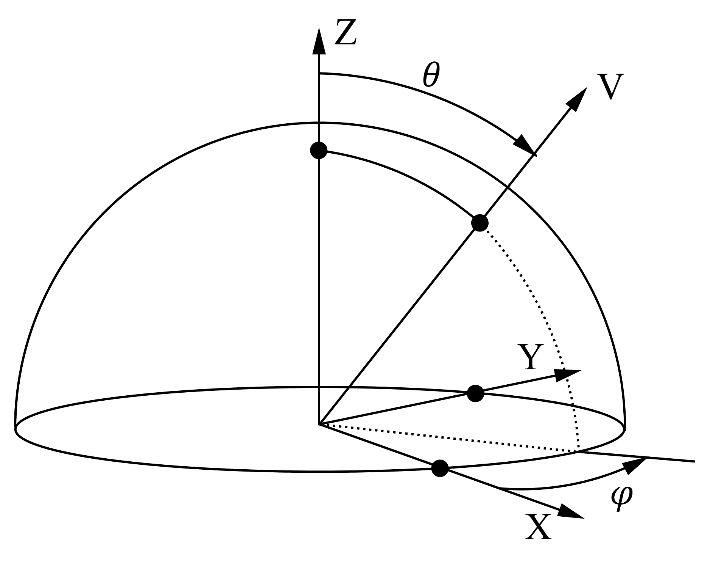
**Horizon-Base Indirect Lighting**

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

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

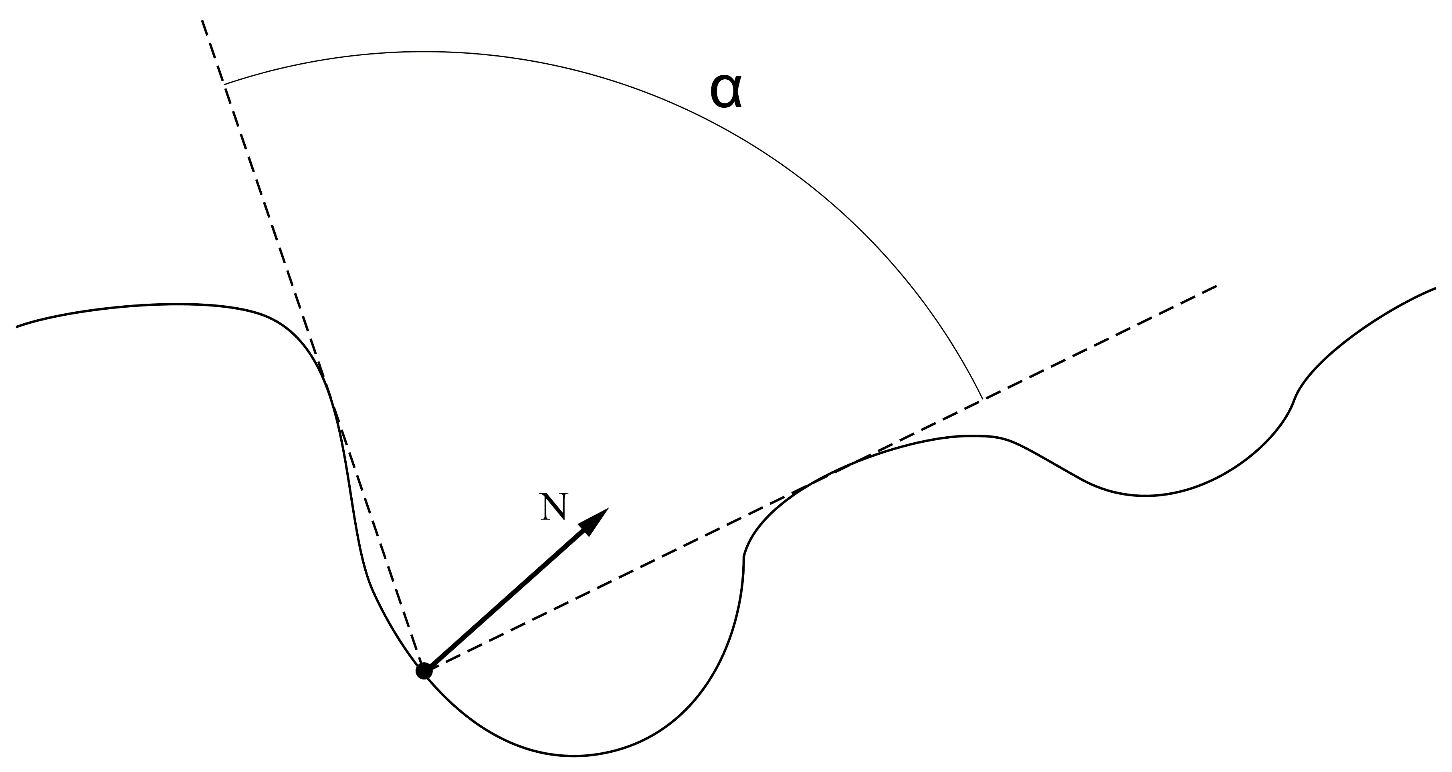
In the following document, vectors will be written in boldface characters (e.g. **,** ) and scalar values will use regular characters (e.g. , ).  
The spherical coordinates used throughout the paper have an elevation angle aligned with the vertical **Z** axis, and an azimuthal angle lying in the tangent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



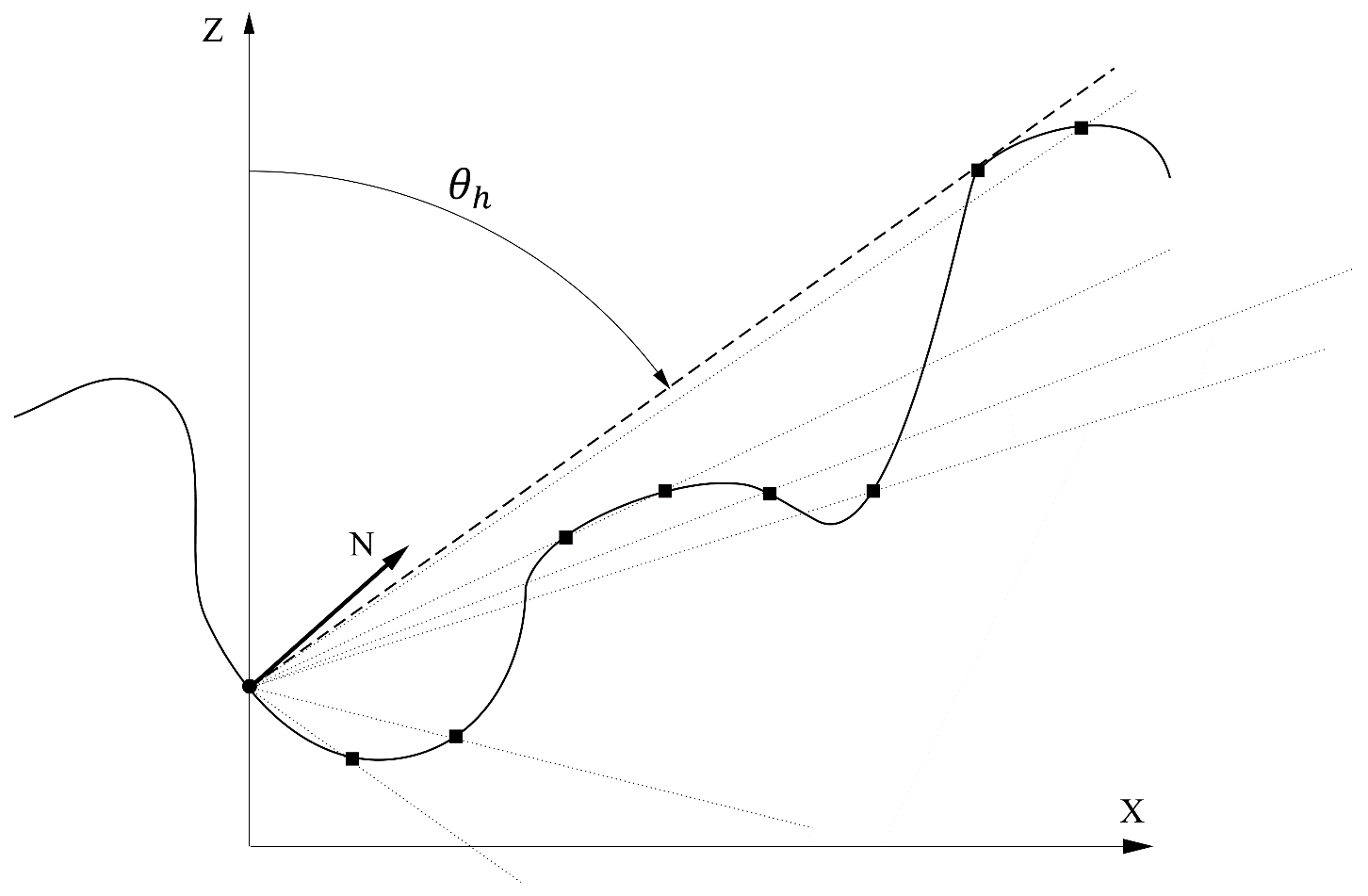
**Fig. 2.** The HBAO algorithm is exploiting the full visibility of the horizon cone of angle 𝛼:  
 we know all the rays inside the cone escape the surface of the heightfield and are the only ones actually contributing to the visibility term.

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:



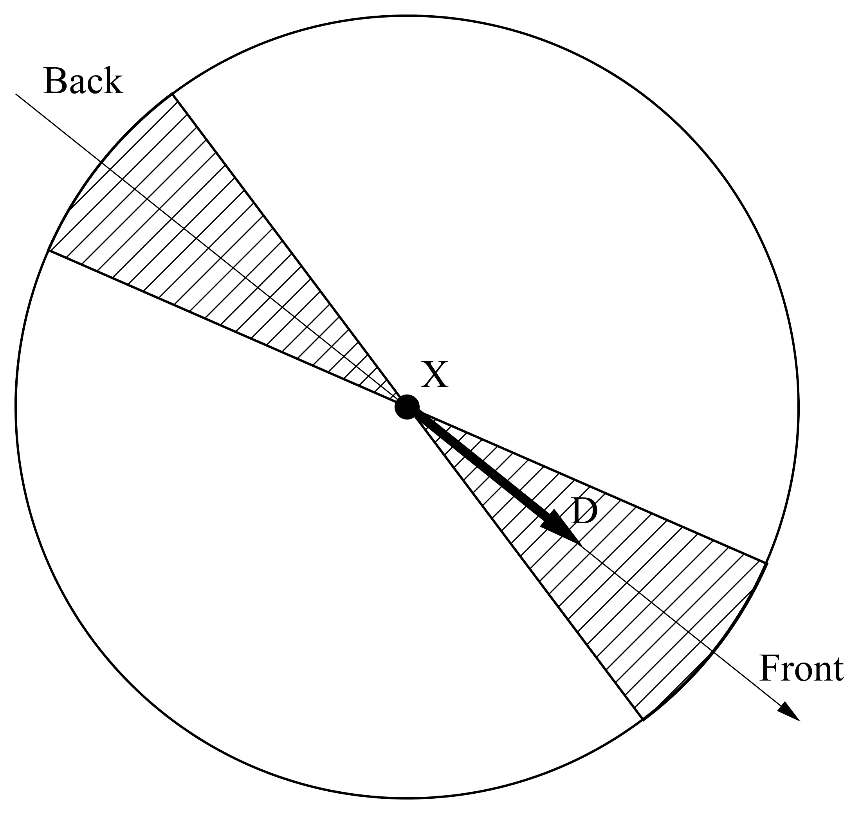
**Fig. 3.** The heightfield is sampled along the X direction and the horizon is updated along the way by reducing the horizon angle each time.

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

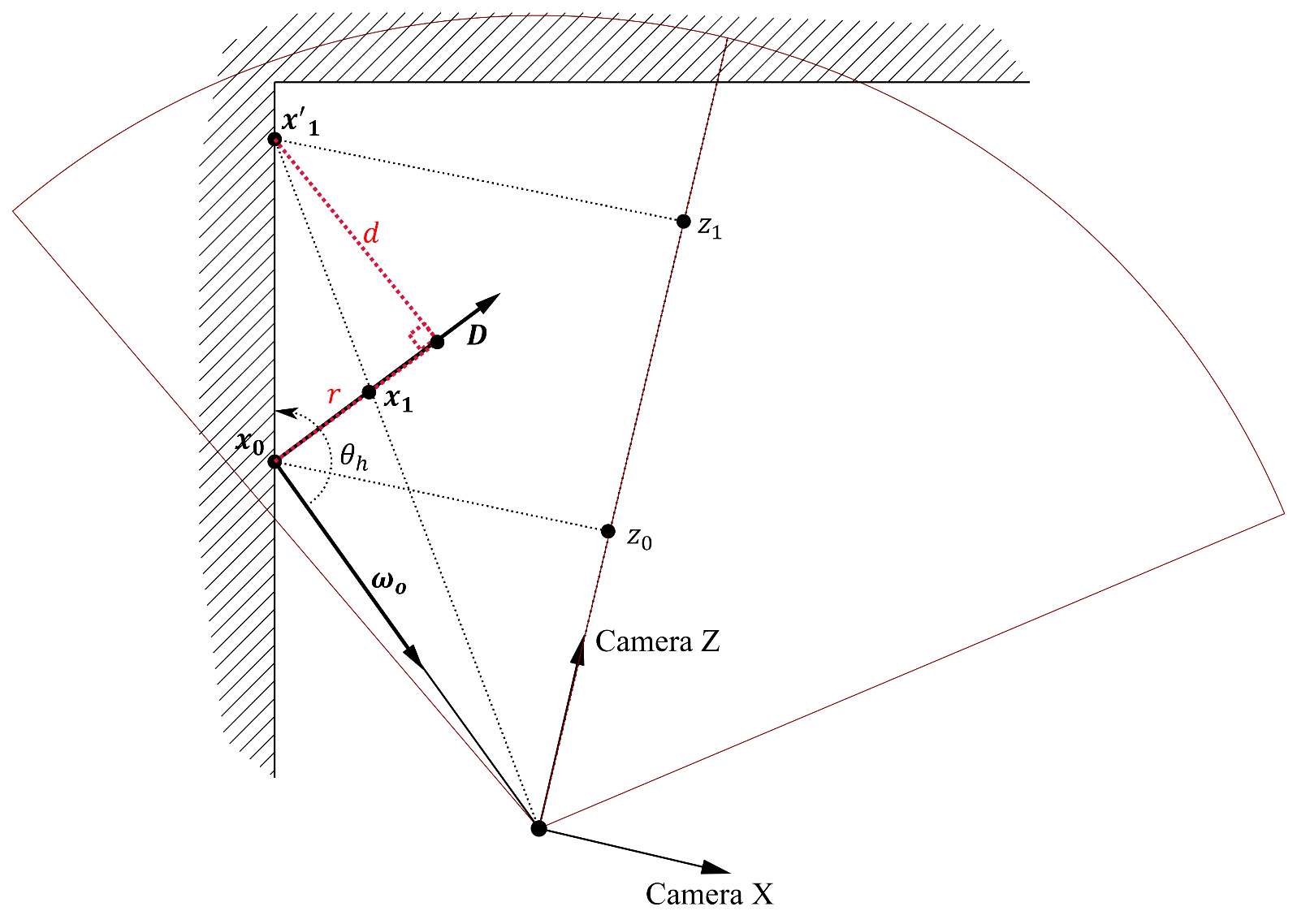
In order to avoid getting a camera-space normal with a negative z component, we must construct a “local” camera space specifically for our sampling location :

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

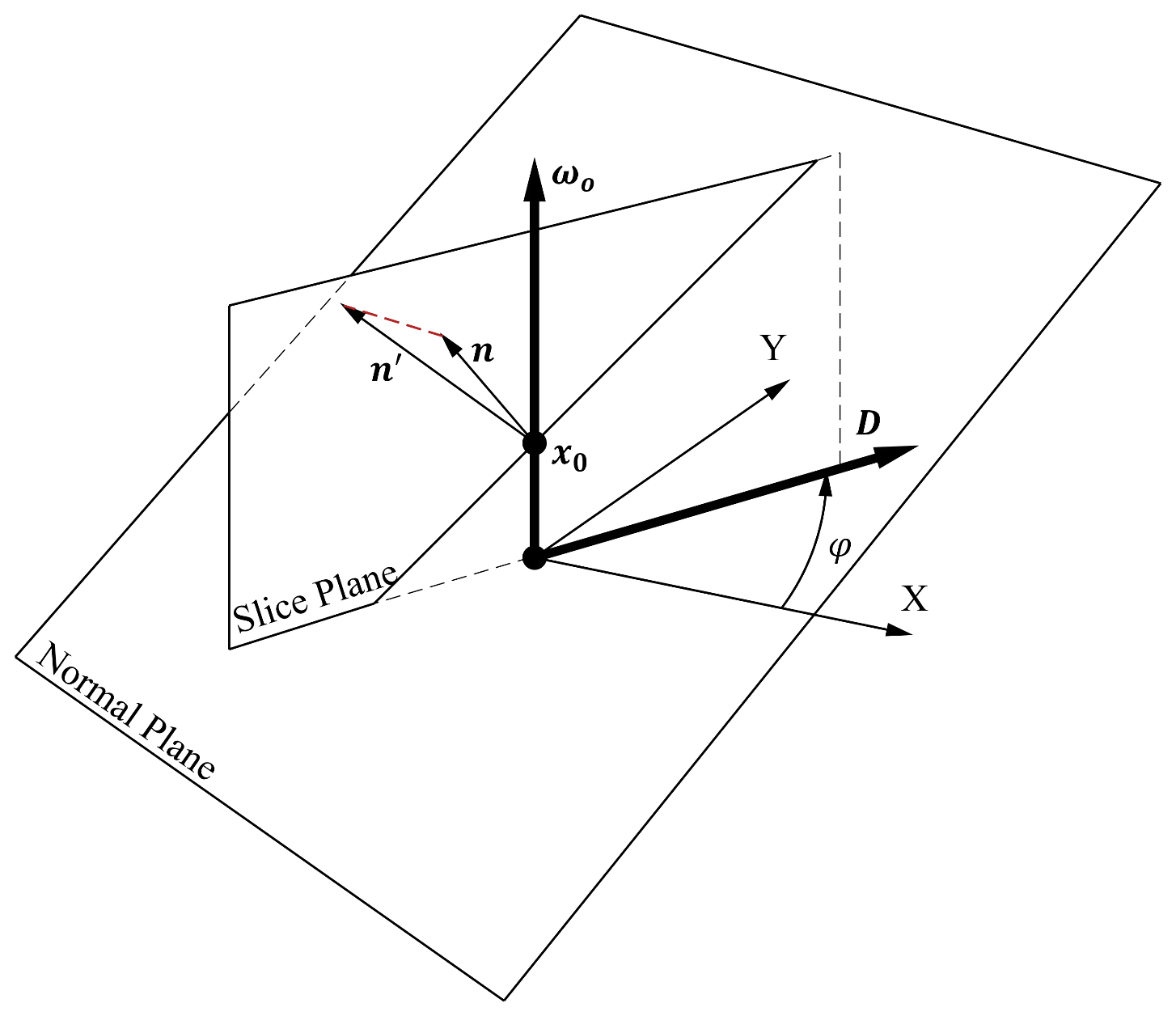
In this paper, we will propose several techniques to make the most out of the information gathered while performing the HBAO algorithm:

* 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.

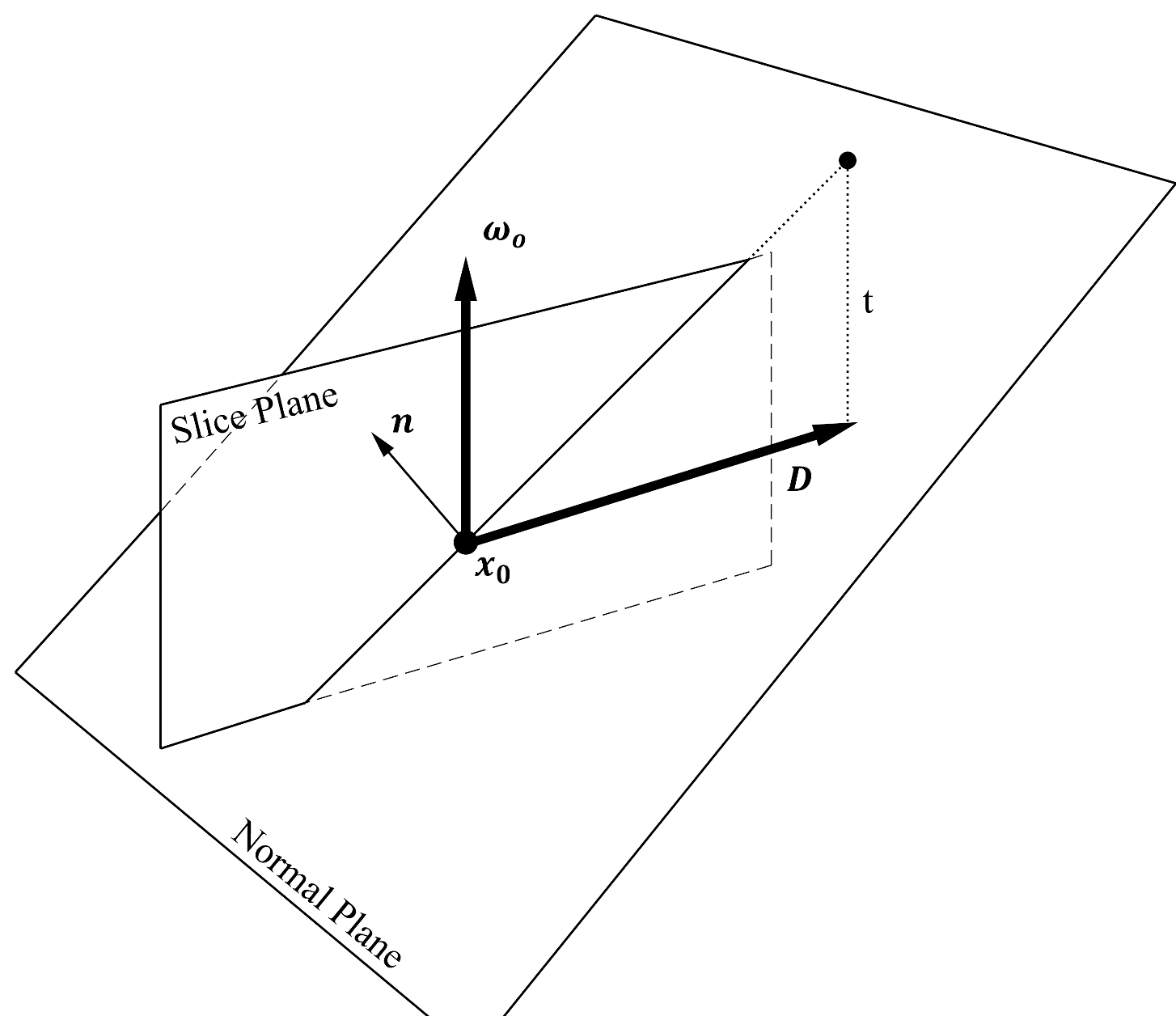
The gray areas of the tangent-space disk from figure 4 are approximated by a “slice” rotating about the axis of the local camera-space described earlier:



**Fig. 6.** 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

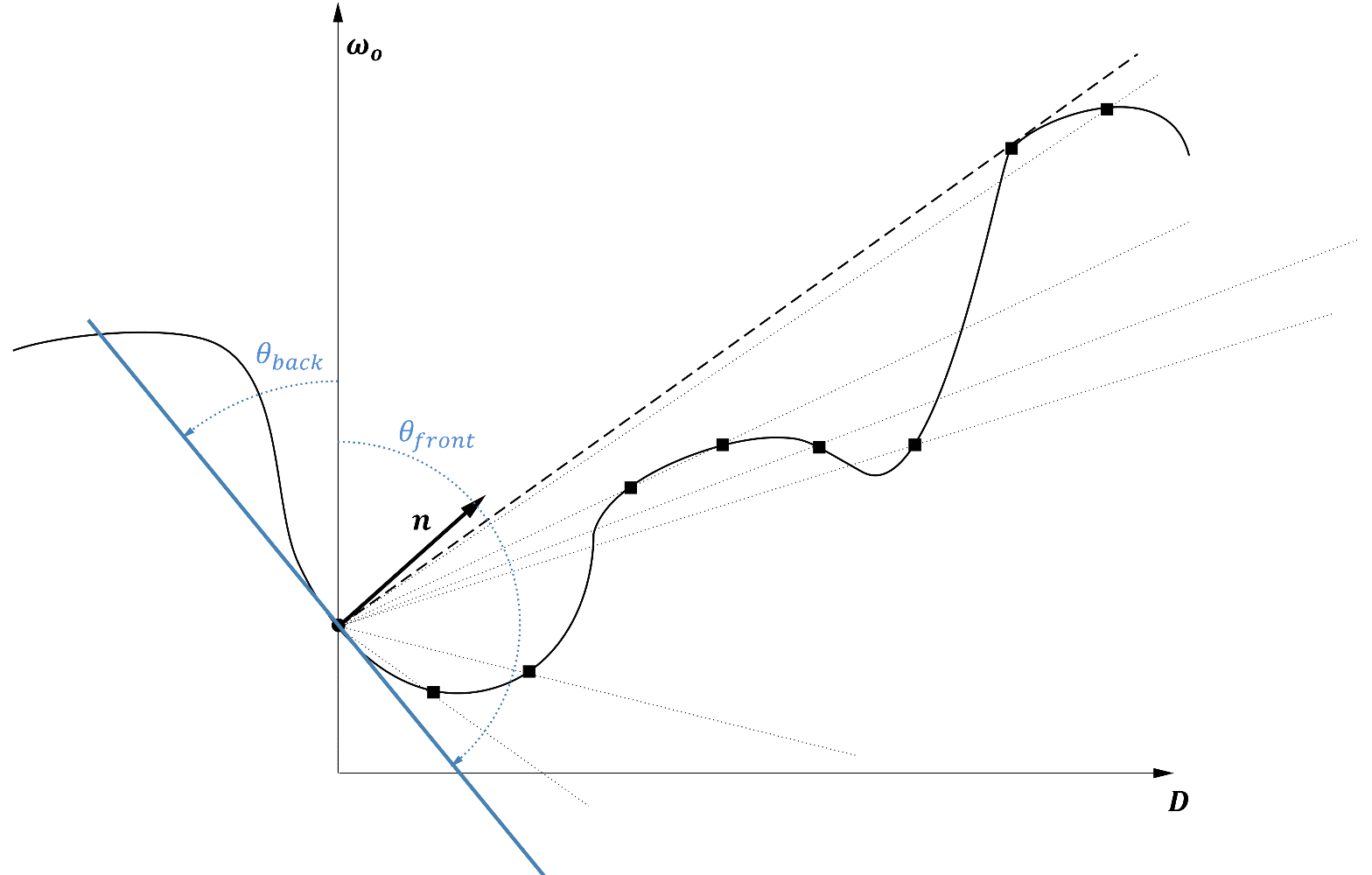
We initialize the horizon angles for the slice by computing the projection of the vector onto the normal plane by following the axis:



**Fig. 7.** 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. 8.** 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:

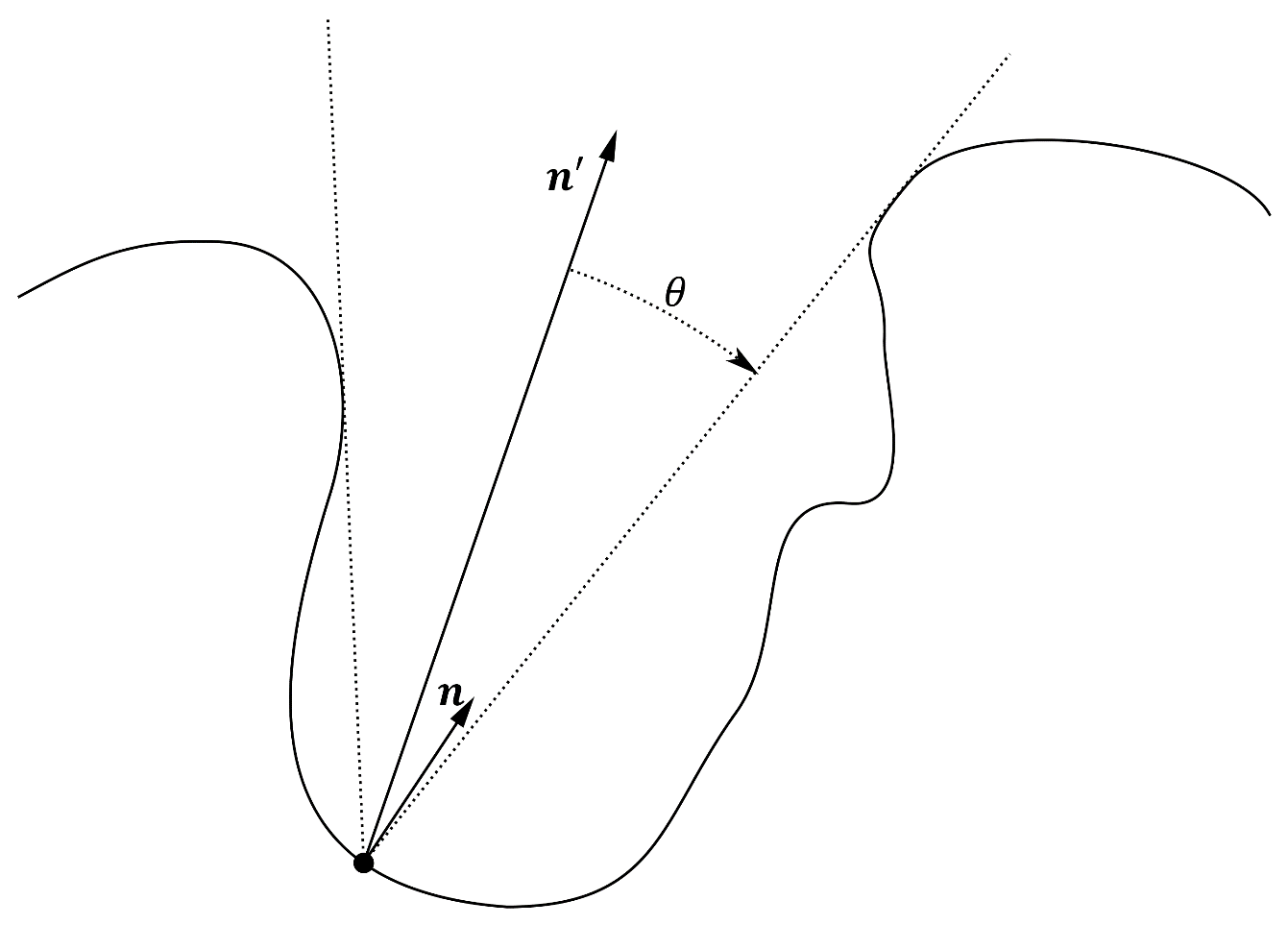
From which we can finally obtain the cosine of the horizon angles that we will use all along this paper:

Where:

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

# Bent Cones

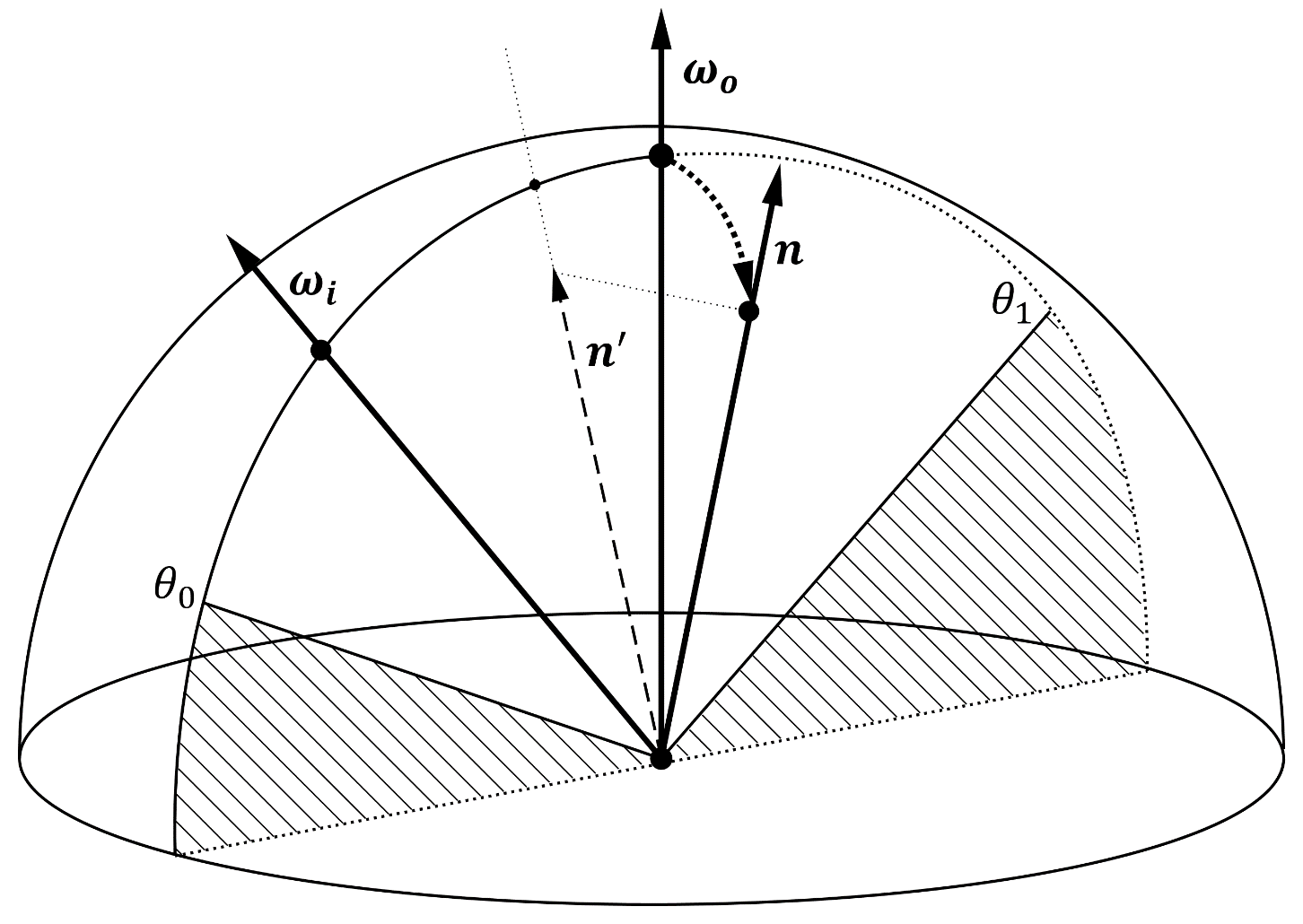
We follow the methodology of the GTAO computation described by Jimenez et al. [[3](#REF_3)] to compute the ambient occlusion but we are also interested in computing what I call a “bent cone”, which is a combination of a bent normal used as the central axis of a cone whose aperture depends on the ambient 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 unoccluded by the environment:



**Fig. 10.** Integration of the vector between angles and , restricted to a single slice. Notice that the normal vector is not necessarily lying in the slice’s plane.

We can get an approximation of the integral for equation 4 by computing a series of integrals restricted to 2D slices rotating about our view axis as shown in the figure above:

Where:

* S is the amount of slices
* is the “back” horizon angle for the slice.
* is the “front” horizon angle for the slice.
* is a vector direction restricted to the slice, with the angle .

Focusing on the inner integral, we rewrite its expression in slice-space as:

Where:

* is the camera-space vector expressed in slice-space
* is the resulting normal vector for the slice

And we get:

From which can finally rebuild a unit camera-space normal:

Where:

* 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**

## Cone Aperture and Ambient Occlusion

We need 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 by:

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 should 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 integrate the visibility of a single slice using our 2 horizon angles:

And finally, AO is given by:

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 with demi-aperture angle :

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:

We can also conclude that the cosine of the aperture angle of our bent-cone is the *geometric equivalent* of the classical AO formulation, but we will prefer it to the AO term since it lends itself very well to a comparison to the dot product generally found in lighting equations.

The effect of accounting for the cone aperture angle is most important, as can be seen in the image below:

**TODO: Show cone aperture result**

**TODO: Show ON/OFF image**

## AO Variance

In order to have a little more variation in our AO values, we also compute the variance for its value for each slice.  
We assume that each slice provides its own little AO value:

We use the running variance algorithm described in [[6](#REF_6)] to accumulate variance as we go along with each new slice:

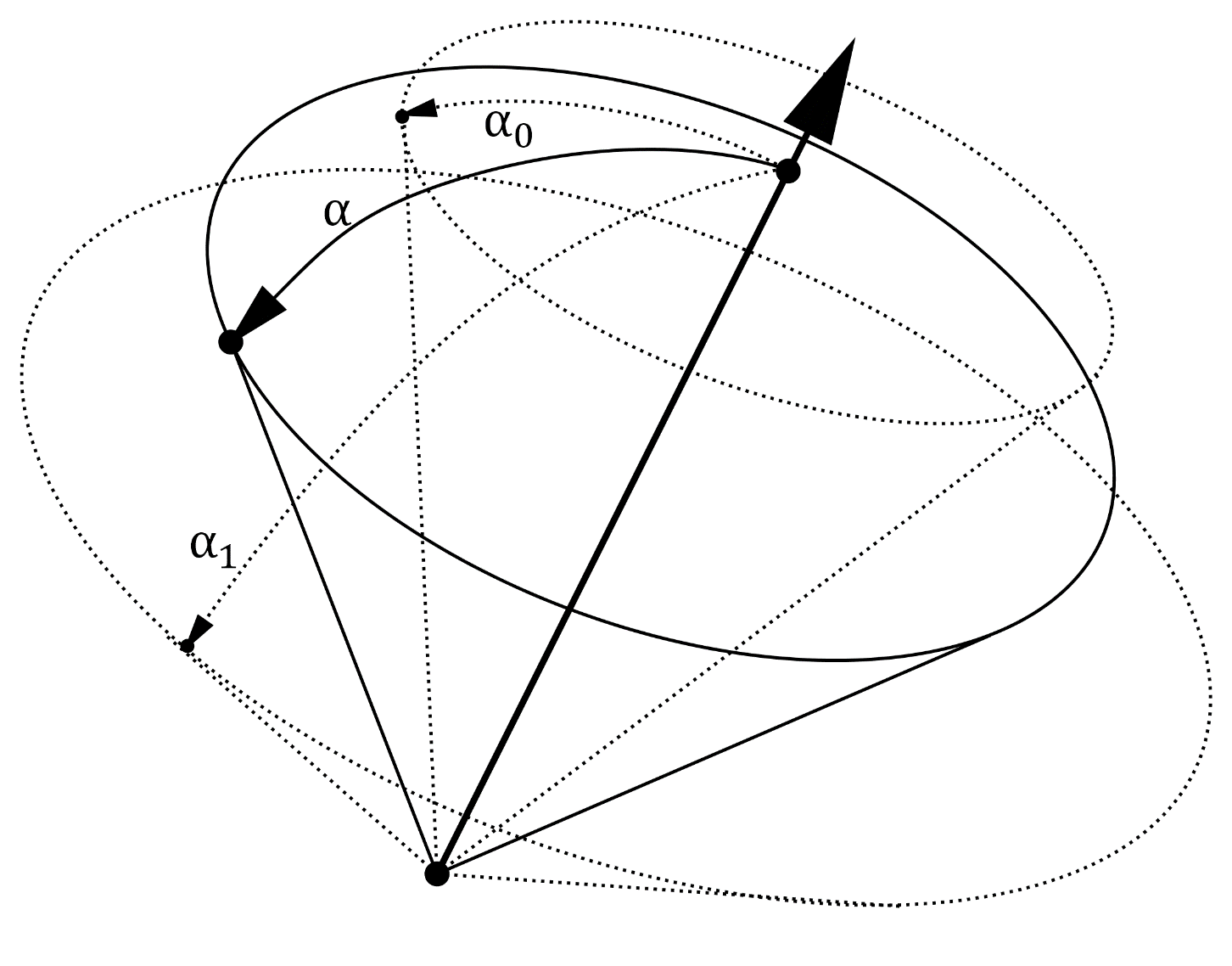
Where:

* is the average of the AO value for slice
* is the variance of the AO value for slice
* is the total amount of computed slices
* is the resulting variance and the standard deviation

We saw from equation 12 that AO and cone aperture were equivalent notations so we could also assume the standard deviation in AO is also giving use a standard deviation in cone angle:

Where is a custom bias that can be applied to increase or decrease the variation of the cone angle.

The resulting angles could be used as a classical “hotspot” and “falloff” angle of regular spot lights at the cost of 1 more storage slot in the resulting bent-cone buffer:



**Fig. 11.** using the standard deviation of AO to compute “hotspot” and “falloff” angles.

The computation of the direct light attenuation could then be augmented with a new attenuation term depending on whether the light direction is perceived by the surface or not:

Where maybe a classical smoothstep function.

**TODO: Show ON/OFF image**

## Depth Filtering

**#TODO:** Not sure I have a definite algorithm for that, keep that for later…

**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:

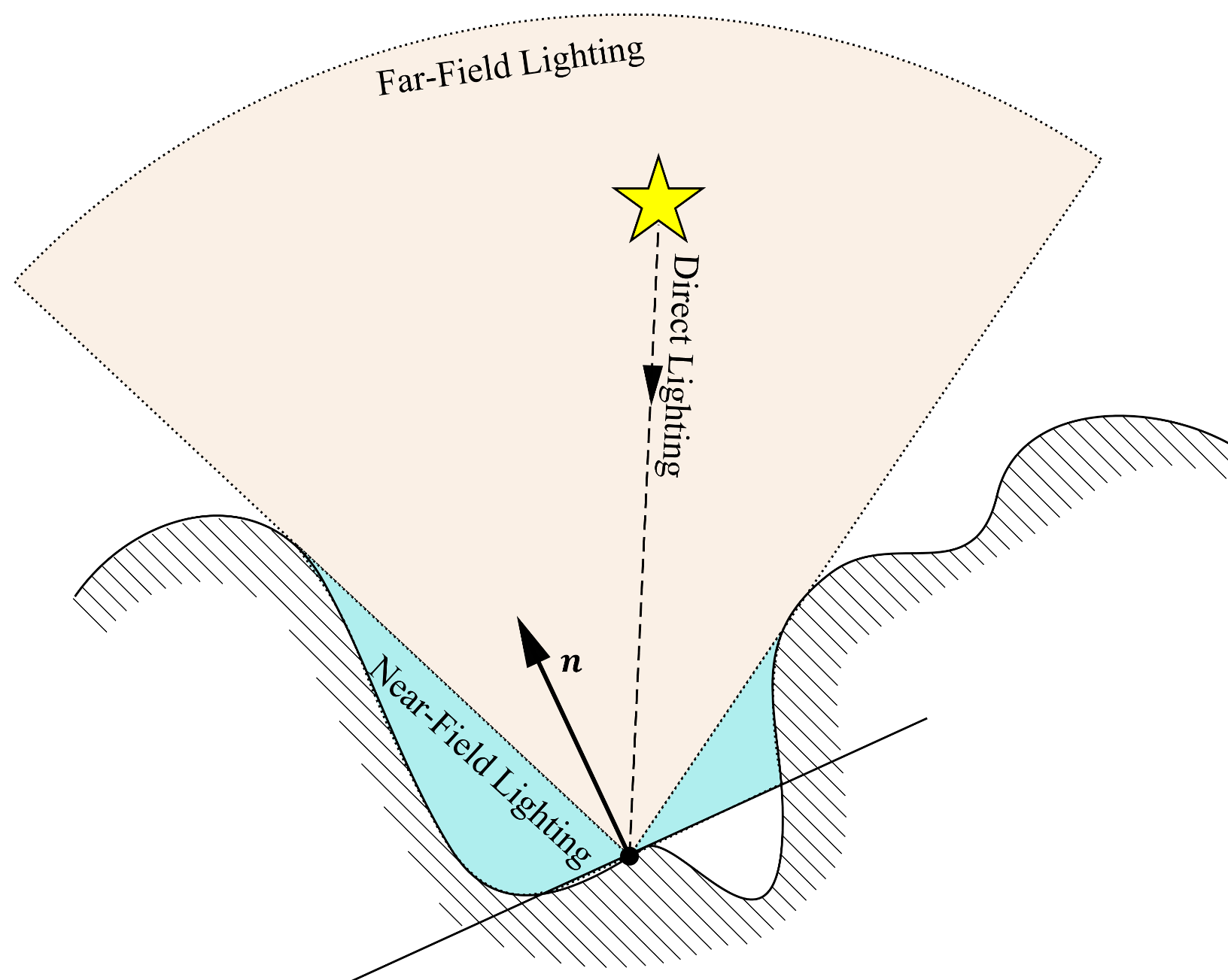
Where:

* 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 collected 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



**Fig. 12.** Representation of the 3 sources of irradiance: Direct lighting by the lights, Far-field lighting from the unoccluded environment and Near-Field Lighting which represents the irradiance that bounced off the neighbor environment. We notice that the Near-Field only accounts for the occluded environment above the normal plane and is the exact complement of the far-field.

We saw in equation 7 how to sample the far-field environment but the full computation for the 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 and is simplified by using , the radiance sampled at location .

Thus, we can rewrite equation 14 as:

***Note:*** We could make the same assumption as in [[3](#REF_3)] and [[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 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 15 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

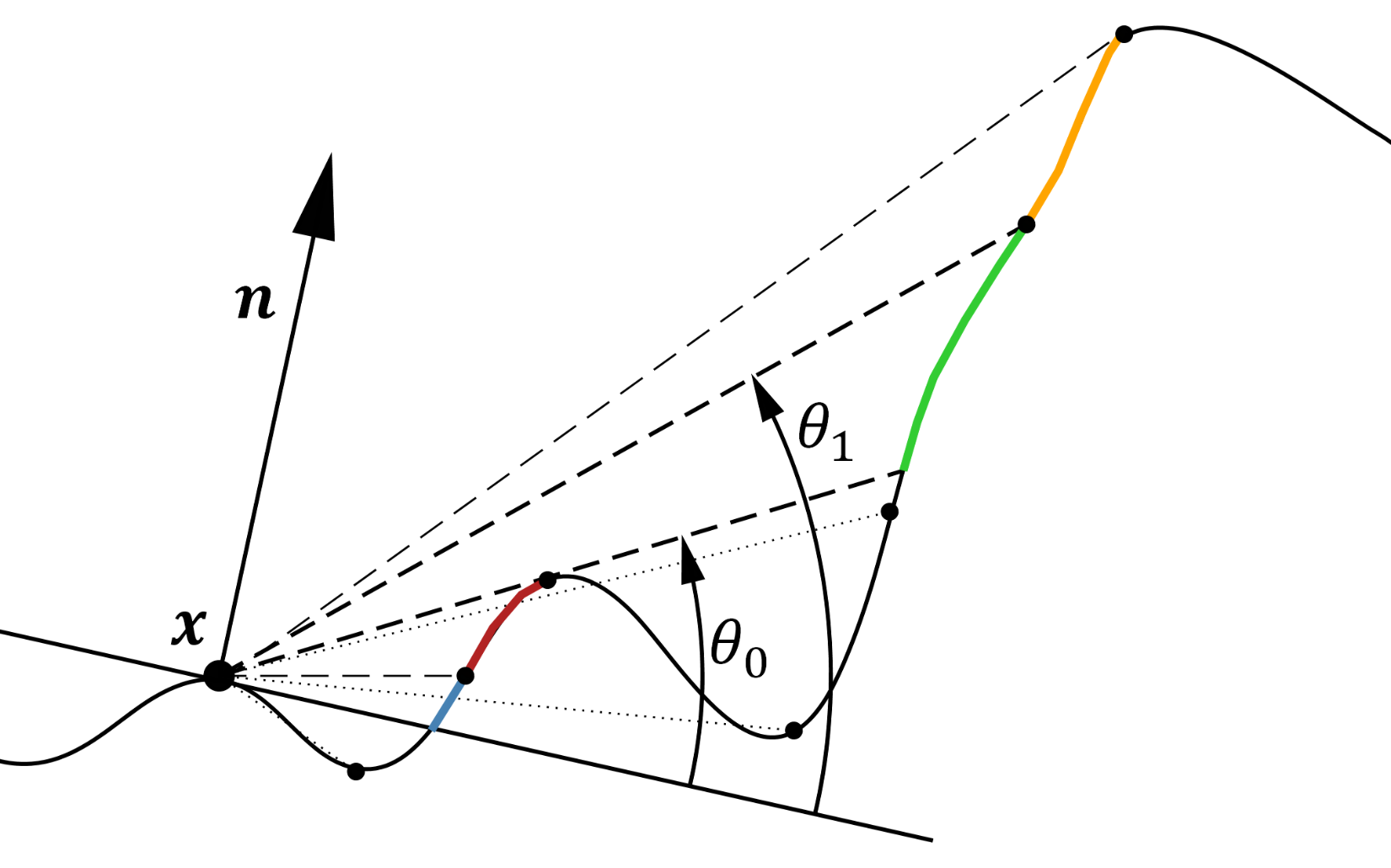
Computing radiance in this fashion would theoretically give us an infinite amount of light bounces.

The details on how to reproject radiance from frame N-1 are given later in section 3.1.

## 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:

****

**Fig. 13.** Sampling the gathered irradiance when the horizon is rising from to : we simply need to sample the neighbor radiance (shown in green) and compute its perceived influence on our central location .

In this figure, we can easily see that the only irradiance we perceive from is the one from when the horizon jumps up a little: only the area represented in green contributes to the perceived irradiance when we rise from to .

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

Solving the integral yields:

***Note:*** We notice that the normal at the neighbor location has no influence over the computation thanks to the fact that we are only considering the diffuse Lambertian reflection here.

## Accounting for the Fresnel Term

*Tagada!*

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

**TODO: Show ON/OFF image**

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:

**TODO: Optimize! Ca rame à mort! ☹**

1. Acknowledgments

Special thanks to Eric Arnebäck for proof reading this paper, Benjamin Lalisse for his clever remarks and general support, 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)*”*

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

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