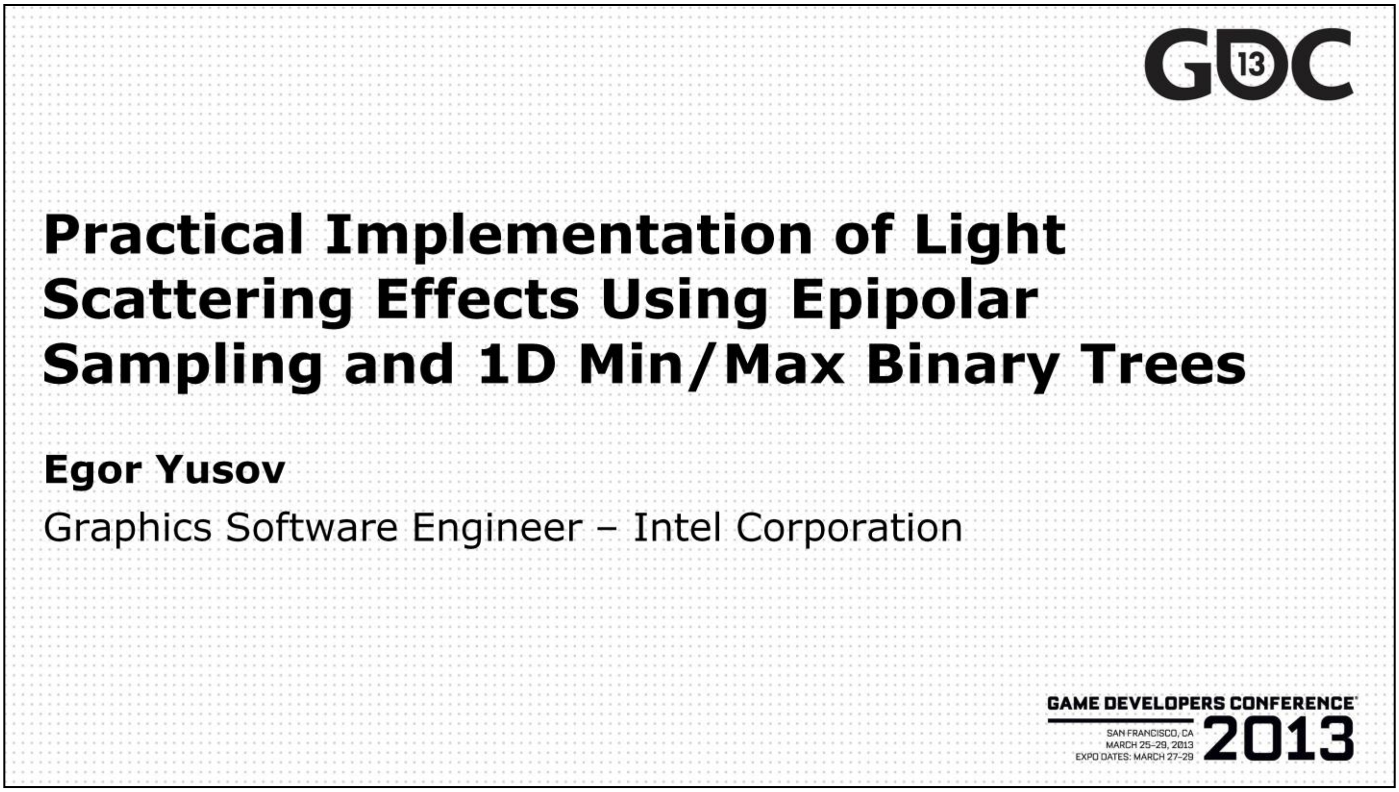


Practical Implementation of Light Scattering Effects Using Epipolar Sampling and 1D Min/Max Binary Trees

Egor Yusov

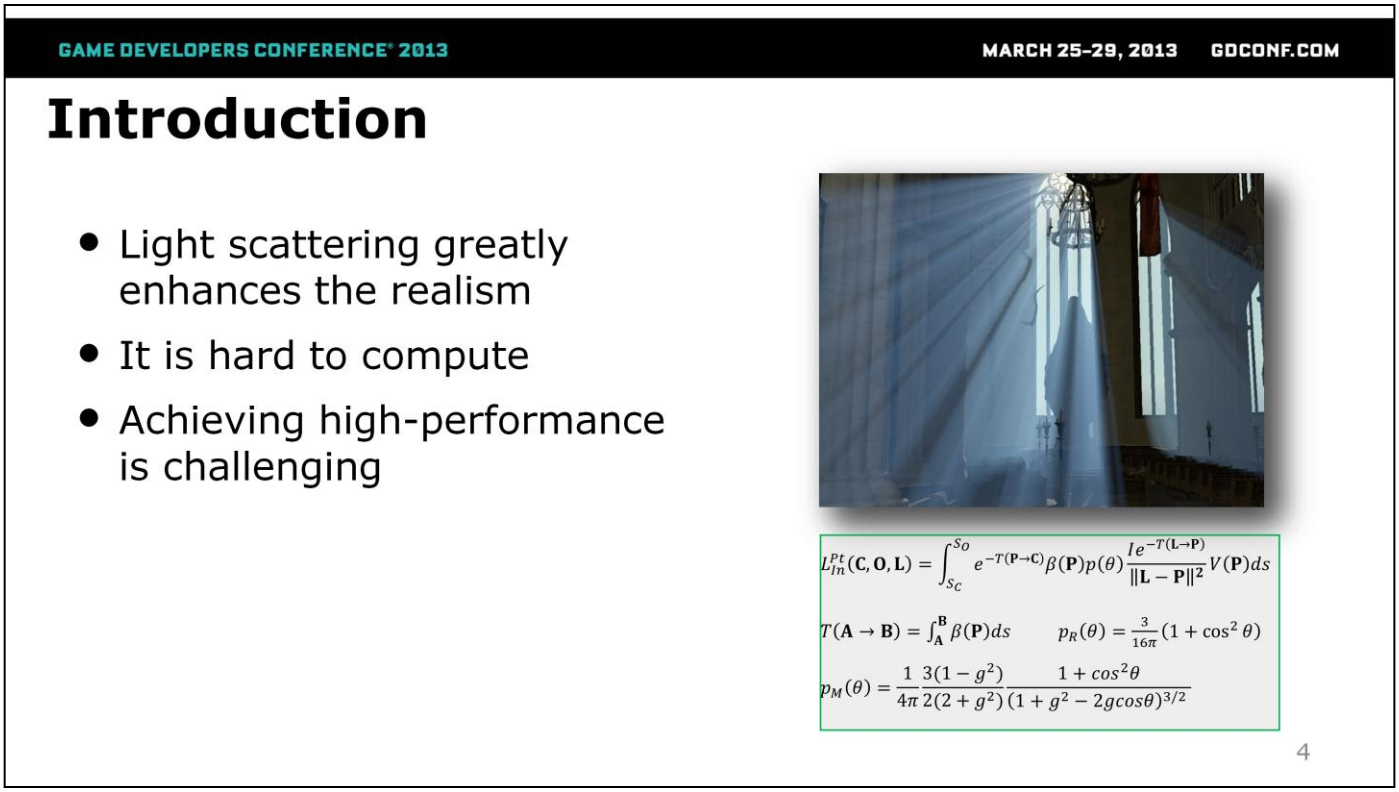
Graphics Software Engineer – Intel Corporation

GAME DEVELOPERS CONFERENCE® 2013
SAN FRANCISCO, CA
MARCH 25–29, 2013
EXPO DATES: MARCH 27–29

Demo

Agenda

- Introduction
- Light Scattering Theory
- Solution to Inscattering Integral
- Epipolar Sampling
- 1D Min/Max Binary Trees
- Implementation
- Performance



Introduction

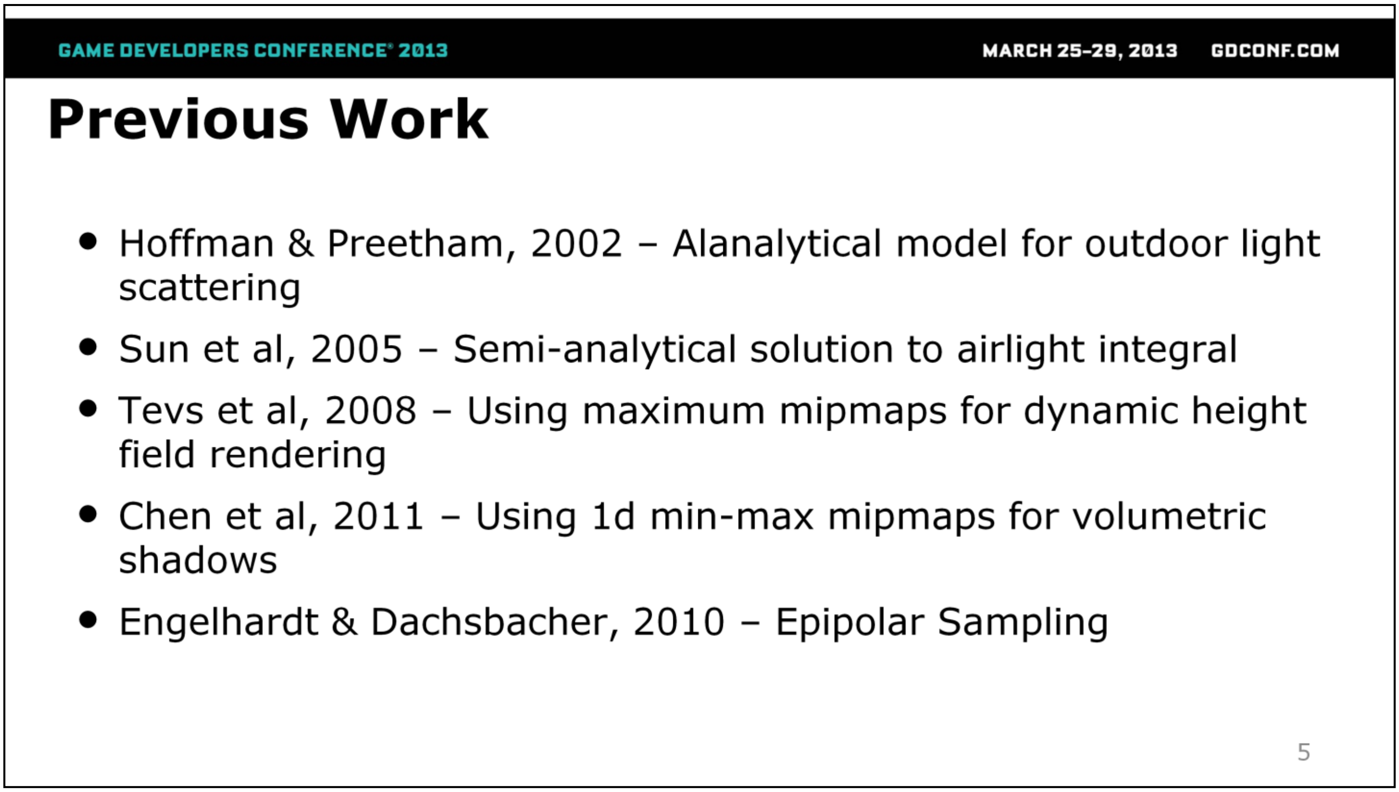
- Light scattering greatly enhances the realism
- It is hard to compute
- Achieving high-performance is challenging

$$L_{In}^{Pt}(\mathbf{C},\mathbf{O},\mathbf{L}) = \int_{S_C}^{S_O} e^{-T(\mathbf{P}\to\mathbf{C})}\beta(\mathbf{P})p(\theta)\frac{Ie^{-T(\mathbf{L}\to\mathbf{P})}}{\|\mathbf{L}-\mathbf{P}\|^2}V(\mathbf{P})ds$$

$$T(\mathbf{A}\to\mathbf{B}) = \int_{\mathbf{A}}^{\mathbf{B}}\beta(\mathbf{P})ds \qquad p_R(\theta) = \frac{3}{16\pi}(1+\cos^2\theta)$$

$$p_M(\theta) = \frac{1}{4\pi}\frac{3(1-g^2)}{2(2+g^2)}\frac{1+cos^2\theta}{(1+g^2-2gcos\theta)^{3/2}}$$

- Light scattering is an important natural phenomenon, which arises when the light interacts with the particles distributed in the media. Rendering such effects greatly enhances the scene realism
- To accurately compute scattering contribution, a complex integral has to be solved for each screen pixel
- Due to the complexity of the computations involved, achieving natural-looking scattering effects at high performance is a challenging problem.



Previous Work

- Hoffman & Preetham, 2002 – Alanalytical model for outdoor light scattering
- Sun et al, 2005 – Semi-analytical solution to airlight integral
- Tevs et al, 2008 – Using maximum mipmaps for dynamic height field rendering
- Chen et al, 2011 – Using 1d min-max mipmaps for volumetric shadows
- Engelhardt & Dachsbacher, 2010 – Epipolar Sampling

There were a lot of work on rendering light scattering effect in a participating medium. And I would like to mention the following most most relevant works

The first one is the work by Hoffman & Preetham who presented an Alanalytical model for outdoor light scattering for directional light source

Sun et al presented semi-analytical solution to airlight integral due to a point light source

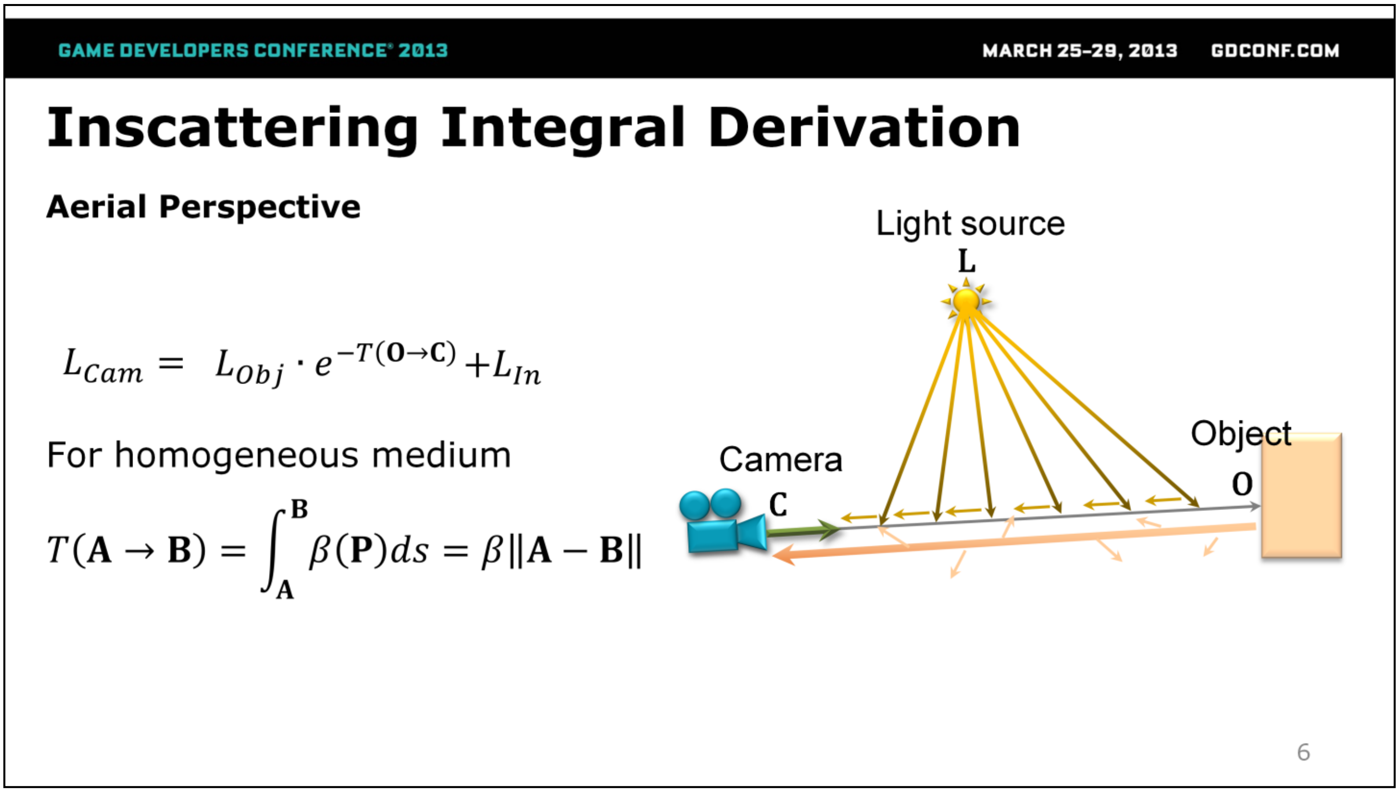
Tevs et al used maximum mipmaps for dynamic height field rendering

And Chen et al later adopted the same idea to accelerate ray marching using 1d min-max mipmaps for volumetric shadows

Engelhardt & Dachsbacher presented epipolar sampling to significantly reduce the number of samples for which computationally expensive ray marching is performed

Our approach combines 1D min/max mipmaps with epipolar sampling to achieve high performance. It also incorporates simple and efficient semi-analytical solution to a light scattering integral due to point light source.

As a result our method is able to achieve high quality rendering at interactive frame rates.



Inscattering Integral Derivation

Aerial Perspective

$$L_{Cam} = L_{Obj} \cdot e^{-T(\mathbf{O}\to\mathbf{C})} + L_{In}$$

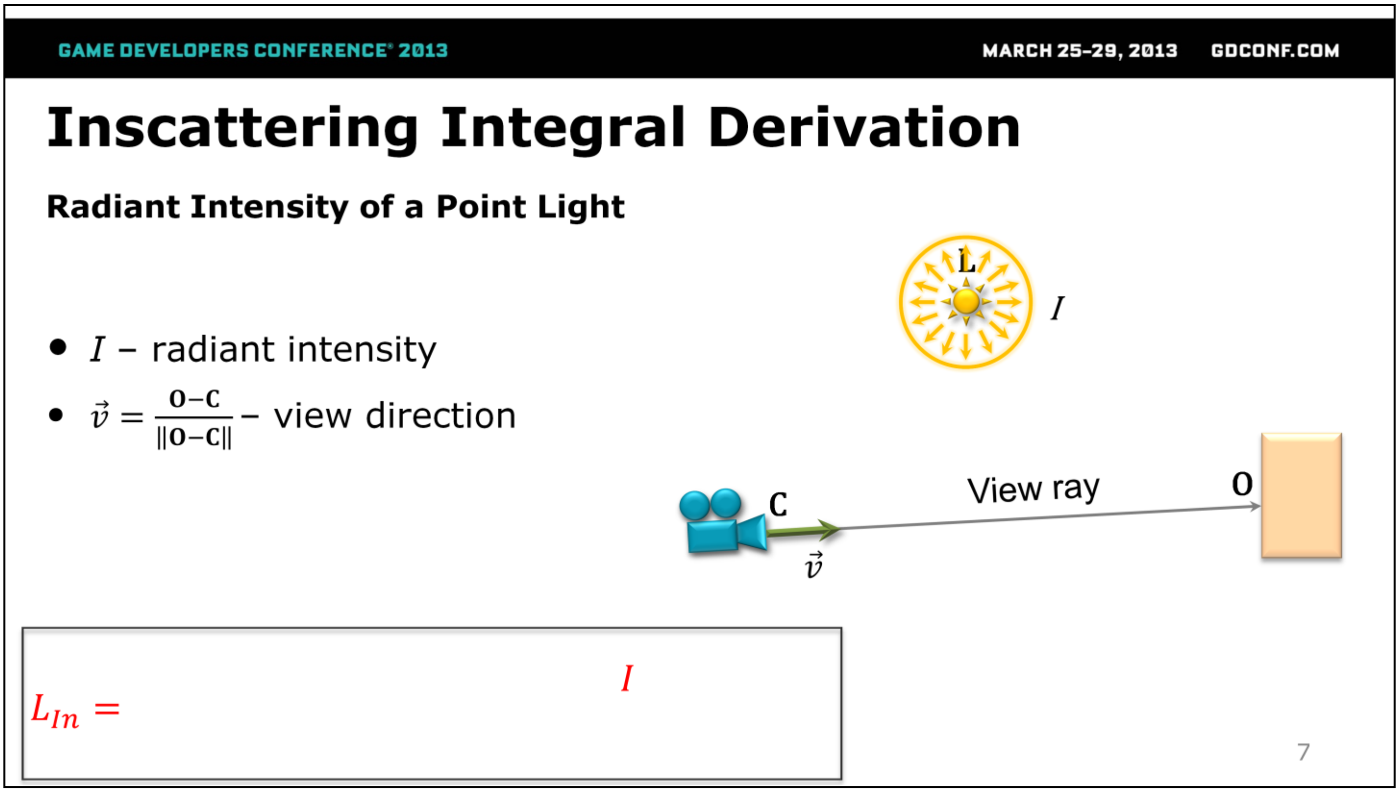
For homogeneous medium

$$T(\mathbf{A}\to\mathbf{B}) = \int_{\mathbf{A}}^{\mathbf{B}} \beta(\mathbf{P})ds = \beta\|\mathbf{A}-\mathbf{B}\|$$

Light source
L
Camera
C
Object
O

Scattering effects affect the appearance of scene objects in two ways which account for a phenomenon called *aerial perspective*

- On one hand some portion of light initially emitted from the object is out-scattered and does not reach the camera
- On the other hand, some sun light is scattered towards the camera on the particles. Thus the final radiance measured at the camera is a sum of two contributions: attenuated object radiance and inscattering
- The T term is called optical depth or optical thickness. It is the integral of scattering coefficient beta over the path. For homogeneous single scattering media we consider it is just the path length multiplied by the scattering coefficient.



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

Inscattering Integral Derivation

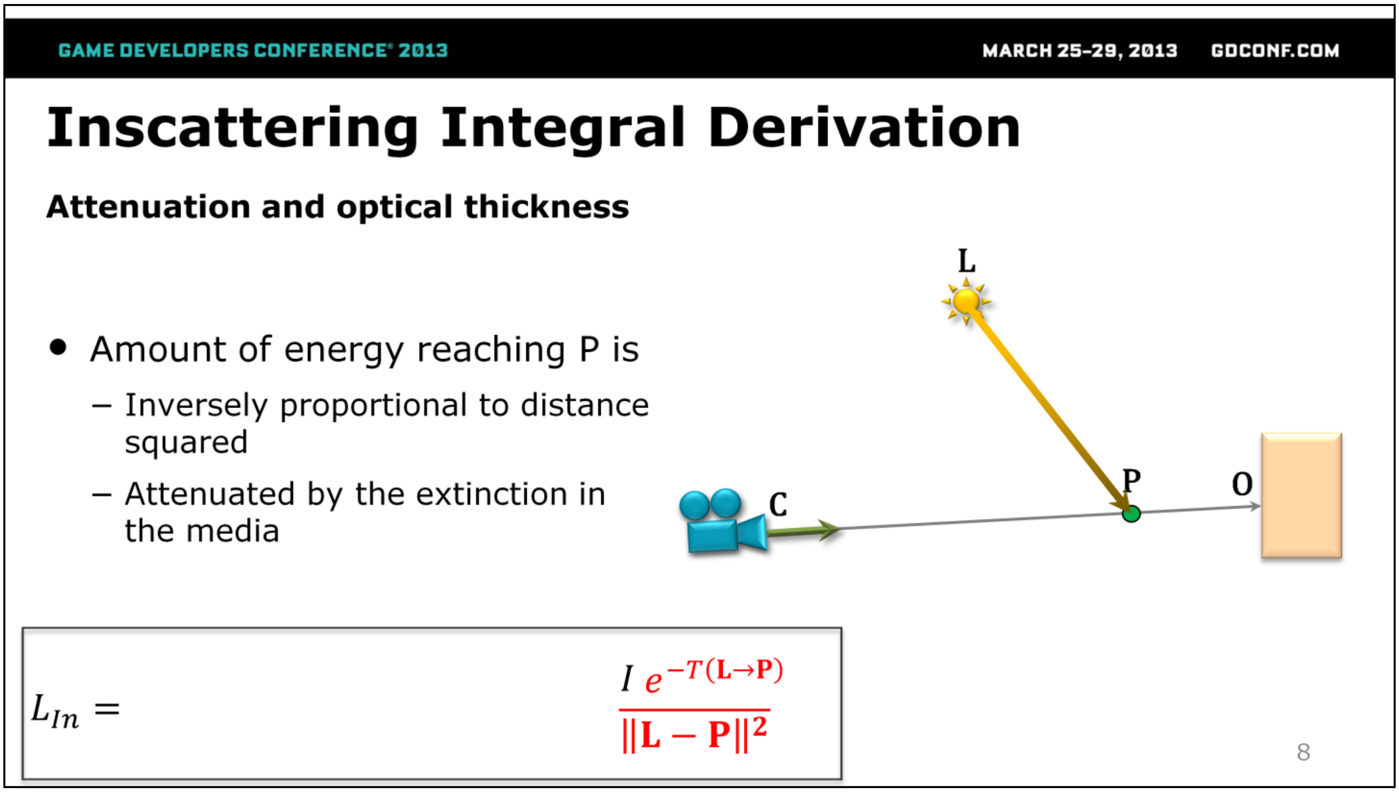
Radiant Intensity of a Point Light

- I – radiant intensity
- $\vec{v} = \frac{\mathrm{O}-\mathrm{C}}{\|\mathrm{O}-\mathrm{C}\|}$ – view direction

$$L_{In} = \qquad I$$

Let’s now start deriving the scattering integral step-by-step and introduce some notations

- The amount of energy a point light source emits is the radiant power intensity I
- View direction is the unit vector directed from camera to object
- Our first step is to notice that the total amount of light scattered towards the camera is proportional to I



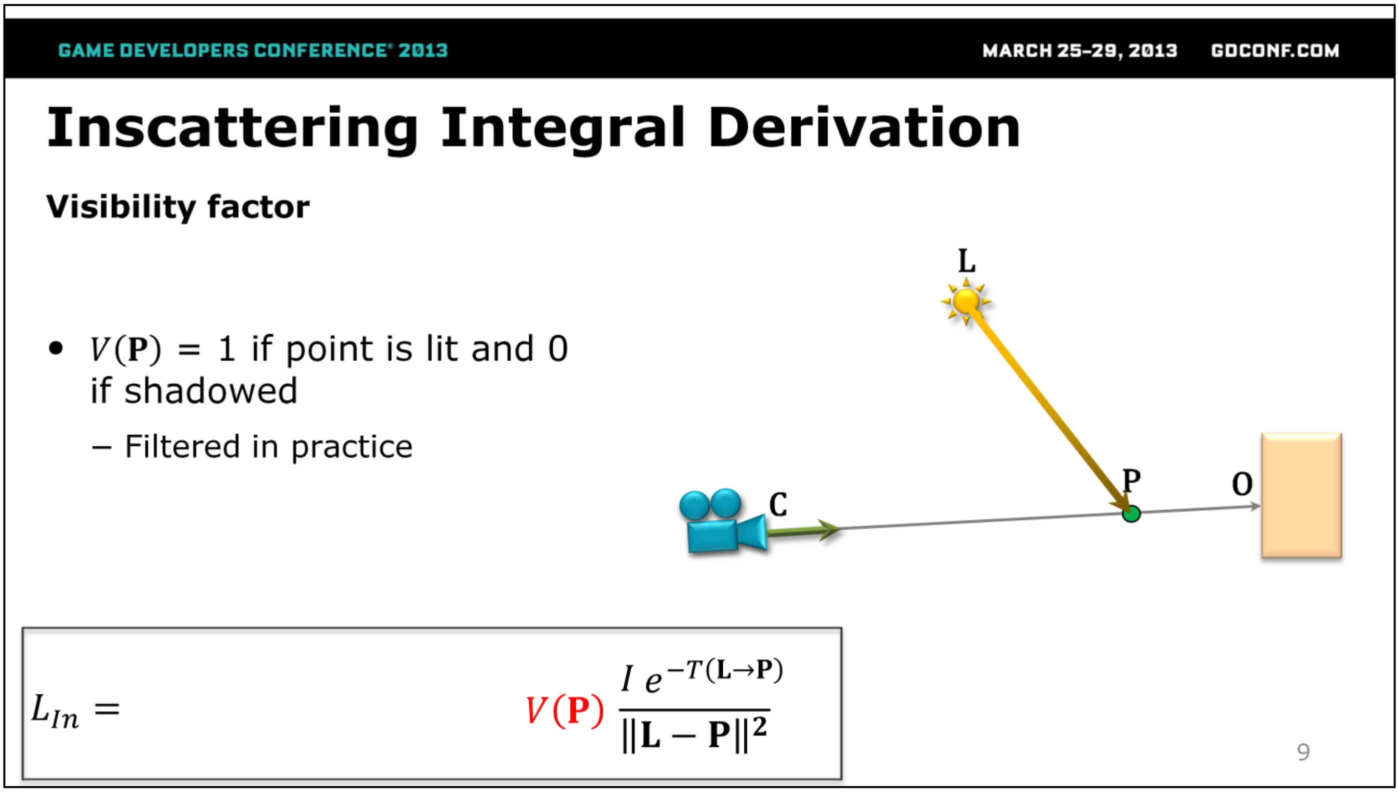
Inscattering Integral Derivation

Attenuation and optical thickness

- Amount of energy reaching P is
  - Inversely proportional to distance squared
  - Attenuated by the extinction in the media

$$L_{In} = \frac{I\, e^{-T(\mathbf{L} \rightarrow \mathbf{P})}}{\|\mathbf{L} - \mathbf{P}\|^2}$$

- Next, let's consider some point P on the view ray
- Amount of energy that reaches this point from the light source
- is inversely proportional to the distance squared
- and some energy is also lost in the media due to out-scattering



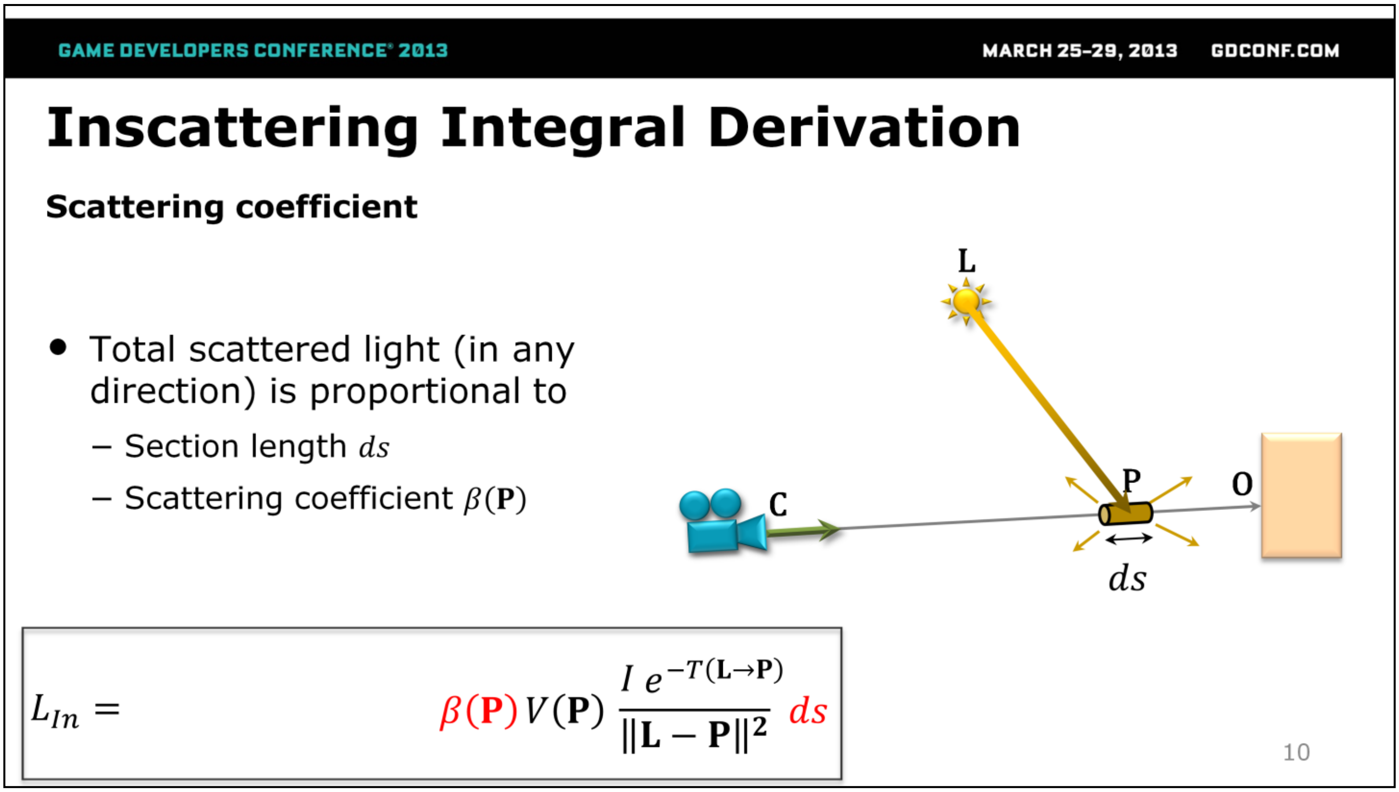
Inscattering Integral Derivation

Visibility factor

- $V(\mathbf{P}) = 1$ if point is lit and 0 if shadowed
  - Filtered in practice

$$L_{In} = \qquad V(\mathbf{P}) \frac{I\, e^{-T(\mathbf{L}\to\mathbf{P})}}{\|\mathbf{L}-\mathbf{P}\|^2}$$

- Next we need to introduce visibility term V(P) indicating if light reaches the point or not. In practice it is filtered to eliminate aliasing



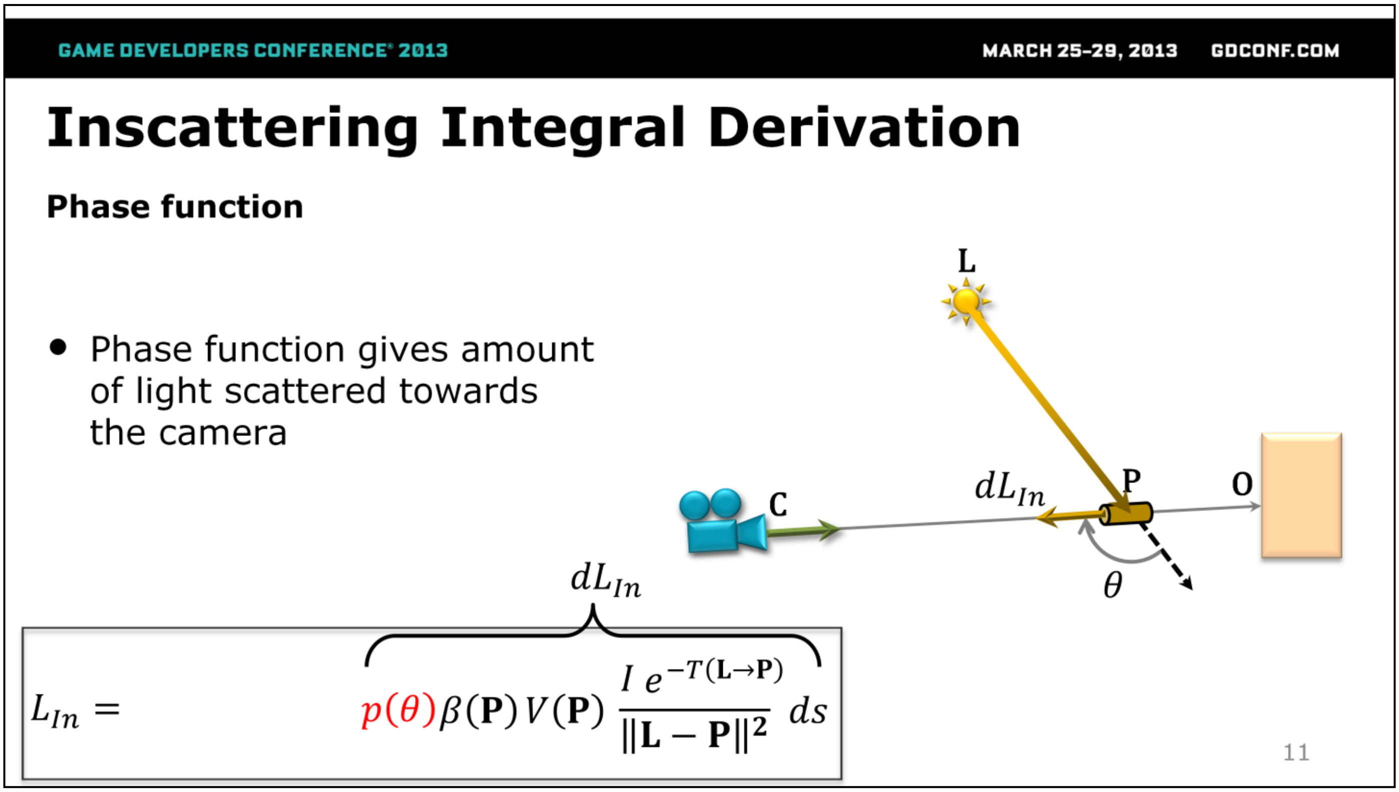
Inscattering Integral Derivation

Scattering coefficient

- Total scattered light (in any direction) is proportional to
  - Section length ds
  - Scattering coefficient $\beta(\mathbf{P})$

$$L_{In} = \beta(\mathbf{P})\, V(\mathbf{P})\, \frac{I\, e^{-T(\mathbf{L}\to\mathbf{P})}}{\|\mathbf{L}-\mathbf{P}\|^2}\, ds$$

- Let’s continue and derive the total amount of energy scattered at infinitely small ray section at point P.
- This amount is proportional to
  - the section length and
  - to the scattering coefficient at this point



Inscattering Integral Derivation

Phase function

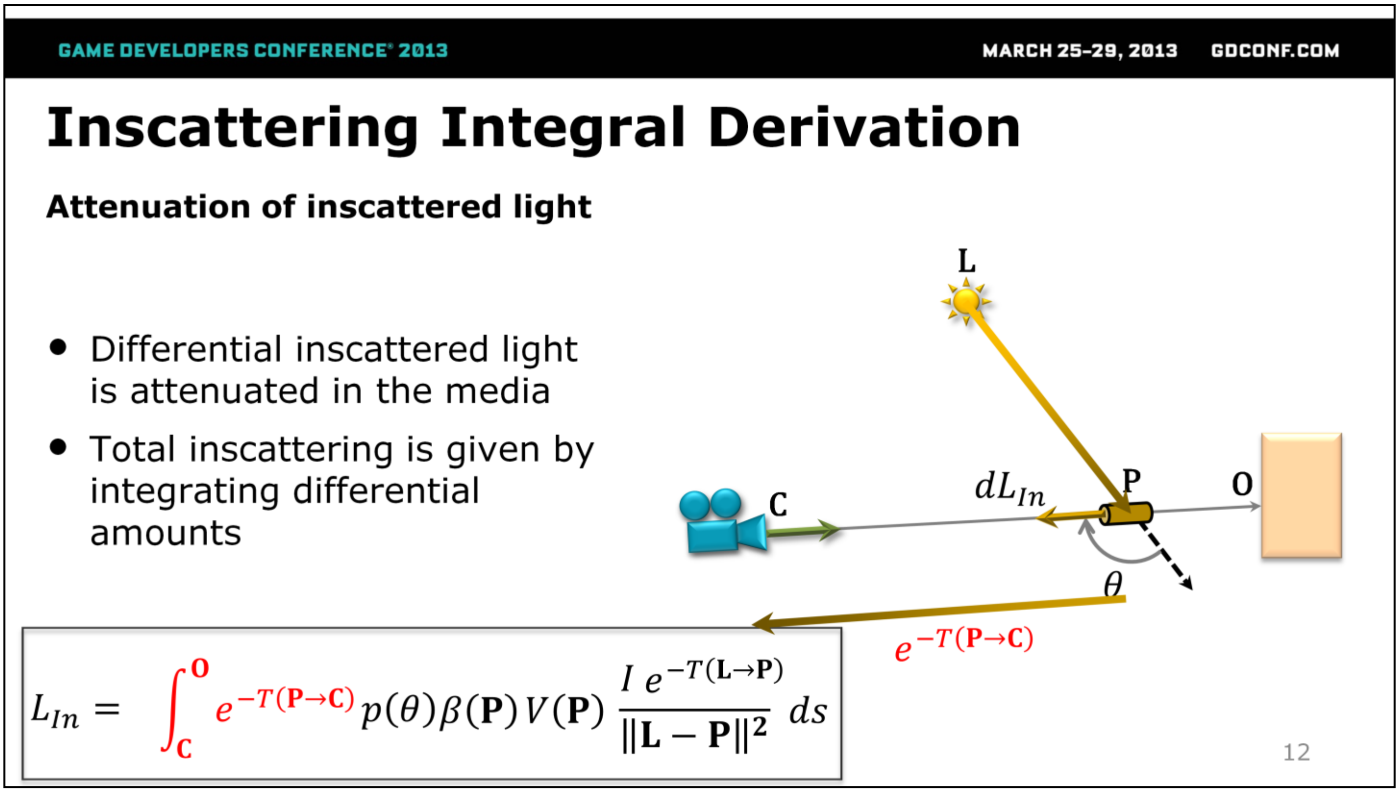
- Phase function gives amount of light scattered towards the camera

$$L_{In} = \overbrace{p(\theta)\beta(\mathbf{P})V(\mathbf{P})\frac{I\,e^{-T(\mathbf{L}\to\mathbf{P})}}{\|\mathbf{L}-\mathbf{P}\|^2}\,ds}^{dL_{In}}$$

But not all energy is scattered towards the camera and some part of it goes in other directions (which is called out-scattering)

- Amount scattered towards the camera is given by the phase function
- All these terms together give differential amount of light scattered towards the camera at point P.

Note that the phase function is assumed to be normalized to unity such that $\int_{\Omega} p(\theta)\,d\omega = 1$ where integration is performed over the whole set of directions Ω.



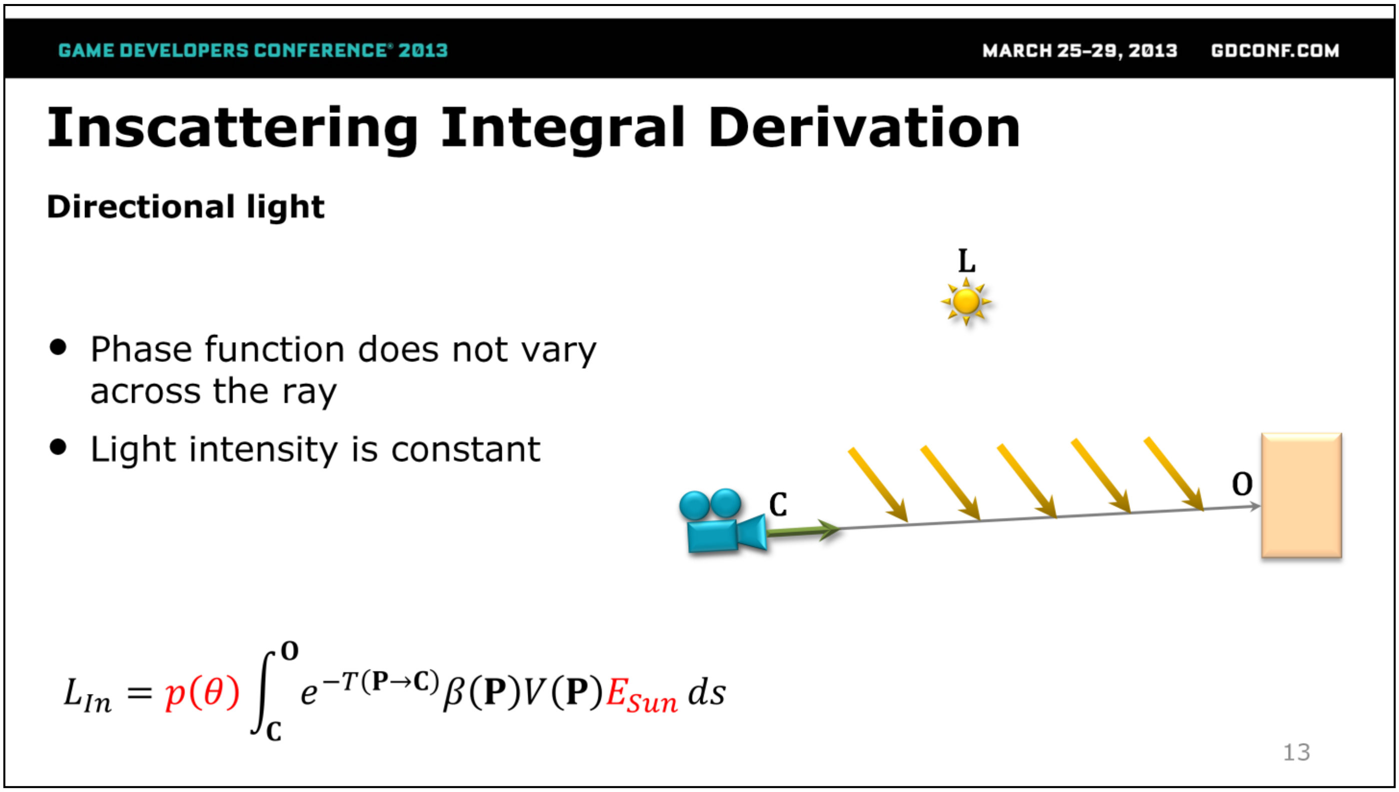
Inscattering Integral Derivation

Attenuation of inscattered light

- Differential inscattered light is attenuated in the media
- Total inscattering is given by integrating differential amounts

$$L_{In} = \int_{\mathbf{C}}^{\mathbf{O}} e^{-T(\mathbf{P}\to\mathbf{C})}\, p(\theta)\beta(\mathbf{P})\, V(\mathbf{P})\, \frac{I\, e^{-T(\mathbf{L}\to\mathbf{P})}}{\|\mathbf{L}-\mathbf{P}\|^2}\, ds$$

- This differential amount of light is attenuated in the media before it reaches the camera
- To get the total inscattering amount we need to integrate all the differential amounts.



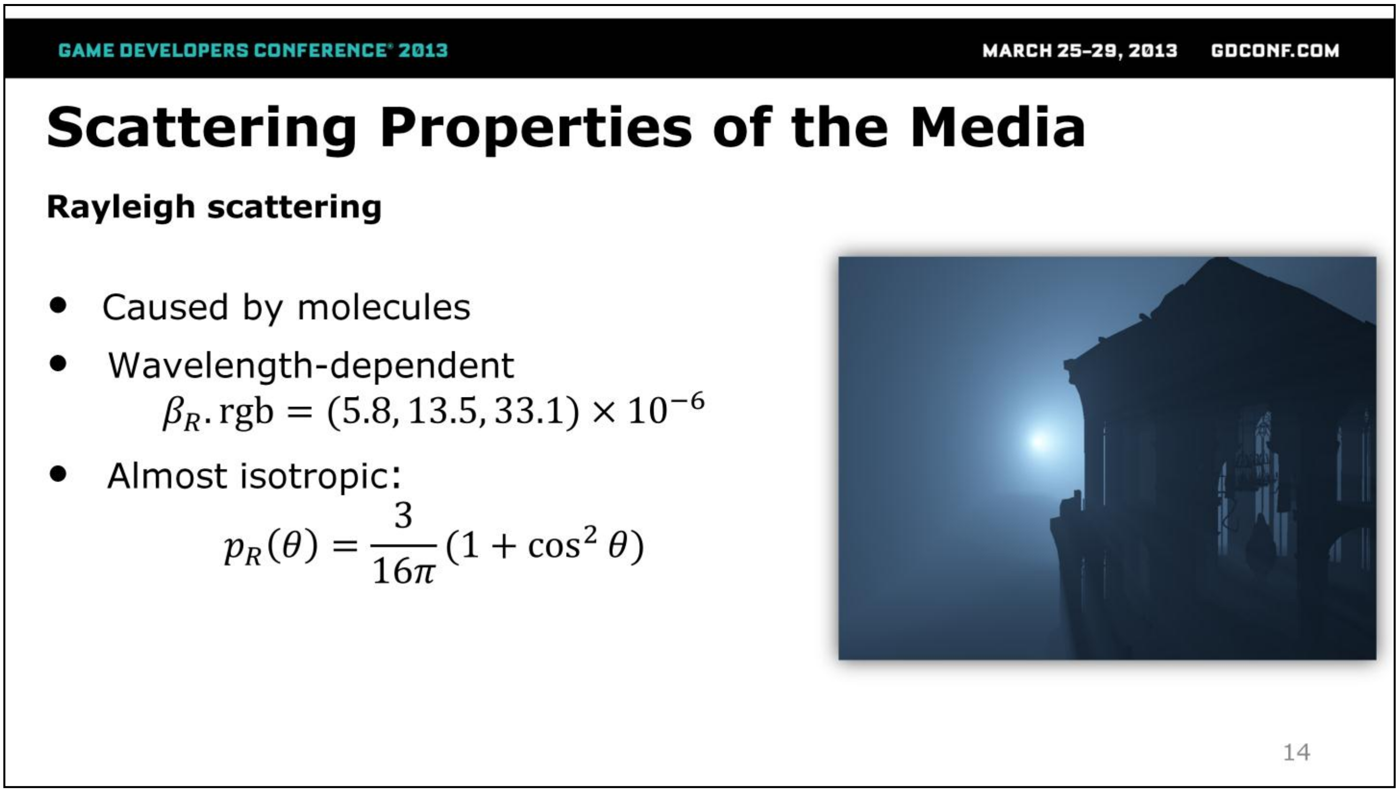
Inscattering Integral Derivation

Directional light

- Phase function does not vary across the ray
- Light intensity is constant

$$L_{In} = p(\theta)\int_{\mathbf{C}}^{\mathbf{O}} e^{-T(\mathbf{P}\to\mathbf{C})}\beta(\mathbf{P})V(\mathbf{P})E_{Sun}\,ds$$

For directional light source, the integral is much simpler because the phase angle does not vary across the ray and the light intensity also does not depend on the distance from the light source



Scattering Properties of the Media

Rayleigh scattering

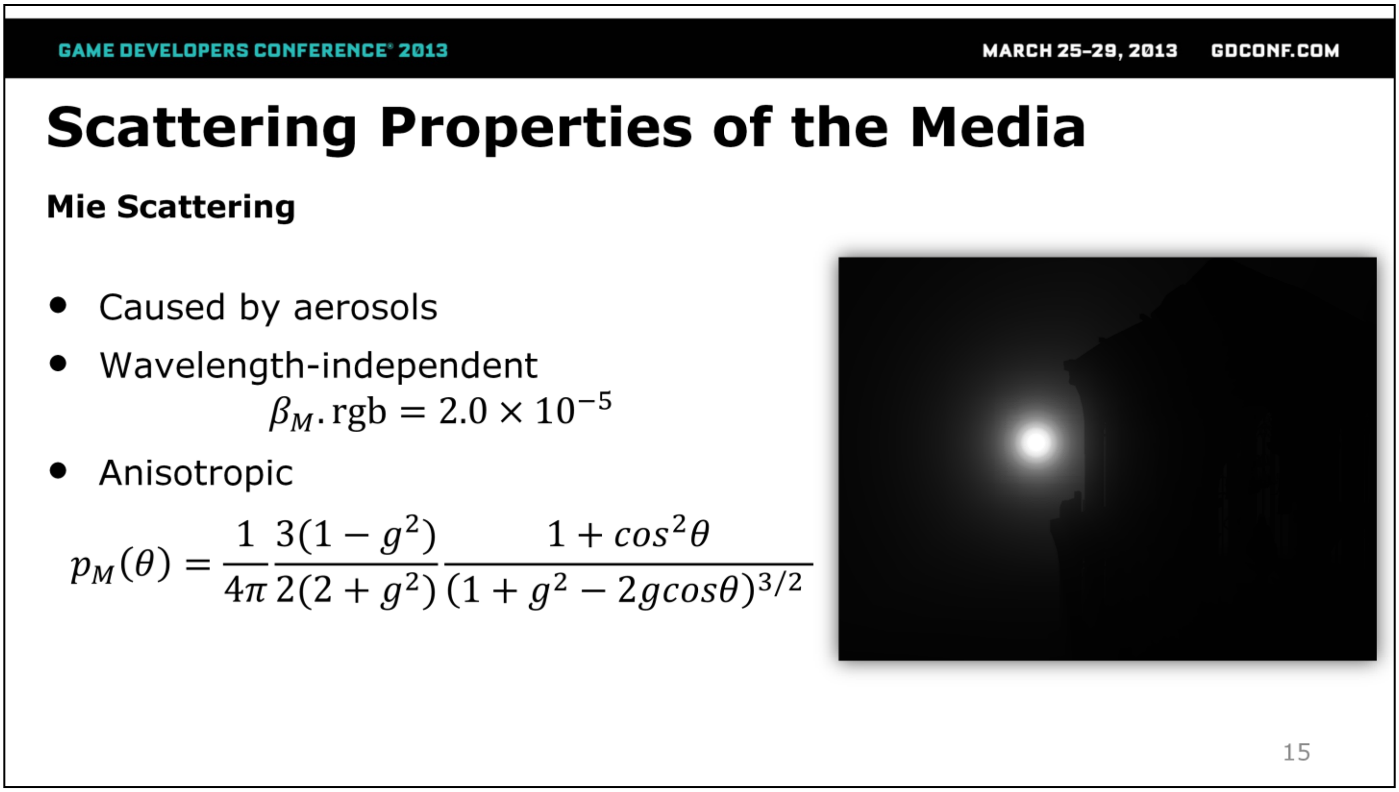
- Caused by molecules
- Wavelength-dependent

$$\beta_R.\mathrm{rgb} = (5.8, 13.5, 33.1) \times 10^{-6}$$

- Almost isotropic:

$$p_R(\theta) = \frac{3}{16\pi}(1 + \cos^2\theta)$$

Folowing many other previous works, we model the participating medium as a mix of two constituents: air molecules and aerosols. The scattering on molecules is described by Rayleigh theory. It is wavelength-dependent, so scattering coefficients are modeled as an RGB vector, but almost isotropic.



Scattering Properties of the Media

Mie Scattering

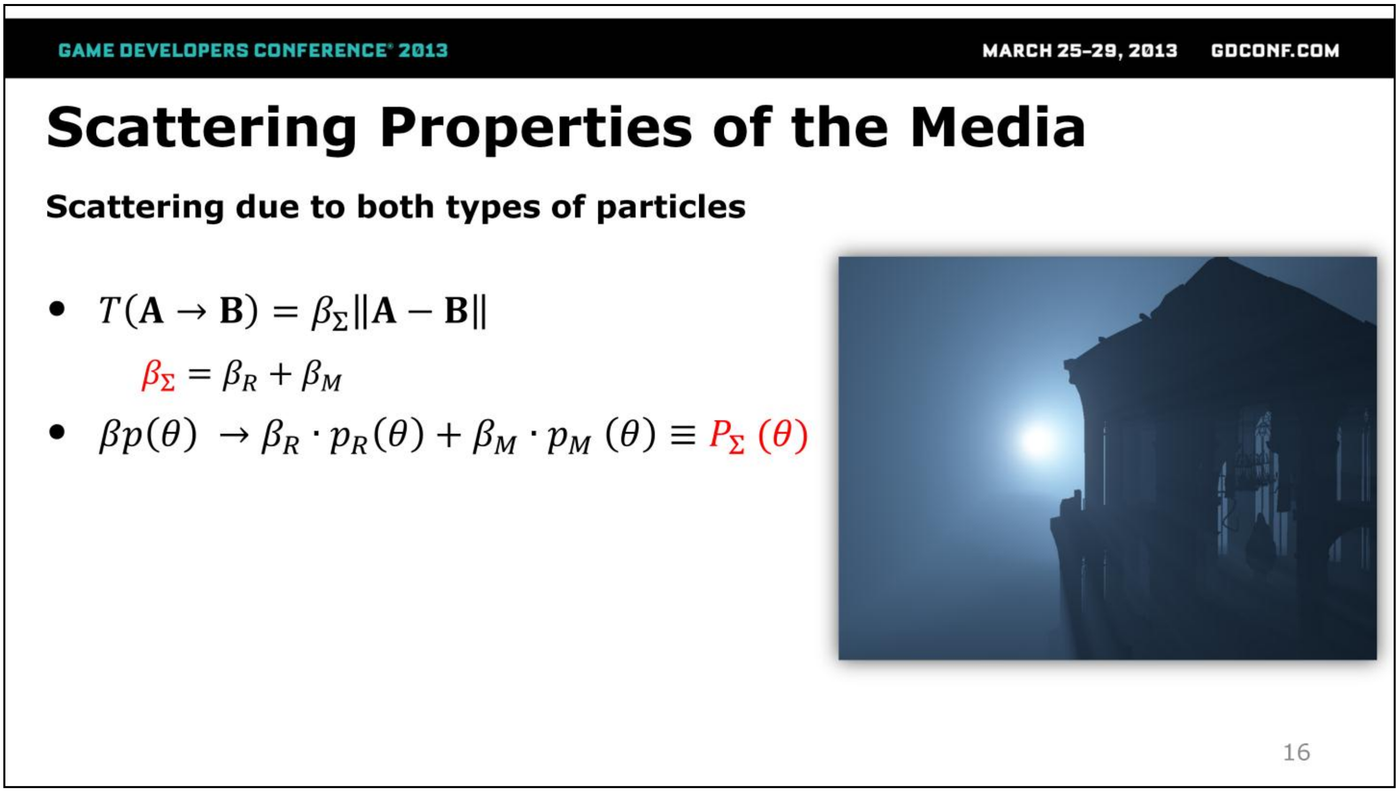
- Caused by aerosols
- Wavelength-independent

$$\beta_M.\mathrm{rgb} = 2.0 \times 10^{-5}$$

- Anisotropic

$$p_M(\theta) = \frac{1}{4\pi}\frac{3(1-g^2)}{2(2+g^2)}\frac{1+cos^2\theta}{(1+g^2-2gcos\theta)^{3/2}}$$

Scattering on aerosols is described by Mie theory. It is much less wavelength-dependent but significantly anisotropic. The phase function is much more complex and can be approximated using Cornette-Shanks function.



Scattering Properties of the Media

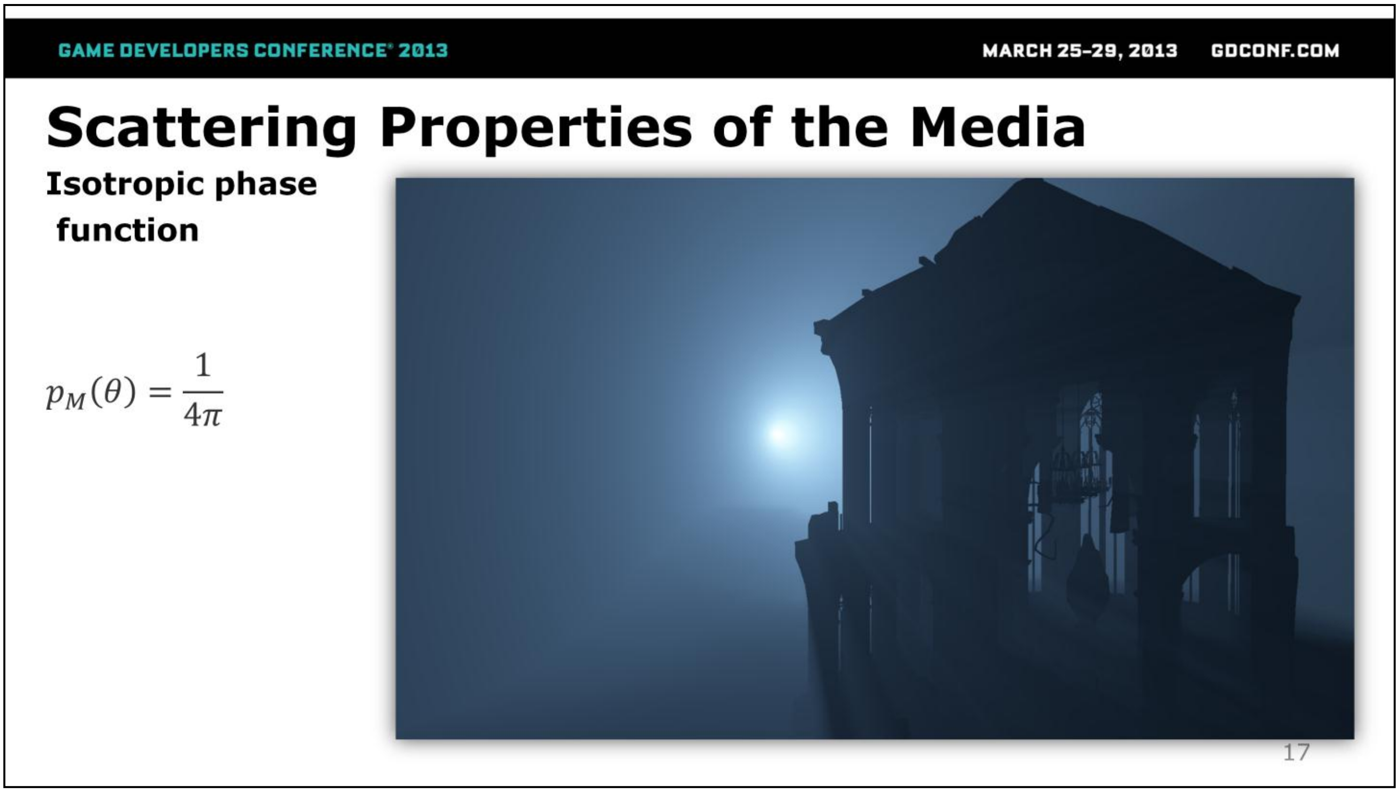
Scattering due to both types of particles

- $T(\mathbf{A} \rightarrow \mathbf{B}) = \beta_{\Sigma} \|\mathbf{A} - \mathbf{B}\|$

  $\beta_{\Sigma} = \beta_R + \beta_M$
- $\beta p(\theta) \rightarrow \beta_R \cdot p_R(\theta) + \beta_M \cdot p_M(\theta) \equiv P_{\Sigma}(\theta)$

In the presense of two consituents, the light is attenuated by both of them. For homogeneous medium, optical thickness is simply the total scattering coefficient multiplied by the path length, while scattering by each type of particle is modulate by its own phase function

We will denote total scattering coefficeint by betha with downscript sigma and the sum of phase functions modulated by the scattering coefficient by capital P with downsript sigma.

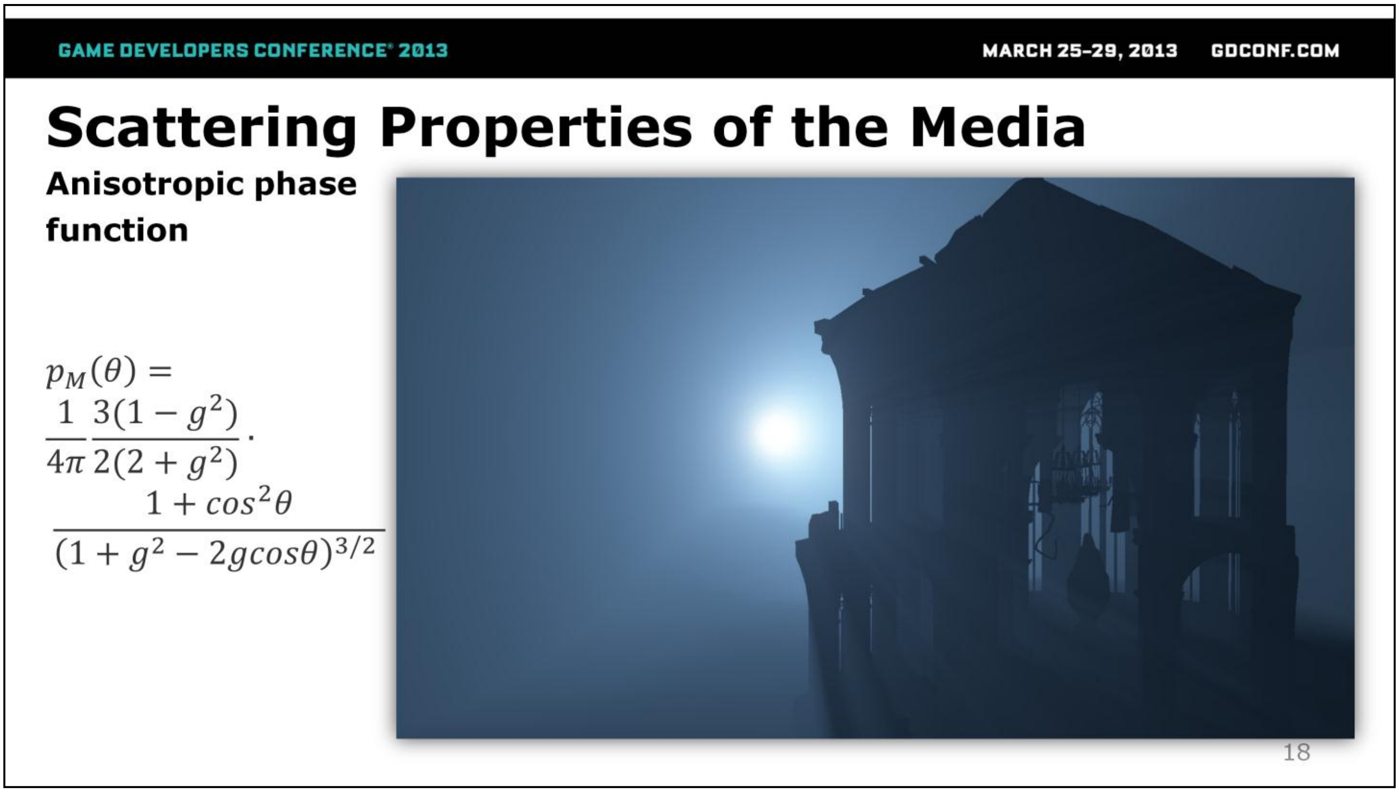


Scattering Properties of the Media

Isotropic phase function

$$p_M(\theta) = \frac{1}{4\pi}$$

This picture shows rendering spot light source with isotropic phase function

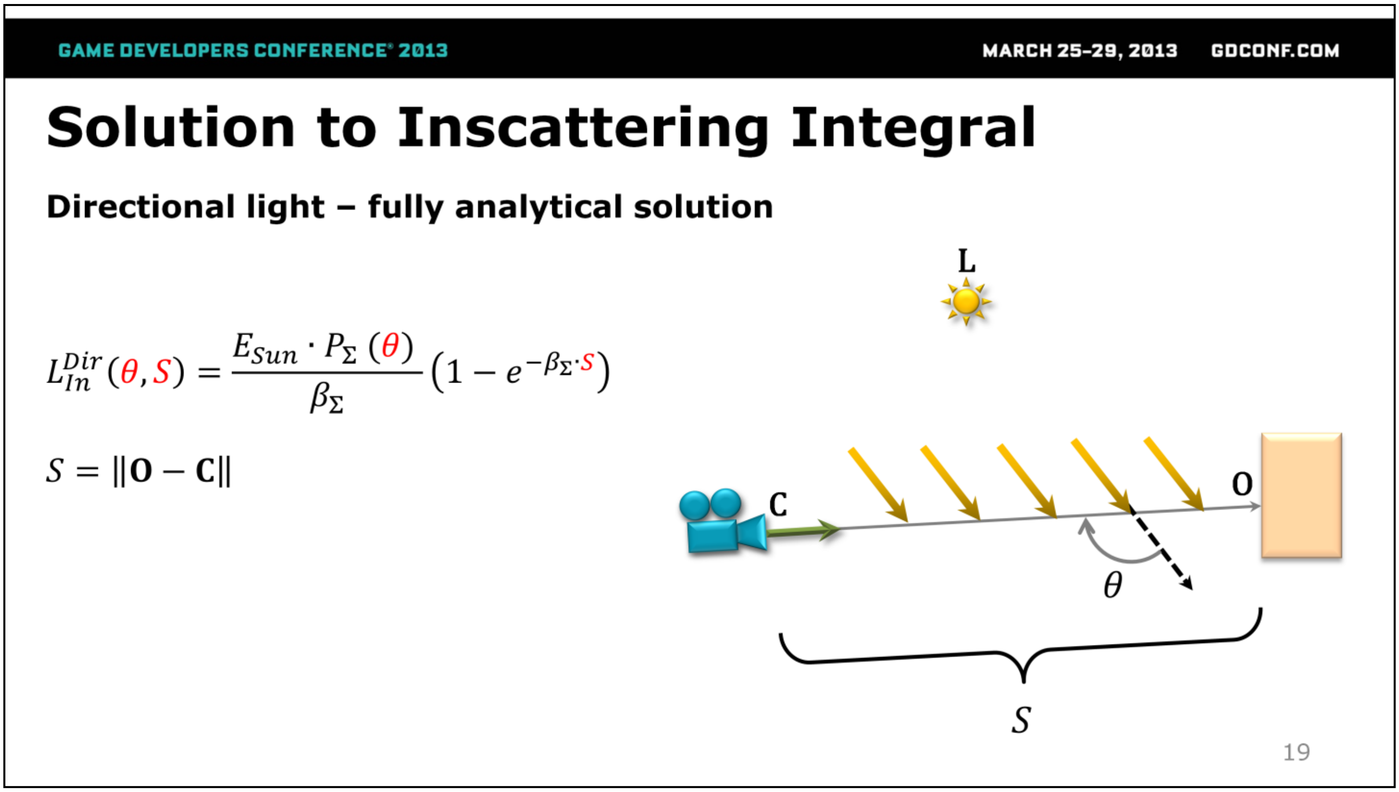


Scattering Properties of the Media

Anisotropic phase function

$$p_M(\theta) = \frac{1}{4\pi}\frac{3(1-g^2)}{2(2+g^2)} \cdot \frac{1+cos^2\theta}{(1+g^2-2gcos\theta)^{3/2}}$$

While anisotropic one creates a nice glow around the light source



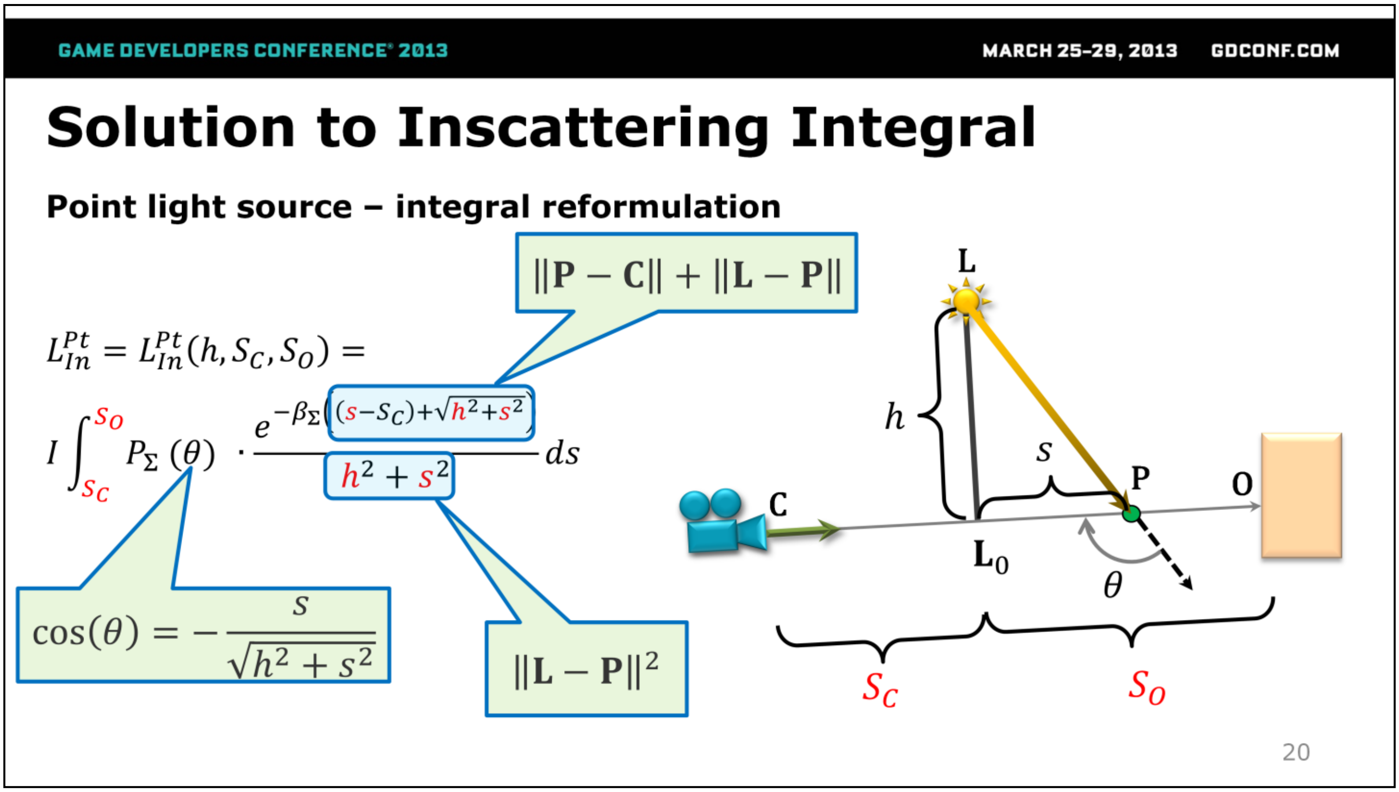
Solution to Inscattering Integral

Directional light – fully analytical solution

$$L_{In}^{Dir}(\theta, S) = \frac{E_{Sun} \cdot P_{\Sigma}(\theta)}{\beta_{\Sigma}}\left(1 - e^{-\beta_{\Sigma} \cdot S}\right)$$

$$S = \|\mathbf{O} - \mathbf{C}\|$$

Let’s now talk about how to solve the inscattering integral. We start from simpler case. For a directional light source, there is a fully analytical solution derived by Hoffman and Preetham. The inscattering contribution depends only on the ray length S and the angle θ between the view ray $\vec{v}$ and the light direction $\vec{L}$.



Solution to Inscattering Integral

Point light source – integral reformulation

$$L_{In}^{Pt} = L_{In}^{Pt}(h, S_C, S_O) =$$

$$I \int_{S_C}^{S_O} P_{\Sigma}(\theta) \cdot \frac{e^{-\beta_{\Sigma}\left(\boxed{(s-S_C)+\sqrt{h^2+s^2}}\right)}}{\boxed{h^2+s^2}} ds$$

$\|\mathbf{P}-\mathbf{C}\| + \|\mathbf{L}-\mathbf{P}\|$

$\cos(\theta) = -\frac{s}{\sqrt{h^2+s^2}}$

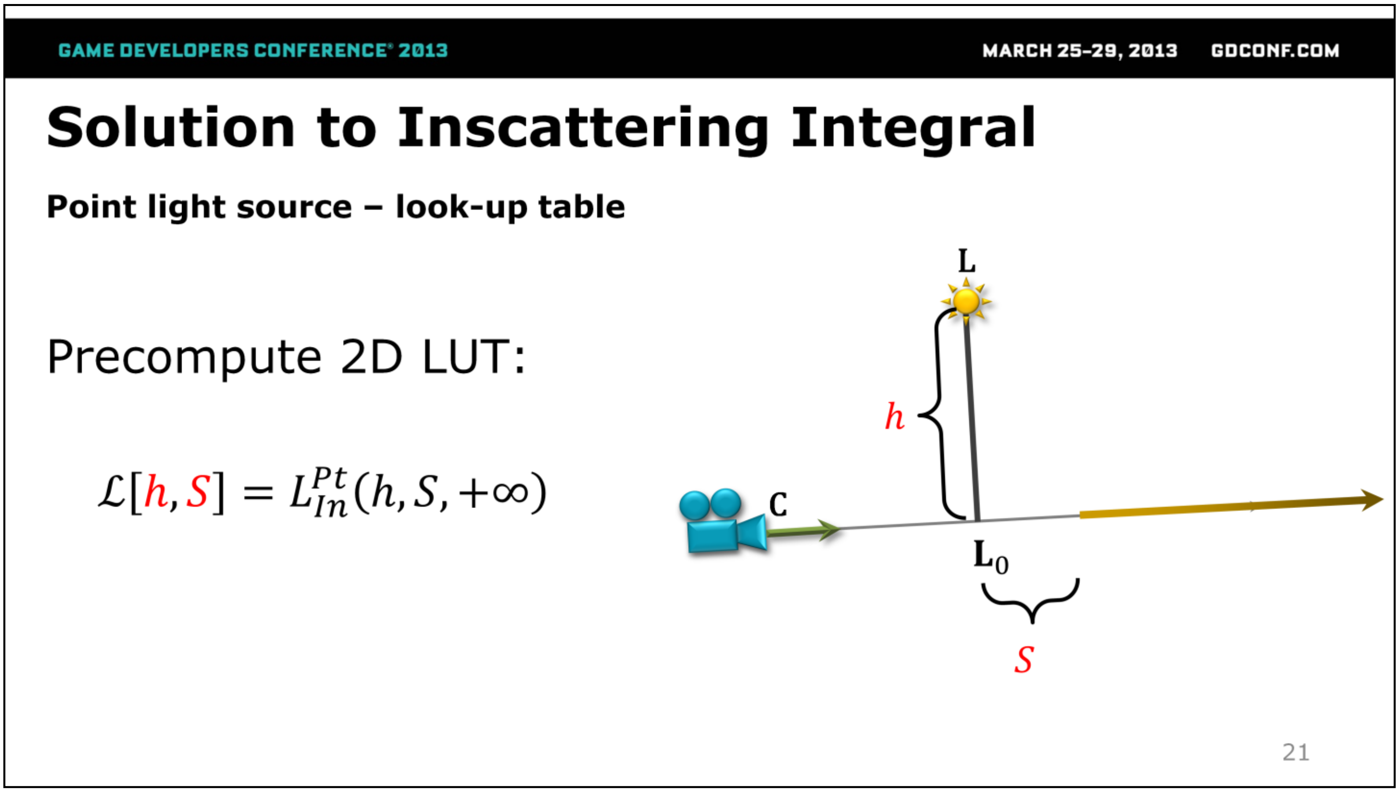
$\|\mathbf{L}-\mathbf{P}\|^2$

Unfortunately, for a point light source, there is no closed form solution. However, we can derive a simple semi-analytical solution which requires one look-up into a precomputed table. To do this, we first rewrite the inscattering integral.

We first relate the variable s to the projection L0 of the light source onto the view ray

- The term in the denominator is then squared distance from the light source to the current point P
- and the highlighted term in the exponent is the sum of distances from the light to the point and from the point to the camera
- The phase functions are usually functions of cosine term which can be easily evaluated

Now we can see that the inscattering integral depends on only three variables: the distance h from the light to the ray and the signed distances from the light projected position to the camera Sc and to the ray termination point So.



Solution to Inscattering Integral

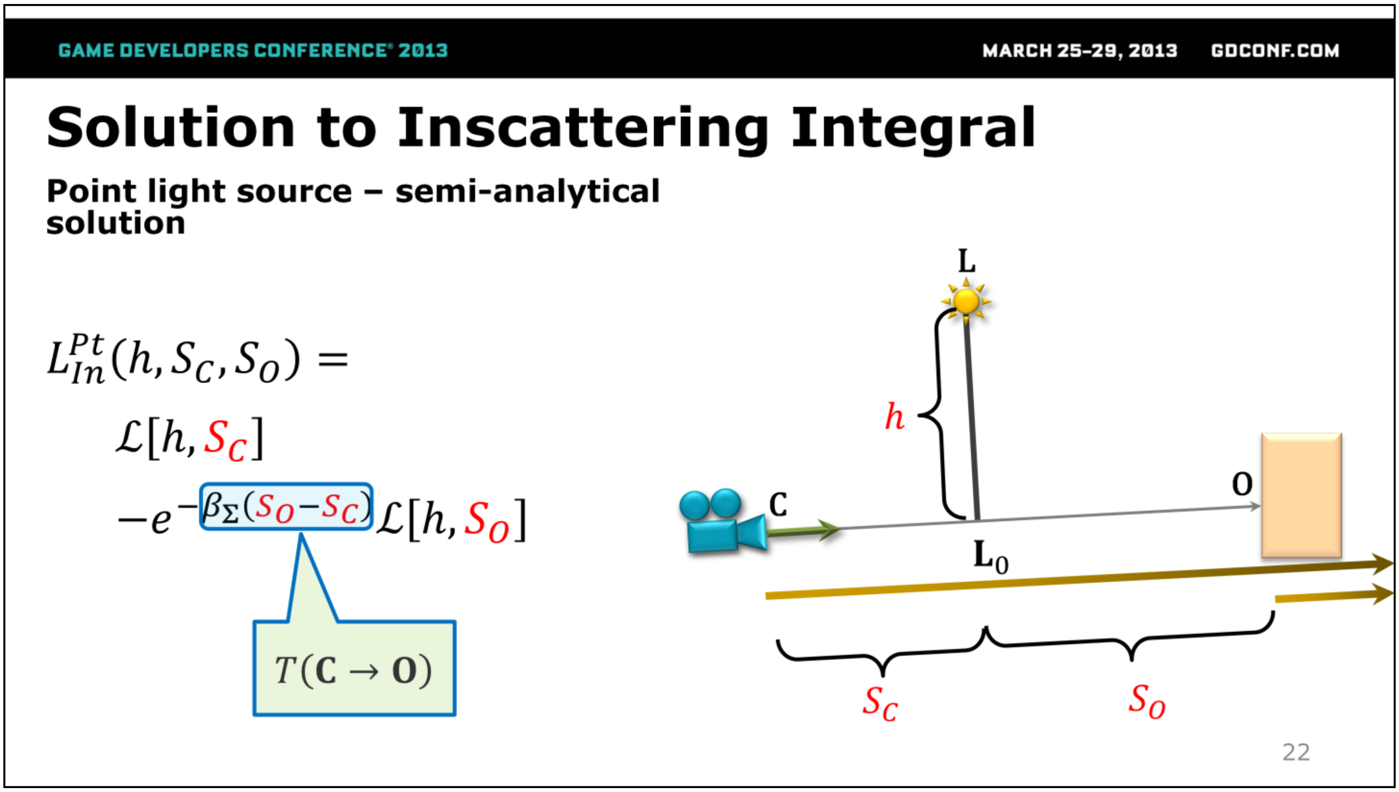
Point light source – look-up table

Precompute 2D LUT:

$$\mathcal{L}[h,S] = L_{In}^{Pt}(h,S,+\infty)$$

To solve inscattering integral, we can precompute the inscattering integral for different values of h and start distance Sc to infinity

These values can be stored in a two-dimensional look-up table



Solution to Inscattering Integral

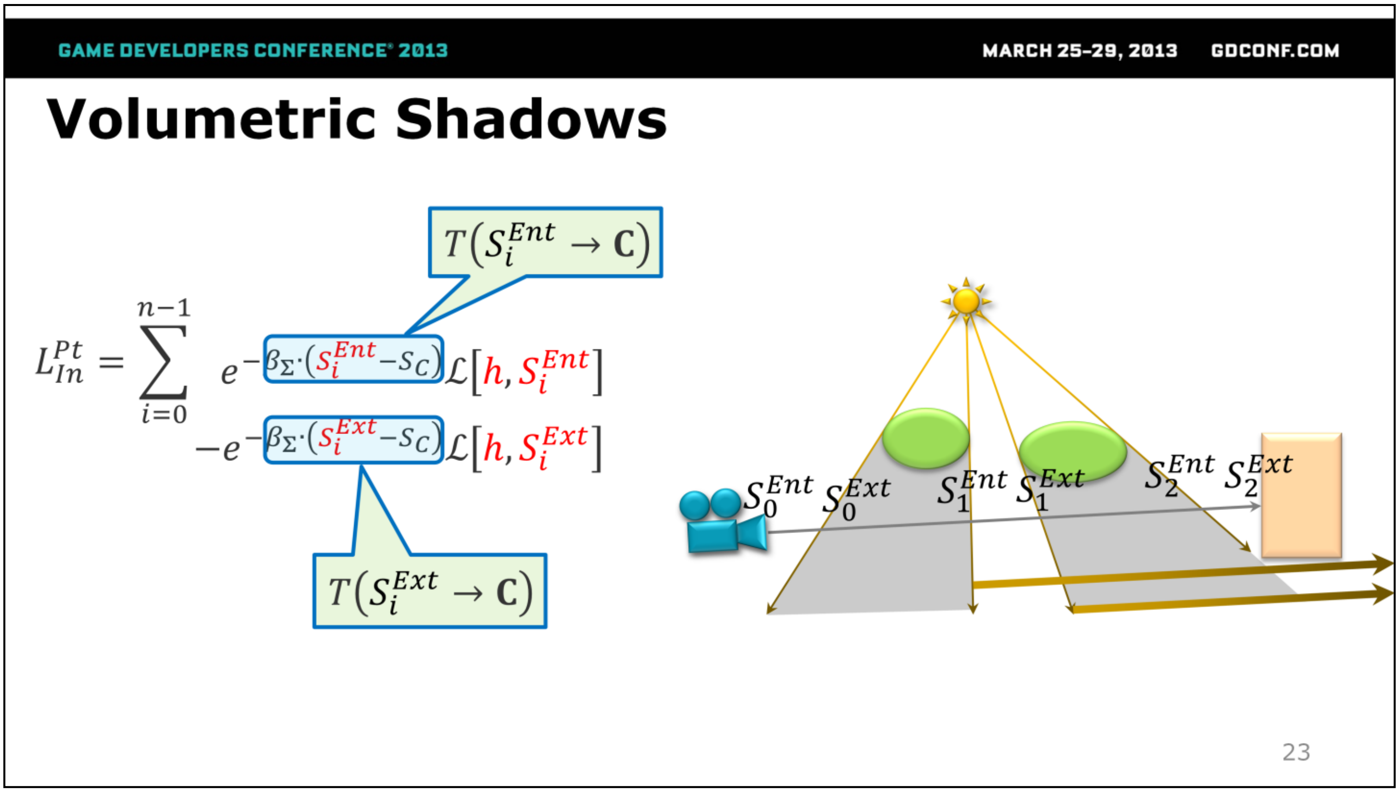
Point light source – semi-analytical solution

$$L_{In}^{Pt}(h,S_C,S_O) = \mathcal{L}[h,S_C] - e^{-\boxed{\beta_\Sigma(S_O-S_C)}}\mathcal{L}[h,S_O]$$

$T(\mathbf{C} \to \mathbf{O})$

The inscattering integral can then be computed as the

- Integral from the camera towards infinity
- Minus integral from the object towards infinity
    - We must not also forget that the light reaching the camera is attenuated in the media



Volumetric Shadows

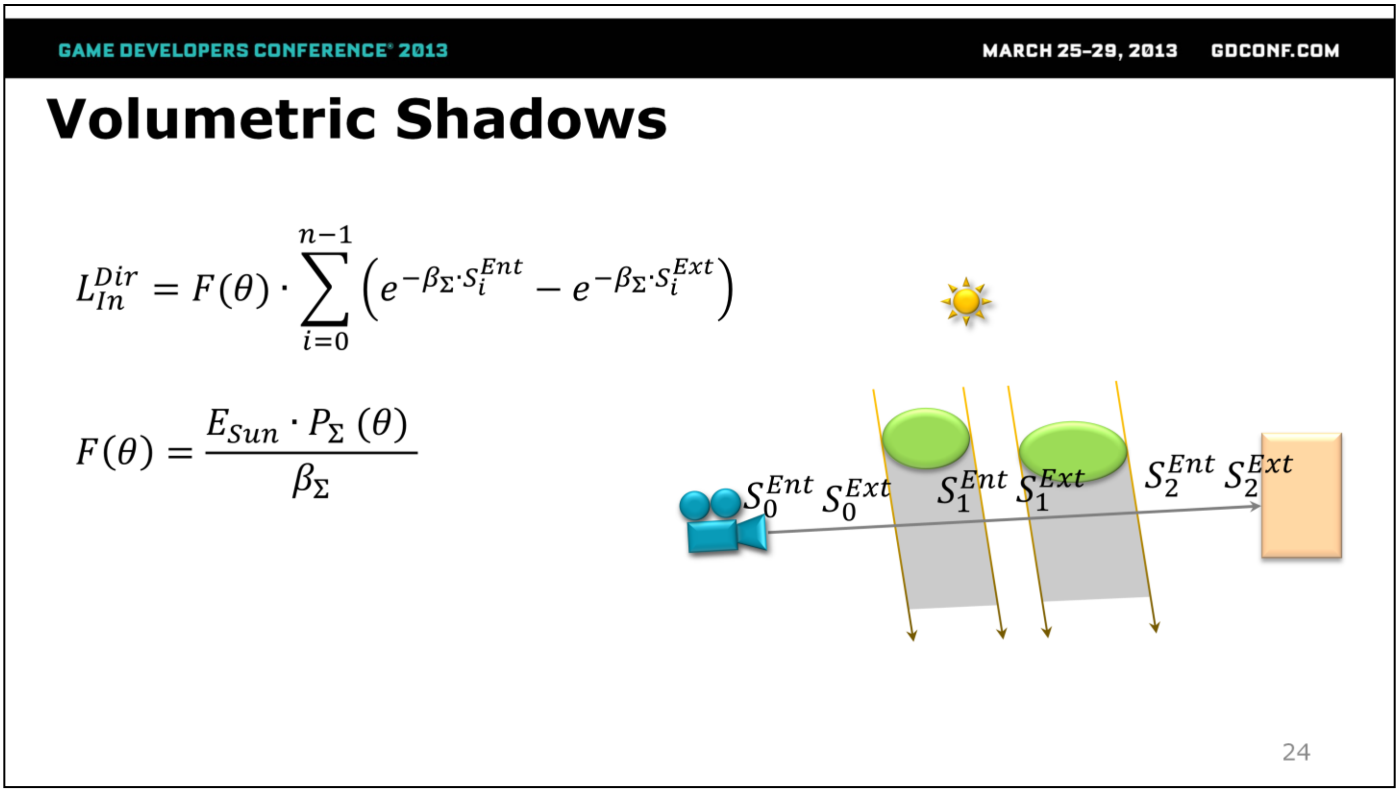
$$L_{In}^{Pt} = \sum_{i=0}^{n-1} e^{\boxed{-\beta_\Sigma\cdot(s_i^{Ent}-s_C)}}\mathcal{L}[h, s_i^{Ent}] - e^{\boxed{-\beta_\Sigma\cdot(s_i^{Ext}-s_C)}}\mathcal{L}[h, s_i^{Ext}]$$

$T(s_i^{Ent} \to \mathbf{C})$

$T(s_i^{Ext} \to \mathbf{C})$

OK, so where is the volumetric shadows? Under single scattering assumption, light is only scattered from directly illuminated ray sections and there is no scattering from shadowed ray sections (where V(P) term is zero)

- Thus due to additive nature of integration, we can subdivide the ray into lit regions and sum contributions from all lit segments
- For each ray segment we can then compute scattering contribution as the integral from the beginning of the section
- Minus contribution from the end of the section. Attenuation in the media must be taken into account here
- The resulting contribution is the sum of contributions from all lit segments



Volumetric Shadows

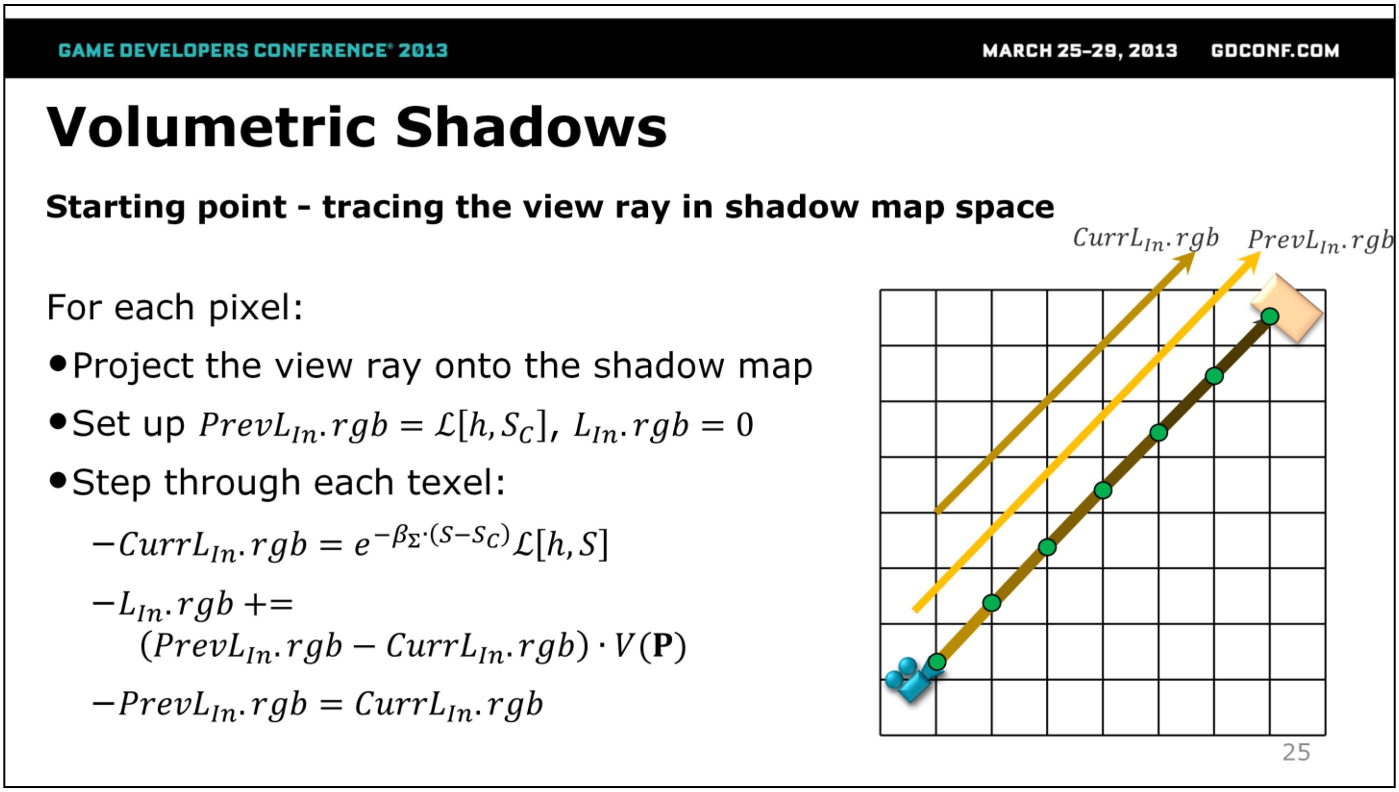
$$L_{In}^{Dir} = F(\theta)\cdot\sum_{i=0}^{n-1}\left(e^{-\beta_{\Sigma}\cdot S_i^{Ent}} - e^{-\beta_{\Sigma}\cdot S_i^{Ext}}\right)$$

$$F(\theta) = \frac{E_{Sun}\cdot P_{\Sigma}\,(\theta)}{\beta_{\Sigma}}$$

For a directional light source there is a analytical solution which does not require look-up tables

Note that in this case F term can be computed outside the summation

The question now is how to determine lit/unlit sections. There are a number of approaches which use light volumes for this purpose, but such methods are very fill-rate intensive and the performance is heavily scene-dependent. So in our approach we use shadow map



Volumetric Shadows

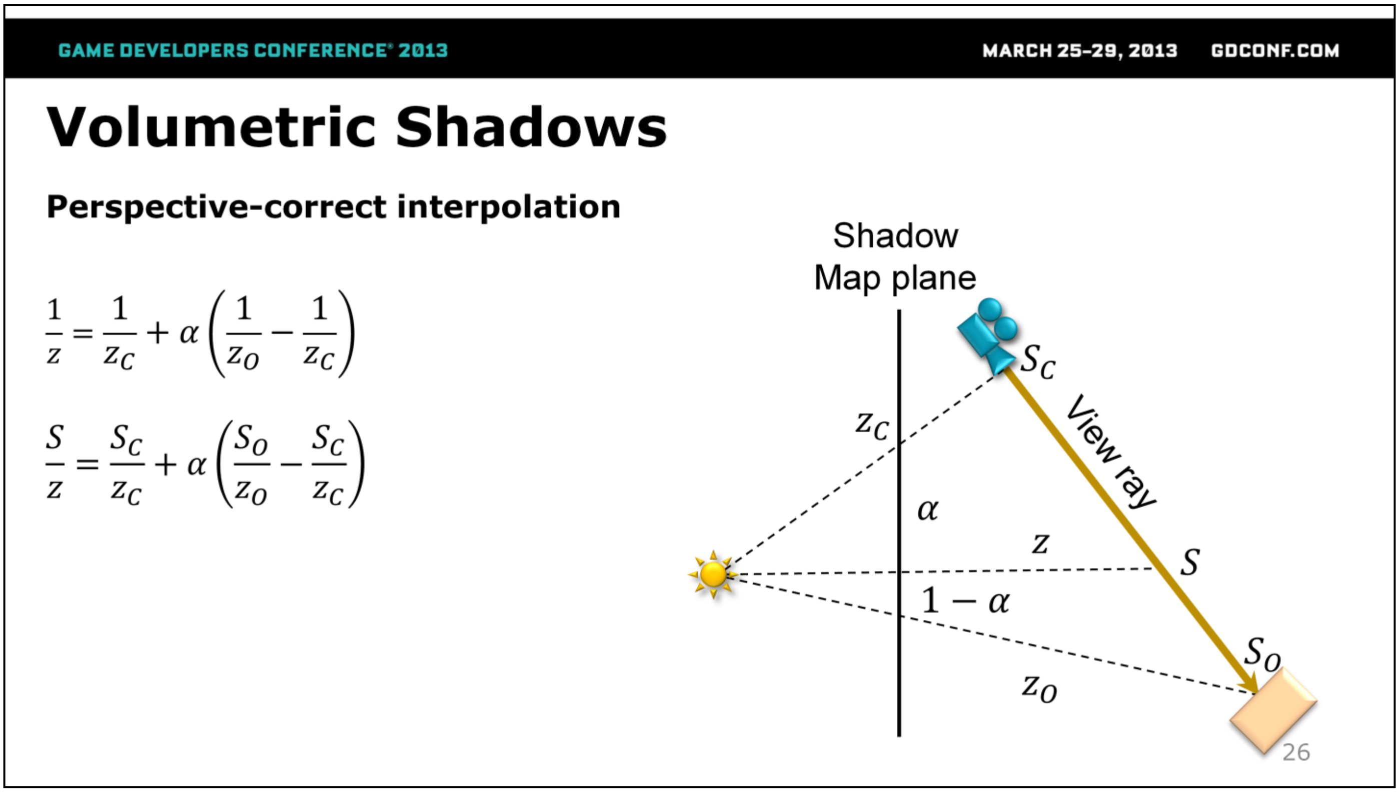
Starting point - tracing the view ray in shadow map space

For each pixel:

- Project the view ray onto the shadow map
- Set up $PrevL_{In}.rgb = \mathcal{L}[h, S_C]$, $L_{In}.rgb = 0$
- Step through each texel:
  - $CurrL_{In}.rgb = e^{-\beta_\Sigma \cdot (S - S_C)} \mathcal{L}[h, S]$
  - $L_{In}.rgb \mathrel{+}= (PrevL_{In}.rgb - CurrL_{In}.rgb) \cdot V(\mathbf{P})$
  - $PrevL_{In}.rgb = CurrL_{In}.rgb$

So as our starting point we can derive the algorithm which does the following:

- Projects the view ray onto the shadow map,
- Sets up total inscattering and inscattering integral value from the end of the current section to infinity
- Goes through each shadow map texel. For each texel it computes inscattering integral from the end of the current section to infinity
- The amount of light scattered from this section is the difference of previous and current integrals multiplied by the visibility term. It is accumulated in the net inscattering variable
- The next texel is then processed until the whole ray is marched
- Note that only one exponent and/our look-up has to be performed in the loop.



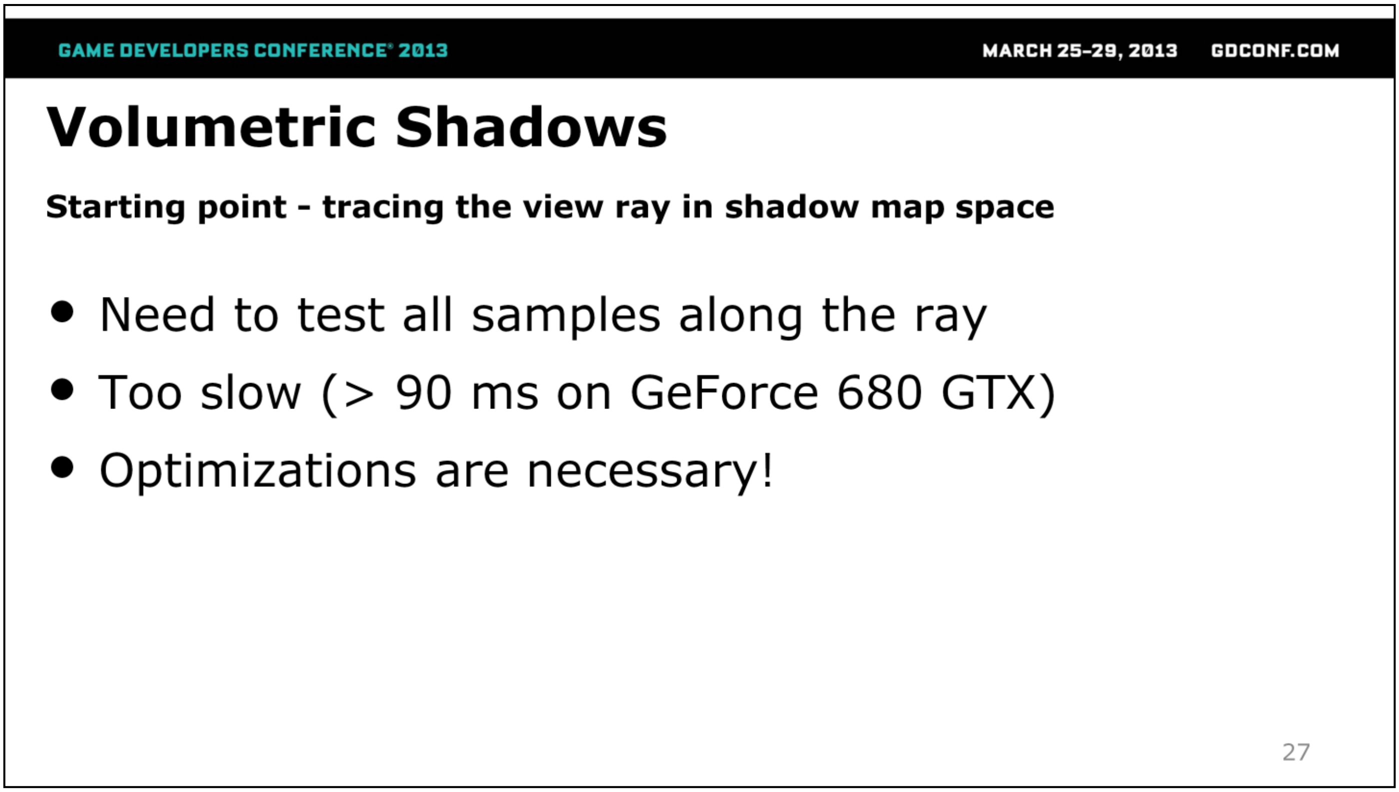
Volumetric Shadows

Perspective-correct interpolation

$$\frac{1}{z} = \frac{1}{z_C} + \alpha\left(\frac{1}{z_O} - \frac{1}{z_C}\right)$$

$$\frac{S}{z} = \frac{S_C}{z_C} + \alpha\left(\frac{S_O}{z_O} - \frac{S_C}{z_C}\right)$$

An important aspect of this algorithm is that we perform ray marching in the light projection space. Thus we can not directly interpolated attributes like distance along the ray. However, there is simple formula which tells how to do this correctly. We need to divide each attribute by the light view space z, interpolate and then divide by interpolated light view space z.

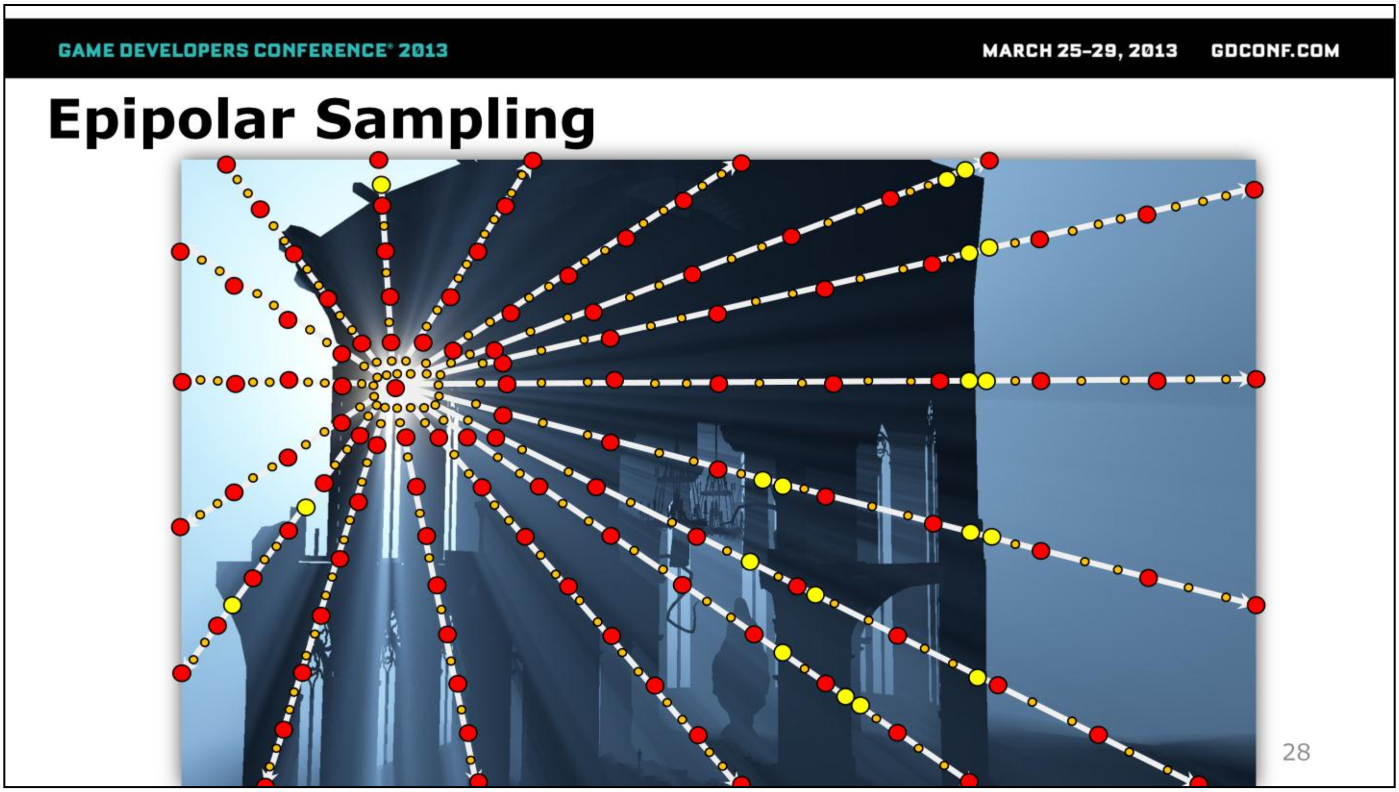


Volumetric Shadows

Starting point - tracing the view ray in shadow map space

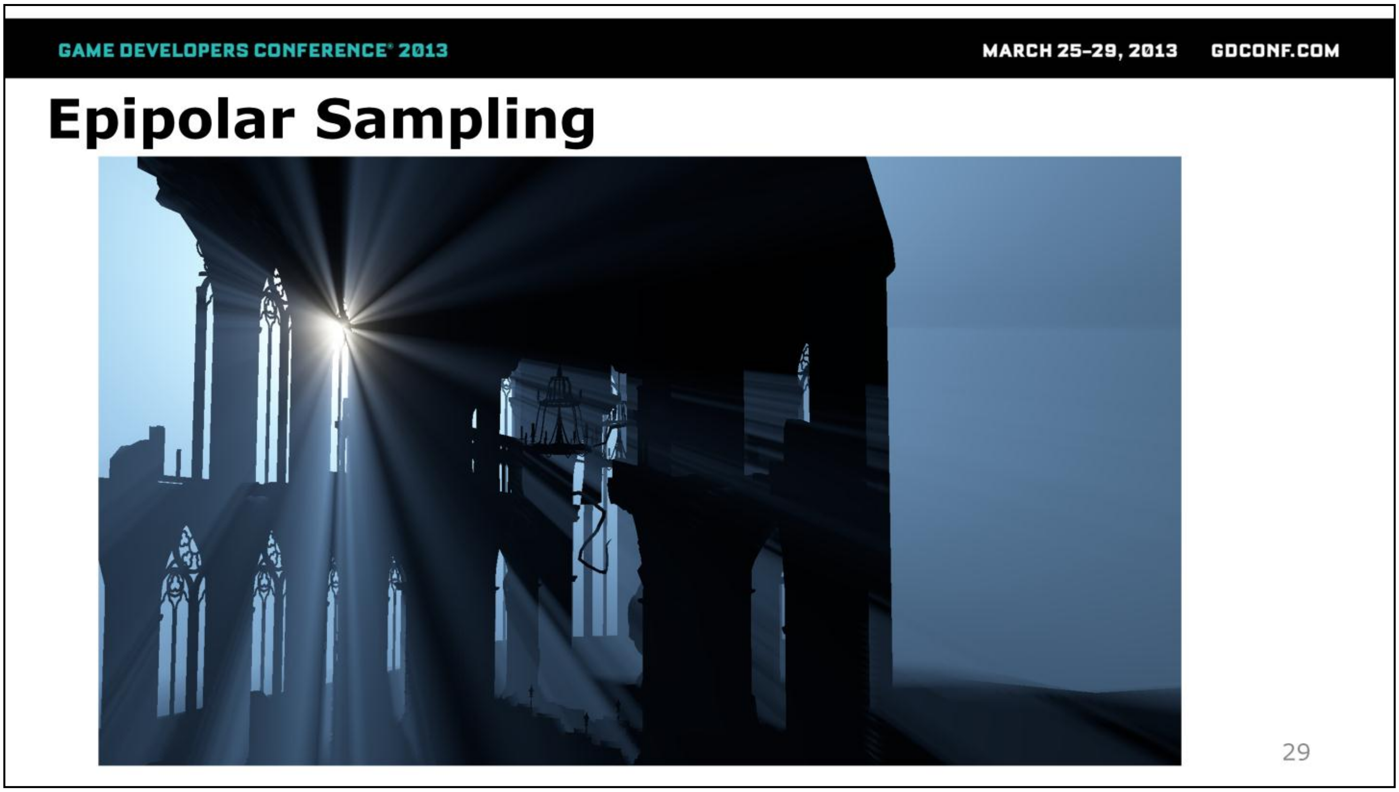
- Need to test all samples along the ray
- Too slow (> 90 ms on GeForce 680 GTX)
- Optimizations are necessary!

Ok, this algorithm generates nice visual results, but we have to execute it for each screen pixel and go through all shadow map texels. Even on high-end discrete GPU such implementation requires enormous amount of time, so optimizations are necessary.



Epipolar Sampling

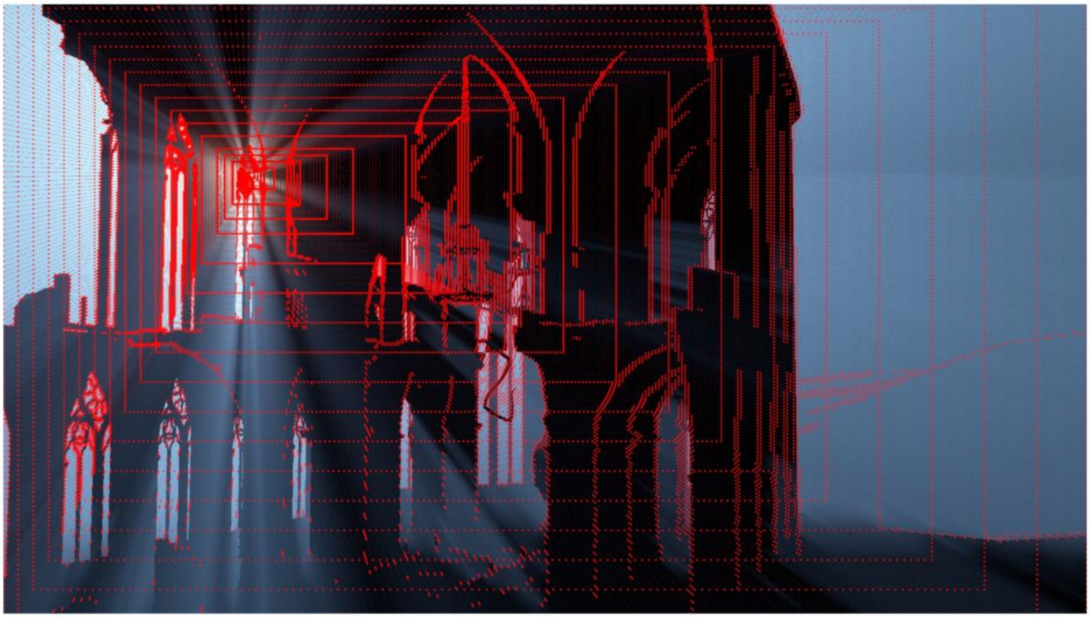
- The first optimization is based on the observation that light shafts seen on the screen have special structure: they all emanate radially from the position of the light on the screen. Engelhardt and Dachsbacher noticed that the inscattered light varies orthogonally to these rays, but mostly smoothly along them. To account for this property they proposed placing samples along the epipolar lines
- To catch low frequency variation, it is sufficient to sparsely locate initial ray marching samples
- Since inscattering light intensity varies abruptly at depth discontinuities, additional ray marching samples are necessary at depth breaks to catch high frequency details
- We can then compute computationally expensive ray marching for the selected number of samples. Since light intensity varies smoothly along the rays, for all the remaining samples, the intensity can be linearly interpolated from the nearby ray marching samples
- After that we can transform inscattering from epipolar coordinates back to rectangular



Epipolar Sampling

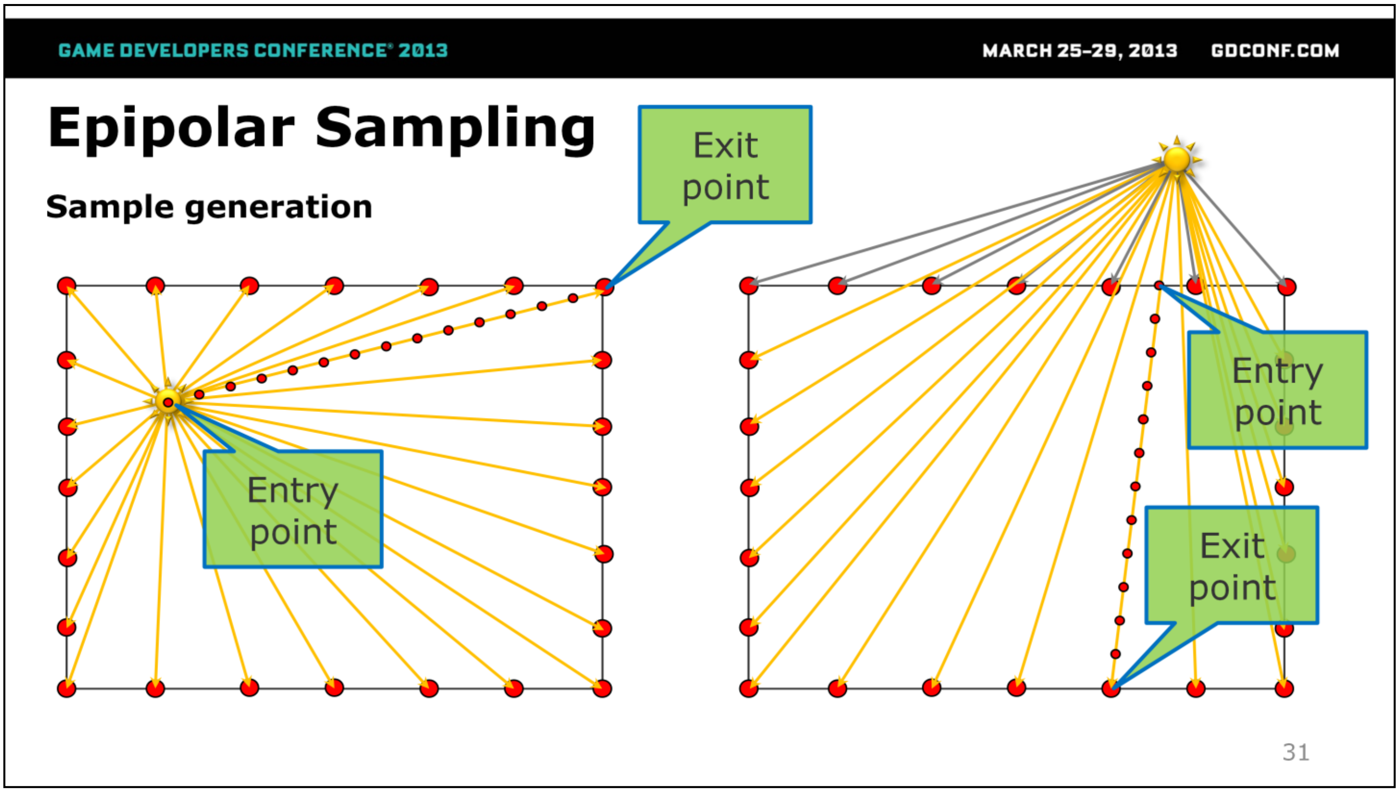
For this particular picture the epipolar sampling would look like this:

Epipolar Sampling



Note that intentionally lower number of samples is used here to show the sampling structure

Notice the regularly spaced rectangles which are initially placed ray marching samples and additional samples placed at depth discontinuities



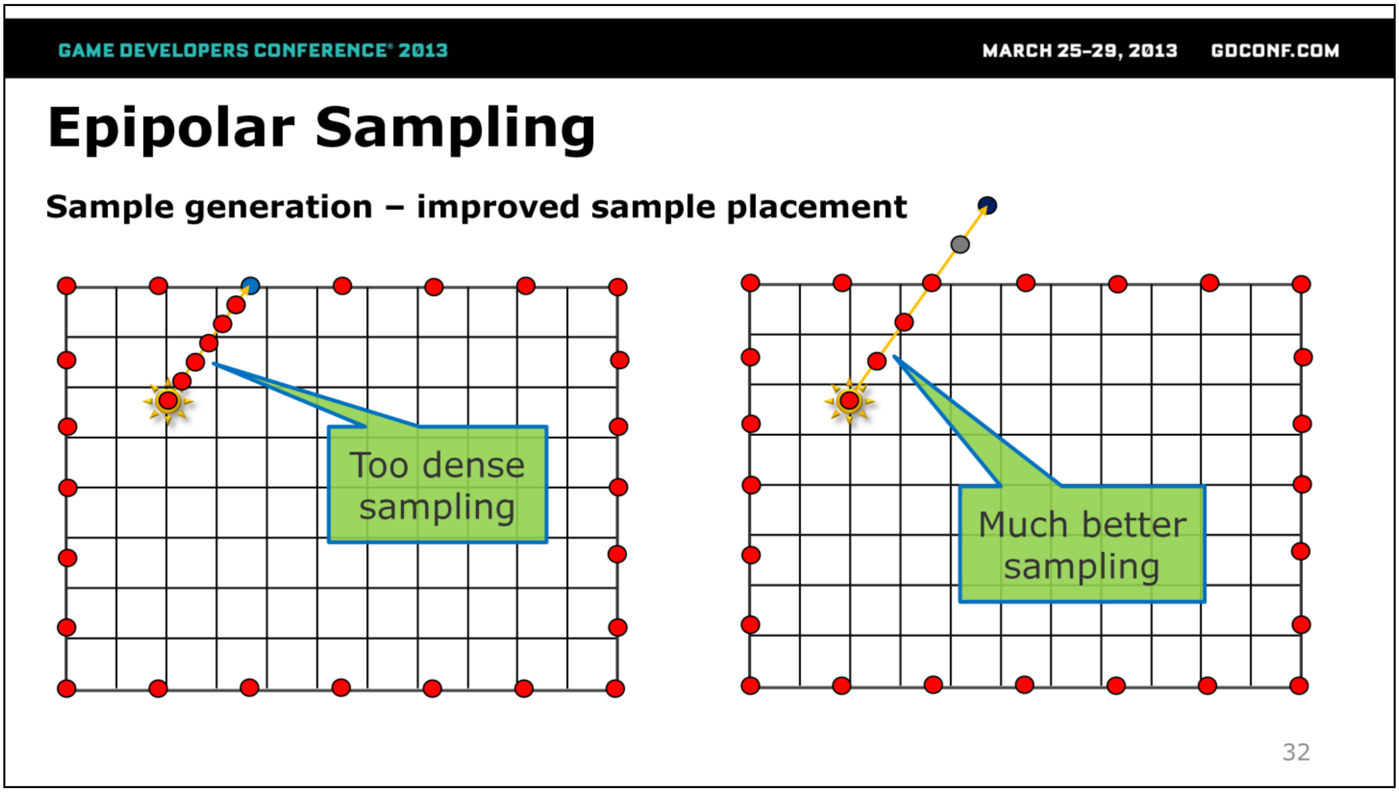
Epipolar Sampling

Sample generation

The first stage of the epipolar sampling algorithm is sample generation.

- If the projected light source is on the screen, then epipolar line generation is done by
  - equidistantly placing a user-defined number of exit points along the border of the screen and
  - connecting the light position (which is in this case the entry point for each line) with these points
  - A predefined number of samples are then evenly placed between entry and exit points of each line.
- If the light source is located outside the screen, then some lines are completely outside the visible area and are culled prior to the subsequent steps
  - The remaining lines are truncated against the screen borders and samples are placed between entry and exit points.

If the light source is behind the camera, its projected position is the point where rays converge in infinity. Sample generation in this case is done in exactly the same way.



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

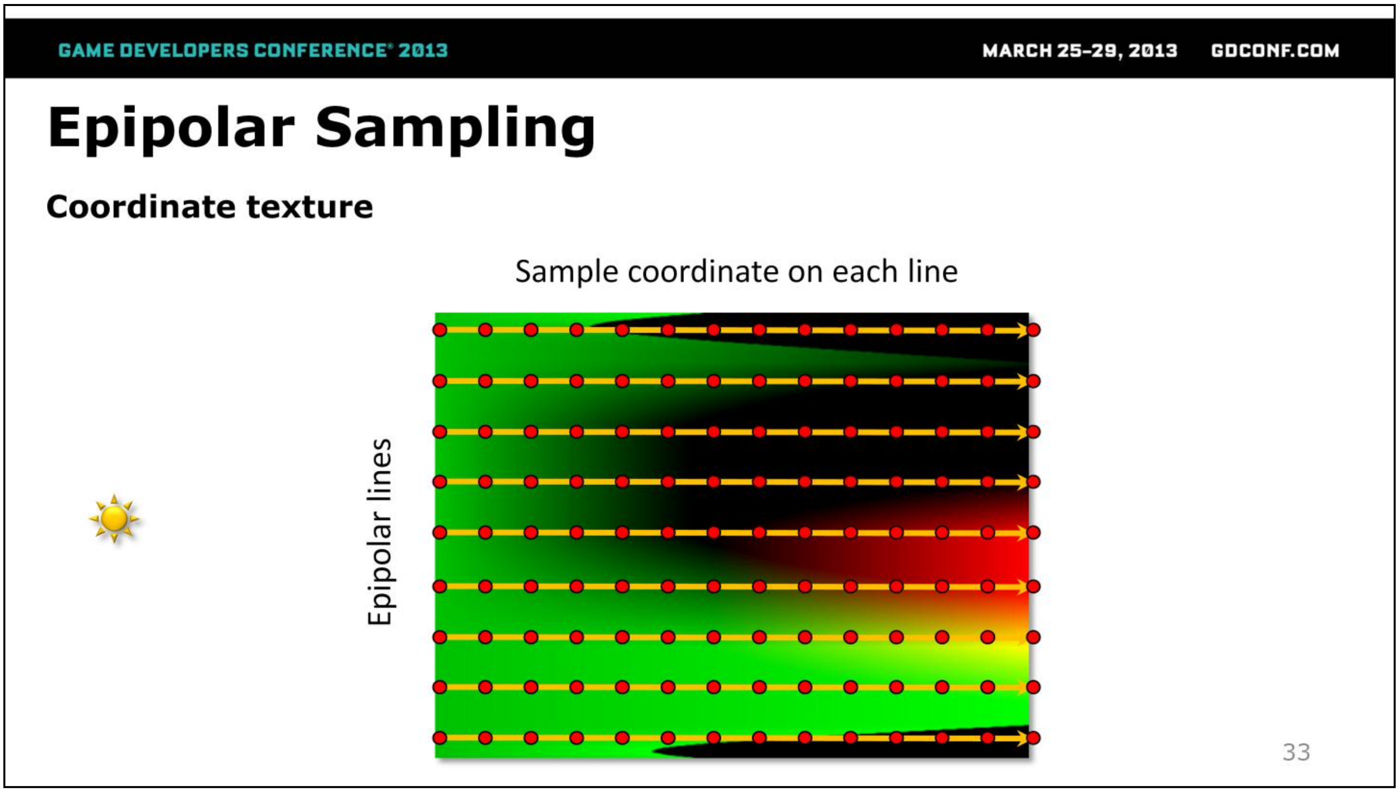
Epipolar Sampling

Sample generation – improved sample placement

Too dense sampling

Much better sampling

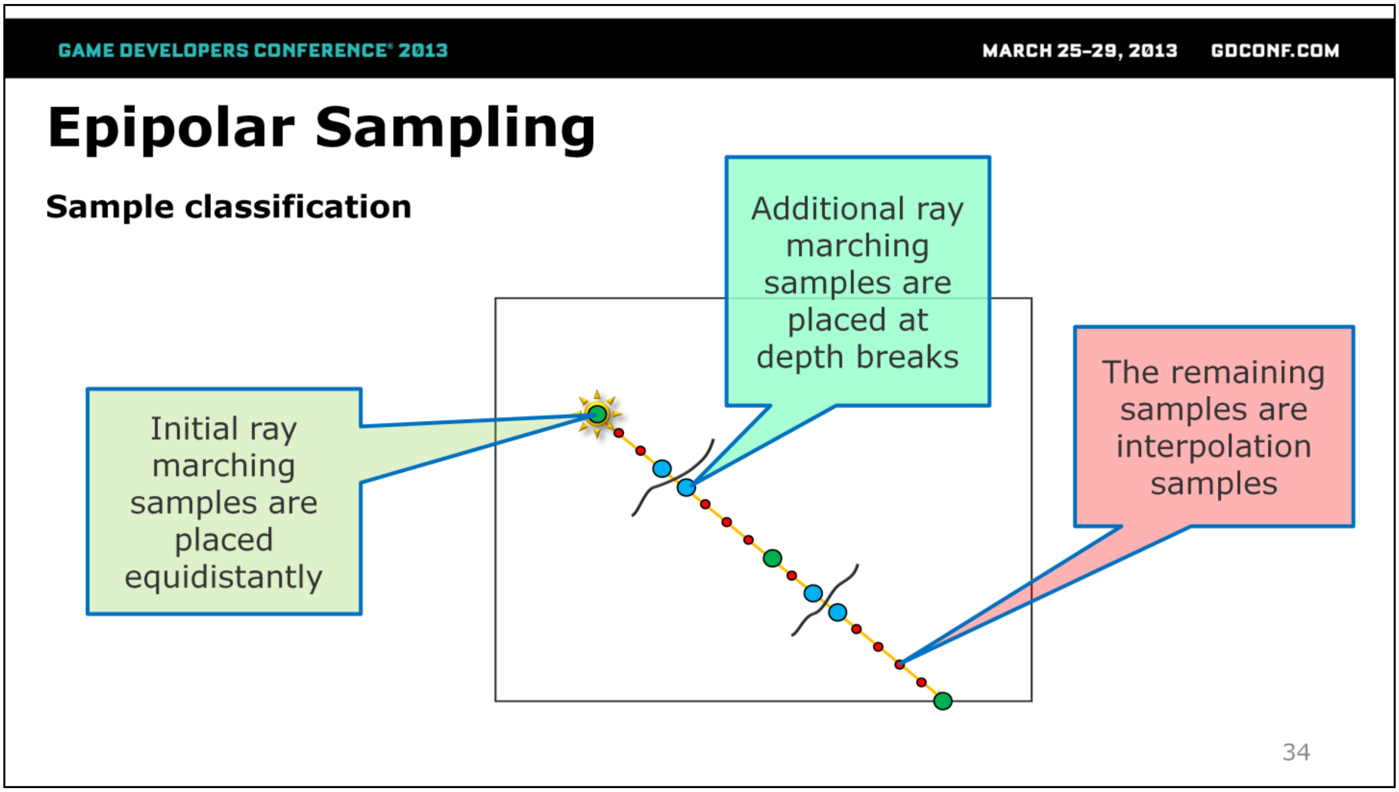
- If the light source is located close to the screen boundary, then screen length of epipolar lines could vary significantly. This will result in using too dense sampling for short lines and doing redundant computations
- To solve this issue, we rescale the epipolar lines by advancing the exit point, striving to provide 1:1 correspondence between samples on the line and screen pixels. This not only reduces the amount of computations, but also results in a more natural circular-shaped distribution of samples against a rectangular-shaped distribution in the base method



Epipolar Sampling

Coordinate texture

Screen space coordinates of each sample are stored in a two-dimensional texture. Each row of this texture corresponds to one epipolar line while each column corresponds to one location on the line.



Epipolar Sampling

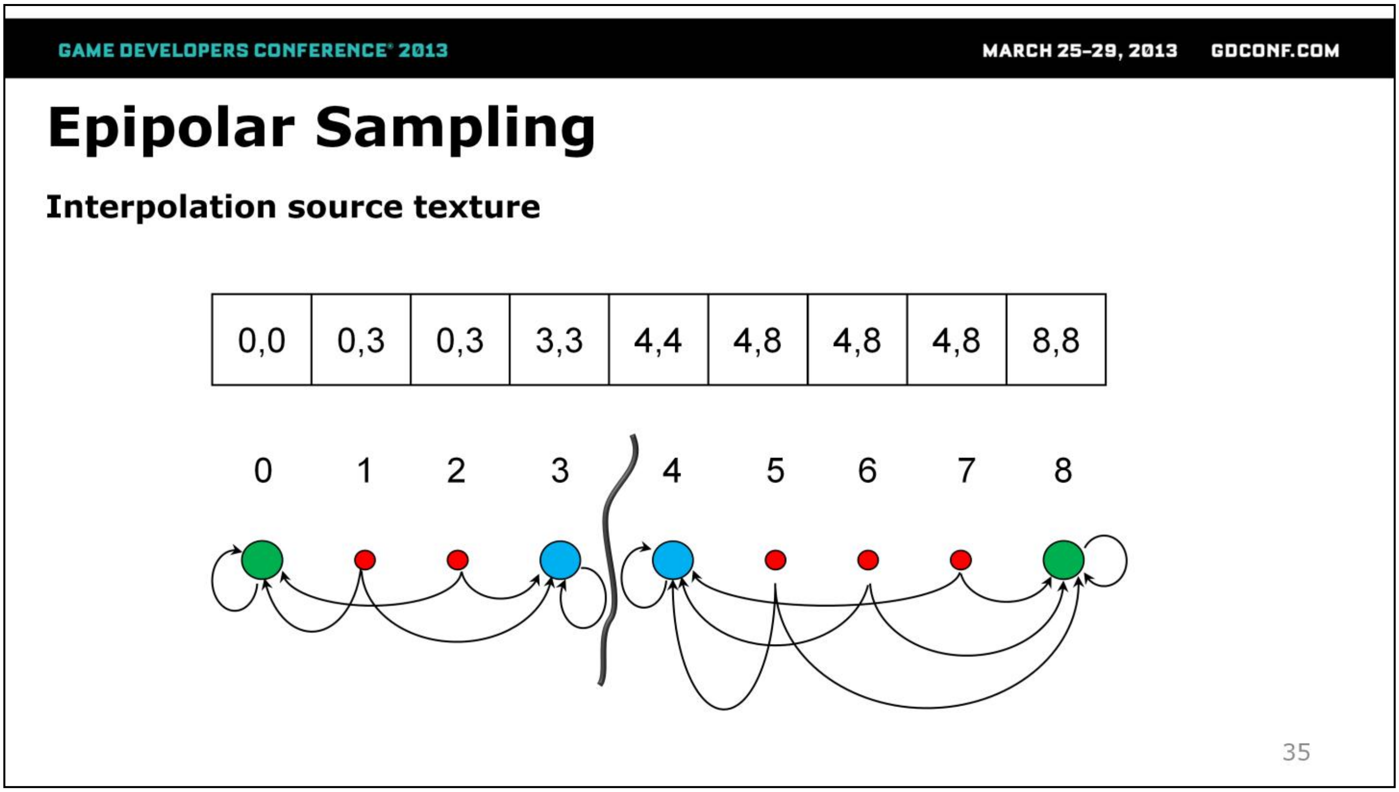
Sample classification

Initial ray marching samples are placed equidistantly

Additional ray marching samples are placed at depth breaks

The remaining samples are interpolation samples

- After samples are generated, initial ray marching samples are equidistantly placed along each epipolar line to catch low-frequency variations.
- After that depth breaks are detected in each line and additional samples are placed directly before and after the break
- The remaining samples are interpolation samples and their intensity is computed by linear interpolation between closest ray-marching samples

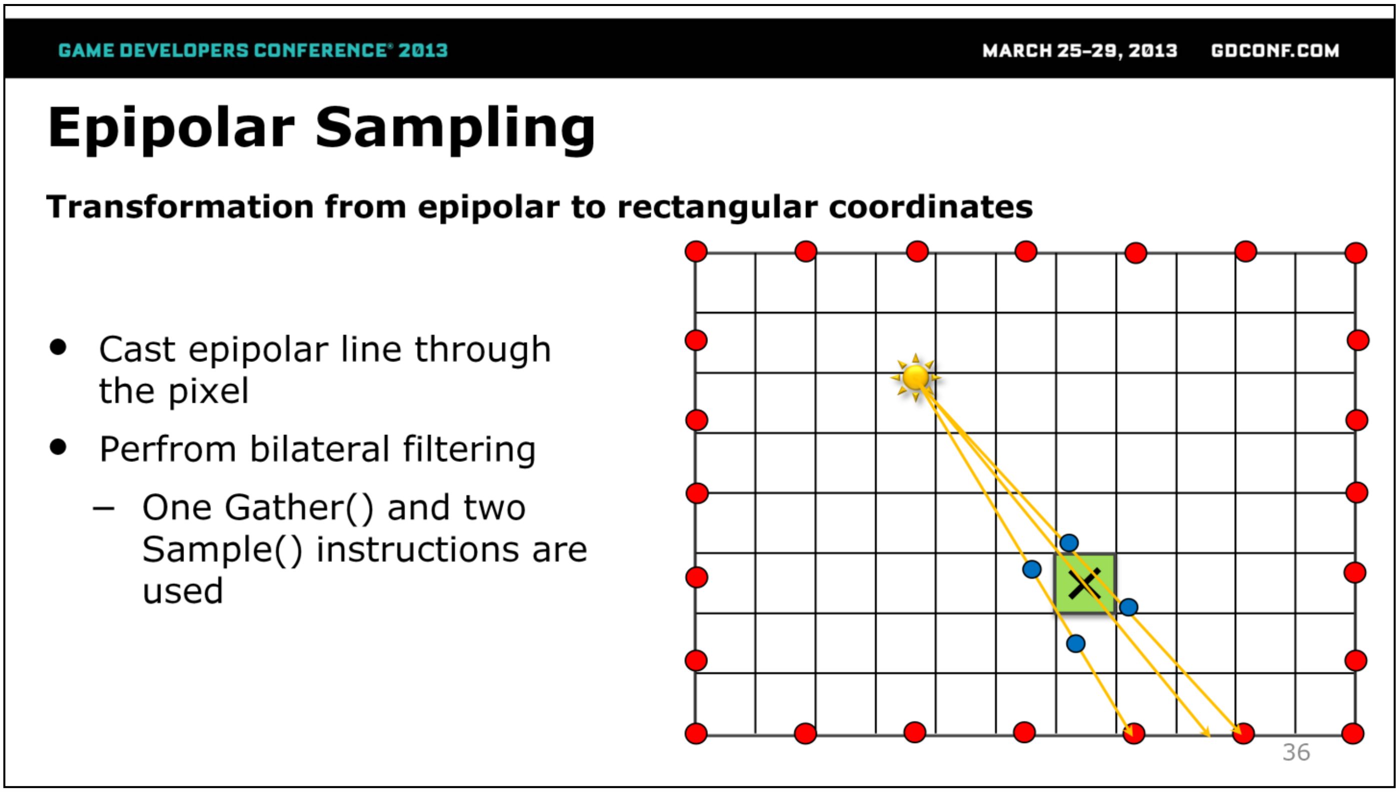


Epipolar Sampling

Interpolation source texture

0,0	0,3	0,3	3,3	4,4	4,8	4,8	4,8	8,8

- To alleviate interpolation step, an auxiliary two-channel integer texture is used. This texture contains indices of two closest ray marching samples from the same slice, from which the inscattered light is interpolated. Ray marching samples are marked to be interpolated from themselves.



Epipolar Sampling

Transformation from epipolar to rectangular coordinates

- Cast epipolar line through the pixel
- Perfrom bilateral filtering
  - One Gather() and two Sample() instructions are used

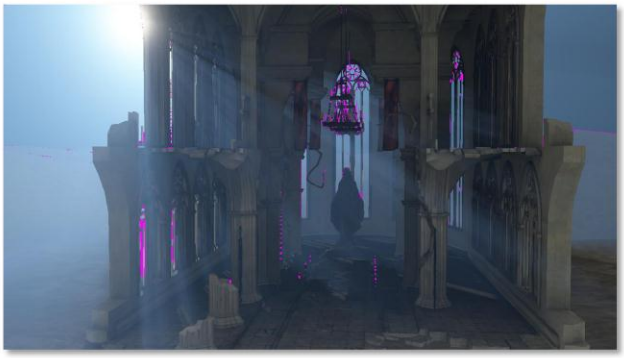
After the inscattering integral is calculated by ray marching or interpolation for each epipolar sample, inscattering is computed for each screen pixel. This is done by the following steps:

- An epipolar line is cast through the pixel and location in epipolar texture is computed
- Bilateral filtering is performed
  - one Gather() for camera space z and two Sample() for inscattered light texture is used

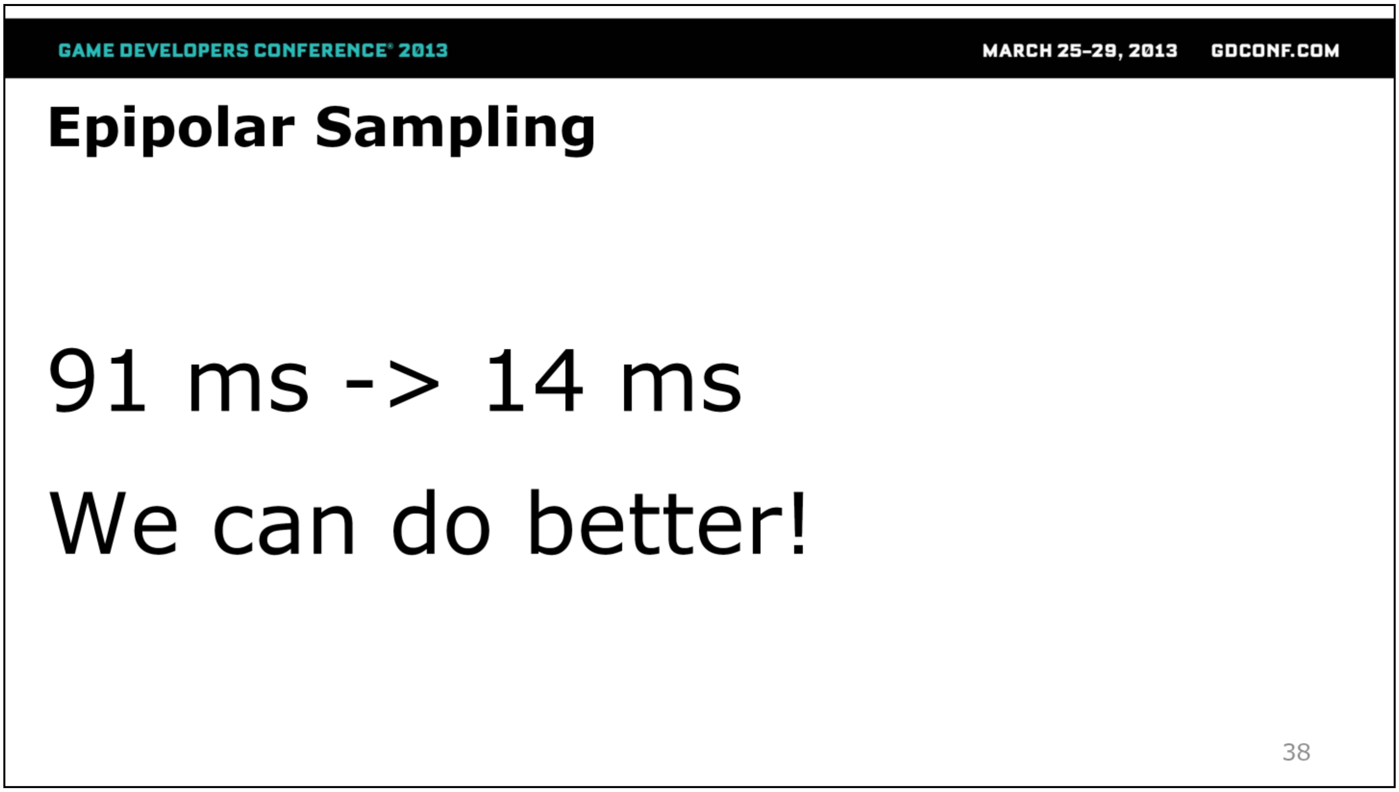
Epipolar Sampling

Fix-up pass

- Not all samples can be properly interpolated from epipolar coordinates
- These samples are marked in stencil
- Additional fix-up pass is performed without 1D min/max acceleration



- It is also possible that there are no appropriate samples to filter from.
- We mark such samples in stencil and
- perform an additional ray marching pass for these samples using fewer steps and no min/max optimization



Epipolar Sampling

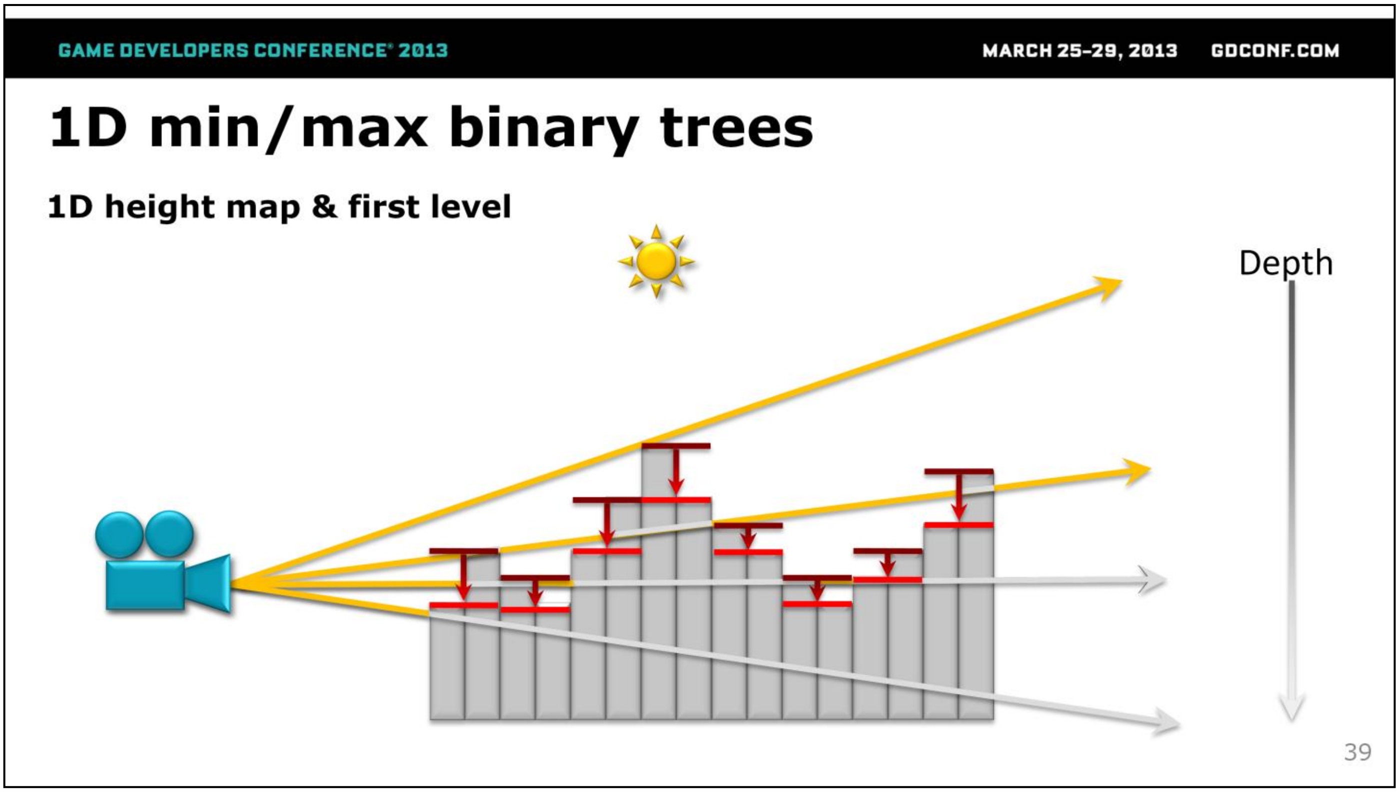
91 ms -> 14 ms

We can do better!

So epipolar sampling dramatically reduced the computation time, but we still can do better.

Stepping through each shadow map texel is still too expensive, especially for high shadow map resolution while using constant number of samples can cause undersampling and result in banding artifacts.

Let’s now talk about how we can improve inscattering integral calculation for each ray marching sample.



1D min/max binary trees

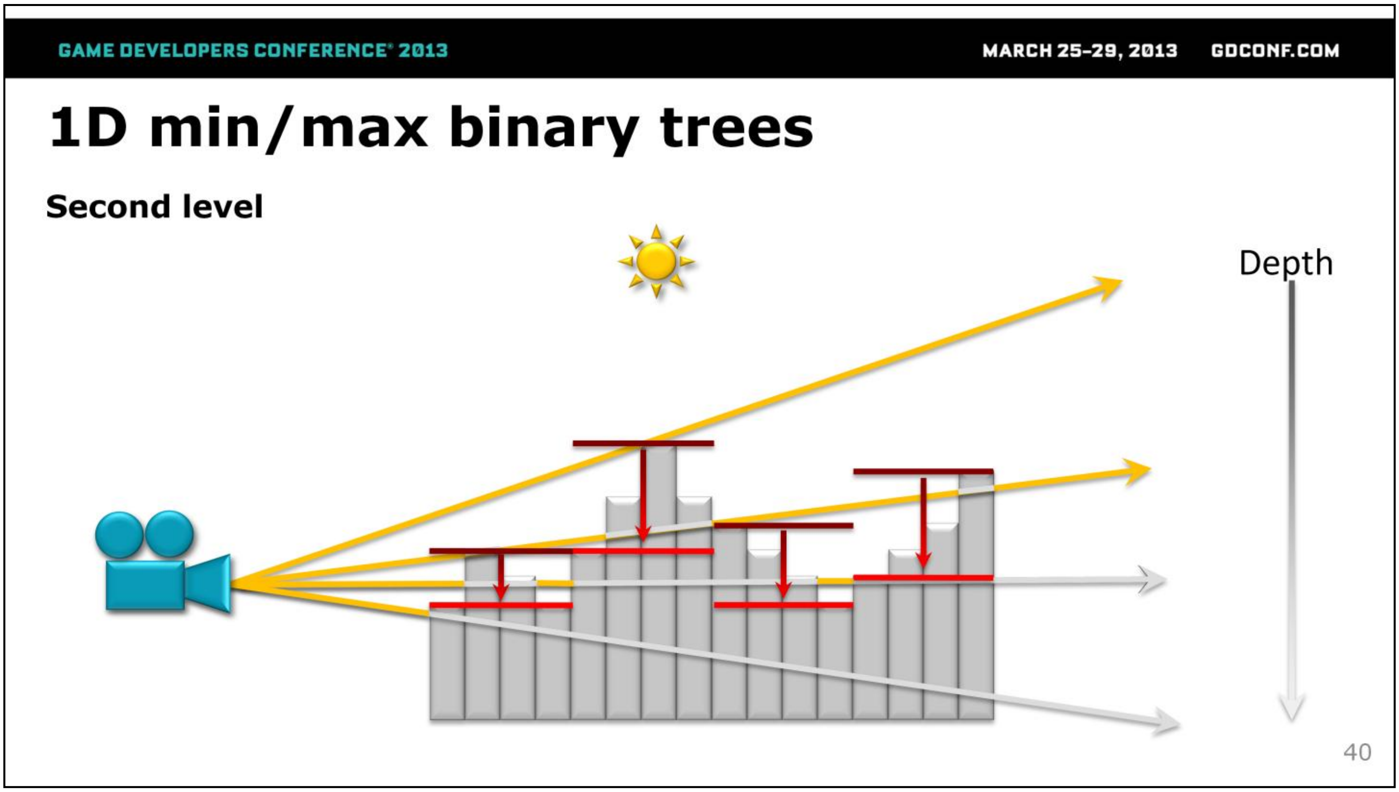
1D height map & first level

Epipolar sampling has one important property: all camera rays in an epipolar slice share the same plane. Intersection of this plane with the shadow map essentially forms a one-dimensional height map. Shadow test is intrinsically a check if current position on the ray is under this height map or above it.

This property was first recognized by Chen et al who proposed constructing 1D min/max binary tree for each epipolar slice and using this structure to identify long lit and shadowed regions on the ray

- These are the nodes of the first level of the tree

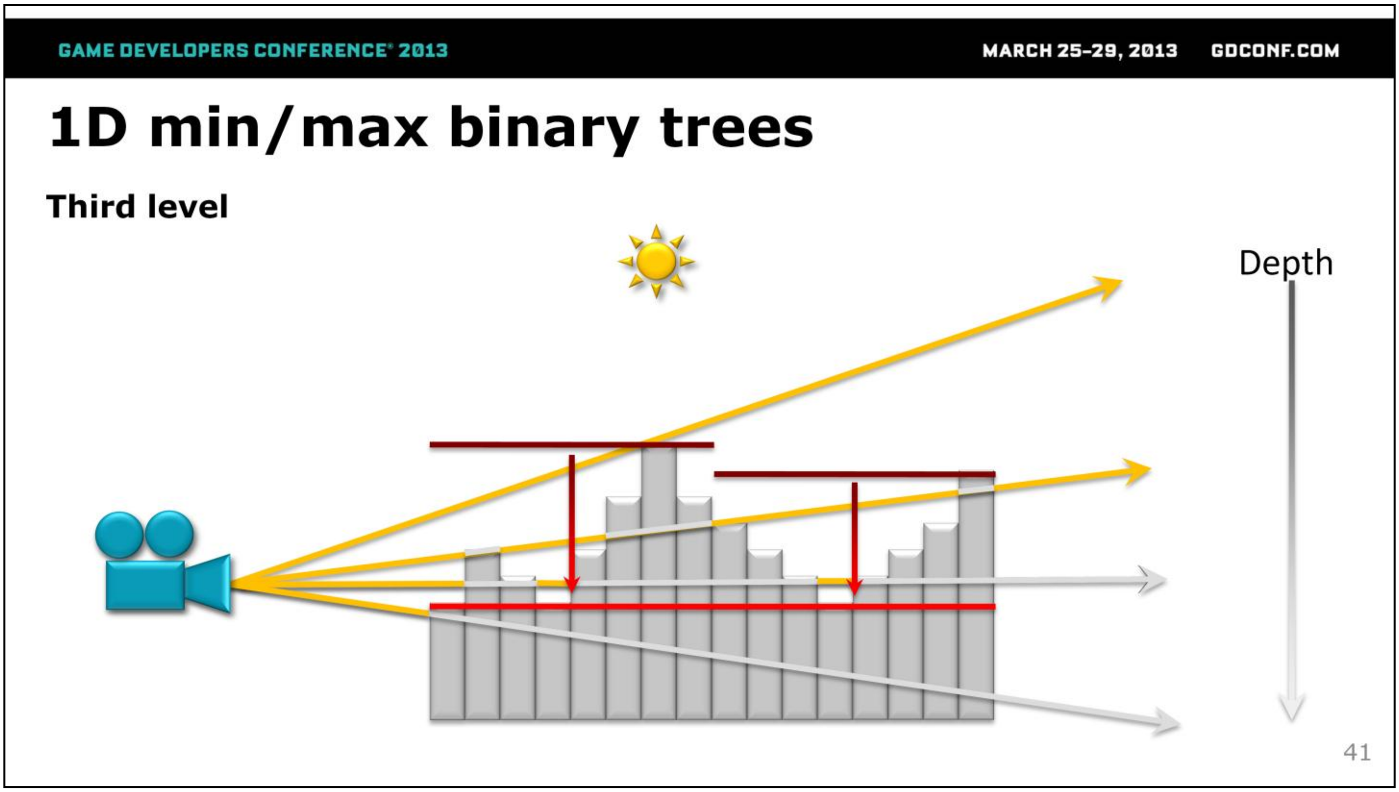
explain what the problem is, why this helps and why it is good



1D min/max binary trees

Second level

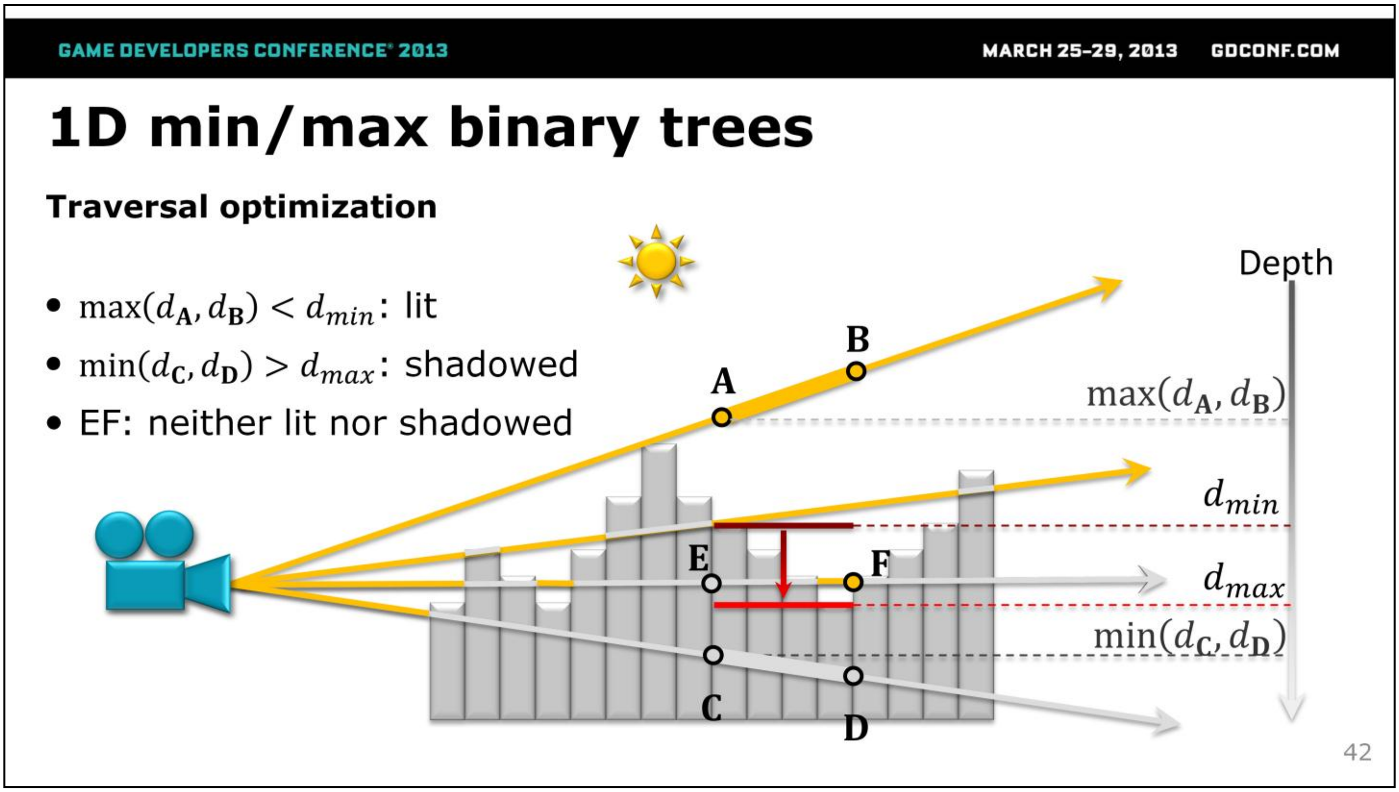
These are the nodes of the second level



1D min/max binary trees

Third level

And here are the nodes of the third level



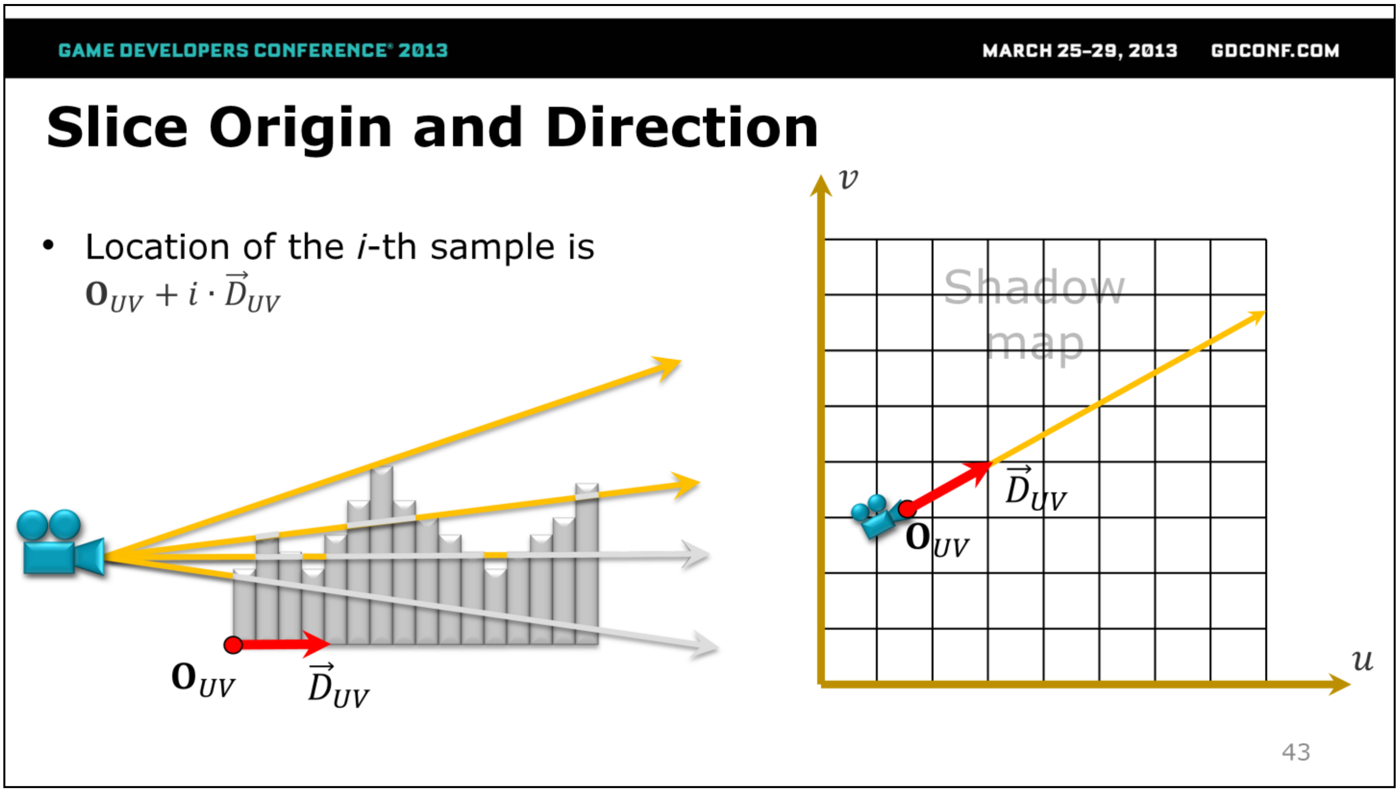
1D min/max binary trees

Traversal optimization

- $\max(d_{\mathbf{A}}, d_{\mathbf{B}}) < d_{min}$: lit
- $\min(d_{\mathbf{C}}, d_{\mathbf{D}}) > d_{max}$: shadowed
- EF: neither lit nor shadowed

Depth
A
B
$\max(d_{\mathbf{A}}, d_{\mathbf{B}})$
d_{min}
E
F
d_{max}
$\min(d_{\mathbf{C}}, d_{\mathbf{D}})$
C
D

- Now, if the maximum value of depths at the ends of the current ray section is less than the minimum depth stored in the min/max tree, then this section is completely lit
- Alternatively, if the minimum value of depths at the ends is greater than the maximum value stored in the tree, then the section is fully in shadow and we can skip it
- It is also possible that neither condition is true. In this case, it is necessary to repeat the test at the next finer tree level



Slice Origin and Direction

- Location of the *i*-th sample is $\mathbf{O}_{UV} + i \cdot \vec{D}_{UV}$

The next question is how to construct these min/max binary trees. For this we need to know from what point and in what direction to perform ray marching.

For all rays in an epipolar slice, ray marching starts from the same position in the shadow map and proceeds in the same direction.

If we know origin and direction, we can compute shadow map location of any sample in the slice

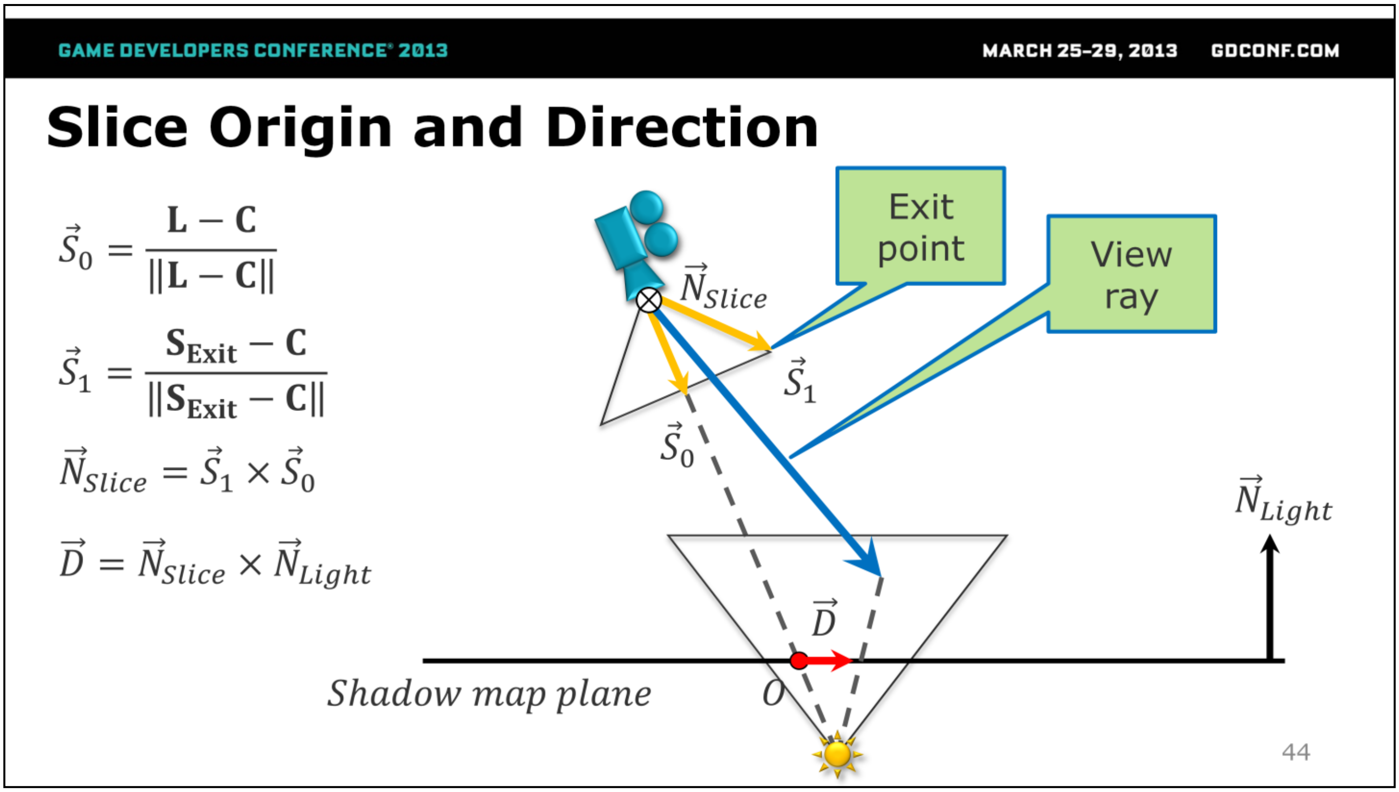
Slice Origin and Direction

$$\vec{S}_0 = \frac{\mathbf{L}-\mathbf{C}}{\|\mathbf{L}-\mathbf{C}\|}$$

$$\vec{S}_1 = \frac{\mathbf{S}_{\mathbf{Exit}}-\mathbf{C}}{\|\mathbf{S}_{\mathbf{Exit}}-\mathbf{C}\|}$$

$$\vec{N}_{Slice} = \vec{S}_1 \times \vec{S}_0$$

$$\vec{D} = \vec{N}_{Slice} \times \vec{N}_{Light}$$



Now, how can we find slice origin and direction.

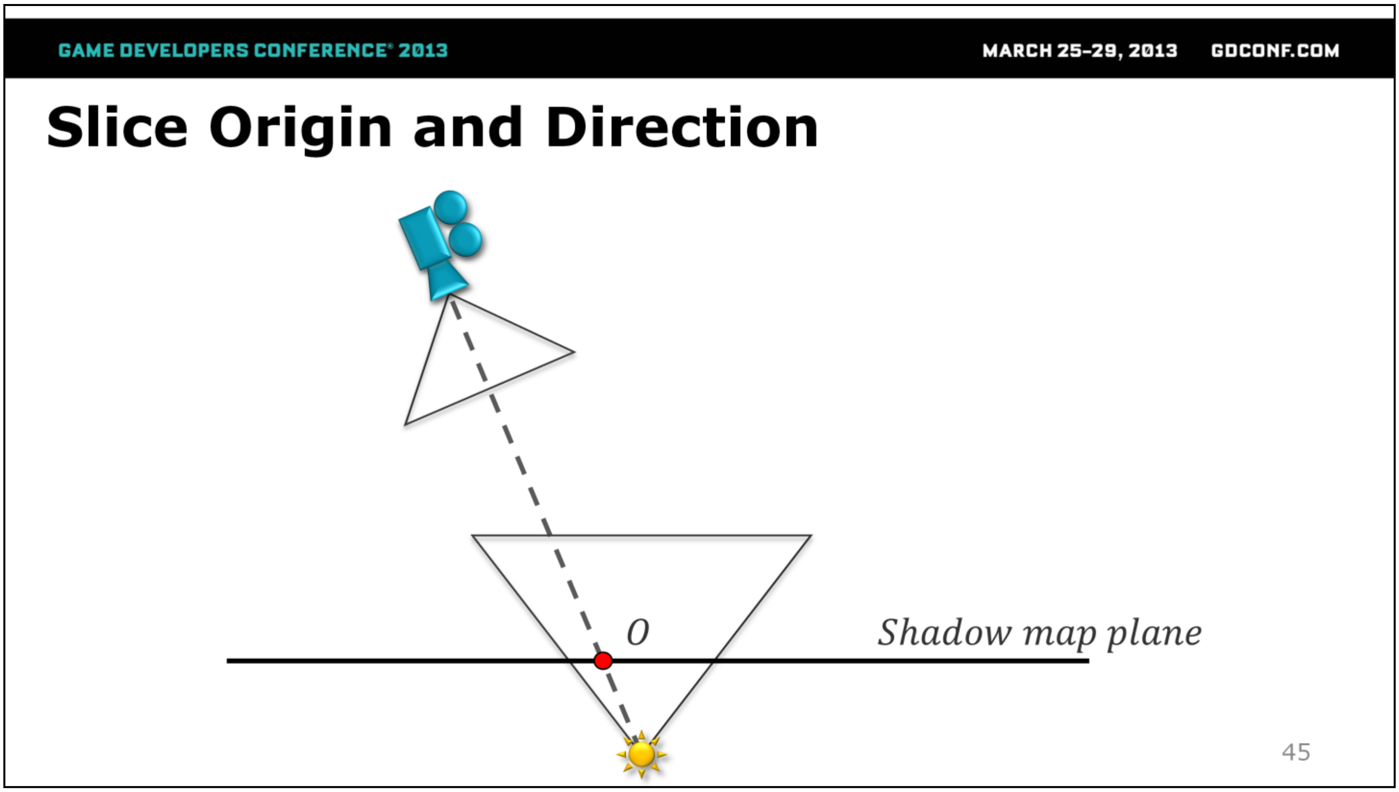
For a directional light source, origin can be computed by simply transforming the camera world space position **C** with the shadow map transform matrix. For spot light source, this method will only work if the camera is located inside the light cone. If the camera is located behind the light projection plane, or if it lies on it, the method will fail.

We thus developed a universal solution which works for both directional and spot light sources for any camera position and orientation. This method first finds intersection of the epipolar plane with the shadow map projection plane in world space, and then projects this vector onto the shadow map.

The first step of this method is to define epipolar slice plane in the world space. The camera obviously lies in all slices, so we need to find normal to the plane. For this, we need two vectors $\vec{S}_0$ and $\vec{S}_1$ in the plane.

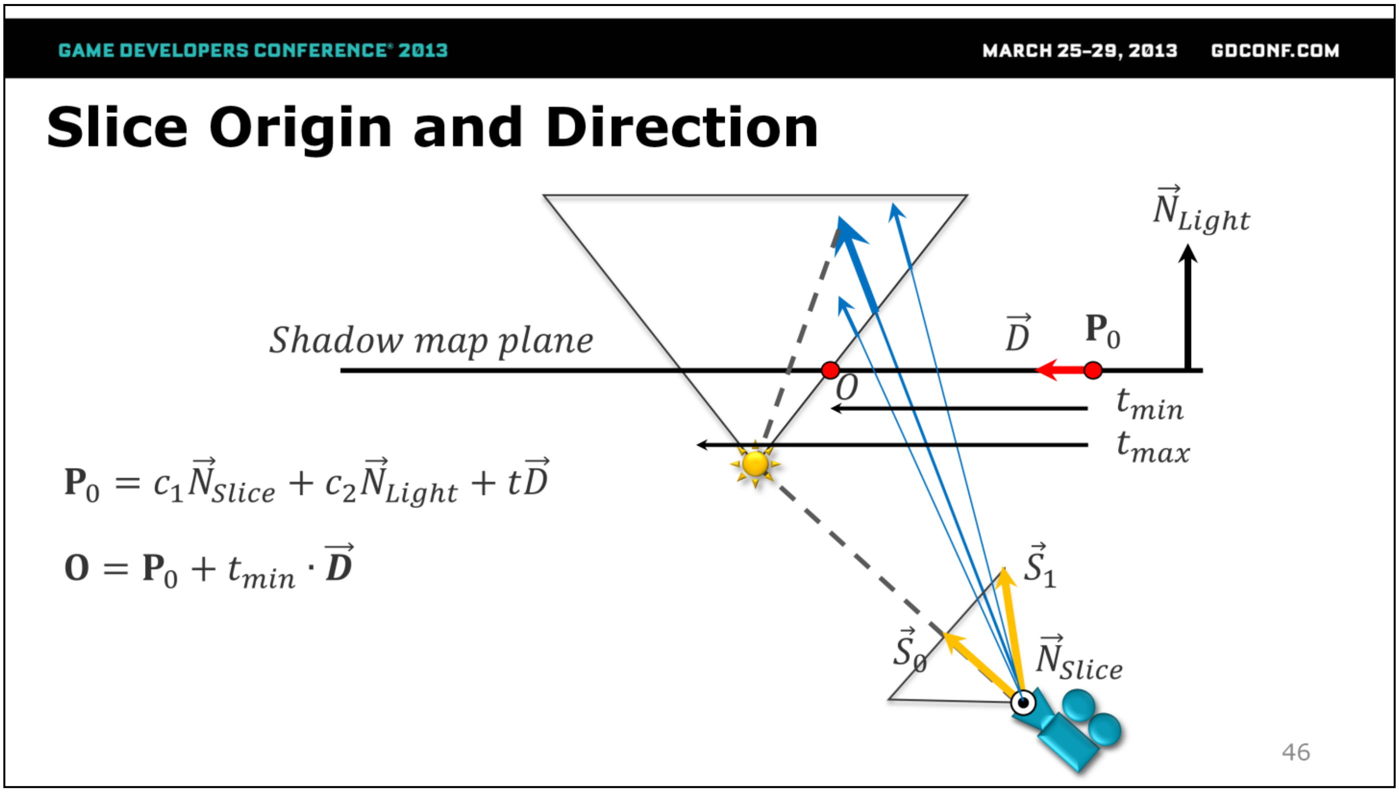
- The first vector is the direction from the camera to light source

- The second vector can be found by reconstructing epipolar slice exit point world space position $\mathbf{S}_{\mathbf{Exit}}$ and taking a vector from the camera through this point
- Epipolar plane normal can be computed as a cross product of S1 and S0
- Finally, slice direction D is the intersection of two planes and thus belongs to both. This means it is orthogonal to both plane normals. Thus it can be computed as a cross product of these vectors.



Slice Origin and Direction

So now we know slice direction D and need to find slice origin O. If the camera is inside the light cone, the origin is the same for all slices and is the projection of the camera position onto the shadow map plane.



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

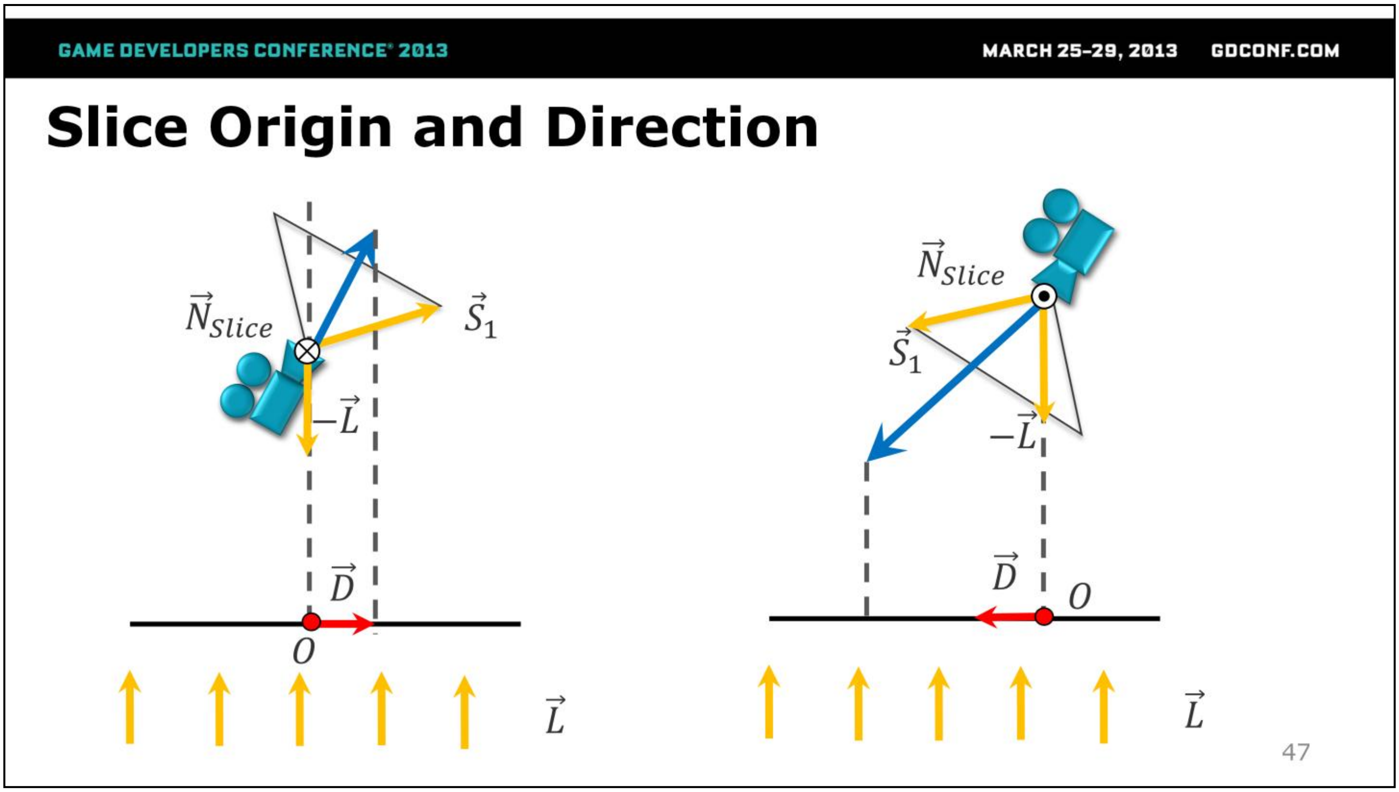
Slice Origin and Direction

$$\mathbf{P}_0 = c_1\vec{N}_{Slice} + c_2\vec{N}_{Light} + t\vec{D}$$

$$\mathbf{O} = \mathbf{P}_0 + t_{min} \cdot \vec{\boldsymbol{D}}$$

If the camera is located outside the cone, all the rays are truncated against the cone.

- Note that all camera rays in a slice hit the same cone side. As a result, for all the rays, ray marching starts from the same point which is the projection of this cone side onto the shadow map plane
- To find this point, we first find some point $\mathbf{P}_0$ on the ray in the following form
- Next, we compute two intersections of the ray with the light cone. If there are no intersections, the ray misses the cone and the ray marching thus will not be performed for all rays in this slice
- The slice origin is then computed as $\mathbf{O} = \mathbf{P}_0 + t_{min} \cdot \vec{\boldsymbol{D}}$

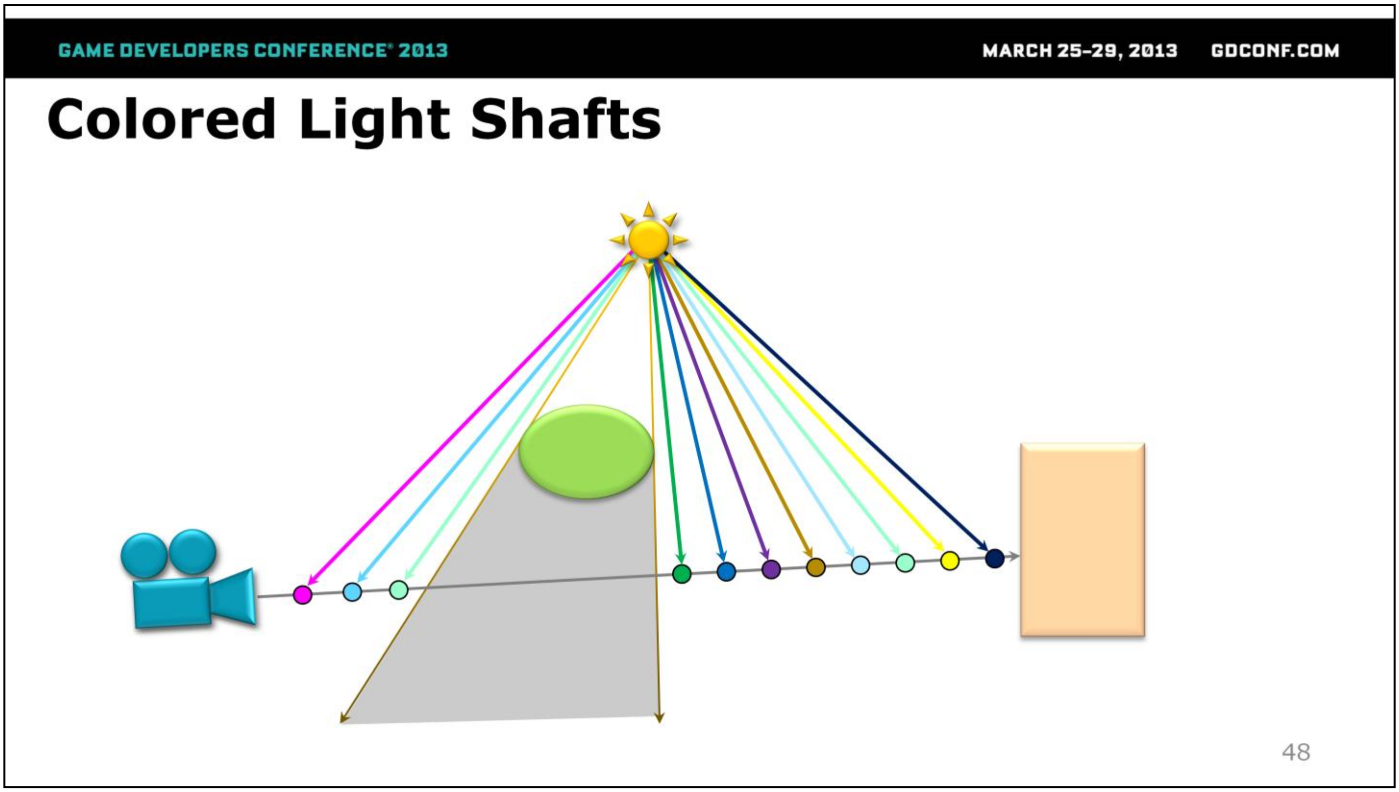


Slice Origin and Direction

The same method works for directional light source if we replace S0 with the negated light direction

To obtain slice origin $\mathbf{O}_{UV}$ in shadow map UV coordinates, it is necessary to transform $\mathbf{O}$ with the shadow map transform matrix. Since $\mathbf{O}$ is guaranteed to lie on the light projection plane, it always has positive z coordinate, which assures the result is always correct. Ray marching starts from the same point which is the projection of this cone side onto the shadow map plane.

The direction $\vec{D}_{UV}$ in shadow map coordinates is then obtained by transforming $\vec{D}$ with the light transform matrix.

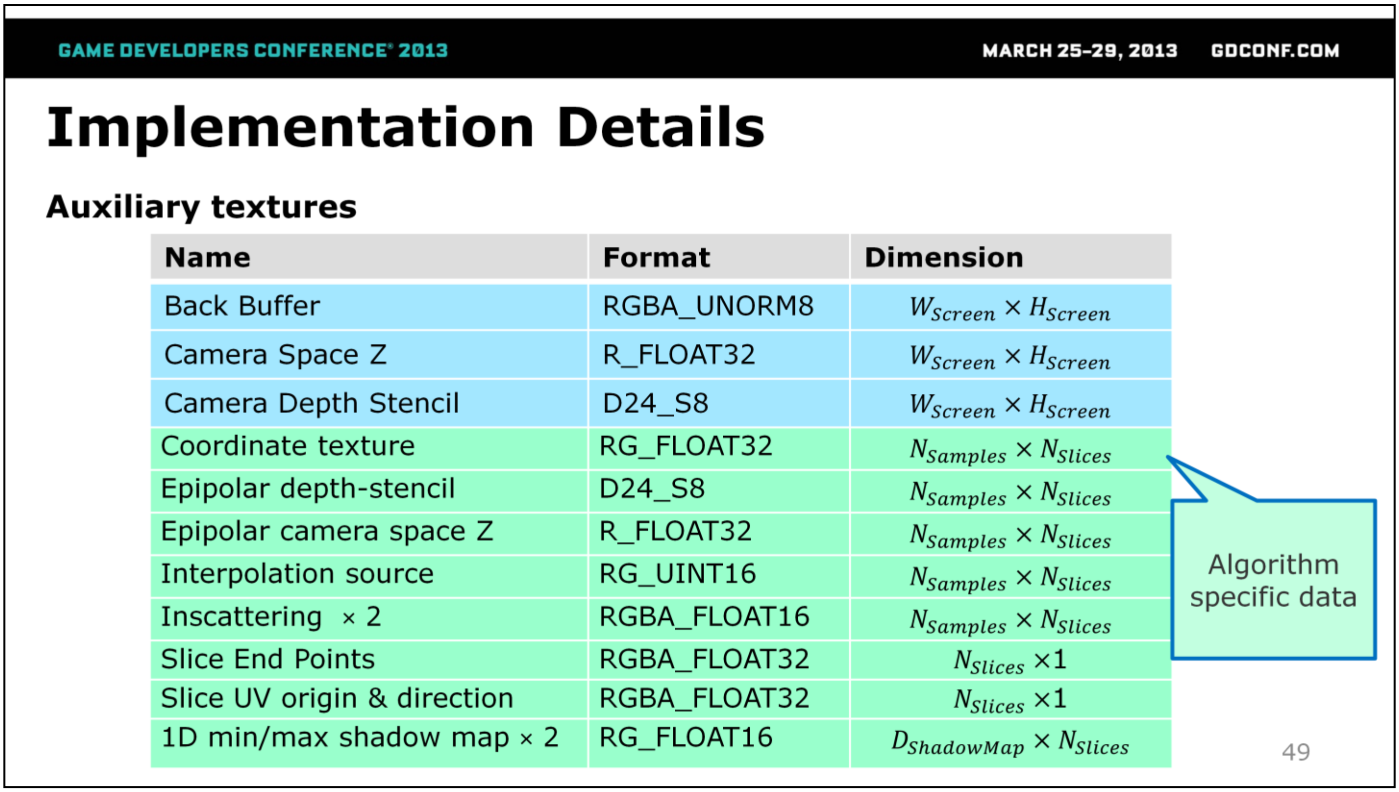


Colored Light Shafts

Our algorithm also supports colored light shafts.

- This is implemented by simply rendering the stained glass into the additional buffer and sampling the light color texture during the ray marching

The min/max optimization is much less efficient in this case since only long shadowed sections can be skipped, while all lit sections must be traversed with the fine step



Implementation Details

Auxiliary textures

Name	Format	Dimension
Back Buffer	RGBA_UNORM8	$W_{Screen} \times H_{Screen}$
Camera Space Z	R_FLOAT32	$W_{Screen} \times H_{Screen}$
Camera Depth Stencil	D24_S8	$W_{Screen} \times H_{Screen}$
Coordinate texture	RG_FLOAT32	$N_{Samples} \times N_{Slices}$
Epipolar depth-stencil	D24_S8	$N_{Samples} \times N_{Slices}$
Epipolar camera space Z	R_FLOAT32	$N_{Samples} \times N_{Slices}$
Interpolation source	RG_UINT16	$N_{Samples} \times N_{Slices}$
Inscattering × 2	RGBA_FLOAT16	$N_{Samples} \times N_{Slices}$
Slice End Points	RGBA_FLOAT32	$N_{Slices} \times 1$
Slice UV origin & direction	RGBA_FLOAT32	$N_{Slices} \times 1$
1D min/max shadow map × 2	RG_FLOAT16	$D_{ShadowMap} \times N_{Slices}$

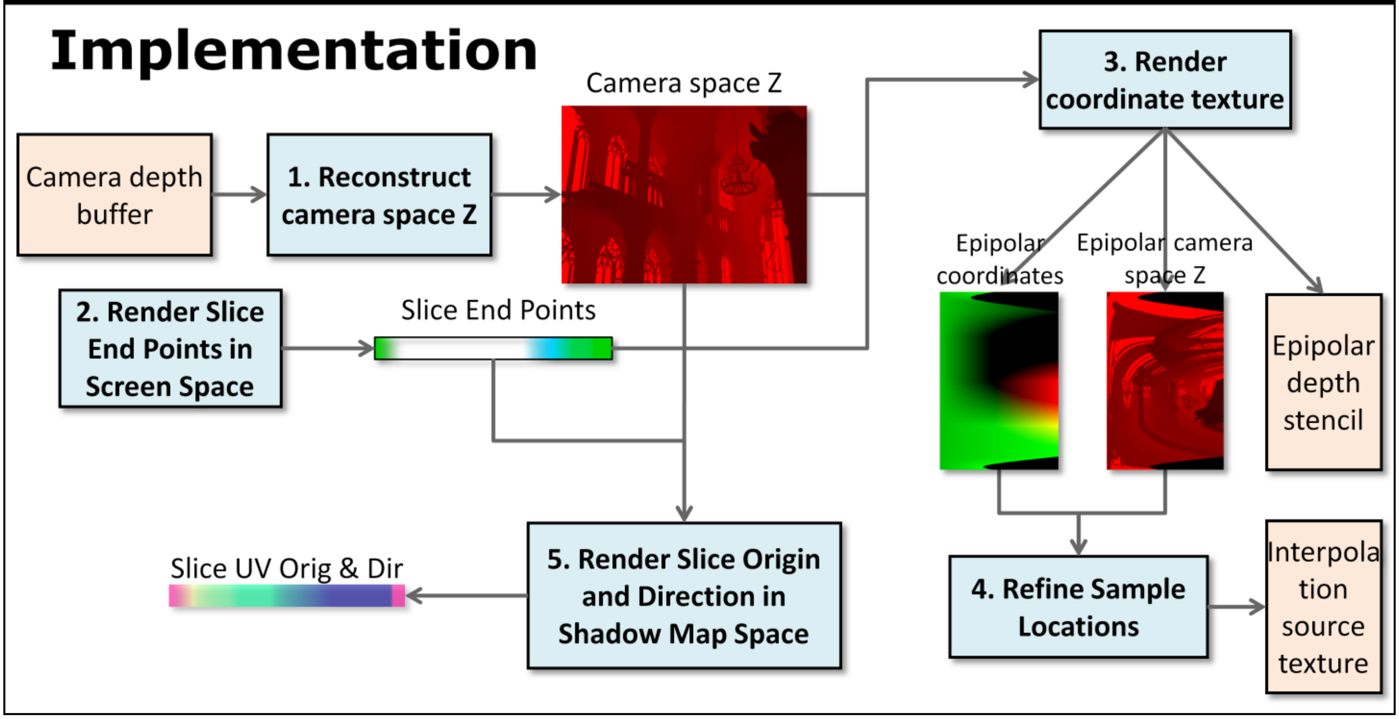
Algorithm specific data

There are a number of textures which are used to store intermediate data required during the processing, which are presented on the slide

The first three textures are back buffer, depth buffer and camera space z are not specific to the algorithm.

GAME DEVELOPERS CONFERENCE® 2013 MARCH 25–29, 2013 GDCONF.COM

Implementation



Camera depth buffer

1. Reconstruct camera space Z

Camera space Z

2. Render Slice End Points in Screen Space

Slice End Points

3. Render coordinate texture

Epipolar coordinates

Epipolar camera space Z

Epipolar depth stencil

4. Refine Sample Locations

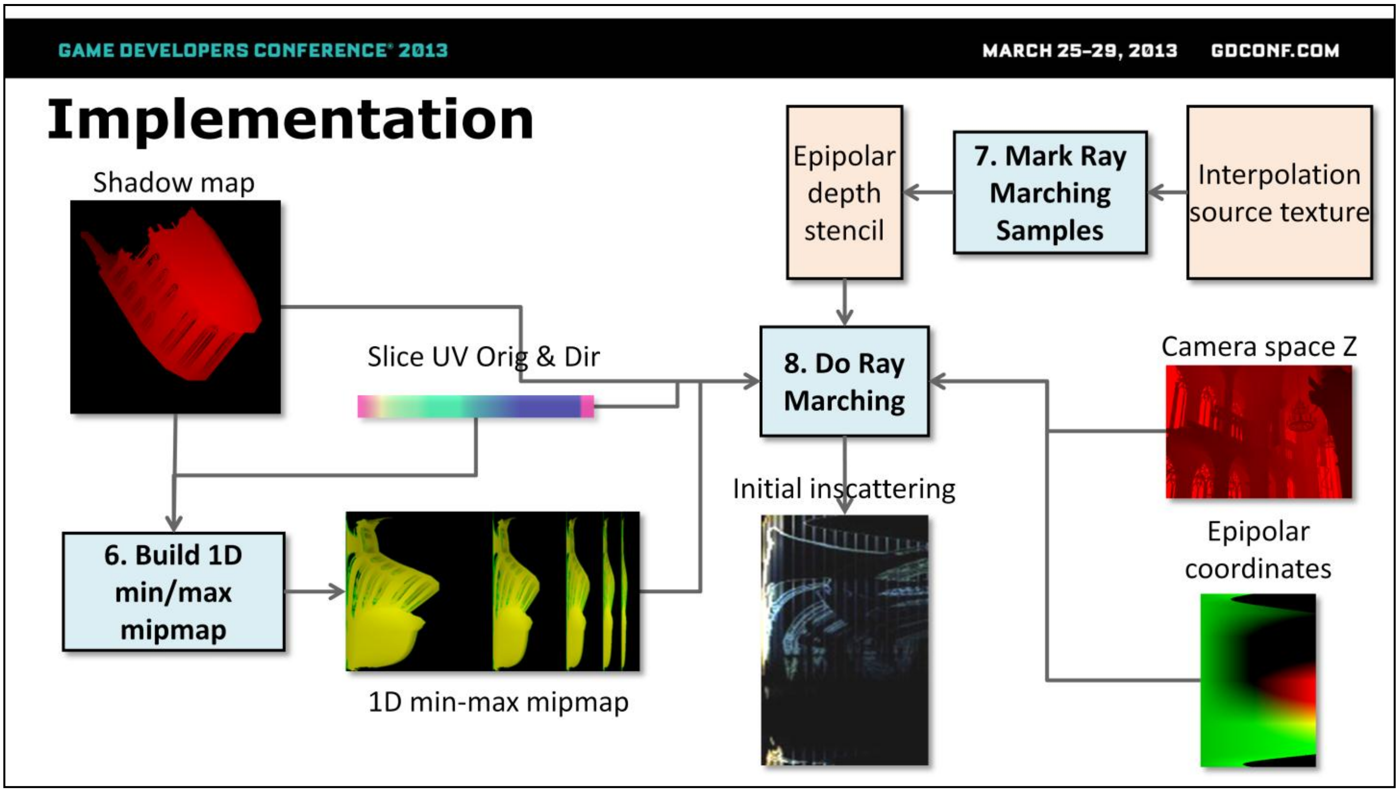
Interpolation source texture

5. Render Slice Origin and Direction in Shadow Map Space

Slice UV Orig & Dir

On the first stage, camera-space z coordinate is reconstructed from the depth buffer. This is required because depth is non-linear, while z coordinate can be safely interpolated

- On the next step, a 1D texture is computed which contains enter and exit points for each epipolar slice
- This texture as well as camera space z is then used to render coordinate texture and camera space z in epipolar coordinates. At this stage, a depth-stencil buffer is also set up which marks valid samples
- Next, depth breaks are detected and interpolation source texture is computed
- Next, another 1D texture is rendered which contains slice origin and direction computed as described above



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

Implementation

Shadow map

Slice UV Orig & Dir

6. Build 1D min/max mipmap

1D min-max mipmap

Epipolar depth stencil

7. Mark Ray Marching Samples

Interpolation source texture

8. Do Ray Marching

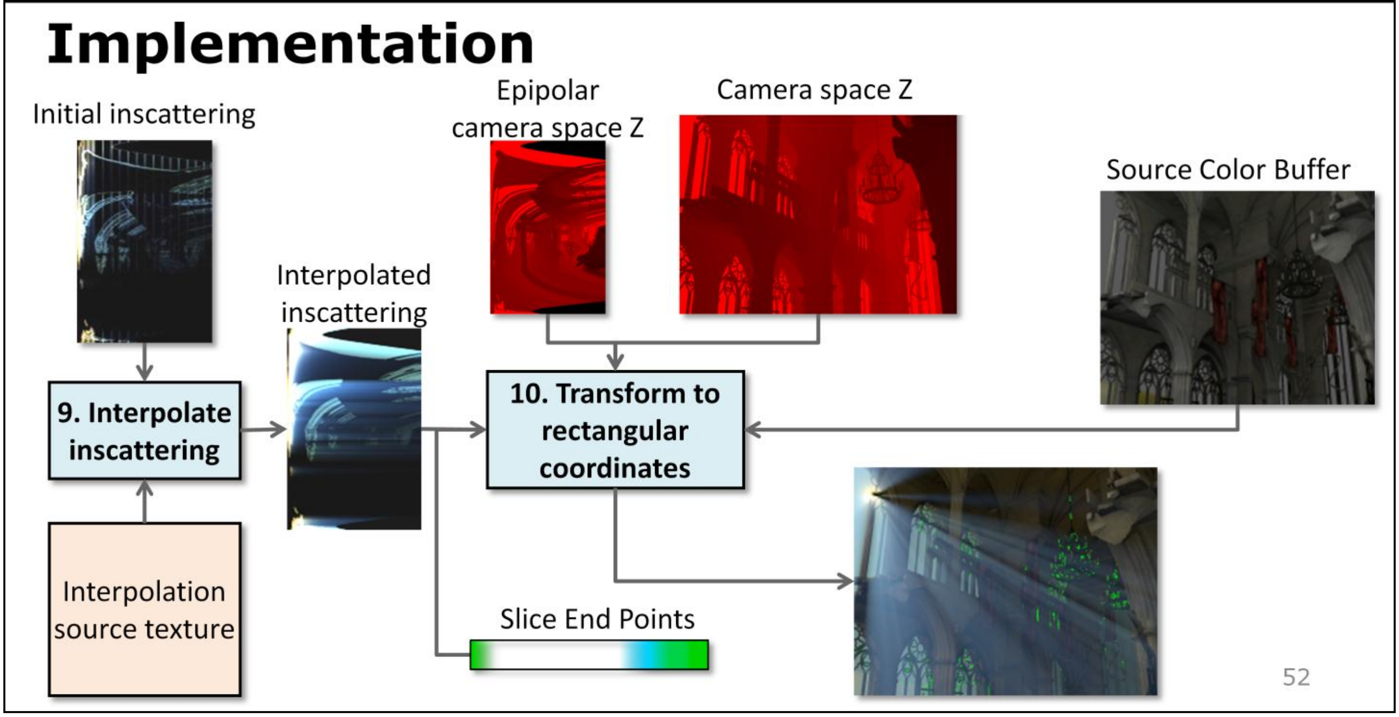
Initial inscattering

Camera space Z

Epipolar coordinates

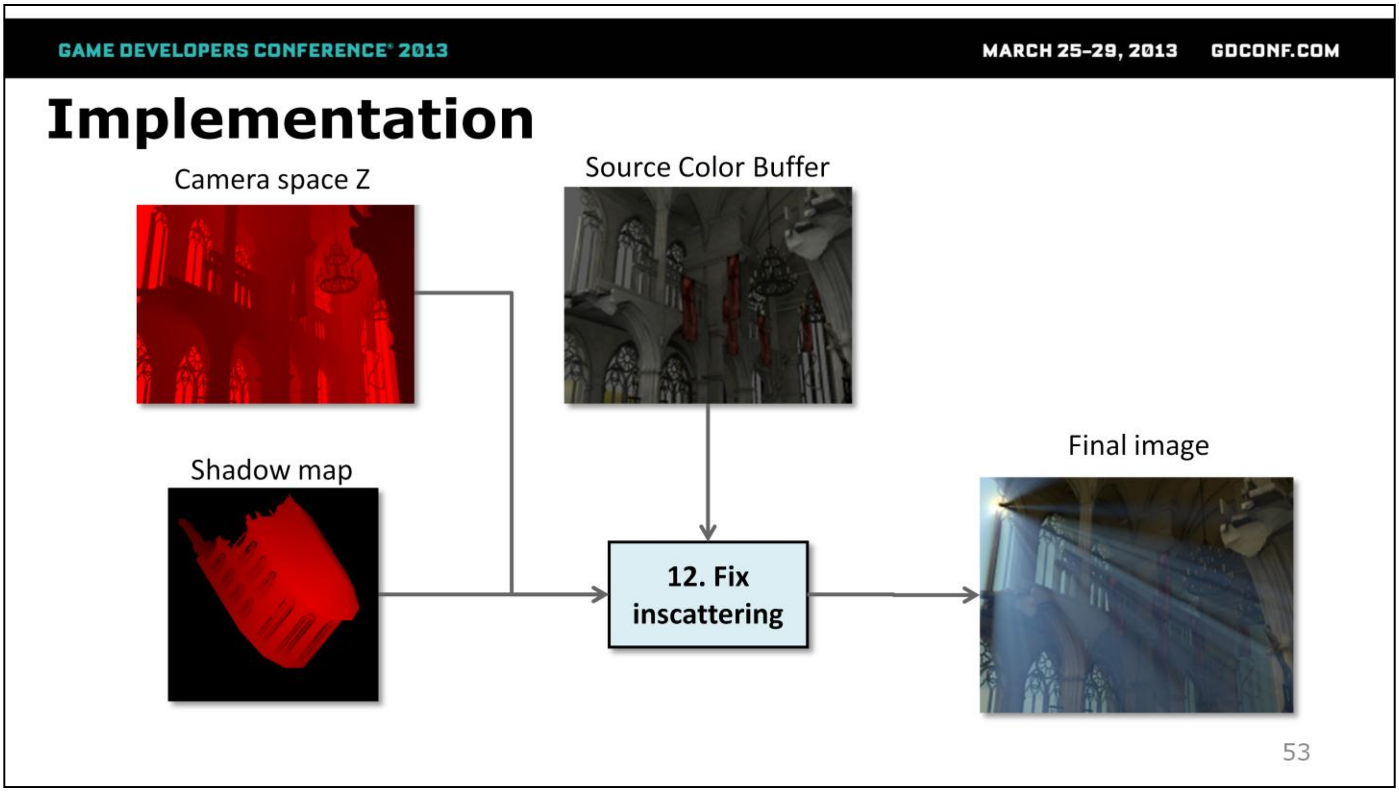
- On the next stage original shadow map and direction and origin textures are used to build 1D min/max binary trees for each epipolar slice
- After that, ray marching samples are marked in the stencil and
- Ray marching algorithm with 1D min/max optimization is executed for each sample

Implementation



Initial inscattering
Epipolar camera space Z
Camera space Z
Source Color Buffer
Interpolated inscattering
9. Interpolate inscattering
10. Transform to rectangular coordinates
Interpolation source texture
Slice End Points

- On the next stage, initial inscattering is interpolated using interpolation source texture
- And inscattering is transformed from epipolar to rectangular coordinates. Both epipolar and rectangular camera space z textures are used at this stage to compute bilateral weights. These pixels, which cannot be interpolated from epipolar coordinates are marked in stencil at this stage



Implementation

Camera space Z

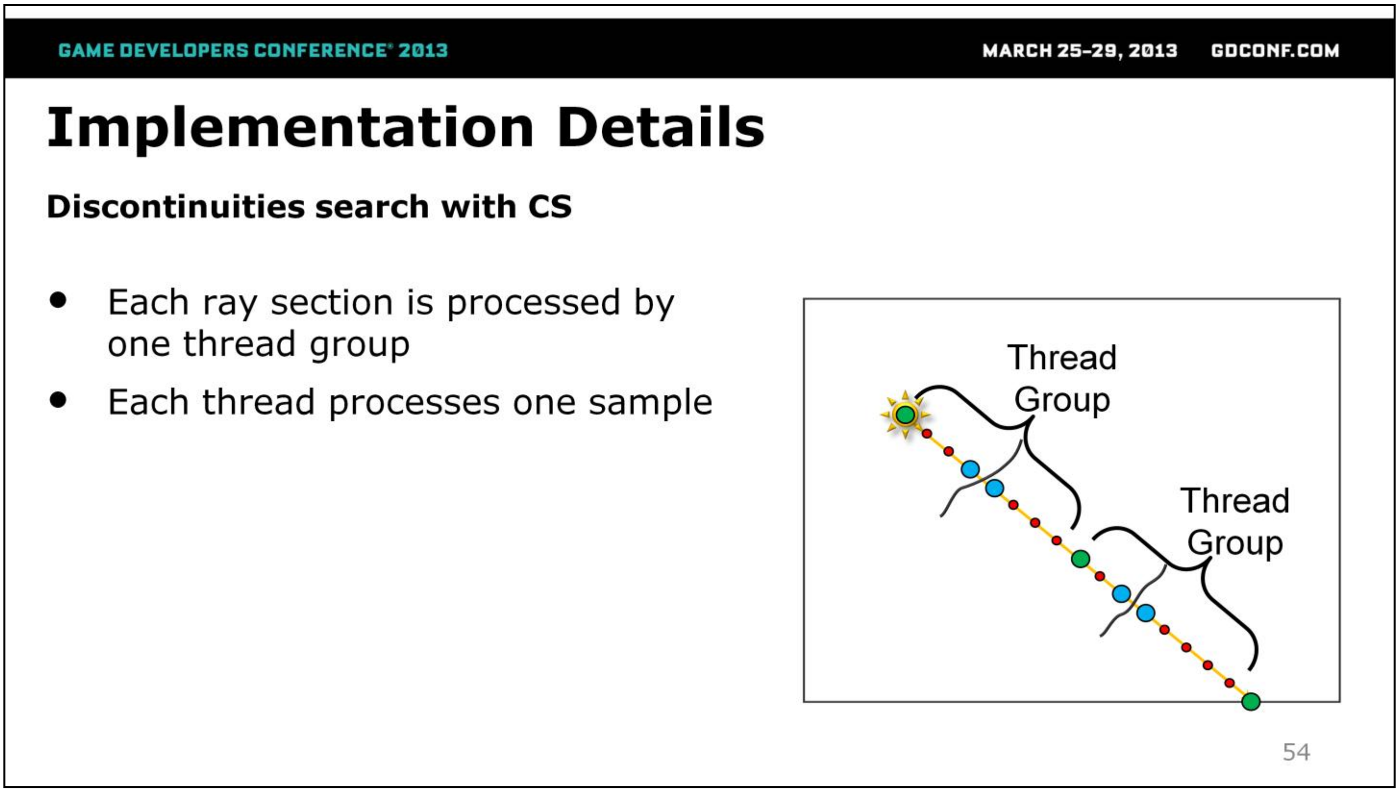
Source Color Buffer

Shadow map

12. Fix inscattering

Final image

* Finally, inscattering fix-up pass is performed for pixels marked in stencil. No 1D min/max optimization is used at this stage.

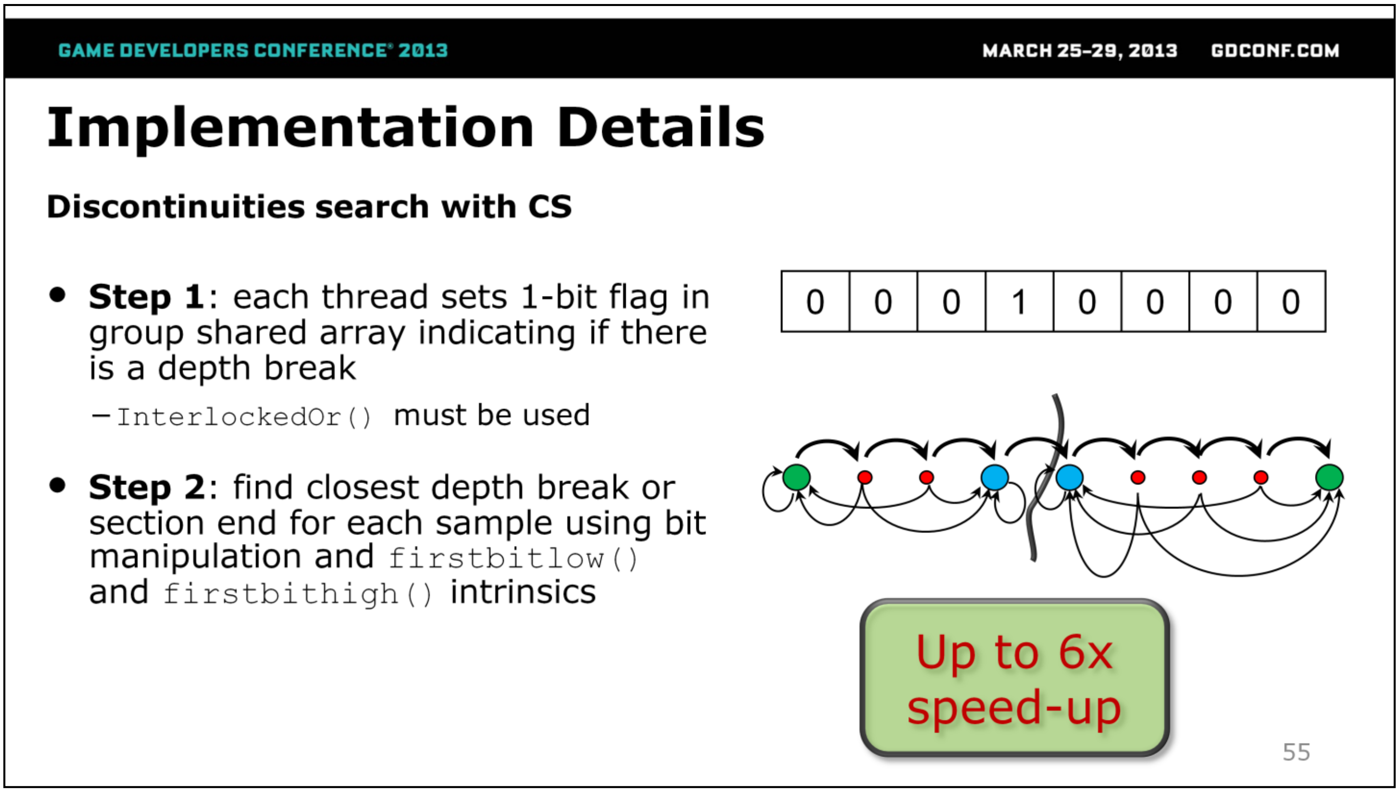


Implementation Details

Discontinuities search with CS

- Each ray section is processed by one thread group
- Each thread processes one sample

The depth discontinuities search stage in the algorithm is implemented with a compute shader. Each thread group of this shader processes one ray section between two neighboring ray marching samples and each thread processes one sample.



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

Implementation Details

Discontinuities search with CS

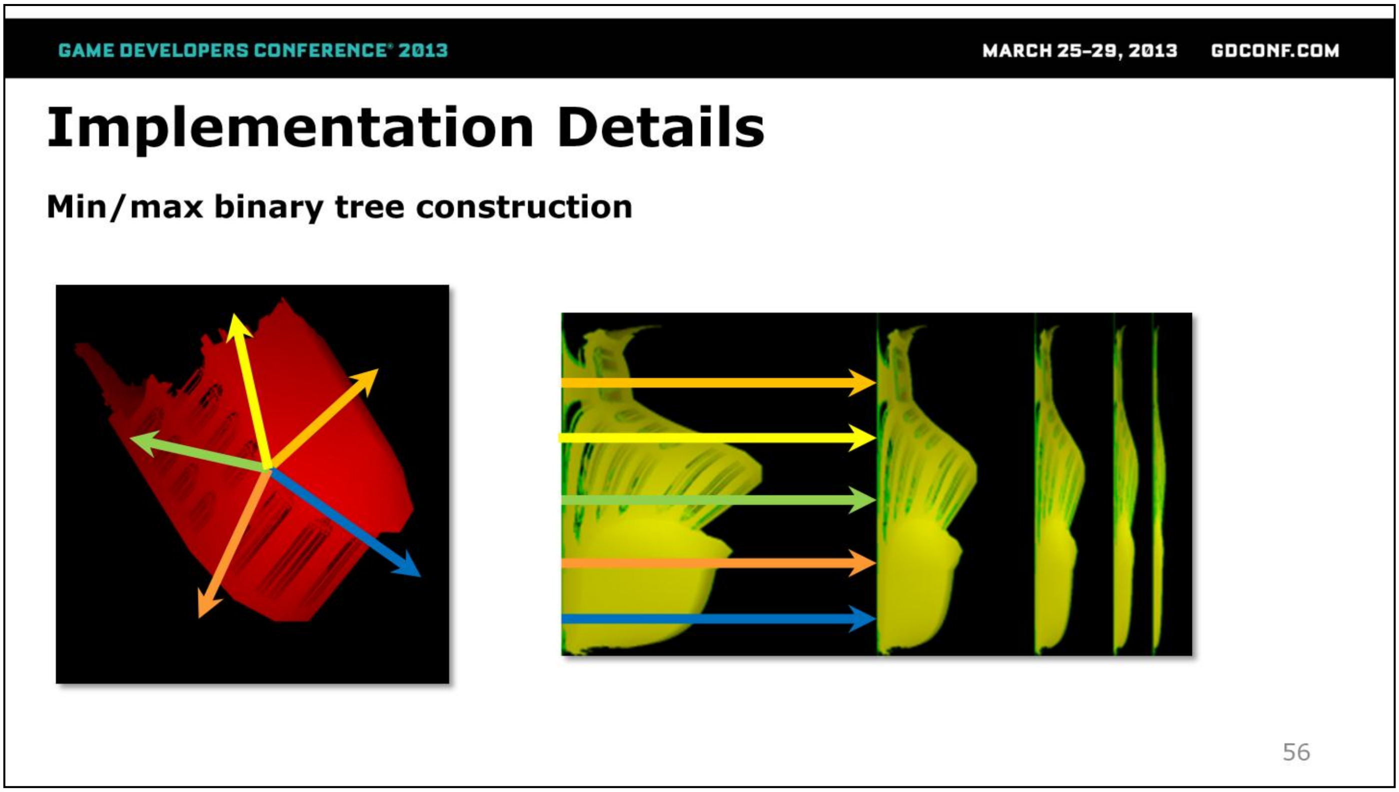
- **Step 1**: each thread sets 1-bit flag in group shared array indicating if there is a depth break
  - `InterlockedOr()` must be used
- **Step 2**: find closest depth break or section end for each sample using bit manipulation and `firstbitlow()` and `firstbithigh()` intrinsics

0	0	0	1	0	0	0	0

Up to 6x speed-up

The process consists of two steps

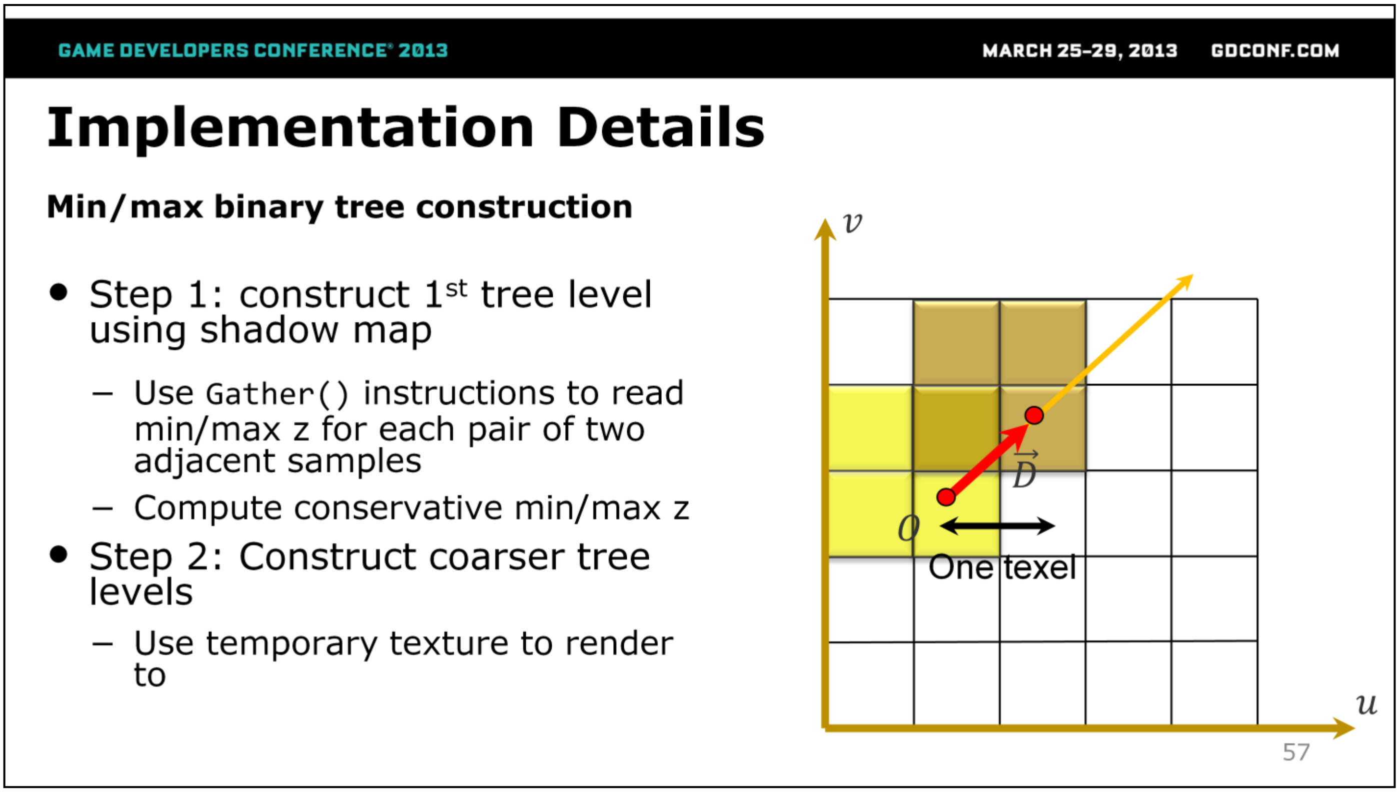
- On the first step, a shared-memory array is populated with 1-bit flags indicating if there is a discontinuity between each two adjacent samples in this segment. The flags are packed as 32 bit uints
- On the second step, interpolation source samples are identified using firstbitlow() and firstbithigh() intrinsic functions that return the bit position of the first non-zero bit starting from the lowest order bit and the highest order bit, respectively
- Our experiments showed that this implementations is up to 6x times faster for long steps than direct implementation of depth breaks detection algorithm presented in original paper [ED10].



Implementation Details

Min/max binary tree construction

This animation shows process of constructing 1D min/max binary trees



Implementation Details

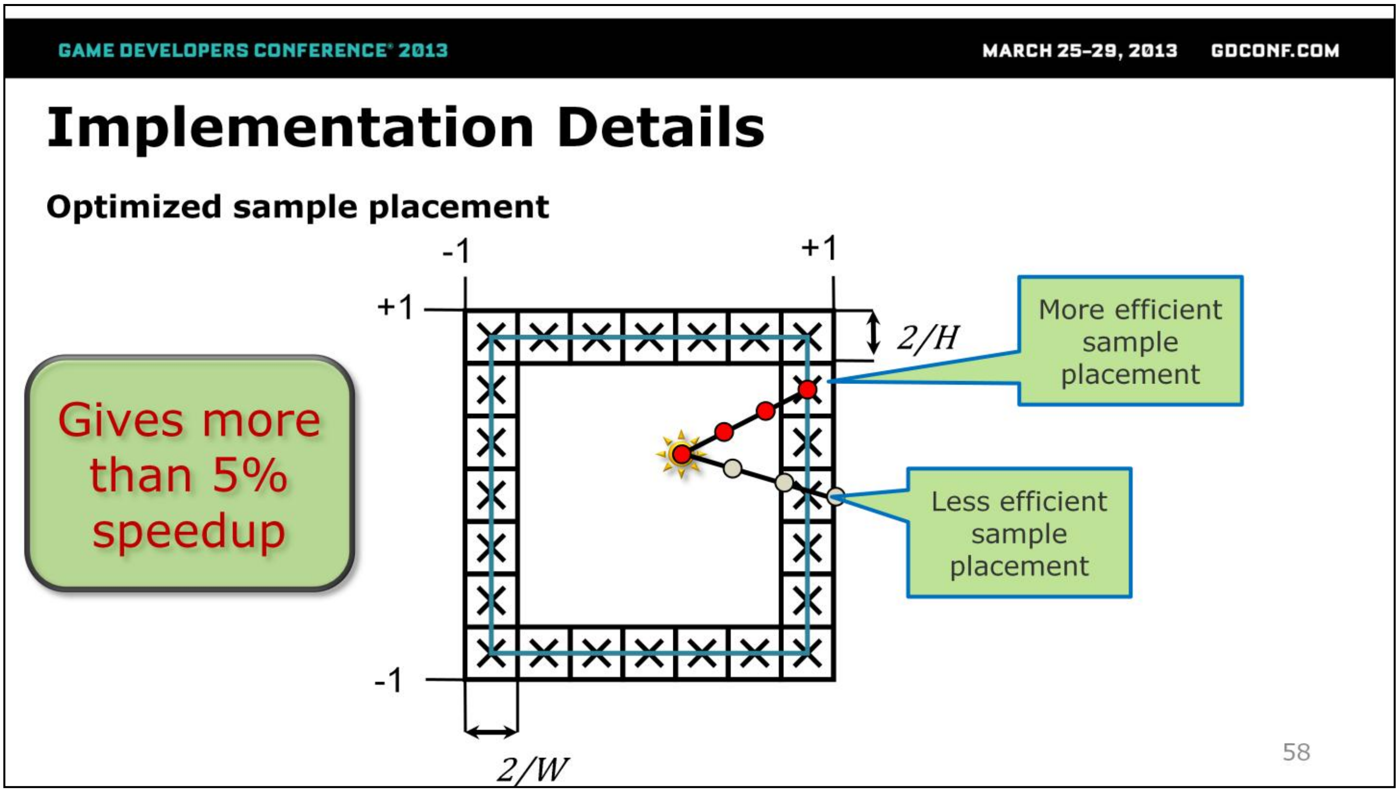
Min/max binary tree construction

- Step 1: construct 1st tree level using shadow map
  - Use `Gather()` instructions to read min/max z for each pair of two adjacent samples
  - Compute conservative min/max z
- Step 2: Construct coarser tree levels
  - Use temporary texture to render to

Creating the 1D min/max binary trees is performed using a well-known flip/flop approach when two textures are alternately used as a source and destination.

- The initial shadow map is used to initialize the first level of the min/max binary trees.
  - Gather() instruction is used here to load four samples which will be required for PCF filtering of each sample and compute conservative minimum and maximum values
  - This guarantees accelerated algorithm produces exactly the same results as if all texels were visited
- After that, coarse tree levels are constructed by loading two min/max values from the corresponding nodes at the next finer level.

We tried using compute shader for this step, but this approach turned out to be less efficient due to poor GPU utilization. We also do not construct full binary trees because it is very unlikely that coarse levels could be reached. Besides, rendering to low-resolution textures is inefficient on modern GPUs.



Implementation Details

Optimized sample placement

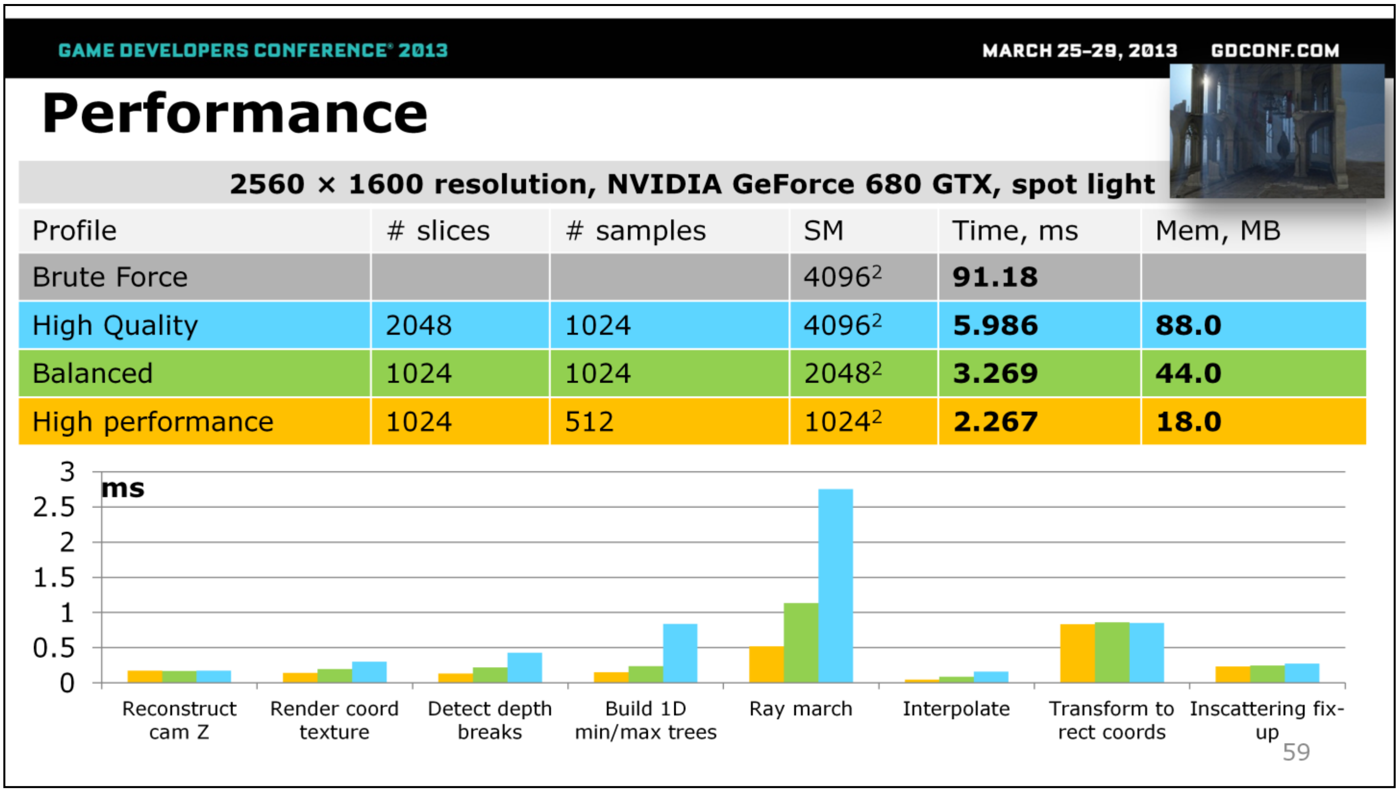
Gives more than 5% speedup

More efficient sample placement

Less efficient sample placement

An important aspect related to sample generation which has to be taken into account is that during the rasterization, pixel attributes are interpolated to pixel centers. As a result, the outermost visible screen pixels do not lie exactly on the screen boundary $[-1,1] \times [-1,1]$, but are biased by 0.5 pixel size inwards. Thus exit point of epipolar lines should be located on shrinked screen boundary $\left[-1+\frac{1}{W}, 1-\frac{1}{W}\right] \times \left[-1+\frac{1}{H}, 1-\frac{1}{H}\right]$ where W and H are width and height of the viewport.

If we do not take this into account, samples will be distributed along the epipolar lines less efficiently. As a result, interpolation from epipolar geometry to rectangular coordinates will be less accurate and more screen pixels will require additional correction pass.



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25–29, 2013 GDCONF.COM

Performance

2560 × 1600 resolution, NVIDIA GeForce 680 GTX, spot light

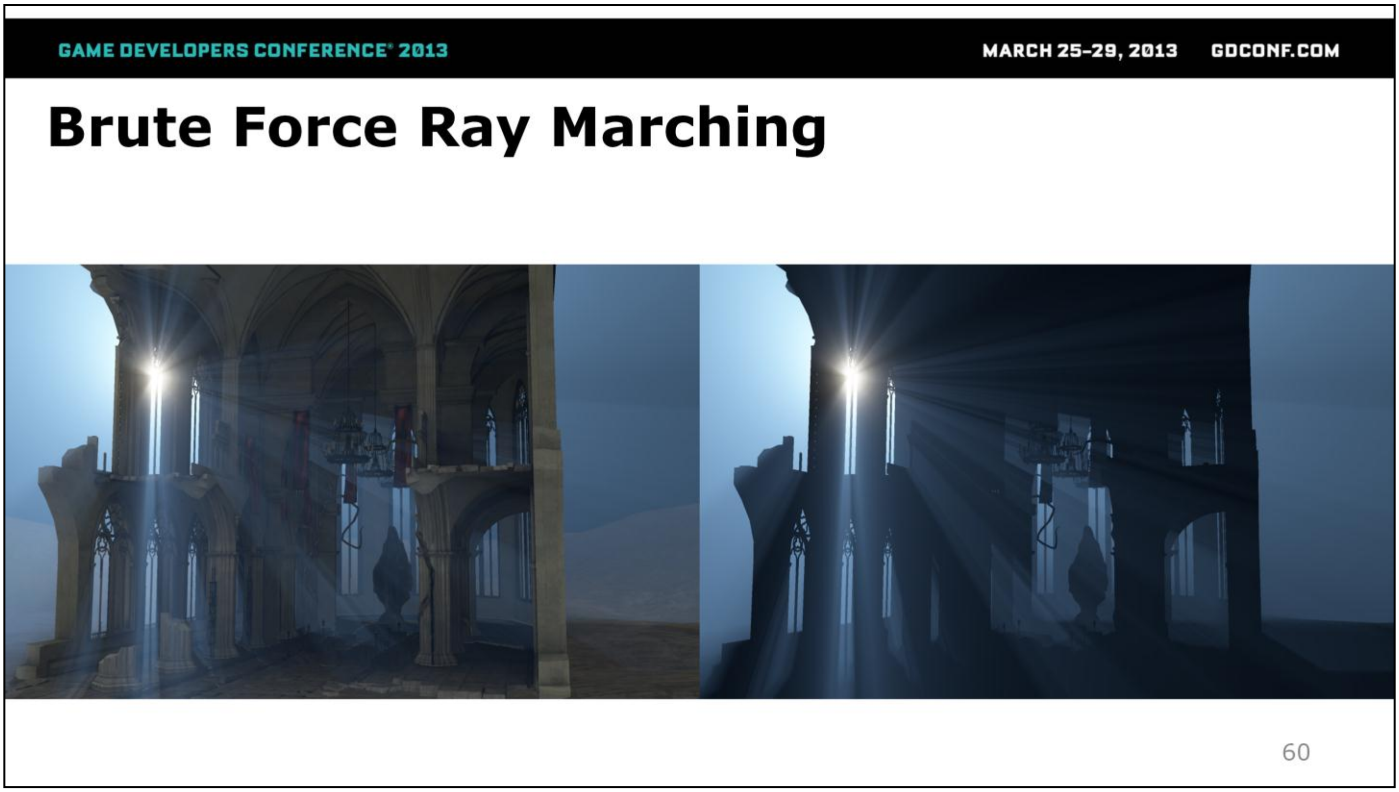
Profile	# slices	# samples	SM	Time, ms	Mem, MB
Brute Force			4096^2	**91.18**	
High Quality	2048	1024	4096^2	**5.986**	**88.0**
Balanced	1024	1024	2048^2	**3.269**	**44.0**
High performance	1024	512	1024^2	**2.267**	**18.0**

59

The technique has a number of different parameters which allow trade quality for performance. This makes it suitable for a wide range of hardware, from high-end GPUs to processor graphics.

This chart shows the performance of different stages for high-end discrete GPU for very high screen resolution and three different quality profiles. It also presents the time of the brute force ray marching algorithm executed for each screen pixel with no 1D min/max optimization.

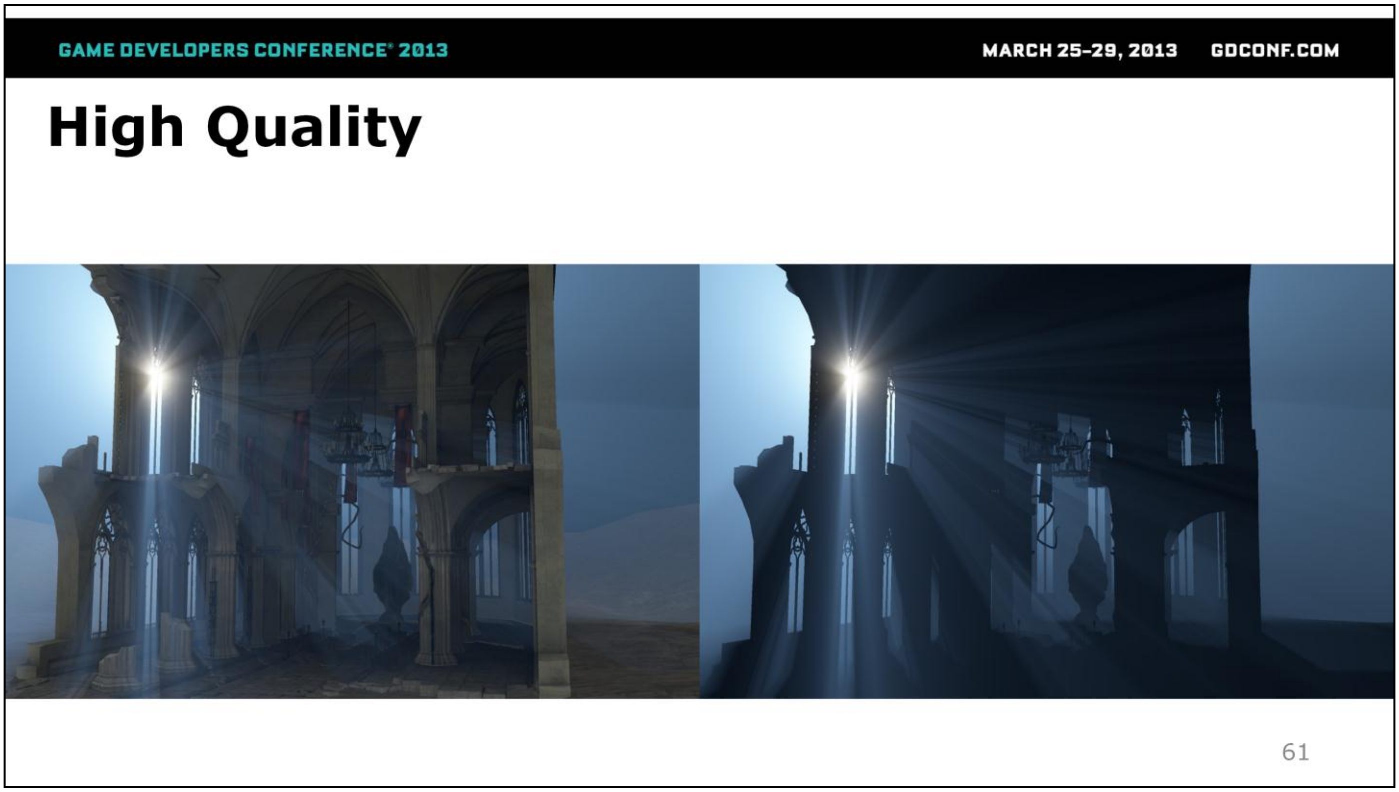
Memory consumption for these quality settings is also shown in the table. From the first glance the required memory amount could seem to be very high. But if we take into account the fact that for such resolution, the back buffer and depth buffer occupy 31 MB and shadow map is 64 MB, than 88 MB is less than total amount of memory required to store these buffers, which looks quite reasonable.



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25–29, 2013 GDCONF.COM

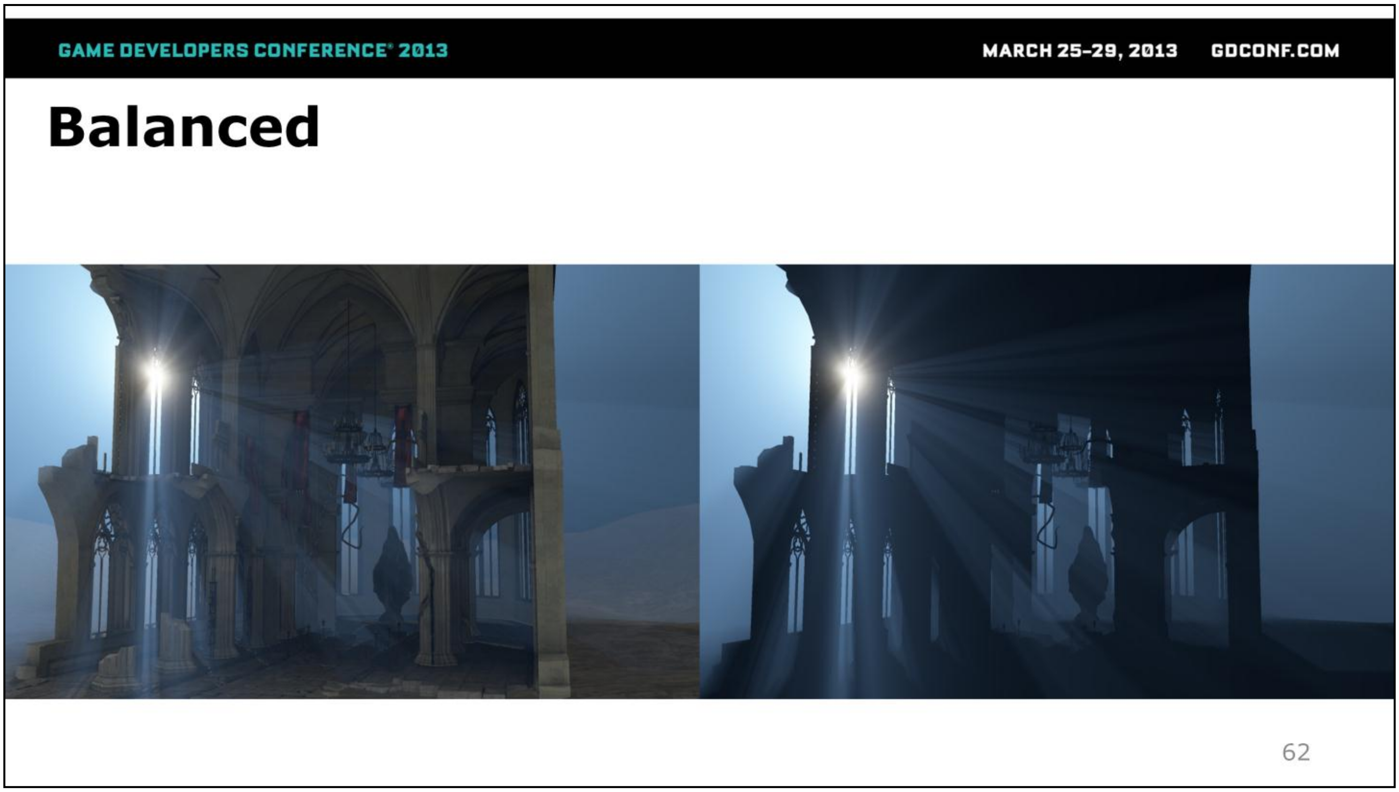
Brute Force Ray Marching

This is the reference picture for brute force ray marching



High Quality

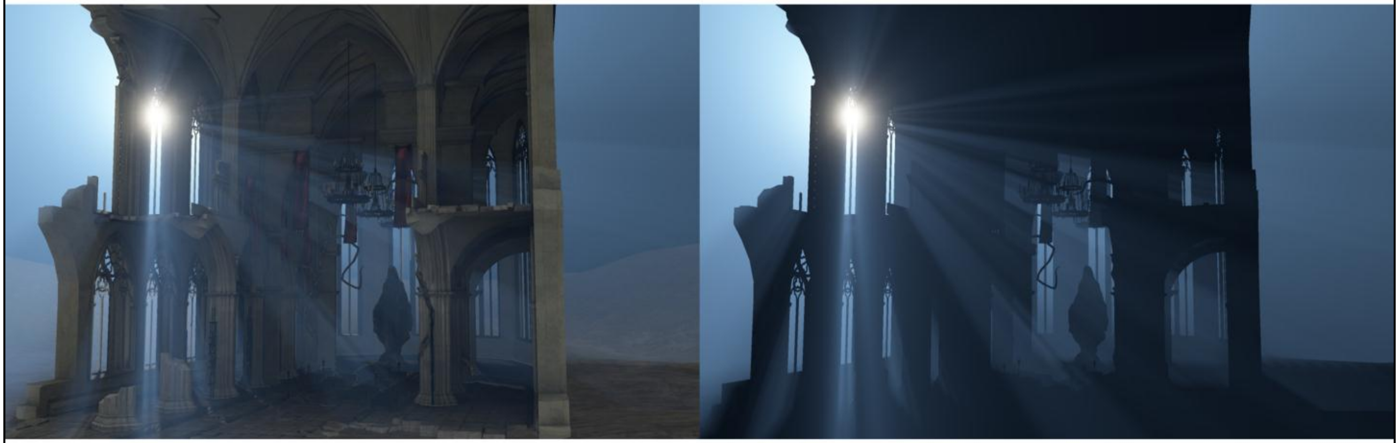
This is the high quality profile which is almost identical to the reference picture



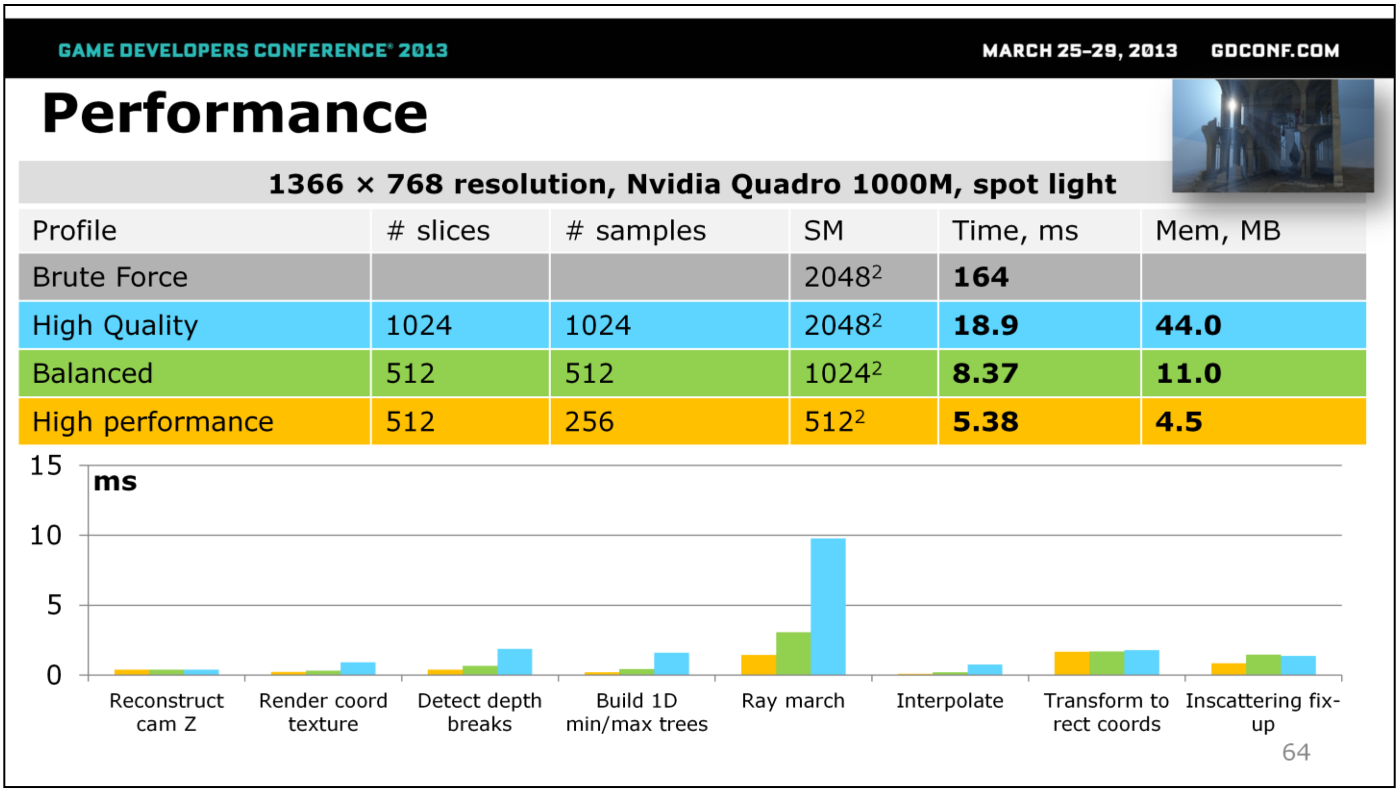
Balanced

In balanced quality profile, rays loose crisper appearance because lower resolution shadow map is used

High Performance



In high performance profile, the rays are even more smoother, however, no apparent artifacts like banding are noticeable

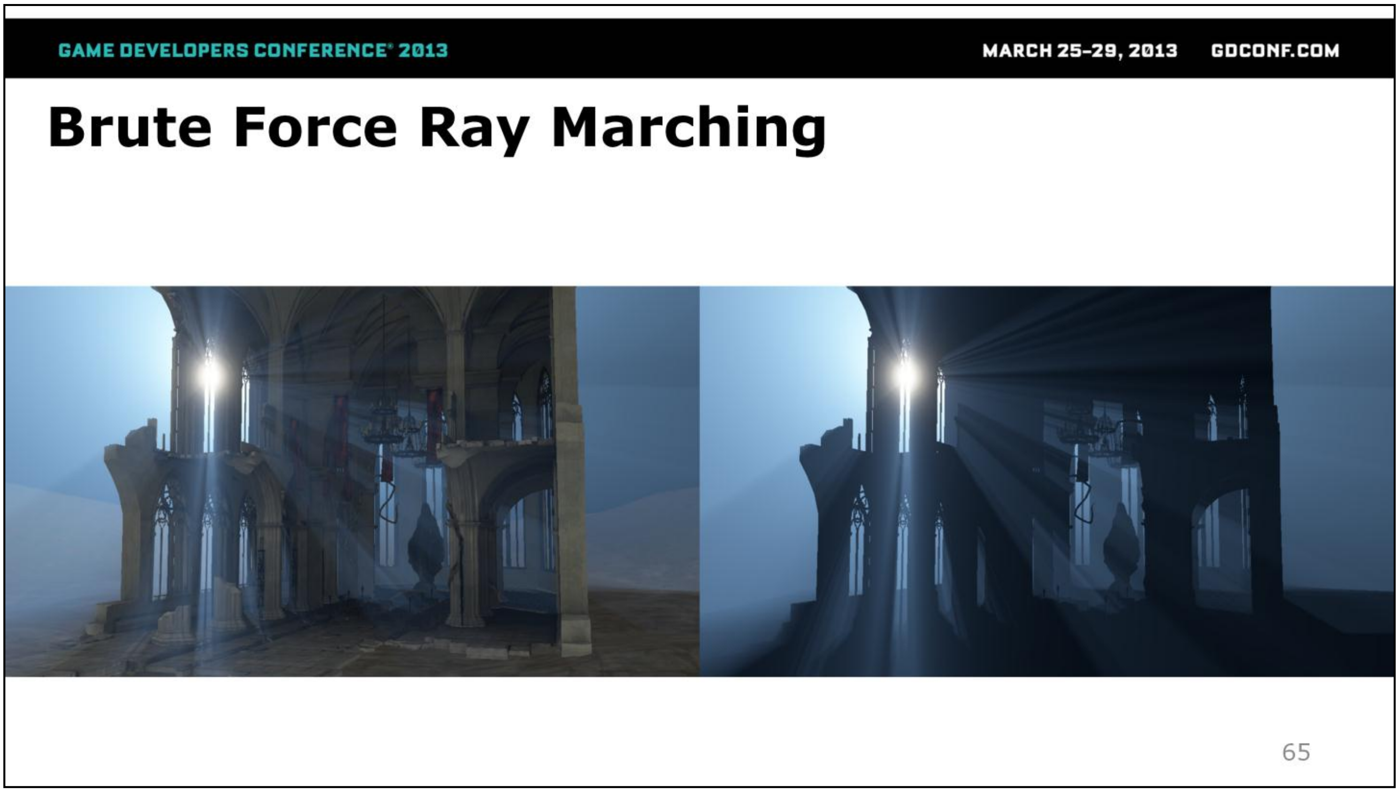


Performance

1366 × 768 resolution, Nvidia Quadro 1000M, spot light

Profile	# slices	# samples	SM	Time, ms	Mem, MB
Brute Force			2048^2	**164**	
High Quality	1024	1024	2048^2	**18.9**	**44.0**
Balanced	512	512	1024^2	**8.37**	**11.0**
High performance	512	256	512^2	**5.38**	**4.5**

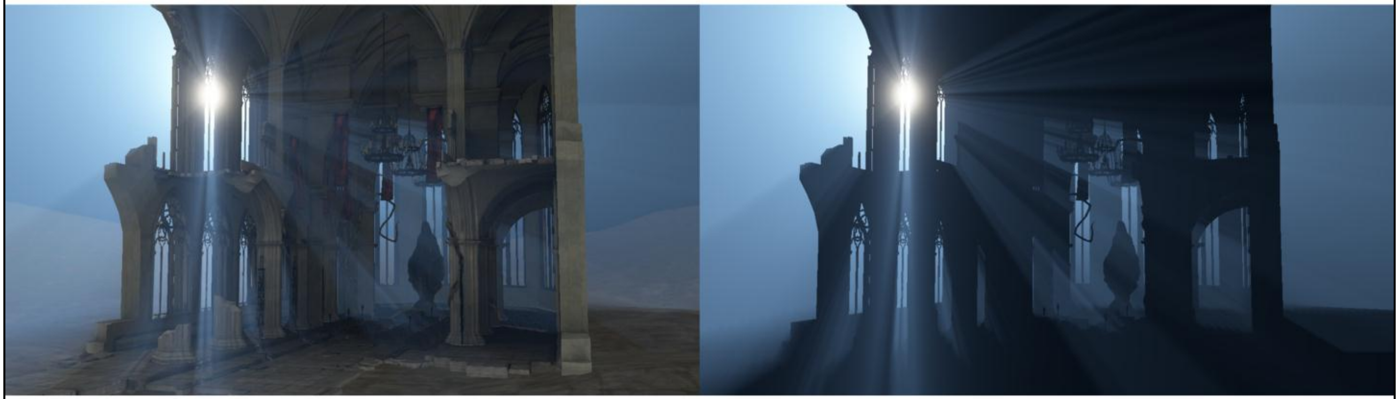
This slide shows performance of the technique for mobile Nvidia GPU. 5.38 ms with 5.5 MB of memory consumption for mobile graphics is not a bad result



Brute Force Ray Marching

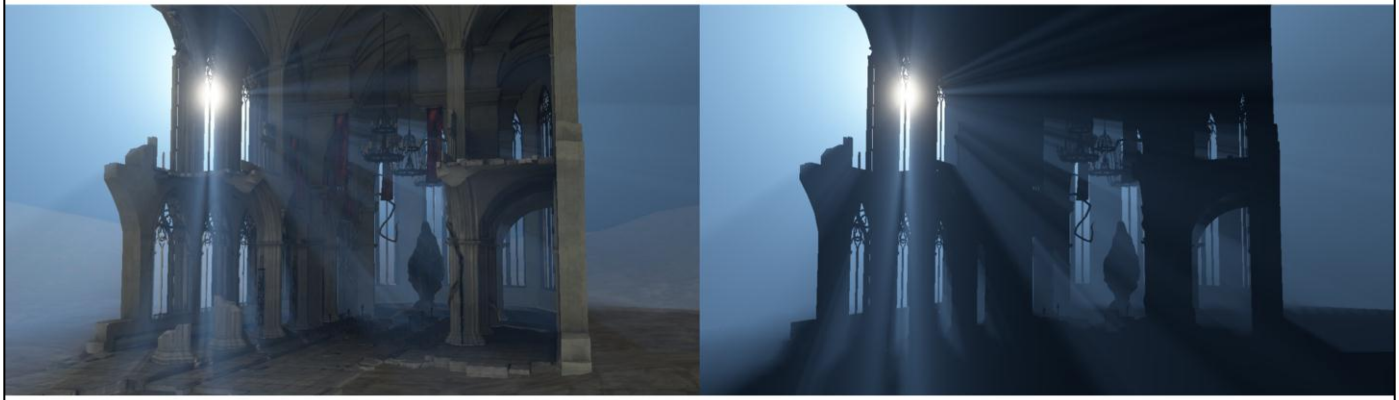
Again this is the reference picture for brute force ray marching

High Quality

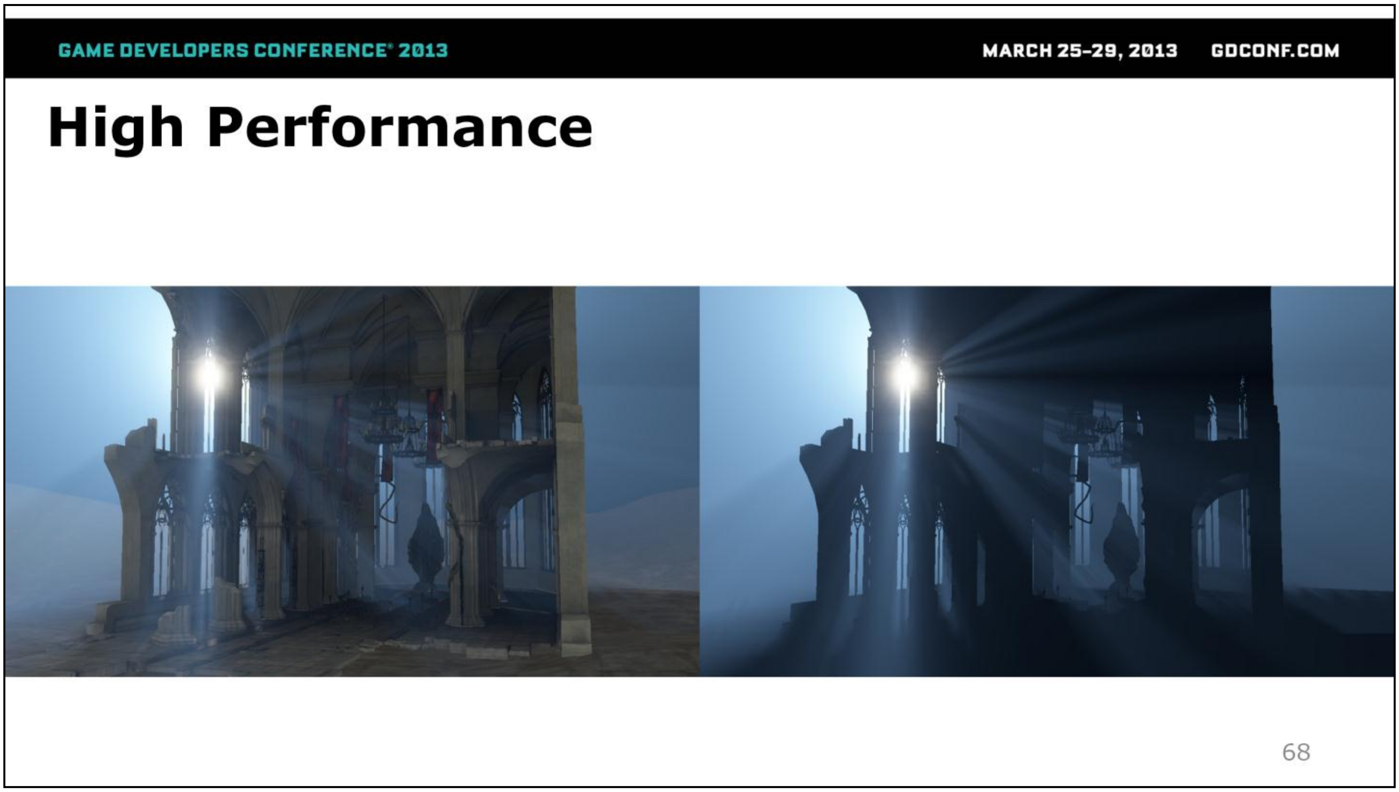


And the high quality profile is almost identical to the reference picture

Balanced

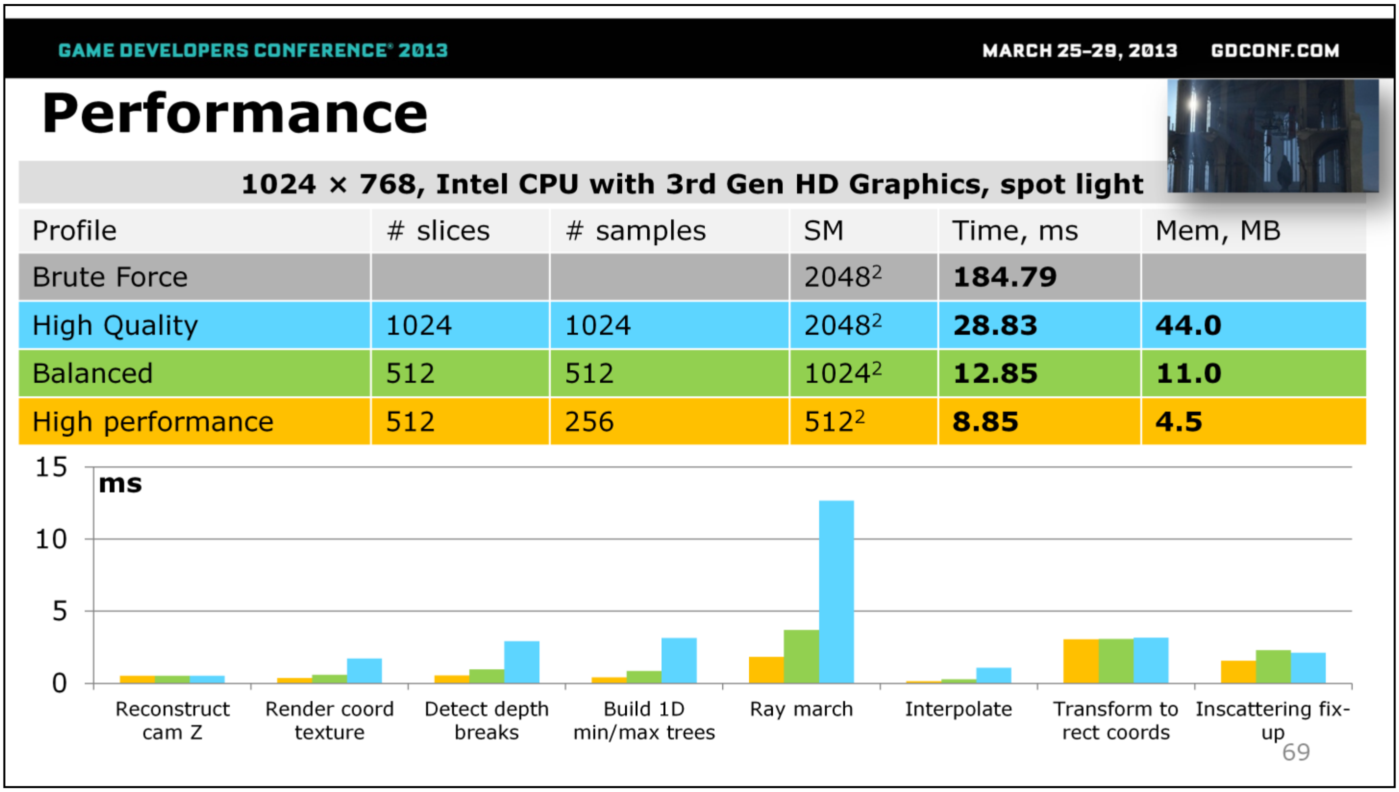


This is the balanced quality where rays become a bit smoother



High Performance

And here is the picture for high performance profile. You can see some artifacts near the light source due to insufficient sampling. These are not very apparent though and with isotropic scattering function are almost unnoticeable. Note that the rest of the rays are smooth and there are no banding or other artifacts



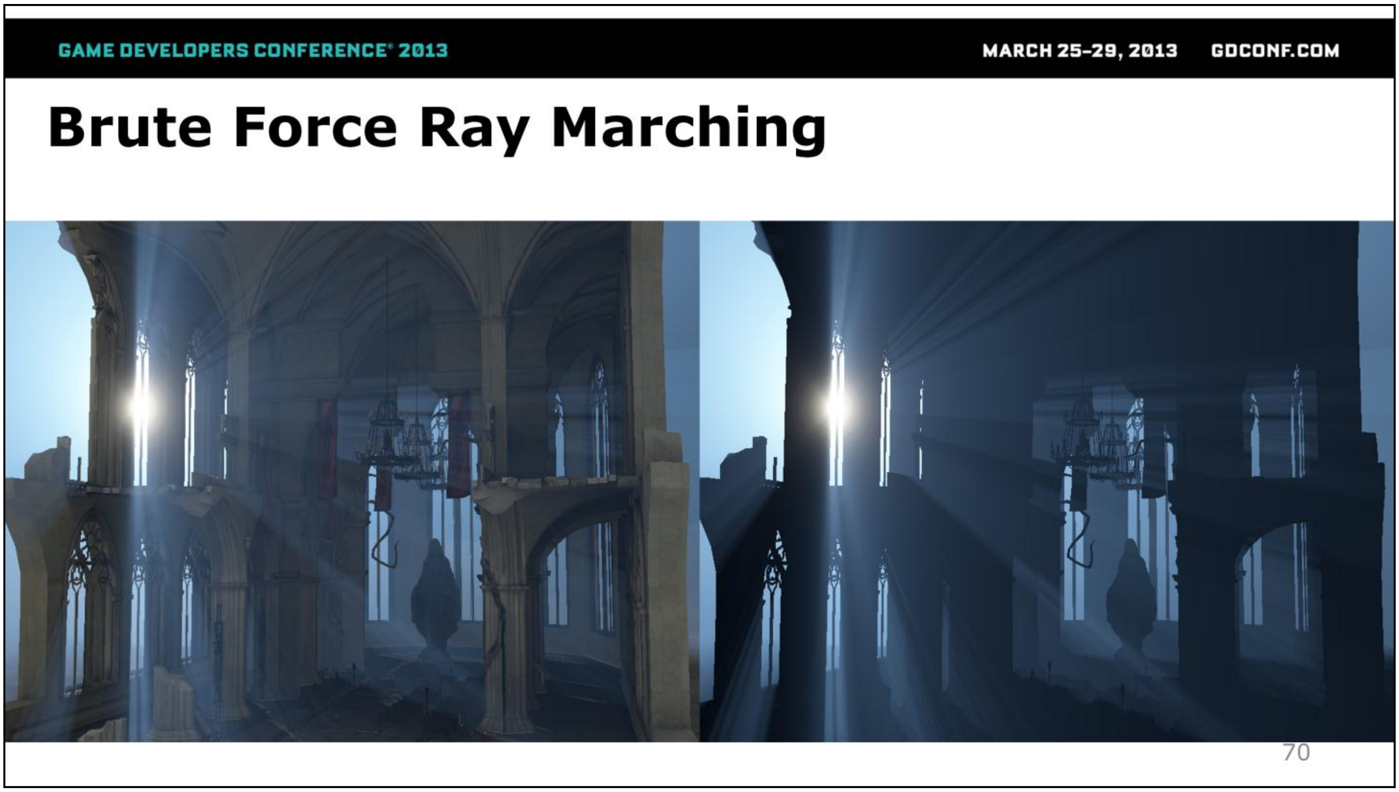
GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

Performance

1024 × 768, Intel CPU with 3rd Gen HD Graphics, spot light

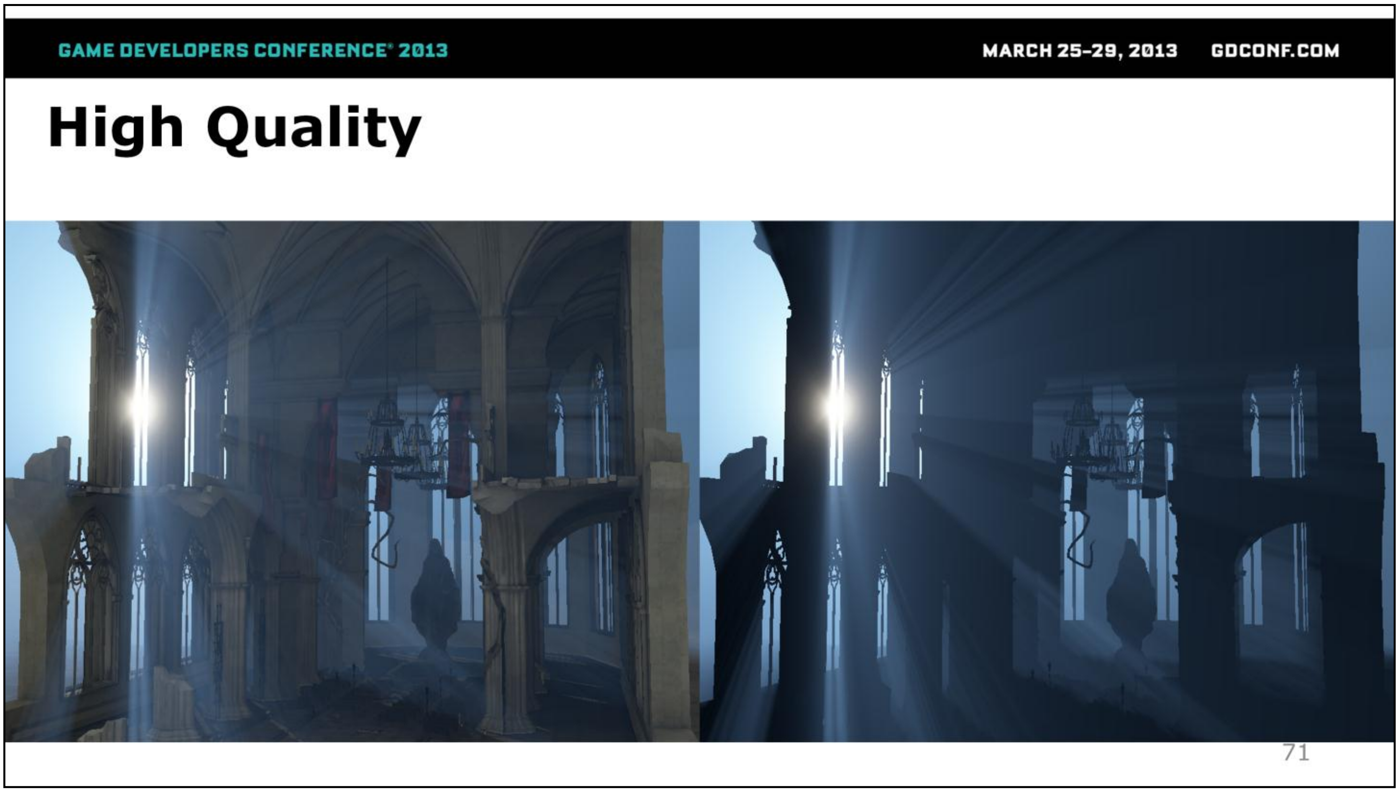
Profile	# slices	# samples	SM	Time, ms	Mem, MB
Brute Force			2048^2	**184.79**	
High Quality	1024	1024	2048^2	**28.83**	**44.0**
Balanced	512	512	1024^2	**12.85**	**11.0**
High performance	512	256	512^2	**8.85**	**4.5**

This chart presents performance for Intel HD graphics. 8.85 ms for processor graphics for rendering such quality effect is not that bad. Even in high quality, the technique still able to render the scene at interactive frame rates on just integrated graphics.



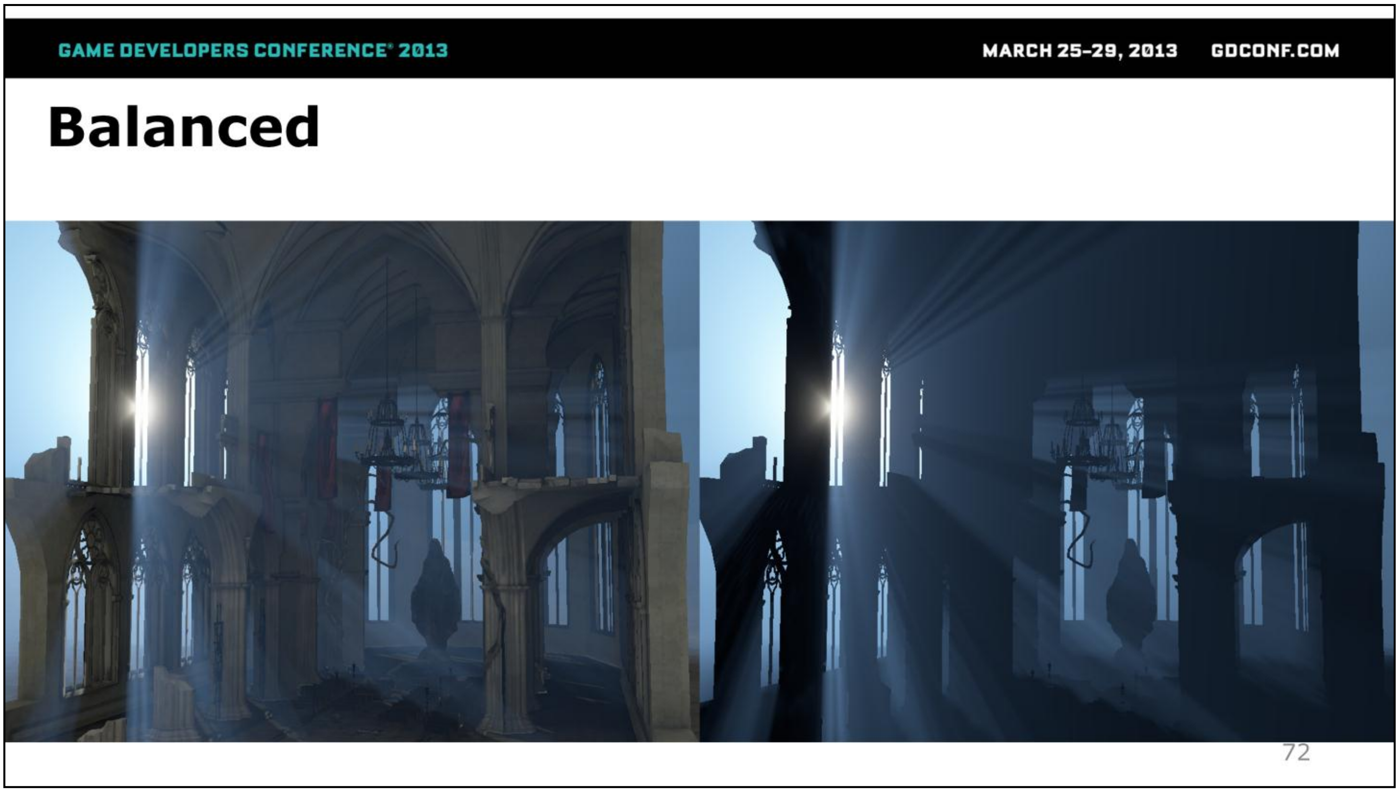
Brute Force Ray Marching

Here is the comparison for different quality settings



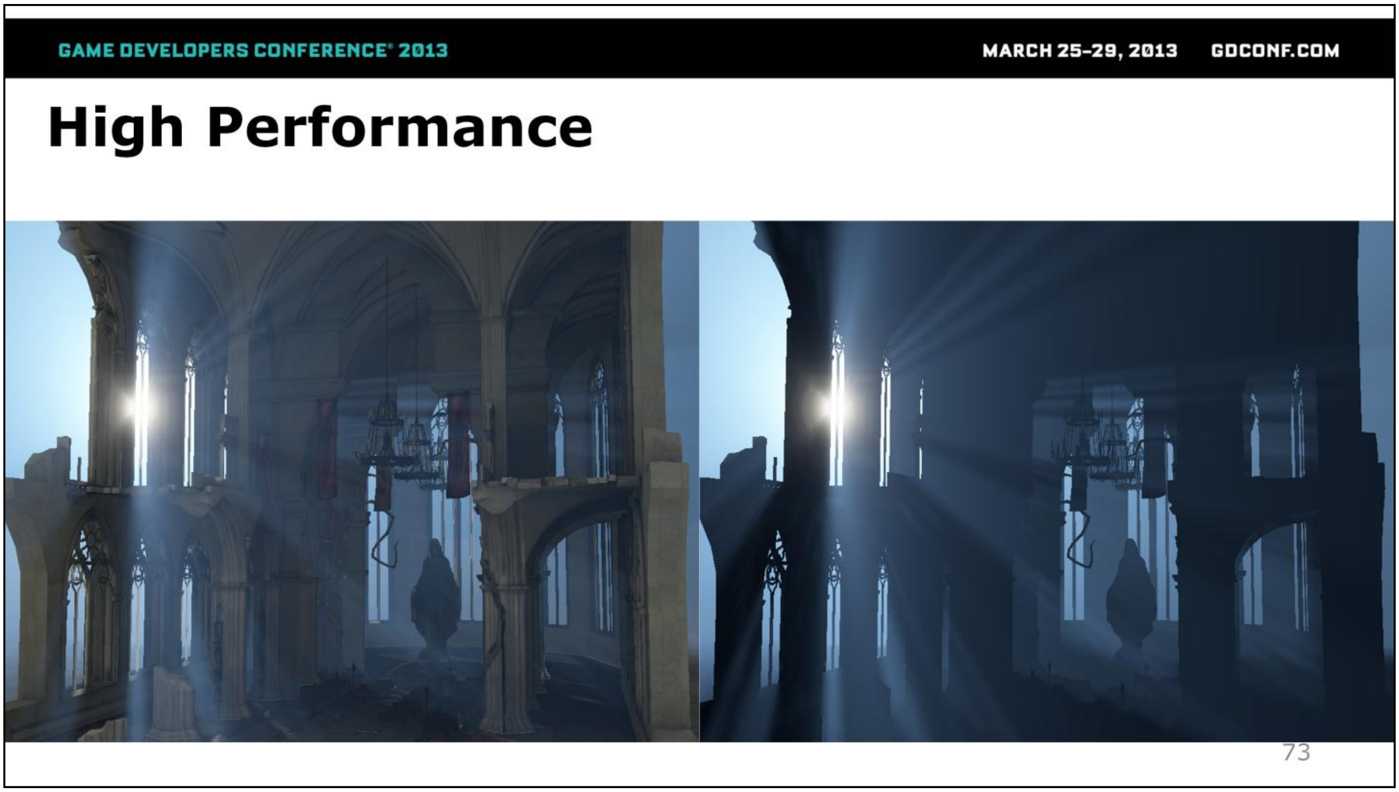
High Quality

The high quality profile is almost identical to the reference picture



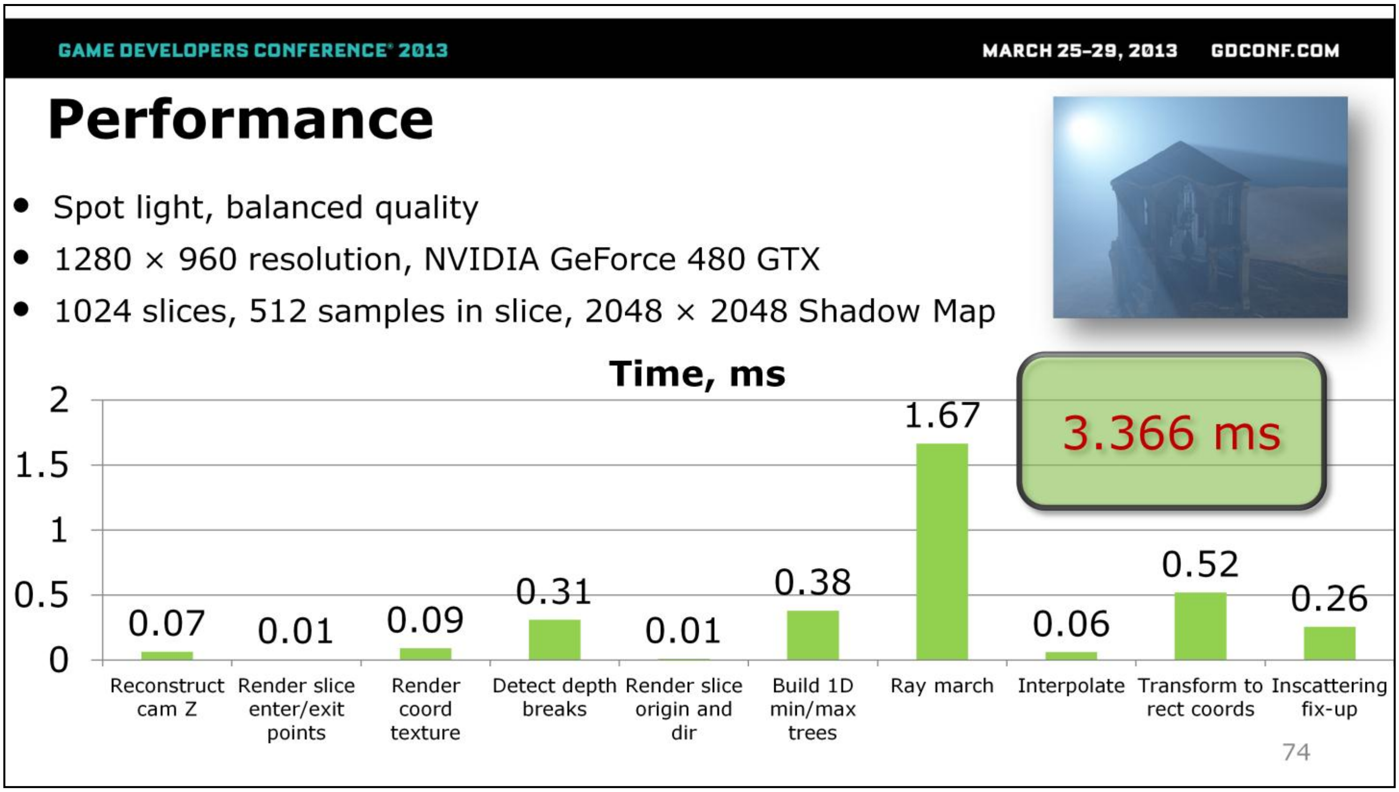
Balanced

This is the balanced quality where rays become a bit smoother



High Performance

Again, even in high performance profile, the effect quality is rather good

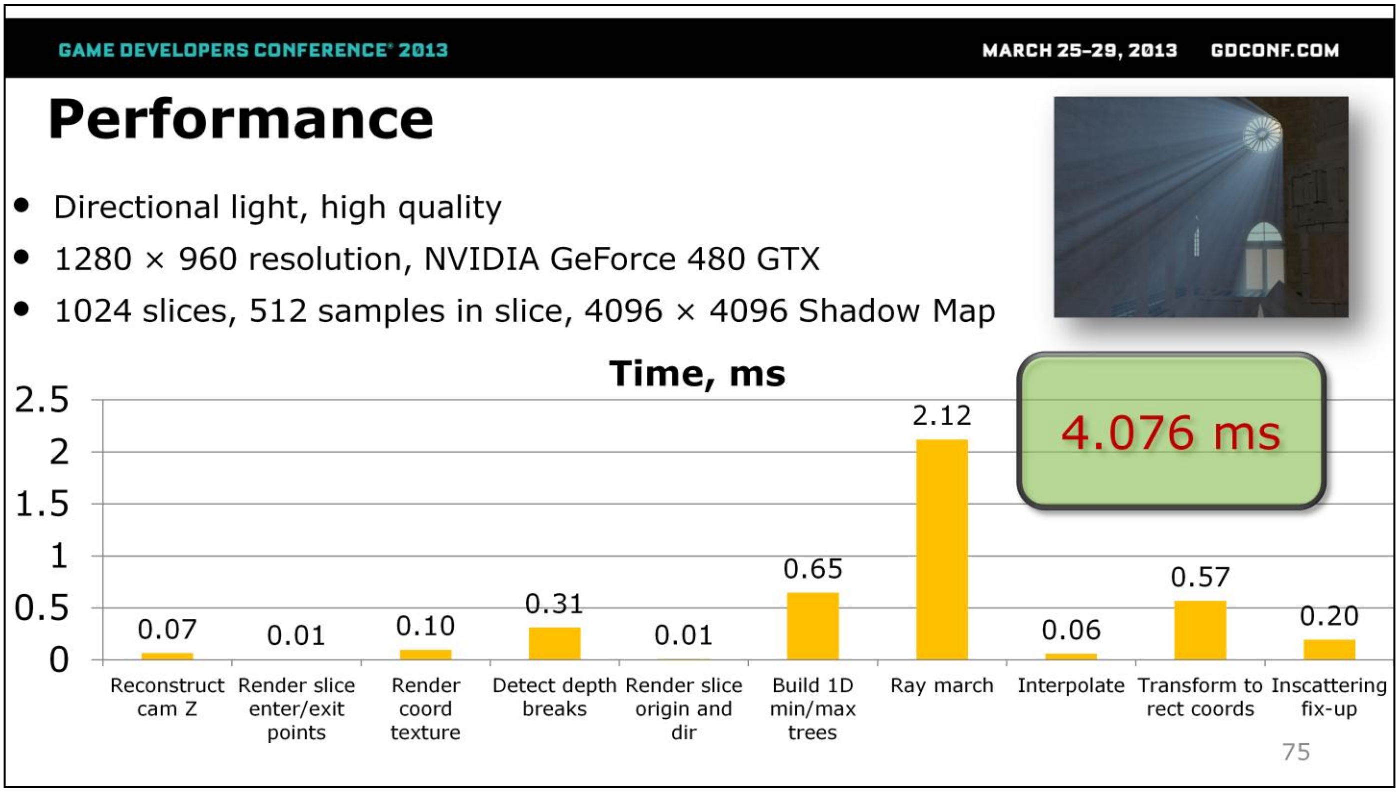


GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

Performance

- Spot light, balanced quality
- 1280 × 960 resolution, NVIDIA GeForce 480 GTX
- 1024 slices, 512 samples in slice, 2048 × 2048 Shadow Map

This chart presents detailed timings for another scene rendered on Nvidia discrete GPU. Total processing time is less than 3.4 ms.



Performance

- Directional light, high quality
- 1280 × 960 resolution, NVIDIA GeForce 480 GTX
- 1024 slices, 512 samples in slice, 4096 × 4096 Shadow Map

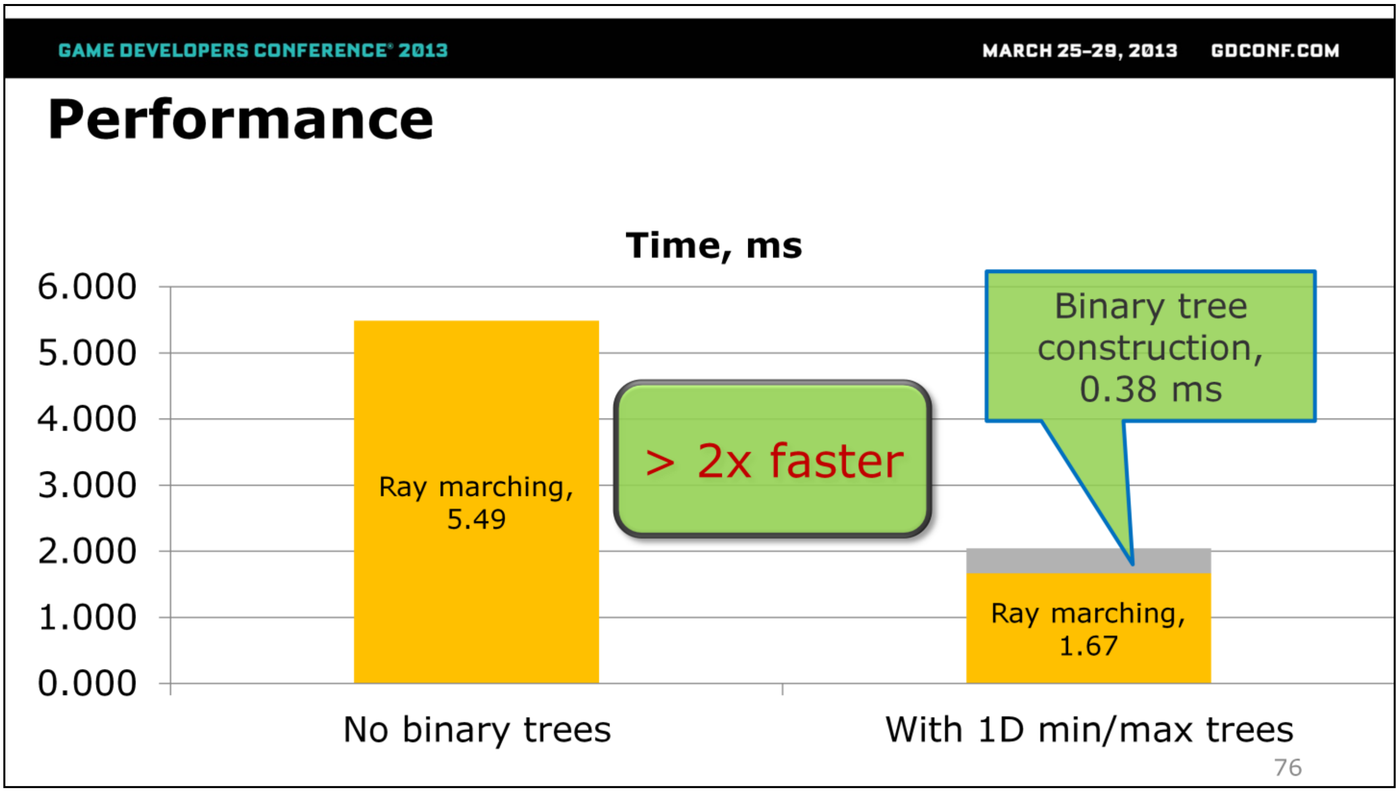
Time, ms

4.076 ms

Here is the performance for another test for directional light source. Fot 4k x 4k shadow map, the technique requires just above 4 ms which is very good result

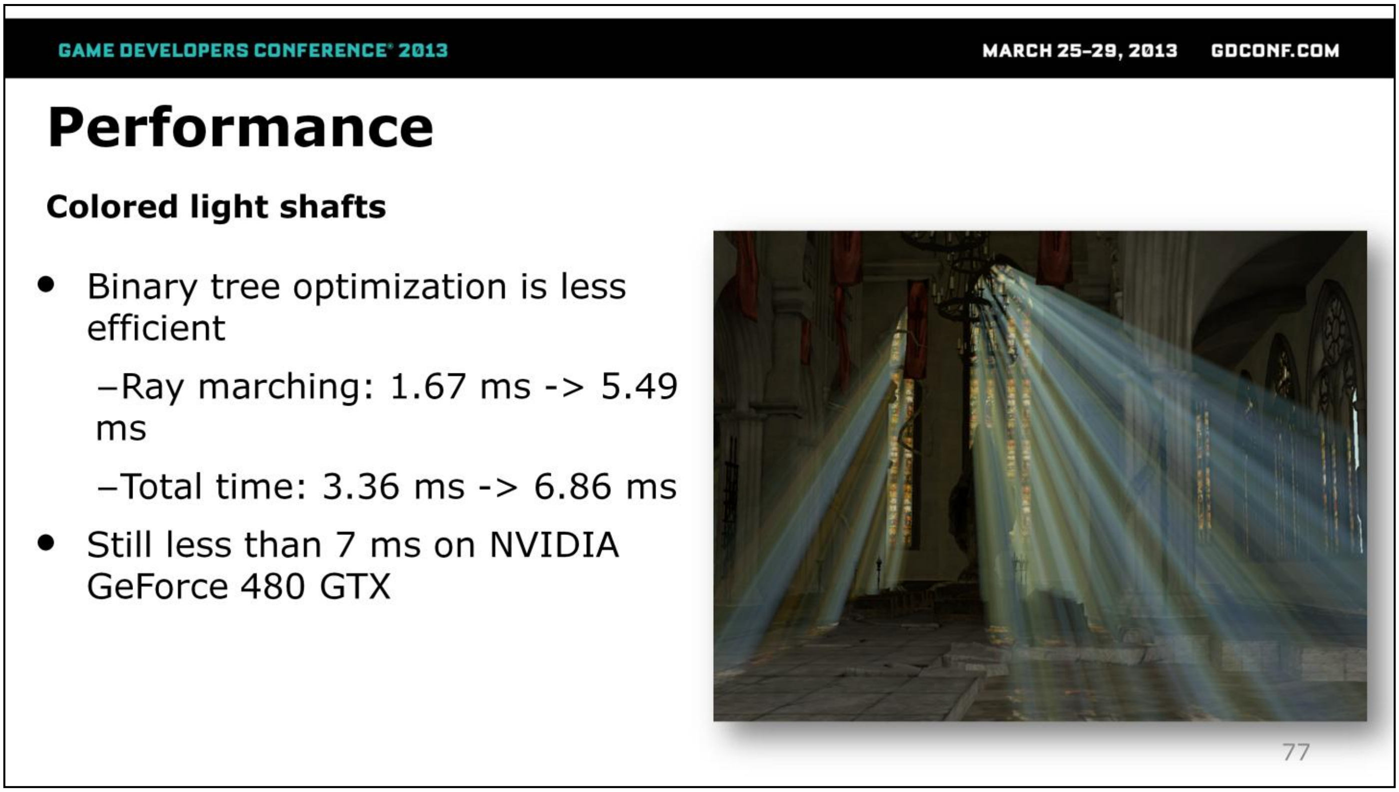
GAME DEVELOPERS CONFERENCE® 2013 MARCH 25–29, 2013 GDCONF.COM

Performance



For a spot light source, without 1D min/max binary tree acceleration, the ray marching time alone is 3.3 times higher while the total frame time is 2x times higher.

For higher quality settings the speedup is much higher



Performance

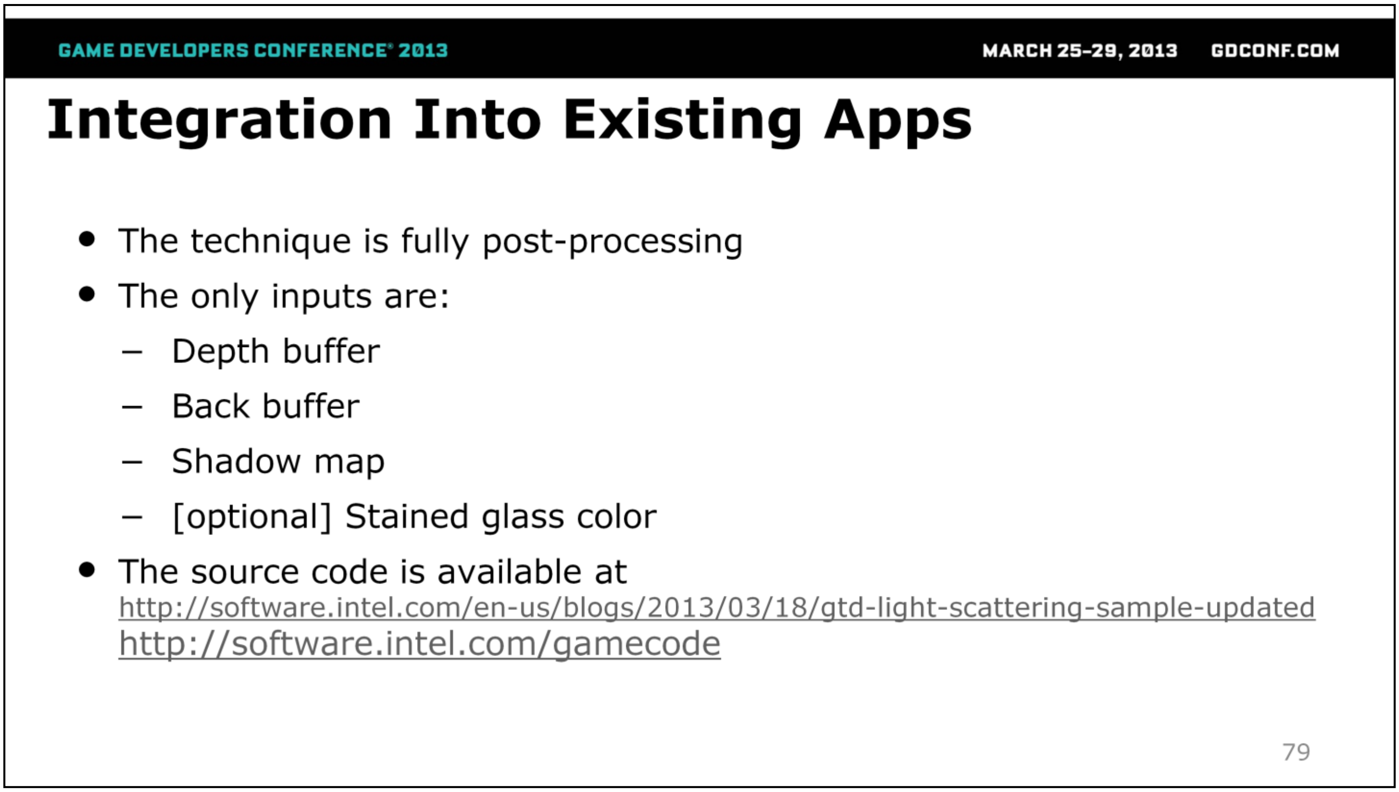
Colored light shafts

- Binary tree optimization is less efficient
  - –Ray marching: 1.67 ms -> 5.49 ms
  - –Total time: 3.36 ms -> 6.86 ms
- Still less than 7 ms on NVIDIA GeForce 480 GTX

The algorithm also supports colored light shafts. However, min/max optimization in this case is much less efficient, because lit regions cannot be skipped and should be traversed with fine step. Nethertheless, the algorithm is able to render colored light shafts at more than 100 fps for the same quality settings

Advantages Over Previous Approaches

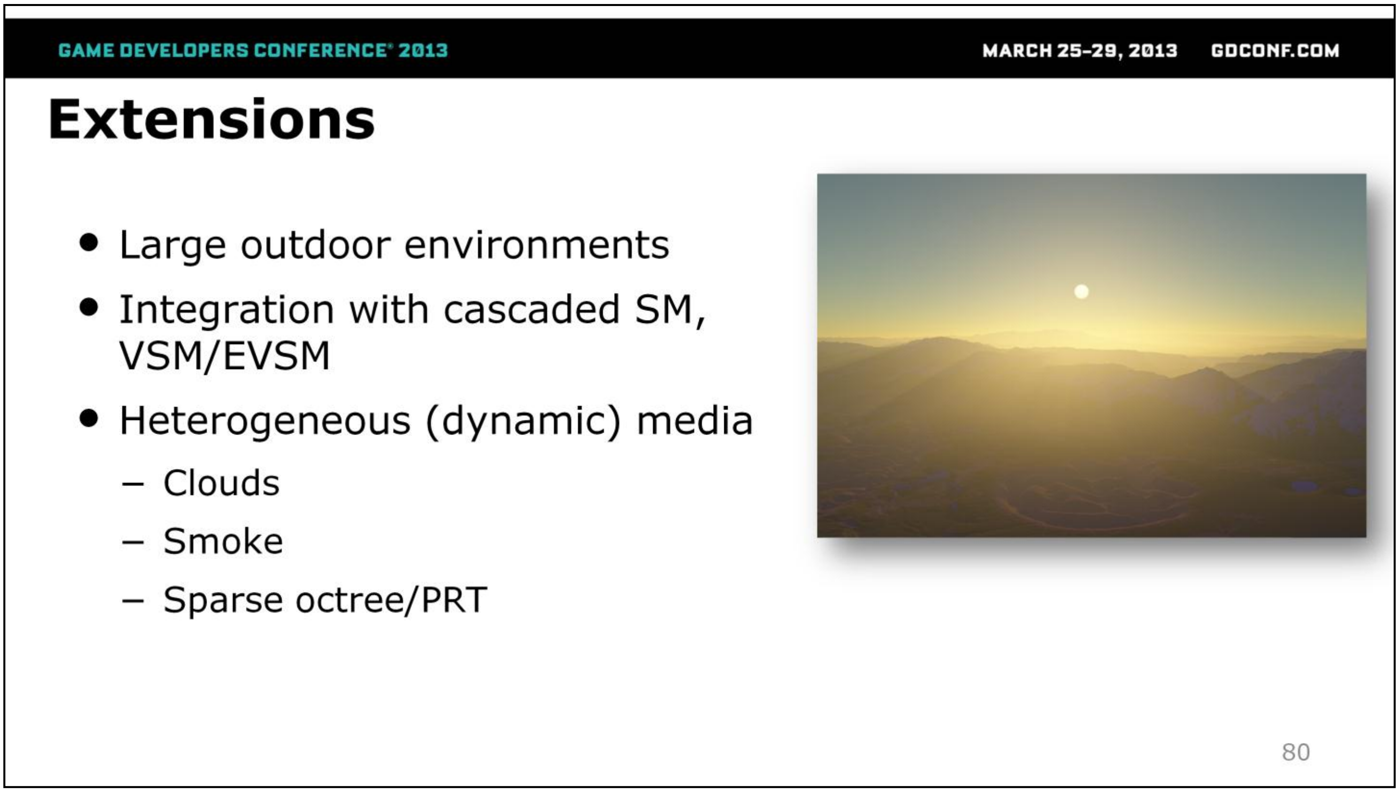
- Semi-analytical solution is used to calculate the integral on each ray segment
  - No sophisticated mathematics like SVD etc.
- Epipolar scattering is combined with 1D min/max binary trees
  - Significant performance improvements
- Efficient implementation methods



Integration Into Existing Apps

- The technique is fully post-processing
- The only inputs are:
  - Depth buffer
  - Back buffer
  - Shadow map
  - [optional] Stained glass color
- The source code is available at
  http://software.intel.com/en-us/blogs/2013/03/18/gtd-light-scattering-sample-updated
  http://software.intel.com/gamecode

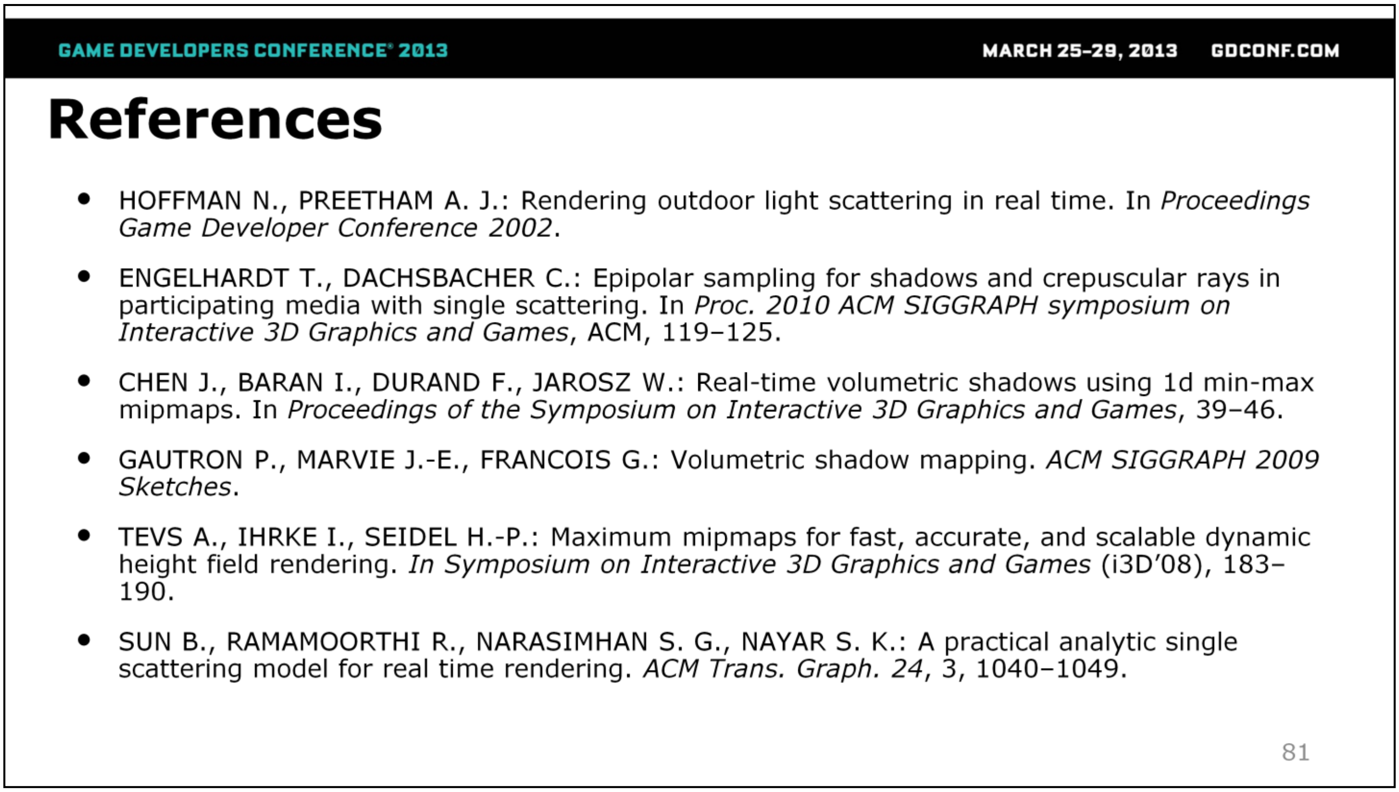
The technique is fully post processing and integration into game engines should not be very difficult



Extensions

- Large outdoor environments
- Integration with cascaded SM, VSM/EVSM
- Heterogeneous (dynamic) media
  - Clouds
  - Smoke
  - Sparse octree/PRT

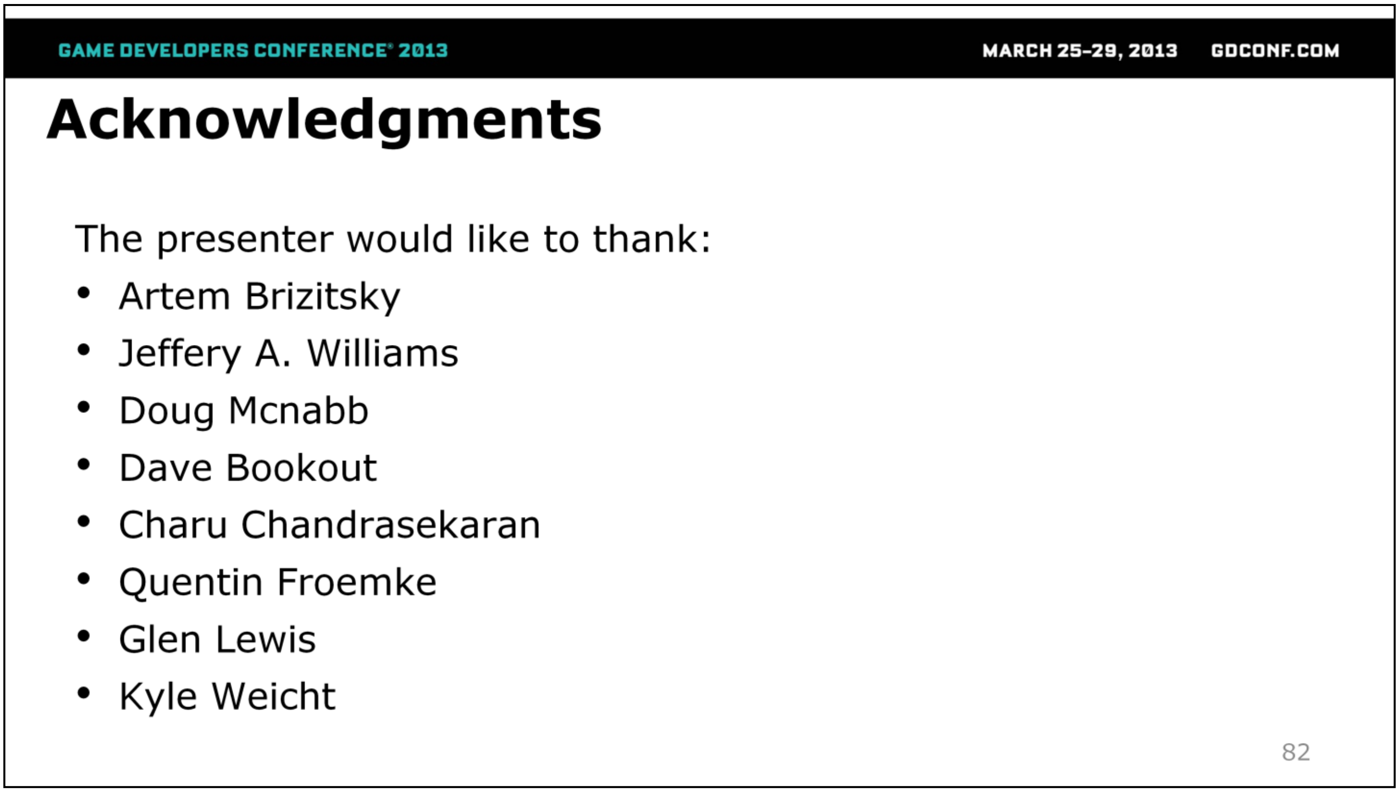
- There are a number of possible directions the technique could be improved. The first way is to apply this approach for rendering large outdoor environments. The first experiments are quite promising
- The technique can also be integrated with other shadowing techniques
- Another interesting area is implementing heterogeneous media like clouds or smoke
  - New technologies like PRT could be used her



References

- HOFFMAN N., PREETHAM A. J.: Rendering outdoor light scattering in real time. In *Proceedings Game Developer Conference 2002*.
- ENGELHARDT T., DACHSBACHER C.: Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. In *Proc. 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, 119–125.
- CHEN J., BARAN I., DURAND F., JAROSZ W.: Real-time volumetric shadows using 1d min-max mipmaps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 39–46.
- GAUTRON P., MARVIE J.-E., FRANCOIS G.: Volumetric shadow mapping. *ACM SIGGRAPH 2009 Sketches*.
- TEVS A., IHRKE I., SEIDEL H.-P.: Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. *In Symposium on Interactive 3D Graphics and Games* (i3D'08), 183–190.
- SUN B., RAMAMOORTHI R., NARASIMHAN S. G., NAYAR S. K.: A practical analytic single scattering model for real time rendering. *ACM Trans. Graph. 24*, 3, 1040–1049.

These are references to the most relevant work



Acknowledgments

The presenter would like to thank:

- Artem Brizitsky
- Jeffery A. Williams
- Doug Mcnabb
- Dave Bookout
- Charu Chandrasekaran
- Quentin Froemke
- Glen Lewis
- Kyle Weicht

Here are some people I would like to thank

Thank you!