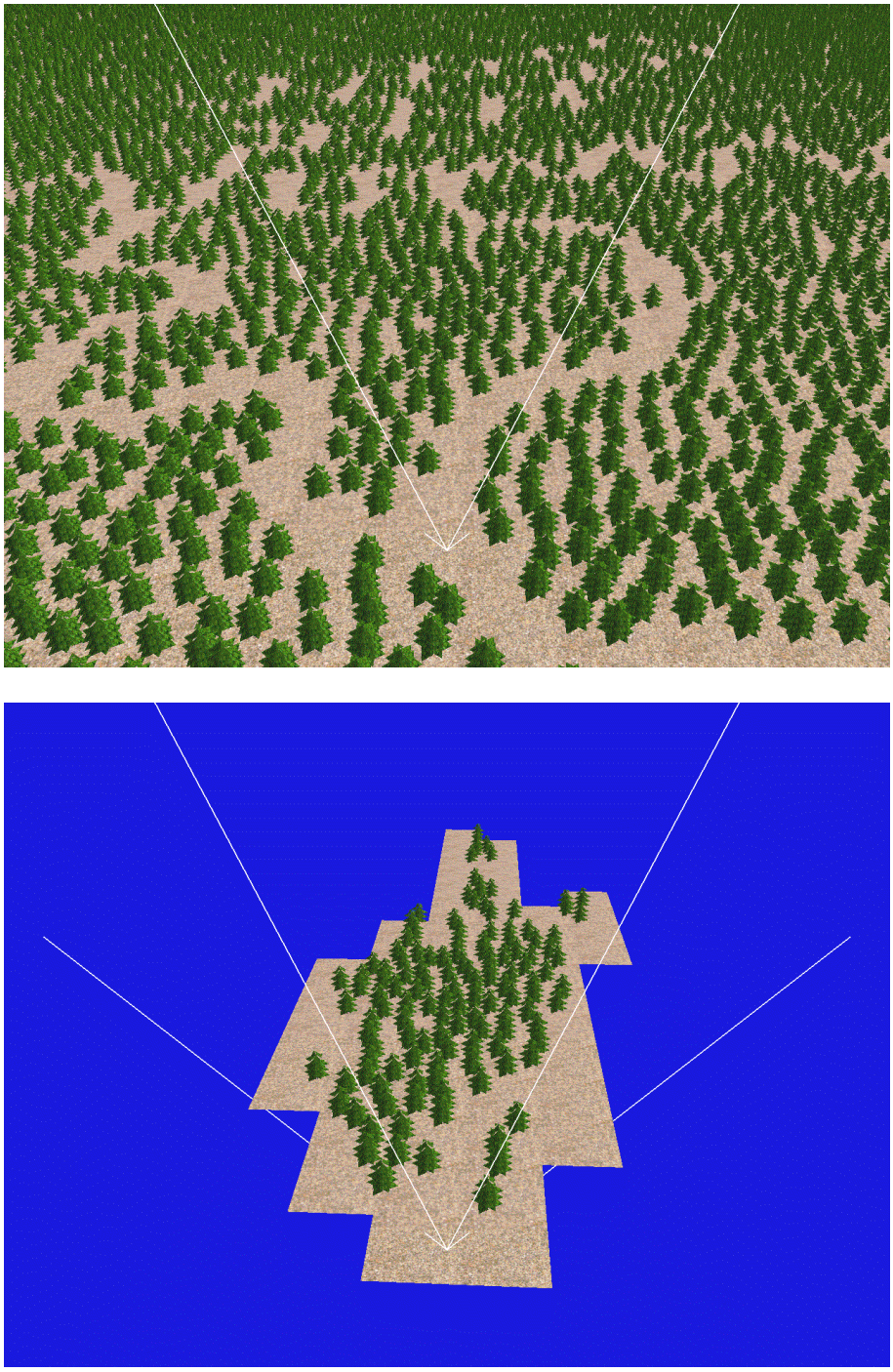

Figure 5.1: Only a small part (bottom) of a large scene (top) is potentially visible from the actual viewing postion and must therefore be drawn. The rest of the scene is invisible and can therefore be culled. The viewing frustum is visualized as wireframe.

The number of rendered primitives can be reduced to some extent by usage of hierarchical view frustum culling and back face culling. View frustum culling [AM00, Cla76, MH99] determines which parts of the scene are outside of the viewing frustum. These objects are definitely invisible and can therefore be culled. This is shown in figure 5.2.

In large scenes view frustum culling is usually applied on a spatial subdivision structure or on a hierarchy of bounding volumes of the scene, which avoids that every single primitive has to be tested separately.

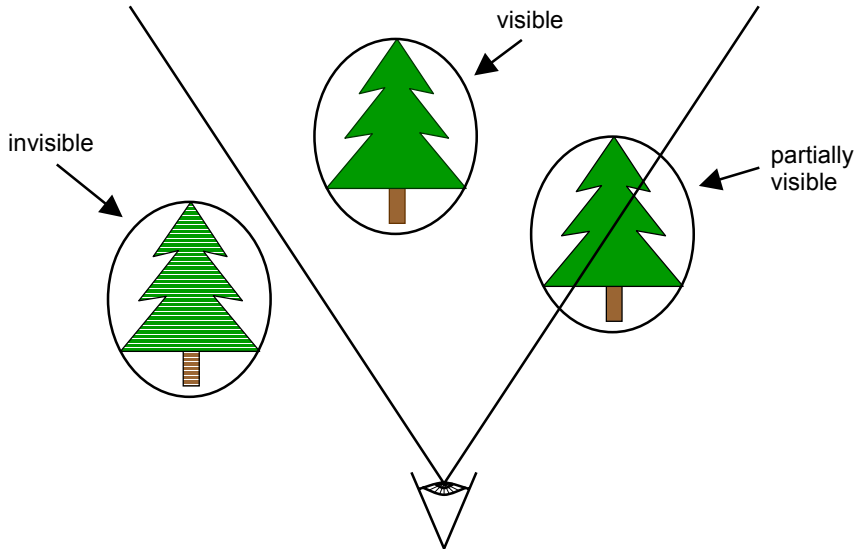

Figure 5.2: View frustum culling with bounding volumes: An object is potentially visible if its bounding volume (here: ellipsoid) is at least partially inside the viewing frustum. If the bounding volume is completely outside of the viewing frustum then the object is invisible and can be culled.

Back face culling [MH99] determines which polygons are facing away from the viewer. These polygons are invisible and can be culled, as shown in figure 5.3. Back face culling can also be done hierarchically [KM+96] so that not every single polygon has to be tested separately.

Nevertheless, only with hierarchical view frustum culling and back face culling in many scenes the number of primitives that would have to be drawn would still be too high.

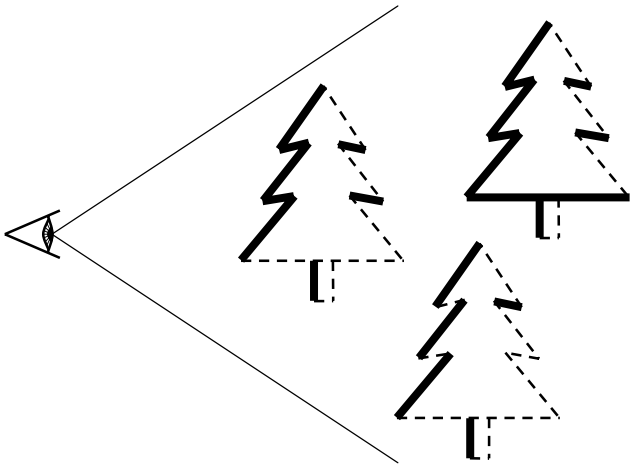

Figure 5.3: Backface culling: Polygons that are facing away from the viewer are invisible and can be culled.

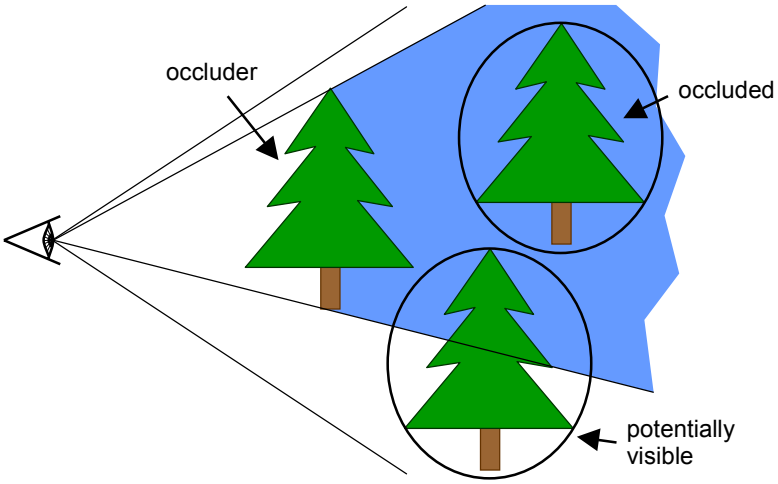

Figure 5.4: Occlusion culling with bounding volumes: An object is invisible and can be culled if its bounding volume (here: ellipsoid) is completely occluded by other objects in front of it. Otherwise the object is potentially visible and has to be drawn.

This problem can be solved by usage of occlusion culling methods [HP01] which try to determine those parts of the scene that are invisible due to occlusion by other parts. Only those parts of the scene which are potentially visible have to be rendered. Parts which are identified as occluded are culled so that they do not have to be processed further. This is shown in figure 5.4.

Many occlusion culling methods use a hierarchical representation of the scene. This allows that the occlusion testing and culling can be done for large parts of the scene at once without having to test all their sub-parts individually. For example if we already know that a building in a city is occluded from the current viewpoint then we do not have to test the occlusion of each object inside the building, because these objects are of course also occluded.

Exact global visibility methods try to solve the visibility problem by determining all visibility events in the scene for all possible viewpoints [DDP96, DDP97, PD90], but due to their complexity they are impractical for large scenes.

The high expense of exact global visibility calculations can be avoided by overestimating the set of visible objects. Most occlusion culling methods do not solve exact visibility. They return objects which are potentially visible, which includes not only those primitives that are completely visible, but also objects that are only partially visible and maybe even some objects that are completely invisible.

The exact visibility of these potentially visible objects is then calculated with an additional visibility technique. In most cases the z-buffer of conventional graphics hardware is used for the exact visibility determination.

Nowadays exist a large number of solutions that realize occlusion culling in different ways. These occlusion culling methods can be categorized according to several different criteria. In the following chapters we describe the most important of these characteristics to give an overview of occlusion culling methods for real-time rendering.

5.1.1 Visibility from region or from viewpoint

Occlusion culling methods have to determine the set of objects that are potentially visible (potentially visible set (PVS)) from the current viewpoint. This can be done for this single viewpoint alone (see figure 5.5), but it can also be done for a entire region (cell) in space [ARB90] (see figure 5.6).

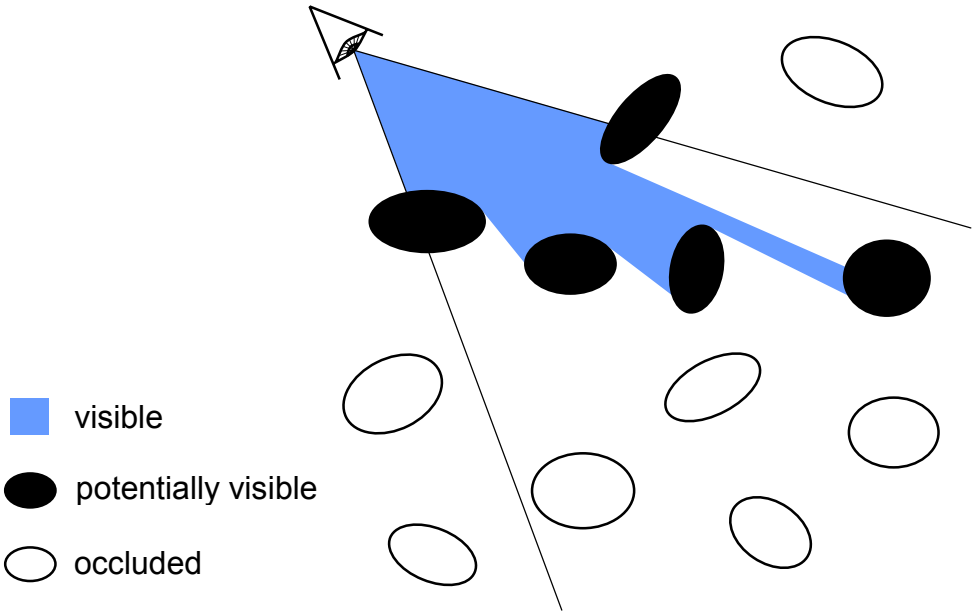

Figure 5.5: From-point visibility: Everything in the blue region is visible from the current viewpoint. The black objects are therefore potentially visible, and the white objects are occluded.

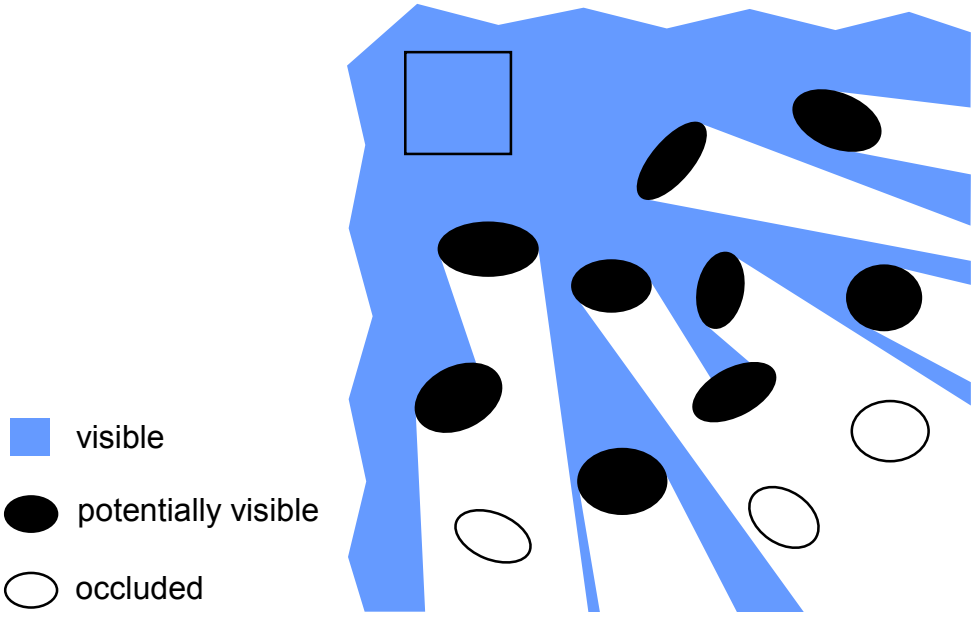

Figure 5.6: From-region visibility: Everything in the blue region is visible from at least one viewpoint within the square cell. The black objects are therefore potentially visible, and the white objects are occluded.

In the latter case the generated potentially visible set consists of all those objects which are completely or partially visible from at least one viewpoint in the region. From-region visibility methods use the coherence of visibility of the viewpoints in a region. They also distribute the computational expense of the visibility calculations over all the viewpoints in the region.

On the other hand the potentially visible set that is returned from a from-point visibility method can be noticeably smaller than the one from a from-region visibility method, because the visibility has to be calculated only for a single viewpoint.

5.1.2 Visibility calculations in a preprocessing step or on the fly

Occlusion culling methods can do their visibility calculations in a preprocessing step that precedes the rendering of the images, or it can be done on-the-fly during rendering.

Methods like for example the technique by Law et al. [LT99], the visibility octree method [SNB99], or the technique by Wang et al. [WBP98], which use a preprocessing step for their visibility calculations, subdivide the static scene into cells. In the preprocessing step these methods calculate the potentially visible set of each of the cells.

These methods are therefore from-region visibility methods, and the potentially visible set of a cell consists of all those objects which are completely or partially visible from at least one viewpoint in the cell.

During the rendering-phase these methods only have to render the objects from the potentially visible set of the cell in which the current viewpoint is located. Therefore these methods have the advantage that their rendering-phase is usually very fast because the potentially visible objects can be rendered without any further occlusion culling overhead.

But the visibility preprocessing also has some disadvantages:

- The visibility precomputation usually requires between several minutes and several hours depending on the complexity of the scene. This makes it impossible to immediately render a scene after it has been modified.

- The memory requirements for the precomputed visibility information can easily become very large. Compression techniques [PS99] can be used to minimize this memory consumtion.

- Only static objects can be used as occluders for the generation of the potentially visible sets in the visibility preprocessing step. Occlusion by dynamic objects is not taken into consideration in the precomputed visibility information.

Visibility preprocessing methods are often used in games [Abr96], because the frame-rate of the released product is the major criterion, and the cost of the time-expensive visibility precomputation after modeling is only secondary.

The typical scenes of these games consist of a large static environment and several dynamic objects which are relatively small. Usually the static environment can be designed in a way that only a very small portion of the scene is visible from each possible viewpoint. Due to their small size in the image the dynamic objects usually do not have to be considered as important occluders. Therefore the visibility preprocessing works quite well.

Most occlusion culling methods that do their visibility calculations on the fly during rendering have the following advatages:

- No time-expensive visibility precomputation is needed. This makes these methods suitable for applications where the scene has to be instantly displayed after it has been modified, for example during modeling, or for scenes that can be interactively manipulated by the user.

- Dynamic objects can be used as occluders. Apart from using temporal coherence (see chapter 5.1.13) on-the-fly occlusion culling methods do not have to distinguish between static and dynamic objects, because visibility is computed for the entire actual scene at the given point in time of the image.

Exceptions are occlusion culling techniques like virtual occluders [KCC00] or directional discretized occluders [BKE00] which do a part of their visibility calculations on the fly during rendering, but which additionally also use a preprocessing step to generate an intermediate visibility information.

Hierarchical occlusion maps [ZM+97] also use a preprocessing step. They need it to generate a database of potential occluders. This occluder precomputation can be avoided with incremental occluder selection as it is used for incremental occlusion maps [AM01], or by combining hierarchical occlusion maps with frustum slicing [Bor00].

The disadvantage of all occlusion culling methods which are done on the fly is that their visibility calculations produce some overhead during rendering.

5.1.3 Visibility calculations in object space or image space

Occlusion culling methods can do their visibility calculations in object space or in image space. Image space methods do their visibility calculations typically on-the-fly during rendering.

Several image-based occlusion culling techniques use a hierarchical representation of their occlusion information in the image [GKM93, Gre96, Gre99, HTP01, HW99, ZM+97]. In figure 5.7 this is shown for a hierarchical occlusion map [ZM+97] as an example. The hierarchical occlusion map consists of a mipmap pyramid of grayscale images. The intensity of a pixel in this pyramid represents the percentage of underlying pixels that are occluded in the full resolution image.

These hierarchical representations allow efficient culling in large occluded image areas, because occluded objects can be culled with a few accesses to entries in the high levels of the occlusion information hierarchy instead of having to access a much larger number of low level entries.

5.1.4 Continuous or point sampled visibility

Continuous visibility methods determine the visibility in all view directions that pass through the image, which is an infinitely large set of view directions. In contrast to that are point sampled visibility methods which determine the visibility only for a limited set of view directions, for example for the centers of all pixels in the image. Point sampling can also be used for object space occlusion culling [GSF99].

5.1.5 Conservatism of visibility

Most occlusion culling methods return conservative visibility information. This means that their returned set of potentially visible objects contains at least all objects that are completely or partially visible.

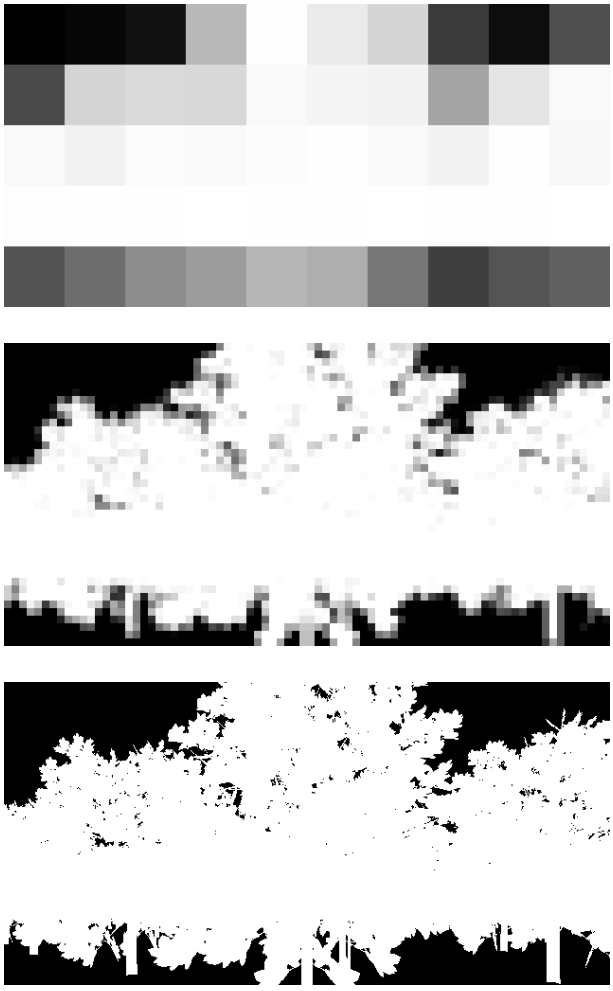

Figure 5.7: The hierarchical occlusion map represents the occlusion in the image hierarchically at several resolutions.

Several methods support non-conservative occlusion culling, which means that they do not guarantee that their returned potentially visible set contains all objects that are completely or partially visible. This increases the performance, but of course it causes artifacts in the image.

Non-conservative occlusion culling can be done by rendering only those objects which are most likely to be visible [KS00a], by using stochastic point sampled visibility in object space [GSF99], by testing only a few of the pixels of a bounding volume [BMH99], or by culling objects which are visible only in a few pixels [ZM+97].

5.1.6 Hardware acceleration

Especially for image space occlusion culling methods it is very important whether they support some kind of hardware acceleration to enhance the performance of their visibility calculations.

Hierarchical occlusion maps [ZM+97] read the frame buffer from the graphics hardware and use it to generate a pyramid of occlusion maps. They also use the mipmap functionality of the graphics hardware to generate the downsampled versions of the occlusion map.

An opacity map [HW99] can be used instead of a hierarchical occlusion map. Whereas the the hierarchical occlusion map resembles a pyramid of mipmaped textures, the opacity map corresponds to a summed area table which allows to perform an overlap test in constant time. The generation of the summed area table has to be done in software.

The hierarchical z-buffer [GKM93] requires a specialized graphics hardware for full efficiency. A variant of the hierarchical z-buffer for parallel architectures has been implemented for Pixel-Planes 5 [Geo95]. An optimized version of the hierarchical z-buffer has been proposed [Gre99] that allows to integrate a hierarchical z-buffer stage into the rendering pipeline of conventional graphics hardware.

Adaptive hierarchical visibility [XS99] is a simplified one layer version of the hierarchical z-buffer where bucket sorted polygon bins are rendered and occlusion tested. It is simpler to implement in graphics hardware than the hierarchical z-buffer.

Hierarchical coverage masks [Gre96] are very efficient in comparison to a software implementation of a conventional scanline rasterizer, but unfortunately hardware acceleration is very limited because conventional graphics hardware can only be used for texturing and shading.

Occlusion queries are currently implemented in graphics hardware on a few systems, for example on Hewlett-Packard's Visualize fx series of graphics accelerators [Hp00, SOG98, Sev99], on Silicon Graphics Visual Workstations [BBY99], and on Nvidia's GeForce3 [Nv01].

These occlusion queries test the occlusion of a given bounding volume by rasterizing it without modifying any buffer. The occlusion query then returns whether the bounding volume passed the z-test (is visible) in any of its pixels.

The relative cost of such an occlusion query in comparison to the cost of drawing triangles varies between different hardware [Sev99].

Such an occlusion query is used for example in the conservative prioritized-layered projection algorithm [KS00b] to cull occluded cells from the set of cells that are candidates for projection. Hardware accelerated occlusion queries could also be extended to return additional occlusion information and to work in parallel [BMH98].

A similar occlusion query can also be implemented on systems where such an occlusion query is not implemented in the graphics hardware. This can be done by usage of the stencil buffer [BMH99]. The write access to the color buffer and the z-buffer is disabled, and then the bounding volume is drawn. For each pixel a bit in the stencil buffer is set to 0 or 1 corresponding to whether the bounding volume passed the z-test in that pixel or not.

After that the pixels in the bounding volume's image area must be read from the stencil buffer, and it must be tested in software if their stencil bits are 0 or 1. Reading the stencil buffer and testing it in software is of course a significant performance bottleneck.

5.1.7 Occluder selection

Occlusion culling methods can use all objects as occluders, or they can select a set of objects and use only these objects as occluders. Using all objects as occluders has the advantage that it maximizes the occlusion, but several occlusion culling methods require to use a selected set of occluders.

Such methods [BHS98, CT97, DMA00, HM+97, ZM+97] select the occluders heuristically based on the assumption that these objects occlude large parts of the scene. In cases where these heuristics do not work will large parts of the scene remain unoccluded.

Additionally simplified representations of the occluders can be synthesized, for example by using sets of boxes that are enclosed by the original geometry of the occluders [ASN00]. This allows to increase the performance of visibility tests.

5.1.8 Occluder fusion

Not only the number and distribution of the occluders is important, but also the ability of the occlusion culling method to support occluder fusion. This means

that several (small) occluders together can occlude parts of the scene that the single occluders alone would not occlude if they were used independently of each other, which is illustrated in figure 5.8 and 5.9.

Occluder fusion is supported for example by the directional discretized occluders method [BKE00], by the extended projections method [DD+00], by the conservative volumetric visibility method [SD+00], or in the visibility preprocessing method for urban walkthroughs by Wonka et al. [WWS00].

Point sampled image space occlusion culling methods implicitly support occluder fusion [GKM93, Gre96, HTP01, HW99, SOG98, ZM+97], because in their image space occlusion information they do not distinguish between different occluders. Therefore the occluders are automatically combined without having to do additional computations for the occluder fusion.

Occluder fusion is very important for occluders like trees, because each single leaf of a tree usually occludes only very few objects, if any, behind it. But all the leafs of the tree together can represent an important occluder that occludes many objects behind it.

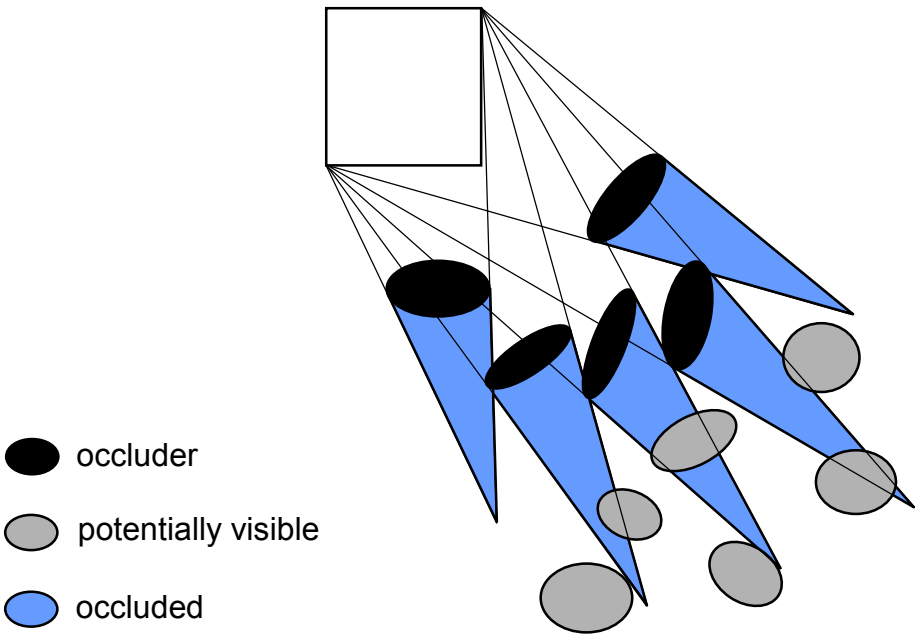

Figure 5.8: The single occluders occlude only a small region (blue) of the scene if they are used independently of each other.

Occlusion culling methods which do not support occluder fusion can usually only be used efficiently in restricted scenes which contain objects that are large enough to represent strong occluders [CF+98].

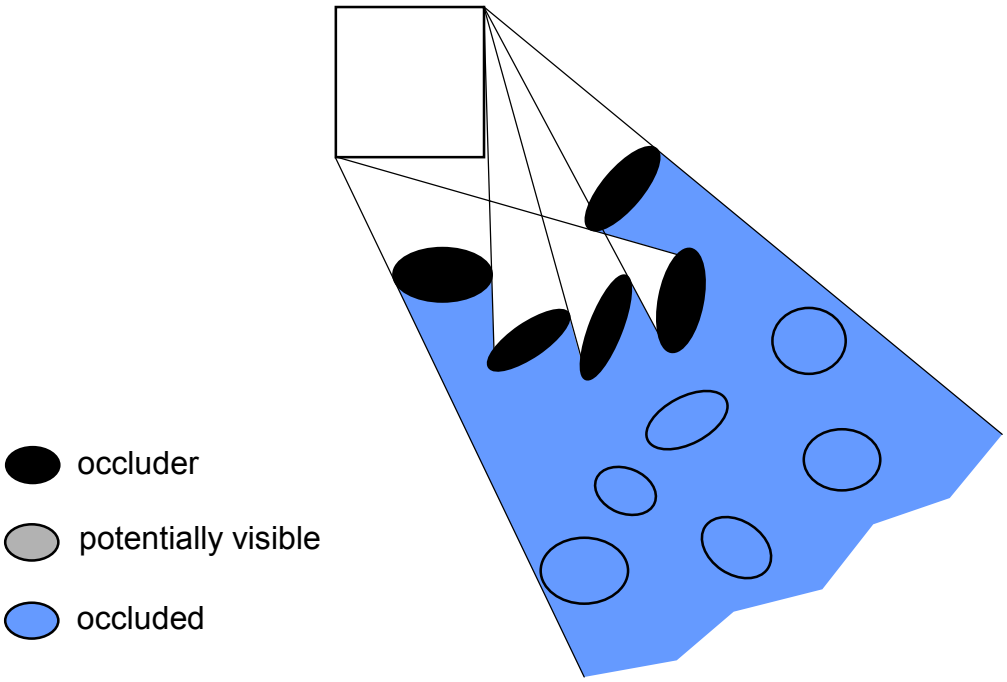

Figure 5.9: Together the same set of occluders as in figure 5.8 occludes a considerably larger region (blue) of the scene.

5.1.9 Supported scenes

Although it is desirable to support general scenes, many occlusion culling methods are nevertheless restricted to certain types of scenes. Visibility precomputation methods are restricted to mainly static scenes. Occlusion culling methods which use portals for their visibility calculations [ARB90, LG95, TS91] usually require architectural environments.

Several methods are restricted to terrains [Ste97, ZE01] which are usually based on a height field, and several other methods are restricted to walkthroughs in urban environments [WGS99], or to 2½D scenes [ST97, WS99] which are modeled on a ground plan.

But of course also the (in)ability of a method to use all objects as occluders and to support occluder fusion decides whether the method is suitable for general scenes or not.

5.1.10 Traversal of the scene

Many occlusion culling methods require that the scene is traversed in a front to back order to make efficient occlusion culling possible. This means that objects which are nearer to the viewpoint are processed before objects that are farther away, so that the nearer objects can occlude the objects behind them.

It is also important to distinguish whether the method requires a certain front to back traversal of the scene, or if the scene can be traversed in any approximative front to back order.

5.1.11 Supported bounding volumes/spatial subdivision structure

Several occlusion culling methods require that a certain kind of bounding volumes or spatial subdivision structure is used, for example a regular spatial grid [YR96]. Several methods use a BSP-tree [FKN80] as spatial subdivision structure in object space [BHS98, DMA00, Nay95], or also to subdivide the image [Nay92].

The hierarchical z-buffer [GKM93] has been proposed in combination with an object space octree, but in principle it can also be used with other space subdivision structures or with hierarchies of bounding volumes.

5.1.12 Temporal coherence

The successive frames of a walkthrough or an animation usually have a high temporal coherence. In the context of occlusion culling this means that it is likely that objects which have been visible/hidden in the previous frame will still be visible/hidden in the current frame.

Some occlusion culling methods utilize temporal coherence [GP95] to enhance their efficiency. This can be done for example by caching the objects' occlusion relations of the previous frame and reusing them in the current frame [CT97].

The hierarchical z-buffer [GKM93] can be initialized by rendering all visible objects of the previous frame before the hierarchical z-buffer is used for visibility testing. After this initialization the hierarchical z-buffer will usually contain most of the visible objects of the current frame.

Temporal coherence can also be utilized by usage of temporal bounding volumes [SG96, SG99], as shown in figure 5.10. A temporal bounding volume encloses a dynamic object not only at a single point in time. Instead it encloses the object at every position that the object has during a time interval.

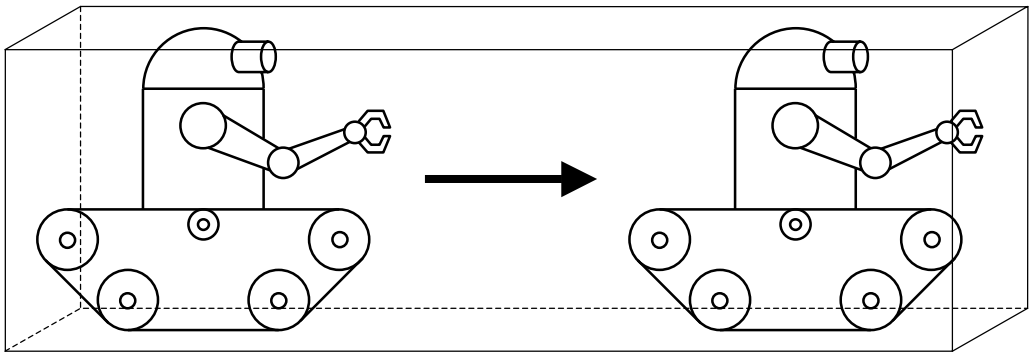

Figure 5.10: A temporal bounding volume encloses a dynamic object at every position during a time interval.

5.2 Lazy occlusion grid

In this chapter we present a new conservative image-space occlusion culling method for general scenes that works efficiently on today's available conventional graphics hardware. It does its occlusion calculations on the fly during rendering, and it does not require time-expensive visibility-preprocessing.

The image is subdivided into a low-resolution grid of tiles. Each tile stores occlusion information that shows whether the image area of the tile is occluded by already drawn objects, and a flag whether this occlusion information is outdated. This allows to determine efficiently whether an object is occluded by already drawn objects, or if it is potentially visible by querying the occlusion information of the few tiles in the object´s image area instead of testing the content of the underlying hardware z-buffer for every pixel in this image area. Occlusion testing of an object with the lazy occlusion grid works as follows:

- Test whether the bounding volume of the object is occluded or potentially visible.

- If this test decides that the bounding volume is potentially visible then the object is drawn conventionally with the z-buffer graphics hardware, otherwise it is culled.

- After an object has been drawn this way all tiles in its bounding volume's image area are marked as outdated so that the next occlusion test that queries the occlusion state of one of these tiles knows that it must update the occlusion state of this tile with a pixel-level occlusion query.

These steps are explained in more detail in chapter 5.2.1.

A major feature that distinguishes our method from related methods like the hierarchical z-buffer [GKM93] is that we update the occlusion information of a tile only when it is queried and if it is currently marked as outdated (*lazy update*) instead of updating it every time after an object has been drawn in its image area. This reduces the number of pixels that have to be read from the hardware z-buffer, because:

- an object is potentially visible if the first unoccluded tile is found in its image area. Therefore up-to-date tiles are queried first, which reduces the chance that outdated tiles have to be queried and updated.

- often several objects draw into a tile´s area before the tile is queried and updated.

We have chosen a flat grid instead of a pyramid [GKM93, Gre96, ZM+97] because of the low average number of tiles that have to be tested per bounding volume, as can be seen in our results in chapter 5.4. The optimal number of pixels per tile that gives the best overall-performance is system-dependent and can easily be determined by testing typical scenes of the desired application with different numbers of pixels per tile.

We use this image-space occlusion test on a bounding volume hierarchy that is traversed in a front-to-back order. If a bouding volume is rated as occluded then it is culled without having to process its sub-objects and sub-bounding volumes. This way a large part of the scene can be culled at once if its bounding volume is rated as occluded.

We propose two different versions of the lazy occlusion grid:

- The *occlusion state-version* is for systems that provide a pixel-level query which returns whether all pixels in a given image area are occluded (all their z-values are less than z_{max}, z_{max} corresponds to an unoccluded pixel). This query is already available on some of today's hardware [Hp00, SOG98]. The cost of this query compared to the cost of drawing primitives varies between different hardware [Sev99]. On other systems the query can be implemented in software if the hardware provides fast reading access to the z-buffer.

  Each tile's occlusion information consists of an occlusion state that can be:

  - *completely unoccluded* (nothing has been drawn into the tile's pixels yet).
  - *partially occluded* (something has already been drawn into some of the tile's pixels).
  - *completely occluded* (something has already been drawn into all of the tile's pixels).
  - *outdated* (the tile's occlusion state must be determined with a pixel-level query).

  A special front-to-back traversal of the bounding volumes is used so that objects are occlusion-tested and drawn in an order that guarantees correct occlusion. If an arbitrary front-to-back traversal would be used then bounding volumes could be falsely occluded by already drawn objects behind them. Note that no primitive-wise front-to-back traversal is needed, because the exact visibility of the primitives is solved by drawing them with the z-buffer graphics hardware.

- The *z_{far}-version* is for systems that provide a pixel-level query which returns the farthest z-value of all pixels in the z-buffer in a given image area [BMH98]. On systems where this query is not available in hardware (unfortunately this is commonly the case today) it can be implemented in software if the hardware provides fast reading access to the z-buffer.

  Each tile's occlusion information consists of the farthest z-value (z_{far}) of its pixels rendered so far, and a flag that shows if z_{far} is outdated. Any appoximative front-to-back traversal [GKM93] can be used because the z_{far}-version allows to test the occlusion of a bounding volume after objects behind it have been drawn.

In chapter 5.2.1 we describe how the occlusion or potential visibility of objects is determined. Next, we explain the usage of a bounding volume hierarchy and the

front-to-back traversal of the scene in chapter 5.2.2. We also present ideas for future extensions of our method in chapter 5.2.3. After that we explain in chapter 5.3 how occlusion culling can be combined with hardware accelerated rendering of directional light maps for interactive walkthroughs in large globally illuminated soft glossy scenes. In chapter 5.4 we present our results, which includes a comparison of our new method with related occlusion culling methods.

5.2.1 Occlusion test

When we test whether a bounding volume is occluded we classify the tiles that intersect the bounding volume into two types, as shown in figure 5.11:

- *Internal tiles* are those which are completely covered by the bounding volume.
- *Border tiles* are those which only partially intersect the bounding volume.

We do this to be able to do a pixel-level query to determine whether the part of the bounding volume that intersects a border tile is occluded if the tile is only partially occluded.

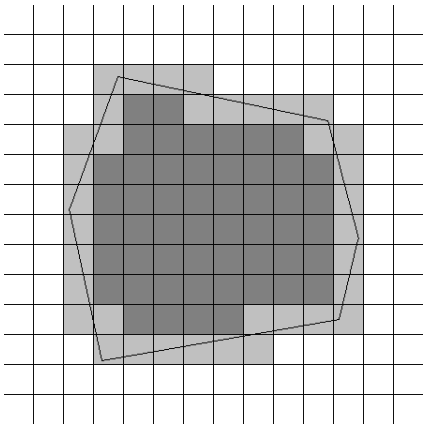

Figure 5.11: Border tiles (light gray) and internal tiles (dark gray) of the tested bounding volume.

5.2.1.1 Occlusion state-version

Initially (before any object is drawn or any bounding volume is tested) the z-buffer is cleared, and all tiles are set to completely unoccluded-state. The test

whether a bounding volume is occluded works as follows in the occlusion state-version:

- Test if one of the bounding volume's up-to-date internal tiles is not in completely occluded-state. If this is true then the bounding volume is potentially visible.

- After that, but only if we not already have potential visibility, test if one of the bounding volume's up-to-date border tiles is in completely unoccluded-state. If this is true then the bounding volume is potentially visible.

- Next, but only if we not already have potential visibility, test each outdated internal tile of the bounding volume with a pixel-level query in the tile's whole area whether the tile is completely occluded or partially occluded, and set the tile's state. If one of these tiles is partially occluded then the bounding volume is potentially visible.

- At last, but only if we not already have potential visibility, test each border tile of the bounding volume which is outdated or in partially occluded-state with a pixel-level query in the intersection area of the tile and the bounding volume. If the pixel-level query returns that the intersection area is not completely occluded then the bounding volume is potentially visible.

The pseudocode of this occlusion test is outlined in figure 5.12. We first try to determine potential visibility by using solely the tiles' states (in the …TileIsUnoccludedWithoutPixelquery functions). Only if this does not result in potential visibility we have to use the more expensive pixel-level occlusion queries (in the …TileIsUnoccludedWithPixelquery functions). The lazy update of outdated tiles is done in the …TileIsUnoccludedWithPixelquery functions.

5.2.1.2 Z_{far}-version

The z_{far}-version works similar to the occlusion state-version. The pseudocode of the occlusion test of the z_{far}-version is outlined in figure 5.13. Initially (before any object is drawn or any bounding volume is tested) the z-buffer is cleared, all tiles' z_{far} are set to z_{max}, and their outdated-flags are cleared.

In contrast to the occlusion state-version the z_{far}-version compares whether the nearest z-value (z_{near}) of the bounding volume is larger than the tile's z_{far} to determine if the bounding volume is occluded or potentially visible in the tile´s area. An outdated tile´s z_{far} is updated with a pixel-level query that returns the

farthest z-value of all pixels in the tile´s area. In figure 5.13 this lazy update is done in the internalTileIsUnoccludedWithPixelquery function.

The z_{far}-version presented so far needs a query that returns the farthest z-value of all pixels in the tile´s area [BMH98]. On systems where this kind of query is not implemented in hardware (unfortunately this is commonly the case today) it is possible to use another kind of query with a modified version of the z_{far}-version. In this modified version the original query for the farthest z-value is replaced with a function that uses a query whether any pixel of the bounding volume passes the z-test (is visible) [Hp00, SOG98] in the tile's area (the same kind of query that is used in the occlusion state-version). If the result of the query is true the function returns the tile´s old z_{far}-value, otherwise it returns the bounding volume´s z_{near}-value.

5.2.2 Front-to-back traversal in a bounding volume hierarchy

In chapter 5.2.1 we have described how to do occlusion culling for single objects (bounding volumes). What we need is to ensure that objects in the front are usually drawn first so that they can occlude objects behind them. This is accomplished by traversing the scene in an approximative front-to-back order.

In a scene that contains a large number of objects it would be inefficient to do the front-to-back sorting and the occlusion test for each object separately. To avoid this we use a bounding volume hierarchy for the scene that is traversed recursively. If a bounding volume is rated as occluded it can be culled without having to do the occlusion test for its sub-bounding volumes. Therefore a large occluded part of the scene can be culled at once with a single occlusion test. Any kind of bounding volume hierarchy can be used, for example an octree, kd-tree, or a hierarchy of polyhedrons or spheres. In our implementation, which is described in chapter 5.4, we have used a hierarchy of axis-aligned bounding boxes.

5.2.2.1 Occlusion state-version

The occlusion state-version of the lazy occlusion grid is used with a special approximative front-to-back traversal which performs the occlusion test of a bounding volume before objects are drawn that are not completely in front of the bounding volume. Otherwise these objects could falsely occlude the bounding volume or its sub-bounding volumes. The occlusion state-version does not need an exact front-to-back traversal of the primitives because exact visibility is solved by drawing the primitives with the z-buffer graphics hardware.

```
Bool isOccluded(bVol)
    if internalTileIsUnoccludedWithoutPixelquery(bVol,internalPixelqueryList)
        return false
    if borderTileIsUnoccludedWithoutPixelquery(bVol,borderPixelqueryList)
        return false
    if internalTileIsUnoccludedWithPixelquery(internalPixelqueryList)
        return false
    if borderTileIsUnoccludedWithPixelquery(bVol,borderPixelqueryList)
        return false
    return true

Bool internalTileIsUnoccludedWithoutPixelquery(bVol,internalPixelqueryList)
    internalPixelqueryList=empty
    for all internal tiles of bVol
        if not tile.completelyOccluded
            if tile.state=outdated
                add tile to internalPixelqueryList
            else //completelyUnoccluded or partiallyOccluded
                return true
    return false

Bool borderTileIsUnoccludedWithoutPixelquery(bVol,borderPixelqueryList)
    borderPixelqueryList=empty
    for all border tiles of bVol
        if not tile.completelyOccluded
            if tile.state=completelyUnoccluded
                return true
            else //partiallyOccluded or outdated
                add tile to borderPixelqueryList
    return false

Bool internalTileIsUnoccludedWithPixelquery(internalPixelqueryList)
    for all tiles in internalPixelqueryList
        if the z-value of a pixel in the tile's area =z_max
            tile.state=partiallyOccluded
            return true
        else
            tile.completelyOccluded=true
    return false

Bool borderTileIsUnoccludedWithPixelquery(bVol,borderPixelqueryList)
    for all tiles in borderPixelqueryList
        if the z-value of a pixel in the area of (tile ∩ bVol) =z_max
            tile.state=partiallyOccluded
            return true
    return false
```

Figure 5.12: Pseudocode of test whether a bounding volume is potentially visible or occluded by already drawn objects inclusive the lazy update of the tiles' states (occlusion state-version of the grid).

```
Bool isOccluded(bVol)
    if internalTileIsUnoccludedWithoutPixelquery(bVol,internalPixelqueryList)
        return false
    if borderTileIsUnoccludedWithoutPixelquery(bVol,borderPixelqueryList)
        return false
    if internalTileIsUnoccludedWithPixelquery(bVol,internalPixelqueryList)
        return false
    if borderTileIsUnoccludedWithPixelquery(bVol,borderPixelqueryList)
        return false
    return true

Bool internalTileIsUnoccludedWithoutPixelquery(bVol,internalPixelqueryList)
    internalPixelqueryList=empty
    for all internal tiles of bVol
        if bVol.z_near≤tile.z_far
            if tile.outdated
                add tile to internalPixelqueryList
            else
                return true
    return false

Bool borderTileIsUnoccludedWithoutPixelquery(bVol,borderPixelqueryList)
    borderPixelqueryList=empty
    for all border tiles of bVol
        if tile.outdated or bVol.z_near≤tile.z_far
            add tile to borderPixelqueryList
        else if tile.z_far=z_max
            return true
    return false

Bool internalTileIsUnoccludedWithPixelquery(bVol,internalPixelqueryList)
    for all tiles in internalPixelqueryList
        tile.outdated=false
        tile.z_far=max z-value of all pixels in tile´s area
        if bVol.z_near≤tile.z_far
            return true
    return false

Bool borderTileIsUnoccludedWithPixelquery(bVol,borderPixelqueryList)
    for all tiles in borderPixelqueryList
        if bVol.z_near≤max z-value of all pixels in the area of (tile ∩ bVol)
            return true
    return false
```

Figure 5.13: Pseudocode of test whether a bounding volume is potentially visible or occluded by already drawn objects inclusive the lazy update of the tiles' states (z_{far}-version of the grid).

This approximative front-to-back order is illustrated in figure 5.14. The front-to-back traversal that realizes this order is outlined in figure 5.15. It draws the frontmost object if it is completely in front of all bounding volumes that are not occlusion-tested yet, and it tests the frontmost bounding volume for occlusion if no object is completely in front of it. To do this the traversal uses two lists:

- bounding volumes that are not occlusion-tested yet are sorted by their respective nearest z-value (z_{near}). Initially this *test-list* contains the root bounding volume.

- bounding volumes that are already rated as potentially visible and that have objects as direct children are sorted by their respective farthest z-value (z_{far}). Initially this *draw-list* is empty. In figure 5.14 the objects themselves are shown instead of the bounding volumes in the draw-list for sake of simplicity of the illustration.

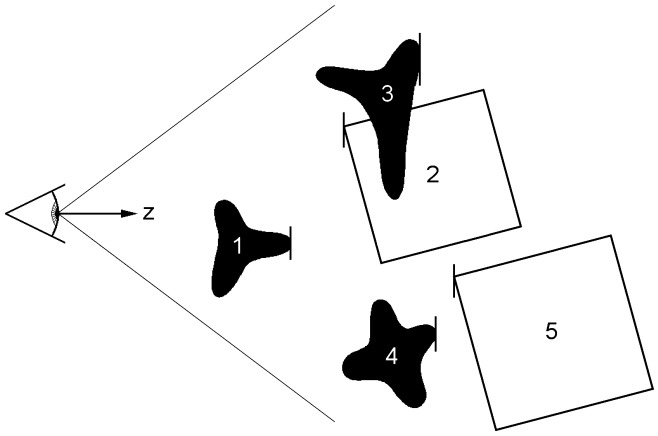

Figure 5.14: Front-to-back traversal for the occlusion state-version of the grid: z_{near}-sorted bounding volumes (white) are occlusion-tested before z_{far}-sorted potentially visible objects (black) are drawn that are not completely in front of them. z_{near}/z_{far} is marked at each bounding volume/object, sub-objects and sub-bounding volumes of the bounding volumes are not shown.

5.2.2.2 Z_{far}-version

The z_{far}-version of the lazy occlusion grid can be used with an arbitrary front-to-back traversal because this version of the grid allows to draw objects before the occlusion test of bounding volumes which are (partially) in front of them is done. A simple front-to-back traversal for the z_{far}-version is outlined in figure 5.16. It utilizes a list of bounding volumes which are sorted by their respective nearest z-

value (z_{near}). Note that a heap could be used as well instead of the list. Note also that we distinguish between sub-objects (no bounding volumes) and sub-bounding volumes.

```
initialize lazy occlusion grid
initialize test-list with root bounding volume if it intersects view frustum,
    else test-list=empty
draw-list=empty
while test-list or draw-list is not empty
    bVol=bounding volume with smallest z_near from test-list or
         bounding volume with smallest z_far from draw-list
         (depends on if z_near or z_far is smaller), remove it from its list
    if bVol is from test-list
        if isOccluded(bVol)=false
            for each of bVol´s sub-bounding volumes
                if sub-bounding volume intersects view frustum
                    sort sub-bounding volume into test-list
            if bVol has objects as direct children
                sort bVol into draw-list
    else //bVol is from draw-list
        draw objects that are direct children of bVol
        mark tiles in bVol´s image area as outdated
```

Figure 5.15: Pseudocode of front-to-back traversal that incorporates occlusion culling with the occlusion state-version of the grid and hierarchical view frustum culling

```
initialize lazy occlusion grid
initialize list with root bounding volume
while list is not empty
    bVol=frontmost bounding volume in list, remove it from list
    if bVol intersects view frustum
        if isOccluded(bVol)=false
            sort bVol´s sub-bounding volumes (if any) into list
            if bVol has objects as direct children
                draw objects that are direct children of bVol
                mark tiles in bVol´s image area as outdated
```

Figure 5.16: Pseudocode of a simple front-to-back traversal that incorporates occlusion culling with the z_{far}-version of the grid and hierarchical view frustum culling.

5.2.3 Future extensions

The extensions of our basic method which are sketched in this chapter use different conservative occlusion tests for special cases to increase the overall-performance. Heuristics are used to decide if these occlusion tests shall be used or not. Note that these extensions do not violate conservatism.

- Quickly rate a tile as completely unoccluded (the tested bounding volume is therefore potentially visible) instead of using a pixel-level query if the probability that the tile is occluded is smaller than a given threshold. This probability can be approximated by adding up the size of the image-areas of the primitives that have been drawn into the tile, which for reason of speed can be done without considering if these areas are intersecting.

- Test whether a bounding volume's border tile is completely occluded if the area of intersection of the bounding volume with the tile is larger than a given threshold. In this case the overhead of querying the whole tile area instead of querying only the smaller intersection area is not so big and we have the chance to detect that the tile is completely occluded.

- Treat border tiles as internal tiles if the image area of the bounding volume is larger than a given threshold, because the image position of such a large bounding volume's border is often quite different to the image position of the real object´s border.

5.3 Occlusion culling and directional light maps

In combination with occlusion culling the hardware accelerated directional light map rendering algorithm from chapter 4.2.2 is not applied to the whole scene at once. Instead, the scene is traversed by the occlusion culling algorithm, and for each potentially visible bounding volume the following modified version of the directional light map rendering algorithm is used:

- Draw the surfaces that are contained in the bounding volume with correct visibility and with black surface color into the z-buffer and color buffer.
- For each $\Psi \in \Omega$:
  - Set a single directional light source, with its direction=Ψ, and with its intensity=k_{max}.
  - For each surface s that is contained in the bounding volume, and for which a DLM $m_{s,\Psi}$ exists (the bounding volume's $m_{s,\Psi}$ of Ψ are stored in a list for that):
    - Set the texture that contains $m_{s,\Psi}$ as current drawing texture if it is not currently set.
    - Set the material of s as current drawing material if it is not currently set.
    - Draw s only in those pixels where it is visible according to the z-buffer, illuminated by the directional light source, and modulated by $m_{s,\Psi}$, and add the resulting fragments to the color buffer.

The z-buffer and the color buffer are only cleared once for the whole scene at the beginning of the rendering of each frame. Directional light maps from the same bounding volume are preferably put into the same texture to minimize the number of state switches where the current texture has to be changed.

5.4 Results

We have implemented and tested occlusion culling with the lazy occlusion grid in combination with directional light maps on a PC with a 900 MHz Thunderbird CPU and a GeForce2 GTS graphics card with 32 MB frame buffer and texture memory under OpenGL. We had no access to graphics hardware that supports pixel-level occlusion queries, therefore we have implemented the pixel-level queries in software by reading the hardware z-buffer, which is done with the glReadPixels function.

The size of the grid´s tiles is 32x32 pixels per tile, and has been determined heuristically as described in chapter 5.2. Of course on other systems the optimal

tile-size may be different. We have measured that our hardware does 34,783 glReadPixels of 32x32 z-values (35,617,792 pixels) per second without flushing the rendering pipeline.

The scene in figure 5.17 consists of 4x4 of the rooms which have been used in figure 4.1. The rooms are connected by doorways. From each possible viewing position only a small part of the scene is visible due to occlusion. The scene contains 19,520 triangles. Directional light maps are used to display the globally illuminated soft glossy scene during an interactive walkthrough. Rendering the whole scene without occlusion culling is therefore expensive not because of the geometrical complexity of the scene, but because of the overhead of displaying the global illumination on the large part of surfaces that are occluded.

We have used n_Ω=14 and k_{max}=2. The directional light maps have required 142 MB texture memory. In total 1,232,000,000 light paths (56,000,000 light paths per CPU) have been shot in a parallel implementation of the photon tracing preprocessing step, as described in chapter 4.2, which took 2.6 hours on a cluster of 11 PCs with dual 1 GHz Pentium3s in a 100 MBit Ethernet network.

Note that the memory size of the directional light maps is considerably larger than the available on-board texture memory of our used graphics hardware. The directional light maps therefore had to be transmitted from main memory to the graphics card on demand. Although there are already systems available with enough on-board texture memory to store the directional light maps of this scene locally on the graphics card, it is very likely that there will always be applications with scenes that require considerably more texture memory than the available graphics hardware can offer.

We have used axis-aligned bounding boxes, one bounding box for each room. The image area of a bounding box is approximated by its bounding rectangle in the image. The bounding boxes are traversed with the front-to-back traversal as described in chapter 5.2.2.1. Note that this traversal also incorporates hierarchical view frustum culling of the bounding boxes [MH99], which uses simple clipping of the bounding boxes' polygons in software.

We have tested the scene with a walkthrough that has been rendered

- with occlusion culling with the occlusion state-version of the lazy occlusion grid (LOG).

- with occlusion culling with the occlusion state-version of the occlusion grid, but each tile is immediately updated after an object has been drawn into its

image area if the tile has not already been marked as completely occluded (busy occlusion grid (BOG)).

- with occlusion culling with the hierarchical z-buffer (HZB) [GKM93]. After an object has been drawn the conventional z-buffer in the image area of the bounding box is read to update the hierarchical z-buffer.

- with occlusion culling solely with a pixel-level query per bounding box (PQ) that tests all pixels in the image area of the bounding box.

- without occlusion culling (no OC), but still with hierarchical view frustum culling.

The scene has been rendered at an image resolution of 640x480 as well as 1280x960 pixels to show to what extent image resolution affects the rendering time. The average rendering time per frame, and the average number of potentially visible surface triangles per frame are shown in table 5.1. Note that this number also includes backfacing surfaces, because backface culling is done by OpenGL. Note also that the number of triangles that are actually sent to OpenGl is ~8 times higher due to the multi-pass rendering with the directional light maps.

We have measured that with our hardware 11-18% of the total rendering time is spent for the glReadPixels calls when we use the lazy occlusion grid and 42-56% when we use the busy occlusion grid. The average number of tiles that have been tested per bounding box with the lazy occlusion grid (including those tiles where no pixel-level query is done) is 23.1 at 640x480. The average frame-rate with the lazy occlusion grid has been 1.8 to 3.1 times faster than with the busy occlusion grid, which shows the importance of the lazy update.

640x480	LOG	BOG	HZB	PQ	no OC
fps [Hz]	10.2	5.6	3.7	2.4	4.9
no. pot. visible surf. triangles	4,541	4,541	4,541	4,541	11,902

1280x960	LOG	BOG	HZB	PQ	no OC
fps [Hz]	6.8	2.2	1.8	1.3	3.6
no. pot. visible surf. triangles	4,558	4,558	4,558	4,558	11,902

Table 5.1: Average frame-rate and average number of potentially visible surface triangles per frame of the walkthrough.

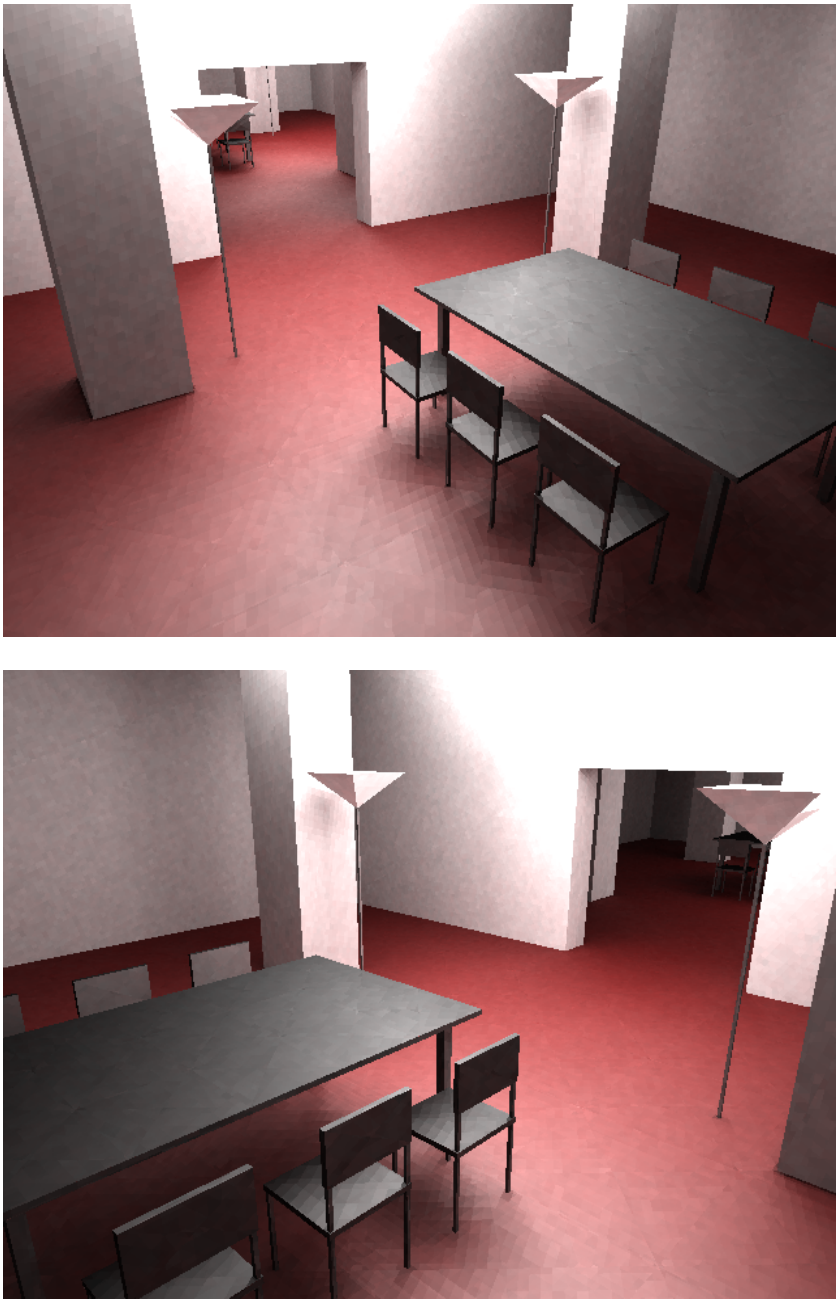

Figure 5.17: Snapshots of a walkthrough in a large globally illuminated soft glossy scene that has been rendered at interactive frame-rates with directional light maps and occlusion culling with the lazy occlusion grid.

Chapter 6

Conclusion

We have presented new methods that improve the quality and performance of photorealistic rendering of globally illuminated scenes, in particular for photon map global illumination simulation, and we have presented hardware accelerated methods that allow to do interactive walkthroughs in large globally illuminated scenes with soft glossy surfaces.

Our new geometry-based photon map radiance estimation method has improved the quality of photon map global illumination simulation by avoiding several illumination artifacts of the existing photon map radiance estimation. This is an important feature in particular for the high quality rendering of caustics, which is done by direct visualization of the photon map radiance estimate.

We have also shown that the efficiency of stochastic ray tracing-based rendering and global illumination techniques, in particular photon map global illumination simulation, can be improved with our new hemispherical particle footprint importance sampling method. The advantage of this new method is its ability to shoot the rays precisely into highly contributing directions, which is caused by the adaptive selection of the footprint radii according to the directional particle density.

Our new hardware accelerated method for displaying directional light maps has allowed to do interactive walkthroughs in globally illuminated soft glossy scenes. This method makes efficient usage of the graphics hardware by minimizing the number of state switches that are required to change the active texture and light source.

Finally we have shown that interactive walkthroughs can also be done in large globally illuminated glossy scenes. This has been realized by using our directional light map method in combination with our new occlusion culling

method. Due to the lazy manner in which the occlusion information in the lazy occlusion grid is updated, our new occlusion culling method considerably reduces the number of pixels that have to be read from the hardware z-buffer to determine occlusion. This results in significantly increased performance.

For very large scenes the memory size of the directional light maps may be larger than the available main memory of the system. Future work should therefore include the development of caching techniques that allow to load the directional light maps on demand from a mass storage device for those parts of the scene that may be visible in the next frames of the walkthrough.

References

[Abr96] Michael Abrash. *Inside Quake: Visible Surface Determination. Ramblings in Real Time*. Dr. Dobb's Sourcebook January/February 1996 pp. 41-45

[AM01] Timo Aila and Ville Miettinen. *dPVS Reference Manual*. Hybrid Holding, Ltd. www.renderware.com/dpvs.html, 2001

[ARB90] John M. Airey, John H. Rohlf and Frederick P. Brooks Jr. *Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments*. Symposium on Interactive 3D Graphics 1990 pp. 41-50

[ASN00] Carlos Andújar, Carlos Sanoa-Vázquez and Isabel Navazo. *LOD Visibility Culling and Occluder Synthesis*. Computer-Aided Design vol. 32 no. 13 2000 pp. 773- 783

[Arv86] J. Arvo. *Backward Ray Tracing*. *Developments in Ray Tracing*. Siggraph 86 course notes

[AM00] Ulf Assarsson and Tomas Möller. *Optimized View Frustum Culling Algorithms for Bounding Boxes*. Journal of Graphics Tools vol. 5 no. 1 pp. 9-22, 2000

[Bar89] H.-J. Bartsch. *Mathematische Formeln*. 22nd ed. p. 246, Fachbuchverlag Leipzig 1989

[BMH98] Dirk Bartz, Michael Meißner and Tobias Hüttner. *Extending Graphics Hardware For Occlusion Queries In OpenGL*. Proceedings of Eurographics/Siggraph Worshop on graphics hardware 1998 pp. 97-103

[BMH99] Dirk Bartz, Michael Meißner and Tobias Hüttner. *OpenGL-assisted Occlusion Culling for Large Polygonal Models*. Computers & Graphics vol. 23 no. 5 1999 pp. 667-679

[BH+99] R. Bastos, K. Hoff, W. Wynn and A. Lastra. *Increased Photorealism for Interactive Architectural Walkthroughs*. Symposium on Interactive 3D Graphics 1999

[BKE00] Fausto Bernardini, James T. Klosowski and Jihad El-Sana. *Directional Discretized Occluders for Accelerated Occlusion Culling*. Proceedings of Eurographics 2000

[BHS98] Jiří Bittner, Vlastimil Havran and Pavel Slavík. *Hierarchical Visibility Culling with Occlusion Trees*. Computer Graphics International 1998 pp. 207-219

[BLS94] P. Blasi, B. Le Saëc and C. Schlick. *An Importance Driven Monte-Carlo Solution to the Global Illumination Problem*. Proceedings of Eurographics Workshop on rendering 1994 p. 173-183

[Bor00] Karsten Bormann. *An Adaptive Occlusion Culling Algorithm for use in Large VEs*. IEEE Virtual Reality 2000 p. 290

[BBY99] Allen Bourgoyne, Renée Bornstein and David Yu. *Silicon Graphics Visual Workstation OpenGL Programming Guide For Windows NT*. Document number 007-3876-001, Silicon Graphics, 1999

[Bus97] E. Bustillo. *A neuro-evolutionary unbiased global illumination algorithm*. Proceedings of Eurographics Workshop on rendering 1997

[Chr97] P. H. Christensen. *Global Illumination for Professional 3D Animation, Visualization, and Special Effects*. Proceedings of Eurographics Workshop on rendering 97 p. 321-326

[Chr99] P. H. Christensen. *Faster Photon Map Global Illumination*. Journal of graphics tools vol. 4 no. 3 1999 p. 1-10

[Cla76] James H. Clark. *Hierarchical Geometric Models for Visible Surface Algorithms*. Communications of the ACM vol. 19 no. 10 1976 pp. 547-554

[CF+98] Daniel Cohen-Or, Gadi Fibich, Dan Halperin and Eyal Zadicario. *Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes*. Proceedings of Eurographics 1998 pp. 243-253

[CT96] S. Coorg and S. Teller. *A Spatially and Temporally Coherent Object Space Visibility Algorithm*. MIT LCS Technical Report 546, 1996

[CT97] Satyan Coorg and Seth Teller. *Real-Time Occlusion Culling for Models with Large Occluders*. ACM symposium on interactive 3D graphics 1997 pp. 83-90

[DMA00] Pavan K. Desikan, T. M. Murali and Pankaj K. Agarwal. *Occlusion Culling Using Exact Shadow Computations and Ray-Shooting Queries*. Preliminary draft, Duke University, 2000

[Dri00] T. Driemeyer. *Rendering with mental ray*. Springer-Verlag 2000

[DDP96] Frédo Durand, George Drettakis and Claude Puech. *The 3D visibility complex: a new approach to the problems of accurate visibility*. Proceedings of Eurographics Workshop on Rendering 1996 pp. 245-256

[DDP97] Frédo Durand, George Drettakis and Claude Puech. *The Visibility Skeleton: A Powerful And Efficient Multi-Purpose Global Visibility Tool*. Proceedings of Siggraph 97 pp. 89-100

[DD+00] Frédo Durand, George Drettakis, Joëlle Thollot and Claude Puech. *Conservative Visibility Preprocessing using Extended Projections*. Proceedings of Siggraph 2000 pp. 239-248

[DW94] P. Dutré and Y. D. Willems. *Importance-driven Monte Carlo Light Tracing*. Proceedings of Eurographics Workshop on rendering 1994 p. 185-194

[DW95] P. Dutré and Y. D. Willems. *Potential-driven Monte Carlo Particle Tracing for Diffuse Environments with Adaptive Probability Functions*. Proceedings of Eurographics Workshop on rendering 1995 p. 339-348

[FKN80] Henry Fuchs, Zvi M. Kedem and Bruce F. Naylor. *On visible surface generation by a priori tree structures*. Proceedings of Siggraph 80 pp. 124-133

[Geo95] Chris Georges. *Obscuration Culling on Parallel Graphics Architecures*. Technical Report TR95-017, UNC-Chapel Hill, 1995

[GG+96] S. J. Gortler, R. Grzeszczuk, R. Szeliski and M. F. Cohen. *The Lumigraph*. Proceedings of Siggraph 96 p. 43

[GSF99] Craig Gotsman, Oded Sudarsky and Jeffrey A. Fayman. *Optimized occlusion culling using five-dimensional subdivision*. Computers & Graphics 23 1999 pp. 645-654

[GH95] D. Green and D. Hatch. *Fast Polygon-Cube Intersection Testing*. Graphics Gems V p. 375-379, 1995

[GKM93] Ned Greene, Michael Kass and Gavin Miller. *Hierarchical Z-Buffer Visibility*. Proceedings of Siggraph 93 pp. 231-238

[GK94] N. Greene and M. Kass. *Error-Bounded Antialiased Rendering of Complex Environments*. Proceedings of Siggraph 94 pp. 59-66

[Gre96] Ned Greene. *Hierarchical Polygon Tiling with Coverage Masks*. Proceedings of Siggraph 96 pp. 65-74

[Gre99] Ned Greene. *Occlusion Culling with Optimized Hierarchical Buffering*. Siggraph 99 Sketches & Applications p. 261

[GP95] Eduard Gröller and Werner Purgathofer. *Coherence in Computer Graphics.* Technical report TR-186-2-95-04, Vienna University of Technology, 1995

[Hec90] P. S. Heckbert. *Adaptive Radiosity Textures for Bidirectional Ray Tracing*. Proceedings of Siggraph 90 p. 145-154

[Hei99] W. Heidrich and H.-P. Seidel. *Realistic, Hardware-accelerated Shading and Lighting*. Proceedings of Siggraph 99 p. 171

[Hei01] W. Heidrich. *Interactive Display of Global Illumination Solutions for Non-diffuse Environments - A Survey*. Computer graphics forum vol. 20 no. 4 p. 225, 2001

[Hp00] Hewlett-Packard. *OpenGL Implementation Guide*. www.hp.com/workstations/support/documentation/manuals/ user_guides/graphics/opengl/ImpGuide/ 01_Overview.html#OcclusionExtension, 2000

[HTP01] Heinrich Hey, Robert F. Tobler and Werner Purgathofer. *Real-Time Occlusion Culling With A Lazy Occlusion Grid*. Proceedings of Eurographics Workshop on Rendering 2001 pp. 215-220

[HP01] Heinrich Hey and Werner Purgathofer. *Occlusion Culling Methods*. Eurographics 2001 State of the Art Reports p. 43

[HP02a] Heinrich Hey and Werner Purgathofer. *Importance Sampling with Hemispherical Particle Footprints*. SCCG 2002 p. 99

[HP02b] Heinrich Hey and Werner Purgathofer. *Advanced Radiance Estimation For Photon Map Global Illumination*. Proceedings of Eurographics 2002

[HP02c] Heinrich Hey and Werner Purgathofer. *Real-time rendering of globally illuminated soft glossy scenes with directional light maps*. Vienna University of Technology, Technical Report TR-186-2-02-05, 2002

[HW99] Poon Chun Ho and Wenping Wang. *Occlusion Culling Using Minimum Occluder Set and Opacity Map*. Proceedings of IEEE International Conference on Information Visualization 1999 pp. 292-300

[HM+97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff and H. Zhang. *Accelerated Occlusion Culling using Shadow Frusta*. ACM Symposium on Computational Geometry (SCG) 1997

[Jen95] H. W. Jensen. *Importance Driven Path Tracing using the Photon Map*. Proceedings of Eurographics Workshop on rendering 1995 p. 359-369

[JC95] H. W. Jensen and N. J. Christensen. *Photon Maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects*. Computers & Graphics vol. 19 no. 2 1995 p. 215-224

[Jen96a] H. W. Jensen. *Rendering Caustics on Non-Lambertian Surfaces*. Graphics Interface 1996 p. 116-121

[Jen96b] H. W. Jensen. *Global Illumination using Photon Maps*. Proceedings of Eurographics Workshop on rendering 1996 p. 21-30

[JC98] H. W. Jensen and P. H. Christensen. *Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps.* Proceedings of Siggraph 98 p. 311-320

[JCS01] H. W. Jensen, P. H. Christensen and F. Suykens. *A Practical Guide to Global Illumination using Photon Mapping*. Siggraph 2001 course notes 38

[KW00] A. Keller and I. Wald. *Efficient Importance Sampling Techniques for the Photon Map*. Vision, Modeling, and Visualization 2000 p. 271-279

[KS00a] James T. Klosowski and Cláudio T. Silva. *The Prioritized-Layered Projection Algorithm for Visible Set Estimation*. IEEE transactions on visualization and computer graphics vol. 6 no. 2 pp. 108-123, 2000

[KS00b] James T. Klosowski and Cláudio T. Silva. *Efficient Conservative Visibility Culling Using The Prioritized-Layered Projection Algorithm*. Siggraph 2000 Course Notes 4

[KCC00] Vladlen Koltun, Yiorgos Chrysanthou and Daniel Cohen-Or. *Virtual Occluders: An Efficient Intermediate PVS representation.* Proceedings of Eurographics Workshop on Rendering 2000 pp. 59-70

[KM+96] Subodh Kumar, Dinesh Manocha, William Garrett and Ming Lin. *Hierarchical Back-Face Computation*. Proceedings of Eurographics Workshop on Rendering 1996 pp. 235-244

[LW95] E. P. Lafortune and Y. D. Willems. *A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing*. Proceedings of Eurographics Workshop on rendering 1995 p. 11-20

[LF97] P. Lalonde and A. Fournier. *Generating Reflected Directions from BRDF Data*. Proceedings of Eurographics 1997 p. 293-300

[Lan91] B. Lange. *The Simulation of Radiant Light Transfer with Stochastic Ray-Tracing*. Proceedings of Eurographics Workshop on rendering 1991

[LB94] B. Lange and M. Beyer. *Rayvolution: An Evolutionary Ray Tracing Algorithm*. Proceedings of Eurographics Workshop on rendering 1994

[LT99] Fei-Ah Law and Tiow-Seng Tan. *Preprocessing Occlusion For Real-Time Selective Refinement*. Symposium on Interactive 3D Graphics 1999 pp. 47-53

[LH96] M. Levoy and P. Hanrahan. *Light Field Rendering*. Proceedings of Siggraph 96 p. 31

[LG95] David Luebke and Chris Georges. *Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets*. Symposium on Interactive 3D Graphics 1995 pp. 105-106

[MRP98] G. Miller, S. Rubin and D. Ponceleon. *Lazy decompression of surface light fields for precomputed global illumination*. Proceedings of Eurographics Worshop on rendering 1998 p. 281

[MH99] Tomas Möller and Eric Haines. *Real-Time Rendering* pp. 192-200, 1999

[Mys97] K. Myszkowski. *Lighting Reconstruction Using Fast and Adaptive Density Estimation Techniques*. Proceedings of Eurographics Workshop on rendering 1997 p. 251-262

[Nay92] Bruce F. Naylor. *Partitioning Tree Image Representation and Generation from 3D Geometric Models*. Graphics Interface ´92 pp. 201-212

[Nay95] Bruce F. Naylor. *Interactive Playing with Large Synthetic Environments*. Symposium on Interactive 3D Graphics 1995 pp.107-108

[NN+96] A. Neumann, L. Neumann, P. Bekaert and Y. D. Willems, W. Purgathofer. *Importance-driven Stochastic Ray Radiosity*. Proceedings of Eurographics Workshop on rendering 1996

[NSI99] K. Nishino, Y. Sato and K. Ikeuchi. *Eigen-Texture Method*. IEEE Computer Vision and Pattern Recognition 1999 vol. 1 p. 618

[Nv01] NVIDIA. *GeForce3: Lightspeed Memory Architecture*. Technical Brief, 2001

[PM93] S. N. Pattanaik and S. P. Mudur. *The Potential Equation and Importance in Illumination Computations*. Computer Graphics forum vol. 12 no. 2 1993 p. 131-136

[PP98] I. Peter and G. Pietrek. *Importance Driven Construction of Photon Maps*. Proceedings of Eurographics Workshop on rendering 1998 p. 269-280

[PD90] Harry Plantinga and Charles R. Dyer. *Visibility, Occlusion and the Aspect Graph*. International Journal of Computer Vision 5(2) 1990 pp. 137-160

[SNB99] C. Saona-Vázquez, I. Navazo and P. Brunet. *The Visibility Octree. A Data Structure for 3D Navigation*. Computers & Graphics 23 pp. 635-643, 1999

[SD+00] Gernot Schaufler, Julie Dorsey, Xavier Decoret and François X. Sillion. *Conservative Volumetric Visibility with Occluder Fusion.* Proceedings of Siggraph 2000 pp. 229-238

[ST97] Dieter Schmalstieg and Robert F. Tobler. *Exploiting coherence in 2½D visibility computation*. Computers & Graphics vol. 21 no. 1 p. 121, 1997

[SS95] P. Schröder and W. Sweldens. *Sherical Wavelets: Efficiently Representing Functions on the Sphere*. Proceedings of Siggraph 95 p. 161-172

[SOG98] Noel D. Scott, Daniel M. Olsen and Ethan W. Gannett. *An Overview of the VISUALIZE fx Graphics Accelerator Hardware*. Hewlett-Packard Journal May 1998 pp. 28-34

[Sev99] Ken Severson. *VISUALIZE Workstation Graphics for Windows NT*. Hewlett-Packard product literature, 1999

[Shi92] P. Shirley. *Nonuniform Random Point Sets via Warping*. Graphics Gems III p. 80-83, Academic Press 1992

[SSS95] M. Stamminger, P. Slusallek and H.-P. Seidel. *Interactive Walkthroughs and Higher Order Global Illumination*. Modeling, Virtual Worlds, Distributed Grapics p. 121, 1995

[SS+00] M. Stamminger, A. Scheel, X. Granier, F. Perez-Cazorla, G. Drettakis and F. Sillion. *Efficient Glossy Global Illumination with Interactive Viewing*. Computer Graphics Forum vol. 19 no. 1 p. 13, 2000

[Ste97] A. James Stewart. *Hierarchical Visibility in Terrains*. Proceedings of Eurographics Workshop on Rendering 1997 pp. 217-228

[SB97] W. Stürzlinger and R. Bastos. *Interactive Rendering of Globally Illuminated Glossy Scenes*. Proceedings of Eurographics Worshop on rendering 1997 p. 93

[Stü98] W. Stürzlinger. *Calculating Global Illumination for Glossy Surfaces*. Computers & Graphics vol. 22 no. 2-3 p. 175, 1998

[SG96] Oded Sudarsky and Craig Gotsman. *Output-Sensitive Visibility Algorithms for Dynamic Scenes with Applications to Virtual Reality*. Proceedings of Eurographics 1996 pp. 249-258

[SG99] Oded Sudarsky and Craig Gotsman. *Dynamic Scene Occlusion Culling*. IEEE transactions on visualization & computer graphics vol. 5 no. 1 pp. 217-223, 1999

[SW00] F. Suykens and Y. D. Willems. *Density Control for Photon Maps*. Proceedings of Eurographics Workshop on rendering 2000 p. 23-34

[SCP99] László Szirmay-Kalos, Balázs Csébfalvi and Werner Purgathofer. *Importance driven quasi-random walk solution of the rendering equation*. Computers & Graphics 23(2) 1999 p. 203-211

[TS91] Seth J. Teller and Carlo H. Séquin. *Visibility Preprocessing For Interactive Walkthroughs*. Proceedings of Siggraph 91 pp. 61-69

[TN+98] R. F. Tobler, L. Neumann, M. Sbert and W. Purgathofer. *A new Form Factor Analogy and its Application to Stochastic Global Illumination Algorithms*. Proceedings of Eurographics Workshop on rendering 1998

[UT97] C. Ureña and J. C. Torres. *Improved Irradiance Computation by Importance Sampling*. Proceedings of Eurographics Workshop on rendering 1997

[PS99] Michiel van de Panne and A. James Stewart. *Effective Compression Techniques for Precomputed Visibility*. Proceedings of Eurographics Workshop on Rendering 1999 pp. 313-324

[VG94] E. Veach and L. Guibas. *Bidirectional Estimators for Light Transport*. Proceedings of Eurographics Workshop on rendering 1994 pp. 147-162

[VG95] E. Veach and L. J. Guibas. *Optimally Combining Sampling Techniques for Monte Carlo Rendering*. Proceedings of Siggraph 95 p. 419

[VG97] E. Veach and L. J. Guibas. *Metropolis Light Transport*. Proceedings of Siggraph 97 p. 65-76

[Voo92] D. Voorhies. *Triangle-Cube Intersection*. Graphics Gems III p. 236-239, 1992

[WH+97] B. Walter, P. M. Hubbard, P. Shirley and D. Greenberg. *Global Illumination Using Local Linear Density Estimation*. ACM Transactions on Graphics vol. 16 no. 3 p. 217, 1997

[WA+97] B. Walter, G. Alppay, E. Lafortune, S. Fernandez and D. P. Greenberg. *Fitting Virtual Lights For Non-Diffuse Walkthroughs*. Proceedings of Siggraph 97 p. 45

[WBP98] Yigang Wang, Hujun Bao and Qunsheng Peng. *Accelerated Walkthroughs of Virtual Environments Based on Visibility Preprocessing and Simplification*. Proceedings of Eurographics 1998 pp. 187-194

[WH92] G. J. Ward and P. S. Heckbert. *Irradiance Gradients*. Proceedings of Eurographics Workshop on rendering 1992 p. 85-98

[WGS99] Michael Wimmer, Markus Giegl and Dieter Schmalstieg. *Fast Walkthroughs with Image Caches and Ray Casting*. Proceedings of Eurographics Worshop on virtual environments 1999 pp. 73-84

[WS99] Peter Wonka and Dieter Schmalstieg. *Occluder Shadows for Fast Walkthroughs of Urban Environments*. Proceedings of Eurographics 1999 pp. 51-60

[WWS00] Peter Wonka, Michael Wimmer and Dieter Schmalstieg. *Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs*. Proceedings of Eurographics Workshop on Rendering 2000 pp. 71-82

[WA+00] D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. H. Salesin and W. Stuetzle. *Surface Light Fields for 3D Photography*. Proceedings of Siggraph 2000 p. 287

[XS99] Feng Xie and Michael Shantz. *Adaptive Hierarchical Visibility in a Tiled Architecture*. Proceedings of Eurographics/Siggraph Workshop on Graphics Hardware 1999 pp. 75-84

[YR96] Roni Yagel and William Ray. *Visibility Computation for Efficient Walkthrough of Complex Environments*. Presence vol. 5 no. 1 pp. 45-60, 1996

[ZE01] Brian Zaugg and Parris K. Egbert. *Voxel Column Culling: Occlusion Culling for Large Terrain Models*. Proceedings of the Joint Eurographics-IEEE TCVG Symposium on Visualization 2001 pp. 85-93

[ZM+97] Hansong Zhang, Dinesh Manocha, Tom Hudson and Kenneth E. Hoff III. *Visibility Culling using Hierarchical Occlusion Maps*. Proceedings of Siggraph 97 pp. 77-88