Ambient Occlusion Fields

Janne Kontkanen*
Helsinki University of Technology

Samuli Laine,
Helsinki University of Technology and
Hybrid Graphics, Ltd.

Abstract

We present a novel real-time technique for computing inter-object ambient occlusion. For each occluding object, we precompute a field in the surrounding space that encodes an approximation of the occlusion caused by the object. This volumetric information is then used at run-time in a fragment program for quickly determining the shadow cast on the receiving objects. According to our results, both the computational and storage requirements are low enough for the technique to be directly applicable to computer games running on the current graphics hardware.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: shadowing techniques, games & GPUs, interactive global illumination, reflectance models & shading

1 Introduction

The idea of ambient occlusion is a special case of the obscurances technique which was first presented by Zhukov et al. [1998], but half of the credit belongs to the movie industry and the production rendering community for refining and popularizing the technique [Landis 2002; Christensen 2002]. Ambient occlusion refers to the attenuation of ambient light due to the occlusion of nearby geometry:

$$A(\mathbf{x},\mathbf{n}) := \frac{1}{\pi}\int_{\Omega} V(\mathbf{x},\omega)\lfloor\omega\cdot\mathbf{n}\rfloor\, d\omega \qquad (1)$$

Here $\mathbf{x}$ is the location and $\mathbf{n}$ is the normal vector on the receiving surface. $V(\omega,\mathbf{x})$ is the visibility function that has value zero when no geometry is visible in direction ω and one otherwise. In the above, $\int_{\Omega}$ refers to integration over a hemisphere oriented according to the surface normal $\mathbf{n}$.

Multiplying the classical ambient term [Phong 1975] with $1-A$ gives a much better looking result than the dull constant, because the ambient occlusion is able to approximate effects that are otherwise attainable only by computing full global illumination. For instance, sharp corners appear darker than open areas and objects cast plausible contact shadows on the surfaces they are resting on.

Compared to a full global illumination solution, ambient occlusion is significantly faster to compute. In addition, the self-occlusion of a rigid object can be computed as a preprocess and then re-used in different environments. Since the data can be stored in a texture map or as vertex attributes, the technique has a negligible run-time overhead. For this reason, ambient occlusion is gaining interest also in the real-time graphics community [Pharr 2004]. However, the inter-object occlusion has to be re-evaluated whenever spatial relationships of the objects change. This is in most cases affordable in offline rendering, but not in real-time applications.

*email:{janne,samuli}@tml.hut.fi

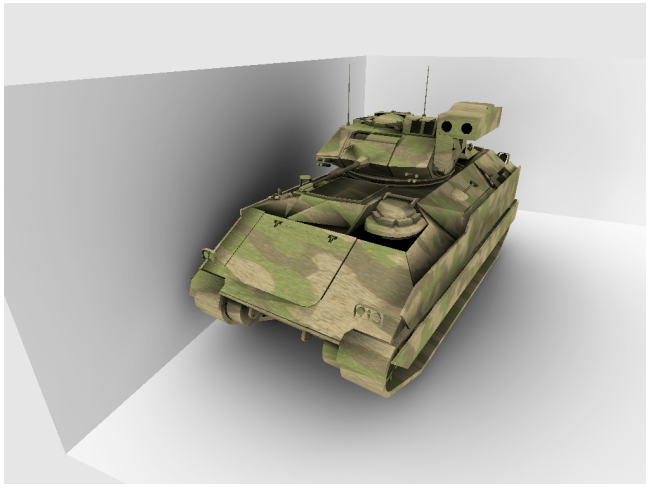

Figure 1: A tank casts shadows on the surrounding environment. The shadows are rendered using the method described in this paper. The scene runs 136 fps at 1024×768 resolution on a Pentium 4 with ATI Radeon 9800XT graphics board.

To quickly evaluate inter-object ambient occlusion we need an efficient method to compute the shadow cast by an occluder at an arbitrary position and surface orientation in the neighborhood. In this paper, we propose precomputing an *ambient occlusion field* for each occluder.

The basic idea is as follows. We approximate the occluder by a spherical cap when computing the ambient occlusion on the receiving surfaces. A spherical cap is defined by a direction and a solid angle. To quickly determine a spherical cap at an arbitrary location in the neighborhood, we precompute the subtended solid angle and the average direction of occlusion as fields around the occluder. To keep the memory requirements low, these fields are stored as radial functions into a cube-map surrounding the occluder.

The proposed method can be used for computing real-time ambient occlusion cast by rigid bodies. The run-time costs are independent of the object's complexity, and both computational costs and memory requirements are such that the approach is practically applicable to computer games. Also, the method poses no requirements for the tessellation of the receiving surface, since the run-time evaluation is done in a fragment program. Figure 1 illustrates contact shadows rendered with our method.

In the next section we discuss the previous work. The concept of the ambient occlusion fields is explained in Section 3 and the implementation in Section 4. Our main contributions are the following:

- A spherical cap approximation for fast computation of ambient occlusion (Section 3.1)

- A radial parameterization of space and simple models for the subtended solid angle and the average direction of occlusion in order to compactly store the data (Section 3.2)
- An efficient implementation that utilizes modern graphics hardware, including a fast precomputation method (Section 4)
- A practical method for combining occluders (Section 4.3)

The results are summarized in Section 5 and we discuss the future work in Section 6. For convenience, the list of used symbols is given in Table 1.

2 Previous Work

Accessibility shading [Miller 1994] is a predecessor of the ambient occlusion technique. Accessibility shading models the local variations of surface materials due to processes such as dirtying, cleaning, tearing, aging or polishing. For example, the accessibility of a surface location determines how much dirt is gathered into it. Computing ambient occlusion can be seen as computing the accessibility of light.

Ambient occlusion has become a popular technique in production rendering [Landis 2002; Christensen 2002]. It is typically computed on the surfaces of each object and stored in a texture map or as vertex attributes. The most straightforward method is to use rasterization or ray tracing to sample the hemispherical visibility around the surface of the object, but a similar result can be achieved by rendering the object from multiple directions and accumulating the visibility on each surface element. In any case, the goal is to evaluate Equation 1 on the surface of the object.

Ambient occlusion is a simplified version of the obscurances illumination model [Zhukov et al. 1998], where the visibility function V of ambient occlusion (Equation 1) is replaced by a function of distance, giving the obscurance W:

$$W(\mathbf{x},\mathbf{n}) := \frac{1}{\pi}\int_{\Omega} \rho(d(\mathbf{x},\omega))\lfloor \omega\cdot\mathbf{n}\rfloor\, d\omega \qquad (2)$$

In above $d(\mathbf{x},\omega)$ refers to the distance of the first intersection when a ray is shot from $\mathbf{x}$ towards ω. ρ is a function that maps the distance suitably. Iones et al. [2003] suggest $1-e^{\tau d}$, where τ is a user defined constant.

Mendez et al. [2003] introduce a method for dynamically updating the obscurances information in the presence of moving objects. Obscurances are re-sampled only in a selected region of the scene by utilizing temporal coherence. However, since the obscurances are evaluated per patch, the method requires a huge number of patches to account for high quality contact shadows. Thus, while the method is usable for approximating the global illumination in large scale, the fine detail present in contact shadows requires a different approach.

Sattler et al. [2004] compute the visibility from the vertices to a number of directions and utilize the coherence in the visibility function to achieve interactive frame rates with deformable objects in dynamic distant illumination. This differs from our work in that the method is intended for computing self occlusion instead of inter-object ambient occlusion effects.

Kautz et al. [2004] use a look-up-table for rasterizing occluding triangles on a hemisphere. Their method finds hemispherical occlusion quickly enough for determining shadows from a low-frequency lighting environment at interactive rates. While this method is intended primarily for computing self-shadows, it can be used also for inter-object shadows. However, due to efficiency reasons, the evaluation must be done per-vertex and thus artifacts are hard to avoid without highly tessellated geometry.

Variable	meaning
$\mathbf{x}$	3D position
ω	direction vector
$\hat{\omega}$	discretized direction vector
$\mathbf{n}$	surface normal
$V(\mathbf{x},\omega)$	visibility function
$A(\mathbf{x},\mathbf{n})$	ambient occlusion
$\tilde{A}(\mathbf{x},\mathbf{n})$	approximate ambient occlusion
$A_k(\mathbf{x},\mathbf{n})$	ambient occlusion by object k
$\Omega(\mathbf{x})$	subtended solid angle of occluder
$\tilde{\Omega}(\mathbf{x})$	approximation of Ω
$\Upsilon(\mathbf{x})$	average direction of occluder
$\tilde{\Upsilon}(\mathbf{x})$	approximation of Υ
r	distance from origin
$a(\omega),b(\omega),c(\omega)$	parameters for function $\tilde{O}$
$\mathbf{C}_0(\omega)$	characteristic center of occlusion
p	controls the shadow in the convex hull
$r_0(\omega)$	distance of convex hull from origin

Table 1: List of used symbols.

Our method can be seen as a real-time shadow algorithm for ambient light. Most research on real-time shadow algorithms concentrate on point light sources or relatively small area lights. Hasenfratz et al. [2003] give an extensive survey on the real-time soft shadow algorithms.

Deep shadow maps [Lokovic and Veach 2000] store a piecewise linear function into each shadow map pixel to represent the fractional visibility of the light source. Unlike conventional shadow mapping, deep shadow maps can model shadows cast by participating media or partially transparent surfaces. However, since the deep shadow maps work only for point lights the technique is not usable for ambient light. The idea of storing functions into texture maps is used widely. For example, Malzbender et al. [2001] use biquadratic polynomial textures to express appearance of an image in varying lighting.

In addition to shadowing techniques, the reverse problem of quickly computing the illumination reflected to the neighborhood of the object is closely related to our work. Sloan et al. [2002] present neighborhood transfer as an application of precomputed radiance transfer. For each point in a 3D grid surrounding an object, they precompute a linear operator that maps distant incident illumination to the neighborhood of the object. This method is able to express full light flow from the low-frequency lighting environment to the space around the the object. Compared to our method, neighborhood transfer is more general, but both storage and computational requirements are much higher. To accurately represent contact shadows, the resolution of the grid needs to be prohibitively high.

Chunhui et al. [2004] suggest precomputing spherical radiance transport maps (SRTMs) to account for inter-object shadowing and caustics due to distant illumination. The illumination is modeled as a set of directional lights. The method solves inter-object shadowing by pre-rasterizing coverage masks of occluder from a discrete set of directions and uses these coverage masks to look up the occlusion of each directional light when rendering the receivers. This approach results in a significant number of look-ups, but real-time performance is achieved by clustering of light sources and careful low-level optimization. However, since the shading is computed per vertex, the method requires finely tessellated receiving geometry to accurately compute contact shadows.

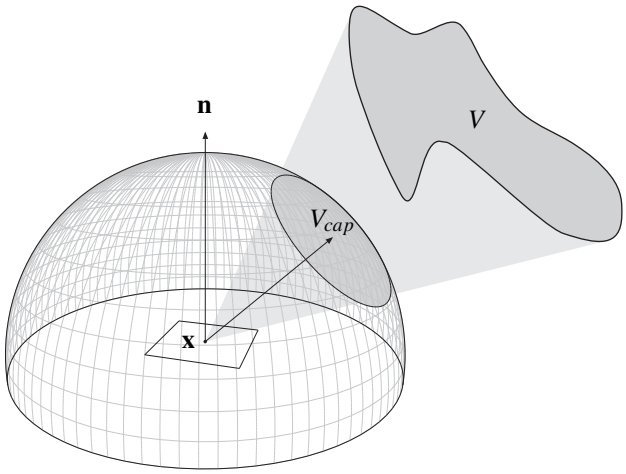

Figure 2: A spherical cap approximation for an occluder. An approximate ambient occlusion is computed at each $\mathbf{x}$ with surface normal $\mathbf{n}$ by approximating the visibility function $V(\mathbf{x},\omega)$ by a visibility of a corresponding spherical cap $V_{cap}(\mathbf{x},\omega)$. The size of the spherical cap is determined so that it subtends the same solid angle as the occluder. The direction of the cap is given by the average of the occluded directions.

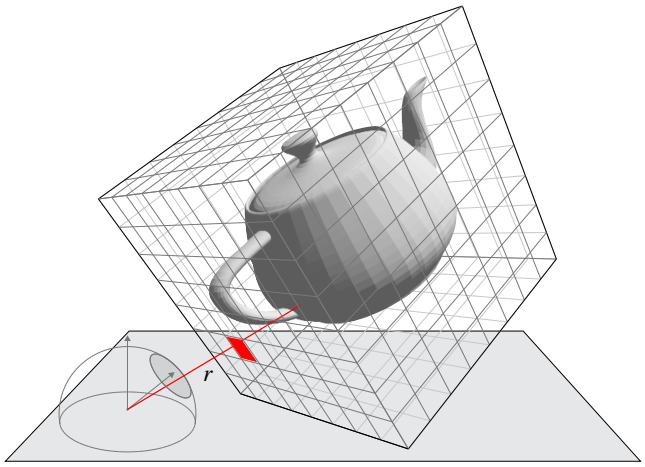

Figure 3: The solid angle Ω subtended by an object and the average direction of occlusion Υ are stored for each direction as functions of distance r. At run-time these functions are fetched from a cube-map and evaluated at the receiving surface in order to compute ambient occlusion.

3 Ambient Occlusion Fields

In this section we introduce our spherical cap approximation for computing ambient occlusion and define the fields for the *solid angle* and the *average direction of occlusion*. Then we discuss how to express the fields compactly as radial functions, and finally how the special case of a receiver entering the convex hull of the occluder is treated. In this section the fields are assumed to be continuous; discretizing for storing into cube-maps is discussed in Section 4.

3.1 Spherical Cap Approximation

To quickly determine ambient occlusion at an arbitrary point in space around an occluder, we approximate the visibility of the occluder with a spherical cap (see Figure 2).

The spherical cap approximation works well in practice, since the ambient occlusion integral (Equation 1) is a smooth, cosine weighted average of the visibility function over a hemisphere. The result is not very sensitive to variations in the shape of the visibility function as long as the proportions of $V=0$ and $V=1$ are approximately correct.

We define $V_{cap}(\mathbf{x},\omega)$ as the visibility function of a cap from position $\mathbf{x}$ towards direction ω. Thus for each location $\mathbf{x}$ there is a corresponding spherical cap representing the occluder. $V_{cap}(\mathbf{x},\omega)$ has value one when ω falls within the cap and zero otherwise. Substituting this into Equation 1 gives an approximation of ambient occlusion:

$$\tilde{A}(\mathbf{x},\mathbf{n}) = \frac{1}{\pi}\int_{\Omega} V_{cap}(\mathbf{x},\omega)\lfloor\omega\cdot\mathbf{n}\rfloor\, d\omega \tag{3}$$

To evaluate the above integral we need to be able to construct the function $V_{cap}(\mathbf{x},\omega)$. We determine the size of the spherical cap from the solid angle subtended by the occluder. This is defined at position $\mathbf{x}$ as:

$$\Omega(\mathbf{x}) := \int_{\Theta} V(\mathbf{x},\omega)\, d\omega \tag{4}$$

Here Θ refers to integration over a sphere. $\Omega(\mathbf{x})$ equals 4π when the object is blocking every direction as seen from point $\mathbf{x}$ and zero when the object is not visible at all. As the direction of the cap, we use the average direction of occlusion:

$$\Upsilon(\mathbf{x}) := \text{normalize}\,\Big(\int_{\Theta} V(\mathbf{x},\omega)\omega\, d\omega\Big) \tag{5}$$

Thus $\Upsilon(\mathbf{x})$ is the componentwise average of all the directions ω for which the visibility function $V(\mathbf{x},\omega)$ evaluates to one.

3.2 Storing Ω and Υ

Now that we have means for evaluating ambient occlusion on an arbitrary surface point based on vector $\Upsilon(\mathbf{x})$ and scalar $\Omega(\mathbf{x})$, we consider how to store these fields compactly in 3D. Since our goal is to express accurate contact shadows, we need high resolution near the object and lower resolution suffices at larger distances. Optimally, the resolution would depend on the distance from the surface of the object. However, since this is not easily achieved in practice, we use a radial parameterization. We parameterize the space by direction ω and distance r from the center of the occluder and express the fields with respect to these parameters: $\Omega(\mathbf{x}) = \Omega(\omega,r), \Upsilon(\mathbf{x}) = \Upsilon(\omega,r)$.

To compactly store Ω and Υ, we assume that given a direction ω, the fields behave predictably as functions of radial distance. Thus in the following we construct models for both quantities as functions of r. This setup is illustrated in Figure 3. For an efficient representation of $\Omega(\omega,r)$ we use the knowledge that the solid angle subtended by an object is approximately proportional to the inverse square root of r. To capture this, we use the following model to express the solid angle:

$$\Omega(\omega,r) \approx \tilde{\Omega}(\omega,r) = \frac{1}{a(\omega)r^2 + b(\omega)r + c(\omega)} \tag{6}$$

In order to fit the above model to data, the coefficients $a(\omega)$, $b(\omega)$ and $c(\omega)$ have to be determined for each direction ω.

To find a model for the average direction Υ, we note that when going farther away from the object, the average direction approaches the

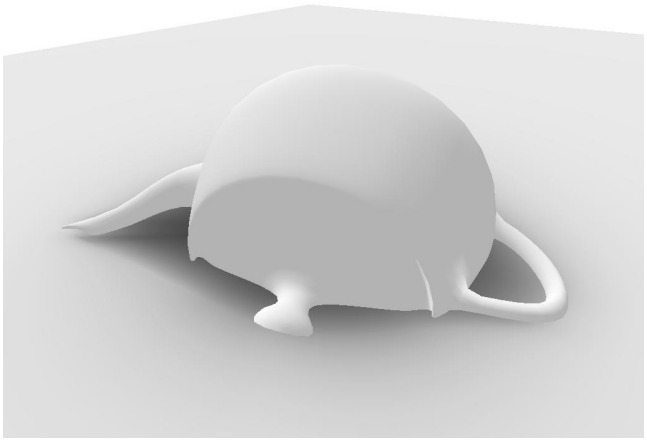

Figure 4: A teapot intersected by a ground plane. The computationally cheap method of approximating the ambient occlusion inside the convex hull gives a perceptually pleasing result.

direction towards the center point of the object, while in the close proximity the direction might deviate considerably. We model this by:

$$\Upsilon(\omega, r) \approx \tilde{\Upsilon}(\omega, r) = \text{normalize}\,(\mathbf{C}_o(\omega) - r\omega) \tag{7}$$

In above, given a direction ω, $\mathbf{C}_o(\omega)$ can be understood intuitively as a characteristic point of the occluder, i.e a point in space in which the direction of occlusion mostly points. The approximation sets $\tilde{\Upsilon}(\omega, r)$ to point from (ω, r) to $\mathbf{C}_o(\omega)$. The challenge of fitting this model to data is to find $\mathbf{C}_o$ that minimizes a suitable error metric. The fitting procedure is discussed in Section 4.1.

3.3 Inside the Convex Hull

A radial line originating from the center of the occluder may intersect the object multiple times. This introduces discontinuities in functions $\Omega(\omega, r)$ and $\Upsilon(\omega, r)$, making both piecewise smooth functions of r. However, in a practical implementation we cannot afford to use a piecewise representation, since this would complicate the look-up and make the amount of stored data much larger. Instead, we make the assumption that the receiving geometry seldom enters the convex hull of the object, and thus it is sufficient that the data is accurate outside the convex hull. Inside the convex hull and outside the occluder itself, we map the ambient occlusion $\tilde{A}$ towards constant A_0 according to the following formula:

$$\begin{aligned} t &= (r / r_0(\omega))^p \\ \tilde{A}(\omega, r) &= t\tilde{A}(\omega, r_0(\omega)) + (1-t)A_0 \end{aligned} \tag{8}$$

where r_0 is the radial distance from the center point of the object to the convex hull, which must be stored for each direction independently. Parameter p allows to adjust how fast the shadow is mapped towards A_0 as we go towards the center. Figure 4 illustrates that the above solution gives satisfactory shadows inside the convex hull.

4 Implementation

In this section we present an efficient implementation of the ambient occlusion fields. We devote separate subsections for preprocessing and run-time components. Finally we discuss how to combine the effects of multiple occluders and present two alternative rendering algorithms.

4.1 Preprocess

As a preprocess, we use the method of least squares to fit the models $\tilde{\Omega}$ and $\tilde{\Upsilon}$ to the computed occlusion and direction samples. Then these radially parameterized functions are stored into two cube-maps surrounding the object. We denote the resulting discretized direction with $\hat{\omega}$. The stored components are:

$a(\hat{\omega})$,$b(\hat{\omega})$,$c(\hat{\omega})$ for $\tilde{\Omega}$	3
$\mathbf{C}_o(\hat{\omega})$ for $\tilde{\Upsilon}$	3
$r_0(\hat{\omega})$, distance from the center to the convex hull	1
Total	7 scalars

Ray tracing or rasterization can be used for computing the samples. We chose rasterization to be able to utilize graphics hardware. To compute Ω and Υ for a single location, six images have to be rasterized to account for all directions. The images contain binary information indicating whether the occluder is visible in a certain direction or not. The images are read from the graphics card to the main memory, and Ω and Υ are computed according to Equations 4 and 5. Since the read-back from the graphics card is a bottleneck in this approach, we use the stencil buffer and frame buffer blending to store the bitmasks on separate bitplanes. With this optimization, we can fetch 24 bitmasks corresponding to three 8-bit color components from the graphics card at once. In our experiments, the preprocessing times ranged from 1 to 66 minutes (see the results in Section 5).

We concentrate the computational effort and accuracy to close proximity of the surface by distributing the sampling points in logarithmic scale with respect to r. We do not sample inside the convex hull or farther than a predefined limit.

To fit $\tilde{\Omega}$ for each $\hat{\omega}$ we minimize the following error:

$$\varepsilon_\Omega(\hat{\omega}) = \sum_{i=1}^{N} (\Omega(\hat{\omega}, r_i) - \tilde{\Omega}(\hat{\omega}, r_i))^2 \tag{9}$$

where $\Omega(\hat{\omega}, r_i)$ refers to the sampled occlusion value (Equation 4) and $\tilde{\Omega}(\hat{\omega}, r_i)$ to the approximated value (Equation 6) at the sampling location $(\hat{\omega}, r_i)$. N is the number of sampling locations. The fitting process yields the coefficients a, b and c for each direction $\hat{\omega}$.

To fit the model for the average direction $\tilde{\Upsilon}$, we minimize the deviation from the sampled directions:

$$\varepsilon_\Upsilon(\hat{\omega}) = \sum_{i=1}^{N} (1 - \tilde{\Upsilon}(\hat{\omega}, r_i) \cdot \Upsilon(\hat{\omega}, r_i))^2 \tag{10}$$

where $\Upsilon(\hat{\omega}, r_i)$ refers to the sampled average direction (Equation 5) and $\tilde{\Upsilon}(\hat{\omega}, r_i)$ to the approximated direction (Equation 7). The optimization yields $\mathbf{C}_o(\hat{\omega})$, the characteristic point of the occluder.

While ambient occlusion is in general spatially smooth, there are certain cases where the function might contain high frequency detail and care must be taken to avoid undersampling artifacts. We use the standard solution for the aliasing problem by low-pass filtering the signal before sampling. This can be done by constructing a high resolution cube-map and downsampling it with a suitable low-pass reconstruction filter. Typical undersampling artifacts and a Gaussian-filtered result are shown in Figure 5.

Even extremely low cube-map resolutions such as 8×8 can be used to render visually pleasing, smooth ambient occlusion. Since current graphics hardware does not perform bilinear filtering across

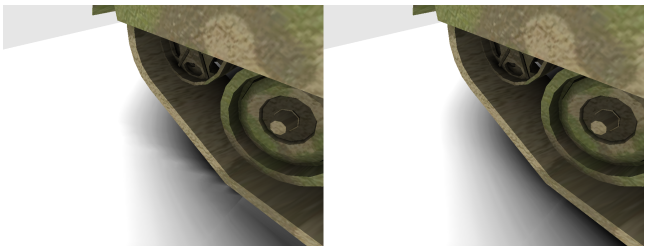

Figure 5: **Left:** Artifacts caused by undersampling. **Right:** The result achieved by downsampling from a higher resolution with a Gaussian reconstruction filter. Both images are using the same 32×32 resolution for the cube maps. Contrast has been adjusted to highlight the artifact.

the edges of a cube-map, border texels must be used to avoid visible seams. Thus, each cube-map face should have one-texel-wide border which duplicates the closest texels from the neighboring face and the border texels setting should be set on. The quality of shadows with different cube-map resolutions is illustrated in Figure 8.

4.2 Run-time

When rendering a receiving surface, the polynomials are fetched from the cube-map associated with the occluder. To compute the ambient occlusion from the direction Υ and subtended solid angle Ω, we need to integrate a cosine weighted spherical cap according to Equation 3. We derived an analytic formula for this purpose, but it turned out to be rather expensive to evaluate it in a fragment program. As a solution, we computed a small look-up-table parameterized by solid angle and the elevation angle relative to surface. Note that the azimuth angle can be ignored, since it does not affect to the result.

The functions $\tilde{\Omega}(\omega, r)$ and $\tilde{\Upsilon}(\omega, r)$ may diverge from the true Ω and Υ as we go outside the range in which they were fitted. The solution is to ramp the ambient occlusion towards zero beyond a certain limit distance. This makes sense also for performance reasons, because it makes the region of influence of the occluder finite, and thus allows us to ignore distant occluders early in the rendering pipeline.

The whole process of looking up polynomials and computing the resulting ambient occlusion values is done in a fragment program, giving per-pixel accuracy and a good performance. A pseudocode for the fragment program is given in Algorithm 1.

Compiling the shader written in Cg to OpenGL ARB assembly language gives 47 instructions in total containing three texture fetches corresponding to the occlusion and direction functions, and the spherical cap integration look-up-table.

4.3 Combining Occluders

To make the ambient occlusion fields feasible in practice, we must consider how to combine shadows from multiple casters efficiently. In theory, two spherical caps representing occluders could be combined for example by using a look-up-table, but since the spherical cap is already a rather rough approximation for an arbitrary object, this is not worth the extra computation. In practice, our choice is simply multiplicatively blend $1-A$ for each occluder to the framebuffer. In the following we argue that this choice has some theoretical justifications. Interestingly, our considerations are similar to

Algorithm 1 Pseudocode for the fragment program used to render the receiving surfaces.

Input:
- $\mathbf{x}$ receiver position in field space
- $\mathbf{n}$ receiver surface normal in fields space

Output:
- $\tilde{A}$ ambient occlusion value

Constants:
- r_{near} near falloff distance
- r_{far} far falloff distance
- A_0 inside-hull ambient occlusion
- p inside-hull falloff

Textures:
- $\mathbf{T}_{Occ}$ occlusion data cube-map
- $\mathbf{T}_{\mathbf{C_0}}$ direction data cube-map
- $\mathbf{T}_{LUT}$ spherical cap integration look-up-table

$r = \|\mathbf{x}\|$
$\omega = \texttt{normalize}(\mathbf{x})$
if $r > r_{far}$ **then** `discard`
$(a, b, c, r_0) = \mathbf{T}_{Occ}[\omega]$
$r_{clamp} = \texttt{max}(r, r_0)$
$\tilde{\Omega} = (a r_{clamp}^2 + b r_{clamp} + c)^{-1}$
$\tilde{\Upsilon} = \texttt{normalize}(\mathbf{T}_{\mathbf{C_0}}[\omega] - r_{clamp}\omega)$
$\tilde{A} = \mathbf{T}_{LUT}[\tilde{\Omega}, \mathbf{n} \cdot \tilde{\Upsilon}]$
if $r < r_0$ **then**
  $t = (r/r_0)^p$
  $\tilde{A} = t\tilde{A} + (1-t)A_0$
end if
$\tilde{A} = \tilde{A} * \texttt{smoothstep}(r_{far}, r_{near}, r)$
return $\tilde{A}$

those of Porter and Duff [1984] in the context of image compositing. For the sake of simplicity, we consider the case of two occluders, but the results are directly applicable to an arbitrary number of occluders.

To exactly compute the ambient occlusion of two occluders a and b, would require the evaluation of the following integral:

$$A_{ab}(\mathbf{x}, \mathbf{n}) = \frac{1}{\pi} \int V_{ab}(\mathbf{x}, \omega) \lfloor \omega \cdot \mathbf{n} \rfloor d\omega \qquad (11)$$

where $V_{ab}(\mathbf{x}, \omega)$ is the combined visibility function of the objects a and b, i.e., it is one when at least one of the objects is visible in

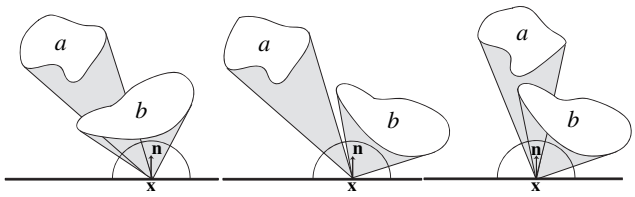

Figure 6: A 2D-illustration of the three different ways two occluders may block point $\mathbf{x}$ on a receiving surface. **Left:** Object a is completely occluded by object b, and just picking up the bigger ambient occlusion, $max(A_a, A_b)$, would yield the correct combined ambient occlusion A_{ab}. **Middle:** The occluders do not overlap each other, and the correct result would be given by adding the ambient occlusion terms of the objects: $A_{ab} = A_a + A_b$. **Right:** The interaction of the occluders is more complex, but the ambient occlusion is always between $max(A_a, A_b)$ and $A_a + A_b$

direction ω and zero otherwise.

We want to get an approximate value for A_{ab} by utilizing the known ambient occlusion values A_a and A_b. We are not using knowledge about V_a or V_b, but we may think of three different cases that are illustrated in Figure 6. When an occluder completely overlaps the other one, the combined ambient occlusion is given by picking up the larger of the values A_a and A_b. When the occluders do not overlap at all the value is given by the sum of the ambient occlusions of each object. It is easy to see that these two cases represent the extremes and the combined ambient occlusion A_{ab} always satisfies:

$$max(A_a, A_b) \leqslant A_{ab} \leqslant A_a + A_b$$

Multiplicative blending of $1 - A$ for each object satisfies the above inequality. In addition, it can be shown that a ray shot from a receiving surface with a cosine weighted probability distribution, hits occluding geometry with probability equal to ambient occlusion. Thus, if we understand A_x as probability of hitting object x, A_{ab} can be interpreted as probability of hitting either or both of the objects. Assuming that A_a and A_b are uncorrelated and combining the probabilities by elementary probability theory yields:

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

This suggests the multiplicative blending of the shadow casted by each occluder into the framebuffer. Note that the result is probabilistic and thus only an approximation for Equation 11. However, as illustrated in Figure 7, the resulting quality is perceptually satisfactory.

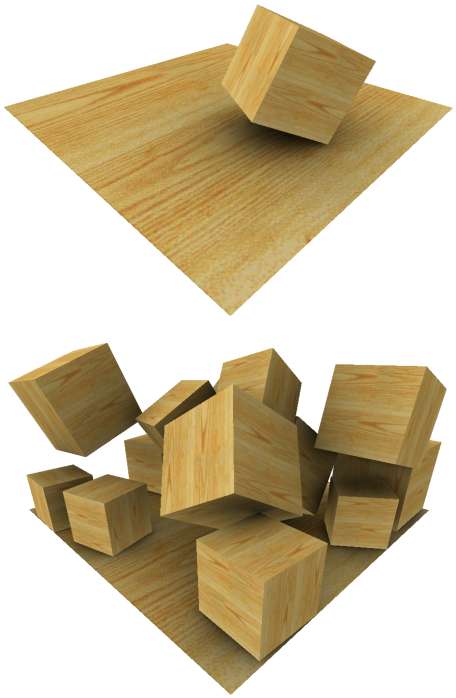

Figure 7: The shadows are combined efficiently from multiple casters by simple multiplicative blending. The cubes cast shadows to each other and to the ground plane. This scene runs at 17 fps with 400 object-to-object shadowing interactions. In addition to the contact shadows, notice the overall darkening of the ground plane in the bottom image. See the accompanying animation sequence.

4.4 Rendering Algorithms for Multiple Occluders

Two alternative algorithms for rendering scenes with multiple occluders are described in Algorithms 2 and 3. The first algorithm renders the receivers once for each occluder. After the first pass only the visible fragments have to be processed as the hidden fragments can be rejected by an early depth test. Despite of this, in real-world applications some receivers such as the static surroundings are often expensive to render, and thus a different algorithm should be used. Algorithm 3 utilizes deferred shading [Deering et al. 1988] to process the scene only once and then re-use the stored visible fragments to render the illumination. This approach is more efficient in practice as the performance degrades linearly to the number of fragments shaded times the number of occluding objects.

Algorithm 2 Simple algorithm for rendering ambient occlusion from multiple casters

```
Render the scene once with ambient light only
for all occluders, O do
  for all receivers, R do
    if R in the region of influence of O then
      Render R with ambient occlusion field of O with multiplicative blending
    end if
  end for
end for
```

Algorithm 3 Deferred shading-based algorithm

```
Render the scene with ambient light only and store the required fragment data
for all occluders, O do
  for all fragments, F do
    if F in the region of influence of O then
      Render F with ambient occlusion field of O with multiplicative blending
    end if
  end for
end for
```

5 Results

We measured the precomputation times and run-time performance on a desktop PC with Pentium 4 running at 2.8 GHz, 1GB RAM and an ATI Radeon 9800XT graphics board. The results are summarized in Table 2.

As seen from the results, the memory consumption and the preprocessing time increases quadratically with the cube-map resolution, but the run-time cost remains approximately the same. We used a 16-bit fixed point format to store the data into cube-maps, so the run-time memory consumption is $(N+2)^2 \times 6 \times 7 \times 2$ bytes with $N \times N$ cube-map resolution. Additional $+2$ is due to border texels. Note that in the precomputation we super-sampled the cube-maps with a double resolution to avoid aliasing artifacts (see Section 4.1). The resolutions shown in the table are the run-time resolutions.

Both the storage cost and the run-time overhead are clearly low enough so that the method is suitable for use in real-time applications that have demanding performance requirements such as computer games.

scene	run-time cube-map resolution	memory consumption in bytes	preprocess time	fps
Tank	8×8	8400	1 min 2 s	136
Tank	16×16	27 216	4 min 6 s	136
Tank	32×32	97 104	16 min 29 s	136
Tank	64×64	365 904	66 min	136
1 Cube	32×32	97 104	5 min 16 s	500
5 Cubes	32×32	97 104	5 min 16 s	95
10 Cubes	32×32	97 104	5 min 16 s	44
20 Cubes	32×32	97 104	5 min 16 s	16

Table 2: The results: memory consumption of cube-maps, preprocessing times and run-time fps. All scenes where rendered at 1024×768 resolution.

6 Discussion & Future Work

We have presented a practical solution for computing inter-object ambient occlusion. We hope that the method gains popularity in the computer game industry and stimulates future work. Our view is that ambient occlusion has not yet been fully exploited either in game industry or in research.

There are many directions for future work. For example, it might be fruitful to consider other kinds of models for storing the fields to further increase the accuracy of the method. Also, while this paper has dealt with constant ambient illumination, it might be possible to efficiently integrate non-constant illumination against a spherical cap as well. This would require to use a spherical cap to represent the unobstructed instead of the obstructed directions. Then this inverse spherical cap could be used for a bent normal -like look-up from a prefiltered environment map [Landis 2002].

Acknowledgements

The authors would like to thank Timo Aila, Jaakko Lehtinen and Lauri Savioja for fruitful discussions and helpful comments. This work was funded by National Technology Agency of Finland, Bitboys, Hybrid Graphics, Nokia, and Remedy Entertainment.

References

CHRISTENSEN, P. H. 2002. Note #35: Ambient Occlusion, Image-Based Illumination, and Global Illumination. *PhotoRealistic RenderMan Application Notes*.

CHUNHUI, M., JIAOYING, S., AND FULI, W. 2004. Rendering with Spherical Radiance Transport Maps. *Computer Graphics Forum 23*, 3, 281–281.

DEERING, M., WINNER, S., SCHEDIWY, B., DUFFY, C., AND HUNT, N. 1988. The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, 21–30.

HASENFRATZ, J.-M., LAPIERRE, M., HOLZSCHUCH, N., AND SILLION, F. 2003. A Survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum 22*, 4.

IONES, A., KRUPKIN, A., SBERT, M., AND ZHUKOV, S. 2003. Fast, Realistic Lighting for Video Games. *IEEE Computer Graphics and Applications 23*, 3, 54–64.

KAUTZ, J., LEHTINEN, J., AND AILA, T. 2004. Hemispherical Rasterization for Self-Shadowing of Dynamic Objects. In *Rendering Techniques 2004 (Proceedings of Eurographics Symposium on Rendering)*, 179–184.

LANDIS, H., 2002. RenderMan in Production, ACM SIGGRAPH 2002 Course 16.

LOKOVIC, T., AND VEACH, E. 2000. Deep Shadow Maps. In *Proceedings of ACM SIGGRAPH 2000*, 385–392.

MALZBENDER, T., GELB, D., AND WOLTERS, H. 2001. Polynomial Texture Maps. In *Proceedings of ACM SIGGRAPH 2001*, 519–528.

MÉNDEZ, A., SBERT, M., AND CATÀ, J. 2003. Real-time Obscurances with Color Bleeding. In *Proceedings of the 19th spring conference on Computer graphics*, 171–176.

MILLER, G. 1994. Efficient algorithms for local and global accessibility shading. In *Proceedings of ACM SIGGRAPH 94*, 319–326.

PHARR, M. 2004. Ambient occlusion. In *GPU Gems*, R. Fernando, Ed., 667–692.

PHONG, B.-T. 1975. Illumination for computer generated pictures. *CACM June 1975 18*, 6, 311–317.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, 253–259.

SATTLER, M., SARLETTE, R., ZACHMANN, G., AND KLEIN, R., 2004. Hardware-accelerated ambient occlusion computation. To appear in the proceedings of International Fall Workshop on Vision, Modeling, and Visualization 2004.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. In *Proceedings of ACM SIGGRAPH 2002*, 527–536.

ZHUKOV, S., IONES, A., AND KRONIN, G. 1998. An ambient light illumination model. In *Rendering Techniques '98 (Proceedings of the Eurographics Workshop on Rendering)*, 45–55.

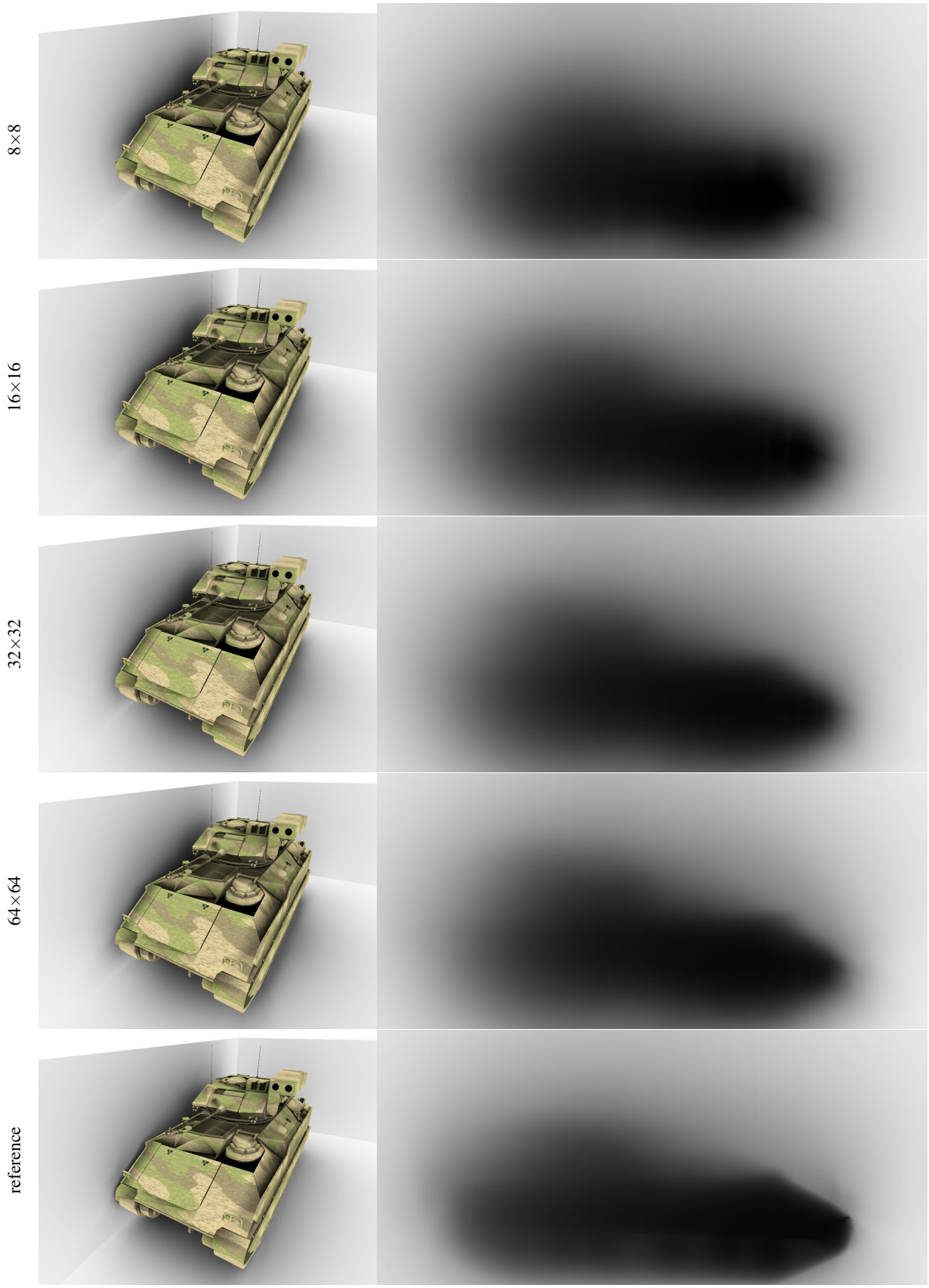

Figure 8: The ambient occlusion computed with different cube-map resolutions. The left column shows a tank casting contact shadows on the box. The right column shows the shadow cast on the side wall for closer inspection. The resolutions range from 8×8 for each cube-map side to 64×64 and the bottom row shows a reference image computed by sampling the ambient occlusion for each pixel (the computation took several hours). The shadows on the box have been computed with our method, while the tank has precomputed self shadows.