State of the Art Report on Ambient Occlusion

Martin Knecht*
Technical University of Vienna, Austria

Abstract

Ambient occlusion is a shading method which takes light occluded by geometry into account. Since this technique needs to integrate over a hemisphere it was first only used in offline rendering tools. However, the increasing resources of modern graphics hardware, enable us to render ambient occlusion in realtime. The goal of this report is to describe the most popular techniques with respect to realtime rendering. First we introduce how ambient occlusion is defined and then we will explain and categorize the presented techniques.

Keywords: ambient occlusion, global illumination, real time

1 Introduction

Ambient Occlusion is a inverted form of surface exposure for incoming light. In the following sections we will first introduce what surface exposure is. After that we will concentrate on the mathematical definition of Ambient Occlusion.

1.1 Surface exposure

The term *surface exposure* is understood as an accessability factor for a given point p in a 3D world. Surface exposure plays a very important role when simulating real world phenomena where distribution of some particles over a given geometry is taken into account. For example Desbenoit et al. [Desbenoit et al. 2004] used surface exposure to alter the growth behavior of lichen.

1.2 Ambient light, Ambient Occlusion

Ambient light is a uniform light, illuminating all points on all objects with the same intensity. The positions and shapes of the objects are not taken into account and thereby giving them an unnatural look and flattening their features. By taking the occlusion factor at a given point p into account, it is possible to get much better results. For instance, areas with sharp corners appear darker than open areas and contact shadows seem plausible [Kontkanen and Laine 2005].

Ambient Occlusion was introduced by Zhukov et al. [Zhukov et al. 1998]. It uses the inverted principal as surface exposure and is defined as the percentage of light, blocked by geometry close to point p:

$$ao(p,n) = \frac{1}{\pi}\int_{\Omega} V(p,\omega)\max(n\cdot\omega,0)d\omega \qquad (1)$$

Here n is the normal of point p and $V(p,\omega)$ is the visibility function which returns zero if no geometry is visible in direction ω and one otherwise. The importance of a particular direction is weighted by the cosine between the normal n and direction ω. $\int_{\Omega}$ refers to the integration over the entire hemisphere with respect to n.

With this method we can approximate global illumination, but with much less computational effort. But still, for realtime applications it is a challenge to do this calculation. So there are many different approaches to solve this problem. Some of them do a precomputation step in where they calculate some occlusion factors and save it for later use.

1.3 Overview

In this report we will discuss several ambient occlusion techniques. Since there are many different approaches we made a classification. In Section 2 we describe methods which use some kind of object based approaches. In Section 3 we will present algorithms that are able to directly calculate ambient occlusion for a given vertex. Section 4 introduces a recently developed image based method and Section 5 explains two special algorithms designed for animated characters.

2 Object based Methods

In this section we will present two object based methods [Kontkanen and Laine 2005; Malmer et al. 2006]. With object based methods we mean, that ambient occlusion is calculated on the level of objects. In both techniques there are some sort of textures attached to the occluder objects.

Previous work was done by Mendez et al. [Mendez et al. 2003]. They introduced a method where obscurances are only re-sampled at moving objects. However, since the calculation is based on patches, a scene needs a high number of patches to get good contact shadows. Greger et al. [Greger et al. 1998] used a grid to store illumination values that was attached to the scene, not to the objects. Gibson et al. [Gibson et al. 2003] introduced a method where objects could be placed interactively into a scene, casting shadows onto the existing scene. Incident light could be arbitrary, but the scene had to remain static. Zhou et al. [Zhou et al. 2005] precomputes a shadow field and a occlusion field for an object inside the scene. At runtime these fields can be used to calculate inter object shadowing.

2.1 Ambient Occlusion Fields

This technique introduced by Kontkanen and Laine [Kontkanen and Laine 2005] makes use of precomputation for a given occluder and

*e-mail: martin.knecht@aon.at

the results are stored in two cube-maps. It is able to handle inter object occlusion but deformable objects are not supported. See Figure 1 for an example image of the algorithm.

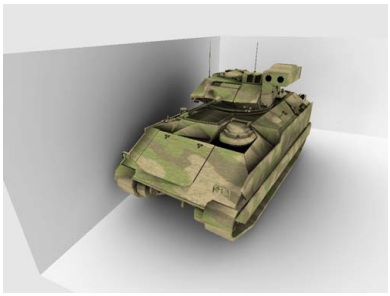

Figure 1: A tank casts shadows on the surrounding environment. The shadows are rendered using the method described in this paper. The scene runs at 136 fps at a resolution of 1024768 on a Pentium 4 with ATI Radeon 9800XT graphics board.
[Kontkanen and Laine 2005]

2.1.1 Description of the algorithm

The idea behind the algorithm is, to approximate the occluder by a spherical cap when computing the ambient occlusion on a receiver. To calculate this quickly a preprocessing step is performed, where the angle of the cone and the average occlusion direction is stored as fields around the occluder. To access these fields quickly and keep memory usage low these fields are stored as radial functions into cube-maps (see Figure 2).

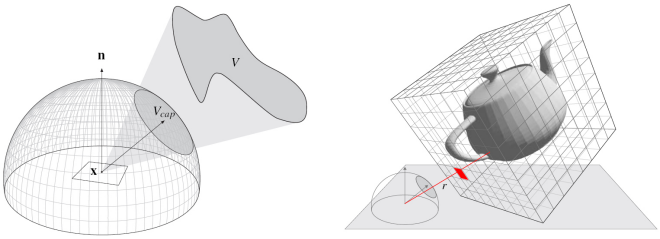

Figure 2: Illustration of the approximated spherical cap [Kontkanen and Laine 2005].

2.1.2 Precomputation

The precomputation does three main steps. First, the size of the spherical cap $\Omega(x)$ at a defined position x has to be calculated. Second, the average occlusion direction $\Gamma(x)$ is calculated and third, a radial parametric representation has to be computed.

The *spherical cap* is calculated using a similar equation as ambient occlusion:

$$\Omega(x) = \int_{\Theta} V(x,\omega) d\omega \tag{2}$$

If the occluder is visible over the *entire* sphere, $\Omega(x)$ equals 4π. To calculate the *average occlusion direction* we use the following equation:

$$\Gamma(x) = normalize(\int_{\Theta} V(x,\omega)\omega d\omega) \tag{3}$$

Function $\Gamma(x)$ is a component wise addition of directions ω where $V(x,\omega)$ returns one.

The third step is to compute a *radial parametric representation* for the given $\Omega(x)$ and $\Gamma(x)$. The representation should ideally fulfill the constrain, that we have higher resolution near the occluder and lower resolution when distance is large. Ω and Γ can be seen as depending on the radius for a given ω. $\tilde{\Omega}(r,\omega)$ approximates this by taking into account that the angle ω is approximately proportional to the inverse square root of r.

$$\tilde{\Omega}(r,\omega) = \frac{1}{a(\omega)r^2 + b(\omega)r + c(\omega)} \tag{4}$$

For $\tilde{\Gamma}(r,\omega)$ they assume that while going further away from the occluder, the average occlusion direction will fade towards the center of the object.

$$\tilde{\Gamma}(x) = normalize(C_0(\omega) - r\omega) \tag{5}$$

$C_0(\omega)$ can be seen as a characteristic point of the object. The next step is to find values for $a(\omega)$, $b(\omega)$, $c(\omega)$ and $C_0(\omega)$. Kontkanen and Laine [Kontkanen and Laine 2005] does this by using a method of least squares to find the minimum error for given ω and r. To appropriately approximate ambient occlusion for receivers inside the convex hull of a occluder, we have to store the distance from the center of the occluder to the convex hull in $r_0(\omega)$. We will explain this situation in more detail in the Discussion Section.

Up to now, we have three scalar values $a(\omega)$, $b(\omega)$ and $c(\omega)$ plus three scalar values for $C_0(\omega)$ plus the distance between the center and the convex hull $r_0(\omega)$, which makes a total of seven scalar values. These values can be stored in two cube-maps.

2.1.3 Runtime

At runtime the algorithm is executed as followed:

For all occluders O **do**

  For all receiver R **do**

    If R influenced by O **then**

      Render R with ambient occlusion field of O with multiplicative blending

    end if

  end for

end for

For every pixel of a receiving object a fragment shader is executed. The polynomials are fetched from the two cube-maps of the occluder. $\tilde{\Omega}(r,\omega)$ and $\tilde{\Gamma}(r,\omega)$ are calculated. To take the relation between the average occlusion direction and the surface normal into account we use a precomputed lookup table which stores the effective occlusion values (see Figure 3).

Note that the occlusion values should decrease to zero with increasing radius r. So it is possible to omit occluders which are far away of the receiver.

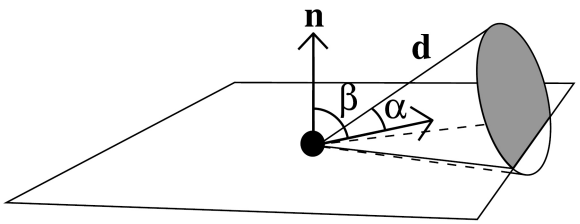

Figure 3: The effective occlusion area is drawn in gray [Malmer et al. 2006].

2.1.4 Combining Occluders

The ambient occlusion values for two occluders are given by A_a and A_b. What we are looking for is the combined A_{ab} occlusion value. Modeling an analytical formula for the spherical caps makes no sense because the caps are already approximations. So as showed in Figure 4 there are three cases which can occur for two occluders a and b. In the first case one occluder completely occludes the other one. The overall occlusion A_{ab} would be $\max(A_a, A_b)$. When the occluders do not overlap A_{ab} would be simply $A_a + A_b$. These two cases can be seen as the extremals, so we can put it into an inequation:

$$\max(A_a, A_b) \leq A_{ab} \leq A_a + A_b \tag{6}$$

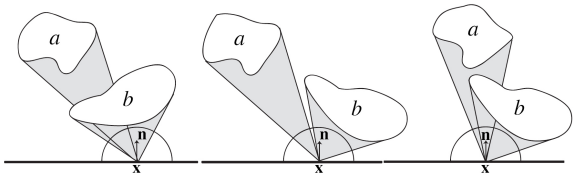

Figure 4: This illustration shows three different cases that can occur. **Left:** Object a is completely occluded by object b. **Middle:** The occluder do not overlap at all. **Right:** The occluder overlap only partially [Kontkanen and Laine 2005].

Furthermore Kontkanen and Laine [Kontkanen and Laine 2005] showed that, when shooting rays from a receivers surface with a cosine probability distribution function, the probability of a hit equals ambient occlusion. So we can assume A_i as probability of a hit on object O_i. And A_{ab} can be seen as the probability on hitting both occluders O_a and O_b. By assuming that A_a and A_b are uncorrelated we can combine the probabilities to:

$$1 - A_{ab} = (1 - A_a)(1 - A_b) \tag{7}$$

So statistically the best guesss is to simply use multiplicative blending with $1 - A_i$.

2.1.5 Discussion

1. As mentioned above ambient occlusion calculation for receivers, which lie inside the convex hull, has to be handled in a different way. A radial ray from the center of an occluder can hit the surface multiple times. This fact would lead to discontinuities in $\tilde{\Omega}$ and $\tilde{\Gamma}$. So Kontkanen and Laine [Kontkanen and Laine 2005] assume that this case will seldom come true. However, if it happens, they modeled a formula which fades ambient occlusion to a predefined occlusion term A_0 with respect to the distance of the convex hull, shown in equations 8 & 9.

$$t = (r/r_0(\omega))^p \tag{8}$$

$$A(\omega, r) = tA(\omega, r_0(\omega)) + (1-t)A_0 \tag{9}$$

   Here r_0 is the radial distance from the center to the convex hull stored in one of the cube maps. Parameter p defines how fast A fades towards A_0.

2. In some cases it can happen that the resulting ambient occlusion term has high frequency detail. To omit these artifacts a simple lowpass process is applied. This is done by rendering a high resolution cube-map and then downsampling it with an appropriate filter kernel.

3. Kontkanen and Laine [Kontkanen and Laine 2005] showed, that with even extremely low cube-map resolution such as 8x8 pixel, convincing results can be achieved. They point out that it is important to do interpolation on the borders because hardware does not support it for cube-maps.

2.2 Fast Precomputed Ambient Occlusion for Proximity Shadows

As Ambient Occlusion Fields, this technique developed by Malmer et al. [Malmer et al. 2006], needs a precomputation step to render ambient occlusion. It can handle inter object occlusion but is not able of handling deformable meshes. See Figure 5 for an example image of the algorithm.

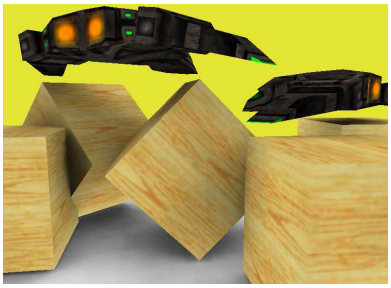

Figure 5: Scene with cubes and spaceships, running at 135 to 175 fps on a Pentium 4, running at 2.8 GHz, with a NVidia GeForce 7800GTX, using a grid resolution of 32^3 [Malmer et al. 2006].

2.2.1 Description of the algorithm

Like in Ambient Occlsuion Fields a texture is attached to an occluder object. But instead of cube-maps they use a volume texture surrounding an object. This volume texture stores the percentage of occlusion from the object over an entire sphere in every voxel.

2.2.2 Precomputation

The precomputation step is quite easy. A volume grid is attached to the occluder as seen in Figure 6. For every voxel the occlusion over the entire sphere of the object is calculated. This can be done with raytracing methods or with graphics hardware support. Each value is stored in the according voxel in a volume texture.

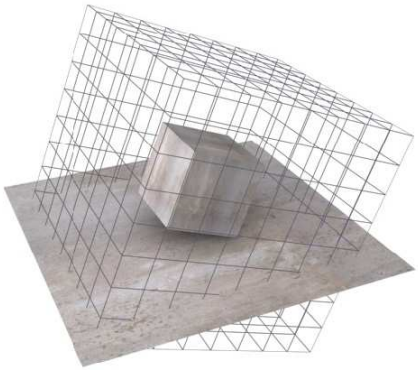

Figure 6: In this illustration we see the occluder with the attached volume grid [Malmer et al. 2006].

2.2.3 Runtime

At Runtime the algorithm is executed as followed:

Render world space position and normals of all shadow receivers in the scene, including occluders.

For all occluders O **Do**

Render the back faces of the occluder's grid (depth-testing is disabled)

For every pixel accessed, execute fragment program

Retrieve the world space position of the pixel

Convert this world space position to voxel position in the grid, passed as a 3D texture.

Retrieve ambient occlusion value in the grid, using linear interpolation.

Ambient occlusion values a from each occluder are blended using multiplicative blending with $1-a$.

The reason for drawing back faces instead of front faces is, that it is unlikely that they will be culled by the far plane. It is much more probably that drawing front faces could lead to artifacts because of culling against the near plane.

Note that world space position and normal has to be available for every pixel.

2.2.4 Simple shading

In each pixel, the corresponding world position is taken and transformed into this 3d texture grid position. Then a simple lookup into the occlusion map is done. Since the occlusion factor was calculated over the entire sphere but one half of the sphere is occluded by the receiver, the ambient occlusion value is multiplied by 2. After that it is painted using multiplicative blending with $1-a$.

There are two drawbacks with this method: First, the occluder sometimes has influences, where it should not, like doing occlusion "through" even behind a wall. Second, self occlusion is not handled and has to be done in a second render pass.

2.2.5 Taking the occlusion direction into account

To overcome this drawbacks Malmer et al. [Malmer et al. 2006] stores along the occlusion value a an average occlusion direction vector d in the volume texture. This constellation can be seen as a cone in direction d and an angle α which is directly linked to a.

Now, the effectively percentage of occlusion depends on the angle β between the receivers normal n and average occlusion direction d and the cone angle α respectively a. The evaluation function can also be precomputed and stored in a lookup table $T_{clip}(a, \cos\beta)$. With this approach self occlusion is handled in a naturally way (see Figure 3 for illustration).

2.2.6 Discussion

1. A very important parameter of this algorithm is the spatial extend of the volume texture. Is it too large, undersampling happens and thus loosing some variations of ambient occlusion. Is it too small we would need too much texture memory. A detailed method to get good results is explained in Kontkanen and Laine [Kontkanen and Laine 2005].

2. If the spatial extend has been calculated, it can happen that artifacts will appear at the borders of the volume texture. To avoid these artifacts the occlusion values have to be rescaled.

3. Voxel which lie inside the occluders volume are always set to one. However, with this assumption occlusion values at the boundary of the occluder will often have non-correct values. To omit these errors the values inside the occluder should be an average of the nearest voxels having an occlusion value not equal to one.

2.3 Comparison

Precomputation: It is obviously that in Kontkanen and Laine [Kontkanen and Laine 2005] precomputation is more complex than it is in Malmer et al. [Malmer et al. 2006]. While there are two cub-maps needed which store 7 scalars, Malmer et al. [Malmer et al. 2006] needs one volume texture with 4 scalars.

Memory: Since ambient occlusion is low frequency, even very small texture sizes will lead to convincing results. For using two cube-maps with a size of 32x32 pixel and a 16bit fixed-point format, a storage cost of approximately 100Kb is needed. Where Malmer et al. [Malmer et al. 2006] uses a 32^3 voxel volume texture with 8bit integer format, they need 32Kb when only storing ambient occlusion and around 128Kb when also storing the average occlusion direction.

Runtime costs: Because of the parametrical representation the fragment shader of Kontkanen and Laine [Kontkanen and Laine 2005] has 47 assembly instruction plus three texture lookups. In comparison the method of Malmer et al. [Malmer et al. 2006] needs a fragment shader that has 16 assembler instructions and four texture lookups.

Convex Hull: The method from Kontkanen and Laine [Kontkanen and Laine 2005] is not really capable of rendering receiving objects inside a convex hull of an occluder, although they perform an additional computation to avoid artifacts. Through using a volume texture and tweaking the values inside of the occluder appropriately, the method of Malmer et al. [Malmer et al. 2006] can handle these cases naturally.

Deferred Shading: Note that both methods can be used with deferred shading.

3 Vertex based Methods

With vertex based methods we mean algorithms that compute the ambient occlusion on the level of vertices. So there are no occluder objects at all. Most of these algorithms come with very little precomputation afford, doing ambient occlusion calculation just on the fly.

Chunhui et al. [Chunhui et al. 2004] precomputes spherical radiance transport maps to solve inter-object occlusion by pre-rasterizing coverage masks of an occluder. At runtime, illumination computation is done per vertex. Sloan et al. [Sloan et al. 2002] introduced precomputed radiance transfer with spherical harmonics to calculate global illumination. However, only scenes with static geometry can be rendered.

3.1 Dynamic Ambient Occlusion and Indirect Lighting

The presented algorithm is able to calculate ambient occlusion nearly on the fly. The preprocessing step is very small in comparison to the previously presented methods. Furthermore through the fact that nearly everything is done while rendering, it is also possible to use deformable meshes. Through the conversation to surface elements, which is described below, we are also able to calculate bounces of indirect light.

3.1.1 Description of the algorithm

The basic idea behind this method from Bunnell [Bunnell 2005] is to treat a geometric mesh as a bunch of surface elements. Surface elements are oriented discs having a position, a normal and an area size. For each vertex of the mesh, a surface element is created by calculating the size of the polygons surrounding it, taking the position and the average normal of the surrounding polygons. So a surface element is an approximation of the surface making it easier to calculate illumination or shadowing against each other. Figure 7 shows the conversion from a mesh to surface element representation.

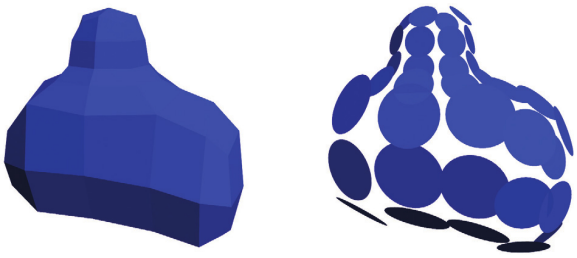

Figure 7: For every vertex a surface element is calculated [Bunnell 2005].

In order to allow animated meshes the surface elements have to be stored in a texture map, enabling us to update them very fast. The main part of the algorithm is just calculating the accessibility from one surface (the emitter) element to the other (the receiver). So the receivers luminance is defined as one minus all other surface elements shadowing the receiver. Bunnell [Bunnell 2005] approximates the amount of shadow by an equation which depends on the area size A of the emitter and the angles between the surface elements:

$$1 - \frac{r \cos\Theta_E \max(1, 4\cos\Theta_R)}{\sqrt{\frac{A}{\pi} + r^2}} \tag{10}$$

As shown in Figure 8 Θ_E and Θ_R are the angles between the corresponding disc normals and the connecting line between the surface elements.

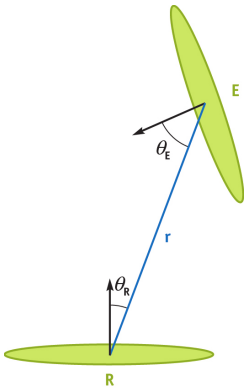

Figure 8: For every vertex a surface element is calculated [Bunnell 2005].

3.1.2 Precomputation

Precomputation necessary for this method is rather small. As we will show later, each surface element is applied to each other. So we have a complexity of $O(n^2)$ which scales very bad. To work against this circumstances we can build up a hierarchic representation of surface elements. This is done by summarize neighboring surface elements to a single new one. When rendering we traverse this hierarchy and if the emitter surface element is far away, we can stop earlier in the hierarchy thus, enabling better scaling with a complexity of $O(n \log n)$. However, when using animated meshes these surface elements have to be recalculated. But since the change of size of the surface elements is very sparse only the position and normals needs to be updated.

3.1.3 Runtime

The algorithm is executed in two passes. In the first pass for a receiving surface element the occlusion of every other surface element is summed and the result is subtracted from one. This pass gives a first approximation on the ambient occlusion but since some surface elements cast shadows but are themselves in a shadow other surface elements will get too dark. See Figure 9 for illustration. So in the second pass again the occlusion is calculated for a given receiver but this time the occlusion value for each surface element is multiplied by its own accessability value from the first pass. With this step any double shadowed surface is corrected. However tripled shadowed surfaces will still be too dark. Additionally to the accessability values the bend normal is calculated and stored in the texture. Since all of the previously described calculations are done in a fragment shader, we have to draw a quad with a size that correlates with the number of vertices/surface elements in the scene.

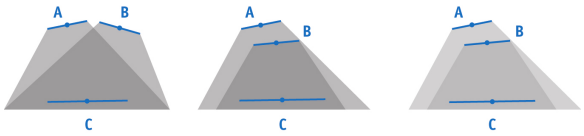

Figure 9: **Left:** Occlusion calculation is correct. **Middle:** Occlusion calculation gets too dark. **Right:** After the second pass, darkening is reduced [Bunnell 2005].

3.1.4 Adding indirect lighting

With some modifications this algorithm can also be used for calculating bounces of indirect light. This is done by replacing the solid angle function 10 with a disc-to-disc radiance transfer function. We will not go into more details but as seen in Figure 10 the additional bounce improves image quality very much.

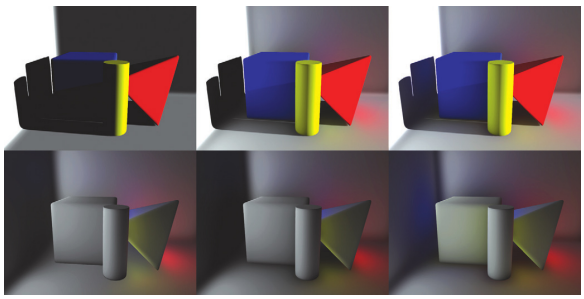

Figure 10: Top, left to right: Scene lit with direct lighting, with direct lighting plus one bounce of indirect lighting, and with direct lighting plus two bounces of indirect lighting. Bottom, left to right: Indirect lighting after one pass, after two passes (one bounce), and after two bounces (four passes total) [Bunnell 2005].

3.2 Hemispherical Rasterization for Self-Shadowing of Dynamic Objects

This method developed by Kautz et al. [Kautz et al. 2004] can be used to calculate self-shadowing for dynamic objects. It utilizes the previous work on Precomputed Radiance Transfer by Sloan et al. [Sloan et al. 2002]. Figure 11 shows an animated model of a hand calculated with this approach.

3.2.1 Description of the algorithm

The main idea behind this method is to use a dynamic form of radiance transfer for calculating self-shadowing. Without going into further detail, we present the main equation for radiance transfer:

$$L_{out}(p,v) = L_{in} \cdot V^*(p,v) \tag{11}$$

where $L_{out}(p,v)$ is the outgoing radiance at point p in direction v. L_{in} represents the incoming light. $V^*(p,v)$ is called the *transfer vector*, which specifies how much incoming radiance from a given direction is transferred to the outgoing direction.

Normally $V^*(p,v)$ is precomputed to get realtime results. In this method they calculate the transfer vector on the fly. In order to do

Figure 11: Animated hand, rendered with BRDF at 6.1 fps [Kautz et al. 2004].

so, the hemispherical integral at point p

$$V^*(p,v) = \int_{\Omega} b_i(\omega) V(p,\omega) f_r(\omega,v) \max(0, n \cdot \omega) d\omega \tag{12}$$

has to be solved for each vertex. We are interested in fast computation of the term $V(p,\omega)$ which will be called the *visibility mask*. Kautz et al. [Kautz et al. 2004] therefore developed the so called visibility rasterizer.

3.2.2 Runtime

At runtime for each vertex a visibility rasterization is performed. Working in the tangent space the visibility mask - a 1 bit regular grid - is inside the unit hemisphere at point p. The point normal coincidences with the visibility mask plane normal. The presented method is very similar to the hemicube algorithm developed by Cohen and Greenberg [Cohen and Greenberg 1985] for calculating form factors.

When a triangle is determined to lie above the tangent plane its spherical projection is rasterized into the visibility mask. More specifically, note that each edge of a triangle and the hemisphere origin define a plane. Now, a pixel in the visible mask is inside the projected triangle if the according point on the hemisphere is above all three planes (see Figure 12 for illustration). Since Kautz et al. [Kautz et al. 2004] uses a low resolution visibility mask of 32x32 pixels, a lookup table for discretized planes can be precomputed. At runtime for every plane a mask set is obtained. By combining the three mask sets by an AND operation the resulting visibility mask can easily be calculated. Multiple triangles are then combined by using the OR operation.

3.2.3 Discussion

1. Note that through the projection on the hemisphere and rasterization the samples are perfectly weighted by the cosine distribution function.
2. To reduces the number of samples when calculating equation 12 Kautz et al. [Kautz et al. 2004] performs downsampling to a visibility mask of 4x4, obtaining an gray-scale occlusion term.

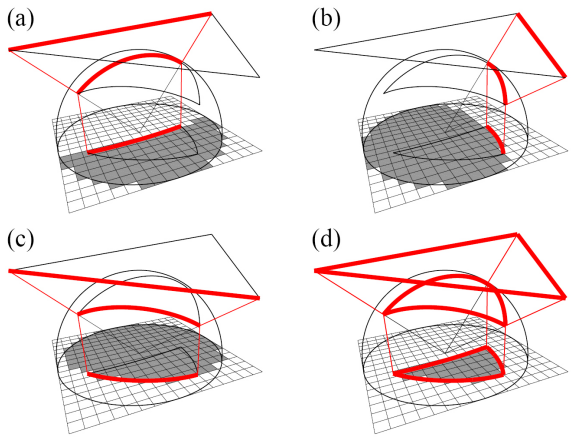

Figure 12: Principle of spherical rasterization. (a)-(c): Each edge of the triangle defines a plane with the origin. A lookup table stores a visibility mask of pixels that lie above a certain plane. For every plane a lookup is performed. (d): The final image of the spherically projected triangle is obtained by using an AND operation on the looked up visibility masks [Kautz et al. 2004].

3. The complexity of computation for the rasterizer is the number of vertices times the number of triangles, which scales very bad for larger objects. To obtain better scaling the fact, that nearer triangles support for more occlusion is exploited. So the near surrounding triangles are normally rasterized and for triangles which are farther away, a low-poly model is used instead of the high resolution model. Since for every triangle it has to be evaluated if it lies above the tangent plane a better performance would be achieved if neighboring triangles would be clustered.

3.3 Hardware-accelerated ambient occlusion computation

This method developed by Sattler et al. [Sattler et al. 2004] does not need any precalculation at all. It is able to handle inter object occlusion as well as deformable meshes. For better performance it is also possible to distribute ambient occlusion calculation over several frames. Figure 13 shows the Stanford dragon illuminated by an environment map.

Figure 13: Stanford dragon rendered with ambient occlusion [Sattler et al. 2004].

3.3.1 Description of the algorithm

The main idea is to discretize the integration over the hemisphere for a given point p. This discretisation is the same as if there where k directional light sources surrounding the object evenly distributed on a sphere. And for every vertex it is tested if a particular light source is visible to the vertex. To perform these tests very quickly Sattler et al. [Sattler et al. 2004] makes use of hardware occlusion queries and depth testing.

3.3.2 Runtime

Consider that we have k light sources surrounding the entire scene. Given a light source we render the entire scene from the light source position. After that we render all vertices as a point set. For every vertex an individual occlusion query is defined. So, when rendering a vertex which lies behind another object, which is determined by depth testing, the occlusion query will return zero otherwise one. To speed up later computation the dot product between the vertex normal n_i and the direction of the lightsource l_i are stored in a matrix M which is called the *visibility matrix*. So matrix M should look as followed:

$$M_{ij} = \begin{cases} n_i \cdot l_i & : \text{vertex visible} \\ 0 & : \text{vertex invisible} \end{cases}$$

The following pseudo code outlines the algorithm:

enable orthographic projection

disable frame-buffer

For all light directions j **do**

set camera at light direction l_j

render objects into depth buffer with polygon offset

For all vertices i **do**

begin query i

render vertex i

end query i

end for

For all vertices i **do**

retrieve result from query i

if result is 'visible' **then**

$M_{ij} = n_i \cdot l_j$

end if

end for

end for

After the matrix M_{ij} has been calculated the resulting color c_i for each vertex v_i can be obtained by following formula:

$$c_i = \sum_{j=1}^{k} M_{ij} I_i \tag{13}$$

where I_i is the RGB color of the light.

3.3.3 Discussion

1. For positioning the directional lights around the sphere Sattler et al. [Sattler et al. 2004] uses subdivisions on an octahedron. They start with $k = 6$ directional lights for subdivision level $s = 0$. For subdivision level $s+1$ they perform midpoint subdivision on the edges and reposition the new light sources on the sphere. In that way they get well distributed vertices around the sphere with the big advantage that the previously calculated vertices can be reused.

2. Every time a object is moved or deformed the matrix M has to be completely recalculated. Sattler et al. [Sattler et al. 2004] developed some approaches which can still get realtime frame rates. The first optimization uses the fact, that not all values of matrix M have to be calculated for a given viewpoint. And if the viewpoint changes temporal coherence can be exploited so that only a few vertices, the ones which appear near the borders of an object, have to be calculated.

3. Another optimization uses the possibilities of the subdivision sphere for the light positions. Every time when an object is moved or deformed the subdivision level is decreased and thus calculation is faster. When nothing changes inside the scene, the subdivision level can be increased and thus converging to the exact solution.

4. Sattler et al. [Sattler et al. 2004] also showed that an environment map can be used to illuminate the scene. This is done by using a Mercator projection of the image and then blurring it. The illumination of a specific light l_i is given by a lookup in the environment map.

3.4 Comparison

The previously described algorithms do most of the calculation on the fly. Except Sattler et al. [Sattler et al. 2004], every method uses some sort of mesh simplification like Bunnell [Bunnell 2005] and Kautz et al. [Kautz et al. 2004] use a low poly model of a mesh.

Bunnell [Bunnell 2005] needs two render passes to get convincing results. Kautz et al. [Kautz et al. 2004] needs one render pass to calculate outgoing radiance, but it is also the only method which has to do a lot of work on the cpu because of the visibility rasterization. Sattler et al. [Sattler et al. 2004] needs for k light sources k render passes to calculate matrix M. However once calculated, ambient occlusion can be evaluated pretty fast.

Bunnell [Bunnell 2005] scales for a number of n vertices by $O(n \log n)$. A similar behavior could be expected from Kautz et al. [Kautz et al. 2004]. Scaling of Sattler et al. [Sattler et al. 2004] mainly depends on the number of light sources used. For the number of vertices it scales by $O(n)$.

Note that vertex based methods all suffer from the same problem. Since only vertices are calculated, a scene needs a high triangulation to get convincing results otherwise under-sampling will occur. See Figure 14 for illustration.

4 Image Based Methods

With image based methods, we mean algorithms that are able to calculate ambient occlusion with a viewing plane driven approach. In this section we will present a recently introduced method to compute ambient occlusion.

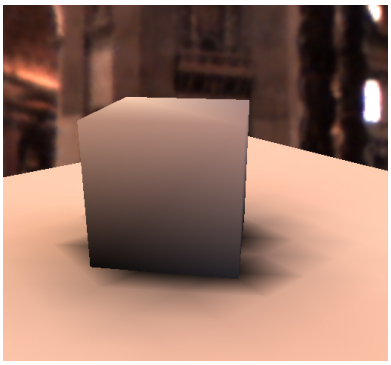

Figure 14: Artifacts due to a coarse mesh [Kautz et al. 2004].

4.1 Hardware Accelerated Ambient Occlusion Techniques on GPUs

The algorithm developed by Shanmugam and Arikan [Shanmugam and Arikan 2007] uses an image based approach to ambient occlusion. Furthermore it differentiates between high frequency and low frequency ambient occlusion. It can be executed completely without prepcomputation and is therefore well suited for animated and deformable objects.

4.1.1 Description of the algorithm

The main idea behind this method is to split the ambient occlusion problem into two parts, high frequency and low frequency ambient occlusion. In this section we will first describe how high frequency ambient occlusion is calculated.

High frequency ambient occlusion can occur, when an object is pretty close to another one. At the shortest distances between the objects, the crossover between bright and dark areas can be rather fast, therefore the term high frequency. To approximate ambient occlusion given in equation 14 they use a representation with spheres as shown in Figure 15. This assumption leads to the following equation:

$$A_\Psi(C,r,P,n) = S_\Omega(P,C,r)\max(n \cdot PC, 0) \tag{14}$$

where S_Ω is the surface area of the spherical cap of the sphere defined by position C and radius r. PC is the unit vector from receiver point P to sphere center C. n is the normal at point P.

As they show by further examples, ambient occlusion factors depend mostly on occluders which are near to the receiving point. Thus they choose an image based approach to calculate high frequency ambient occlusion. A so called ND-buffer which stores the normal and depth values for each viewing pixel is used to exploit spacial relationships. Due to the nature of perspective projection, points which are near to each other in a 3d world will also be on the 2d projected plane. Please note that there are some special cases which will be described later in the Discussion section. Nearby occluders are that ones which lie inside a given r_{far}. Each pixel is approximated as a sphere $\langle Q_i, r_i \rangle$ depending on the pixels depth. So when looking at nearby pixels in the ND-buffer it is possible to approximate ambient occlusion. To do so we have to reproject r_{far} into the 3d world to get the effective size of the lookup area. In the following equation we calculate ambient occlusion for a given pixel p in the ND-buffer:

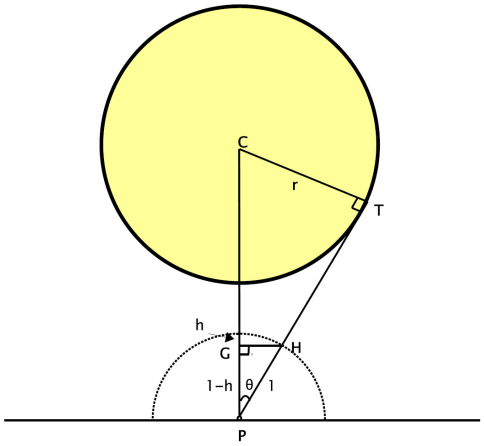

Figure 15: Spheres are used to approximate occluders [Sattler et al. 2004].

$$A(P,n) = \sum_{|P-Q_i|<r_{far}} A_\Psi(Q_i, r_i, P, n) \tag{15}$$

where P is the 3d position of the receiver pixel p. And Q_i and r_i are the neighboring 3d sphere approximations for a given pixel q_i of the ND-buffer.

For the low frequency ambient occlusion Shanmugam and Arikan [Shanmugam and Arikan 2007] uses spherical approximation methods for solid objects [Wang et al. 2006; Bradshaw and O'Sullivan 2002]. These sphere approximation can be used as distant occluders. So when a occluder is far away of the receiver there exists a distance d_{far} where the spherical cap will fall below a certain user defined value ε. To calculate d_{far} they developed the following equation depending on the radius r of an approximated sphere:

$$d_{far} = r\frac{1}{\sin(\cos^{-1}(1-\frac{\varepsilon}{2\pi}))} \tag{16}$$

d_{far} represents the limited influence of an approximated sphere with radius r. d_{far} can now be projected onto the imageplane (d'_{far}). By rendering billboards with this size a fragment shader is executed for every pixel which is influenced by this occluder. In this shader the ambient occlusion term is calculated by using the data out of the ND-buffer (P,n) and evaluate $A_\Psi(C,r,P,n)$. The result is additively blended. With this approach it is possible to render distance occluders very fast by just drawing billboards onto the image plane.

4.1.2 Discussion

1. Previously we discussed, that near objects in 3d are also near in perspective projected 2d. But note that this is not true the other way around. So it does sometimes occur, that pixels are looked up in 2d which are far away in 3d to the receiving point p. However, this does not really impact the ambient occlusion term since its influence is decreased by r^2.
2. In equation 15 the calculation of ambient occlusion is depending on r_{far}. It is possible to use different ways for the lookup of neighboring pixels. For example we could use random sample points or just take every pixel inside r_{far}.
3. Shanmugam and Arikan [Shanmugam and Arikan 2007] also mentioned that because of the depth test, neighboring pixels may not be represented in the NB-buffer. To omit this problem depth-peeling, as described by Everitt [Everitt 2001], is recommend.
4. Similar to the occlusion problem in Bunnell [Bunnell 2005] some points can be too dark by applying several occluders. A possible solution for some cases would be to tweak the ε value.

4.1.3 Comparison

As we have now presented object-, vertex- and image based methods, we will show in Table 1 a comparison of their features. The numbers at the methods correspond to the section where they were presented:

Method	*type*	*p*	*i*	*s*	*d*	*complexity*
2.1 ao. fields	object	+	+	-	-	$O(n_o n_r)$
2.2 prox. shadows	object	+	+	+	-	$O(n_o n_r)$
3.1 surface elements	vertex	$\ll$	+	+	+	$O(n_v \log n_v)$
3.2 rasterization	vertex	$\ll$	-	+	+	$\approx O(n_v \log n_t)$
3.3 lightsphere	vertex	-	+	+	+	$O(n_l)$
4.1 image based	image	$\ll$	+	+	+	$O(s_x s_y)$

Table 1: Shows a comparison between the differen ambient occlusion methods.

Legend:

p ... Shows if preprocessing is necessary. The algorithms presented in 3.1, 3.2 and 4.1 where set to $\ll$ because they all use some kind of mesh approximation, which can be done in a precomputation step.

i ... ability to handle inter object occlusion

s ... ability to handle self occlusion. Note that for ambient occlusion fields [Kontkanen and Laine 2005], self-occlusion inside the convex hull is only supported by fading towards a preset occlusion factor.

d ... support for deformable meshes

In the complexity column, n_o, n_r stands for number of occluders and number of receivers. n_v, n_t are used for vertex based methods and are linked to the number of vertices and triangles. Complexity of the method presented in Section 3.3, depends on the number of light sources n_l used. The image based approach described in Section 4.1 depends on the screen resolution s_x, s_y.

Object based approaches are a good and easy to implement way, when using ambient occlusion with some non-deformable objects. But both presented methods need a precomputation step. Vertex based methods instead need only a small precomputation step, Sattler et al. [Sattler et al. 2004] for instance is able to handle everything on the fly, for the price of multiple render passes. An interesting approach is introduced by Shanmugam and Arikan [Shanmugam and Arikan 2007]. They use an image based approach and thus, the performance mainly depends on the screen resolution and not on the complexity of the scene, like it does in vertex based methods.

5 Ambient Occlusion for Animated Characters

The fourth group of ambient occlusion algorithms focus on calculating ambient occlusion for an animated character. Start position for both algorithms, explained in this section, is a model with several reference poses. For each pose the animation parameters and the ambient occlusion values (for each vertex) are available.

5.1 Ambient Occlusion for Animated Characters

This method developed by Kontkanen and Aila [Kontkanen and Aila 2006] is able to calculate ambient occlusion for animated characters. To do so, a precomputation step is need.

5.1.1 Description of the algorithm

The idea of this algorithm is to do a linear mapping between the animation parameters, such as the angle of a joint, and the ambient occlusion factor of a given vertex.

5.1.2 Precomputation

The precomputation step is necessary to find the right coefficients for the linear mapping. There are two types of input data: First, the reference poses of the animated character, for example joint angles. Second, the corresponding ambient occlusion values for each vertex. Each reference pose is stored in a vector j containing the animation parameters $j_0, j_1, ...j_{N-1}$.

With this information we can try to find a linear mapping from the reference pose to ambient occlusion values:

$$a_v = j^T t_v \tag{17}$$

where t_v is a per-vertex vector containing N coefficients.

Now that there are several reference poses, they all can be put into a matrix representation, where A are the occlusion values and J the animation parameters.

$$A = JT \tag{18}$$

In J, all reference poses are aligned in vertical order. In A, the per-vertex ambient occlusion for each reference pose is stored. In T the effect of a given animation parameter is stored for all vertices in vertical order.

Each column of T describes how the animation parameters affect a specific vertex. So we have to solve this equation system to get the coefficients in T. When there are more reference poses than animation parameters, which is normally the case, the system is over determined and there exists no exact solution. However Kontkanen and Aila [Kontkanen and Aila 2006] calculate the optima in an MSE-sense the following way:

$$T = J^+ A \tag{19}$$

where J^+ refers to the pseudo-inverse described by Golub and Loan [Golub and Loan 1996] of the animation pose matrix J. By solving this equation, we get coefficients of T which represent the linear mapping between animation parameters and ambient occlusion values for each vertex.

5.1.3 Runtime

At runtime the only thing which has to be performed for each vertex is the calculation of ambient occlusion for a given set of animation parameters. Kontkanen and Aila [Kontkanen and Aila 2006] tested it with two animated characters. One with 73 and the other with 54 scalar animation parameters. Thus the computation complexity was roughly given by 4 times of a vertex transformation.

5.1.4 Discussion

1. There is one special case which cannot be modeled with this approach. If an ambient occlusion value does not linearly depend on any animation parameter. To handle this situation an additional column with constant coefficients is added to J. So this creates a new virtual animation parameter that does not depend on any other animation parameter and though can handle this problem.

2. Another point is, that the model assumes that the ambient occlusion values linearly depend on animation parameters. Although this assumption works very well for most cases there are several in which this will not work.

3. Another assumption is, that the animation parameters are handled independently by summing up the individual occlusion values. But this does not always work. For example, if a hand shadows the nose of a character, depends on many parameters not only on the hand joint parameter. To omit this error a higher order model would be necessary but would lead to significantly more computational and storage cost. So this was ignored.

5.2 Real-Time Ambient Occlusion for Dynamic Character Skins

Kirk and Arikan [Kirk and Arikan 2007] present a method for fast ambient occlusion rendering for dynamic characters. It reduces runtime-computation and storage costs by doing a more complex precomputation step.

5.2.1 Description of the algorithm

This work is closely related to Kontkanen and Aila [Kontkanen and Aila 2006]. It also tries to find mappings between the animation parameter and the ambient occlusion values. Instead of using joint angles as input parameter they use the position of handles plus an additional point outside the handle to get a coordinate frame as shown in Figure 16.

Since a mapping between animation parameters and ambient occlusion values is not linear they developed a different calculation method. To achieve a linear mapping they exploit the fact that ambient occlusion will change nearly linear when only small changes are applied to the animation parameters.

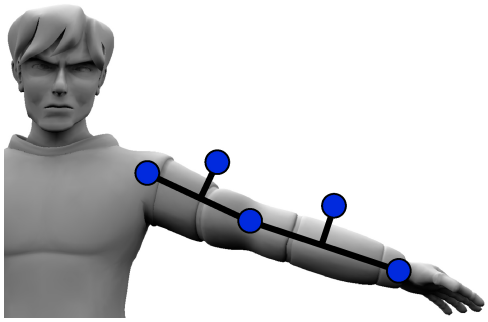

Figure 16: Illustrates the way handles are defined [Kirk and Arikan 2007].

5.2.2 Precomputation

To get only small pose changes they clustered the animation parameters with a k-means method. So each cluster has only small changes in poses. Furthermore PCA can be applied to the pose vector of the cluster, which improves memory usage significantly.

The next step is to calculate linear mappings of ambient occlusion per vertex for a given cluster. This is done in a similar way to Kontkanen and Aila [Kontkanen and Aila 2006].

To perform even better, spatial coherence is exploited. This means that vertices which are near to each other usually have similar ambient occlusion terms. So like in pose space we can perform a clustering over the vertices. Since the number of vertices can be pretty high, Kirk and Arikan [Kirk and Arikan 2007] mentioned another method than k-means. They use a greedy hierarchical clustering technique. In more detail, all vertices are put into the same cluster. The center of the cluster is calculated. The vertex with the largest distance to the center becomes an own cluster. Then the vertices are reassigned depending on the clusters centers. This process can now be recursively applied to the two clusters until a certain criteria like the maximum number of clusters is fulfilled.

5.2.3 Runtime

At runtime for a given pose the nearest pose cluster has to be found. This is done by simply finding the cluster with the nearest pose center. However, when the character is animated, several pose clusters will be transitioned and thus leading to ambient occlusion discontinuities. To omit these problem moving least squares described by Levin [Levin 1998] and Shen et al. [Shen et al. 2004] can be used. The ambient occlusion is then a weighted sum of the pose spaces.

5.2.4 Discussion

1. Kirk and Arikan [Kirk and Arikan 2007] used a test character with 4130 reference poses. Storing for every vertex one float ambient occlusion value this lead to a total memory amount of approximately 592MB. With the clustering they only needed about 23MB of memory which equals to a compression rate of over 25.

2. This ambient occlusion approach does not take occlusion by other objects into account. However, it should be possible to combine the presented method with other global ambient occlusion techniques.

5.2.5 Comparison

The methods described in the previous sections were pretty similar. Both tried to find a mapping of animation parameters to ambient occlusion values. While Kontkanen and Aila [Kontkanen and Aila 2006] use linear mapping Kirk and Arikan [Kirk and Arikan 2007] use a more complex approach. And this is the main difference between these two techniques. While the method from Kontkanen and Aila [Kontkanen and Aila 2006] is easier to implement and faster in calculation it is not able to approximate nonlinear cases. Kirk and Arikan [Kirk and Arikan 2007] on the other side need two clustering steps in precomputation. The result leads to more storage and computational cost but is therefore able to approximate nonlinear behavior.

6 Conclusion & Future Work

We presented a lot of algorithms, all capable of rendering ambient occlusion at realtime frame rates. They were grouped into four different categories. First, algorithms which work on the object level by attaching a texture onto an occluder. Second, vertex based algorithms were presented. These algorithms have the great benefit of being able to calculate nearly everything on the fly. Third, a recently introduced image based method was described. Fourth, algorithms specialized for rendering animated characters, by mapping animation parameters to ambient occlusion values, were presented.

It is remarkable that there exist a lot of very different methods to calculate ambient occlusion in realtime. So realtime ambient occlusion will still be a field of research. In future, it is to expect, that ambient occlusion will be applicable for large scale scenes and that additional features, like light bounces as shown by Bunnell [Bunnell 2005], will be added.

Realtime Ambient Occlusion will stay as a research area for years since it is for now the only way to approximate dynamic global illumination in realtime.

References

BRADSHAW, G., AND O'SULLIVAN, C. 2002. Sphere-tree construction using dynamic medial axis approximation.

BUNNELL, M. 2005. *GPU Gems 2*, har/cdr (1. april 2005) ed. Addison-Wesley Longman, 223–233.

CHUNHUI, M., SHI, J., AND WU, F. 2004. Rendering with spherical radiance transport maps. *Comput. Graph. Forum 23*, 3, 281–290.

COHEN, M. F., AND GREENBERG, D. P. 1985. The hemi-cube: a radiosity solution for complex environments. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 31–40.

DESBENOIT, B., GALIN, E., AND AKKOUCHE, S. 2004. Simulating and modeling lichen growth. Tech. Rep. RR-LIRIS-2004-007, LIRIS UMR 5205 CNRS/INSA de Lyon/Universit Claude Bernard Lyon 1/Universit Lumire Lyon 2/Ecole Centrale de Lyon, Mar.

EVERITT, C. 2001. Interactive order-independent transparency. available at http://developer.nvidia.com/object/ interactive_order_transparency.html (5th july 2007).

GIBSON, S., COOK, J., HOWARD, T., AND HUBBOLD, R. 2003. Rapid shadow generation in real-world lighting environments. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 219–229.

GOLUB, G. H., AND LOAN, C. F. V. 1996. *Matrix Computations*, third ed. Johns Hopkins University Press, Baltimore, MD.

GREGER, G., SHIRLEY, P., HUBBARD, P. M., AND GREENBERG, D. P. 1998. The irradiance volume. *IEEE Comput. Graph. Appl. 18*, 2, 32–43.

KAUTZ, J., LEHTINEN, J., AND AILA, T. 2004. Hemispherical rasterization for self-shadowing of dynamic objects. In *Proceedings of Eurographics Symposium on Rendering 2004*, Eurographics Association, 179–184.

KIRK, A. G., AND ARIKAN, O. 2007. Real-time ambient occlusion for dynamic character skins. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*.

KONTKANEN, J., AND AILA, T. 2006. Ambient occlusion for animated characters. In *Rendering Techniques 2006 (Eurographics Symposium on Rendering)*, T. A.-M. Wolfgang Heidrich, Ed., Eurographics.

KONTKANEN, J., AND LAINE, S. 2005. Ambient occlusion fields. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 41–48.

LEVIN, D. 1998. The approximation power of moving least-squares. *Mathematics of Computation 67*, 224, 1517–1531.

MALMER, M., MALMER, F., ASSARSON, U., AND HOLZSCHUCH, N. 2006. Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools*.

MENDEZ, A., SBERT, M., AND CAT, J. 2003. Real-time obscurances with color bleeding. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, ACM Press, New York, NY, USA, 171–176.

SATTLER, M., SARLETTE, R., ZACHMANN, G., AND KLEIN, R. 2004. Hardware-accelerated ambient occlusion computation. In *9th Int'l Fall Workshop VISION, MODELING, AND VISUALIZATION (VMV)*, 119–135.

SHANMUGAM, P., AND ARIKAN, O. 2007. Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 73–80.

SHEN, C., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2004. Interpolating and approximating implicit surfaces from polygon soup. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM Press, New York, NY, USA, 896–904.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. 21*, 3, 527–536.

WANG, R., ZHOU, K., SNYDER, J., LIU, X., BAO, H., PENG, Q., AND GUO, B. 2006. Variational sphere set approximation for solid objects. *The Visual Computer 22*, 9, 612–621.

ZHOU, K., HU, Y., LIN, S., GUO, B., AND SHUM, H.-Y. 2005. Precomputed shadow fields for dynamic scenes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM Press, New York, NY, USA, 1196–1201.

ZHUKOV, S., INOES, A., AND KRONIN, G. 1998. An ambient light illumination model. In *Rendering Techniques '98*, Springer-Verlag Wien New York, G. Drettakis and N. Max, Eds., Eurographics, 45–56.