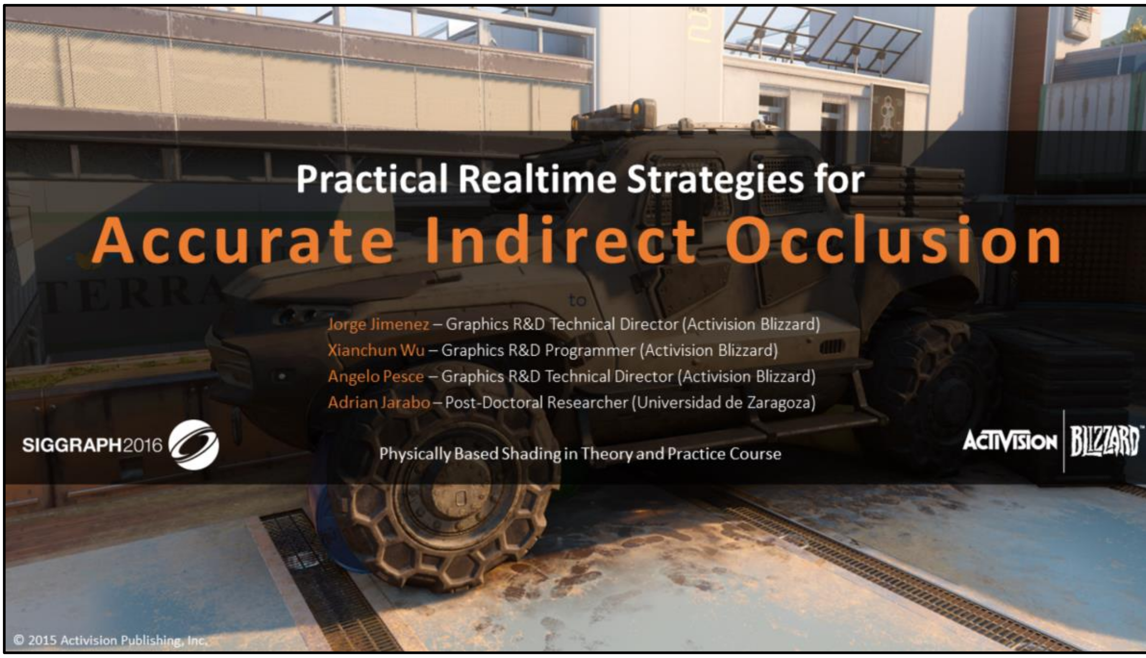


Hi,

Thanks for the introduction.

So, I'll be talking about
accurate indirect occlusion.

Authors

Jorge Jimenez
Activision Blizzard

Xianchun Wu
Activision Blizzard

Angelo Pesce
Activision Blizzard

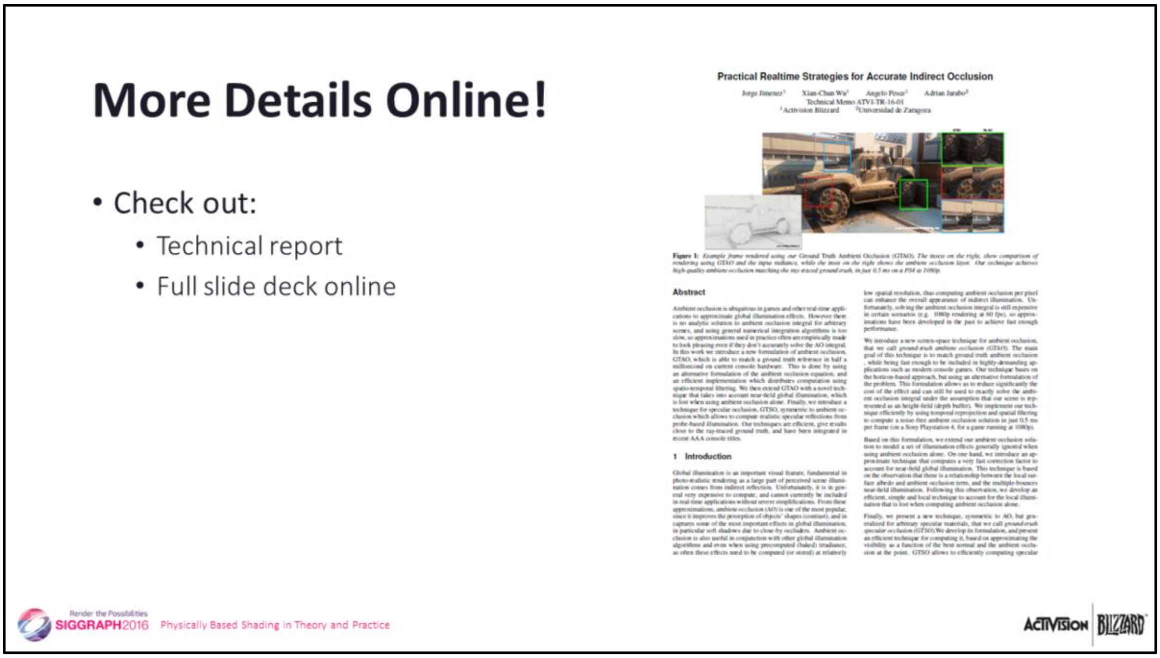
Adrian Jarabo
Universidad de Zaragoza

First of all I'd like to show all the authors of this work,

Xianchun Wu, Angelo Pesce, Adrian Jarabo and me.

Notice for Offline Reading

- Hidden slides in this presentation
- Slide show mode won't show them
- Don't miss the speaker notes, most slides have them



Due to time constraints, I'll try to keep the talk high level.

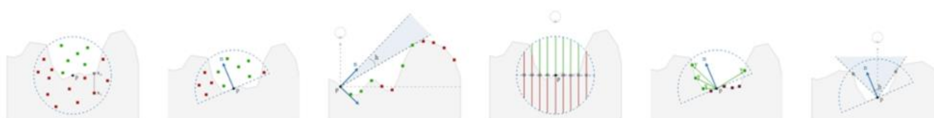
You will find all the details online in our technical report and the full slide deck, so we definitely recommend to check them out.

Yet Another SSAO?

A Comparative Study of Screen-Space Ambient Occlusion Methods

Student
Frederik Peter Aalund
DTU

Supervisor
Andreas Bærentzen
DTU



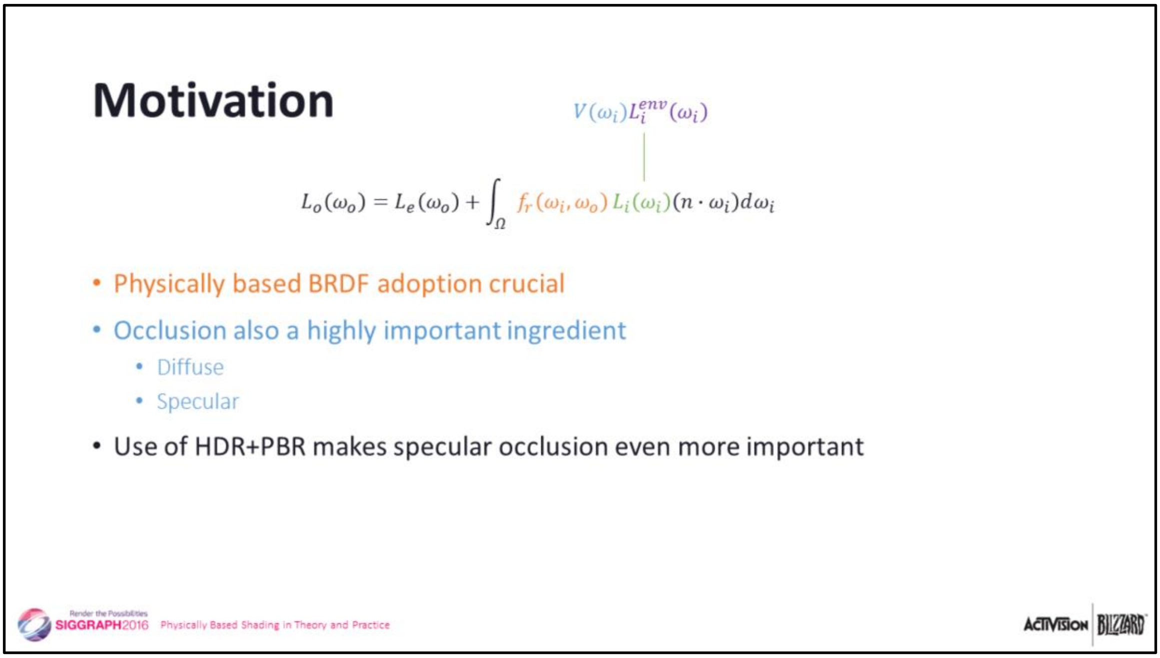
Motivation

$$L_o(\omega_o) = L_e(\omega_o) + \int_{\Omega} f_r(\omega_i, \omega_o)\, L_i(\omega_i)(n \cdot \omega_i) d\omega_i$$

- Physically based BRDF adoption crucial

In the past few years, the adoption of physically-based BRDFs has been a crucial improvement to the consistency and realism of real-time rendering.

Large efforts have been made to improve this term of the rendering equation, marked in orange.



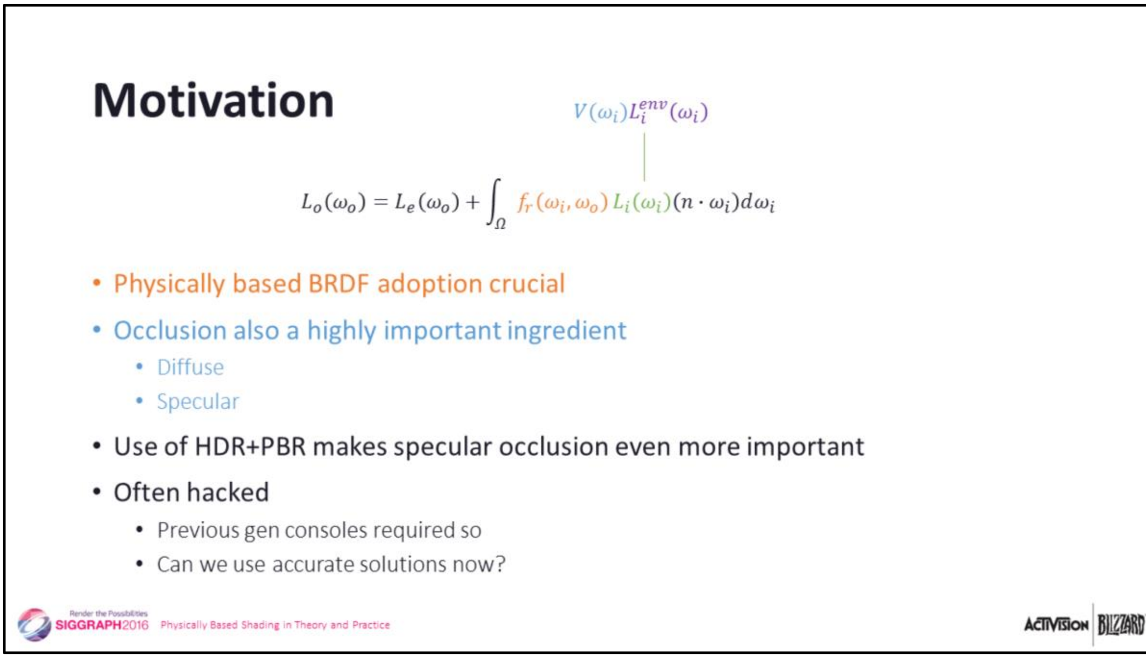
And we have seen improvements in the accuracy of incoming radiance, in green, with the adoption of environment look up tables for image-based lighting.

However, its implicit visibility term, in blue, has received less attention.

We think that both diffuse and specular occlusion are very important ingredients of the rendering equation,
and in this talk we will explore them in more detail.

Without them, no matter how accurate our BRDF and lighting models are,
we are missing that component that makes objects stick to the ground.

That makes them feel part of a connected world, rather than individually composited objects.



In real-time rendering, we often hack or heavily approximate the occlusion,

This is understandable given the constraints of the previous generation of consoles.

But the question that we asked ourselves is: do we still need to do so?

Can we use accurate approaches in reasonable budgets, under the constraints of 60 frames per second?

Bear with me, and we will discover this out…

Methodology

- Monte Carlo Ground Truth
  - Implemented critical parts twice to ensure correctness


3d ray tracer (3d geometry)


Screen-space ray marcher (height map)


Horizon-based numerical integrator

The core of our methodology for both the diffuse and specular occlusion has been to constantly compare with Monte Carlo ground truth at each step we performed, to ensure the correctness of our techniques.

Methodology

- Monte Carlo Ground Truth
  - Implemented critical parts twice to ensure correctness
- Analytical problem
  - Available data (engine)
  - Performance targets


3d ray tracer (3d geometry)


Screen-space ray marcher (height map)


Horizon-based numerical integrator

We derived analytical solutions where possible,
and from there…

Methodology

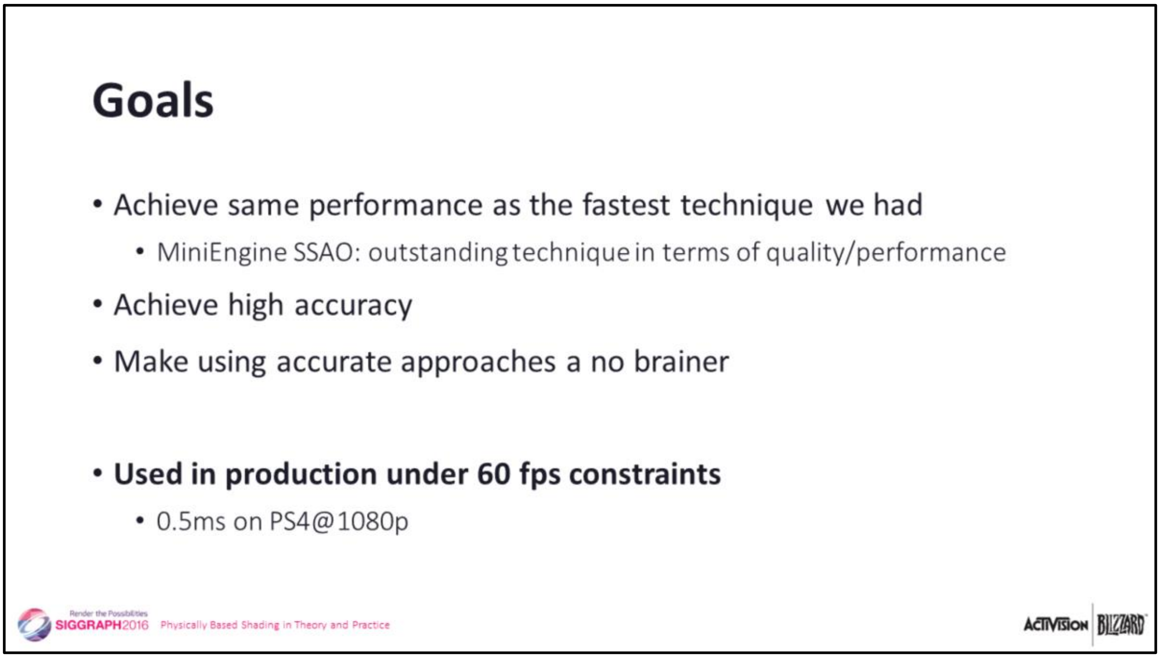
- Monte Carlo Ground Truth
  - Implemented critical parts twice to ensure correctness
- Analytical problem
  - Available data (engine)
  - Performance targets
- Closed-form solution + Fitting residual

3d ray tracer (3d geometry)

Screen-space ray marcher (height map)

Horizon-based numerical integrator

…we found approximations for the residual error from the ground truth.



The ultimate goal was to achieve better quality while staying in the same performance budget as previous techniques,
making this solution a simple drop-in replacement.

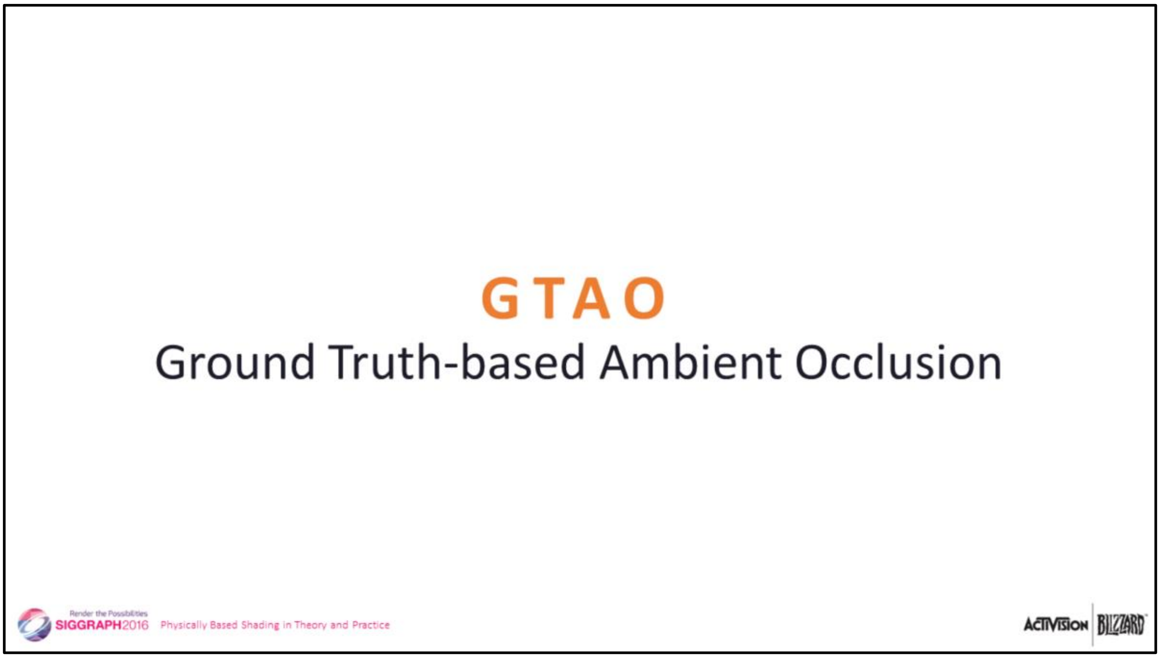
For ambient occlusion, our budget was 0.5ms on the PS4 at 1080p.

Note that I will not cover optimizations during the talk, but you will find them on the online material.

MiniEngine SSAO:
https://github.com/Microsoft/DirectX-Graphics-Samples/blob/master/MiniEngine/Core/Shaders/AoRenderCS.hlsli

Overview

- Ambient Occlusion (GTAO): V_d in this presentation
  - Uniform Weighting
  - Cosine Weighting
  - Multiple Bounces
- Specular Occlusion (GTSO): V_s in this presentation
  - Cone/Cone Intersection Method
  - Cone/Lobe Intersection Method



The first part of this talk will showcase a new screen-space ambient occlusion technique that we called GTAO.

I'd like to start by showing some in-engine renderings.

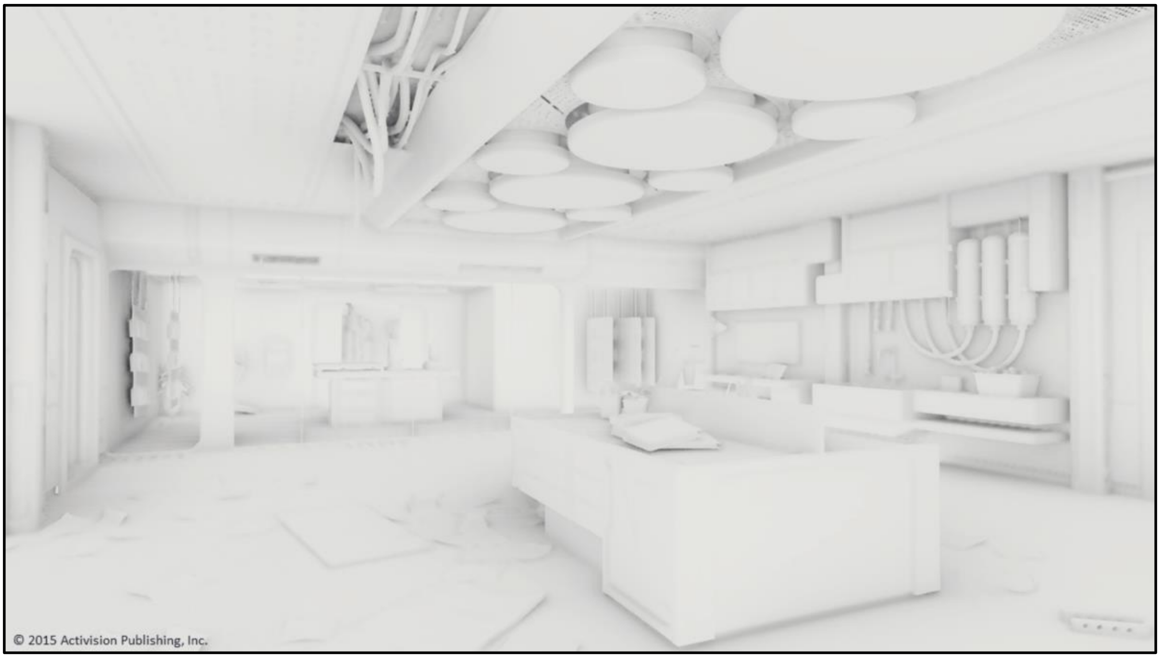
Here we can see an image without any indirect occlusion at all…

…and here with GTAO.

This image clearly shows the importance of having accurate occlusion.

A key observation is that with accurate occlusion, we can see subtle but important changes in lighting in some areas, for example near the wall corners…
…but really strong changes in others, like the tubes on the ceiling.

Let’s go back and forward a couple of times so that you can observe the differences again…



Here you can see the ambient occlusion of this scene.

This is another shot, without ambient occlusion…

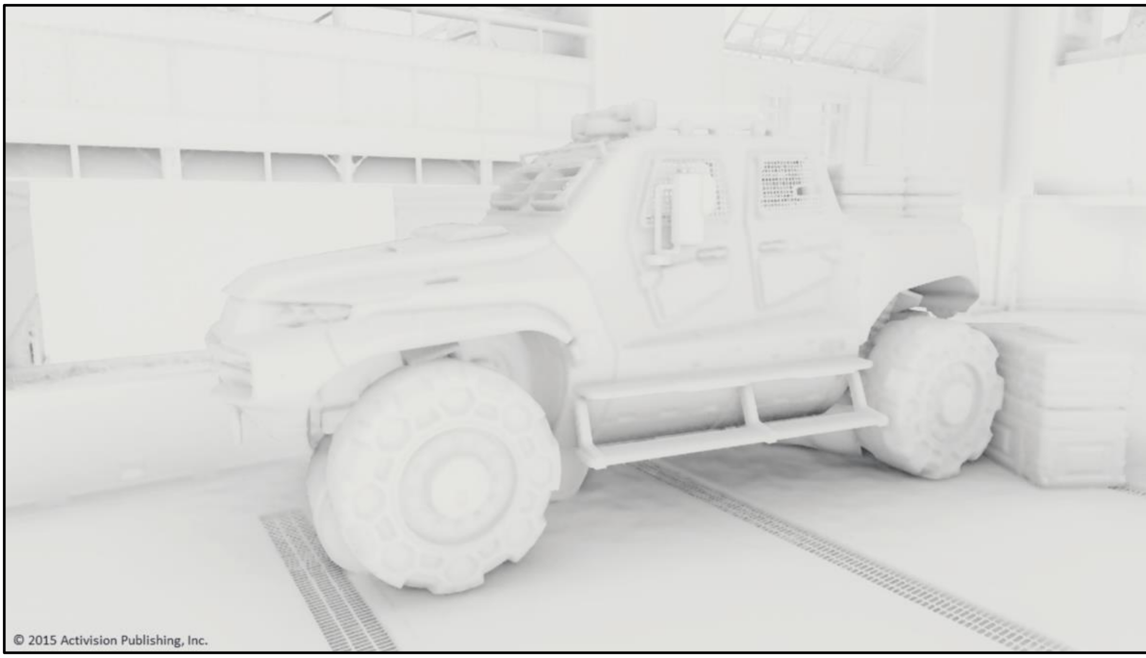
…and with GTAO.

Notice how our technique is not shy of darkening where it needs to be darkened.


© 2015 Activision Publishing, Inc.

And a final example without occlusion…

…and with GTAO


© 2015 Activision Publishing, Inc.


KITCHEN
© 2015 Activision Publishing, Inc.

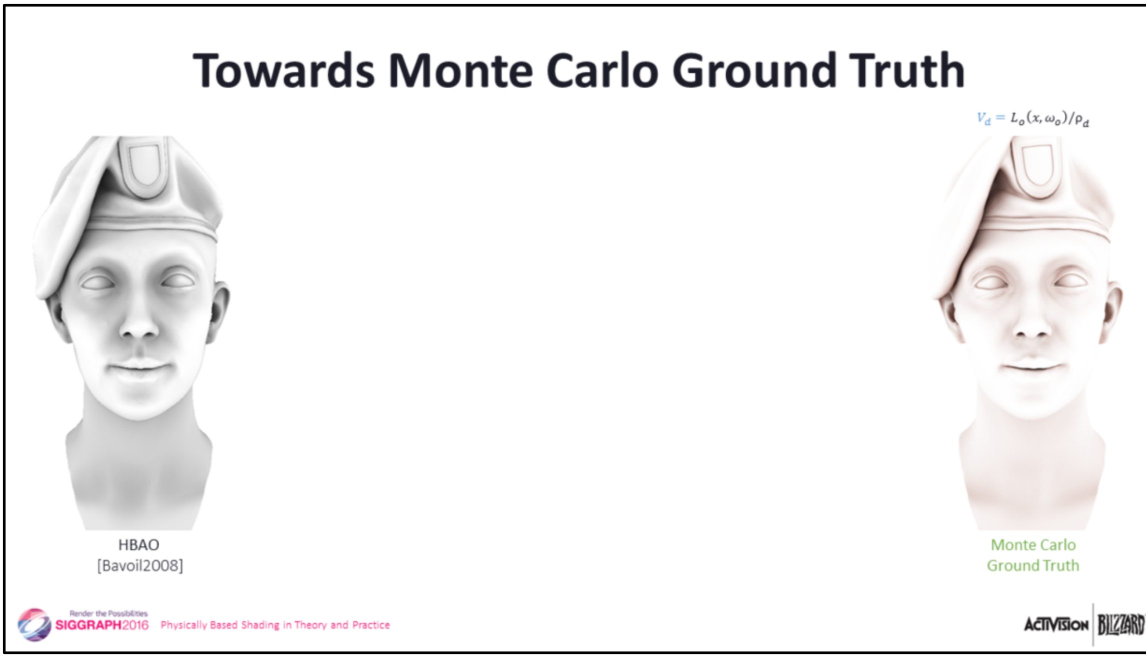

KITCHEN
© 2015 Activision Publishing, Inc.


KITCHEN
© 2015 Activision Publishing, Inc.


NO SIGNAL
© 2015 Activision Publishing, Inc.


NO SIGNAL
© 2015 Activision Publishing, Inc.


© 2015 Activision Publishing, Inc.



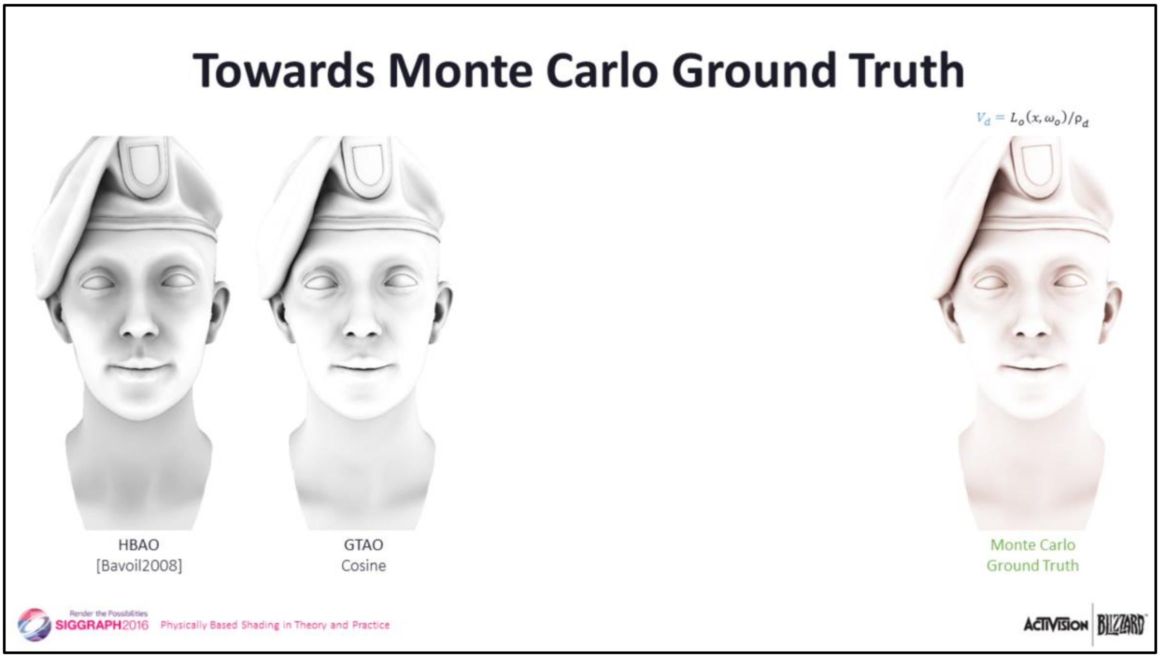
I'll start with a teaser of what will be presented.

On the left, we have uniform weighting ambient occlusion, which is what we often use.

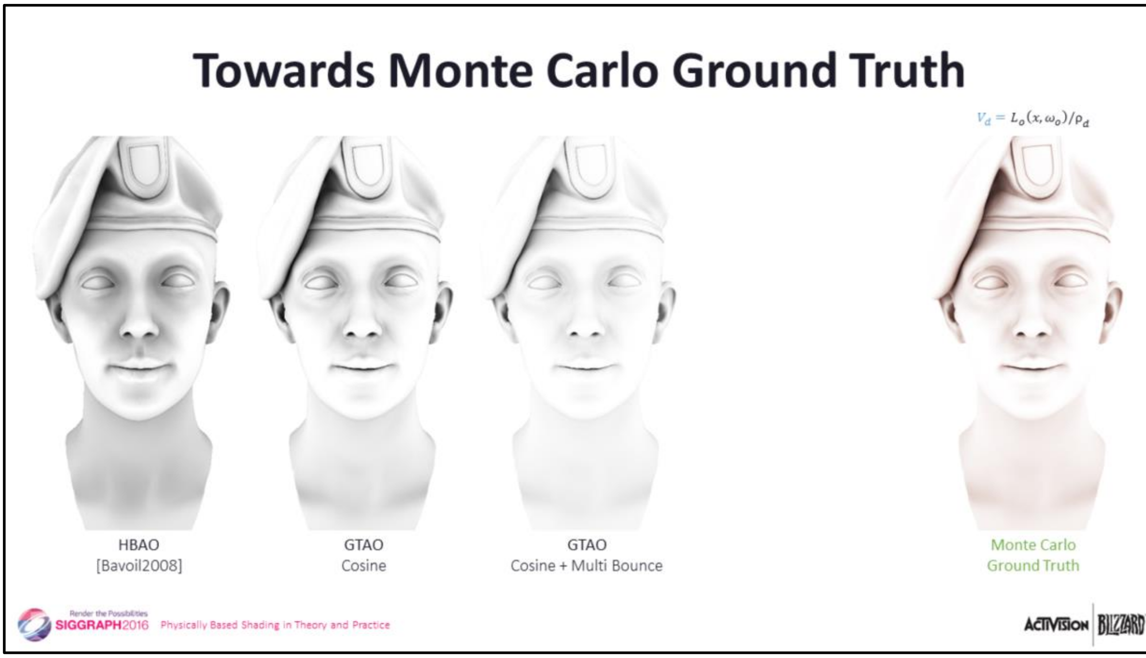
We will show our journey from this, towards achieving a close match to the occlusion in a Monte Carlo rendering with multiple light bounces, which is on the right.

Notice how different they look.

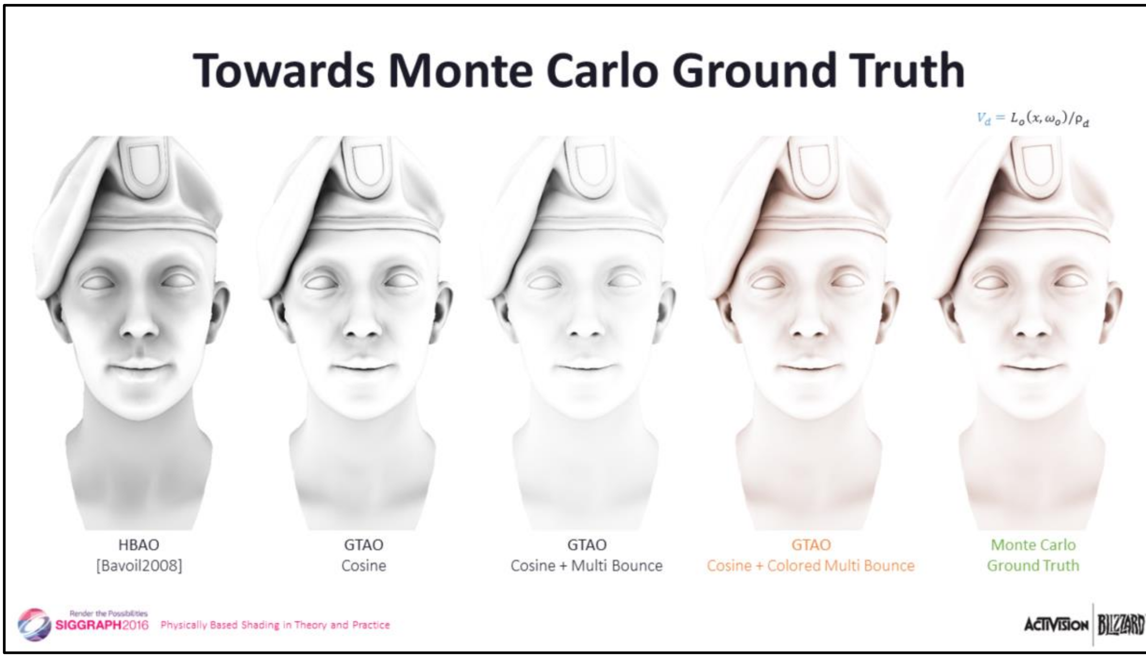
So, we want to go further than the classic occlusion equation, and attempt to match real lighting occlusion instead.



We will show how to add the cosine term to horizon-based ambient occlusion…



…and we will show what happens when we consider more than a single bounce of light…



…and the extension for colored objects.

Here we can see that our final solution, marked in orange,
is a close match for the Monte Carlo reference.

What Is Ambient Occlusion?

$$L_o(\omega_o) = L_e(\omega_o) + \int_\Omega f_r(\omega_i, \omega_o)\, L_i(\omega_i)(n \cdot \omega_i)\, d\omega_i$$

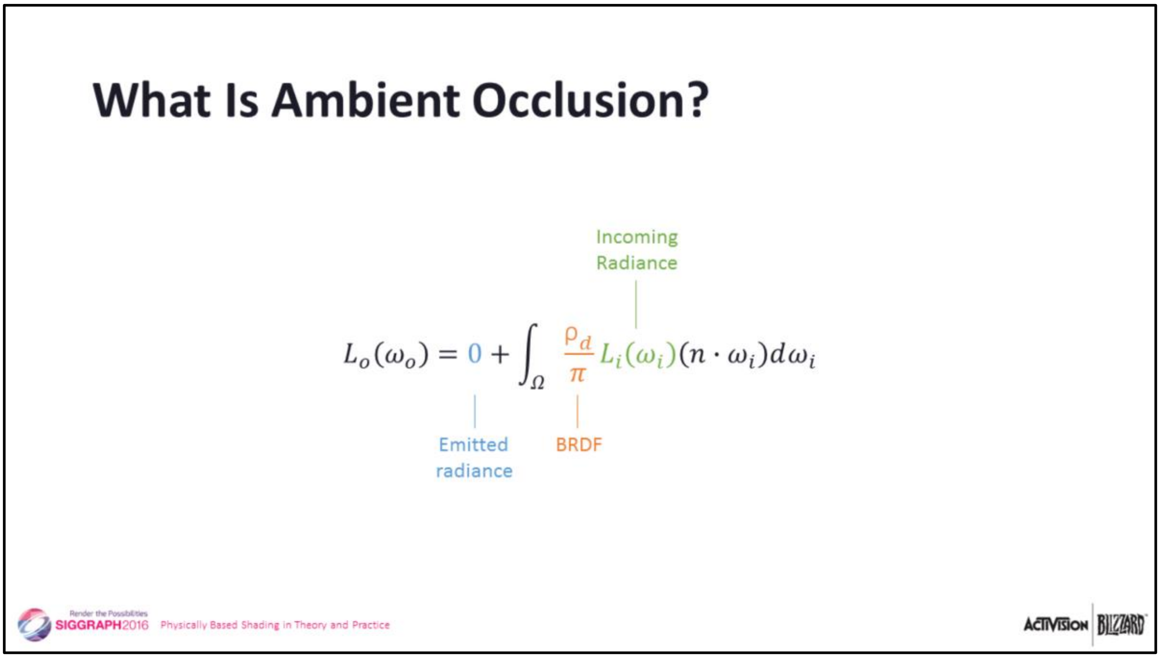
Emitted radiance — BRDF — Incoming Radiance

I'd like to start with the core or basics of our technique, and then show how we improved on that.

So for that, we need to define, what is ambient occlusion?

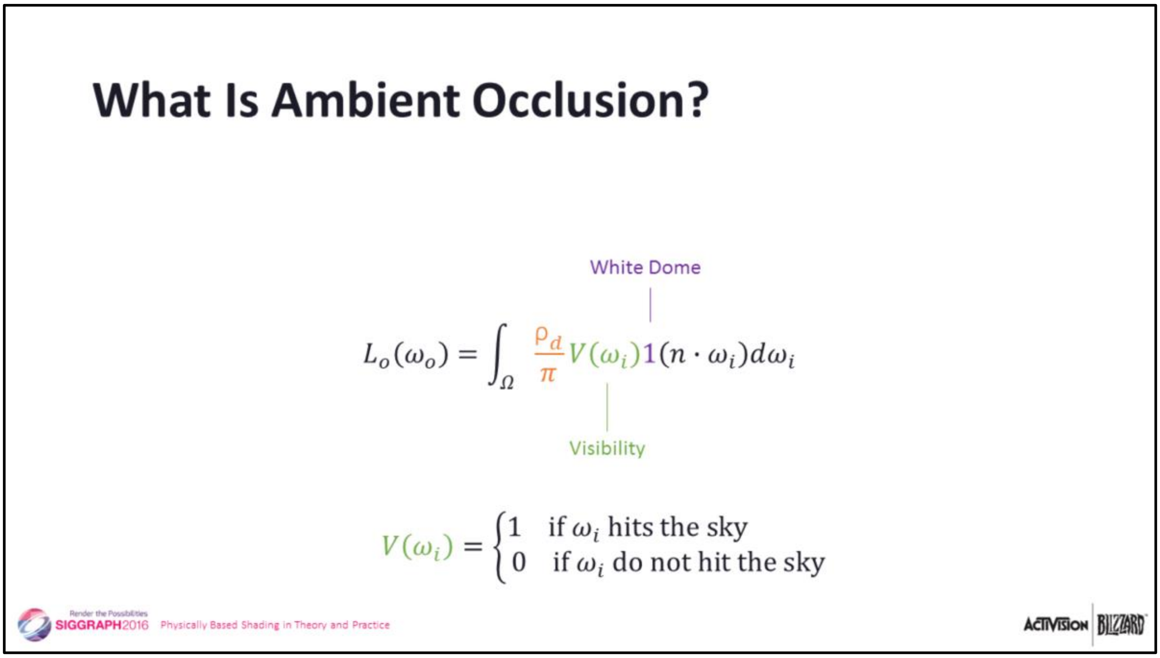
Let's start with the rendering equation, where you can see the emitted and incoming radiance and the BRDF.

If we assume there is no emission, and we use the Lambertian BRDF…



…we get this.

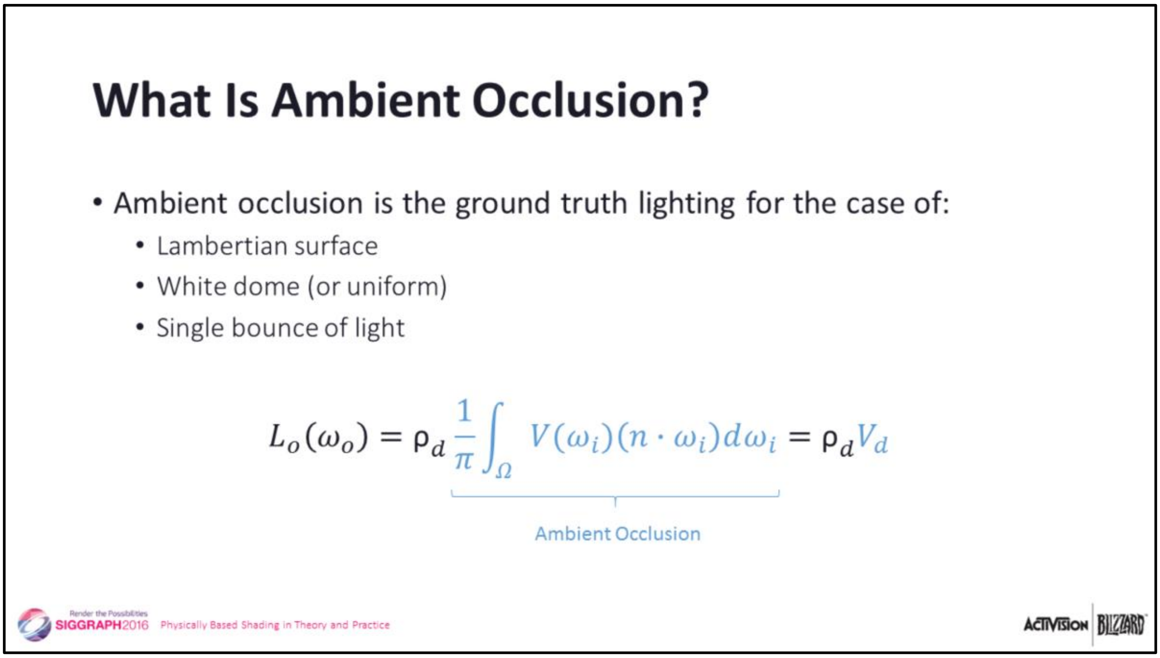
If we then assume a constant white dome is illuminating the scene, and we only calculate a single bounce of light…



…we obtain this.

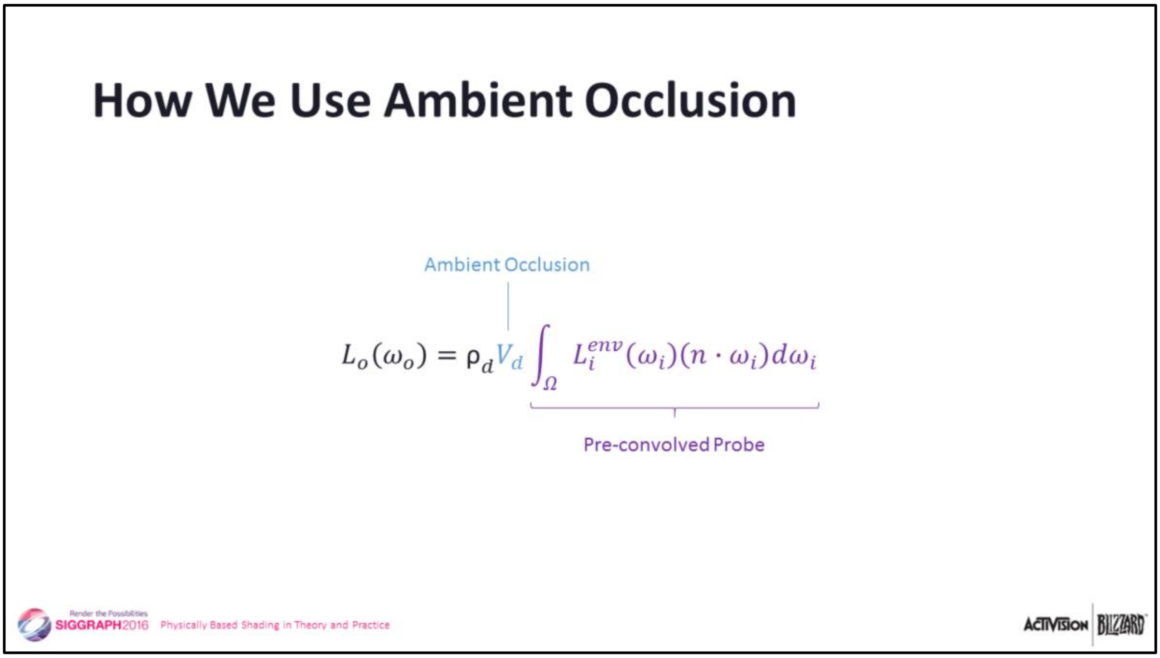
Notice a new visibility term appeared, that specifies if a ray hits the sky or not.

Then if we rearrange the terms…



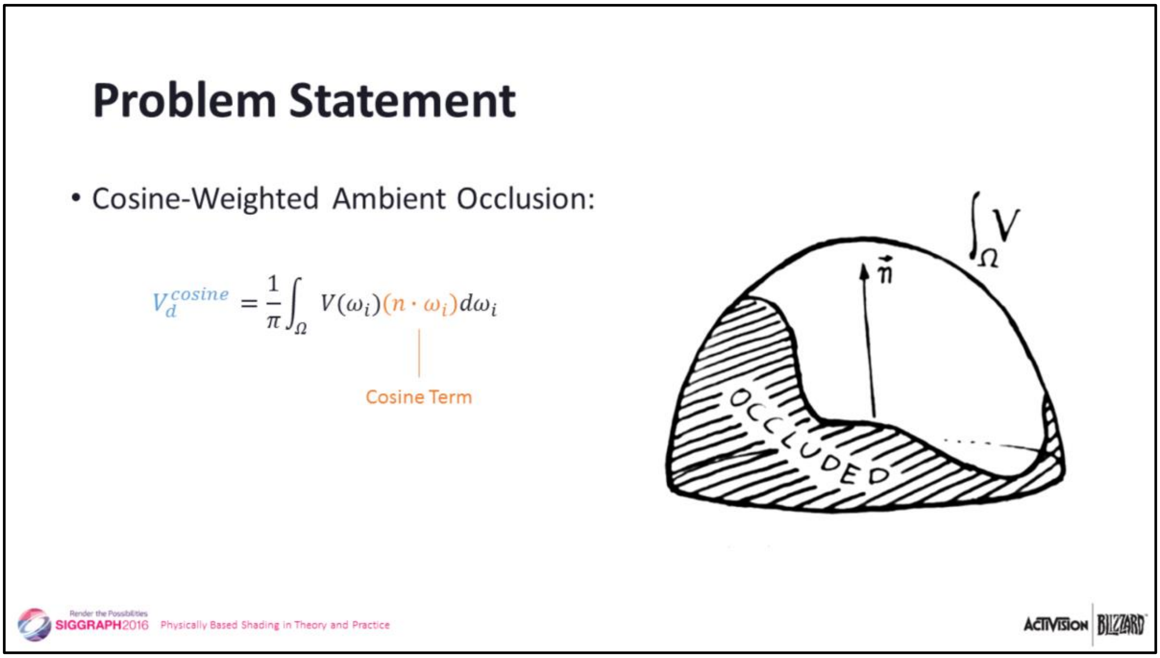
…we arrive to the classic definition of ambient occlusion.

So, we can say that the ambient occlusion multiplied by the albedo is the lighting for the case of:
a white dome,
a single bounce of light and
a Lambertian surface.



In real scenes though we don't typically have uniform white lighting.

The typical approach to account for this is to just multiply the ambient occlusion by the pre-convolved probe,
which will still be accurate for the case of a white dome, but not for any other scenario.



So, now that we have defined what is ambient occlusion, let’s continue with the problem statement.

We want to find a solution to the ambient occlusion equation, which in simple terms is just the visible area of the hemisphere,
weighted by a cosine term modelling the foreshortening.

Sometimes, ambient occlusion is solved with uniform weighting…

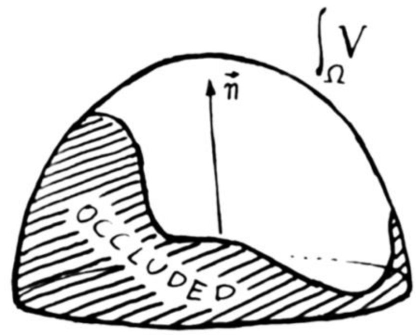
Problem Statement

- Cosine-Weighted Ambient Occlusion:

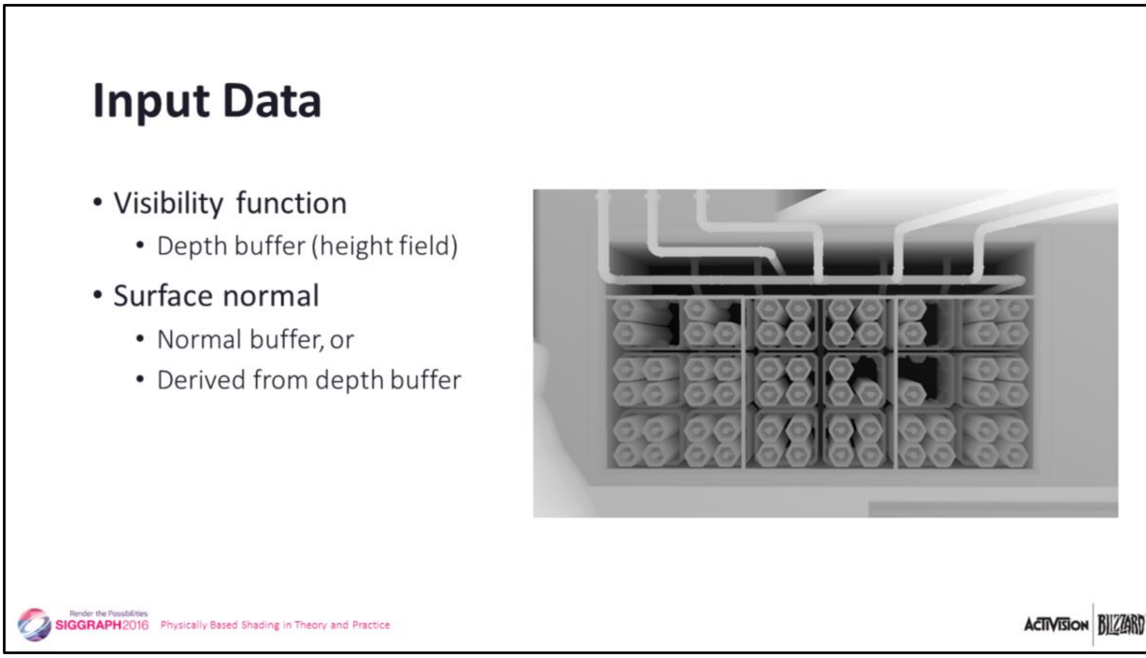
$$V_d^{cosine} = \frac{1}{\pi}\int_{\Omega} V(\omega_i)(n \cdot \omega_i)d\omega_i$$

- Uniformly-Weighted Ambient Occlusion:

$$V_d^{uniform} = \frac{1}{2\pi}\int_{\Omega} V(\omega_i)\, d\omega_i$$



...which unfortunately doesn't yield ground truth results for the assumptions we made, so we won't use it here.



This is the input data that we have.

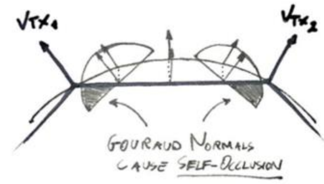
The surface normal, which can either come from a normal buffer or derived from the depth buffer.

And the visibility function, which in our case comes from a depth buffer given that we work in screen space.

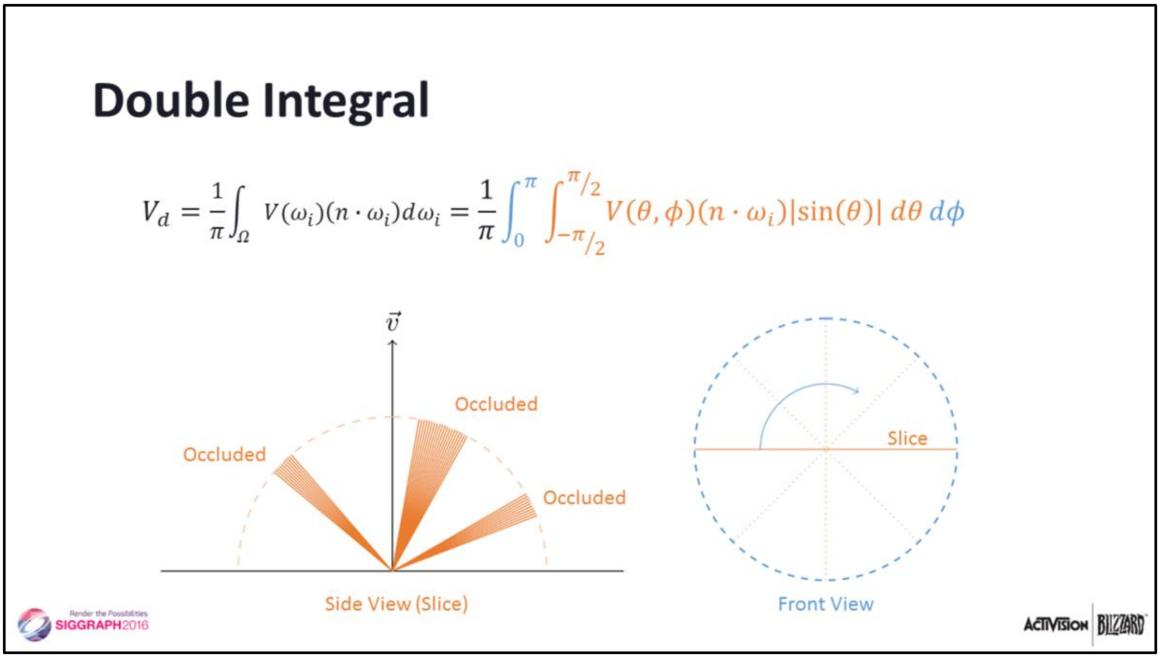
This means that the scene is represented as an height field, and as we'll see later on, this is a very important observation.

Which Normals?

- Shading
  - Gouraud & Normal-Maps
- Geometric
  - Can be approximated via z-buffer differentials



- No perfect solution
  - Gouraud interpolation causes erroneous self-occlusion
    - Normals don't belong to the actual surface
  - Normal-Maps could be desirable
    - Often easier/best to bake their occlusion in the surface (not if tiling/compositing)
    - Be sure not to double-occlude (in the maps and in the SSAO)
  - Geometric normals can make the faceted nature of the mesh visible
    - As hemisphere orientation "snaps" sharply when moving from triangle to triangle
  - This problem affects Monte-Carlo path tracing as well



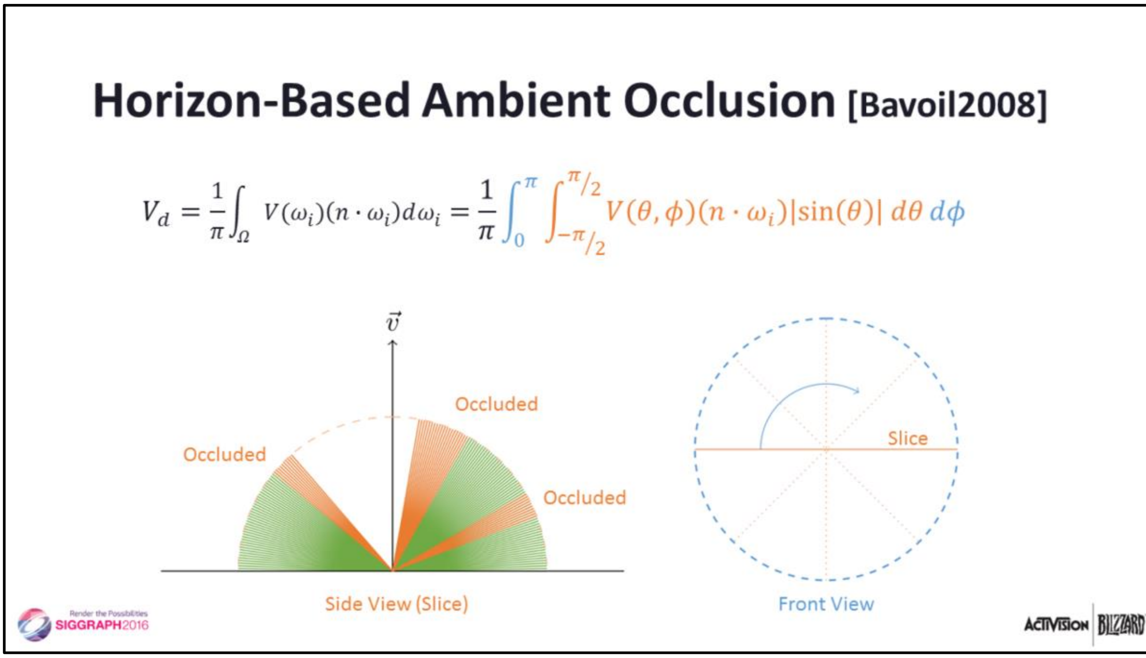
We can calculate the ambient occlusion integral as a double integral in polar coordinates.

The inner integral integrates the visibility for a slice of the hemisphere, as you can see in the left,
and the outer integral swipes this slice to cover the full hemisphere.

The simplest solution would be to just numerically solve both integrals.

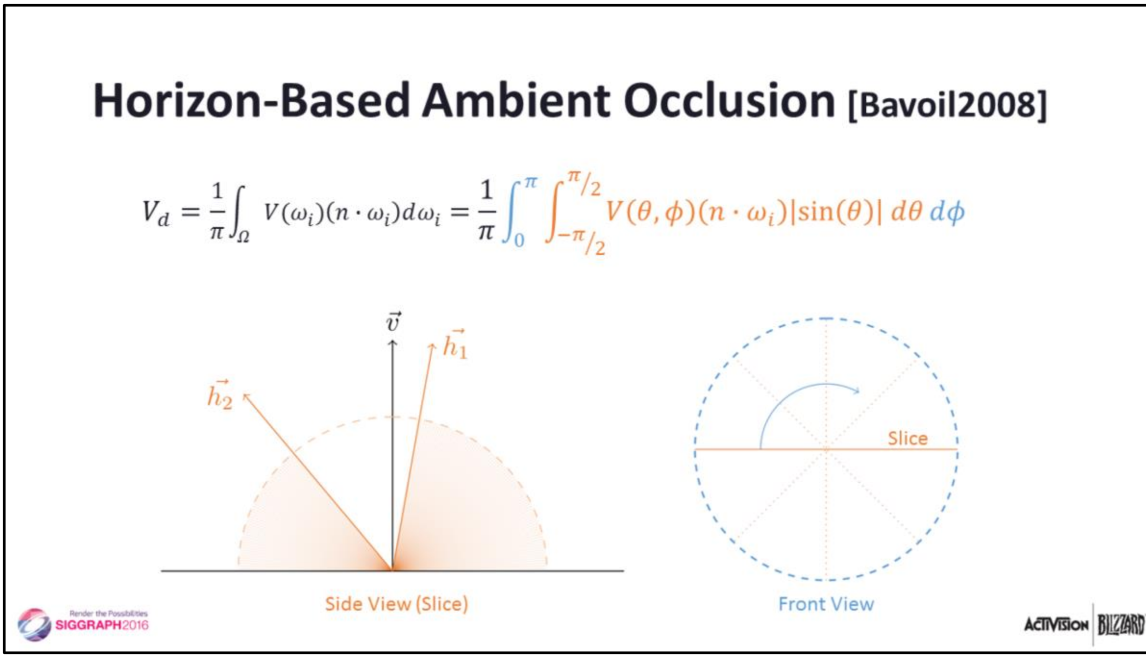
But the solution we chosen, horizon-based ambient occlusion, which was introduced by Louis Bavoil in 2008,
made the key observation that the occlusion as pictured here can't happen when working with height fields.

Using height-fields we would never be able to tell that the areas in…



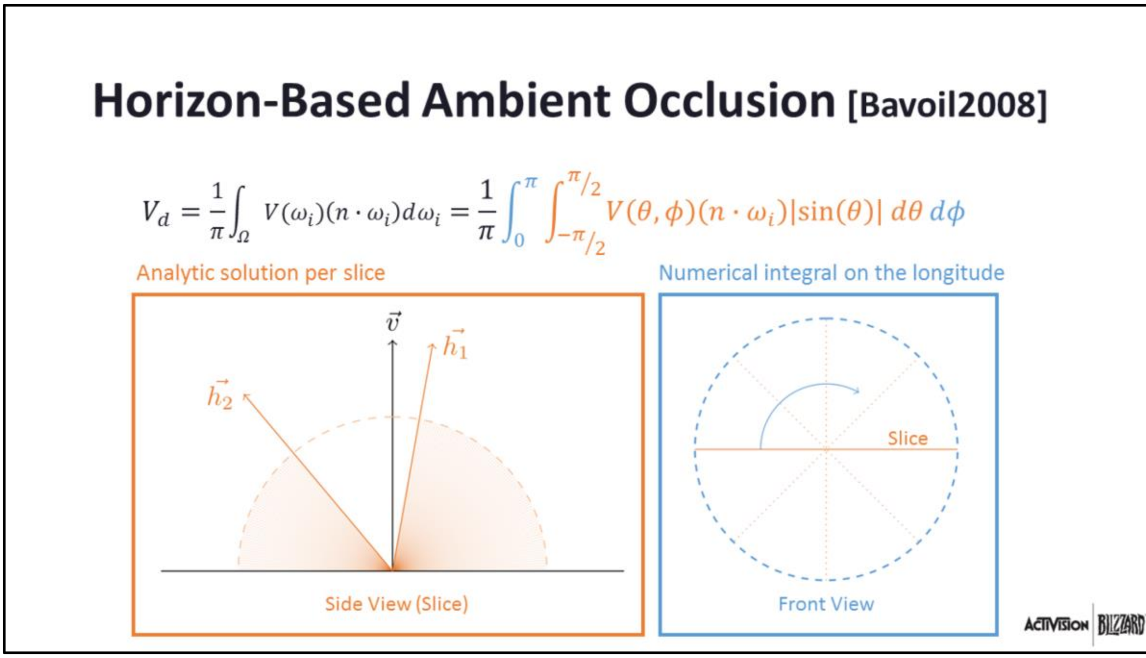
…green here, are actually visible.

The key consequence of this, is that we can just search for the two horizons h1 and h2…



…and that captures all the visibility information that can be extracted from a height map,
for a given slice.

So, with this information at hand, we no longer need to calculate both integrals numerically…



…and can instead perform the inner integral, in orange, analytically,

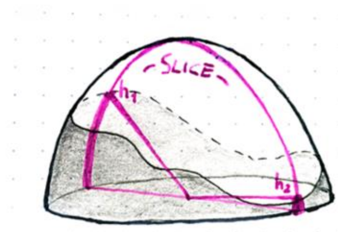
which is substantially faster.

The original horizon-based ambient occlusion technique used uniform weighting,

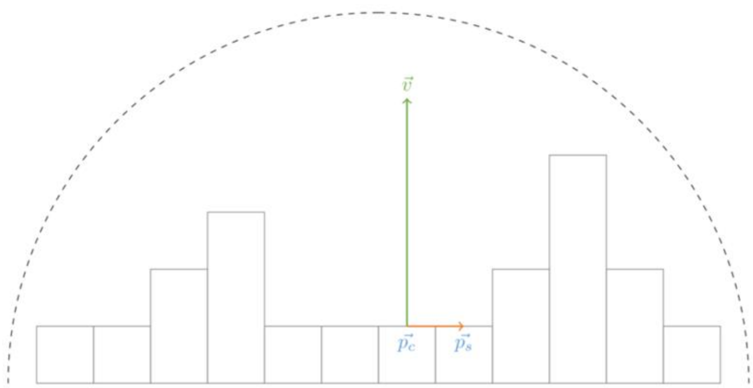
so this means that we need to figure out how to do this analytical integral for the cosine weighting case.

Horizon-Based Ambient Occlusion [Bavoil2008]

- Discrete data ⟶ Not fully closed-form
- Depth buffer ⟶ Single visibility aperture
  - We only see the visibility by tracing the depth buffer
  - Depth buffer considered by SSAO algorithms to be a height field
- Horizon model [Bavoil2008] is the best we can do!
  - Compute the occluded angles of each longitudinal slice
  - Compute the AO integral in the unoccluded area

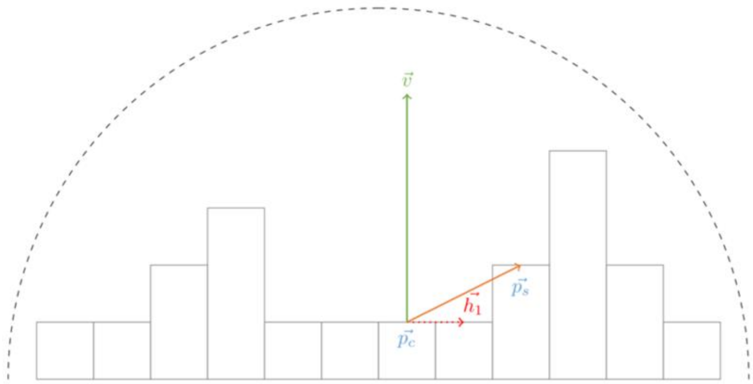


Searching for Horizons



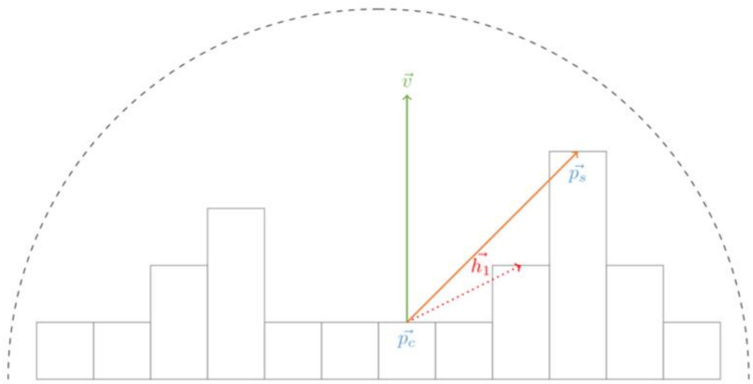
Side View (Slice)

Searching for Horizons



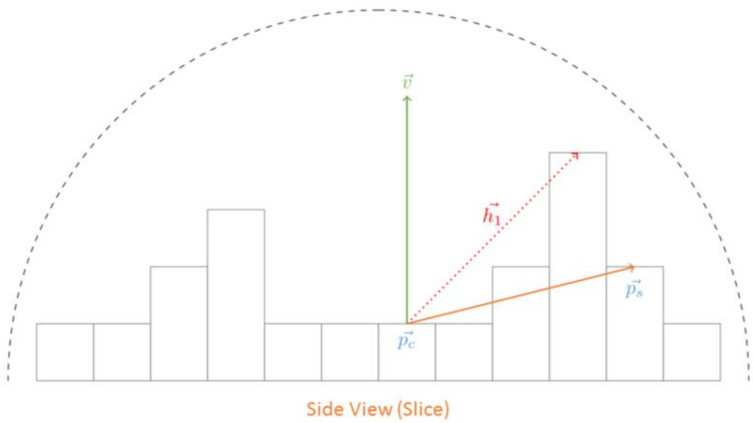
Side View (Slice)

Searching for Horizons

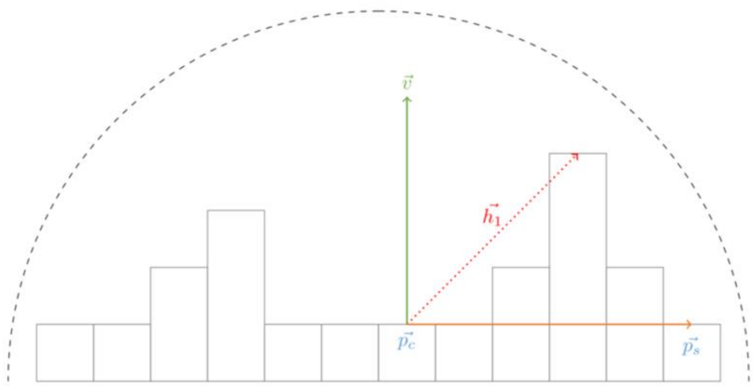


Side View (Slice)

Searching for Horizons

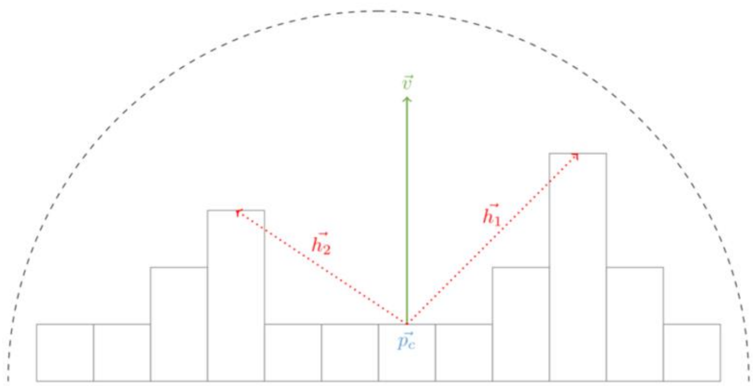


Searching for Horizons

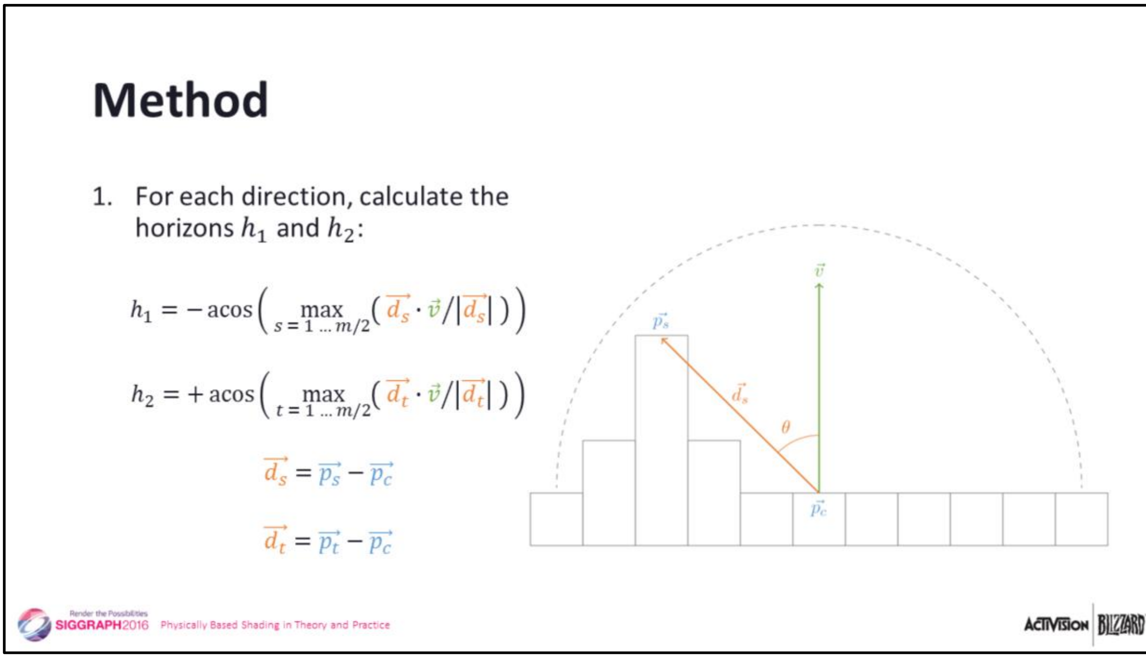


Side View (Slice)

Searching for Horizons



Side View (Slice)



$\vec{p_s}$ and $\vec{p_c}$: sample and center positions in view space

$\vec{v} = -\vec{p_c}$: view vector

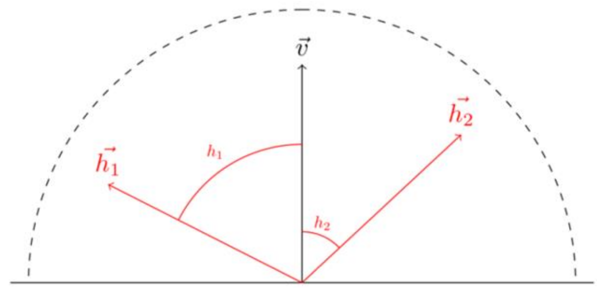
m: sample count

In this diagram, $\vec{d_s}$ samples to the left and $\vec{d_t}$ to the right.

Method

2. Calculate the visibility v_d for this slice:

$$v_d = IntegrateArc(h_1, h_2)$$



Method

- Slightly different from HBAO:
  - No sample attenuation (obscurance)
    - Can't match ground truth renderings if we use obscurance
    - One integral per direction rather than per sample (reduces ALU overhead)
    - We still do conservative attenuation (more details later on)
    - Instead of using obscurance to avoid the overdarkening produced by ignoring near-field interreflections, we add this lost light (more details later on)
  - Not using attenuation allows to integrate with respect to the view vector $\vec{v}$ rather than the XY plane
    - We integrate from $\vec{v}$ to horizon instead of integrating from XY to horizon and from XY to tangent (halves number of integrals)
  - Simple search loop (`dot`, `rsqrt` and `max`)
    - Shader becomes completely memory bound
    - Aiming for ground truth calculations becomes free (we do *optimal* math with a given sample set)
  - Integrates 180° slices instead of 90° ones
    - Reduces ALU overhead by sharing calculations
- HBAO is mathematically equivalent to our method (with uniform weighting) if no obscurance is used

NVIDIA's HBAO Implementation

- Contains a performance optimization that prevents to reproduce the results shown in this presentation:

```
data.MaxRadiusPixels = 0.1f * min(data.FullResolution[0], data.FullResolution[1]);
data.MaxRadiusPixels = 300.0f;
```

- Compute shader version does further optimizations that bias the results
- Pixel shader one is the recommended for visual reference

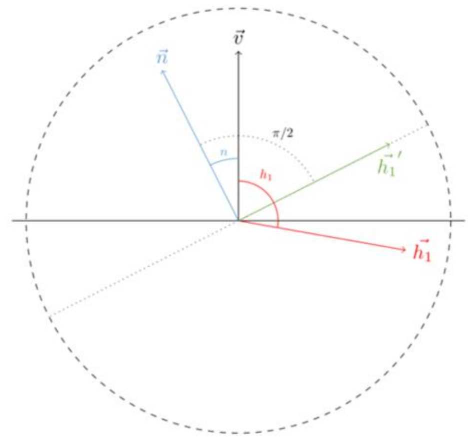
- **Note that HBAO+ is the latest implementation and seems to not be horizon based**

Integration Domain

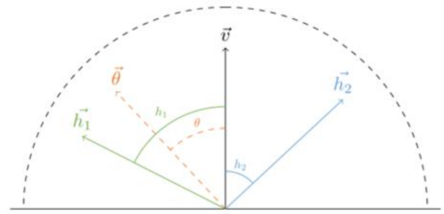
Note: h_1 is negative

$$h_1{}' = n + \max(h_1 - n, -\pi/2)$$
$$h_2{}' = n + \min(h_2 - n, \pi/2)$$

- We (have to) slice in screen-space circles
- The spherical integral axis is the view-vector
- We search for horizons in the full sphere
  - The horizons can be in the negative view hemisphere (h_1)
  - We have to clamp them with the normal hemisphere ($h_1{}'$)



Uniform Weighting



- Ambient occlusion equation:

$$V_d^{uniform} = \frac{1}{2\pi}\int_0^{\pi} \underbrace{\int_{-\pi/2}^{\pi/2} V(\theta,\phi)|\sin(\theta)|\, d\theta}_{v_d}\, d\phi$$

- Using horizon angles h_1 and h_2, we have the visibility for a single slice v_d:

$$v_d = IntegrateArc(h_1, h_2) =$$

$$\int_0^{h_1} |\sin(\theta)|\, d\theta + \int_0^{h_2} |\sin(\theta)|\, d\theta =$$

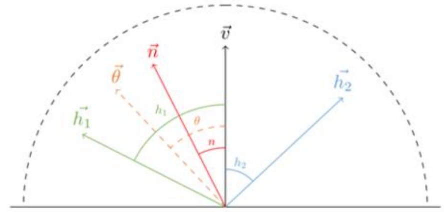
$$(1 - \cos(h_1)) + (1 - \cos(h_2))$$

Monte Carlo Ground Truth

GTAO
Uniform

Render the Possibilities
SIGGRAPH2016

Cosine Weighting



- Ambient occlusion equation:

$$V_d^{cosine} = \frac{1}{\pi}\int_0^{\pi} \underbrace{\int_{-\pi/2}^{\pi/2} V(\theta,\phi)\cos(\theta - n)\,|\sin(\theta)|\,d\theta}_{v_d}\,d\phi$$

- Using horizon angles h_1 and h_2, we have the visibility for a single slice v_d:

$$v_d = IntegrateArc(h_1, h_2, n) =$$

$$\int_0^{h_1} \cos(\theta - n)|\sin(\theta)|\,d\theta + \int_0^{h_2} \cos(\theta - n)|\sin(\theta)|\,d\theta =$$

$$\frac{1}{4}(-\cos(2h_1 - n) + \cos(n) + 2h_1\sin(n)) + \frac{1}{4}(-\cos(2h_2 - n) + \cos(n) + 2h_2\sin(n))$$

Render the Possibilities SIGGRAPH2016

ACTIVISION BLIZZARD

To solve ambient occlusion with cosine weighting,
we integrate the visibility from the view vector to h1, marked in green,
and the visibility from the view vector to h2, marked in blue,
taking the cosine term into account, marked in purple on the equation.

It is a bit more involved than this, but I'll leave the details to the technical report and the online slides.

[Note: here v_d and $IntegrateArc$ are NOT the same as in the $V_d^{uniform}$ case, we just avoided renaming everything with "uniform" and "cosine" not to clutter the equations]

Cosine Weighting Normal Projection

- So far, we assumed that normal lies on the sampling slice

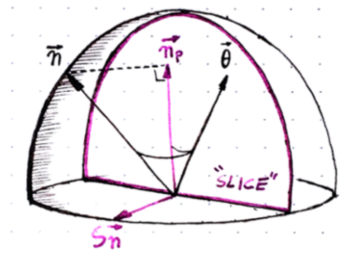
- [Timonen2013b] showed that:

$$V_d^{cosine} = \frac{1}{\pi}\int_0^{\pi}\int_{-\pi/2}^{\pi/2} V(\theta,\phi)(\vec{n}\cdot\vec{\theta})|\sin(\theta)|\, d\theta\, d\phi$$

$$= \frac{1}{\pi}\int_0^{\pi}\|\overrightarrow{n_p}\|\int_{-\pi/2}^{\pi/2} V(\theta,\phi)(\overrightarrow{n_p}\cdot\vec{\theta})|\sin(\theta)|\, d\theta\, d\phi$$

  where $\overrightarrow{n_p}$ is the normal projected onto the sampling plane

- In practical terms:
  - Project (and normalize) the normal to the slice plane for the integration
  - Multiply each slice contribution by the length of the projected normal



Cosine Weighting LUT vs Analytical

- Cosine weighting already used for horizon-based ambient occlusion [Timonen2013b]
  - Tabulated LUT solution with per sample attenuation
- Analytic solution practical for us
  - ALU not a problem as we're memory bound
  - By design we do not use per sample attenuation
  - Only one integral per direction instead of per sample
- Transcendental functions after optimization:
  - 2 `cos` and 1 `sin`
  - 3 `acos` also needed for setting up the integration domain
- Use fast `sqrt` [Drobot2014a] and `acos` [Eberly2014]

```
float GTAOFastSqrt( float x )
{
    // [Drobot2014a] Low Level Optimizations for GCN
    return asfloat( 0x1FBD1DF5 + ( asint( x ) >> 1 ) );
}

float GTAOFastAcos( float x )
{
    // [Eberly2014] GPGPU Programming for Games and Science
    float res = -0.156583 * abs( x ) + GTAO_PI / 2.0;
    res *= GTAOFastSqrt( 1.0 - abs( x ) );
    return x >= 0 ? res : GTAO_PI - res;
}
```

Possible Pitfall

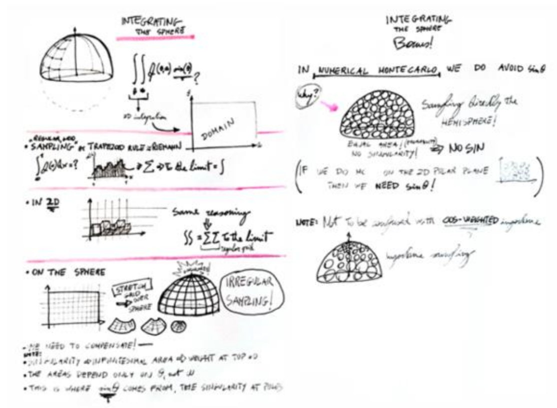
- Using:

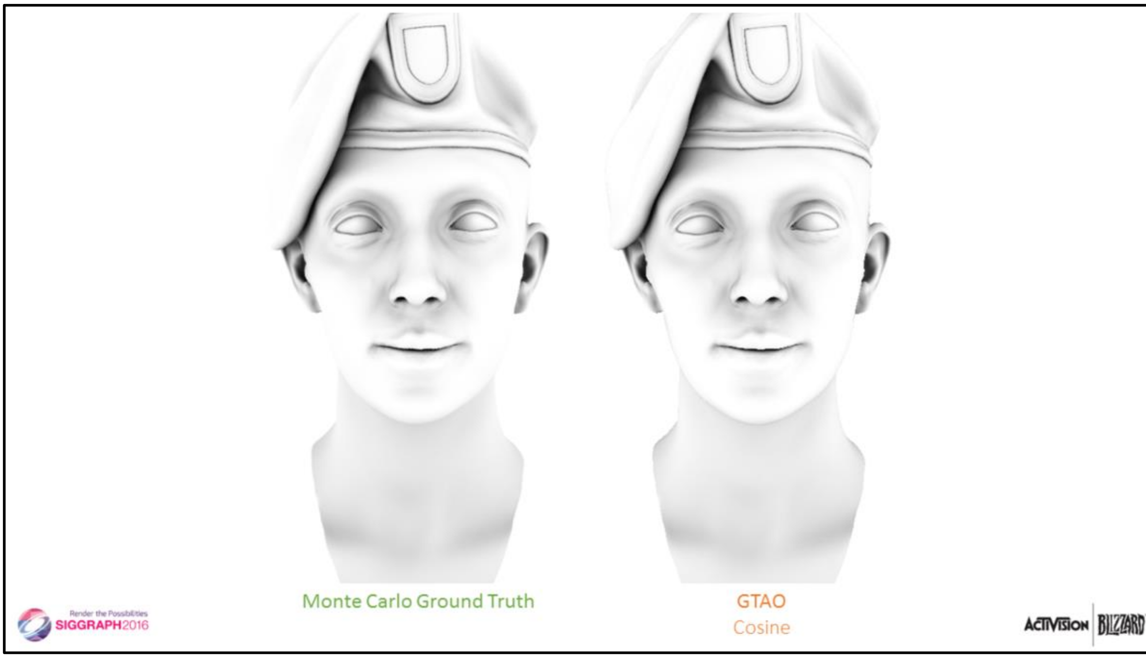
$$\int_{-\pi/2}^{\pi/2} V(\theta, \phi)\cos(\mathrm{n})|\sin(n)|\, d\theta$$

- Instead of:

$$\int_{-\pi/2}^{\pi/2} V(\theta, \phi)\cos(\mathrm{n})|\sin(\theta)|\, d\theta$$

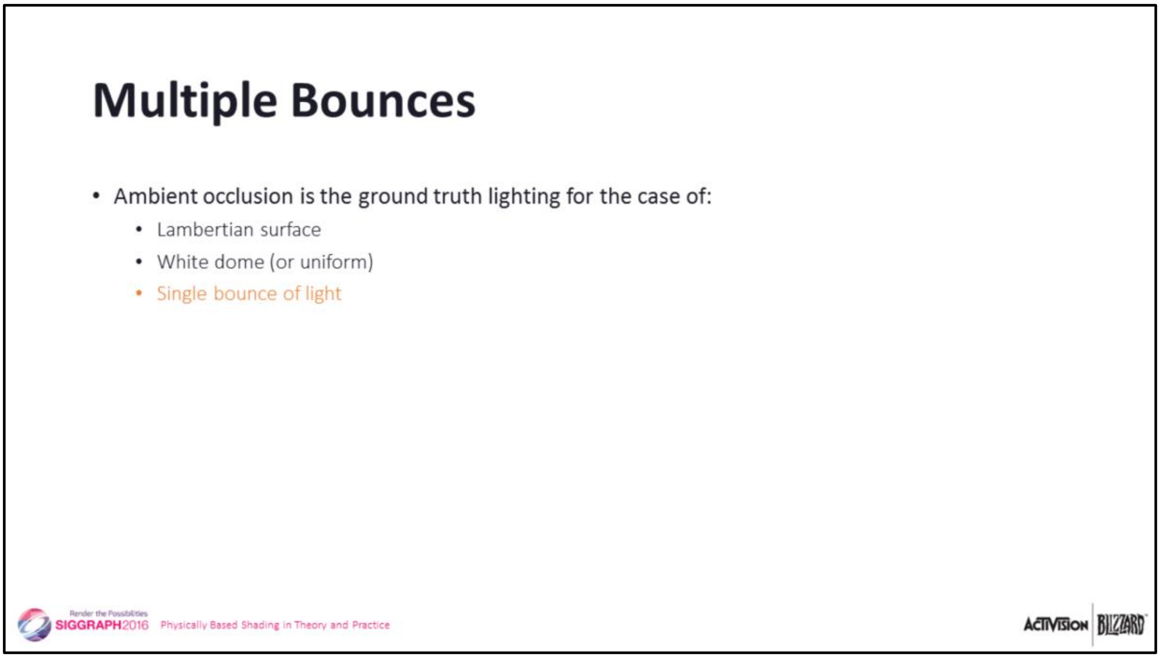
- Why not?
  - We search for horizons on the whole sphere
  - $\sin(\theta)$ ensures regular sampling in the view-space hemisphere
  - $\cos(\mathrm{n})$ due to the cosine weighting (obviously) done with respect to the normal angle $\mathbf{n}$



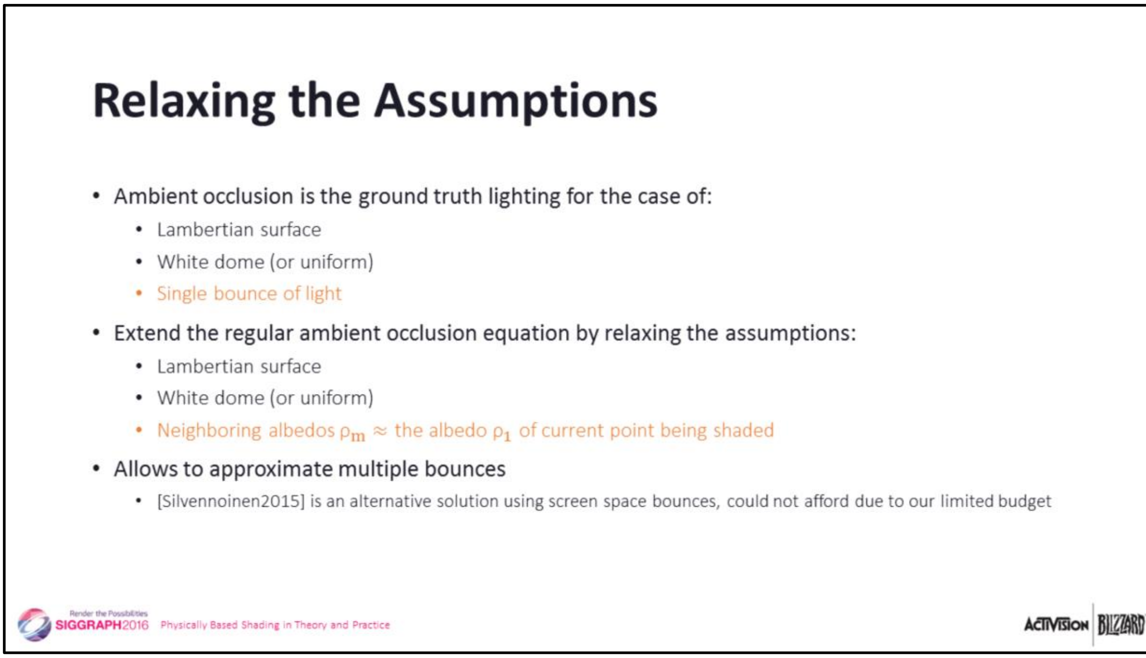


In this slide we have a comparison of the ground truth
and our results so far.

So, we have done a quick recap of how to efficiently calculate the integral using horizon-based ambient occlusion,
and then we have shown how to take the cosine term into account.

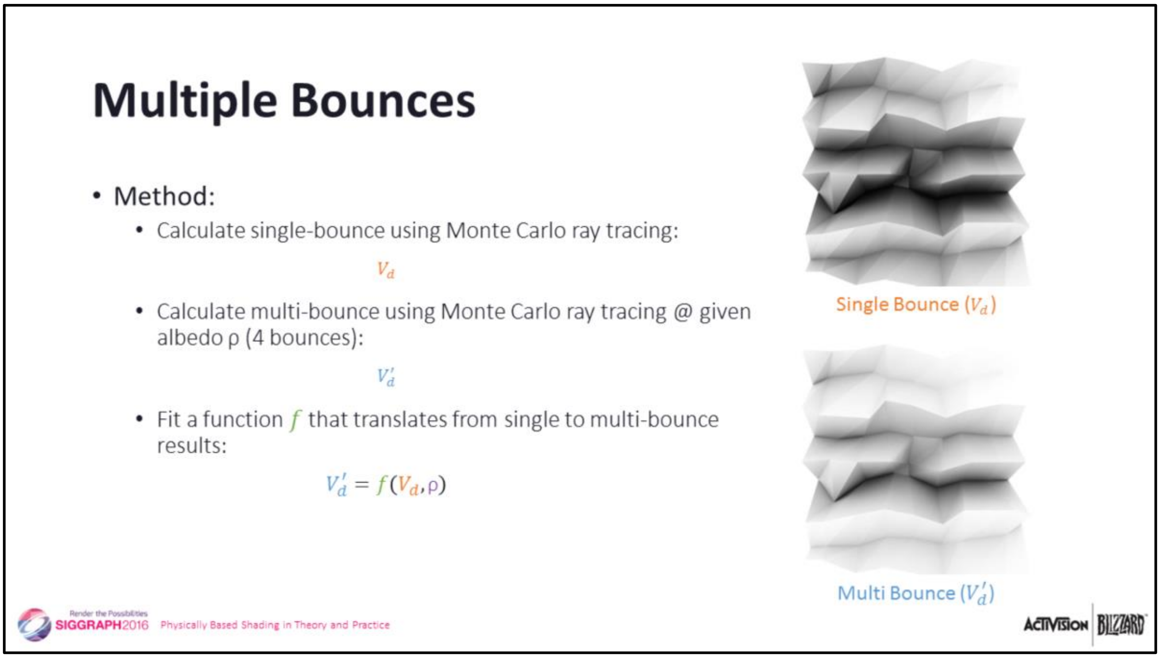


Now, to move this forward, we will relax the assumption of using a single bounce of light…



…and replace it with the assumption that the albedo for the point being shaded is similar to that in the close neighborhood.

Doing this allows to approximate multiple bounces of light.



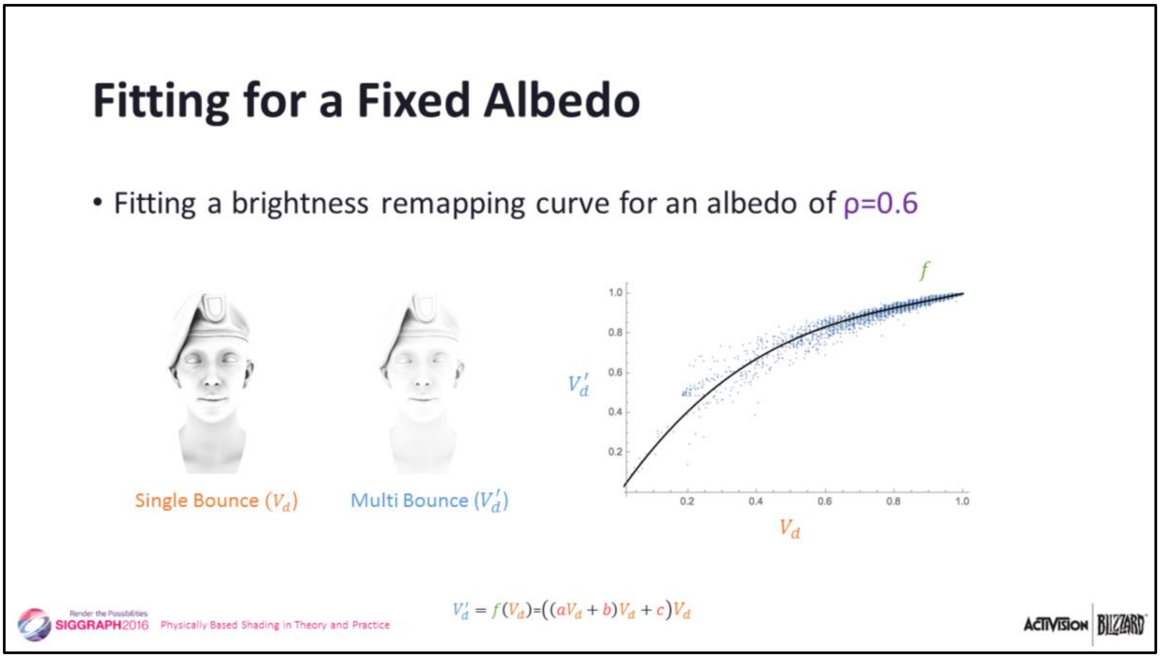
The high level idea is simple.

We calculate references using single and multi bounce Monte Carlo raytracing, for a given albedo, which is shown in the images on the right.

Now the idea is that we can perhaps fit a function that will map the single-bounce results to the multi-bounce ones.

Note that most previous techniques tried to avoid the overdarkening produced by AO introducing Ambient Obscurance,
which empirically assigns less weight to far away occluders.

We instead try to directly account for the lost energy with a correction function on the AO value,
and derive this function from data.



So, we started looking at the data for a given albedo, in this case 0.6.

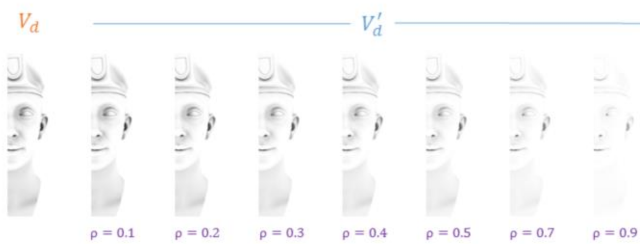
On the right we have a plot of how intensities in the single bounce image on the horizontal axis,
map to the ones in the multi bounce image on the vertical axis.

For this data, we found that fitting a cubic polynomial function was sufficient to model this correlation.

Fitting Over Varying Albedo

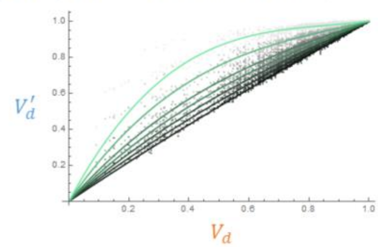
- Seven single to multi-bounce references, for various albedos ρ
- Fit a cubic polynomial per albedo:

$$V_d' = ((aV_d + b)V_d + c)V_d$$



Cubic Polynomial Coefficients

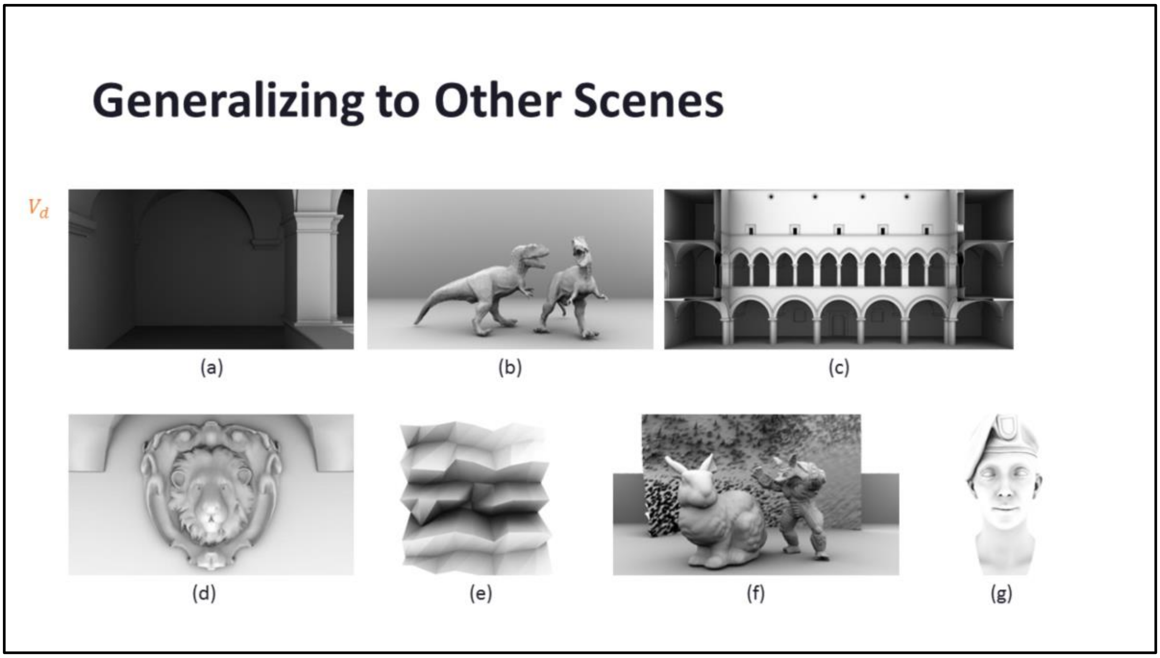
ρ = 0.1	a → 0.0363517	b → -0.162324	c → 1.12599
	a → 0.0999267	b → -0.376556	c → 1.27692
	a → 0.183839	b → -0.632143	c → 1.44889
ρ = 0.4	a → 0.289824	b → -0.933065	c → 1.64413
	a → 0.437788	b → -1.3147	c → 1.87812
	a → 0.805044	b → -2.22354	c → 2.4206
ρ = 0.9	a → 1.35375	b → -3.48326	c → 3.13291



Then, to generalize to varying albedos, we calculated seven multi-bounce references, for albedos ranging from 0.1 to 0.9.

For each albedo, we calculate its own cubic polynomial fit.

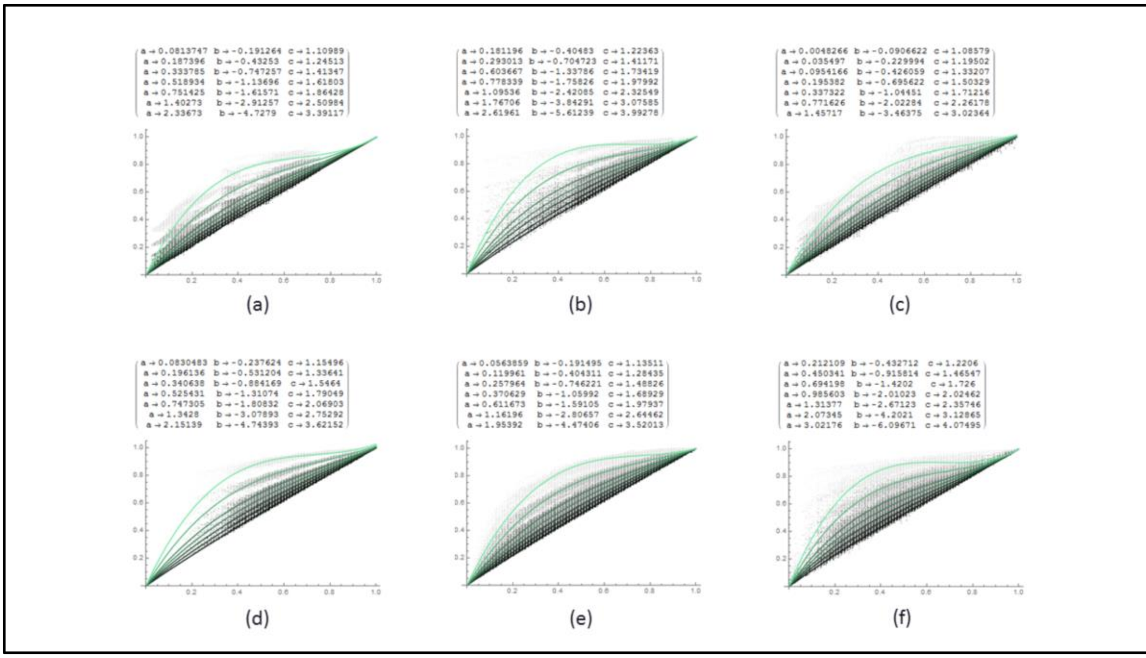
On the right you can see the 7 polynomials, and their coefficients, and on the plot how the curve changes shape as the albedo increases.



We found our fits to work great for our test case, which was a human head.

But we wanted to discover if the technique generalizes to other cases.

Se we prepared a larger dataset and performed fittings for all the scenes shown here.



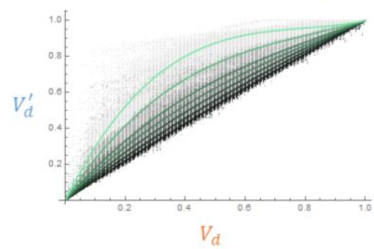
You can observe that even if our fitting functions are not exactly the same for all the scenes,
they all look reasonably similar.

Especially for the lower albedos, which are the ones that we more often find in nature.

Final Fit Over All Data

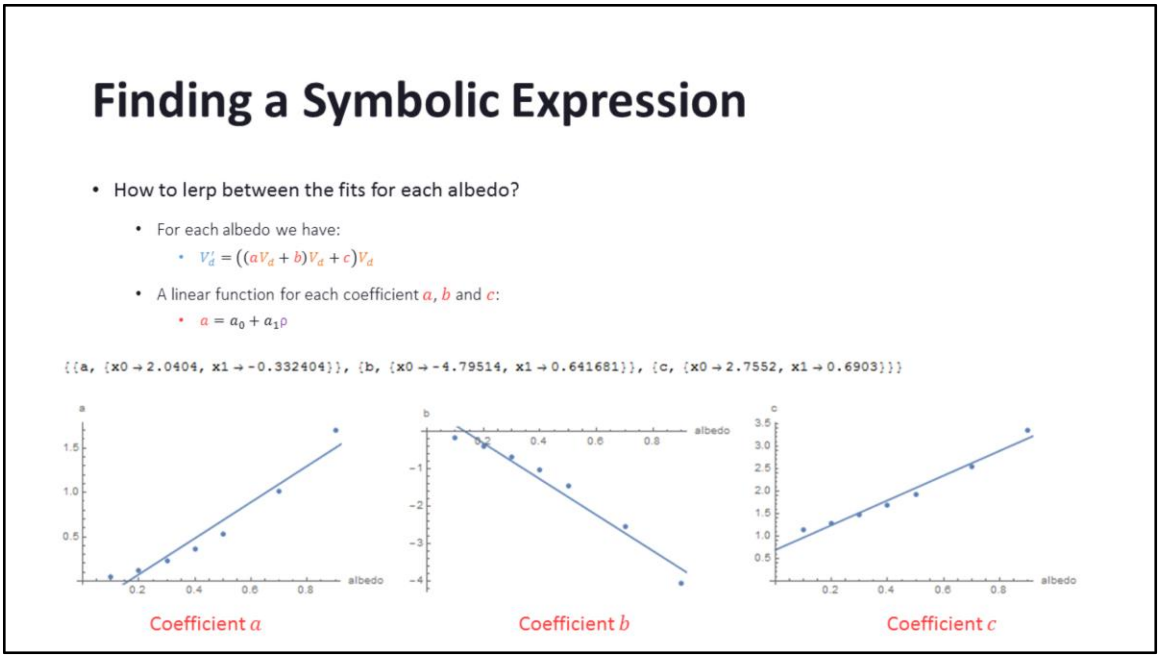
- Using all the input data jointly

a → 0.0518442	b → -0.183348	c → 1.1315
a → 0.122544	b → -0.406173	c → 1.28363
a → 0.225479	b → -0.690896	c → 1.46542
a → 0.35755	b → -1.03356	c → 1.67601
a → 0.531614	b → -1.45586	c → 1.92425
a → 1.00882	b → -2.54987	c → 2.54104
a → 1.7034	b → -4.05494	c → 3.35154



So, using all this data, we did a final fitting and obtained this.

At this point we know how to do the multibounce mapping for the seven albedos that we used for the fitting, but how to generalize to arbitrary ones?



To find out, we plotted the coefficients a, b, c of the cubic polynomial of each albedo that we fitted.

So, in these figures, you have albedo in the horizontal axis, and the coefficient value in the vertical one.

As you can see they are pretty much linear, so we approximated the polynomial coefficients as function of the albedo with linear equations.

Recap

- $V_d' = f(V_d, \rho)$

So, to recap.

We have a function that maps from single bounce AO to multi bounce AO, for a given albedo.

Recap

- $V_d' = f(V_d, \rho)$
  $= ((aV_d + b)V_d + c)V_d$

For this functions, we used a cubic polynomial.

Recap

- $V_d' = f(V_d, \rho)$
  $= \big((aV_d + b)V_d + c\big)V_d$

- $a = a_0 + a_1\rho$
- $b = b_0 + b_1\rho$
- $c = c_0 + c_1\rho$

And the coefficients a, b and c for this cubic polynomial are obtained using a linear function per coefficient.

Shader Code

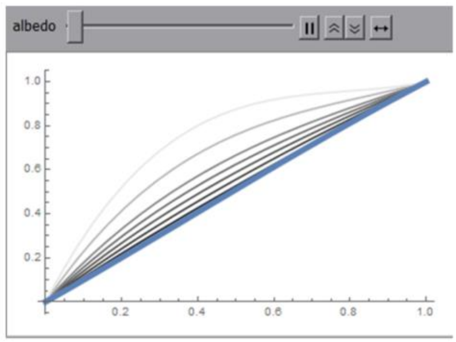
```
float3 GTAOMultiBounce( float visibility, float3 albedo )
{
	float3 a =  2.0404 * albedo - 0.3324;
	float3 b = -4.7951 * albedo + 0.6417;
	float3 c =  2.7552 * albedo + 0.6903;

	float x = visibility;
	return max( x, ( ( x * a + b ) * x + c ) * x );
}
```


And this is the resulting shader snippet.

In the end, it’s quite simple and very fast.

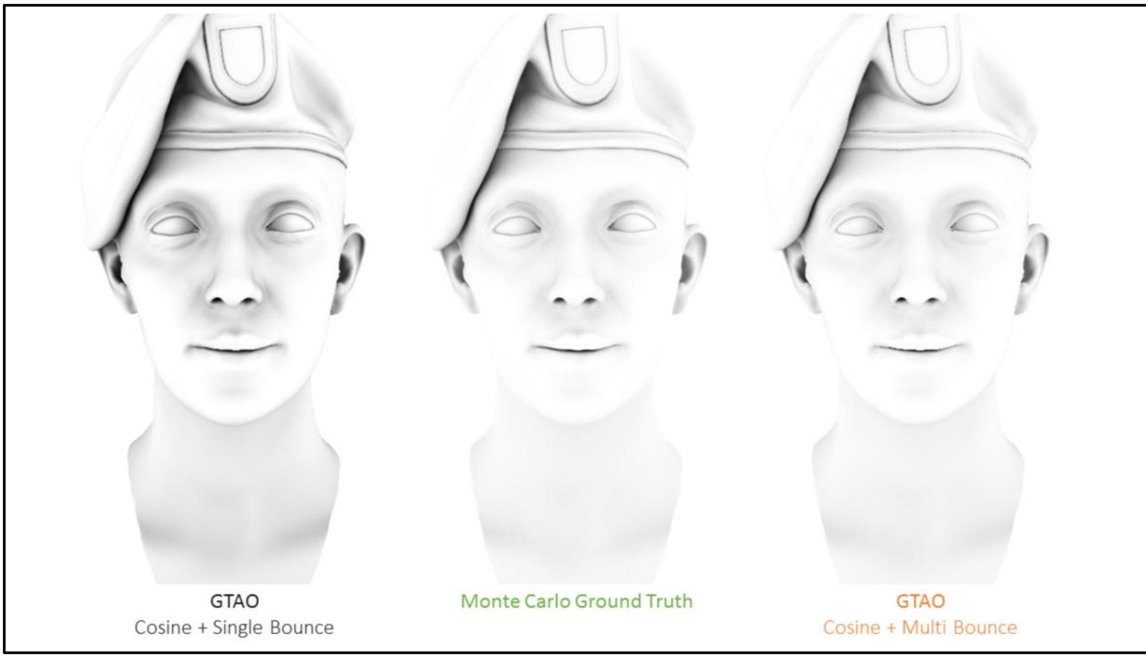
Two inputs, visibility and albedo, and single output, colored multi bounce visibility.

Finding a Symbolic Expression



So, here you have it in action.

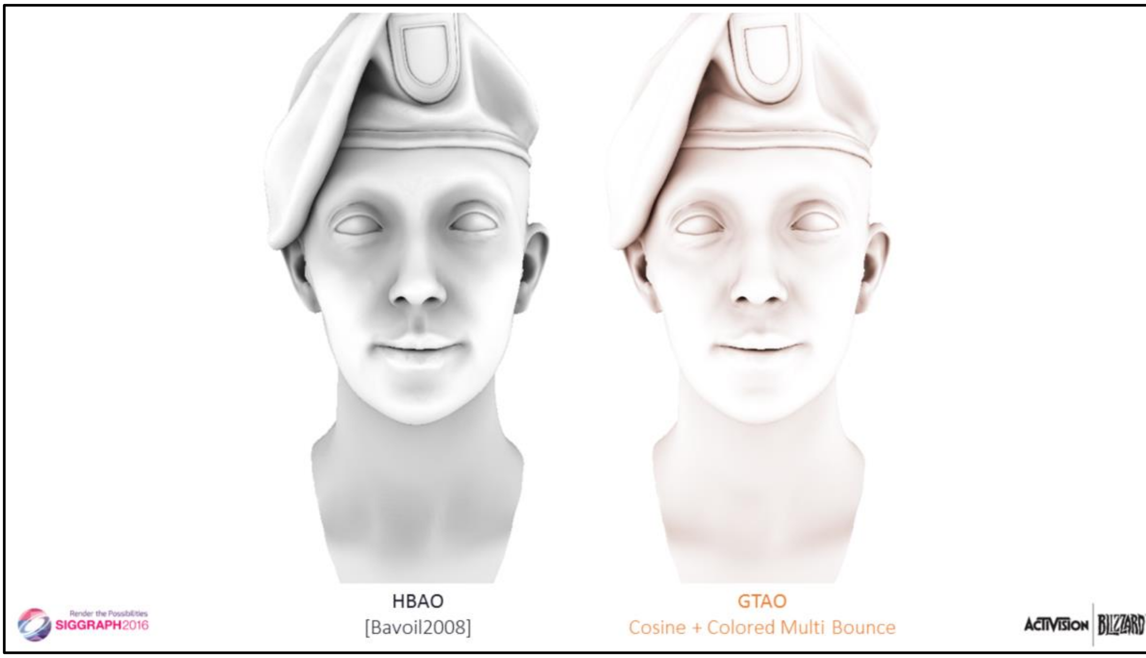
We can see how the shape of the mapping changes, as we modify the albedo.


GTAO
Cosine + Single Bounce
Monte Carlo Ground Truth
GTAO
Cosine + Multi Bounce

This slide shows a comparison between doing a single bounce on the left,

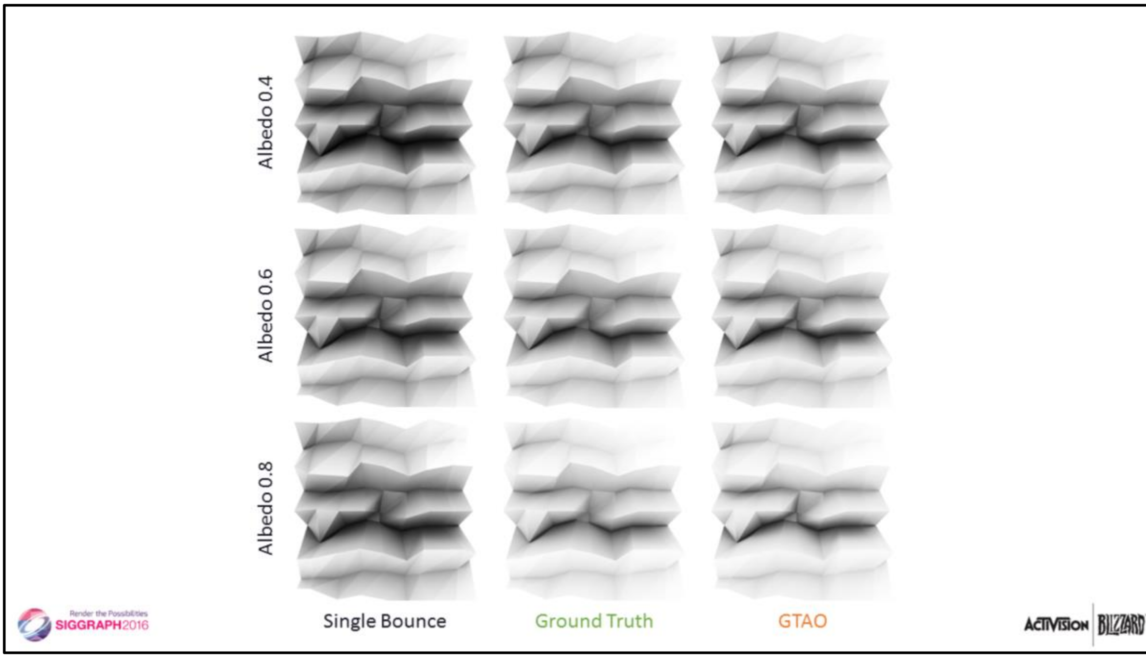
the Monte Carlo reference in the middle,

and the results obtained using our multi-bounce fitting function on the right.



To finish this part, I want to compare our starting point, on the left, with our final results, on the right.

And as you can see, considering the cosine and the multiple bounces yield a significant visual difference.

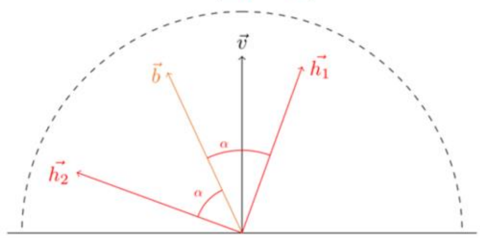


Albedo 0.4
Albedo 0.6
Albedo 0.8
Single Bounce
Ground Truth
GTAO
Render the Possibilities
SIGGRAPH2016
ACTIVISION
BLIZZARD

Bent Normal

- Average of horizon vectors:

$$\vec{b} = 0.5(\overrightarrow{h_1} + \overrightarrow{h_2})$$



- Or faster: average both horizon angles and recover 3d vector

Achieving the Performance Target

- Half resolution
- Traverse a depth mipmap chain during sampling
  - Near samples: use half-res depth
  - Further away samples: progressively use lower mips
- D3D11_FILTER_MINIMUM_*
  - Gets closest to camera height on 2x2 bilinear block
  - For small offsets: closest height ≅ max horizon
  - Improves samples per direction (4x)
    - Not ideally distributed (close together), but still useful
  - Alternative: `min(Gather4())`

Achieving the Performance Target

- Distribute outer integral (directions) over space and time
  - 1 direction per pixel
  - 16 spatial rotations in 4x4 tiles
  - 6 temporal rotations
  - $1 \times 4 \times 4 \times 6 = 96$ effective directions per pixel
- Distribute inner integral (samples within a direction) over space and time
  - 8 samples per direction
  - 4 unique spatial offsets per 4x4 tile
  - 4 temporal offsets
  - $8 \times 4 \times 4 \cong 128$ effective samples per direction
- Thanks to Michal Drobot for all the ideas and support!

GTAO Output
1 direction per pixel

We are not covering the details in this talk, but this is perhaps an important one.

So, the problem was, how we do all this in engine in 0.5ms?

Horizon-based approaches are perhaps the optimal way to calculate ground truth approximations,
they are still slower than empirical solutions.

In this budget we could only afford half resolution and 1 direction per pixel, which as you can imagine is quite noisy.

Achieving the Performance Target

- Distribute outer integral (directions) over space and time
  - 1 direction per pixel
  - 16 spatial rotations in 4x4 tiles
  - 6 temporal rotations
  - $1 \times 4 \times 4 \times 6 = 96$ effective directions per pixel
- Distribute inner integral (samples within a direction) over space and time
  - 8 samples per direction
  - 4 unique spatial offsets per 4x4 tile
  - 4 temporal offsets
  - $8 \times 4 \times 4 \cong 128$ effective samples per direction
- Thanks to Michal Drobot for all the ideas and support!

Spatial Denoiser
16 directions per pixel

Then we applied a 4x4 bilateral filter, as usual, which creates 16 directions per pixel.

It looks ok on static images, but working in half resolution it was still quite unstable.

Achieving the Performance Target

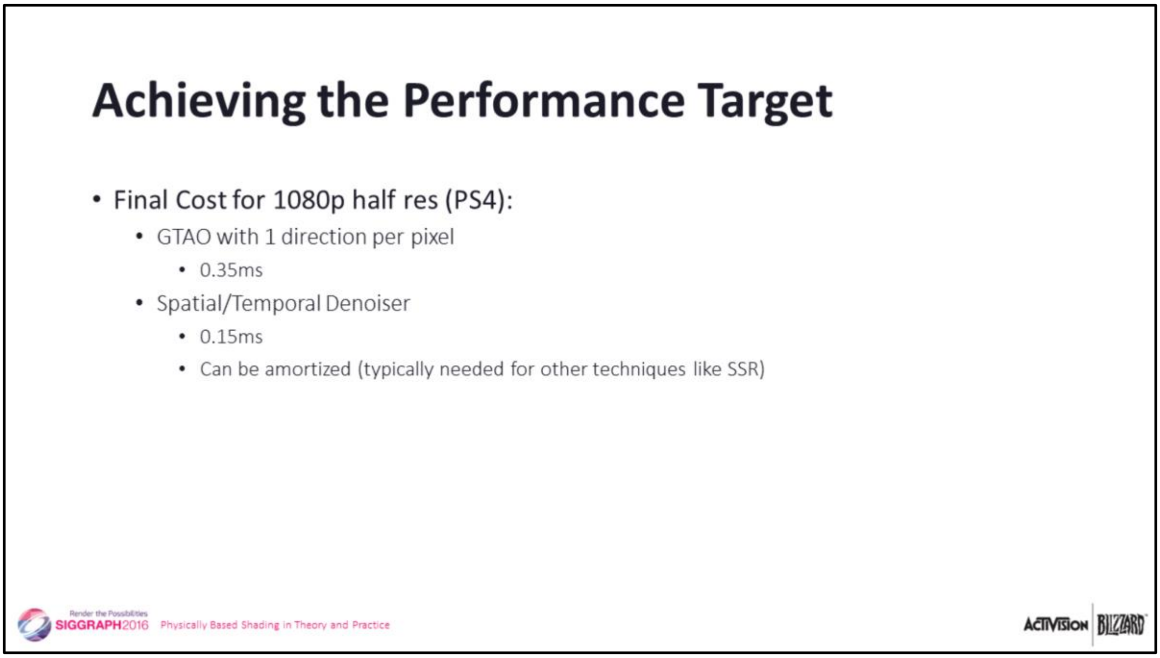
- Distribute outer integral (directions) over space and time
  - 1 direction per pixel
  - 16 spatial rotations in 4x4 tiles
  - 6 temporal rotations
  - $1 \times 4 \times 4 \times 6 = 96$ effective directions per pixel
- Distribute inner integral (samples within a direction) over space and time
  - 8 samples per direction
  - 4 unique spatial offsets per 4x4 tile
  - 4 temporal offsets
  - $8 \times 4 \times 4 \cong 128$ effective samples per direction
- Thanks to Michal Drobot for all the ideas and support!

Spatial + Temporal Denoiser
96 Directions per pixel

So, we had to heavily rely on temporal filtering to stabilize the image, and to increase the directions to 96 per pixel.

It is typical to use temporal filtering for ambient occlusion, but it is usually seen as a finisher.

It our case, we strongly rely on it, specially for improving the temporal stability.



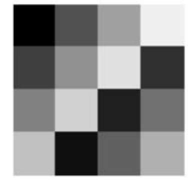
The cost of the base GTAO was around 0.35ms, and the spatial and temporal denoising 0.15ms.

Note that the denoising can be amortized if we need to denoise other half resolution images, like for example SSR ones,
as many memory accesses and calculations would be actually shared.

Noise Distribution

- Spatial distribution for directions
  - 4x4 uniform
  - Sorted in such a way that each row contains a full rotation
    - Horizontal/vertical features prominent in architecture
    - More likely for samples to be averaged horizontally/vertically by the bilateral filter
    - Sorting in rows ensures that if just a single row of the 4x4 kernel is averaged, it contains all the directions

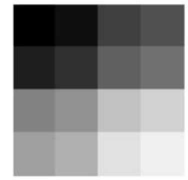
Our Approach



4x4 Tile

Noise Distribution

- Spatial distribution for directions
  - 4x4 uniform
  - Sorted in such a way that each row contains a full rotation
    - Horizontal/vertical features prominent in architecture
    - More likely for samples to be averaged horizontally/vertically by the bilateral filter
    - Sorting in rows ensures that if just a single row of the 4x4 kernel is averaged, it contains all the directions
    - Worked better than Morton order

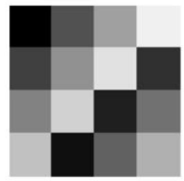


4x4 Tile

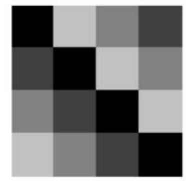
Morton Order

Noise Distribution

- Spatial distribution for offsets
  - 4 spatial values per row
  - Shifted for each row
  - Similar directions alternate 2 different offsets (observe the +xy diagonal)



4x4 Tile
(Directions)



4x4 Tile
(Offsets)

Noise Distribution

```
float noise = ( 1.0 / 16.0 ) * ( ( ( ( position.x + position.y ) & 0x3 ) << 2 ) +
                                      ( position.x & 0x3 ) );
```

Spatial Directions

```
float noise = ( 1.0 / 4.0 ) * ( ( position.y - position.x ) & 0x3 );
```

Spatial Offsets

```
float rotations[] = { 60.0f, 300.0f, 180.0f, 240.0f, 120.0f, 0.0f };
float rotation = rotations[frameCount % 6] / 360.0f;
```

Temporal Directions

```
float offsets[] = { 0.0f, 0.5f, 0.25f, 0.75f };
float offset = offsets[frameCount / 6 ) % 4];
```

Temporal Offsets

Spatial Denoiser Details

- 4x4 Bilateral filter
  - 4 AO gathers
  - 4 depth gathers
  - Can be reduced to 1 gather if packed [Drobot2014a]
- Thresholding
  - Linear depth
  - Relative soft threshold
    - Pixels with depth delta bigger than 10% of current depth don't accumulate [Bavoil2012]
    - Using a linear ramp for weighting
  - Gradient threshold
    - First derivatives

Linear Depth +
First Derivatives

Spatial Denoiser Details

- 4x4 Bilateral filter
  - 4 AO gathers
  - 4 depth gathers
  - Can be reduced to 1 gather if packed [Drobot2014a]
- Thresholding
  - Linear depth
  - Relative soft threshold
    - Pixels with depth delta bigger than 10% of current depth don't accumulate [Bavoil2012]
    - Using a linear ramp for weighting
  - Gradient threshold
    - First derivatives
    - Second derivatives (for slopes)

Linear Depth +
First + Second Derivatives

Spatial Denoiser Details

- 4x4 Bilateral filter
  - 4 AO gathers
  - 4 depth gathers
  - Can be reduced to 1 gather if packed [Drobot2014a]
- Thresholding
  - Linear depth
  - Relative soft threshold
    - Pixels with depth delta bigger than 10% of current depth don't accumulate [Bavoil2012]
    - Using a linear ramp for weighting
  - Gradient threshold
    - First derivatives
    - Second derivatives (for slopes)
- Also tried log-space depth + first derivatives
  - Linear depth + 1^{st} + 2^{nd} order derivatives yield sharper details yet smoother flat surfaces

Log Depth +
First Derivatives

Render the Possibilities SIGGRAPH2016 Physically Based Shading in Theory and Practice

ACTIVISION BLIZZARD

Temporal Denoiser Details

- Runs after the spatial denoiser
- Halfres exponential accumulation buffer
  - Running on its own, separate from temporal AA
  - Tuned for the specific problem (more aggressive)
- 4x4 bilateral filter offsets the image (not symmetric)
  - Temporal filtering will accumulate offsets and show trailing
  - Offset the 4x4 bilateral filter by 1 pixel each frame to compensate
    - Odd frames: move top/left
    - Even frames: move bottom/right
- Dynamic convergence time
  - Slower convergence for objects moving slowing
  - Faster converge for faster moving objects
  - Using proper convergence time [Jimenez2016]

Temporal Denoiser Details

- Bilinear reprojection produces bleeding of visibility on edges
  - Fetch 2x2 visibility and depth block using Gather4:
    - Current Frame Depth (`currentDepth`)
    - Previous Frame Depth (`previousDepth`)
    - Visibility (`visibilityHistory4`)
  - Test each sample in the 2x2 independently
    - Temporal bilateral filter
    - Only differences in depth in the disocclusion direction matter
      - Don't use abs
    - Weight each sample with a bilinear weight
      - If all the samples pass the depth test, same results as bilinear reprojection
    - Similar in spirit to depth testing [Jimenez2016]

```
for ( int i = 0; i < 4; i++ )
{
    float bilateralWeight = saturate( 1.0 - depthScale * ( currentDepth - previousDepth[i] ) );
    visibilityHistory += bilinearWeights[i] * lerp( visibility, visibilityHistory4[i], bilateralWeight );
}
```

Temporal Denoiser Details

- Neighborhood Clamp
  - Depth testing don't work for moving objects casting occlusion over static ones
    - Not a disocclusion case, but still creates ghosting trails
    - Neighborhood clamp mitigates the problem
  - Noisy neighborhood leads to ghosting even when using the neighborhood clamp [Jimenez2016]
  - Apply a low pass on the neighborhood
    - Fetch 4 diagonal corners with bilinear [Jimenez2016]
    - Neighborhood incorporates additional directions and reduces variance → closer to final result (less ghosting)

Temporal Denoiser Details

- Widen the neighborhood min/max window using velocity weighting
  - If velocities are very different, keep the window tight
  - Otherwise open it (unlikely to introduce ghosting)
- Similar idea to [Drobot2014b] soft clamping and color weighting
  - Allows values further than min/max to still be accepted
  - Color weighting revert to neighborhood clamp when ghosting potentially detected

Clamp Window

Velocity Weighting Fail: NeiborhoodMin NeiborhoodMax

Velocity Weighting Pass: NeiborhoodMin NeiborhoodMax

Temporal Denoiser Details

- Widen the neighborhood min/max window using velocity weighting
- Rationale:
  - If velocity weighting says reprojection is safe, we can trust it
  - However, velocity weighting gives false ghosting positives
    - Fast camera rotations, specially on the sides of the screen
  - Revert to a tight neighborhood clamp when velocity weighting detects ghosting, rather than just rejecting

Temporal Denoiser Details

- Widen the neighborhood min/max window using velocity weighting
- Shader Code:

```
float velocityWeight = VelocityWeight( currentVelocity, previousVelocity );
float window = velocityWeight * ( visibilityMax - visibilityMin );

visibilityMin -= config.clampWindowScale * window; // config.clampWindowScale = 0.5
visibilityMax += config.clampWindowScale * window;

return clamp( visibilityHistory, visibilityMin, visibilityMax );
```

Minimizing Artifacts

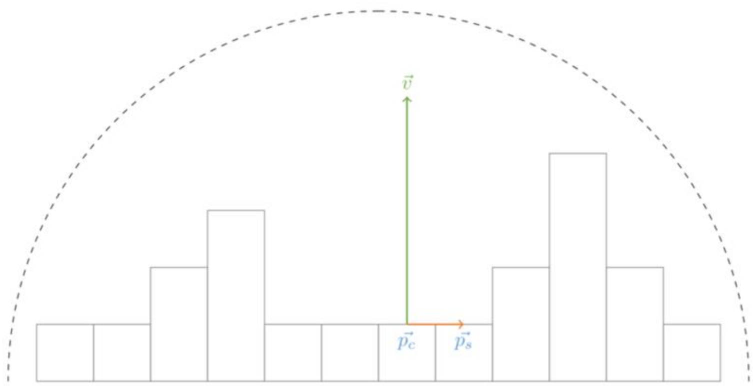
- Conservative attenuation
  - Soft-clamp integration sampling distance
  - 150 inches of full effect
  - From 150 to 200 soften sample contribution from 1 to 0
  - Ensure ground truth near occlusion
  - Far occlusion was contained in our baked lighting solution
- Screen-space radius according to the distance from camera
  - Necessary to make AO view-independent
  - Clamped to a max radius in pixels (avoids cache trashing on very near objects)
- Thickness heuristic for thin features
  - Do not trust a single layer depth buffer!

Thickness Heuristic

- Can't infer thickness from depth buffer
- Depth peeling not practical for us
- Horizon-based AO leads to thin features to cast too much occlusion

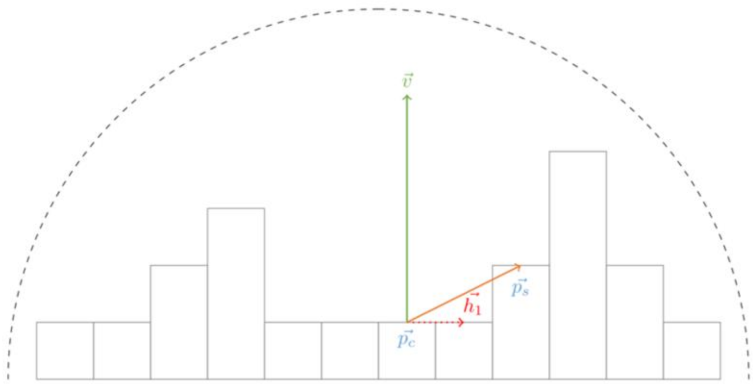
- We developed a thickness heuristic:
  - Assume object thickness similar to screen space width
  - Conservative with architectural features such as 90° corners

Thickness Heuristic



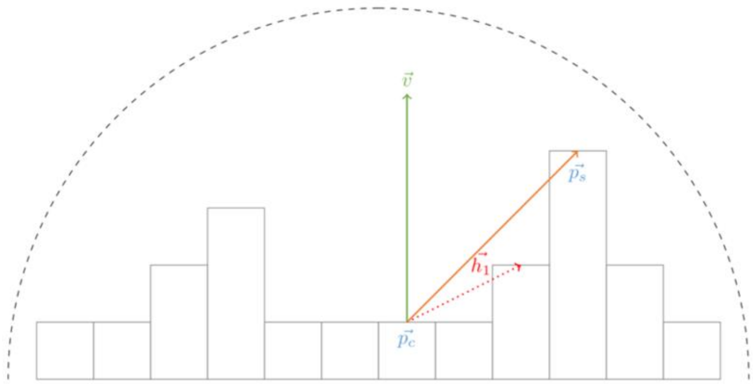
Side View (Slice)

Thickness Heuristic



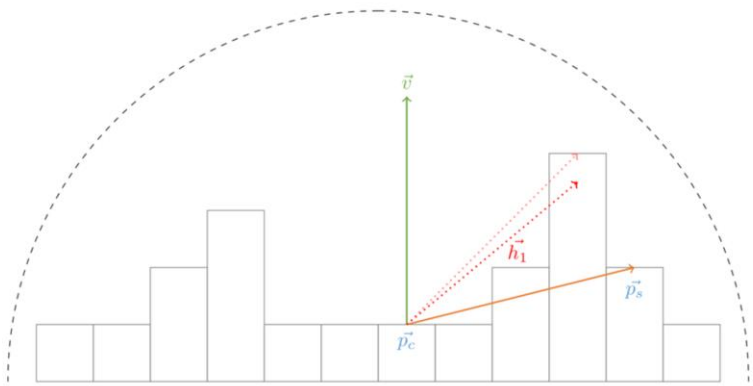
Side View (Slice)

Thickness Heuristic



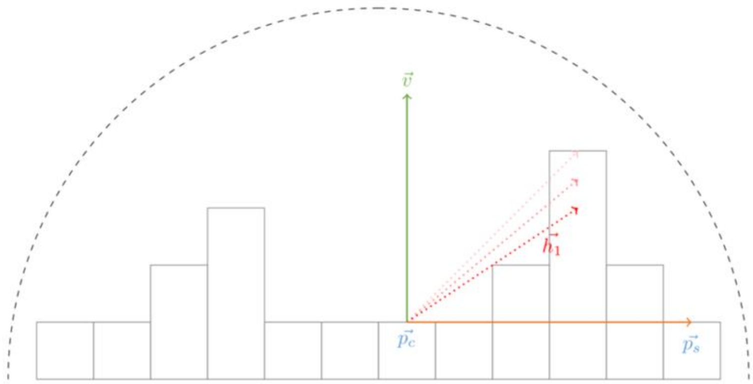
Side View (Slice)

Thickness Heuristic



Side View (Slice)

Thickness Heuristic



Side View (Slice)

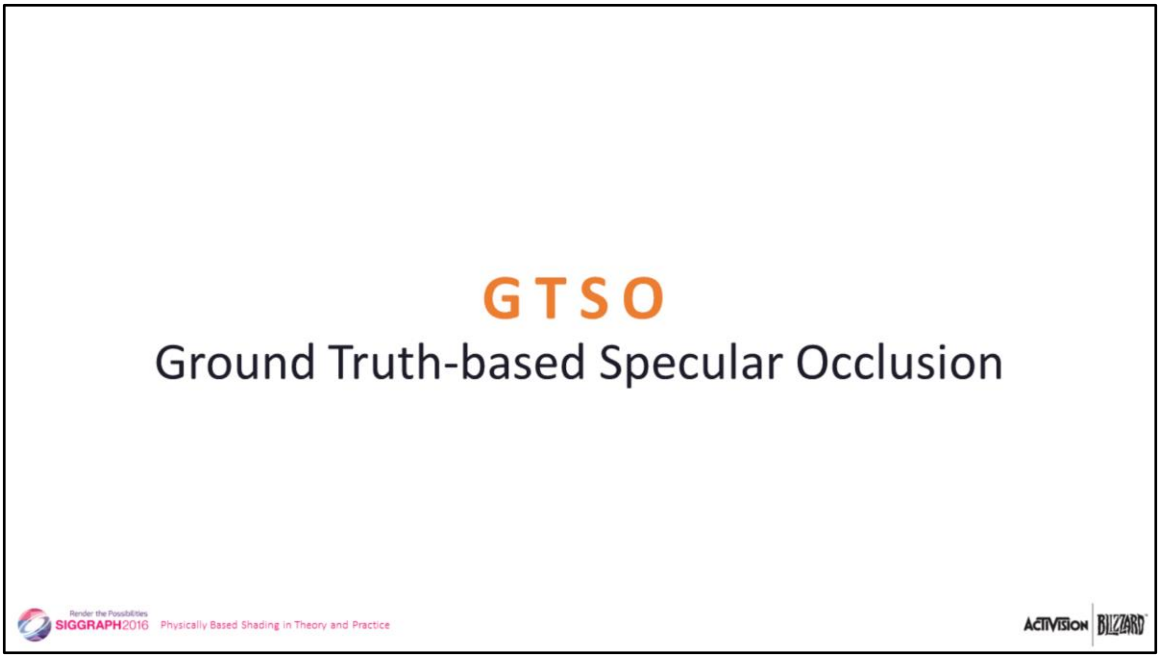
Thickness Heuristic

Without Thickness Heuristic

With Thickness Heuristic

Final Remarks

- Ideas and math presented are compatible with line sweep ambient occlusion [Timonen2013a] [Timonen2013b] [Silvennoinen2015]
  - Horizon-based integration math is the same
  - Multibounce fit is agnostic to the source of the AO as long as it is ground truth



So, we're done with ambient occlusion, and now we will dive into the details of our specular occlusion technique,
which we called GTSO.

To motivate the importance of accurate specular occlusion,
I'll start showing its importance for character rendering.

Here you have a character rendering without specular occlusion…

…and here with it [back and forth].

We think that specular occlusion is as important as ambient occlusion, but unfortunately it doesn’t receive as much attention.

$$V(\omega_i) = \begin{cases} 1 & \text{if } \omega_i \text{ hits the sky} \\ 0 & \text{if } \omega_i \text{ do not hit the sky} \end{cases}$$

Introduction

- Split integral approximation [Lazarov2013] [Karis2013]:

$$\int_\Omega L_i(\omega_i) f(\omega_i,\omega_o) cos\theta_i d\omega_i \cong \int_\Omega \underbrace{V(\omega_i) L_i^{env}}_{L_i}(\omega_i) f(\omega_i,\omega_o) cos\theta_i d\omega_i \cong \underbrace{\int_\Omega \overbrace{V(\omega_i)}^{\text{Visibility}} L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i}_{\text{Probe convolution}} \underbrace{\int_\Omega f(\omega_i,\omega_o) cos\theta_i d\omega_i}_{\text{Environment LUT}}$$

I'd like to start with a brief recap of the split integral approximation for image based lighting.

It approximates the rendering equation by splitting it to two pieces, in orange and blue.

In orange, the probe convolution,
and in blue, what is called the environment LUT.

Notice that we added a visibility term here, in green,
which is typically ignored or approximated via simple hacks.

Introduction

- We further split the visibility calculation V_s, which is our *specular occlusion*
- Can be thought as prefiltering the visibility

$$\int_\Omega V(\omega_i) L_i^{env}(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i \cong \int_\Omega^* V(\omega_i) L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_\Omega f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$

$$\cong \underbrace{\int_\Omega^* V(\omega_i) D(h) cos\theta_i d\omega_i}_{V_s} \int_\Omega^* L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_\Omega f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$

The core of our technique consists on adding a further split for visibility, in green here,
which can be seen as prefiltering the visibility.

[stop for a few seconds]

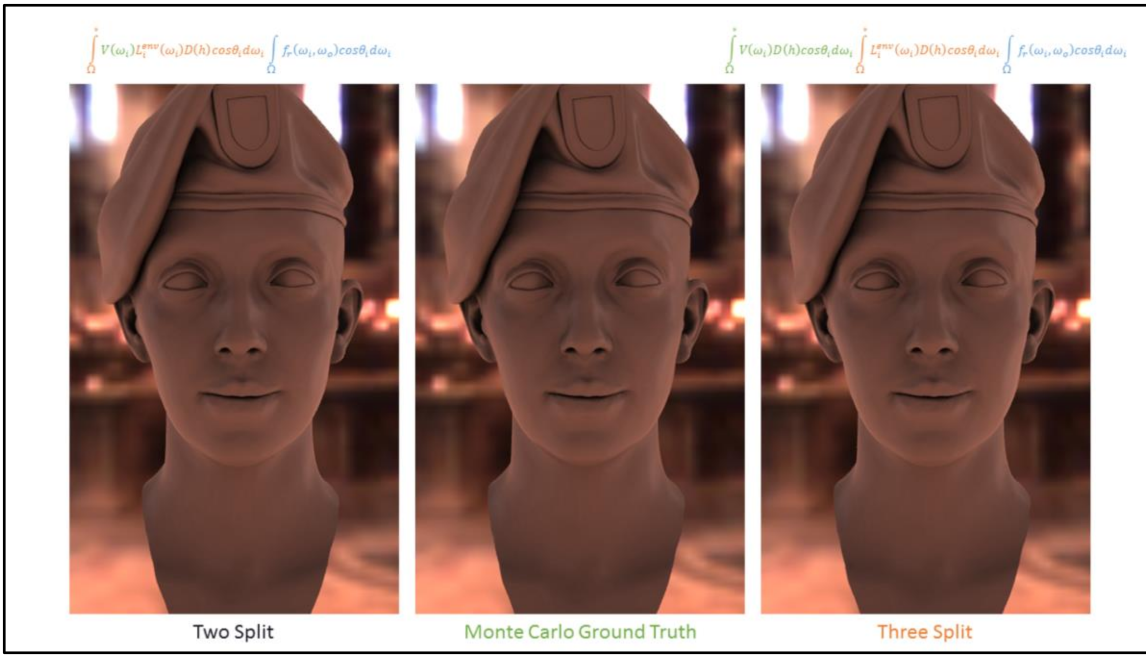
Note that the stars on the integrals…

Split Sum Term Normalization

- Star (*) means to normalize the integral
  - As in the split integral approximation previous work
- For the new visibility term:

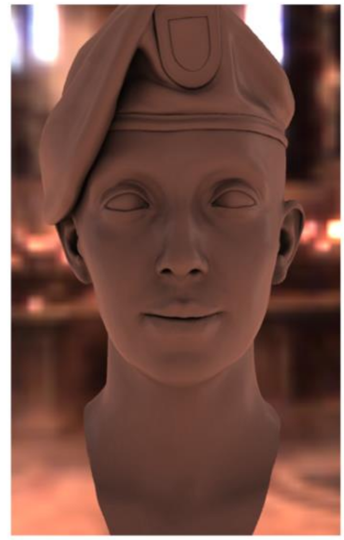
$$\int_{\Omega}^{*} V(\omega_i)D(h)cos\theta_i d\omega_i = \frac{\int_{\Omega} V(\omega_i)D(h)cos\theta_i d\omega_i}{\int_{\Omega} D(h)cos\theta_i d\omega_i}$$

…means that we normalize them, something that is also done by the previous split integral approximations.

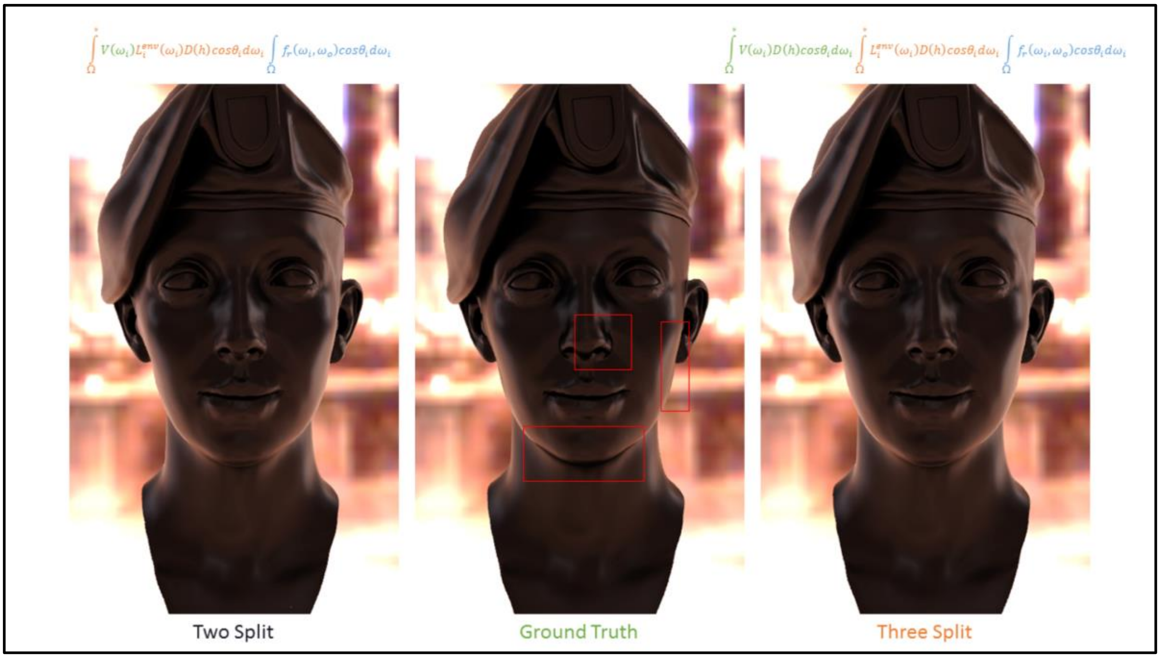


Here you have a comparison of the original two split integral on the left, and our three split on the right.

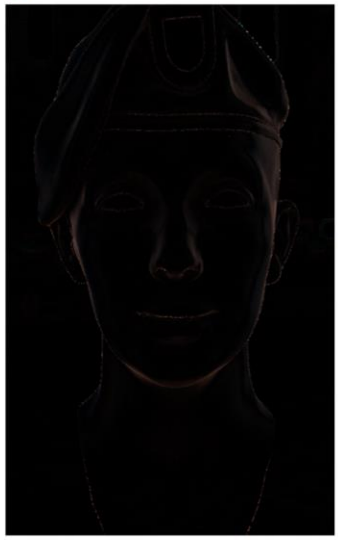
You can see how the three split approximation is still quite close to ground truth, and doesn't introduce more error than the two split one.



Monte Carlo Ground Truth

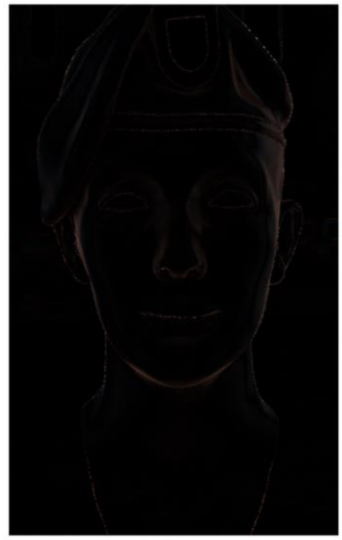

$\int_{\Omega} V(\omega_i) L_i^{env}(\omega_i) D(h) \cos\theta_i d\omega_i \int_{\Omega} f_r(\omega_i, \omega_o) \cos\theta_i d\omega_i$
$\int_{\Omega} V(\omega_i) D(h) \cos\theta_i d\omega_i \int_{\Omega} L_i^{env}(\omega_i) D(h) \cos\theta_i d\omega_i \int_{\Omega} f_r(\omega_i, \omega_o) \cos\theta_i d\omega_i$
Two Split
Ground Truth
Three Split

$$\int_{\Omega}^{*} V(\omega_i) L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_{\Omega} f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$



Two Split

$$\int_{\Omega}^{*} V(\omega_i) D(h) cos\theta_i d\omega_i \int_{\Omega}^{*} L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_{\Omega} f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$



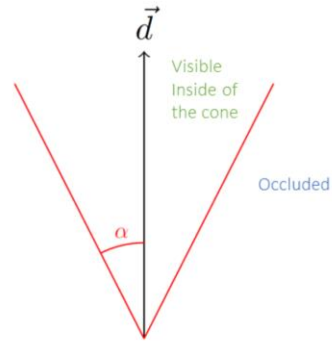
Three Split Approximation

Render the Possibilities SIGGRAPH2016

ACTIVISION BLIZZARD

First V_s Attempt

- Assumptions:
  - Visibility can be approximated as a cone
  - BRDF can be approximated as a cone
- Input:
  - Bent normal + AO
- Idea:
  - Calculate the occlusion using the intersection of the reflection and visibility cones
  - Similar to [Oat2007], but for specular lighting



Cone defined by:
Direction $\vec{d}$, Aperture α

SIGGRAPH2016 Physically Based Shading in Theory and Practice

So, let’s dive into our first specular occlusion attempt.

We made the assumption that both the visibility and the BRDF can be approximated by cones.

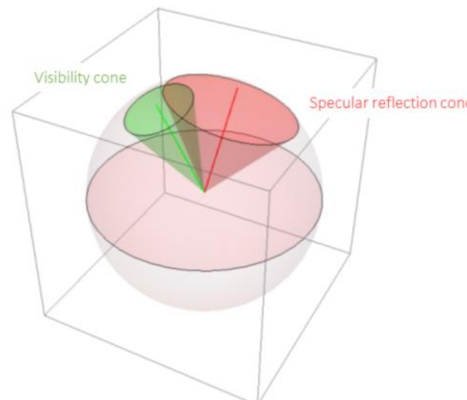
The idea is then to calculate the occlusion using the intersection of these cones.

The visibility cone can be obtained from the bent normal and occlusion values, which can either be baked or computed by GTAO.

And the reflection cone can be derived from the reflection direction and roughness.

Method

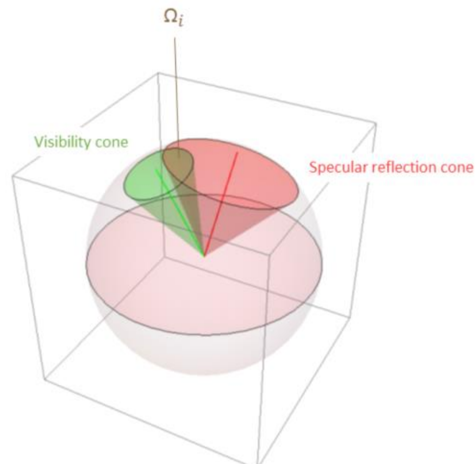
1. Calculate:
   - Visibility cone
   - Specular reflection cone



For this, we first calculate the visibility and specular cones.

Method

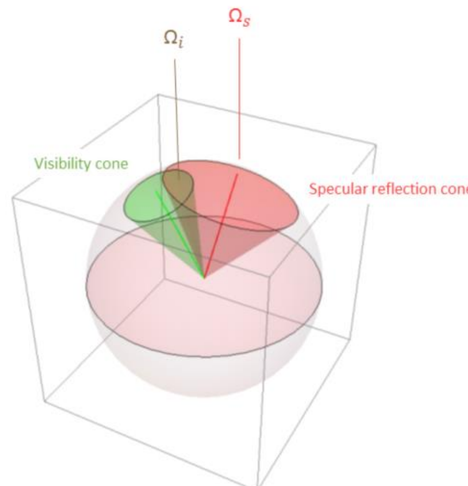
1. Calculate:
   - Visibility cone
   - Specular reflection cone
2. Calculate solid angle of the intersection Ω_i



Then we calculate the solid angle of the intersection.

Method

1. Calculate:
   - Visibility cone
   - Specular reflection cone
2. Calculate solid angle of the intersection Ω_i
3. Calculate solid angle of the specular reflection cone Ω_s

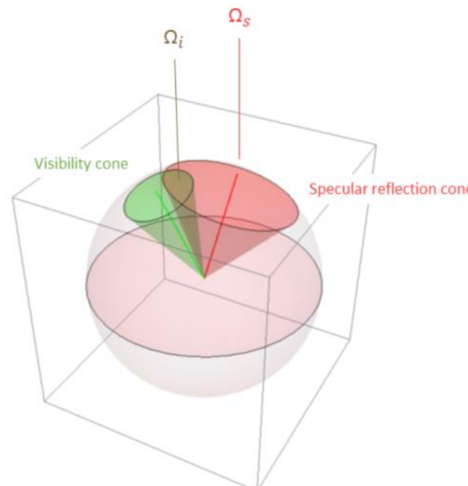


The solid angle of the reflection cone.

Method

1. Calculate:
   - Visibility cone
   - Specular reflection cone
2. Calculate solid angle of the intersection Ω_i
3. Calculate solid angle of the specular reflection cone Ω_s
4. Calculate percentage of occlusion:

$$V_s = \frac{\Omega_i}{\Omega_s}$$



And with both at hand, we can calculate the percentage of the occlusion by doing a simple ratio.

Input

Visibility cone direction — Convert to visibility aperture α_v

- Bent normal + ambient occlusion
- Reflection direction + gloss

Reflection cone direction — Convert to reflection aperture α_s

Calculating Aperture from Ambient Occlusion

- Ambient occlusion equation (uniform weighting):

$$V_d^{uniform} = \frac{1}{2\pi}\int_0^{2\pi}\int_0^{\alpha_v} \sin(\theta)\, d\theta\, d\phi = 1 - \cos(\alpha_v)$$

- Therefore:

$$\cos(\alpha_v) = 1 - V_d$$

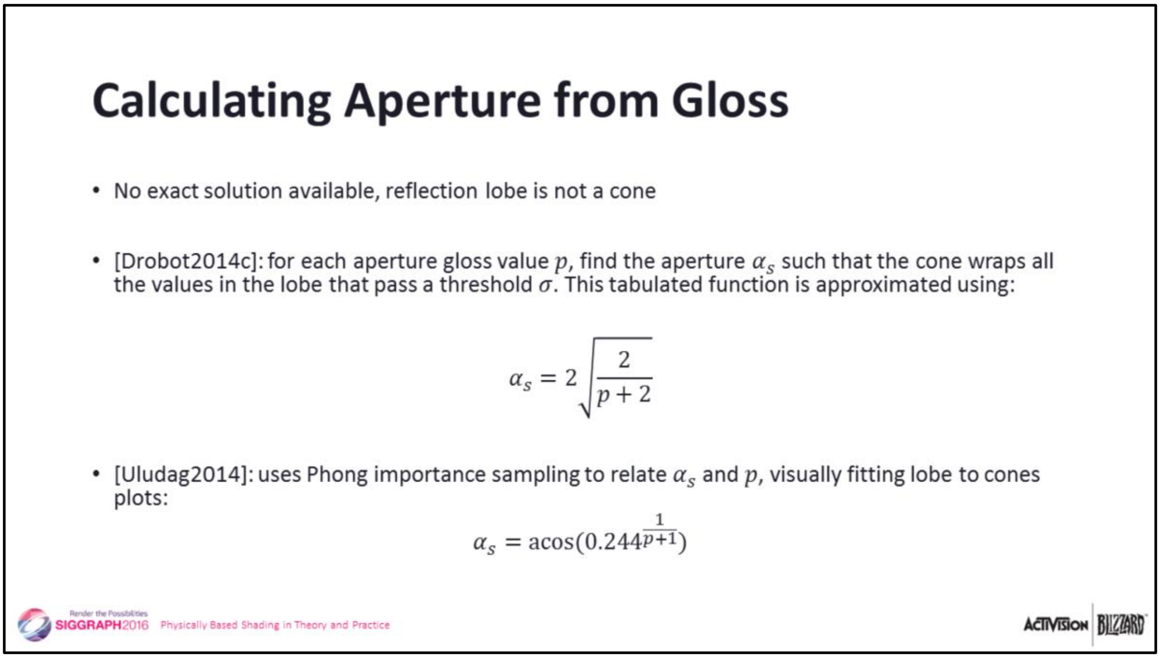
Calculating Aperture from Ambient Occlusion

- Ambient occlusion equation (cosine weighting):

$$V_d^{cosine} = \frac{1}{\pi}\int_0^{2\pi}\int_0^{\alpha_v} \cos(\theta)\sin(\theta)\, d\theta\, d\phi = 1 - \cos(\alpha_v)^2$$

- Therefore:

$$\cos(\alpha_v) = \sqrt{1 - V_d}$$



Note: [Uludag2014] appears with a typo on the article, and shows it as $\alpha_s = \cos(0.244^{\frac{1}{p+1}})$ instead.

Calculating Aperture from Gloss

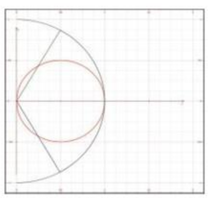
- Similar in spirit to [Uludag2014], we used Phong importance sampling equation [Walter2007] (Eq. 31) to relate them:

$$\alpha_s = \mathrm{acos}\left(u^{\frac{1}{p+2}}\right)$$

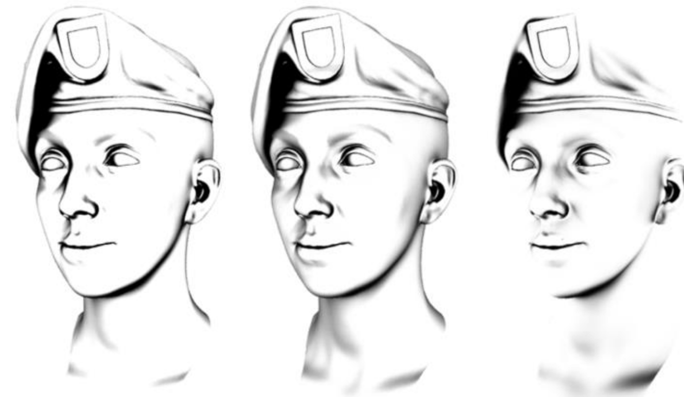
- But in contrast with [Uludag2014], we do not fit the **plots**

- Rather the variable u is optimized by comparing the resulting specular occlusion **images** with the Monte Carlo ground truth reference using DSSIM
  - We obtained an optimal value of $u = 0.01$

Working with Final Image Pixels



[Uludag2014]
Plot fitting



Ours
[Uludag2014] gloss to aperture

Ours
Our gloss to aperture fit

Phong Reference

Render the Possibilities SIGGRAPH2016

ACTIVISION BLIZZARD

Calculating Aperture From Gloss

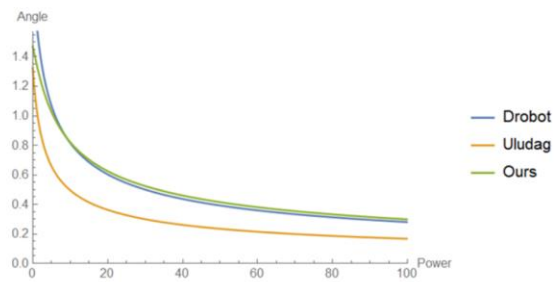
- We can use roughness r for faster evaluation:

$$r = \sqrt{\frac{2}{p+2}}$$

$$cos(\alpha_s) = 0.01^{\frac{1}{p+2}} = 0.01^{0.5r^2} = e^{-2.30259r^2} = 2^{-3.32193r^2}$$

Calculating Aperture From Gloss

- We found our approach to be very similar to [Drobot2014c], but more accurate for smaller gloss values



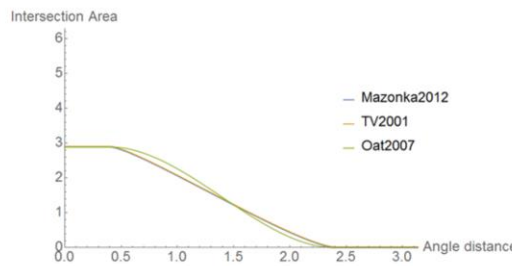
Intersection Calculation

- α_v = visibility aperture angle
- α_s = reflection aperture angle
- β = angle between bent normal and reflection direction

$$\Omega_i = Intersection(cos(\alpha_v), cos(\alpha_s), 0.5cos(\beta) + 0.5)$$

Intersection Calculation

- Accurate Options:
  - [Mazonka2012]
  - [TV01]
- Approximate One:
  - [Oat2007]
- [Mazonka2012] and [TV01]:
  - Derived differently
  - Mathematically equivalent



Final Specular Visibility Calculation

Solid angle of the intersection

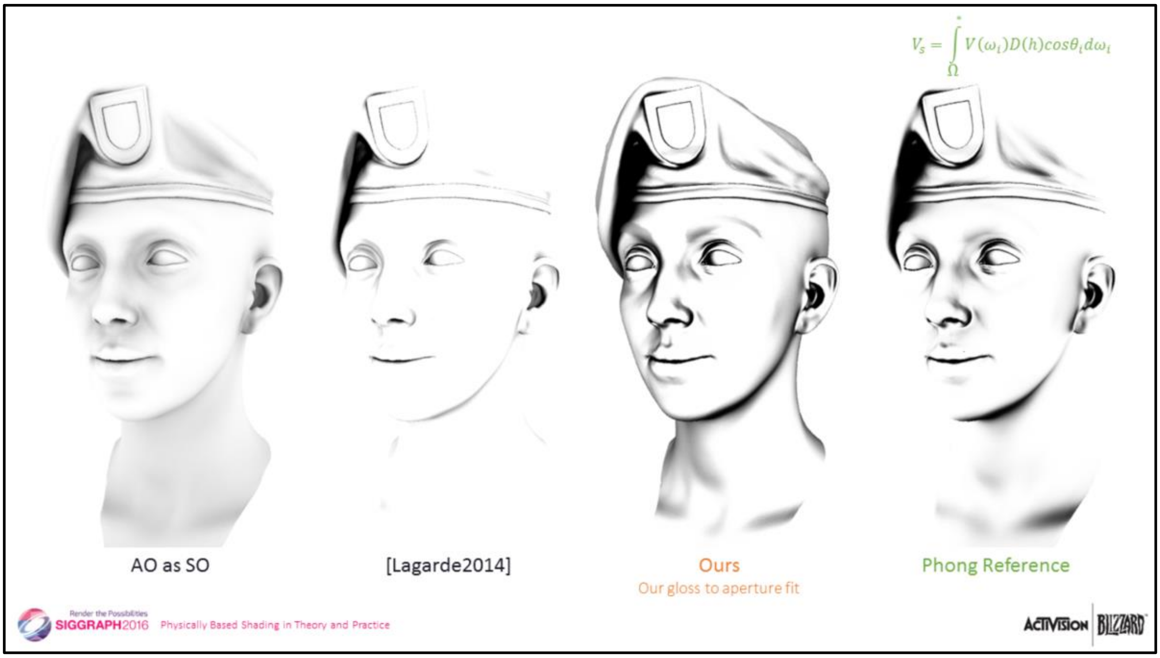
$$V_s = \frac{\Omega_i}{\Omega_s} = \frac{Intersection(\cos(\alpha_v), \cos(\alpha_s), 0.5\cos(\beta) + 0.5)}{2\pi(1 - \cos(\alpha_s))}$$

Solid angle of the reflection cone

- V_s can be baked to a 32x32x32 BC4 look up table (8-bit)
  - 16x16x16 BC4 still acceptable, quality-wise

Or in more detail, this.

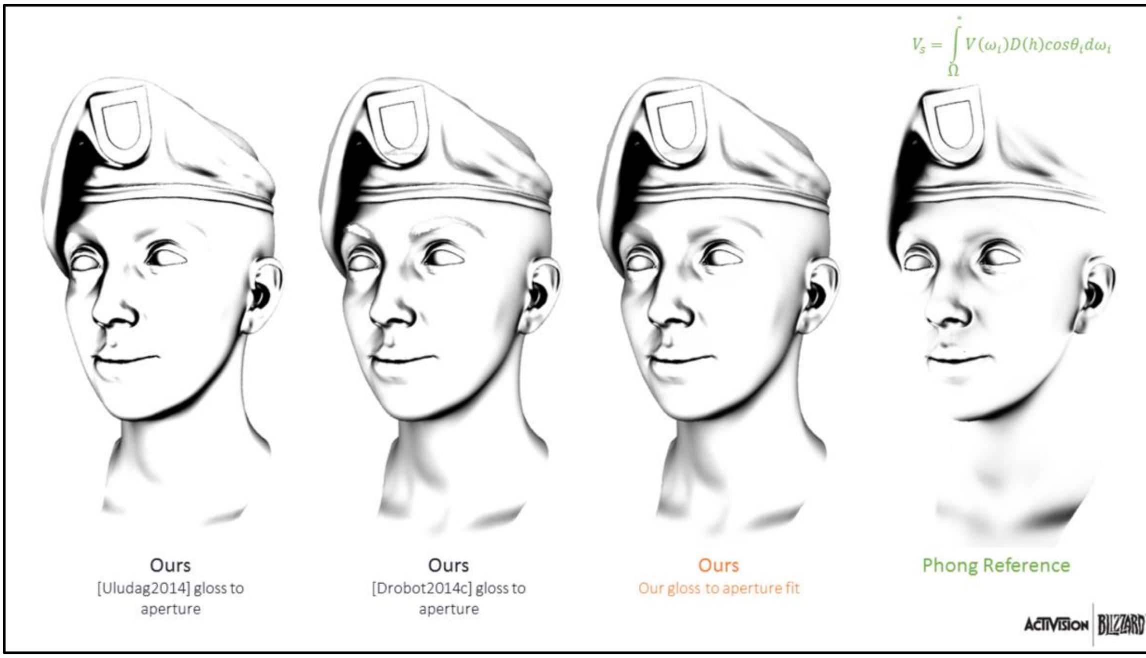
The visibility V sub s here is a function of three parameters, so we actually baked it into a 3d lookup table,
which will be important as we will se later on.



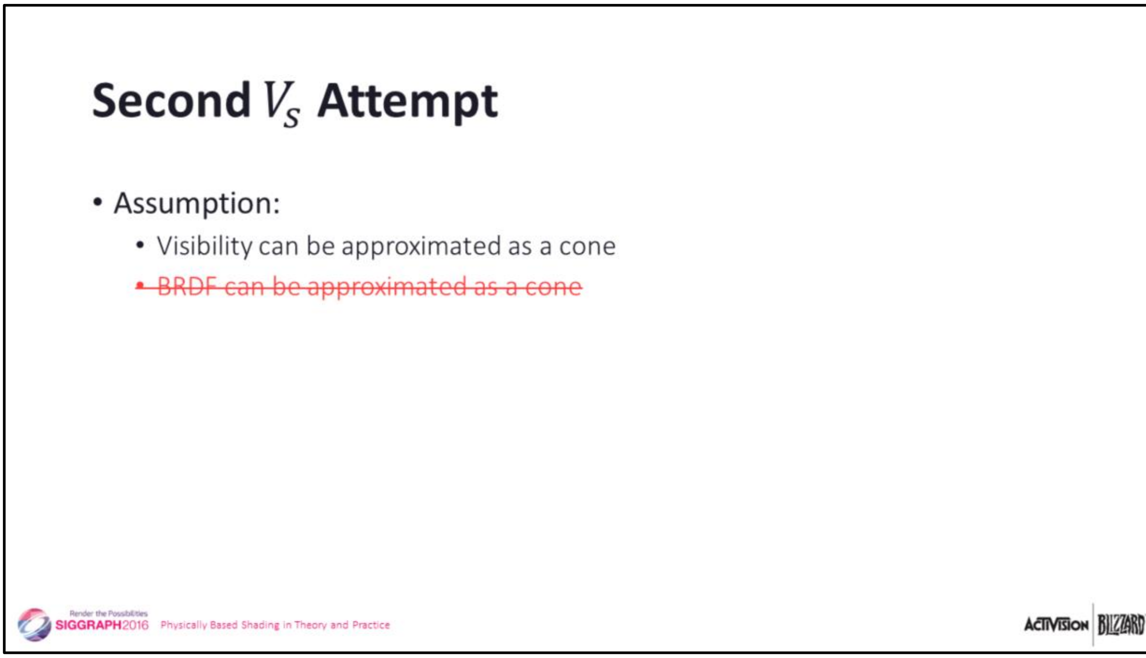
So, time for some comparisons.

On the left we are using AO as specular occlusion,
next we have [Lagarde2014] approximation,
next we have the cone/cone intersection approximation that we have just explained, in orange,
and on the right the reference.

Note that while not perfect, the cone/cone intersection technique more closely matches our reference.



$V_s = \int_{\Omega}^{+} V(\omega_i) D(h) cos\theta_i d\omega_i$
Ours
[Uludag2014] gloss to aperture
Ours
[Drobot2014c] gloss to aperture
Ours
Our gloss to aperture fit
Phong Reference



To improve on these results, we wanted to relax our assumptions.

In particular to stop approximating the reflection lobe with a cone.

Intersection with Accurate Reflection Lobes

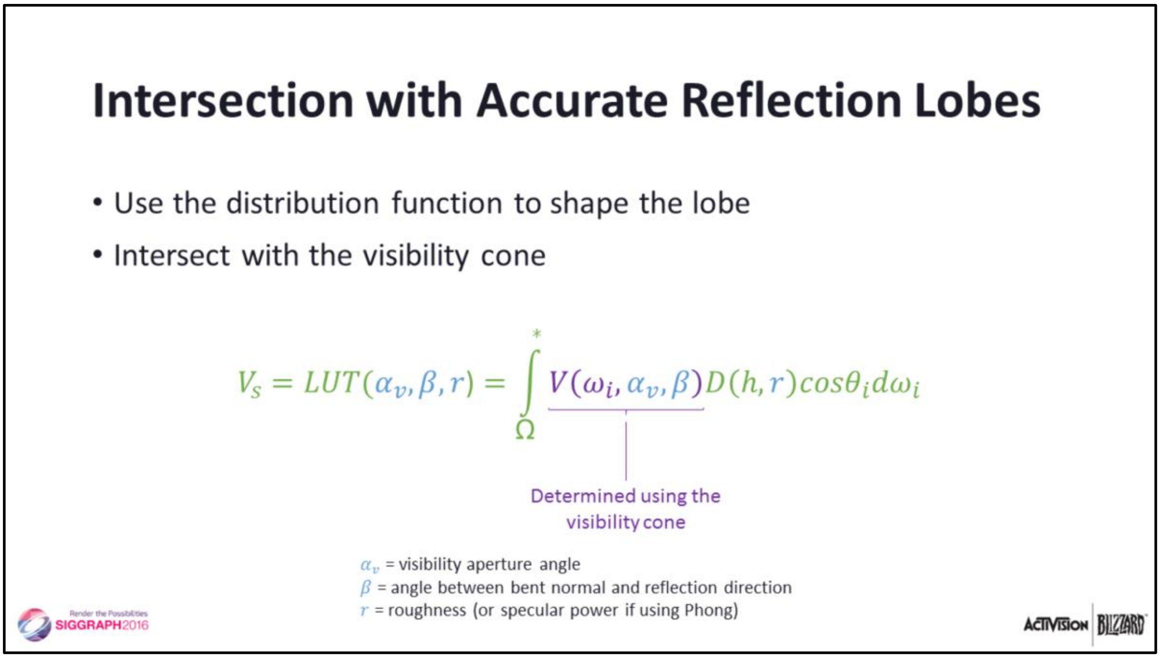
$$V_s = \frac{\Omega_i}{\Omega_s} = \frac{Intersection(\cos(\alpha_v), \cos(\alpha_s), 0.5\cos(\beta) + 0.5)}{2\pi(1 - \cos(\alpha_s))}$$

- V_s currently calculates the intersection of two cones
- V_s is baked into a LUT
- We can calculate the intersection with any other shape efficiently

- **Key Idea:**
  - Bake the intersection of the visibility cone with the actual reflection lobe

If you remember this, we baked the intersection into a look up table.

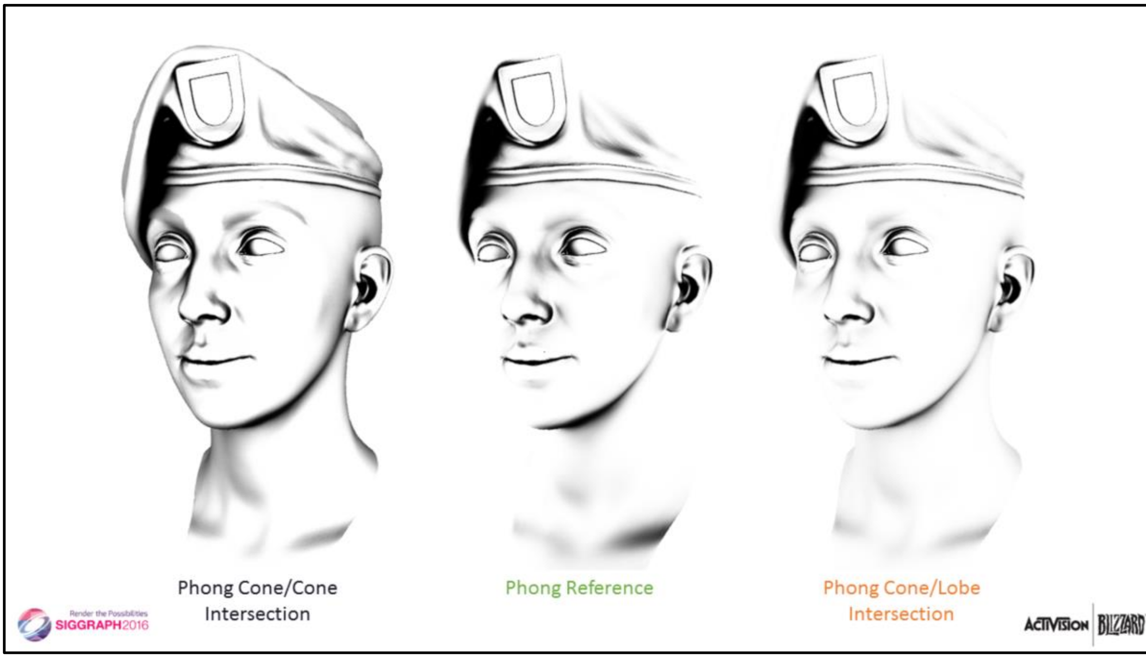
So, why we need to use a cone to represent our BRDF, when we can actually calculate the intersection with the real lobe shape offline?



What we bake into the lookup table is in particular, this.

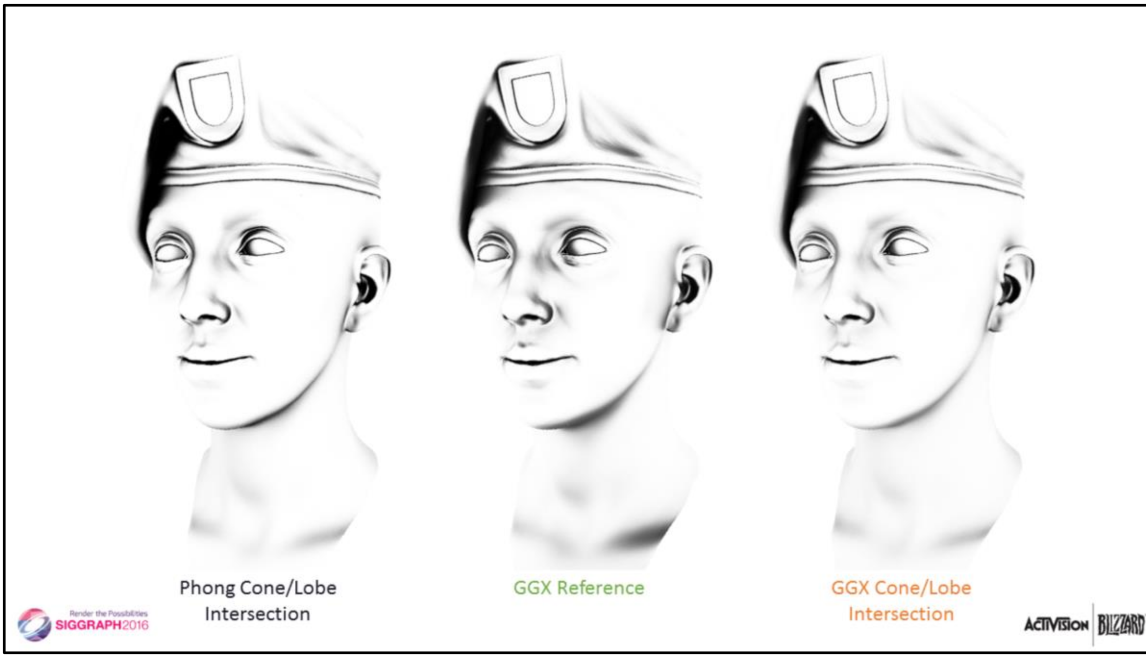
Still a 3d look up table, but with slightly different parameters, as we obviously need to pass the roughness.

The integral basically calculates the reflection lobe over the hemisphere, but masking the rays that are outside of the visibility cone.



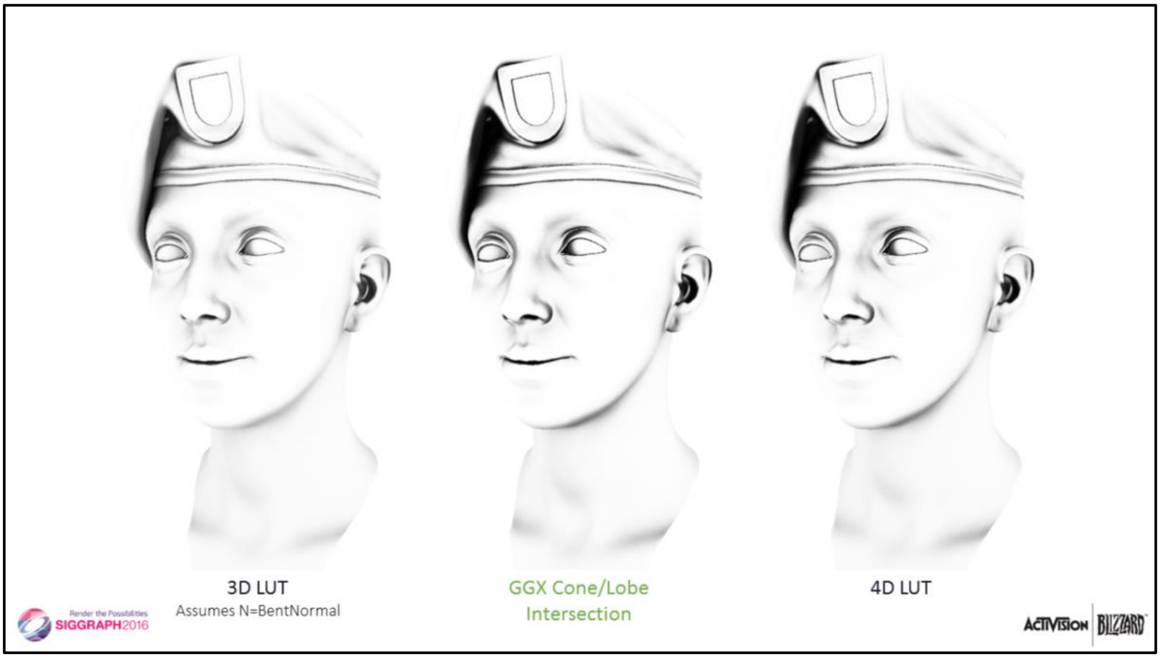
In this comparison, we can see on the left our previous cone/cone intersection,
on the middle the reference,
and on the right the new cone/lobe intersection that we have just presented.

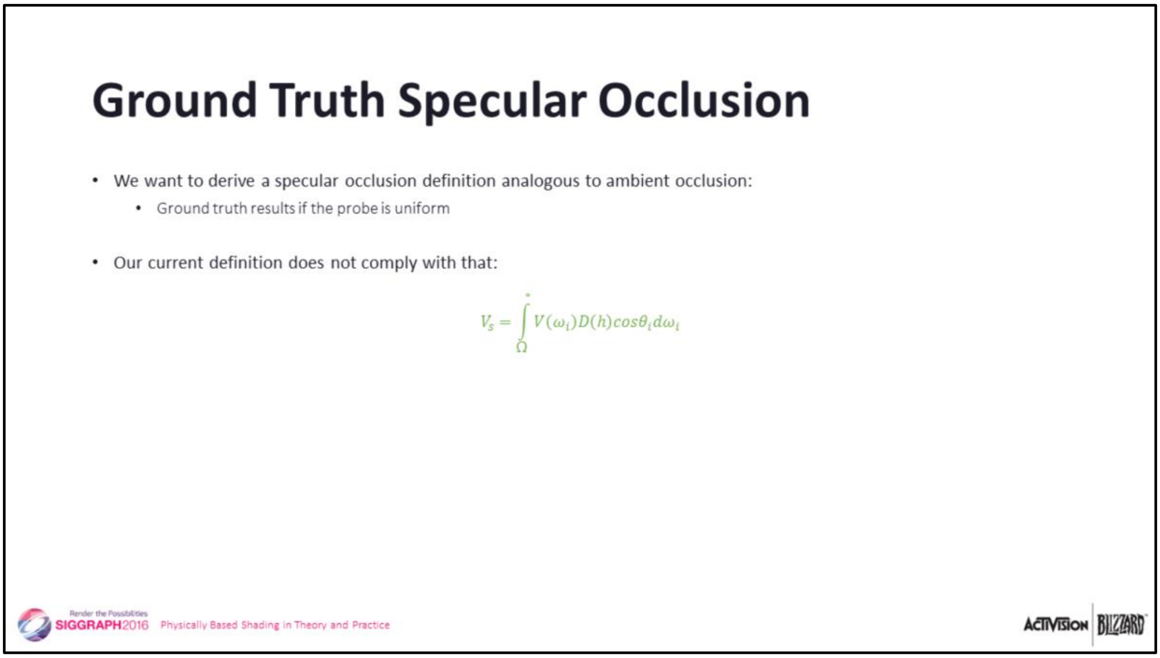
Note how it substantially improves on the overdarkening that we were getting near the silhouette of the character.



All the previous comparisons were done using Phong (as the previous work used Phong for gloss to aperture calculations), but we can now use any BRDF we want, like GGX.

So from now on, we will use GGX, which is what we used in-engine for rendering.


3D LUT
Assumes N=BentNormal
GGX Cone/Lobe Intersection
4D LUT



The next step for us, was to derive a specular occlusion definition that is analogous to ambient occlusion.

That means that it should be ground truth if the probe has a constant value, but our current definition does not comply with that.

If we include the full BRDF in the visibility term, instead of just using the distribution function…

Ground Truth Specular Occlusion

- We want to derive a specular occlusion definition analogous to ambient occlusion:
  - Ground truth results if the probe is uniform
- Our current definition does not comply with that:

$$V_s = \int_{\Omega} V(\omega_i) D(h) cos\theta_i d\omega_i$$

- However, this definition complies:

$$V_s = \int_{\Omega} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$

…and we expand the normalization factor we mentioned earlier…

Ground Truth Specular Occlusion

- We want to derive a specular occlusion definition analogous to ambient occlusion:
  - Ground truth results if the probe is uniform
- Our current definition does not comply with that:

$$V_s = \int_{\Omega}^{*} V(\omega_i) D(h) cos\theta_i d\omega_i$$

- However, this definition complies:

$$V_s = \frac{\int_{\Omega} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i}{\int_{\Omega} f_r(\omega_i, \omega_o) cos\theta_i d\omega_i}$$

...and we substitute in the rendering equation…

Ground Truth Specular Occlusion

- We want to derive a specular occlusion definition analogous to ambient occlusion:
  - Ground truth results if the probe is uniform
- Our current definition does not comply with that:

$$V_s = \int_{\Omega}^{*} V(\omega_i) D(h) cos\theta_i d\omega_i$$

- However, this definition complies:

$$V_s = \frac{\int_{\Omega} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i}{\int_{\Omega} f_r(\omega_i, \omega_o) cos\theta_i d\omega_i}$$

Cancels Out

$$\int_{\Omega} V(\omega_i) L_i^{env}(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i \cong V_s \int_{\Omega}^{*} L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_{\Omega} f_r(\omega_i, \omega_o)\, cos\theta_i d\omega_i = \int_{\Omega}^{*} L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_{\Omega} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$

You can see that the normalization denominator and the environment look up table cancel out,

and we reach to the result on the bottom right.

Let me zoom in.

Ground Truth Specular Occlusion

$$\int_{\Omega} V(\omega_i) L_i^{env}(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i \cong \int_{\Omega} L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_{\Omega} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$

Here we can see that if we replace the incoming radiance, in purple, with a white dome …

Ground Truth Specular Occlusion

$$\int_{\Omega} V(\omega_i) 1 f_r(\omega_i, \omega_o) cos\theta_i d\omega_i = \int_{\Omega} 1 D(h) cos\theta_i d\omega_i \int_{\Omega} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$

Render the Possibilities SIGGRAPH2016 Physically Based Shading in Theory and Practice

…the orange part will be completely gone, as it integrates to 1…

Ground Truth Specular Occlusion

$$\int_{\Omega} V(\omega_i) f_r(\omega_i, \omega_i) cos\theta_i d\omega_i = \int_{\Omega} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$

…so we reach an equality, rather than an approximation, for the case of a white dome.

Ground Truth Specular Occlusion

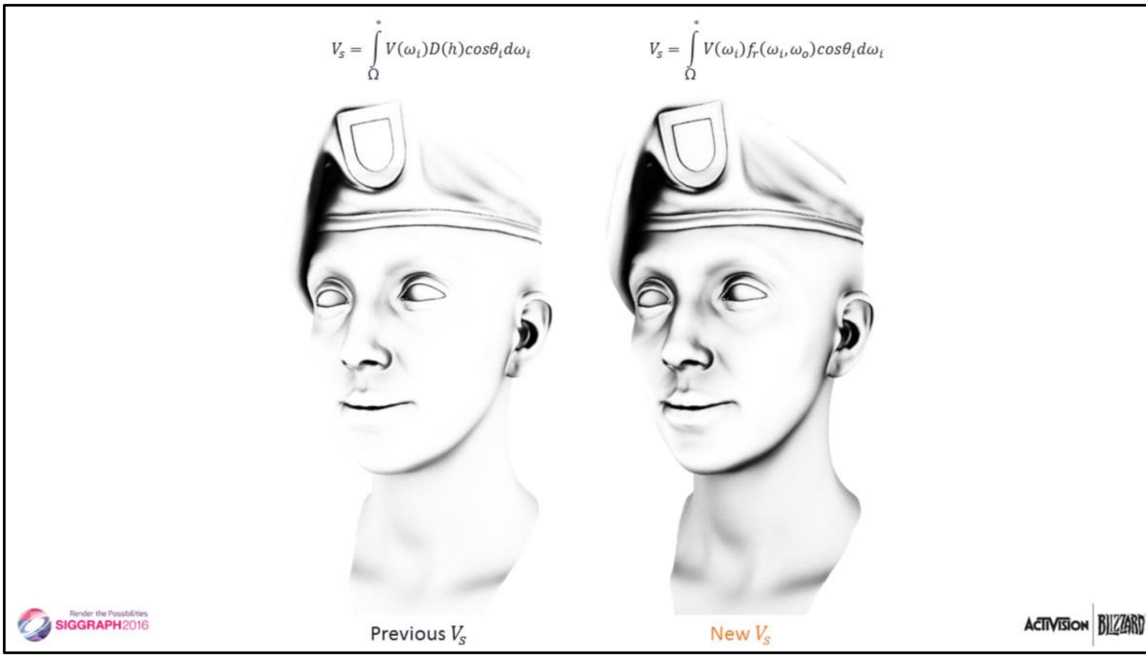
- If we compare the original split approximation (our starting point):

$$\int_{\Omega} V(\omega_i) L_i^{env}(\omega_i) f(\omega_i, \omega_o) cos\theta_i d\omega_i \cong \int_{\Omega}^{*} V(\omega_i) L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_{\Omega} f(\omega_i, \omega_o) cos\theta_i d\omega_i$$

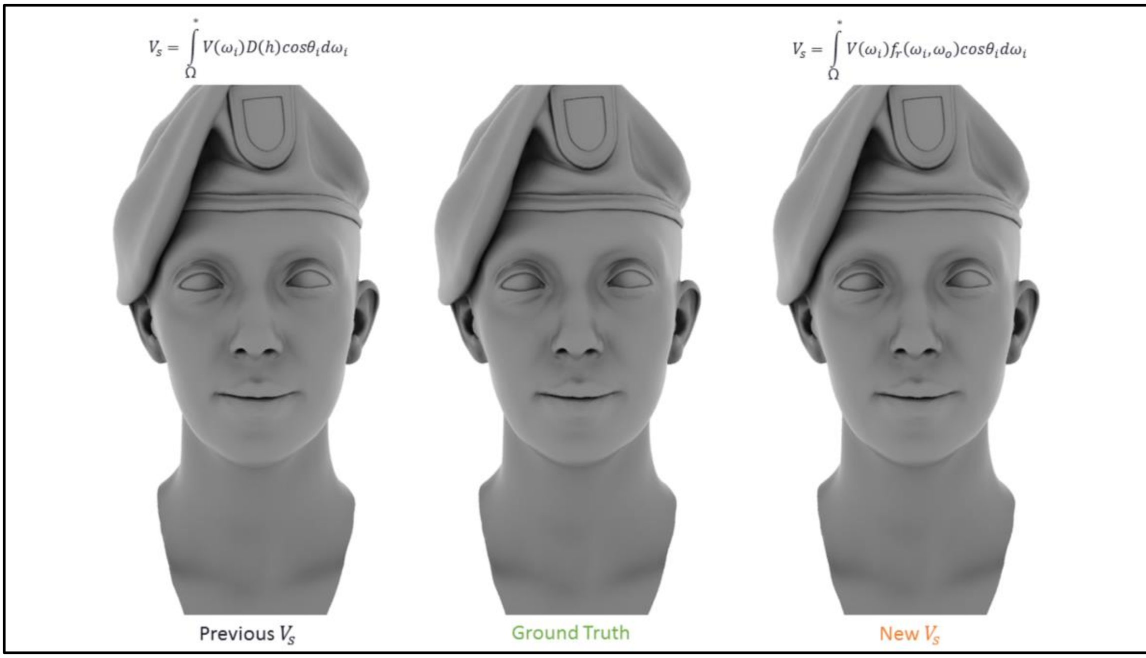
- With the final formulation results from previous slide:

$$\int_{\Omega} V(\omega_i) L_i^{env}(\omega_i) f(\omega_i, \omega_o) cos\theta_i d\omega_i \cong \int_{\Omega}^{*} L_i^{env}(\omega_i) D(h) cos\theta_i d\omega_i \int_{\Omega} V(\omega_i) f(\omega_i, \omega_o) cos\theta_i d\omega_i$$

- Notice that the only difference is that $V(\omega_i)$ moved to the right integral



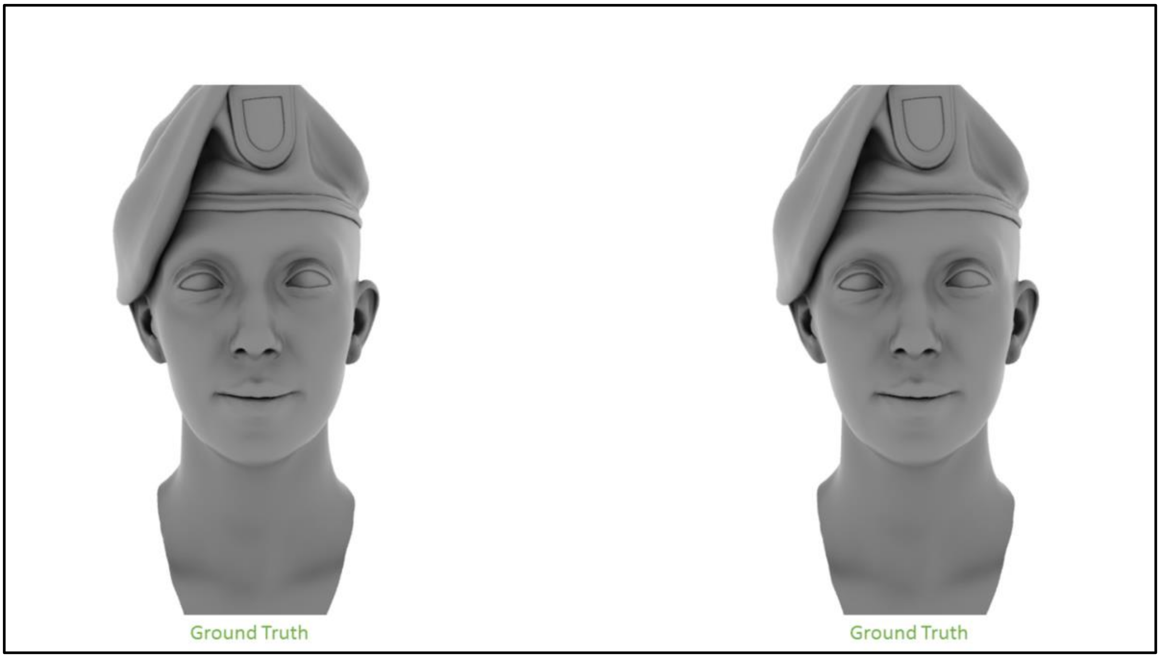
This is how this new formulation looks like, when compared with our previous one.



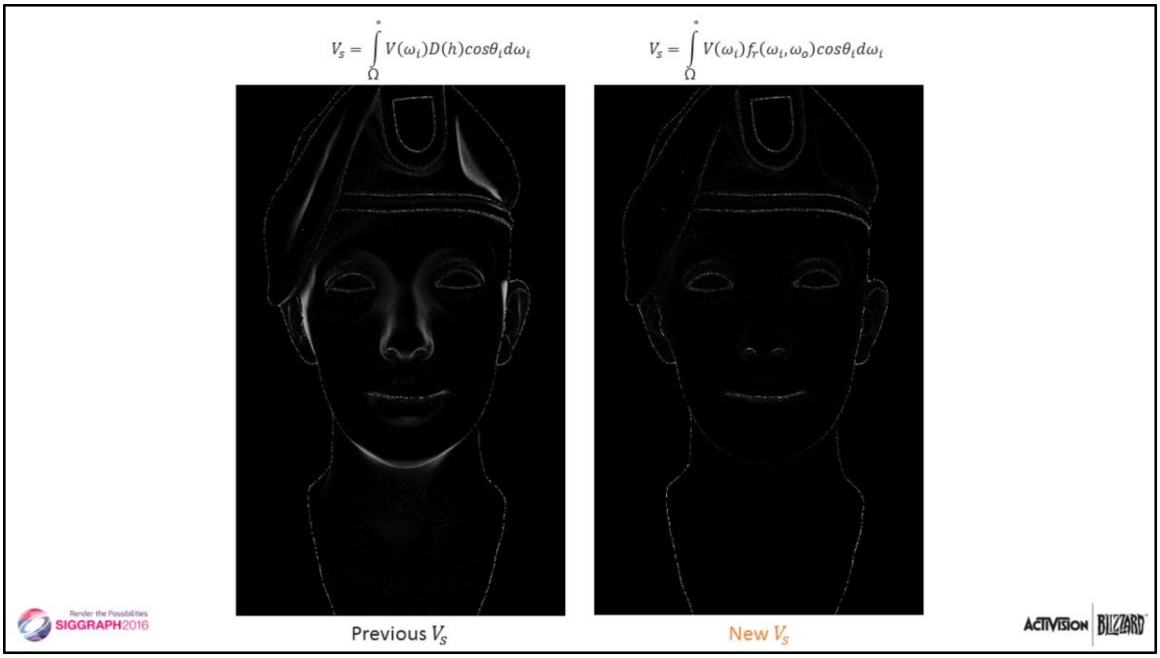
And here, you have some final renders for a white dome.

With our old formulation, on the left,
ground truth, in the middle,
and the new formulation on the right.

To better see that it completely matches the ground truth, lets flip the images back an forth.



The new approach on the right is completely equivalent to the ground truth.



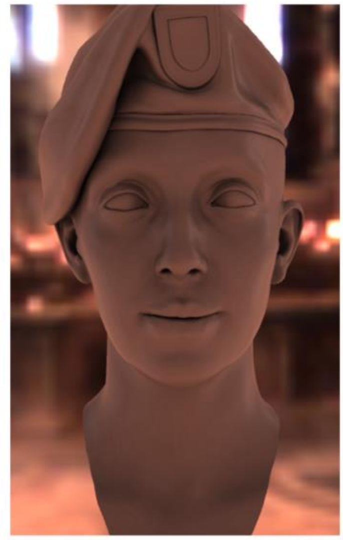
Note: differences are due to aliasing differences, given that we are using Monte Carlo for these renders.

$$V_s = \int_{\Omega}^{*} V(\omega_i) D(h) cos\theta_i d\omega_i$$

$$V_s = \int_{\Omega}^{*} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$

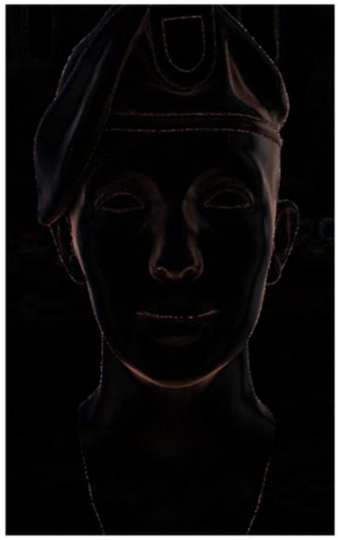
Previous V_s | Monte Carlo Ground Truth | New V_s


Monte Carlo Ground Truth

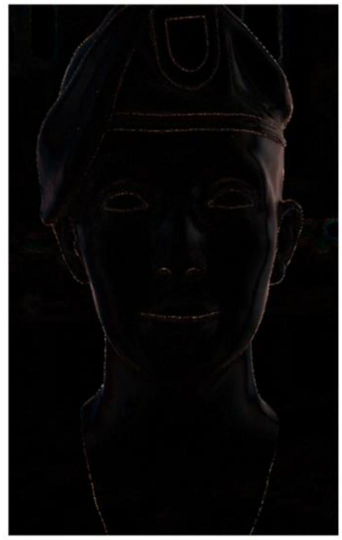

Monte Carlo Ground Truth

$$V_s = \int_{\Omega}^{*} V(\omega_i) D(h) cos\theta_i d\omega_i$$

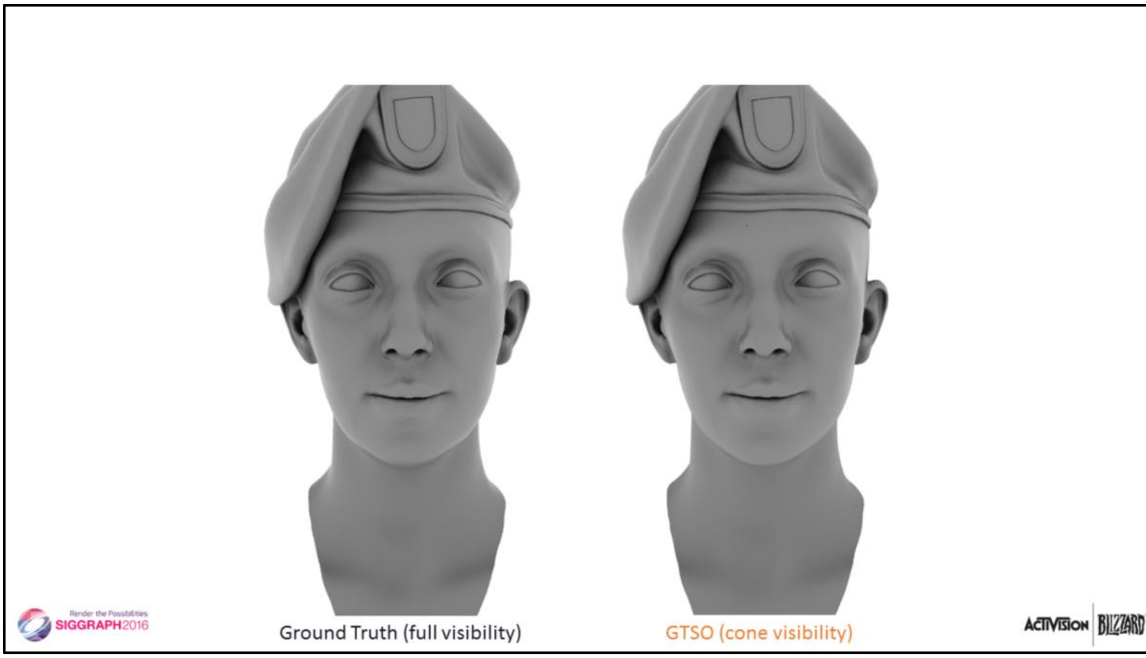
$$V_s = \int_{\Omega}^{*} V(\omega_i) f_r(\omega_i, \omega_o) cos\theta_i d\omega_i$$



Previous V_s

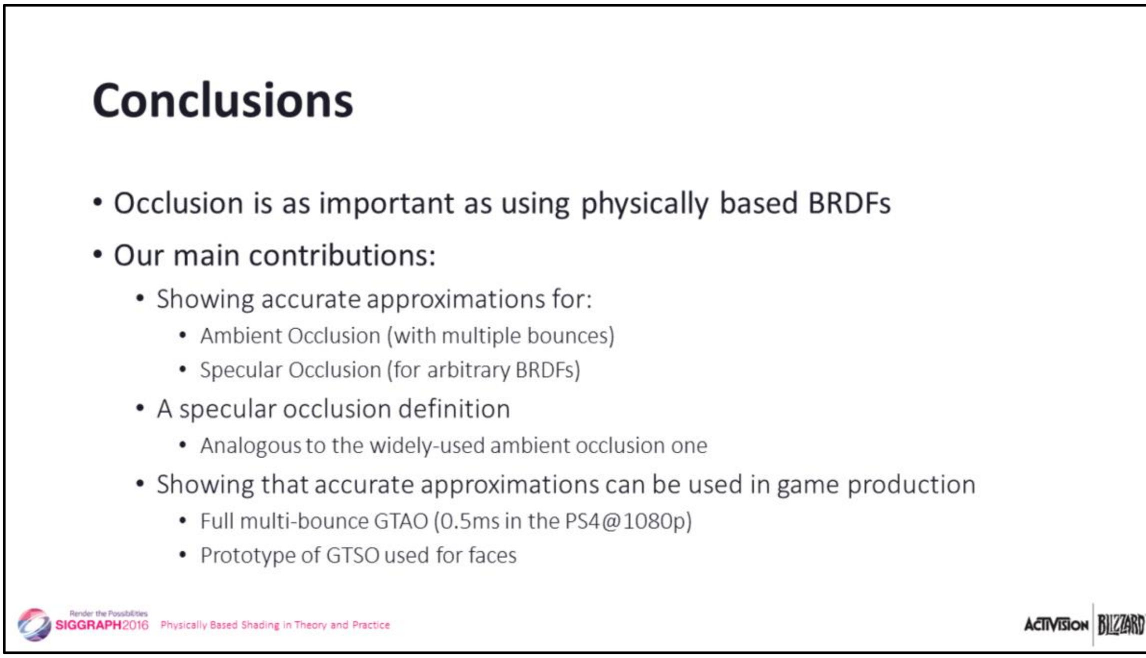


New V_s



The previous comparisons used the real visibility, on the interest of showing the match with the ground truth for a white dome.

But as you can see here, using a cone visibility is still a very good match to the ground truth.

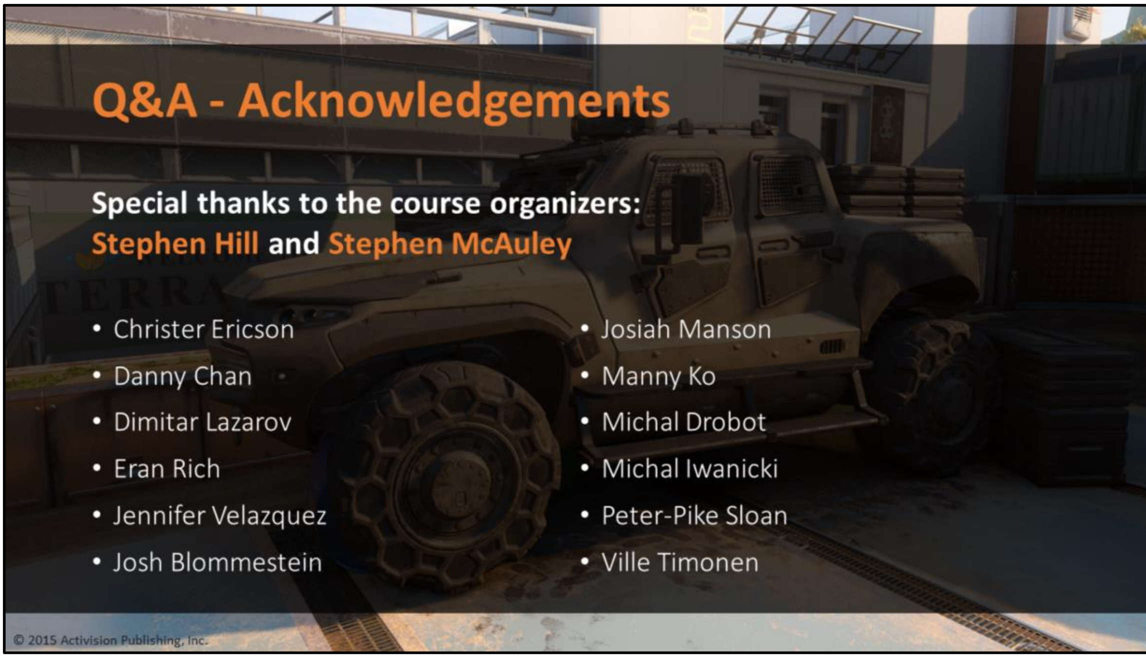


Going to the conclusions, I'd like to remark that, in our opinion, occlusion is as important as using physically based BRDFs,
if the goal is to achieve correct and photorealistic results.

To recap:
Our first contribution was to derive accurate approximations for both ambient and specular occlusion,
without constraints on the number of bounces for ambient occlusion,
nor on the BRDF we use for specular occlusion, given that it is actually baked.

The second contribution was to define an equation for specular occlusion that is analogous to the ambient occlusion one, meaning that it yields ground truth results when using white domes.

And finally, as our techniques have been employed in production under strict performance budgets,
we have shown that we have less reasons now,
to employ inaccurate hacks for indirect occlusion in modern hardware.



So, this ends our presentation, I hope you liked it, and please do not hesitate to make any questions after the session is finished.

References & Relevant Work

- [Aalund2013] A Comparative Study of Screen-Space Ambient Occlusion Methods
- [Bavoil2008] Image-Space Horizon-Based Ambient Occlusion
- [Bavoil2012] Stable SSAO in BF3 with STF
- [Bavoil2014] Deinterleaved Texturing for Cache-Efficient Interleaved Sampling
- [Drobot2014a] Low Level Optimizations for GCN
- [Drobot2014b] Hybrid Reconstruction Anti-Aliasing
- [Drobot2014c] GPU Pro5 Physically Based Area Lights
- [DSSIM] https://en.wikipedia.org/wiki/Structural_similarity
- [Filion2008] StarCraft II Effects & Techniques
- [Jimenez2013] Next-Generation Character Rendering
- [Jimenez2016] Filmic SMAA: Sharp Morphological and Temporal Antialiasing
- [Lagarde2014] Moving Frostbite to Physically based rendering
- [Landis2002] Production-ready global illumination
- [Mattausch2010] High Quality Screen Space Ambient Occlusion using Temporal Coherence
- [Mazonka2012] Solid Angle of Conical Surfaces, Polyhedral Cones, and Intersecting Spherical Caps

References & Relevant Work

- [McGuire2011] The alchemy screen-space ambient obscurance algorithm
- [McGuire2012] Scalable Ambient Obscurance
- [Mittring2007] Finding next gen: Cryengine 2
- [Mittring2012] The technology behind the unreal engine 4 elemental demo
- [Oat2007] Ambient Aperture Lighting
- [Silvennoinen2015] Multi-Scale Global Illumination in Quantum Break
- [Sloan2010] Volumetric obscurance
- [Timonen 2010] Scalable Height Field Self-Shadowing
- [Timonen2013a] Line-Sweep Ambient Obscurance
- [Timonen2013b] Screen-Space Far-Field Ambient Obscurance
- [TV2001] How common is the funnel-like energy landscape in protein-protein interactions?
- [Uludag2014] GPU Pro5 Hi-Z Screen-Space Cone-Traced Reflections
- [Walter2007] Microfacet Models for Refraction through Rough Surfaces
- [Wright2015] Dynamic Occlusion with Signed Distance Fields