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

Egor Yusov
Graphics Software Engineer – Intel Corporation
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

\[ L_{sc}^{(c, o, l)} = \int_{Sc} e^{-\tau_{c}(P-C)} \beta(P) \frac{e^{-\tau_{c}(L-P)}}{||L-P||^2} V(P) d\sigma \]

\[ \tau(A \rightarrow B) = \int_{A}^{B} \beta(P) d\sigma \]

\[ p_{o}(\theta) = \frac{1}{16\pi} (1 + \cos^2 \theta) \]

\[ p_{m}(\theta) = \frac{1}{4\pi} \frac{3(1 - g^2)}{2 + g^2} \frac{1 + \cos^2 \theta}{(1 + g^2 - 2g \cos \theta)^{3/2}} \]

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

**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

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.
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.

\[ L_{Cam} = L_{Obj} \cdot e^{-T(O \rightarrow C)} + L_{In} \]

For homogeneous medium

\[ T(A \rightarrow B) = \int_{A}^{B} \beta(P) ds = \beta \|A - B\| \]
Inscattering Integral Derivation

Radiant Intensity of a Point Light

- \( I \) – radiant intensity
- \( \hat{v} = \frac{o-c}{\|o-c\|} \) – view direction

\[
L_{ln} = I
\]

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

\[
L_{ln} = \frac{I e^{-T(L \rightarrow P)}}{||L - P||^2}
\]

- 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
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.
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 \( ds \)
- the scattering coefficient \( \beta(P) \)

\[
L_{in} = \beta(P) V(P) \frac{I e^{-T(L \rightarrow P)}}{||L - P||^2} ds
\]
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 $\Omega$. 
Inscattering Integral Derivation

Attenuation of inscattered light

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

\[ L_{ln} = \int_C^0 e^{-T(P \rightarrow C)} p(\theta) \beta(P) V(P) \frac{I e^{-T(L \rightarrow P)}}{\|L - P\|^2} \, ds \]

- 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.
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.
Following 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

#### Rayleigh scattering

- Caused by molecules
- Wavelength-dependent
  \[ \beta_R.\text{rgb} = (5.8, 13.5, 33.1) \times 10^{-6} \]
- Almost isotropic:
  \[ p_R(\theta) = \frac{3}{16\pi} (1 + \cos^2 \theta) \]
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.

\[ p_M(\theta) = \frac{3(1 - g^2)}{4\pi 2(2 + g^2)} \frac{1 + \cos^2\theta}{(1 + g^2 - 2g\cos\theta)^{3/2}} \]
In the presence of two constituents, 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 modulated by its own phase function.

We will denote total scattering coefficient by beta with subscript sigma and the sum of phase functions modulated by the scattering coefficient by capital P with subscript sigma.

\[ T(A \to B) = \beta_{\Sigma} ||A - B|| \]

\[ \beta_{\Sigma} = \beta_R + \beta_M \]

\[ \beta p(\theta) \to \beta_R \cdot p_R(\theta) + \beta_M \cdot p_M (\theta) \equiv P_{\Sigma} (\theta) \]
This picture shows rendering spot light source with isotropic phase function

\[ p_M(\theta) = \frac{1}{4\pi} \]
While anisotropic one creates a nice glow around the light source
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 $\theta$ between the view ray $\vec{v}$ and the light direction $\vec{L}$.

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

$$S = ||\vec{O} - \vec{C}||$$
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.
To solve inscattering integral, we can precompute the inscattering integral for different values of $h$ and start distance $S_c$ to infinity. These values can be stored in a two-dimensional look-up table.
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
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
For a directional light source there is an analytical solution which does not require look-up tables.

Note that in this case the $F$ 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.
So as our starting point we can derive the algorithm which does the following:

• Projects the view ray onto the shadow map
• Sets up $\text{Prev}_{\text{In}}\cdot\text{rgb} = L[h, S_c]$, $\text{In}_{\text{In}}\cdot\text{rgb} = 0$
• Step through each texel:
  
  $\begin{align*}
  -\text{Curr}_{\text{In}}\cdot\text{rgb} &= e^{-\beta_c(S-S_c)} L[h, S] \\
  -\text{In}_{\text{In}}\cdot\text{rgb} &= (\text{Prev}_{\text{In}}\cdot\text{rgb} - \text{Curr}_{\text{In}}\cdot\text{rgb}) \cdot V(P) \\
  -\text{Prev}_{\text{In}}\cdot\text{rgb} &= \text{Curr}_{\text{In}}\cdot\text{rgb}
  \end{align*}$

• The next texel is then processed until the whole ray is marched
• Note that only one exponent and/or look-up has to be performed in the loop.
An important aspect of this algorithm is that we perform ray marching in the light projection space. Thus we cannot 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.
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 inscatterering from epipolar coordinates back to rectangular.
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.
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.
• 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.
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.
• 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
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.
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.
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
These are the nodes of the second level
And here are the nodes of the third level
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.
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.
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 \( S_{\text{Exit}} \) and taking a vector from the camera through this point

• Epipolar plane normal can be computed as a cross product of \( S_1 \) and \( S_0 \)

• 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.
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.
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 $P_0$ 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 $O = P_0 + t_{\text{min}} \cdot \vec{D}$. 

The diagram illustrates the following:
- $P_0 = c_1 \vec{N}_{\text{Slice}} + c_2 \vec{N}_{\text{Light}} + t\vec{D}$
- $O = P_0 + t_{\text{min}} \cdot \vec{D}$
- $\vec{N}_{\text{Slice}}$ and $\vec{N}_{\text{Light}}$ represent the normal vectors for the cone slice and light cone, respectively.
- $t_{\text{min}}$ and $t_{\text{max}}$ are the minimum and maximum intersection times.

The diagram visualizes the shadow map plane and the cone, with rays emanating from the camera and intersecting with the cone.
The same method works for directional light source if we replace \( S_0 \) with the negated light direction.

To obtain slice origin \( O_{UV} \) in shadow map UV coordinates, it is necessary to transform \( O \) with the shadow map transform matrix. Since \( O \) 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.
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.
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.
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.
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.
• 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
* Finally, inscattering fix-up pass is performed for pixels marked in stencil. No 1D min/max optimization is used at this stage.
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.
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 implementation 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
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.
  - 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

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.
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 $[-1 + \frac{1}{W}, 1 - \frac{1}{W}] \times [-1 + \frac{1}{H}, 1 - \frac{1}{H}]$ 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.
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.
This is the reference picture for brute force ray marching
This is the high quality profile which is almost identical to the reference picture
In balanced quality profile, rays loose crisper appearance because lower resolution shadow map is used
In high performance profile, the rays are even more smoother, however, no apparent artifacts like banding are noticeable.
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.
Again this is the reference picture for brute force ray marching
And the high quality profile is almost identical to the reference picture
This is the balanced quality where rays become a bit smoother
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.
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.
Here is the comparison for different quality settings
The high quality profile is almost identical to the reference picture
This is the balanced quality where rays become a bit smoother
Again, even in high performance profile, the effect quality is rather good
This chart presents detailed timings for another scene rendered on Nvidia discrete GPU. Total processing time is less than 3.4 ms.
Here is the performance for another test for directional light source. For 4k x 4k shadow map, the technique requires just above 4 ms which is very good result.
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.
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. Nevertheless, 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/gamecode

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