Accurately blending the occlusion values and directions between two cones Supplemental material to *Fast Precomputed Ambient Occlusion*

Mattias Malmer[1], Fredrik Malmer[1], Ulf Assarsson[2,3] & Nicolas Holzschuch[3]
[1] Syndicate Ent.
[2] Chalmers University of Technology
[3] ARTIS-GRAVIR/IMAG INRIA*

July 10, 2006

1 Introduction

In this supplemental material, we give a more accurate algorithm for blending the occlusion values between two ambient occlusion cones. We assume that we have several occluders in a scene. For each occluder, we can compute the ambient occlusion as a cone of occluded directions, defined by a direction $\boldsymbol{d}$ and an aperture α. Combining the occlusion between two occluders implies computing the direction and aperture of the equivalent cone.

Ideally, we would like to treat the occluders sequentially, maintaining an occlusion cone containing the effects of all the previously treated occluders. For each occluder, we would retrieve the combined occlusion cone, blend this cone with the occlusion cone from the occluder, and write the result as the new combined occlusion cone. Unfortunately, this technique requires the ability to read and write to the same buffer in a fragment program, or programmable blending capacities. Neither of them are available on current graphics hardware, so we had to design a workaround: we keep two separate buffers. When we render an occluder, we read the combined occlusion value from the first buffer, then we blend it with the occlusion value from the current occluder, and write the result to the second buffer. Once the occluder has been processed, we copy the second buffer into the first, then start processing the next occluder. This copying slows down the computation and imposes a synchronization between occluders. As read/write buffers become available, the need for this workaround will disappear.

2 Algorithm

At each step, we have two occlusion cones: one for the current occluder, A, and one cone representing the combined occlusion for all previously treated occluders, B, each of them defined by an axis $\boldsymbol{d}_i$ and an aperture α_i (see Figure 1). We combine together the occlusion for these two cones (see figure 2). In the general case, the two cones are not disjoint, so we cannot satisfy ourselves with summing their occluded areas and averaging their central directions. Instead, we use a lookup table T_{merge} that returns the union of the two ambient-occlusion cones, taking their possible intersection into account.

Our algorithm works as follows: first, we clip the occlusion cone from the current occluder against the tangent plane to the receiver, using the T_{clip} lookup table. This clipping gives us a new occlusion cone, with axis $\boldsymbol{d}_{A'}$ and aperture $\alpha_{A'}$. We then retrieve the cone resulting from combining previous occluders, as

*ARTIS is a research team in the GRAVIR laboratory, a joint research unit of CNRS, INRIA, INPG and UJF.

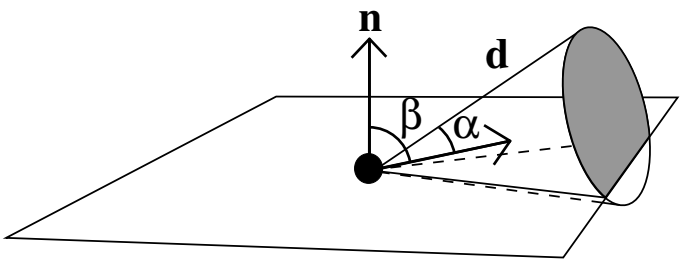

Figure 1: Each occluder cone is defined by its axis and its aperture.

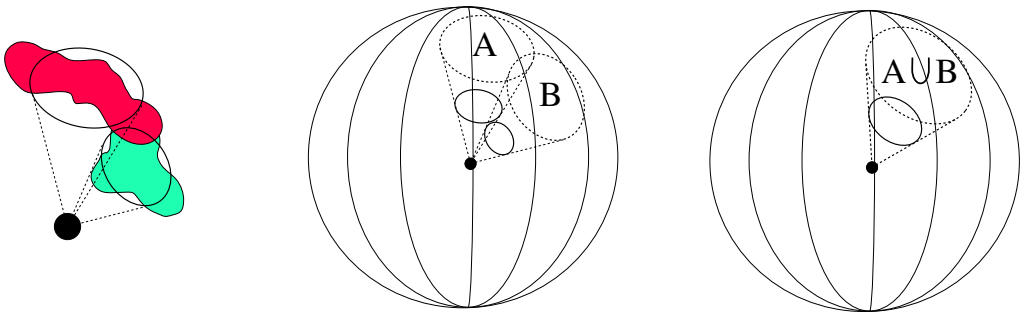

Figure 2: The occlusion from two separate shadow casters (a) are combined by replacing their two cones (b) by a resulting new cone, representing the occlusion of their union (c).

$(\boldsymbol{d}_B, \alpha_B)$, and the T_{merge} lookup texture gives us the new combined occlusion cone, which we write onto the write buffer.

The fragment shader from the main paper (section 3.3.1) needs the following extension:

FRAGMENT SHADER

```
7  float2 (αA, f) = texRECT(Tclip, float2(αA, βA))
8  float3 dA = normalize(lerp(dA, n, f))
9  float4 (dB, αB) = texRECT(OcclusionBuffer, pscreen )
10 out.color.xyz = dA * αA + dB // dB premultiplied
11 float cosγ = dot(dA, dB)
12 out.color.w = tex3D(Tmerge , float3(αA , αB , cosγ ))
```

Note on line 10, that dB is stored in the occlusion buffer as premultiplied with α_B . `out.color.xyz` is the major direction of occlusion at the pixel, and `out.color.w` is the combined occlusion value.

The formula and code for precomputing T_{merge} are given in the following section.

3 Computing T_{merge}

As for computing T_{clip}, we want to compute the area of the intersection between the two cones, then substract this value from the sum of the areas of both cones.

The intersection is defined as a surface integral. By performing partial integration, we convert it to a 1D integral; we then perform numerical integration to compute the value of this 1D integral.

We start with two cones, defined by their axes and apertures: $(\boldsymbol{d}_A, \alpha_A)$ and $(\boldsymbol{d}_B, \alpha_B)$. Without loss of generality, we can assume that $\alpha_A > \alpha_B$. We compute $\gamma = \arccos(\boldsymbol{d}_A \cdot \boldsymbol{d}_B)$. If $\gamma > \alpha_A + \alpha_B$, then the intersection is obviously empty. Similarly, if $\gamma < \alpha_A - \alpha_B$, then the intersection is equal to the area of the smaller cone, $(\boldsymbol{d}_B, \alpha_B)$:

```
float intersect(float alphaA, float alphaB, float gamma)
{
```

```
  if (alphaA < alphaB) swap(alphaA, alphaB);
  if (beta >= alphaA + alphaB) return 0;
  if (beta <= alphaA - alphaB) return alphaB

  return intersect2(alphaA, alphaB, gamma);
}
```

In the general case, the intersection is equal to:

$$I = \int_{\gamma-\alpha_B}^{\alpha_A} 2\arccos\left[\frac{\cos\alpha_B - \cos\theta\cos\gamma}{\sin\theta\sin\gamma}\right]\sin\theta\,d\theta \tag{1}$$

(for proof of this formula, see Section 4). We sample this integral using the following code:

```
int num_samples = 1000;
float float_epsilon = 1e-4;
float intersect2(float alphaA, float alphaB, float gamma)
{
  // The limits of the integral:
  float lim_inf = gamma - alphaB;
  if (lim_inf < 0) lim_inf  = 0;
  float lim_sup = alphaA;
  if (lim_sup <= lim_inf + float_epsilon) return 0;

  float theta;
  float sum = 0;
  float delta = (lim_sup - lim_inf)/num_samples;
  for (theta = lim_inf; theta < lim_sup; theta+= delta) {
    float g = (cos(alphaB) - cos(theta)*cos(gamma))/(sin(theta)*sin(gamma));
    if (g > 1) g = 1;
    if (g < -1) g = -1;
    sum += 2*acos(g)*sin(theta)*delta;
  }
  return sum/(4*M_PI);
}
```

4 Intersection of two cones

The proof of equation 1 is similar to computing the area of a cone clipped by a plane, in the first supplemental material. We parameterize in spherical coordinates, using the axis of the largest cone as the axis of the parameterization. We can orient our referential so that the axis of the smallest cone has coordinates:

$$\boldsymbol{d}_B = \begin{pmatrix} \sin\gamma \\ 0 \\ \cos\gamma \end{pmatrix}$$

The integral we are trying to compute is defined by:

$$I = \int_{A\cup B} \sin\theta\,d\theta\,d\varphi, \tag{2}$$

where the integral is on the intersection between the two cones. The fact that the point (θ,φ) lies in the first cone translates as a condition over theta:

$$\theta < \alpha_A \tag{3}$$

The fact that the point (θ, φ) lies in the second cone translates as a condition over the angle between the point and the axis of the cone:

$$\boldsymbol{u}_{\theta,\varphi} \cdot \boldsymbol{d}_B > \cos\alpha_B, \tag{4}$$

which, in turn, can be expressed as a condition over φ:

$$\begin{aligned} \cos\varphi &> \frac{\cos\alpha_B - \cos\theta\cos\gamma}{\sin\theta\sin\gamma} \\ \cos\varphi &> \cos\varphi_L \end{aligned}$$

Reporting these conditions into equation 2, we get:

$$\begin{aligned} I &= \int_{\theta=\gamma-\alpha_B}^{\alpha_A} \int_{\varphi=-\varphi_L}^{\varphi_L} \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\varphi \\ &= \int_{\gamma-\alpha_B}^{\alpha_A} 2\varphi_L \sin\theta \, \mathrm{d}\theta \\ &= \int_{\gamma-\alpha_B}^{\alpha_A} 2 \arccos\left[\frac{\cos\alpha_B - \cos\theta\cos\gamma}{\sin\theta\sin\gamma}\right] \sin\theta \, \mathrm{d}\theta \end{aligned}$$