**Horizon-Base Indirect Lighting**

## The ideal companion for your far-field indirect lighting solution.

February 2018 – Benoît “Patapom” Mayaux  
b.mayaux@gmail.com

In the following document, vectors will be written in boldface characters (e.g. **,** ) and scalar values will use regular characters (e.g. , ).  
The spherical coordinates used throughout the paper have an elevation angle aligned with the vertical **Z** axis, and an azimuthal angle lying in the tangent plane measured from axis **X**:

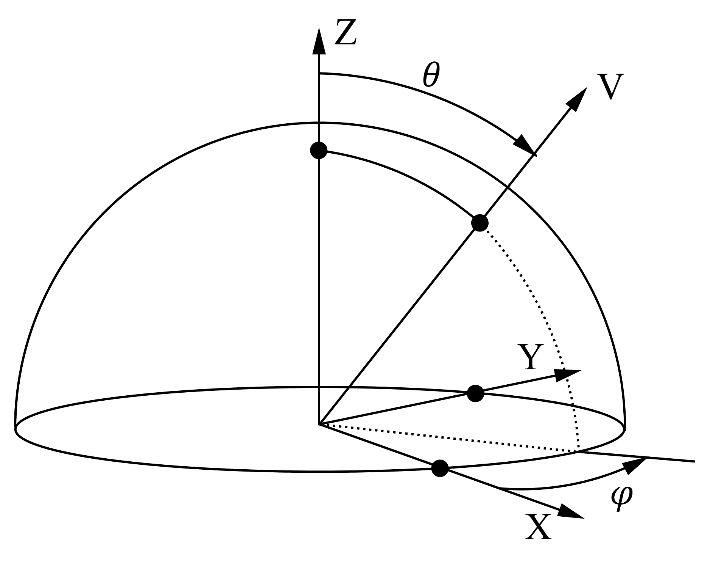


Fig. 1. The frame for spherical coordinates used in this paper.

1. Introduction

The Horizon-Based Ambient Occlusion (HBAO) technique introduced by Bavoil et al. [[1](#REF_1)] proposed to improve the computation of the Ambient Occlusion integral by skipping all the rays that we know for sure would intersect the heightfield/depth buffer and thus wouldn’t contribute to the visibility term:

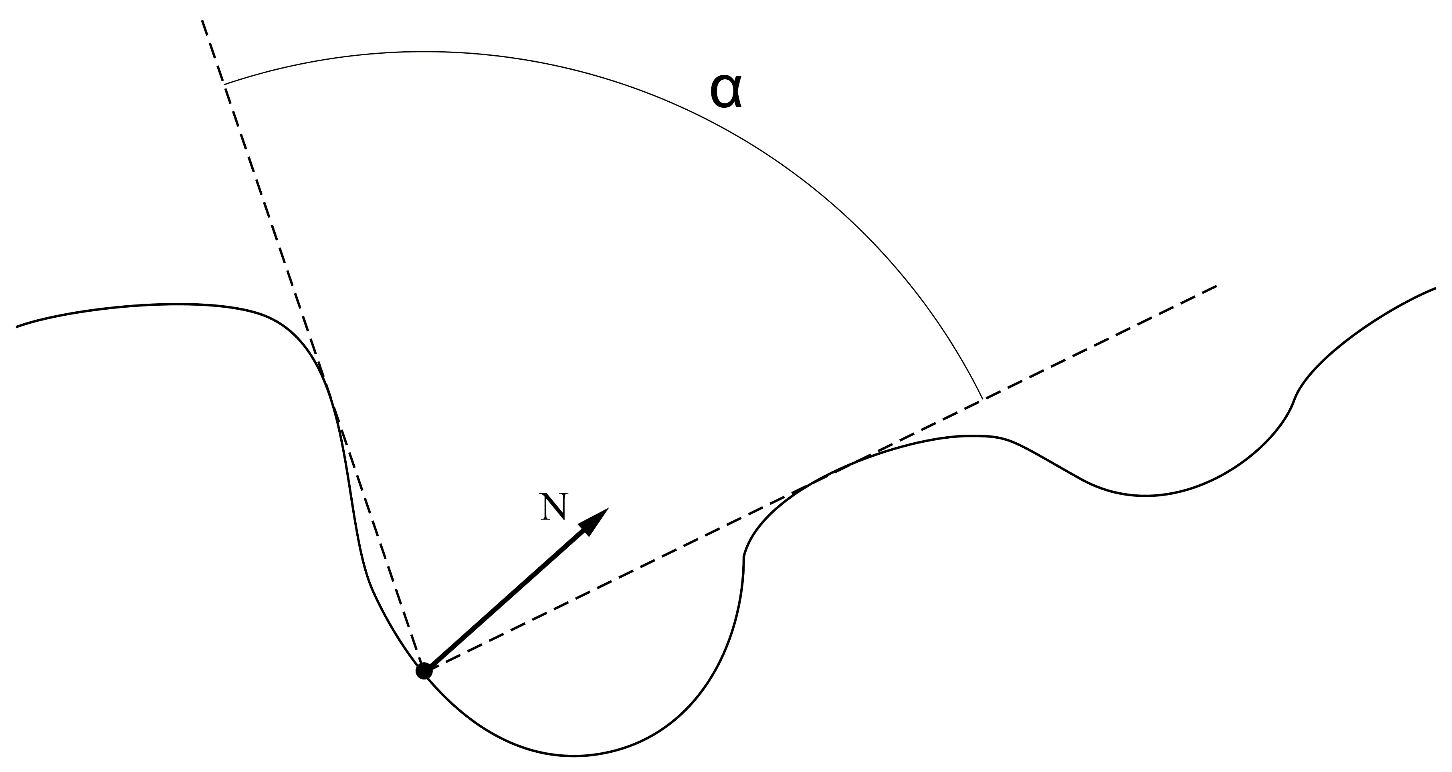


Fig. 2. The HBAO algorithm is exploiting the full visibility of the horizon cone of angle 𝛼:  
 we know all the rays inside the cone escape the surface of the heightfield and are the only ones actually contributing to the visibility term.

The ambient occlusion term is then simplified into:

Where:

* is the location of the pixel for which we are calculating the AO
* is the incoming ray direction
* is the upper hemisphere of directions whose solid angle is 2π
* is the portion of solid angle covered by the vector
* is the visibility term returning 1 if the ray escapes to infinity, 0 if the ray intersects the heightfield
* is the horizon angle in azimuthal direction

This technique allows us to avoid tracing rays for the entire hemisphere, instead we simply need to determine the horizon angle for a particular azimuthal angle by sampling the height field in screen space:

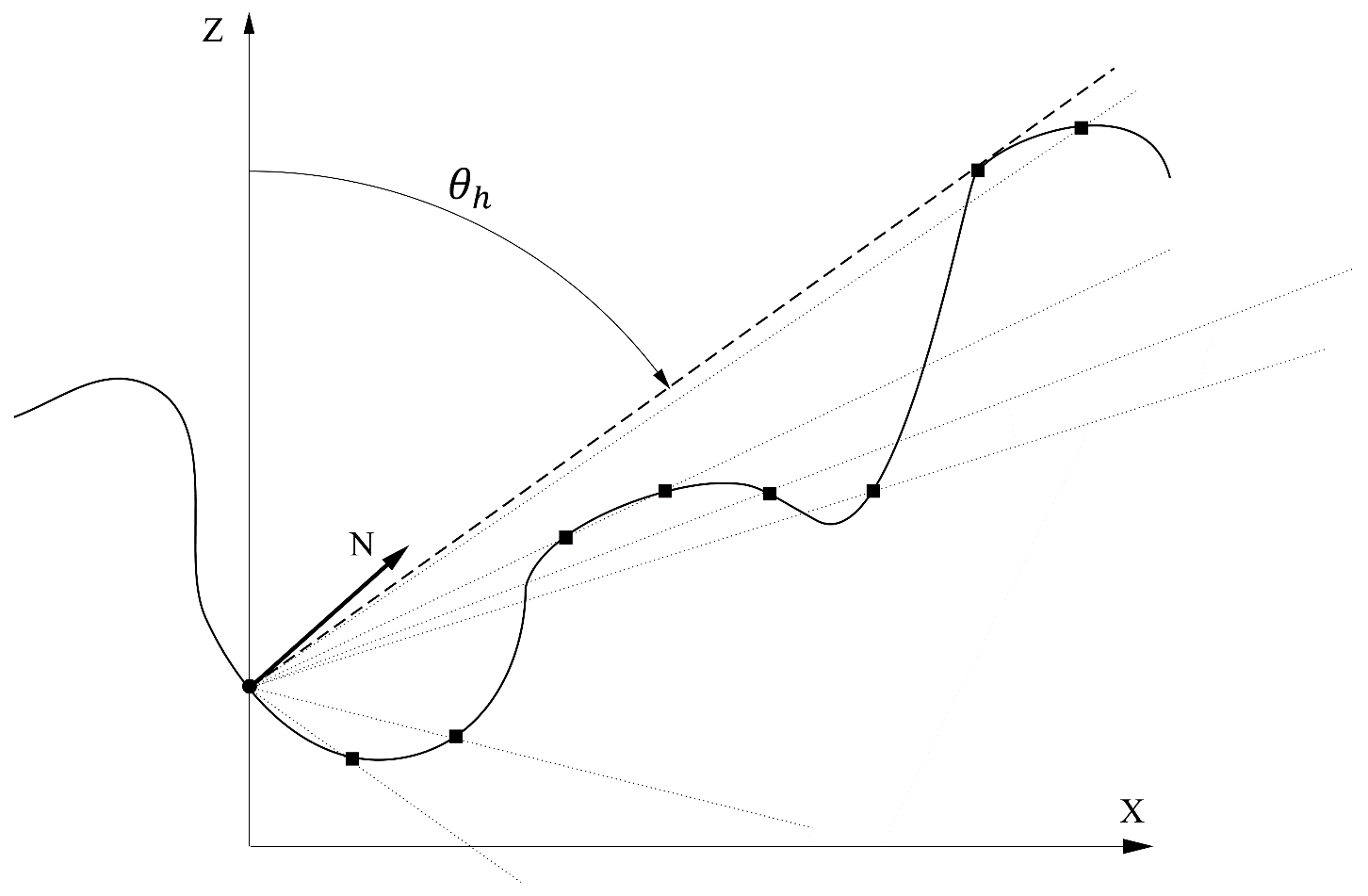


Fig. 3. The heightfield is sampled along the X direction and the horizon is updated along the way by reducing the horizon angle each time.

The portion of AO computed by the inner integral is:

And so, the final expression for the AO becomes:

# Back & Front Sampling

In this paper, to avoid doing the same work twice, we will assume that we always trace the front and back parts of the slices of the tangent-space disk at the same time:

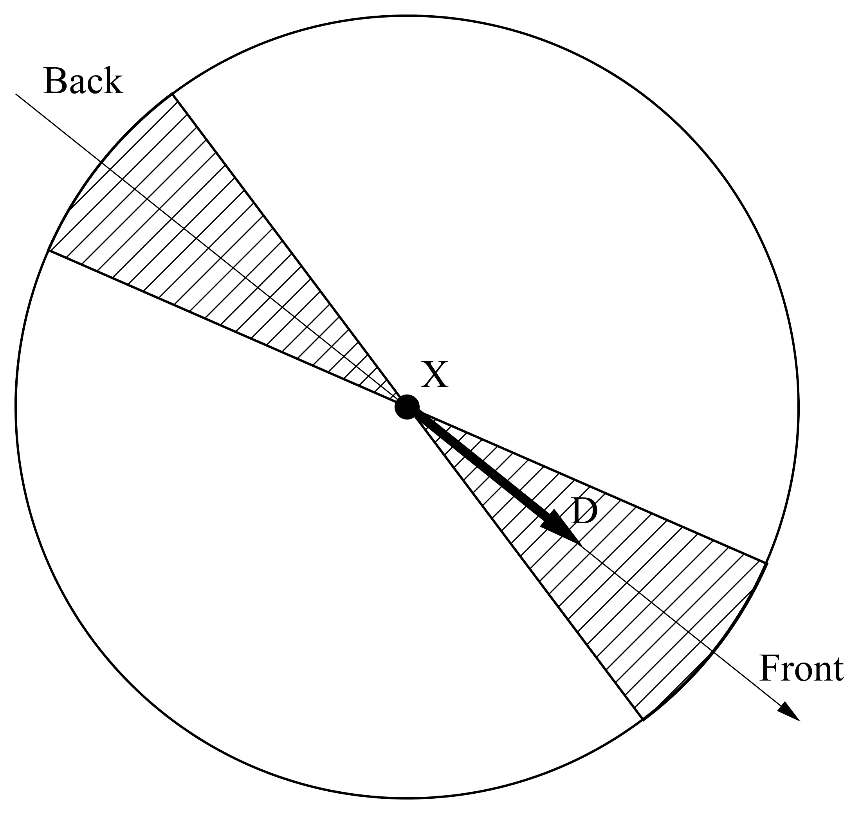


Fig. 4. The tangent-Space disk is sampled both in the forward and backward direction

Estimating the AO in such a simultaneous back & front manner simply becomes:

# Local Camera Space

In order to avoid getting a camera-space normal with a negative z component, we must construct a “local” camera space specifically for our sampling location :

Where:

* is the normalized view vector pointing from our world-space location toward the camera
* is the world-space camera position
* is the world-space camera up vector

From which we obtain:

* the new local camera-space “right” vector
* the new local camera-space “up” vector

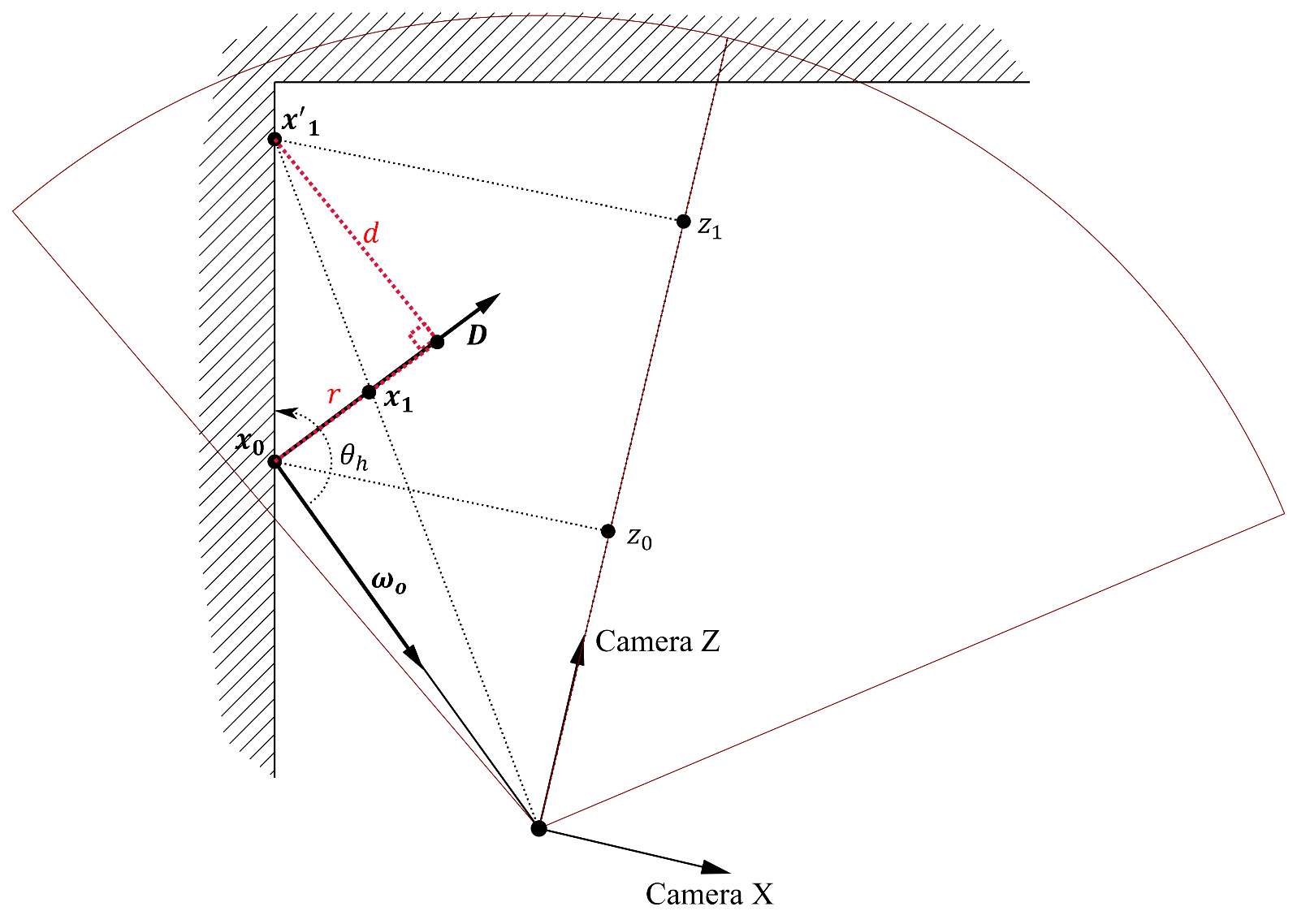


Fig. 5. The important difference between the “global” camera space and the “Local” camera space reconstructed from point and camera view vector .  
Moving to a neighbor location implies some reprojection steps to obtain , and used to compute the new horizon angle . These steps are described in section 2.1.2.

This new local camera space will be used to express the normal, the horizon angles and later, the bent cones.

1. Improvements over HBAO

In this paper, we will propose several techniques to make the most out of the information gathered while performing the HBAO algorithm:

* The first section will discuss how to use the camera-space normal, if it’s available, to make the result more robust.
* The second section will discuss how to obtain the bent-cone from our samples  
  The generated bent-cone buffer will then replace the normal buffer and become a very efficient tool to compute a better scene lighting.
* Finally, the third section will explain how to eventually re-use the indirect diffuse lighting from the previously rendered frame to compute a very important near-field indirect lighting term to improve the scene lighting.2.

# Using the Normal

We can first improve a little upon the quality of the HBAO algorithm by using the normal to initialize the horizon angles.  
The camera-space normal vector is often available from the G-Buffer produced by most deferred renderers.

The gray areas of the tangent-space disk from figure 4 are approximated by a “slice” rotating about the axis of the local camera-space described earlier:

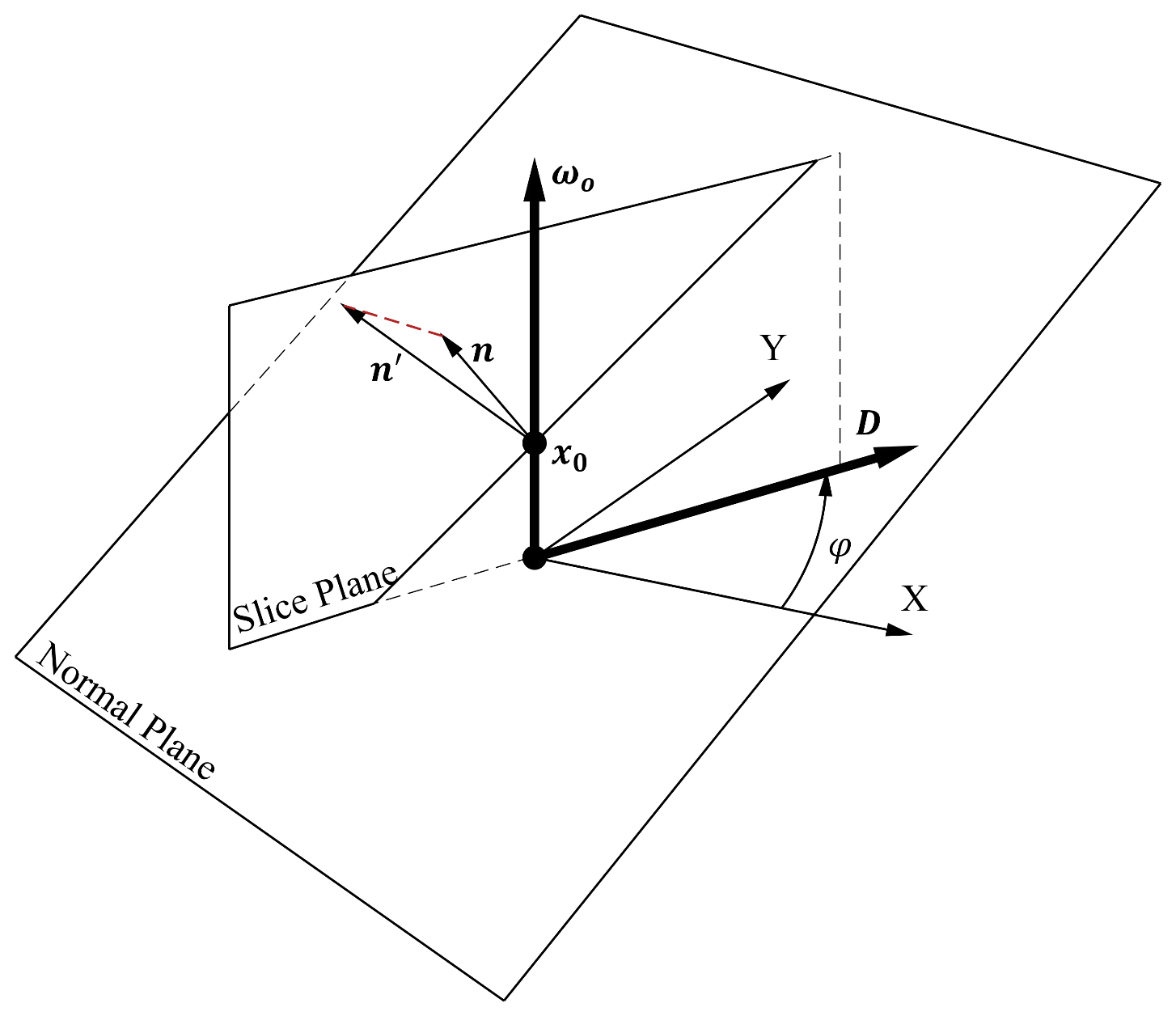


Fig. 5. a camera-space slice rotating by an angle about the axis pointing toward the camera**.**  
We point out the important fact that the normal is not necessarily lying in the plane of the slice and will need to be projected onto the slice, yielding the  vector.

## Initial Horizon Angles

We initialize the horizon angles for the slice by computing the projection of the vector onto the normal plane by following the axis:

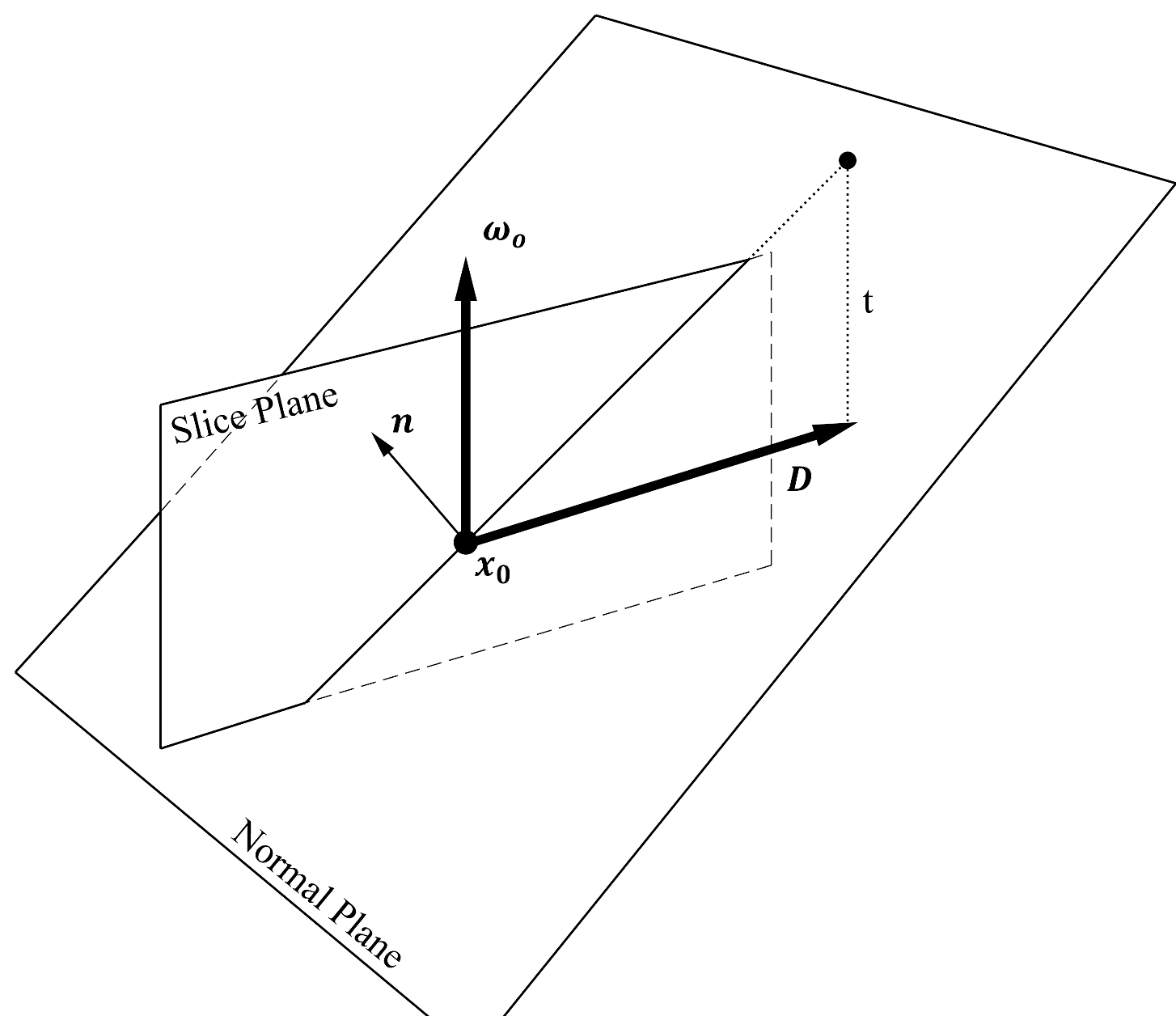


Fig. 6. Projection of the unit vector onto the normal plane following the axis to find the initial “minimal” horizon angles for the slice.

Where:

* is the slice’s azimuthal direction expressed in local camera-space
* is some azimuthal angle in the tangent plane of the local camera-space
* is the view direction expressed in local camera-space
* is the normal expressed in camera-space
* is the intersection distance with the normal plane
* and are respectively the initial front and back horizon angles

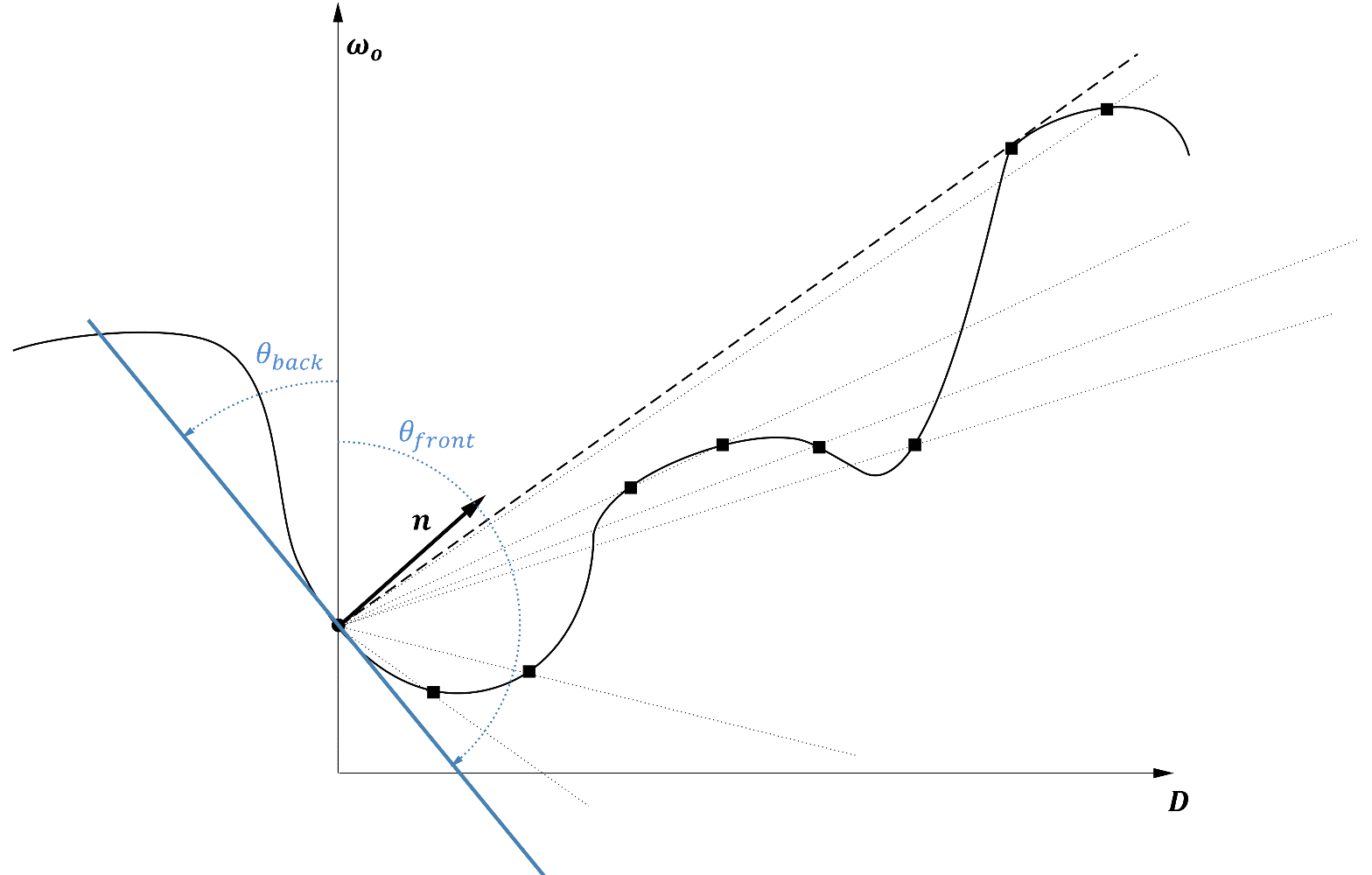


Fig. 7. Using the normal to determine and as our initial horizon values for each slice of the camera-space sampling disk.

**TODO: Show ON/OFF image**

## Updating the Horizon Angles

In figure 5 we saw that when we move from our central location to the neighbor location by following the world-space vector for a small distance we get**:**

If we sample the depth-buffer at the screen-space location corresponding to this new location , it will give us depth from which we obtain the new world-space location that we can finally express back into the local camera space:

From which we can finally obtain the cosine of the horizon angles that we will use all along this paper:

Where:

* is the measured elevation angle for the neighbor sample at
* is the horizon angle we keep updating as we move along

# Bent Cones

We follow the methodology of the GTAO computation described by Jimenez et al. [[3](#REF_3)] but we will not compute the ambient occlusion, instead we are interested in what I call a “bent cone” which is a combination of a bent normal used as the central axis of a cone whose aperture depends on the ambient occlusion:

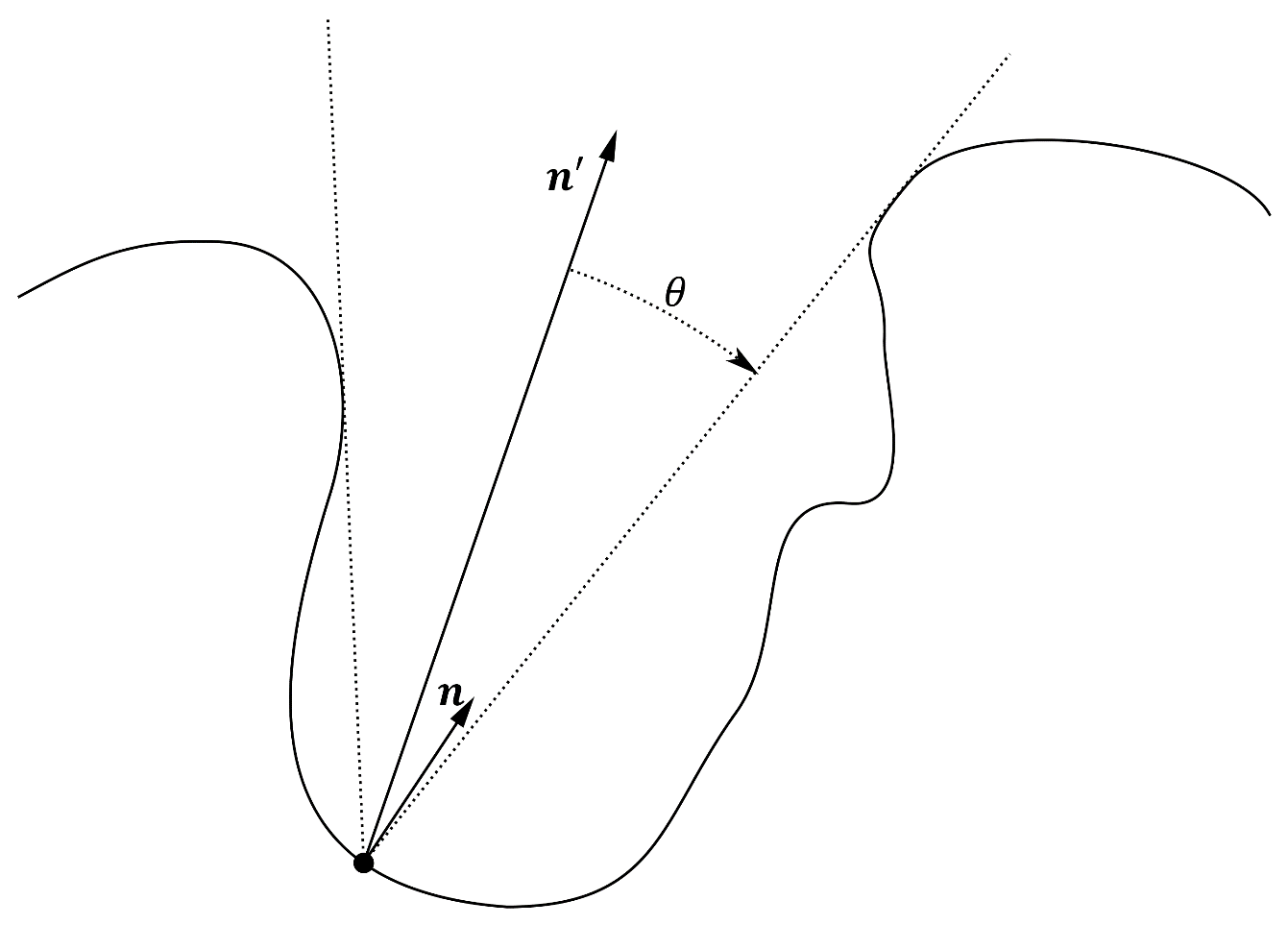


Fig. 9. The “Bent-Cone”, a bent normal with an angle.

## Bent Normal

In order to first get the bent normal, we need to compute the average direction of a vector weighted by the cosine of the angle with the normal (since directions at grazing angles don’t contribute much to the bending) and unobscured by the heightfield:

Where:

* is the resulting average “bent normal” (in camera space)
* N is the amount of hemispherical samples
* is the direction of the ith sample (in camera space)
* is the direction of the normal to the surface (in camera space)

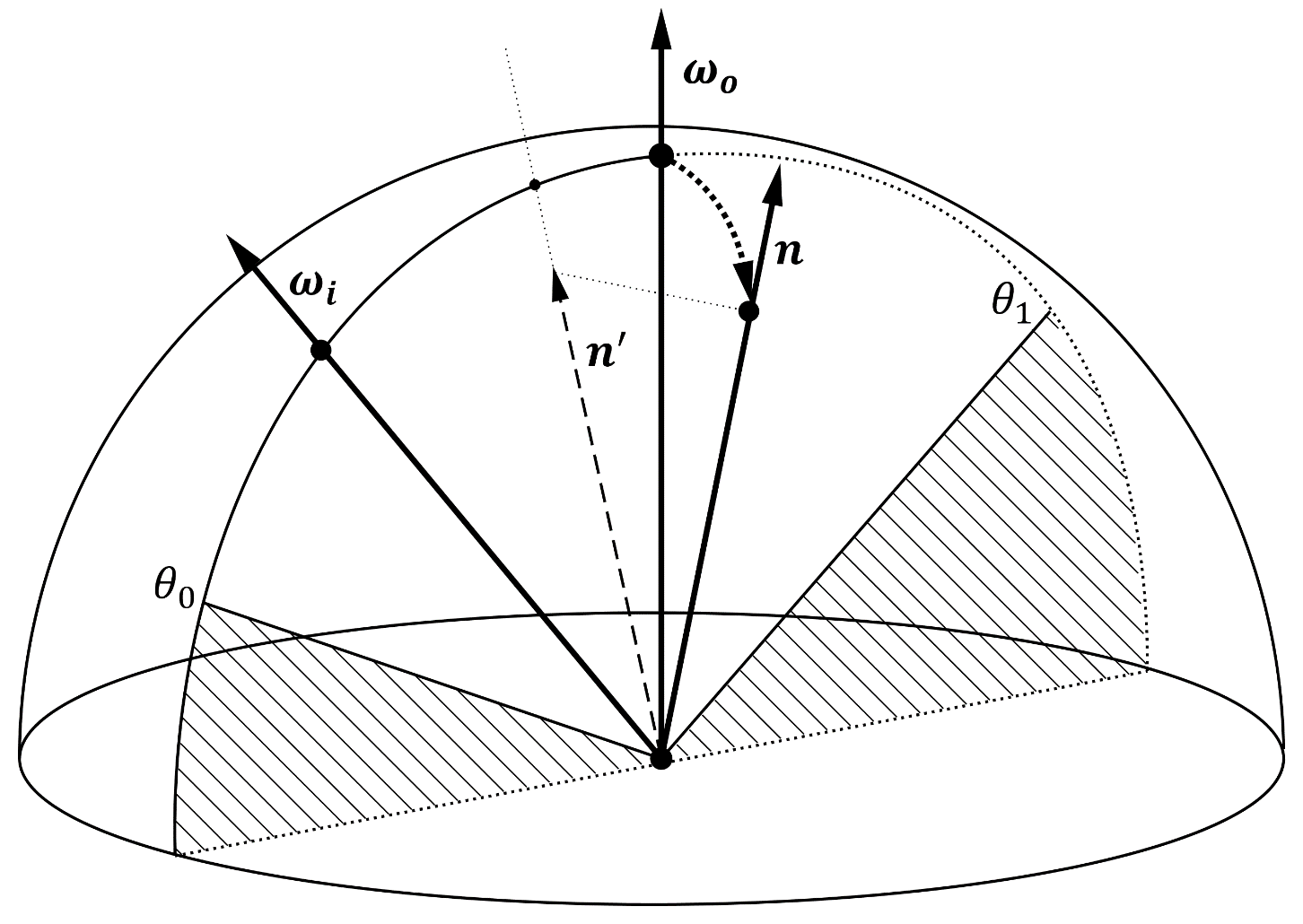


Fig. 10. Integration of the vector in the slice between angles and weighted by the dot product with normal vector . Notice that the normal vector is not necessarily lying in the slice’s plane.

We can compute the exact integral for equation 4 in our 2D “slice space” by writing:

Equation 5 can be expanded into:

Similarly, equation 6 gives:

We can then rebuild a camera-space normal:

Where:

* is the amount of computed slices
* and are the and values for the slice
* and are the camera-space slice vectors described by equations group (1)

**TODO: Show world space result bent-normal**

**TODO: Show ON/OFF image**

Nouvelle formulation de la bent normal où on vire l’influence du dot product avec la normale :

5 becomes:

6 becomes:

Le rendu de la bent normal est évidemment plus smooth mais dans le ground truth fitting, je trouve qu’on s’éloigne de la ground truth, ça smooth trop les contours… A voir si c’est fixé avec un meilleur calcul de l’ouverture ???

## Cone Aperture and Ambient Occlusion

We need the cone’s aperture angle and, overall, the solid angle covered by the cone, to be significant when we will use it to sample the distant environment and perform direct lighting.

Sampling the distant (*i.e.* far-field) environment is computed with:

Where:

* is the irradiance at for surface normal from the far-field environment
* is the incoming radiance at from direction
* is the set of all directions covering the upper hemisphere
* is the solid angle covered by the surface perceived along direction
* is the visibility term we saw in section 1

I wrote in [[4](#REF_4)] that equation 7 is often simplified into:

Where is the unoccluded irradiance from a diffuse cube map or some SH representation.

Still in [[4](#REF_4)], it is noted that this simplification suffers from a loss that must be compensated by applying a factor given by so that finally:

With:

We see from equation 8 that we need to compute the AO term:

From figure 10 we see it is easy to compute the visibility of a single slice using our 2 horizon angles by integrating:

And finally:

Where is the amount of computed slices.

To finalize our bent-cone, we are interested in another formulation for the AO, which would be the solid angle covered by a cone:

The true integral of the visibility term in equation 8 actually yields the solid angle:

From this we can deduce that:

And finally, the aperture angle of our bent-cone is given by:

**TODO: Show cone aperture result**

**TODO: Show ON/OFF image**

## Depth Filtering

Sqdqd

**TODO: Show filtering result**

**TODO: Show ON/OFF image**

# Indirect Lighting

The classical lighting equation to compute the outgoing radiance from a pixel at in viewing direction is essentially given by:

Where:

* is the incoming radiance at from direction
* is the surface’s BRDF
* is the surface normal
* is the set of all directions covering the upper hemisphere
* is the solid angle covered by the surface perceived along direction

Focusing only on diffuse Lambertian reflection, we can rewrite eq. (1) as:

Where:

* is the irradiance arriving at surface location and normal .
* represents the diffuse BRDF for a surface with albedo .   
  The division by is here to guarantee energy conservation since .  
  NOTE: although a RGB quantity will be noted simply in the rest of the document

**TODO: Figure with direct, far and near**

We saw in equation 7 how to sample the far-field environment but the full irradiance is given by:

Where:

* is the direct diffuse lighting irradiance computed using all the light sources of the scene
* represents the missing near-field environment term, the part of the energy that bounced off the close environment and perceived indirectly by our sampling point.  
  We notice this term is using the opposite of the visibility function and is thus the exact complement of the far-field environment term since it only gathers radiance from the occluded environment

Contrary to [[4](#REF_4)] where the near-field bounces were approximated via manually fitted ad-hoc terms, in this paper we will see how to compute the actual near-field term.

## Recursive Irradiance Bounces

We begin by noticing that the incoming radiance term from equation 14 must be a diffuse reflection of the light on the neighbor environment:

Where and are the location and normal of the neighbor environment perceived in direction .

Thus, we can rewrite equation 14 as:

We could make the same assumption as in [[4](#REF_4)] and assume the neighbor reflectance to be the same as our current reflectance at but we will see that we are going to sample a buffer where we stored the the diffuse *radiance* , so the and terms are already combined together and this trick is not possible anymore (nor is it wanted in our case).  
The only possible optimization would be to avoid sampling the neighbor radiance and use our central radiance for all neighbor samples instead, but the quality would suffer.

So, interestingly, we see that all in all, estimating the near-field irradiance simply involves sampling the radiance from neighbor surfaces.

Assuming the diffuse radiance values computed during the *last frame N-1* are available then we could solve the lighting diffuse equation 13 by computing:

With:

* is the radiance computed at frame N that will be stored in the diffuse radiance buffer that will, in turn, be re-used at frame N+1
* is the direct lighting irradiance computed at frame N
* is the far-field radiance computed at frame N
* is the radiance computed at frame N-1 and stored at neighbor location

Computing radiance in this fashion would theoretically give us an infinite amount of light bounces.

## Solving the indirect lighting integral

All we are left to solve is the following integral:

Coming back to the context of our integration slice where we are slowly computing the horizon angles for our front and back directions:

**TODO: Figure with horizon rising, irradiance gathered**

In this figure, we can easily see that the only irradiance we perceived is the one from when the horizon jumps up a little: only the areas represented in red contribute to our sampling of the radiance.

So, for a small jump of the horizon angle from to , we have the following integral to solve:

Assuming is constant for the entire interval then becomes:

With:

* is our rotating incoming vector for the current slice
* is the slice’s direction vector from equation 2
* is the normal vector projected onto the slice
* is the previous (i.e. lower) horizon angle
* is the new (i.e. raised) horizon angle
* is the neighbor radiance sampled at the neighbor location where we are currently updating the horizon

We solve the integral:

1. Integration with the lighting pipeline

Focusing only on diffuse Lambertian reflection, we can rewrite eq. (1) as:

# Reprojecting Last Frame Radiance

Focusing only on diffuse Lambertian reflection, we can rewrite eq. (1) as:

# Using the bent-cone for direct illumination

Focusing only on diffuse Lambertian reflection, we can rewrite eq. (1) as:

# Using the bent-cone for far-field indirect illumination

Focusing only on diffuse Lambertian reflection, we can rewrite eq. (1) as:

# Putting it all together

Focusing only on diffuse Lambertian reflection, we can rewrite eq. (1) as:

Focusing only on diffuse Lambertian reflection, we can rewrite eq. (1) as:

**TODO: Integrate F0!!!!**

**TODO: Show ON/OFF image**

1. Performance

Focusing only on diffuse Lambertian reflection, we can rewrite eq. (1) as:

1. Acknowledgments

Special thanks to Eric Arnebäck for proof reading this paper, Benjamin Lalisse for his clever remarks, Martin Gérard for his precious help with my math, [Geoffrey Rosin](https://www.artstation.com/kikette) for his amazing textures, and Sandra for moral support 😊.

1. References

*[1] Bavoil, L. and Sainz M. 2008. “Image-Space Horizon-Based Ambient Occlusion”*

*[2] Bavoil, L. and Jansen, J. 2013. “Particle Shadows & Cache-Efficient Post-Processing”*

*[3] Jimenez, J. Wu, X-C. Pesce, A. and Jarabo, A. 2016. “Practical Realtime Strategies for Accurate Indirect Occlusion”*

*[4] Mayaux, B. 2018, “*[*Improved Ambient Occlusion*](https://drive.google.com/file/d/1SyagcEVplIm2KkRD3WQYSO9O0Iyi1hfy)*”*

*[] Mayaux, B. “Spherical Harmonics Irradiance Estimate for a Cone”*

*[] Cook, J. D. “*[*Accurately computing running variance*](https://www.johndcook.com/blog/standard_deviation/)*”*