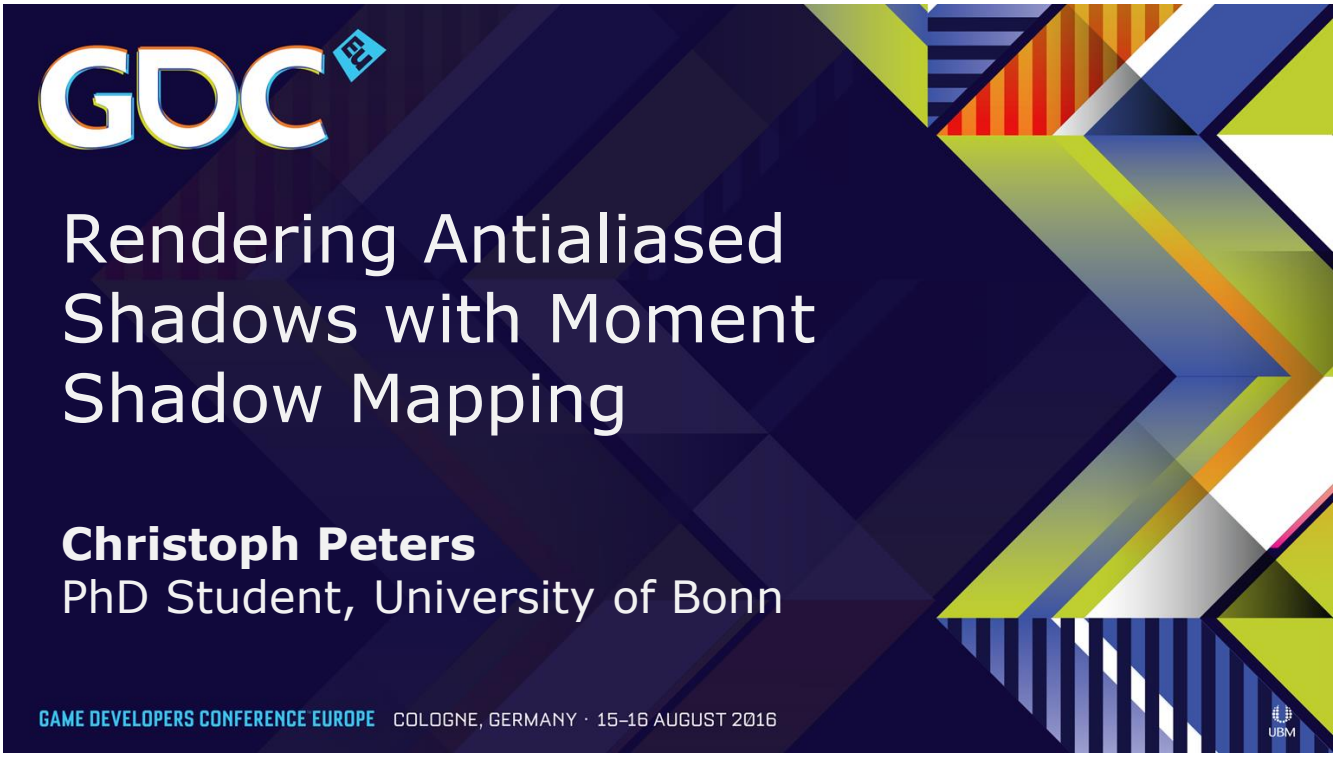


Welcome to this lecture on rendering antialiased shadows with moment shadow mapping. My name is Christoph Peters.

About this PDF

- Original slides: Lots of videos, some animations,
- PDF: Reformatted for static display,
- Don‘t miss out:
  - Video version with audio is available,
  - Powerpoint version is available,
- MomentsInGraphics.de

You are viewing a version of the slides that has been reformatted for static display in a PDF. The original slides contain lots of videos and some animations. At many points the PDF version may be less clear. If you want to get a more complete version, a video with audio or the original Powerpoint version are available at http://MomentsInGraphics.de/ .

Introduction

- I'm not in the games industry (yet),
- Research done as PhD student at the University of Bonn,
- Papers presented at the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D) in 2015 and 2016.

universitätbonn

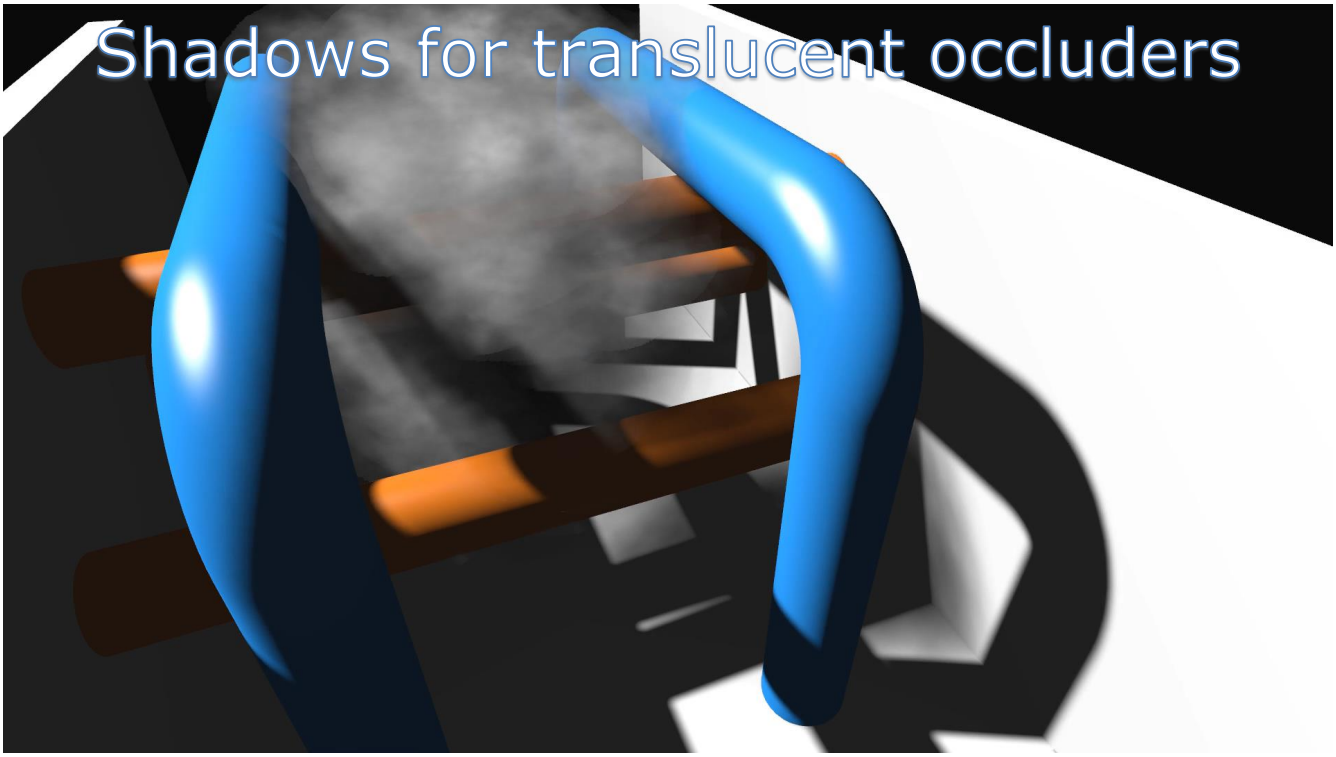
i3D

Unlike most speakers here I am not in the games industry. At least not yet. CLICK

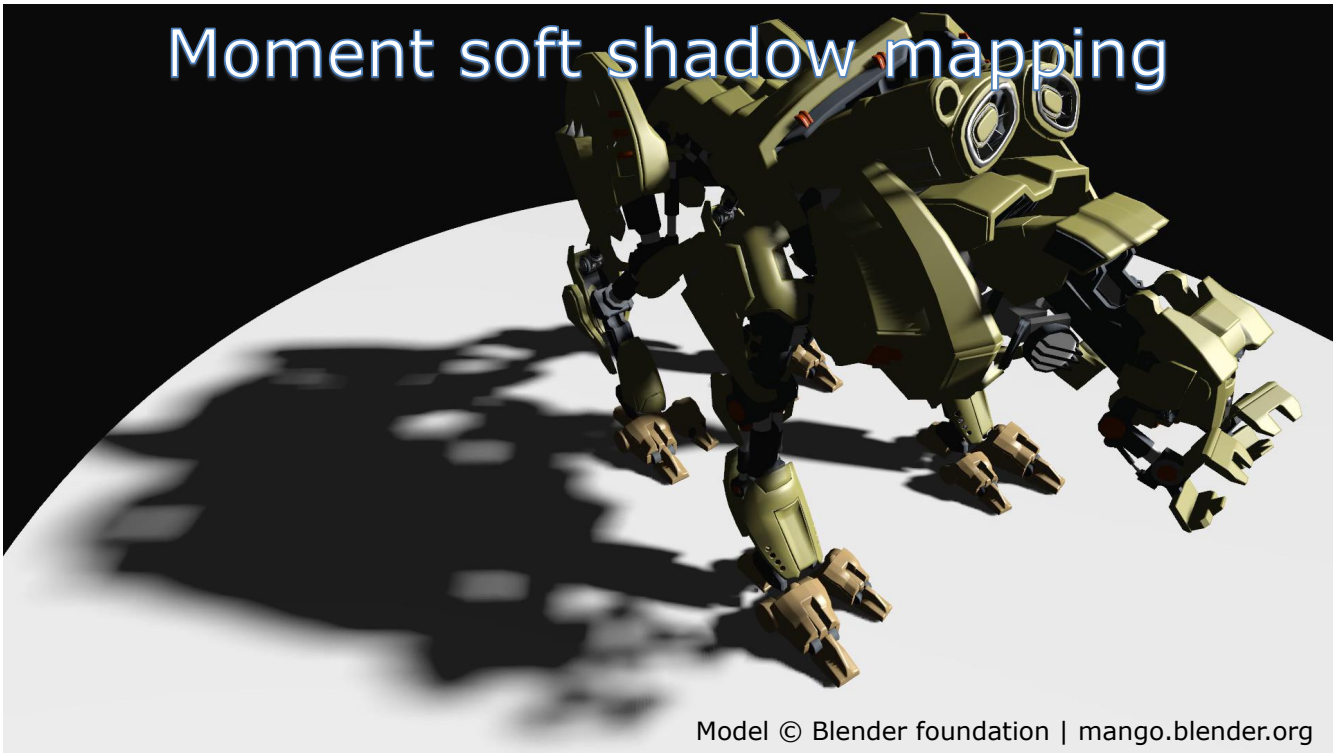
I will be presenting work that I did as PhD student at the University of Bonn in the computer graphics group of Reinhard Klein. CLICK

In particular, I will be speaking about papers published at i3D 2015 and i3D 2016.

Maybe you have seen some of this before. The first technique I will be speaking about is moment shadow mapping for filtered hard shadows. As you can see, these shadows exhibit little aliasing and appear correct throughout the scene.



Moment shadow mapping is not restricted to opaque occluders. We can also render shadows for translucent occluders such as this smoke plume.



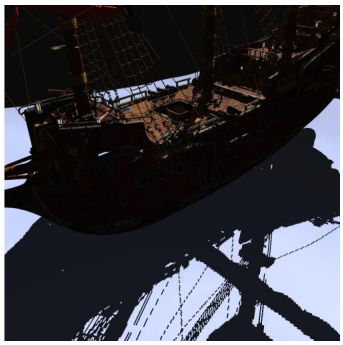
The approach also enables soft shadows for area lights. Note how these shadows harden at contact points.

Even single scattering in homogenous participating media can be rendered efficiently at high resolutions without any upscaling. We will cover all these techniques in detail.

To get started lets recall some basics of shadow map filtering.

Shadow mapping [Williams78]

Shadow map



Scene

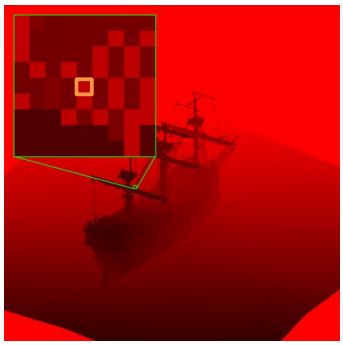
- Shadow map stores light space depth,
- Shadow test with one lookup,
- Aliasing ☹ .

In modern real-time applications, dynamic shadows are typically rendered by means of shadow maps. To the left, you can see a shadow map. It is an image rendered from the point of view of the light source. Each texel stores the depth in light space of the visible surface. CLICK

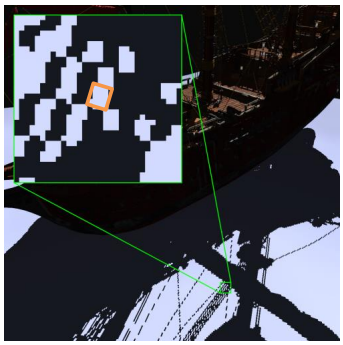
If we want to shade a pixel on screen, we look up the corresponding depth in the shadow map. If the pixel is lit, its actual depth should agree with the depth stored in the shadow map. CLICK

This shadow test is very efficient, but it is an image-based approach without any filtering. Therefore, we get severe aliasing.

Shadow map aliasing



Shadow map



Scene

- One texel,
- Covers many pixels,
- Fine features lost,
- Unstable.

To better understand this artifact, we look at a single texel in the shadow map. CLICK
This texel maps to many pixels on screen. CLICK
Therefore, fine features in the shadows are not resolved properly and the results are very unstable.

Shadow map sampling

- Diminish undersampling with a better shadow map projection, e.g.:
  - Trapezoidal shadow maps [Martin04],
  - Cascaded shadow maps [Zhang06],
  - Sample distribution shadow maps [Lauritzen11].
- Helps a lot, but not enough,
- Not our topic today.

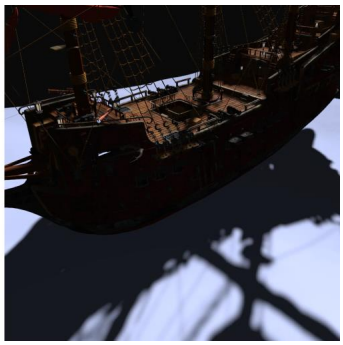
One way to diminish such undersampling artifacts is to use a better shadow map projection. Available techniques are trapezoidal shadow maps, cascaded shadow maps and sample distribution shadow maps. CLICK

All of these techniques help a lot but by themselves they are not enough to get rid of aliasing. CLICK

I do recommend that you use these techniques but they are not our topic today.

Percentage-closer filtering [Reeves87]

Shadow map



Scene

- Filter shadows,
- Oldest solution,
- Still widely used,
- Aliasing ☹ .

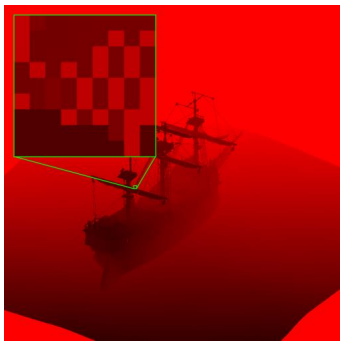
To reduce aliasing further, we need to filter our shadows. CLICK

The oldest technique for this purpose is percentage-closer filtering. CLICK

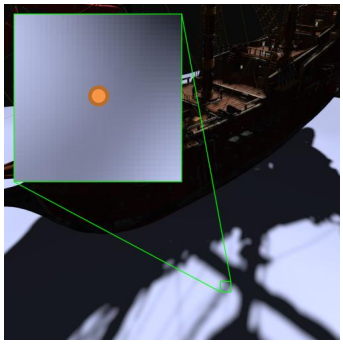
It is still widely used and if you do not know what you are using, you are probably using percentage-closer filtering. CLICK

It diminishes aliasing but aliasing introduced during generation of the shadow map remains.

Percentage-closer filtering



Shadow map



Scene

- Per pixel:

We now take a detailed look at its inner workings. Lets say we want to shade the pixel highlighted to the right. CLICK

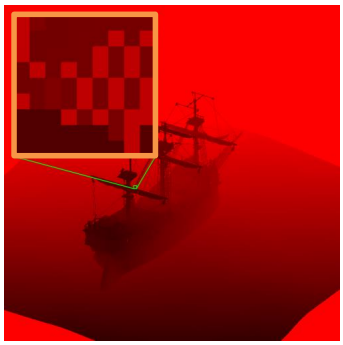
Then we sample a neighborhood of this pixel in the shadow map. CLICK

For every single sample, we perform the shadow test to get a binary shadow intensity. CLICK

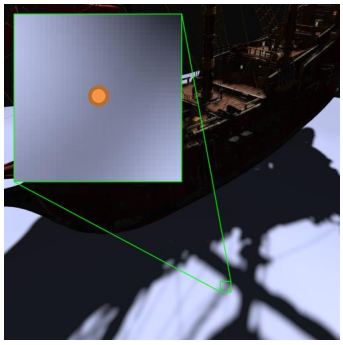
These shadow intensities can be filtered, for example by means of a Gaussian filter. We get a filtered shadow intensity which can be anywhere between zero and one. CLICK

Of course this procedure is quite costly because we need many shadow map samples for every single pixel on screen.

Percentage-closer filtering



Shadow map



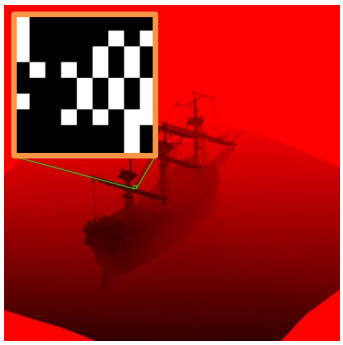
Scene

- Per pixel:

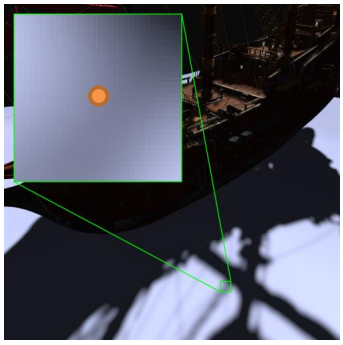
1. Sample filter region,

We now take a detailed look at its inner workings. Lets say we want to shade the pixel highlighted to the right. CLICK
Then we sample a neighborhood of this pixel in the shadow map. CLICK
For every single sample, we perform the shadow test to get a binary shadow intensity. CLICK
These shadow intensities can be filtered, for example by means of a Gaussian filter. We get a filtered shadow intensity which can be anywhere between zero and one. CLICK
Of course this procedure is quite costly because we need many shadow map samples for every single pixel on screen.

Percentage-closer filtering



Shadow map

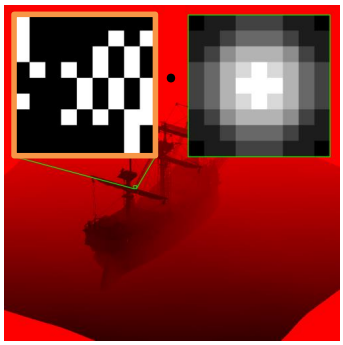
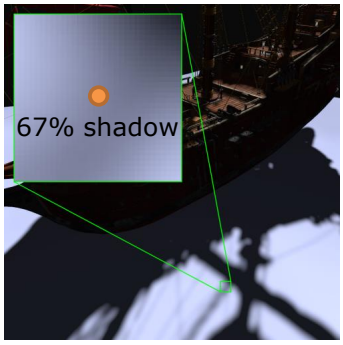


Scene

- Per pixel:
1. Sample filter region,
2. Threshold,

We now take a detailed look at its inner workings. Lets say we want to shade the pixel highlighted to the right. CLICK
Then we sample a neighborhood of this pixel in the shadow map. CLICK
For every single sample, we perform the shadow test to get a binary shadow intensity. CLICK
These shadow intensities can be filtered, for example by means of a Gaussian filter. We get a filtered shadow intensity which can be anywhere between zero and one. CLICK
Of course this procedure is quite costly because we need many shadow map samples for every single pixel on screen.

Percentage-closer filtering

$\sum$  = 

Shadow map Scene

- Per pixel:
1. Sample filter region,
2. Threshold,
3. Filter.
- Costly!

We now take a detailed look at its inner workings. Lets say we want to shade the pixel highlighted to the right. CLICK

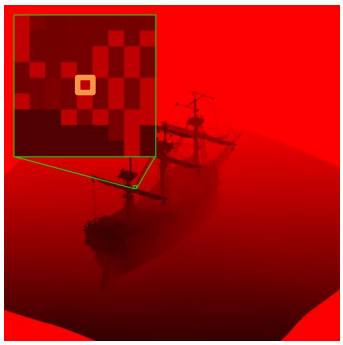
Then we sample a neighborhood of this pixel in the shadow map. CLICK

For every single sample, we perform the shadow test to get a binary shadow intensity. CLICK

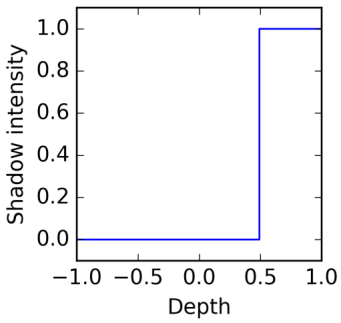
These shadow intensities can be filtered, for example by means of a Gaussian filter. We get a filtered shadow intensity which can be anywhere between zero and one. CLICK

Of course this procedure is quite costly because we need many shadow map samples for every single pixel on screen.

Depth distributions



Shadow map

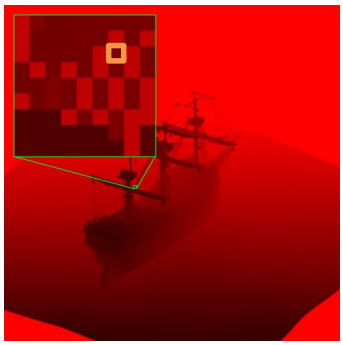


Depth distribution

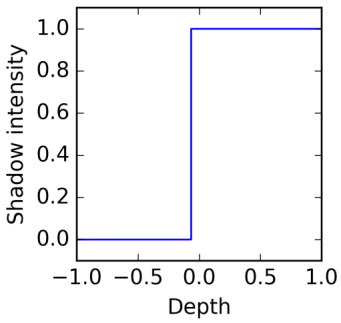
- Shadow as function of depth,

To improve on this situation, we need to interpret percentage-closer filtering a bit differently. CLICK
A texel of the shadow map (left) provides us with a depth-dependentshadow intensity for the corresponding light-ray (right). Up to the stored depth everything is fully lit. Beyond this depth there is full shadow. CLICK
We get such a function for every texel. CLICK
CLICK
Now if we build a weighted combination of these functions using weights from a filter kernel, we get the depth-dependent shadow intensity computed by percentage closer filtering. It is a monotonous function between zero and one modeling the distribution of depth values in the filter region. Our goal is to precompute compact representations of such depth distributions.

Depth distributions



Shadow map

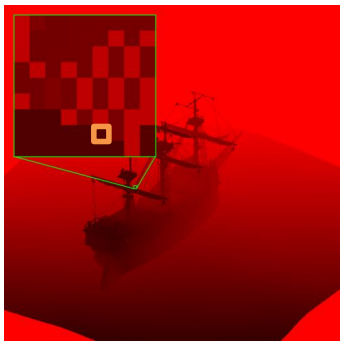


Depth distribution

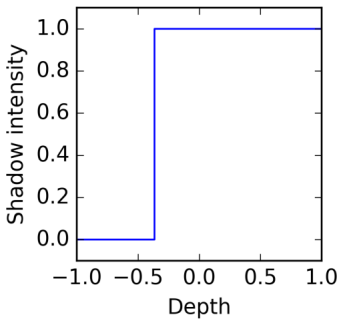
- Shadow as function of depth,
- Step for one texel,

To improve on this situation, we need to interpret percentage-closer filtering a bit differently. CLICK
A texel of the shadow map (left) provides us with a depth-dependentshadow intensity for the corresponding light-ray (right). Up to the stored depth everything is fully lit. Beyond this depth there is full shadow. CLICK
We get such a function for every texel. CLICK
CLICK
Now if we build a weighted combination of these functions using weights from a filter kernel, we get the depth-dependent shadow intensity computed by percentage closer filtering. It is a monotonous function between zero and one modeling the distribution of depth values in the filter region. Our goal is to precompute compact representations of such depth distributions.

Depth distributions



Shadow map

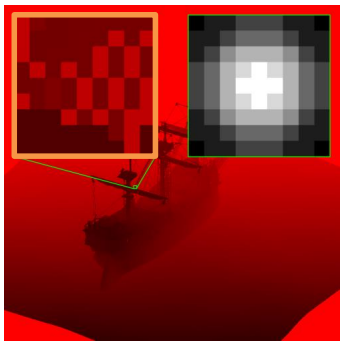


Depth distribution

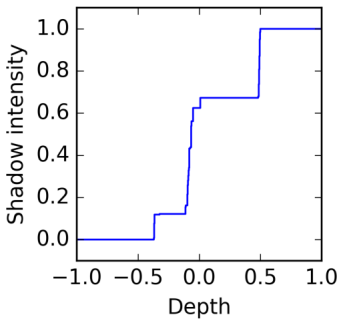
- Shadow as function of depth,
- Step for one texel,

To improve on this situation, we need to interpret percentage-closer filtering a bit differently. CLICK
A texel of the shadow map (left) provides us with a depth-dependentshadow intensity for the corresponding light-ray (right). Up to the stored depth everything is fully lit. Beyond this depth there is full shadow. CLICK
We get such a function for every texel. CLICK
CLICK
Now if we build a weighted combination of these functions using weights from a filter kernel, we get the depth-dependent shadow intensity computed by percentage closer filtering. It is a monotonous function between zero and one modeling the distribution of depth values in the filter region. Our goal is to precompute compact representations of such depth distributions.

Depth distributions



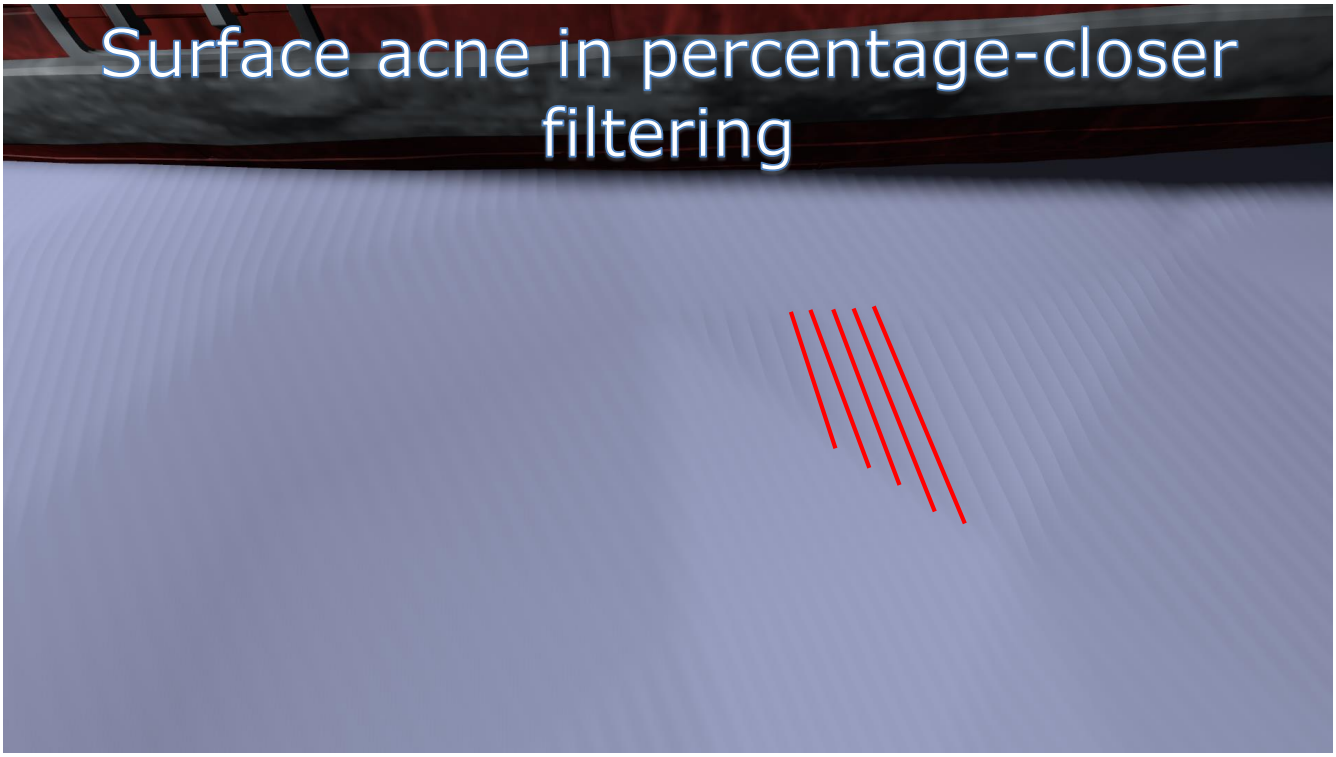
Shadow map



Depth distribution

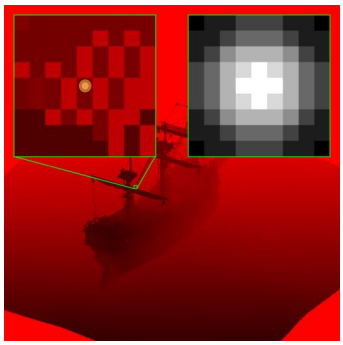
- Shadow as function of depth,
- Step for one texel,
- Monotonous in general.

To improve on this situation, we need to interpret percentage-closer filtering a bit differently. CLICK
A texel of the shadow map (left) provides us with a depth-dependentshadow intensity for the corresponding light-ray (right). Up to the stored depth everything is fully lit. Beyond this depth there is full shadow. CLICK
We get such a function for every texel. CLICK
CLICK
Now if we build a weighted combination of these functions using weights from a filter kernel, we get the depth-dependent shadow intensity computed by percentage closer filtering. It is a monotonous function between zero and one modeling the distribution of depth values in the filter region. Our goal is to precompute compact representations of such depth distributions.

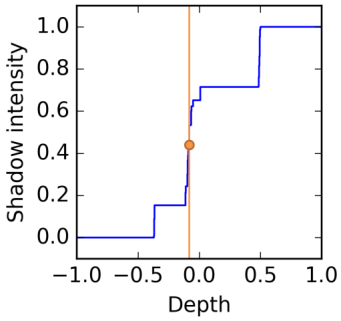


This mode of visualization also helps to understand an artifact of percentage-closer filtering that is shown here. The ground is lit at a grazing angle and percentage-closer filtering computes that parts of the ground shadow nearby parts of the ground. This is an artifact known as surface acne. CLICK Combined with the discretization of the shadow map it leads to obvious stripe patterns.

Surface acne and remedies



Shadow map



Depth distribution

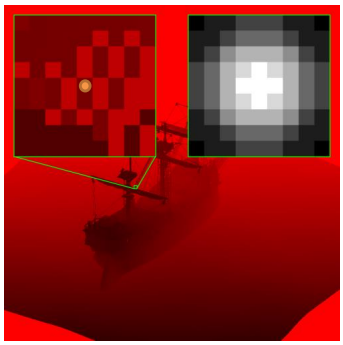
- Surface has many depths,
- Shadows itself,

To better understand this artifact, lets look at it in terms of depth distributions. To the left, you can see a filter region with a pixel marked for which we want to compute a shadow. The depth distribution on the right immediately provides the shadow intensity for the pixel depth. However, many texels belonging to the same surface contribute to this shadow. It is darker than it should be. CLICK

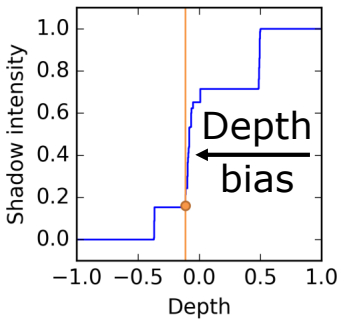
This artifact can be eliminated by pushing the depth of the pixel to the left using a depth bias. CLICK

However, determining this depth bias is an art in itself. And by that I mean that your artists will waste a lot of time doing it.

Surface acne and remedies



Shadow map



Depth distribution

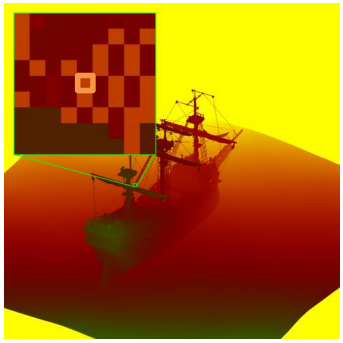
- Surface has many depths,
- Shadows itself,
- Apply bias to pixel depth,
- An art in itself.

To better understand this artifact, lets look at it in terms of depth distributions. To the left, you can see a filter region with a pixel marked for which we want to compute a shadow. The depth distribution on the right immediately provides the shadow intensity for the pixel depth. However, many texels belonging to the same surface contribute to this shadow. It is darker than it should be. CLICK

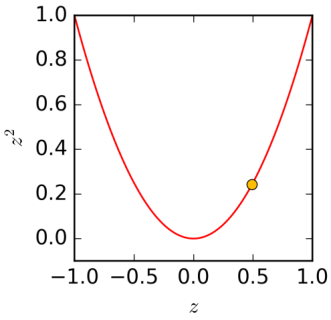
This artifact can be eliminated by pushing the depth of the pixel to the left using a depth bias. CLICK

However, determining this depth bias is an art in itself. And by that I mean that your artists will waste a lot of time doing it.

Variance shadow maps [Donnelly06]



R=z
G=z^2

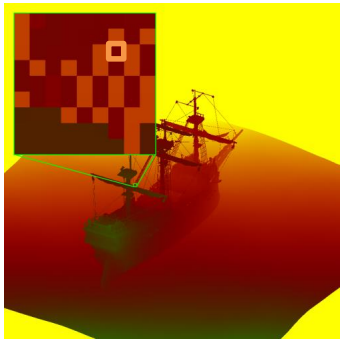


Stored data

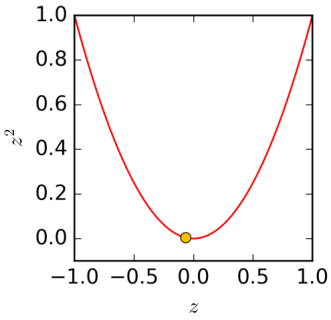
- Store z, z^2,
- Redundant,

Variance shadow maps provide a way to filter shadow maps more efficiently than by means of percentage-closer filtering. Here to the left, you see such a variance shadow map. It is a two-channel texture. The red channel is the same as for a common shadow map. However, the green channel stores the squared depth. Of course this information is completely redundant for every individual texel. We basically store a point on a parabola (right). If we know the x-coordinate, we know the y-coordinate. CLICK
Again this is the case for every texel. CLICK
CLICK
However, this changes if we apply a filter (e.g. a Gaussian blur) to the variance shadow map. The stored point is now the center of mass of all the points stored by the individual texels that enter the filter. The point does not lie on the parabola anymore and we actually get some additional information about the depth distribution. CLICK
Following notations from probability theory, the values stored in the filtered variance shadow map are called first and second moment of the depth distribution. CLICK
This information about the depth distribution can be used for an approximate reconstruction. The result of percentage-closer filtering is shown in blue, the approximation of variance shadow mapping is shown in green. The shadow intensity is always underestimated to avoid surface acne. CLICK
This approximation may look very coarse but at least it is good at some important points where there is actually a surface receiving shadow. CLICK
On the other hand, it converges to one slowly and this will cause visible artifacts.

Variance shadow maps [Donnelly06]



R=z
G=z^2

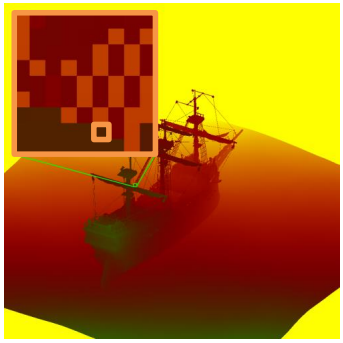


Stored data

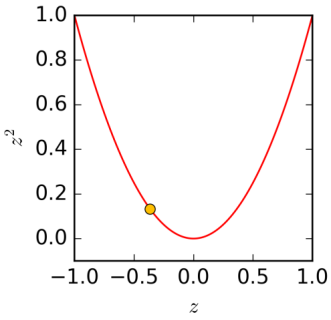
- Store z, z^2,
- Redundant,

Variance shadow maps provide a way to filter shadow maps more efficiently than by means of percentage-closer filtering. Here to the left, you see such a variance shadow map. It is a two-channel texture. The red channel is the same as for a common shadow map. However, the green channel stores the squared depth. Of course this information is completely redundant for every individual texel. We basically store a point on a parabola (right). If we know the x-coordinate, we know the y-coordinate. CLICK
Again this is the case for every texel. CLICK
CLICK
However, this changes if we apply a filter (e.g. a Gaussian blur) to the variance shadow map. The stored point is now the center of mass of all the points stored by the individual texels that enter the filter. The point does not lie on the parabola anymore and we actually get some additional information about the depth distribution. CLICK
Following notations from probability theory, the values stored in the filtered variance shadow map are called first and second moment of the depth distribution. CLICK
This information about the depth distribution can be used for an approximate reconstruction. The result of percentage-closer filtering is shown in blue, the approximation of variance shadow mapping is shown in green. The shadow intensity is always underestimated to avoid surface acne. CLICK
This approximation may look very coarse but at least it is good at some important points where there is actually a surface receiving shadow. CLICK
On the other hand, it converges to one slowly and this will cause visible artifacts.

Variance shadow maps [Donnelly06]



R=z
G=z^2

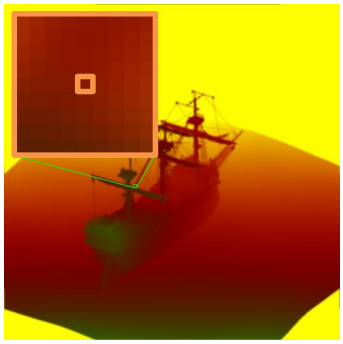


Stored data

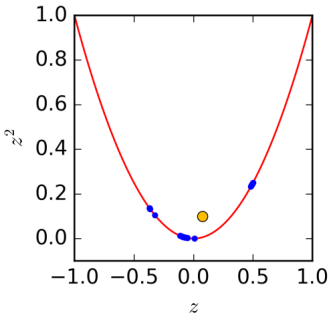
- Store z, z^2,
- Redundant,

Variance shadow maps provide a way to filter shadow maps more efficiently than by means of percentage-closer filtering. Here to the left, you see such a variance shadow map. It is a two-channel texture. The red channel is the same as for a common shadow map. However, the green channel stores the squared depth. Of course this information is completely redundant for every individual texel. We basically store a point on a parabola (right). If we know the x-coordinate, we know the y-coordinate. CLICK
Again this is the case for every texel. CLICK
CLICK
However, this changes if we apply a filter (e.g. a Gaussian blur) to the variance shadow map. The stored point is now the center of mass of all the points stored by the individual texels that enter the filter. The point does not lie on the parabola anymore and we actually get some additional information about the depth distribution. CLICK
Following notations from probability theory, the values stored in the filtered variance shadow map are called first and second moment of the depth distribution. CLICK
This information about the depth distribution can be used for an approximate reconstruction. The result of percentage-closer filtering is shown in blue, the approximation of variance shadow mapping is shown in green. The shadow intensity is always underestimated to avoid surface acne. CLICK
This approximation may look very coarse but at least it is good at some important points where there is actually a surface receiving shadow. CLICK
On the other hand, it converges to one slowly and this will cause visible artifacts.

Variance shadow maps [Donnelly06]



R=1st moment
G=2nd moment

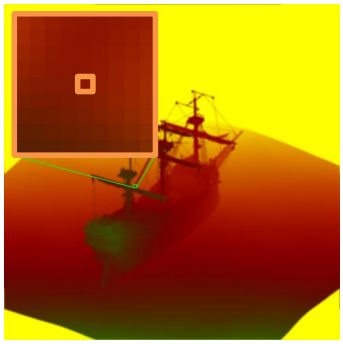


Stored data

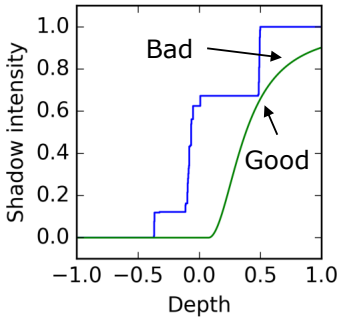
- Store z, z^2,
- Redundant,
- Until filtered,
- 2 moments,

Variance shadow maps provide a way to filter shadow maps more efficiently than by means of percentage-closer filtering. Here to the left, you see such a variance shadow map. It is a two-channel texture. The red channel is the same as for a common shadow map. However, the green channel stores the squared depth. Of course this information is completely redundant for every individual texel. We basically store a point on a parabola (right). If we know the x-coordinate, we know the y-coordinate. CLICK
Again this is the case for every texel. CLICK
CLICK
However, this changes if we apply a filter (e.g. a Gaussian blur) to the variance shadow map. The stored point is now the center of mass of all the points stored by the individual texels that enter the filter. The point does not lie on the parabola anymore and we actually get some additional information about the depth distribution. CLICK
Following notations from probability theory, the values stored in the filtered variance shadow map are called first and second moment of the depth distribution. CLICK
This information about the depth distribution can be used for an approximate reconstruction. The result of percentage-closer filtering is shown in blue, the approximation of variance shadow mapping is shown in green. The shadow intensity is always underestimated to avoid surface acne. CLICK
This approximation may look very coarse but at least it is good at some important points where there is actually a surface receiving shadow. CLICK
On the other hand, it converges to one slowly and this will cause visible artifacts.

Variance shadow maps [Donnelly06]



R=1st moment
G=2nd moment



- Store z, z^2,
- Redundant,
- Until filtered,
- 2 moments,
- Reconstruct with lower bound.

Variance shadow maps provide a way to filter shadow maps more efficiently than by means of percentage-closer filtering. Here to the left, you see such a variance shadow map. It is a two-channel texture. The red channel is the same as for a common shadow map. However, the green channel stores the squared depth. Of course this information is completely redundant for every individual texel. We basically store a point on a parabola (right). If we know the x-coordinate, we know the y-coordinate. CLICK

Again this is the case for every texel. CLICK

CLICK

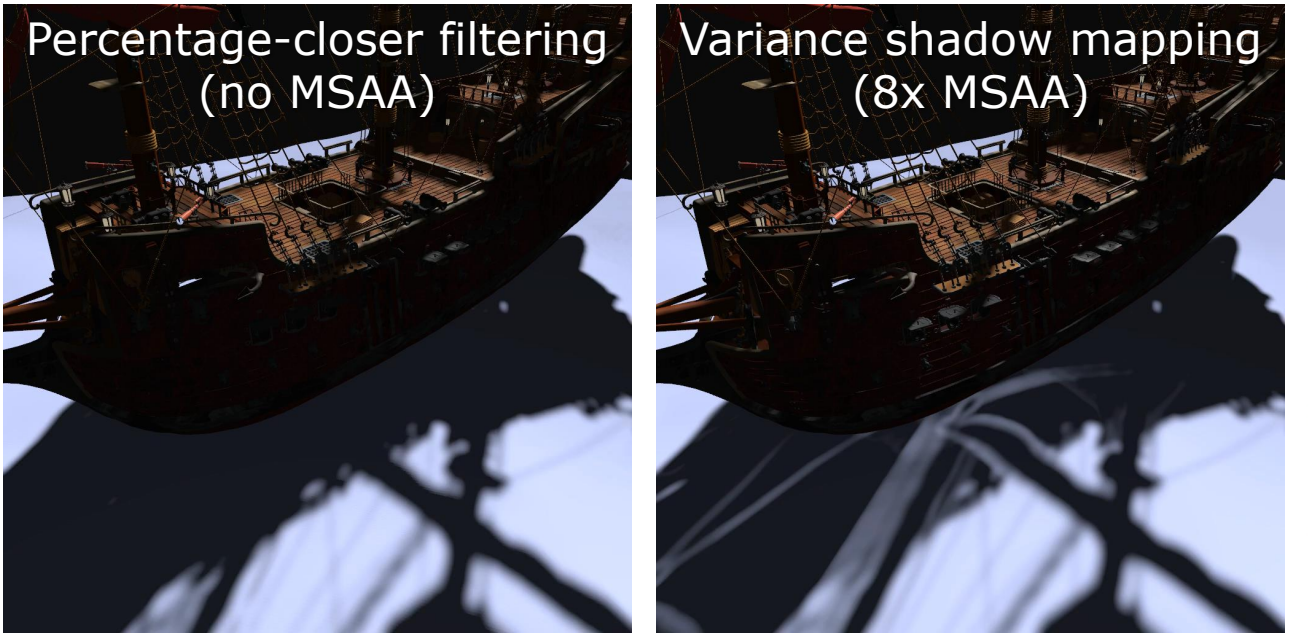
However, this changes if we apply a filter (e.g. a Gaussian blur) to the variance shadow map. The stored point is now the center of mass of all the points stored by the individual texels that enter the filter. The point does not lie on the parabola anymore and we actually get some additional information about the depth distribution. CLICK

Following notations from probability theory, the values stored in the filtered variance shadow map are called first and second moment of the depth distribution. CLICK

This information about the depth distribution can be used for an approximate reconstruction. The result of percentage-closer filtering is shown in blue, the approximation of variance shadow mapping is shown in green. The shadow intensity is always underestimated to avoid surface acne. CLICK

This approximation may look very coarse but at least it is good at some important points where there is actually a surface receiving shadow. CLICK

On the other hand, it converges to one slowly and this will cause visible artifacts.



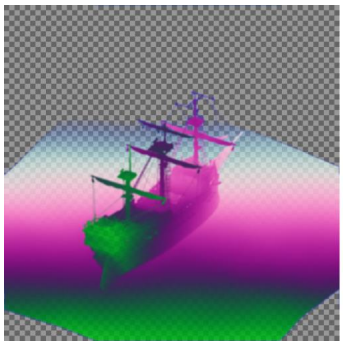
Here you can see the result of variance shadow mapping (right) next to percentage- closer filtering (left). We make two observations. Variance shadow mapping exhibits substantially less aliasing. This is because we use 8x multisample antialiasing when we generate the variance shadow map. For percentage-closer filtering this is not possible in an efficient manner and aliasing remains. On the other hand, variance shadow mapping gives us some very obvious artifacts. Silhouettes of shadow casters are visible as bright lines in the shadows. This is known as light leaking and it is simply due to the slow convergence to the maximal shadow intensity that I showed you on the previous slide.

Other filterable shadow maps

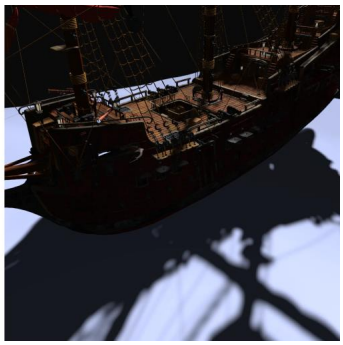
- Convolution shadow maps [Annen07] store Fourier coefficients,
- Exponential shadow maps [Salvi08,Annen08] store $\exp(c \cdot z)$,
- Exponential variance shadow maps [Lauritzen08] store $\exp(c_+ \cdot z)$, $\exp(c_+ \cdot z)^2$, $\exp(-c_- \cdot z)$, $\exp(-c_- \cdot z)^2$.

Various similar approaches have been proposed, such as convolution shadow maps, exponential shadow maps and exponential variance shadow maps. They all use shadow maps with some number of channels and store some vector that is derived from the depth. Exponential variance shadow maps used to be the most practical technique using 64 bits per texel. It has been used in several shipping titles.

Moment shadow mapping (ours)



R,G,B,A store
z, z^2, z^3, z^4



Scene

- 4 channels,
- 4 moments,
- 64 bits per texel,
- Little light leaking.

Basically, moment shadow mapping is yet another technique in this branch but it is a good one. It uses a shadow map with four channels. The channels store four powers of the depth. After filtering, these are four moments of the depth distribution. Note that we use a checkerboard to visualize the alpha channel. CLICK

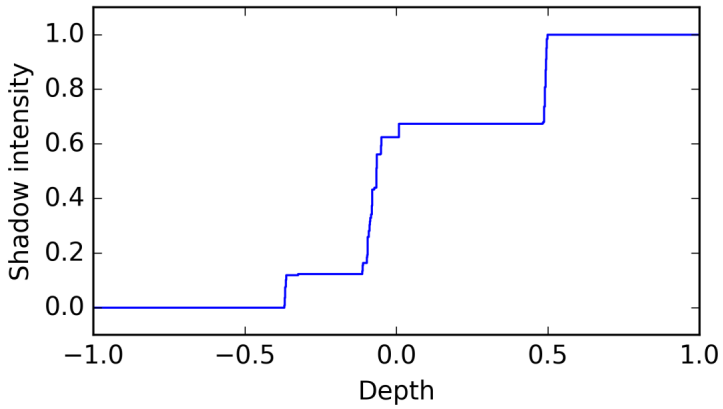
The four moments fit into 64 bits. CLICK

The result is a high-quality heuristic for filtered hard shadows at a moderate cost. There is little light leaking and multisample antialiasing is applicable.

Lets take a closer look.

From moments to shadows

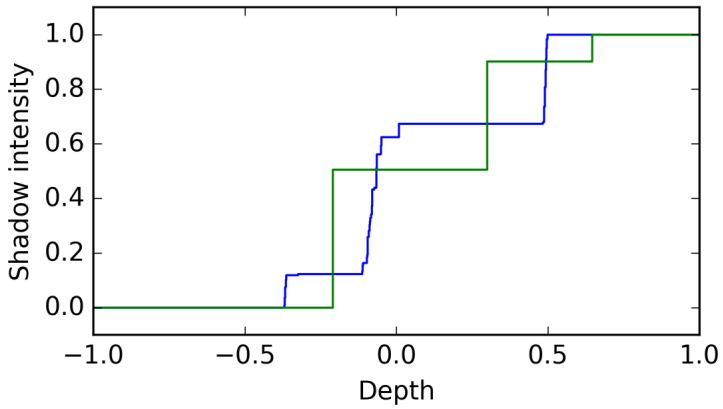
Many distributions share same 4 moments.



We want to reconstruct depth distributions from their moments. But this problem is ill-defined. Here is an example of a depth distribution. CLICK
Here is another one. Its moments are the same as for the first distribution. CLICK
This is also true for this distribution CLICK
and this one CLICK
and all of these. CLICK
There is not much we can tell about the shape of these functions but we can tell that they have to lie within specific bounds. The lower bound is what we will use as approximation to the shadow intensity. CLICK
This way, we avoid surface acne. At the same time we make the most of our data because knowing only the moments, the ground truth may agree with the lower bound.

From moments to shadows

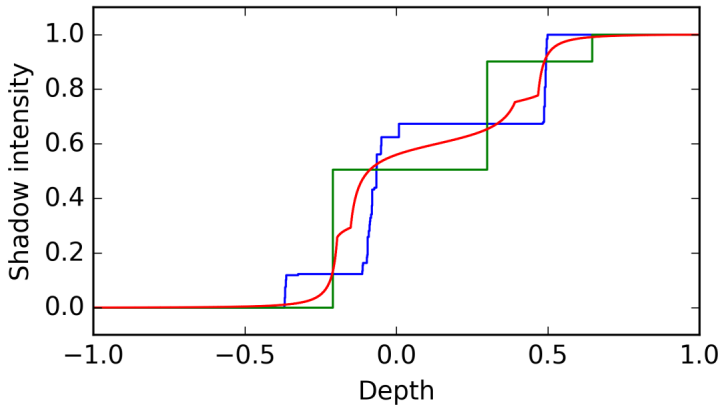
Many distributions share same 4 moments.



We want to reconstruct depth distributions from their moments. But this problem is ill-defined. Here is an example of a depth distribution. CLICK
Here is another one. Its moments are the same as for the first distribution. CLICK
This is also true for this distribution CLICK
and this one CLICK
and all of these. CLICK
There is not much we can tell about the shape of these functions but we can tell that they have to lie within specific bounds. The lower bound is what we will use as approximation to the shadow intensity. CLICK
This way, we avoid surface acne. At the same time we make the most of our data because knowing only the moments, the ground truth may agree with the lower bound.

From moments to shadows

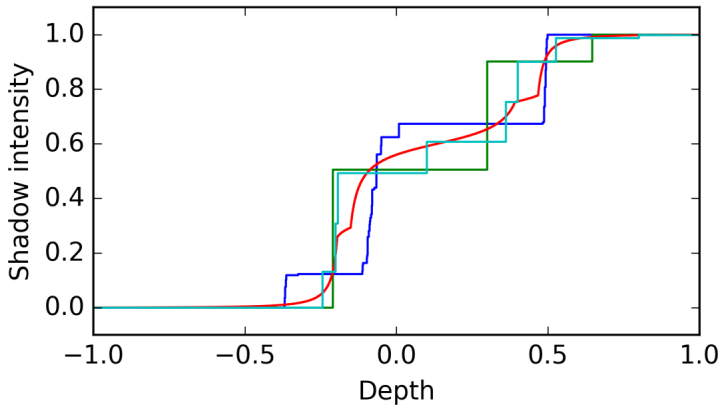
Many distributions share same 4 moments.



We want to reconstruct depth distributions from their moments. But this problem is ill-defined. Here is an example of a depth distribution. CLICK
Here is another one. Its moments are the same as for the first distribution. CLICK
This is also true for this distribution CLICK
and this one CLICK
and all of these. CLICK
There is not much we can tell about the shape of these functions but we can tell that they have to lie within specific bounds. The lower bound is what we will use as approximation to the shadow intensity. CLICK
This way, we avoid surface acne. At the same time we make the most of our data because knowing only the moments, the ground truth may agree with the lower bound.

From moments to shadows

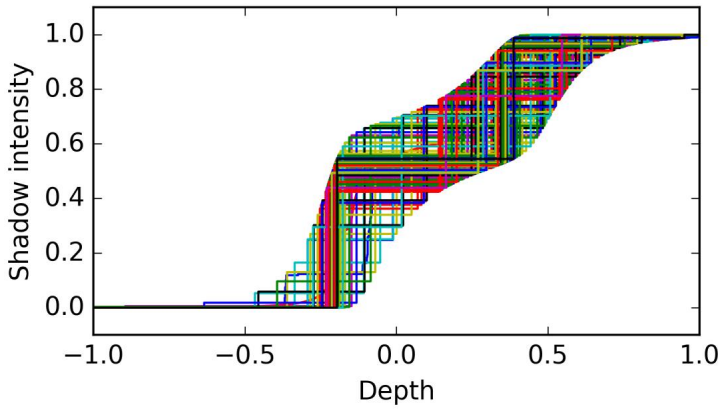
Many distributions share same 4 moments.



We want to reconstruct depth distributions from their moments. But this problem is ill-defined. Here is an example of a depth distribution. CLICK
Here is another one. Its moments are the same as for the first distribution. CLICK
This is also true for this distribution CLICK
and this one CLICK
and all of these. CLICK
There is not much we can tell about the shape of these functions but we can tell that they have to lie within specific bounds. The lower bound is what we will use as approximation to the shadow intensity. CLICK
This way, we avoid surface acne. At the same time we make the most of our data because knowing only the moments, the ground truth may agree with the lower bound.

From moments to shadows

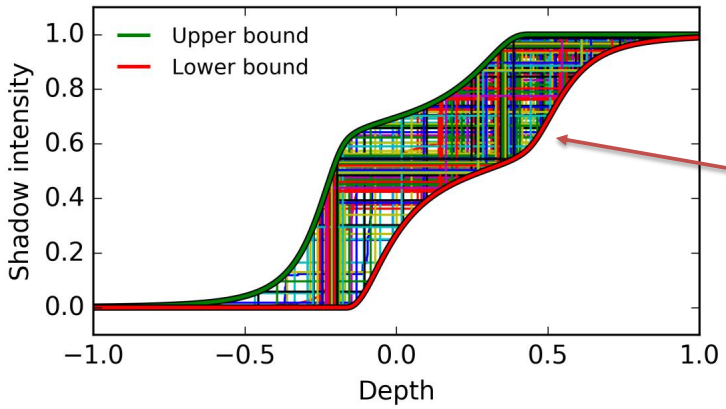
Many distributions share same 4 moments.



We want to reconstruct depth distributions from their moments. But this problem is ill-defined. Here is an example of a depth distribution. CLICK
Here is another one. Its moments are the same as for the first distribution. CLICK
This is also true for this distribution CLICK
and this one CLICK
and all of these. CLICK
There is not much we can tell about the shape of these functions but we can tell that they have to lie within specific bounds. The lower bound is what we will use as approximation to the shadow intensity. CLICK
This way, we avoid surface acne. At the same time we make the most of our data because knowing only the moments, the ground truth may agree with the lower bound.

From moments to shadows

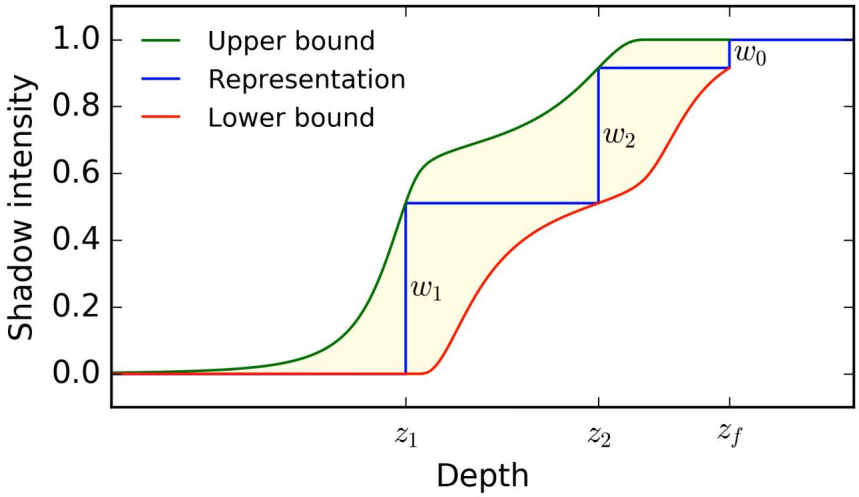
Many distributions share same 4 moments.



Avoids surface acne

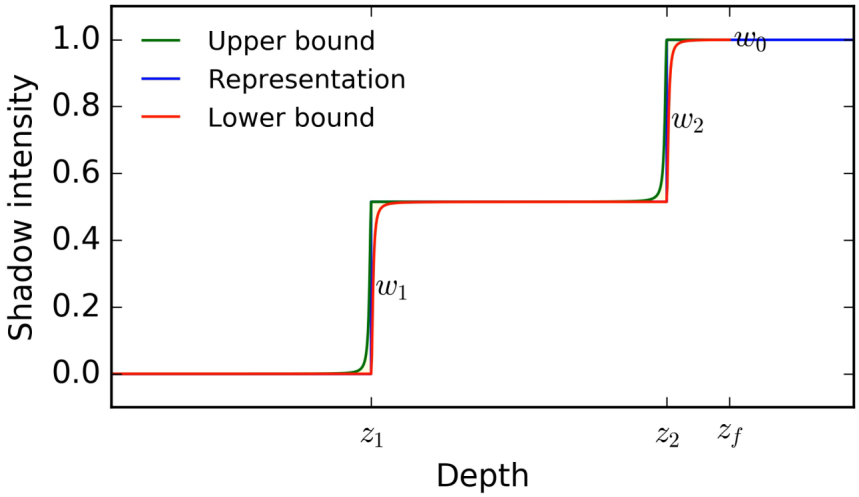
We want to reconstruct depth distributions from their moments. But this problem is ill-defined. Here is an example of a depth distribution. CLICK
Here is another one. Its moments are the same as for the first distribution. CLICK
This is also true for this distribution CLICK
and this one CLICK
and all of these. CLICK
There is not much we can tell about the shape of these functions but we can tell that they have to lie within specific bounds. The lower bound is what we will use as approximation to the shadow intensity. CLICK
This way, we avoid surface acne. At the same time we make the most of our data because knowing only the moments, the ground truth may agree with the lower bound.

Construction of the bounds



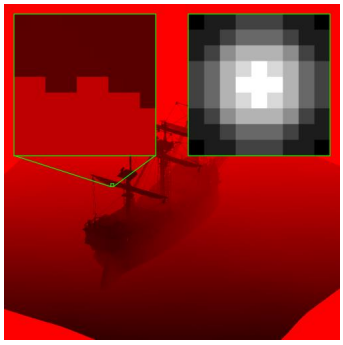
We need an efficient way to compute these bounds. I won't go into the details here but the basic idea is to exploit a useful mathematical theorem. It states that the bounds are always realized by very specific depth distributions. They only use three different depth values to match the four given moments. One of them is the depth where we are minimizing. Thus, we only need to compute the two other depth values and the height of the three steps. That's exactly what the moment shadow mapping algorithm does and I'll be showing you the implementation later.

Bounds are often very narrow

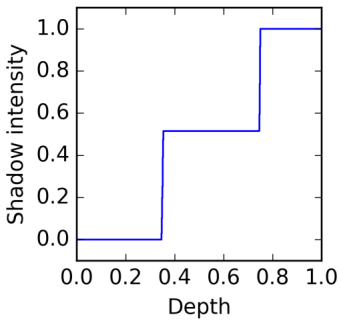


If you find that the bounds in the previous example are not sharp enough, take a look at this example. As you can see, the bounds are extremely narrow, so the reconstruction is very accurate.

Bounds are often very narrow



Shadow map



Depth distribution

- Two surfaces in filter region,
- Perfect recon-struction ☺ ,
- Covers most cases.

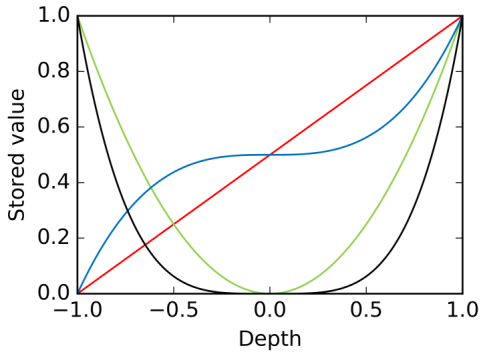
The origin of this example is shown here. As you can see, the filter region in the shadow map covers the silhouette of a shadow caster. Thus, it contains two distinct surfaces which correspond to two steep increases in shadow intensity. CLICK

If we idealize this case with exactly two depth values, the reconstruction is known to be perfect. CLICK

It is rare that a small filter region covers more than two surfaces. Thus, we get a very good reconstruction in most cases.

Quantization

Problem: Just storing z, z^2, z^3, z^4 in 64 bits gives too much rounding error.



Stored basis functions

R — $\frac{1}{2} \cdot z + \frac{1}{2}$

G — z^2

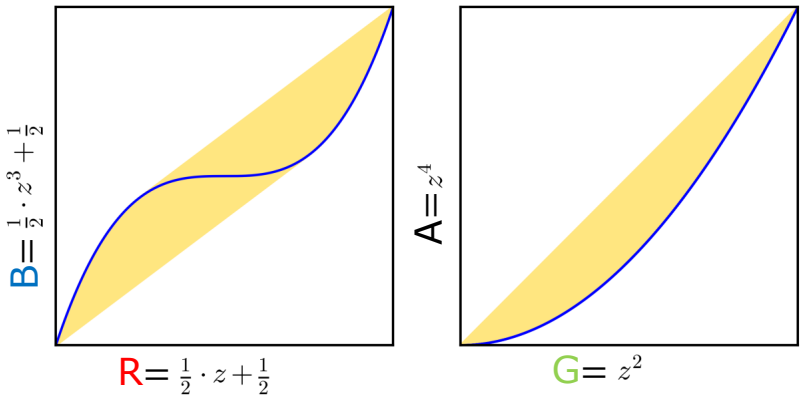
B — $\frac{1}{2} \cdot z^3 + \frac{1}{2}$

A — z^4

There is one problem though. We cannot really afford more than 64 bits per texel for filtered hard shadows. But if we just store powers of the depth, rounding errors are too strong. Basically, the four channels of our shadow map would store the four basis functions shown at the bottom.

Quantization

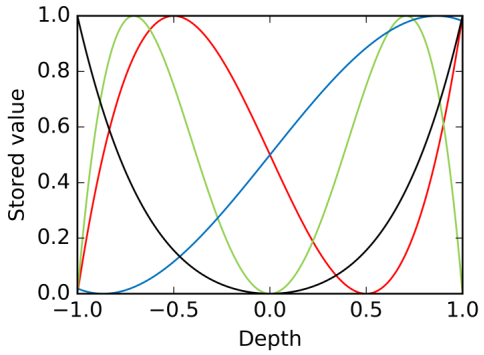
Available range of values is poorly utilized.



To understand why this does not work well, we visualize the values that we store. Each of the four values is plotted on one axis and since we have four channels, we need two plots. The values that we store initially lie on the blue line but after filtering they may lie anywhere in the convex hull, which is shown in yellow. However, we never store any values in the white area. Thus, most values that we could be storing are never actually used. The available memory is not utilized efficiently.

Quantization

Solution: Use an optimized basis of polynomials. Now 64 bits suffice.



Stored basis functions

R — $-\frac{3}{2} \cdot z + 2 \cdot z^3 + \frac{1}{2}$

G — $4 \cdot (z^2 - z^4)$

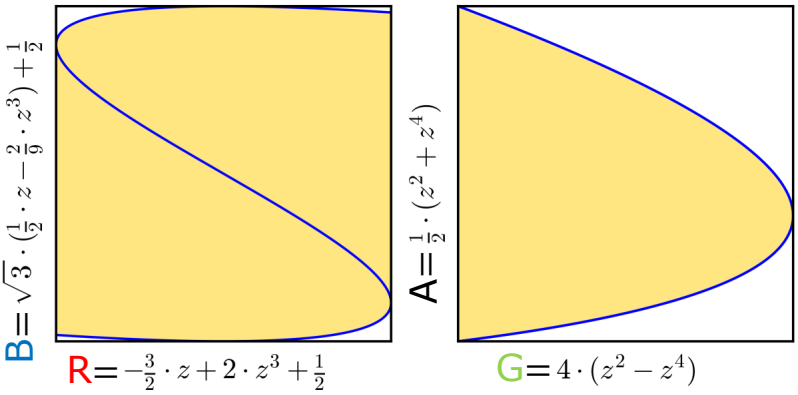
B — $\sqrt{3} \cdot (\frac{1}{2} \cdot z - \frac{2}{9} \cdot z^3) + \frac{1}{2}$

A — $\frac{1}{2} \cdot (z^2 + z^4)$

To overcome this problem, we use a different basis of quartic polynomials. It is optimized numerically to use the memory as efficiently as possible. Now 64 bits suffice.

Quantization

Volume of utilized range is maximized.

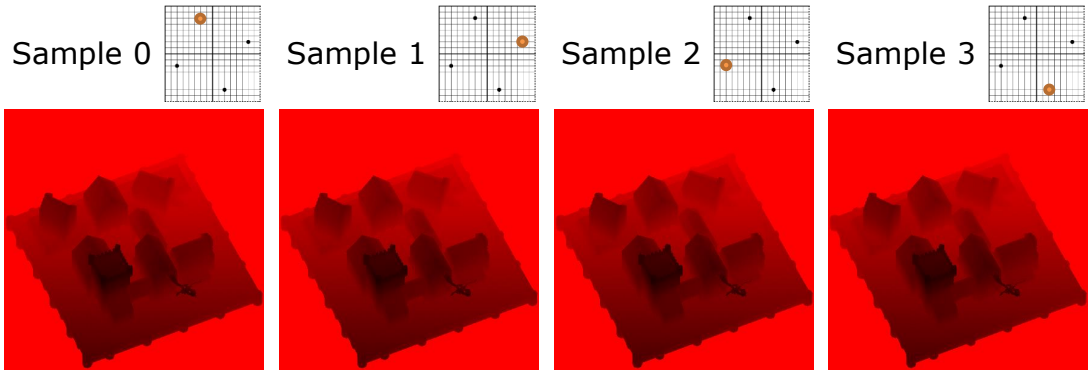


Here you can see the corresponding range of utilized values again. As you can see it has grown immensely. In fact, its area is as large as it could be.

That should be enough theory for one talk, so lets get to the implementation.

Generate a moment shadow map

1. Render to a multisampled depth buffer.



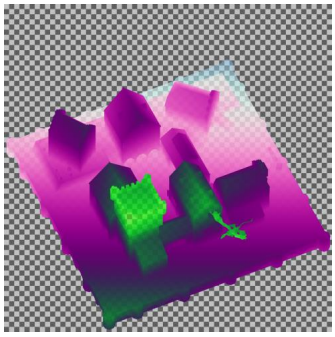
R16 normalized uint, 1024², 4x multisampling

For all techniques we discuss today, the first step is to generate a moment shadow map. In doing so, it is best to utilize hardware-accelerated multi-sample antialiasing. Thus, we render to a multisampled depth buffer (without having a render target bound) to produce a multisampled shadow map. Basically, we get one version of the shadow map for each sample. They are slightly offset at the subpixel level. Hardware- acceleration makes this very fast.

Generate a moment shadow map

2. Compute moments in resolve.

Resolve



Each depth $z \in [-1, 1]$ maps to moments

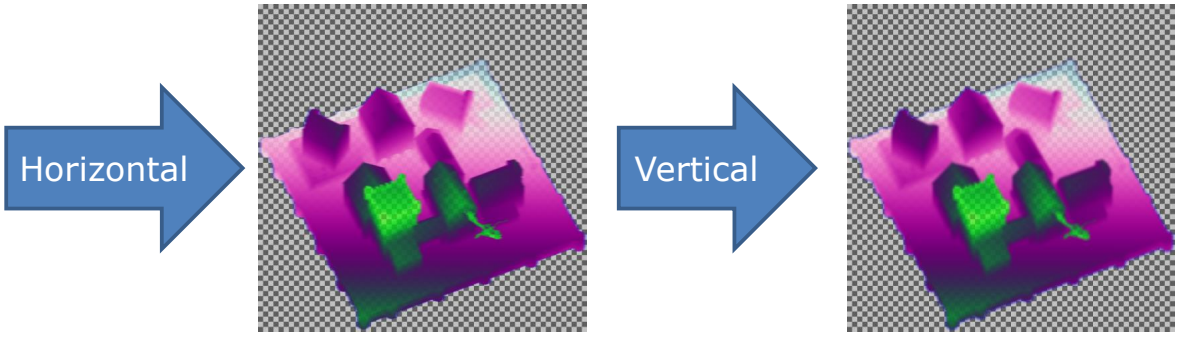
$$\begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix} = \begin{pmatrix} 1.5 & 0 & -2 & 0 \\ 0 & 4 & 0 & -4 \\ \frac{\sqrt{3}}{2} & 0 & -\frac{\sqrt{12}}{9} & 0 \\ 0 & 0.5 & 0 & 0.5 \end{pmatrix} \cdot \begin{pmatrix} z \\ z^2 \\ z^3 \\ z^4 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0 \\ 0.5 \\ 0 \end{pmatrix}$$

R16G16B16A16 normalized uint, 1024^2, no mipmaps

We create the moments as we resolve this multisampled texture into a common texture. For each texel we get all the depth values from the multisampled depth buffer and convert them to moments by computing powers. Then we apply a matrix transform which corresponds to the basis of polynomials I explained before. The mean of these vectors is stored to the four-channel moment shadow map. It should use normalized unsigned integers and at this point we do not need mipmaps.

Generate a moment shadow map

3. Filter (e.g. 2-pass Gaussian + mipmap).



R16G16B16A16 normalized uint, 1024^2, full mipmaps

Now that we have got a filterable shadow map we should apply some filtering to diminish aliasing. Typical choices would be a two-pass Gaussian and mipmapping. Applying the Gaussian is done with a horizontal and a vertical pass. Efficient mipmap generation is usually offered by the API.

Retrieve moments

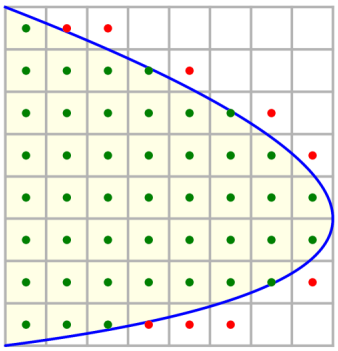
- To shade a fragment,
- Sample the filtered moment shadow map (with mipmapping, anisotropy, etc.),
- Undo the quantization transform:

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} := \begin{pmatrix} -\frac{1}{3} & 0 & \sqrt{3} & 0 \\ 0 & 0.125 & 0 & 1 \\ -0.75 & 0 & 0.75 \cdot \sqrt{3} & 0 \\ 0 & -0.125 & 0 & 1 \end{pmatrix} \cdot \left(\begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix} - \begin{pmatrix} 0.5 \\ 0 \\ 0.5 \\ 0 \end{pmatrix} \right)$$

Now our moment shadow map is ready to use and we can take care of shading a fragment. First we retrieve the moments by taking a filtered sample from the moment shadow map. We can use all the great filtering features that the hardware provides such as mipmapping and anistropic filtering. CLICK Next we undo the quantization transform by simply multiplying with the inverse matrix.

Bias the moments

- Example: Two moments stored in 2·3 bit.
  - Rounding errors invalidate the moments,



In spite of the quantization transform, rounding errors are still a problem. To visualize why, we look at two of our four moments and store them in only three bit. CLICK

The problem is that moments in the convex hull (shaded yellow) may be rounded to points outside this convex hull (marked red). If we use these erroneous moments as input, they do not correspond to any meaningful depth distribution. The algorithm cannot get meaningful results with such meaningless inputs. CLICK

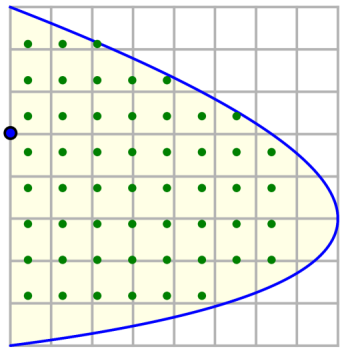
Our solutions is to pull all points back into the convex hull by means of a simple linear interpolation. We scale them towards a fixed point (blue) inside the convex hull. CLICK

Of course, the rounding errors are smaller when we use 16 bits per moment and we only need an interpolation weight of 6*10^-5. This value is scene independent. It only depends on the amount of rounding error that you introduce. CLICK

Our biasing strategy increases light leaking a little but not too much.

Bias the moments

- Example: Two moments stored in 2·3 bit.



- Rounding errors invalidate the moments,
- Lerp them back in: $\alpha = 0.15$

$$b' := (1-\alpha)\cdot b + \alpha\cdot(0{,}0.63{,}0{,}0.63)^{\mathrm{T}}$$

- At 4·16 bit: $\alpha = 6\cdot 10^{-5}$,
- Slightly more light leaking.

In spite of the quantization transform, rounding errors are still a problem. To visualize why, we look at two of our four moments and store them in only three bit. CLICK

The problem is that moments in the convex hull (shaded yellow) may be rounded to points outside this convex hull (marked red). If we use these erroneous moments as input, they do not correspond to any meaningful depth distribution. The algorithm cannot get meaningful results with such meaningless inputs. CLICK

Our solutions is to pull all points back into the convex hull by means of a simple linear interpolation. We scale them towards a fixed point (blue) inside the convex hull. CLICK

Of course, the rounding errors are smaller when we use 16 bits per moment and we only need an interpolation weight of 6*10^-5. This value is scene independent. It only depends on the amount of rounding error that you introduce. CLICK

Our biasing strategy increases light leaking a little but not too much.

Compute the shadow intensity

```
01  float ComputeMSMShadowIntensity(float4 b,float FragmentDepth){
02      float L32D22=mad(-b[0],b[1],b[2]);
03      float D22=mad(-b[0],b[0], b[1]);
04      float SquaredDepthVariance=mad(-b[1],b[1], b[3]);
05      float D33D22=dot(float2(SquaredDepthVariance,-L32D22),
06                       float2(D22,                 L32D22));
07      float InvD22=1.0f/D22;
08      float L32=L32D22*InvD22;
09      float3 z;
10      z[0]=FragmentDepth;
11      float3 c=float3(1.0f,z[0],z[0]*z[0]);
12      c[1]-=b.x;
13      c[2]-=b.y+L32*c[1];
14      c[1]*=InvD22;
15      c[2]*=D22/D33D22;
16      c[1]-=L32*c[2];
17      c[0]-=dot(c.yz,b.xy);
18      float InvC2=1.0f/c[2];
19      float p=c[1]*InvC2;
20      float q=c[0]*InvC2;
21      float r=sqrt((p*p*0.25f)-q);
22      z[1]=-p*0.5f-r;
23      z[2]=-p*0.5f+r;
24      float4 Switch=
25               (z[2]<z[0])?float4(z[1],z[0],1.0f,1.0f):(
26               (z[1]<z[0])?float4(z[0],z[1],0.0f,1.0f):
27               float4(0.0f,0.0f,0.0f,0.0f));
28      float Quotient=(Switch[0]*z[2]-b[0]*(Switch[0]+z[2])+b[1])
29                     /((z[2]-Switch[1])*(z[0]-z[1]));
30      return saturate(Switch[2]+Switch[3]*Quotient);
31  }
```

Input:

- Biased moments b',
- Fragment depth z_f.

Output:

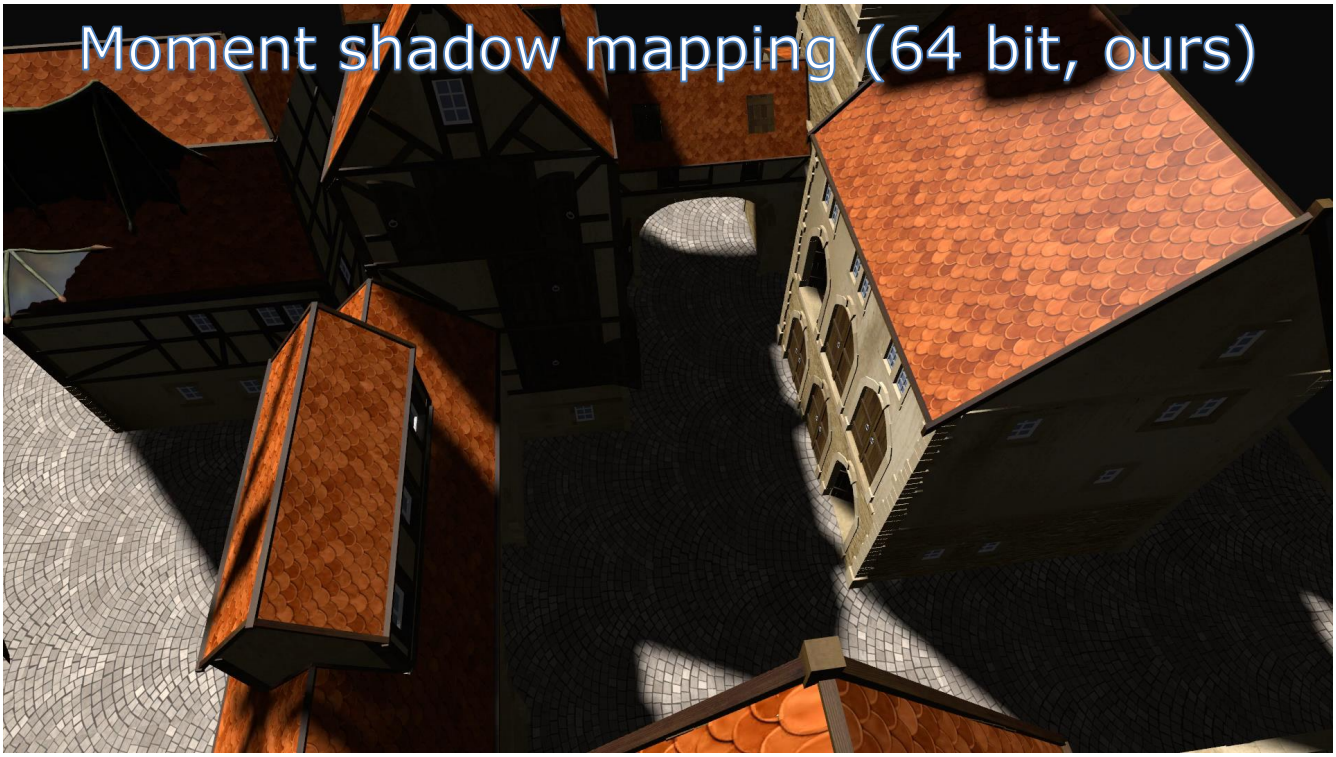
- Shadow intensity,
- Divide by ~98% to diminish remaining light leaking.

Now we are ready to shade fragments. Here you see an HLSL implementation of the core algorithm. Its inner workings are beyond the scope of this lecture but the basic idea is what I explained to you earlier. I recommend that you use this code verbatim. Its inputs are the biased moments and the depth of the fragment that is to be shaded. CLICK

The output is the filtered shadow intensity. CLICK

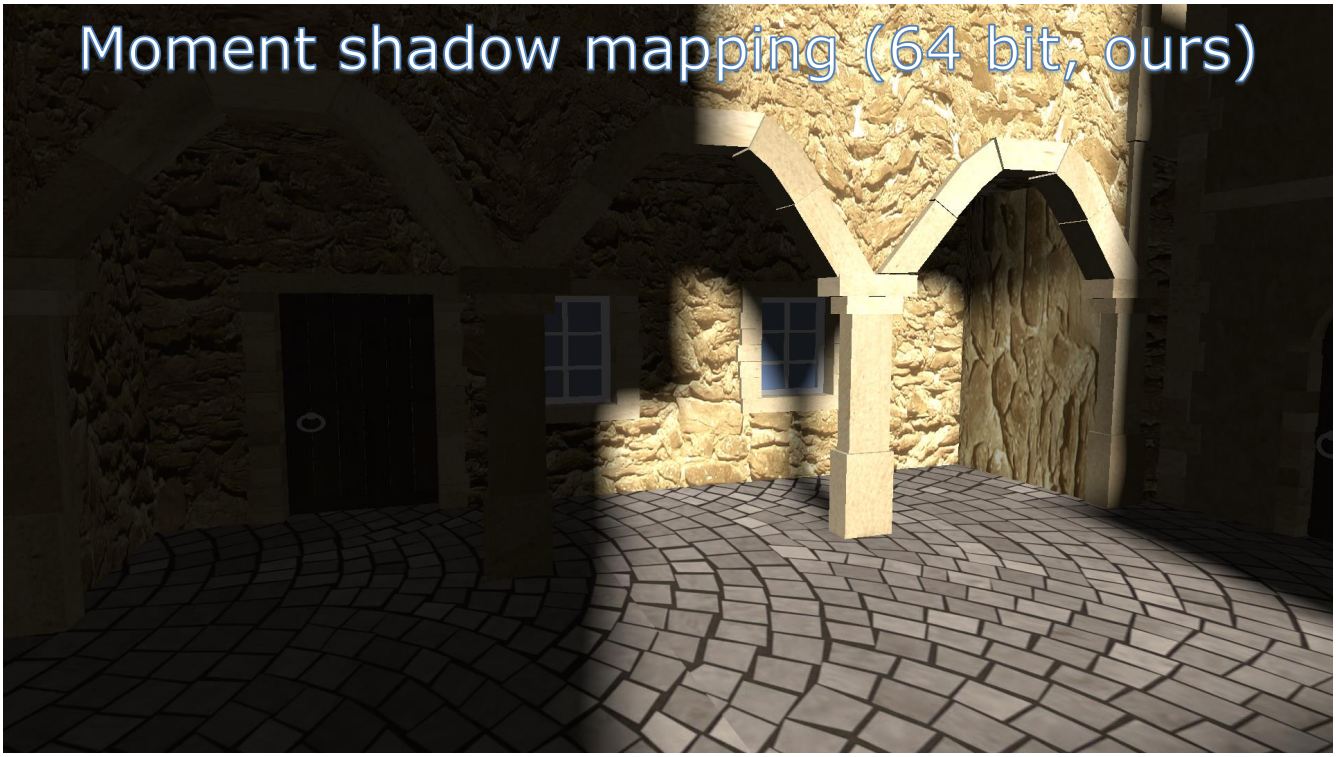
You may want to divide this by something like 98 percent and clamp to cut off weak light leaking. This is particularly important if you are doing an sRGB conversion because this conversion strengthens light leaking.

Now it is time to take a look at the filtered hard shadows that we get with moment shadow mapping.

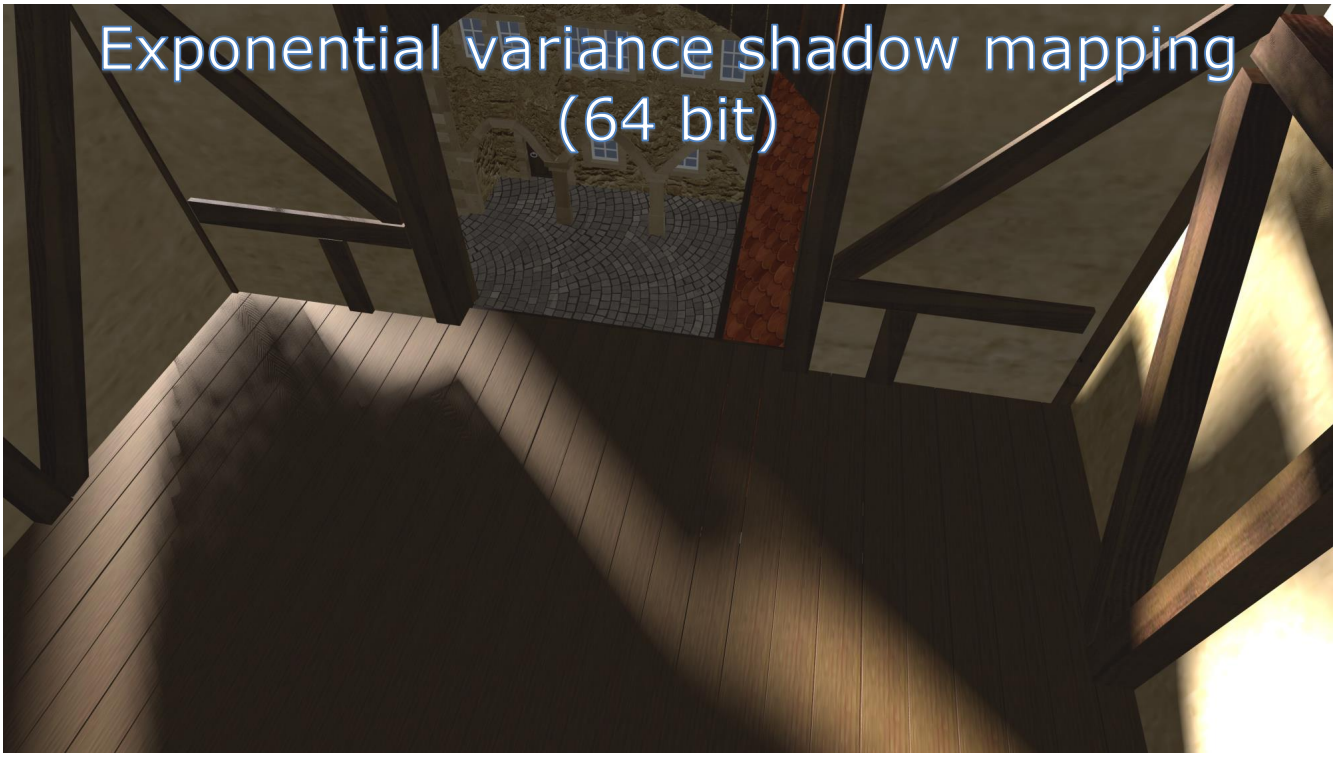


Our technique provides pleasant antialiased shadows throughout this moderately complex test scene. If you look very closely, you can see some light leaking in short- range shadows (e.g. in the shadow of the small roof at the bottom left when the dragon passes over it) but most of the time the result is fine.

Our strongest competitor is exponential variance shadow mapping. Here you see results of this technique using 64 bits per shadow map texel. CLICK
There is some fairly obvious light leaking.



Our technique does not lead to such an artifact using the same amount of memory.

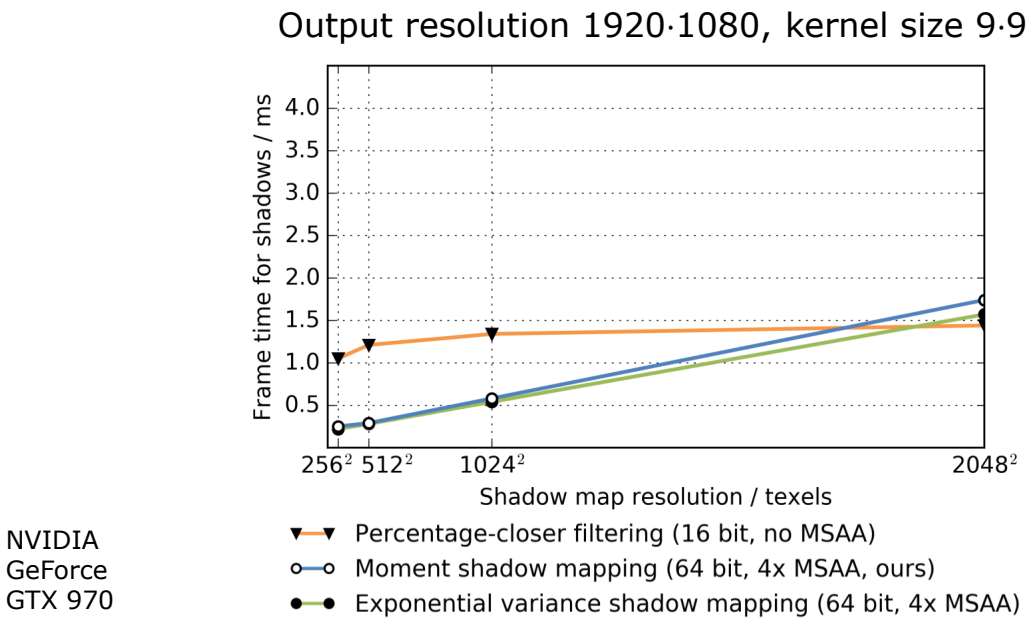


Here‘s an indoors example where exponential variance shadow maps fail quite badly. The wall to the left is lit from the outside. A lot of light leaks into its shadow on the ground. There is also a lot of light leaking for the shadow on the wall to the right. Note that these images are over-exposed to make the artifacts more visible.

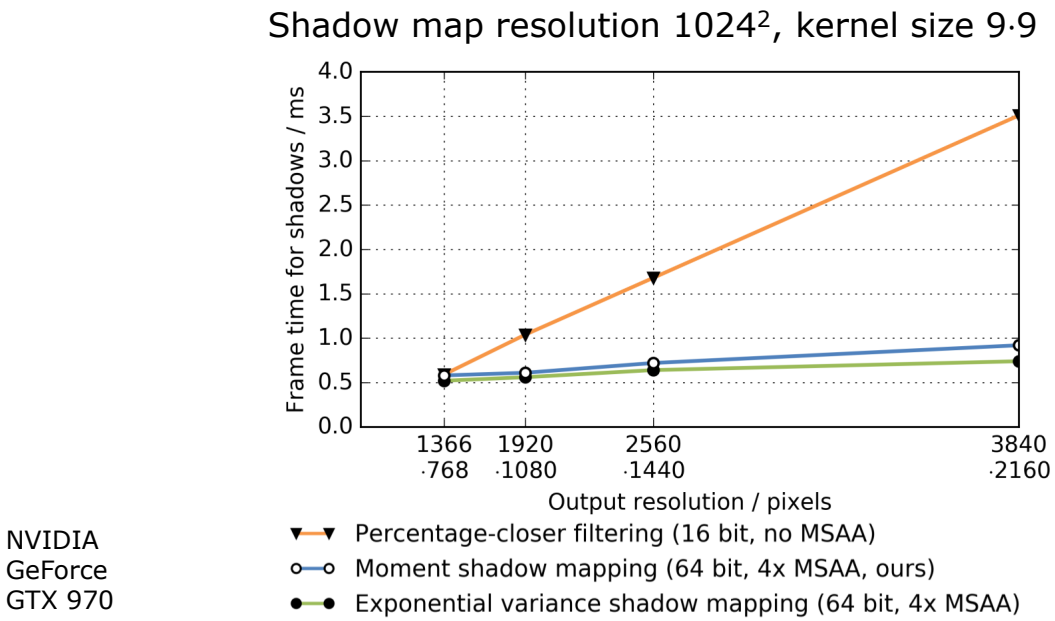
With moment shadow mapping, the light leaking is reduced heavily but does not vanish entirely. In general, moment shadow mapping with 64 bits performs much better than exponential variance shadow mapping with 64 bits but you still should expect some light leaking over short distances.

For reference, here's the result of percentage-closer filtering. Even here there are some artifacts because to avoid surface acne, we need a large depth bias.

Of course quality is only part of what we care about. Next, we look at the run time.

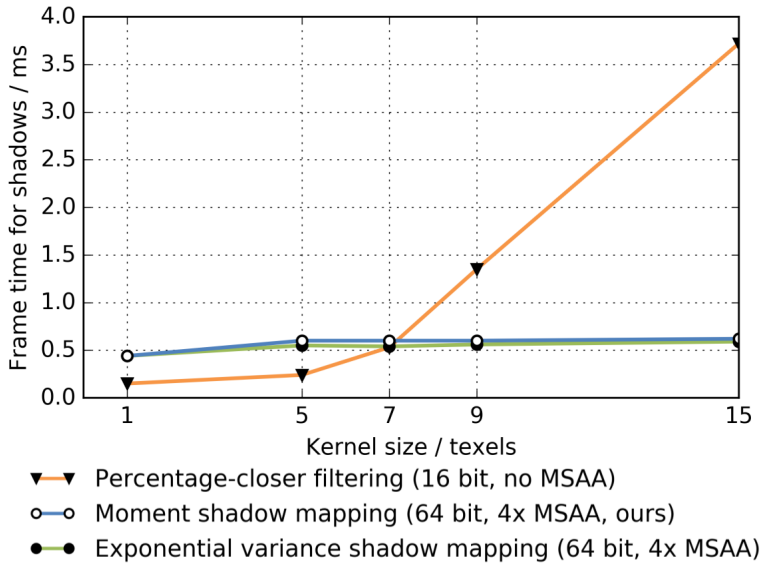


When we discuss the run time, there are three major parameters that affect it. The output resolution (i.e. the resolution on screen), the size of our filter region and the resolution of the shadow map. Here we vary the shadow map resolution while keeping the other two parameters fixed. We then look at the frame time that rendering shadows adds to our total frame time. As you can see, percentage-closer filtering barely gets more expensive as we increase the shadow map resolution. For techniques with filterable shadow maps, the cost per texel is higher because we use more bandwidth, multisample antialiasing and prefiltering. In this example, the filterable shadow maps are slightly slower at 2048^2. However, the effective resolution of the filterable shadow maps is higher due to multisample antialiasing.



Looking at the dependence on the output resolution, we get quite a complementary result. The cost per output pixel is very high for percentage-closer filtering because we need many texture samples. Filterable shadow maps only require one texture sample per pixel and therefore, the cost per pixel is low. Therefore, they scale well to 4k rendering and virtual reality applications where high resolutions are needed. We also note that moment shadow mapping only has a slightly higher cost per pixel than exponential variance shadow mapping in spite of the more complex algorithm. This is because we are bandwidth-limited.

Shadow map resolution 1024^2, output resolution 1920·1080



NVIDIA GeForce GTX 970

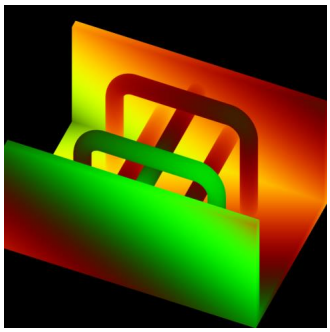
Percentage-closer filtering (16 bit, no MSAA)
Moment shadow mapping (64 bit, 4x MSAA, ours)
Exponential variance shadow mapping (64 bit, 4x MSAA)

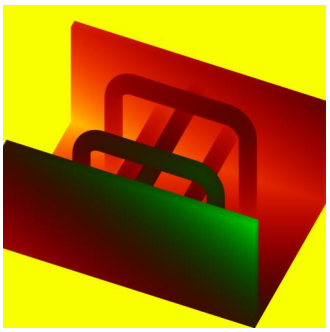
Finally, we look at the dependence on the kernel size. Here, percentage-closer filtering behaves in a rather catastrophic manner. Its run time is quadratic in the kernel radius. For filterable shadow maps, we use the two-pass Gaussian. Its run time barely depends on the kernel size.

Next we will briefly discuss how moment shadow maps can be used for translucent occluders.

Render to a moment shadow map with alpha blending*

Channel 1, 2 Channel 3, 4





This is achieved by simply rendering to a moment shadow map with alpha blending enabled. Here you see a moment shadow map with some opaque occluders. To properly show all four channels, we split it up into two images. Rendering shadows for translucent occluders is done by simply rendering to this moment shadow map with alpha blending. CLICK

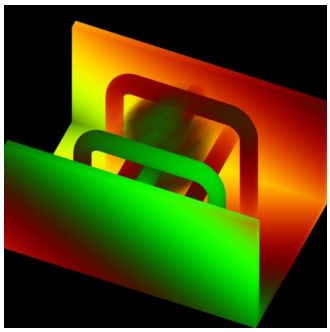
For example, here we have rendered a smoke plume modeled by several alpha- blended planes. CLICK

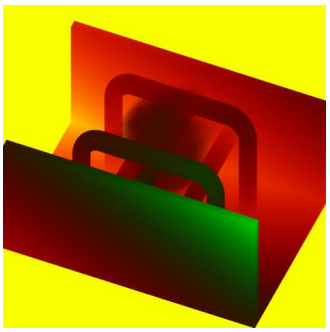
The blend mode that we use is just standard alpha-blending, i.e. linear interpolation. CLICK

You have to trust me, that it makes sense to do so. If you don‘t, you can find a rigorous derivation in the paper.

Render to a moment shadow map with alpha blending*

Channel 1, 2 Channel 3, 4





Blend mode:
α ·source +
$(1-\alpha)$ ·destination

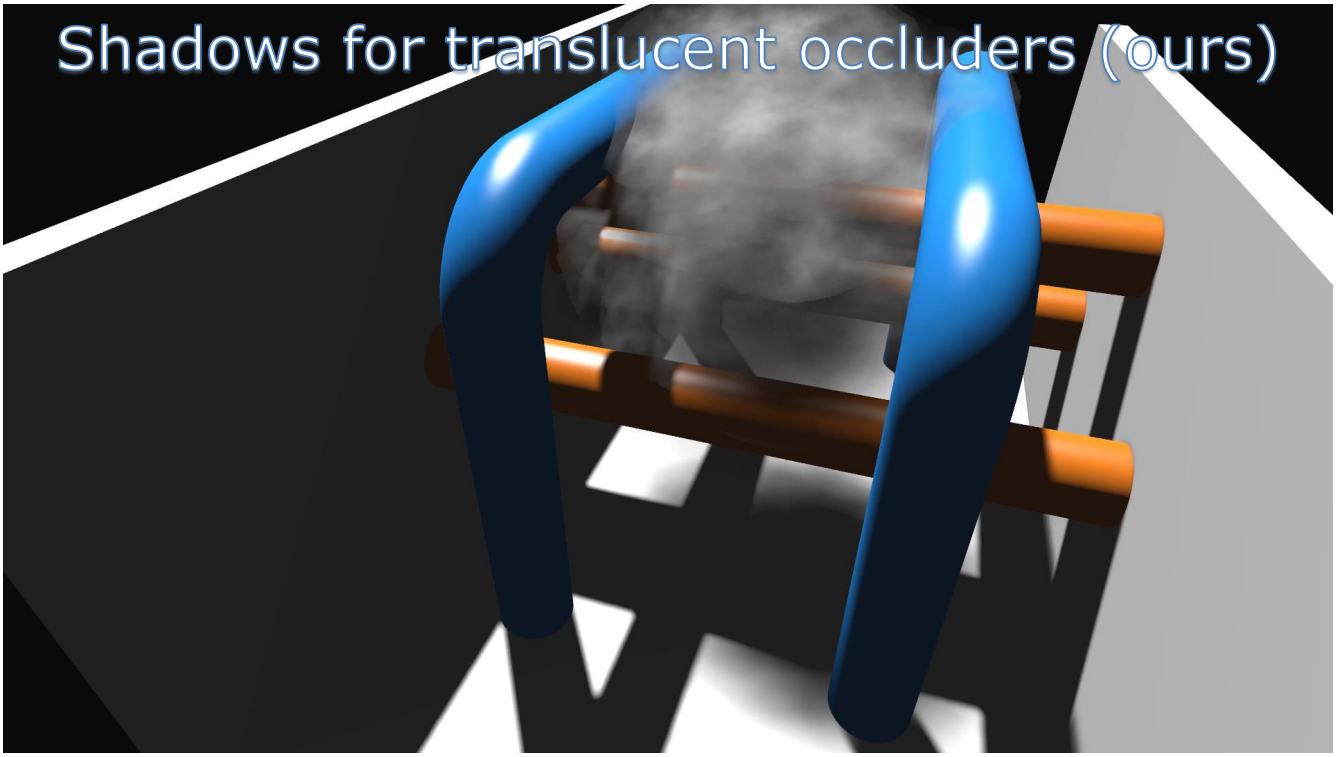
* Trust me, it makes sense if you think about it [Peters16].

This is achieved by simply rendering to a moment shadow map with alpha blending enabled. Here you see a moment shadow map with some opaque occluders. To properly show all four channels, we split it up into two images. Rendering shadows for translucent occluders is done by simply rendering to this moment shadow map with alpha blending. CLICK

For example, here we have rendered a smoke plume modeled by several alpha- blended planes. CLICK

The blend mode that we use is just standard alpha-blending, i.e. linear interpolation. CLICK

You have to trust me, that it makes sense to do so. If you don‘t, you can find a rigorous derivation in the paper.

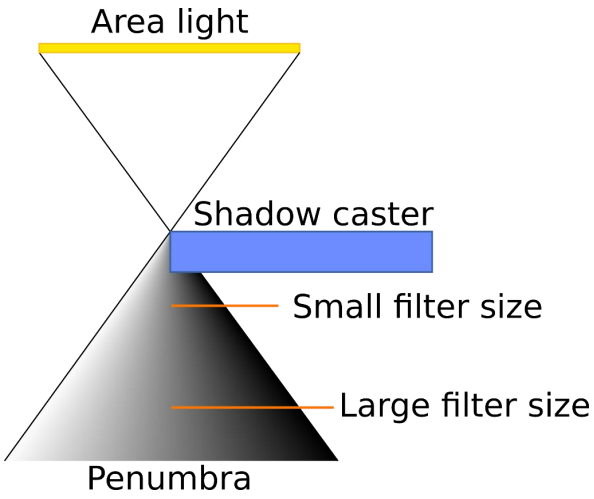


Here you can see the result of this approach. As you can see, we get plausible shadows throughout the entire scene. The smoke casts partial shadow onto the walls and the ground, there is proper self-shadowing within the smoke and the pipes receive a plausible shadow as well. All of this is done with a single 64-bit moment shadow map. You may get some additional light leaking but the technique is made compelling by its simplicity and efficiency.

Lets move on to soft shadows.

Percentage-closer soft shadows [Fernando05]

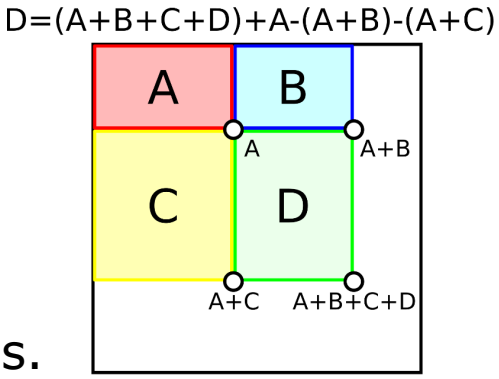
1. Blocker search
   - Compute average blocker depth
2. Penumbra estimation
   - Compute kernel size
3. Percentage-closer filtering



Our technique is based on percentage-closer soft shadows, which is a three step procedure. First the shadow map is sampled to find the average depth of shadow casters for a pixel. CLICK
Based on this depth, we estimate how large the penumbra should be at this depth. Close to the caster it is small, but it grows linearly. CLICK
Finally, we apply percentage-closer filtering with a corresponding kernel size. Of course, this procedure requires many samples and thus it is quite expensive.

Summed-area tables (SAT)

- Each texel stores sum to the left top,
- Get sum over any rectangle D from 4 samples,
- Constant time ☺ ,
- Created by horizontal/ vertical compute passes.



We use a summed-area table to accelerate it. In a summed-area table each texel stores the sum of texels which are to the left top in the source texture. For example the texel marked with A stores the sum over the red rectangle. CLICK

To compute the sum over an arbitrary rectangle D, we take samples at the four corners. Then one addition and two subtractions reveal the desired sum at a constant cost. CLICK

This takes constant time, independent of the size of the rectangle. CLICK

Construction of the summed-area table can be done using two compute shaders. First we compute horizontal prefix sums and then vertical prefix sums.

SAT over a moment shadow map

- Generate a SAT from a moment shadow map,
- Store 4 moments in 4 32-bit uints,
  - Multiply by $\frac{2^{32}}{n}$, then round to integer,
  - n is number of texels in largest kernel,
  - May overflow after n adds but end result is fine,
  - Precision of $\log_2 \frac{2^{32}}{n}$ bits: 27^2 kernel $\rightarrow$ 22.5 bits.

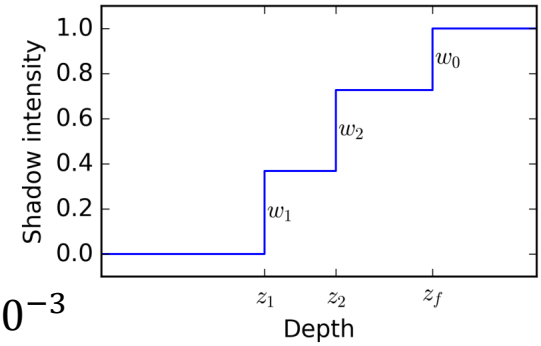
We create a summed-area table over a moment shadow map. The details are a bit intricate because we need to ensure a sufficient precision. Therefore, we store each of the four moments in a 32-bit unsigned integer. The floating point moments between zero and one are multiplied by 2^32/n. Then we round to integer. After n additions, we may get integer overflows. However, n is the number of texels in the largest kernel that we use. Thus, the end result is fine if we just let the overflows happen. The precision of the moments is given by the base-2 logarithm of this factor. For example, a 27^2 kernel yields a precision of 22.5 bits. This is almost as good as single-precision floats.

Moment-based blocker search

- **Input:** Moments for search region, depth z_f.
- **Output:** Average blocker depth.

1. Reconstruct with three depth values,
2. Compute average blocker depth, e.g.

$$\frac{w_1 \cdot z_1 + w_2 \cdot z_2 + \varepsilon \cdot z_f}{w_1 + w_2 + \varepsilon} \qquad \varepsilon := 10^{-3}$$



Now we want to shade a pixel. Using the summed-area table, we can easily get the four moments for the search region. We also know the depth of the pixel. In the blocker search, we want to compute the average depth of shadow casters. CLICK

As a first step, we reconstruct a matching depth distribution using three depth values. Hopefully this looks familiar to you. It is what I showed you when I explained moment shadow mapping. Therefore, the algorithm for the reconstruction is already available. CLICK

Now we simply assume that this reconstruction is correct because having more than three surfaces in the search region is unlikely. Under this assumption, the average blocker depth can be computed directly. CLICK

To make this robust to cases where the denominator is small, we add a little bias towards the depth of the pixel.

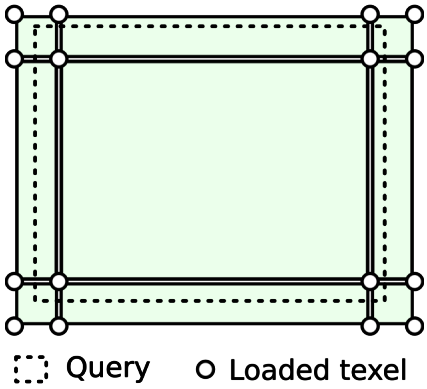
Penumbra estimation and filtering

- Penumbra size is estimated as in percentage-closer soft shadows,
- A single interpolated query to the SAT provides moments for the filter region,
  - Rectangular filter → rectangular light source,
  - A little unrealistic, but the shadows look fine.
- Moment shadow mapping yields shadow.

With the average blocker depth we can estimate the size of our penumbra as in percentage-closer soft shadows. CLICK
A single interpolated query to the summed-area table gives us the moments for the corresponding filter region. Note that this filter region has to be rectangular, i.e. we assume a rectangular light source. This might not be what you really want in many cases but the soft shadows still look plausible. CLICK
Then we apply standard moment shadow mapping and we get our shadow intensity. The moment bias can be smaller here.

Caveat: Bilinear interpolation

- Have to interpolate uints in the shader.
1. Load 4·4 texels,
2. Compute sums for 3·3 rectangles,
3. Convert to float,
4. Combine.
- Costly but acceptable.



There is one pitfall though. Your API probably refuses to perform interpolation on the integers in the summed-area table and even if it would do it, it would probably do it wrongly. We have to interpolate the integers in the shader. CLICK

Suppose we are given an arbitrary query rectangle. CLICK

First we load the four texels adjacent to the four corners of this query. CLICK

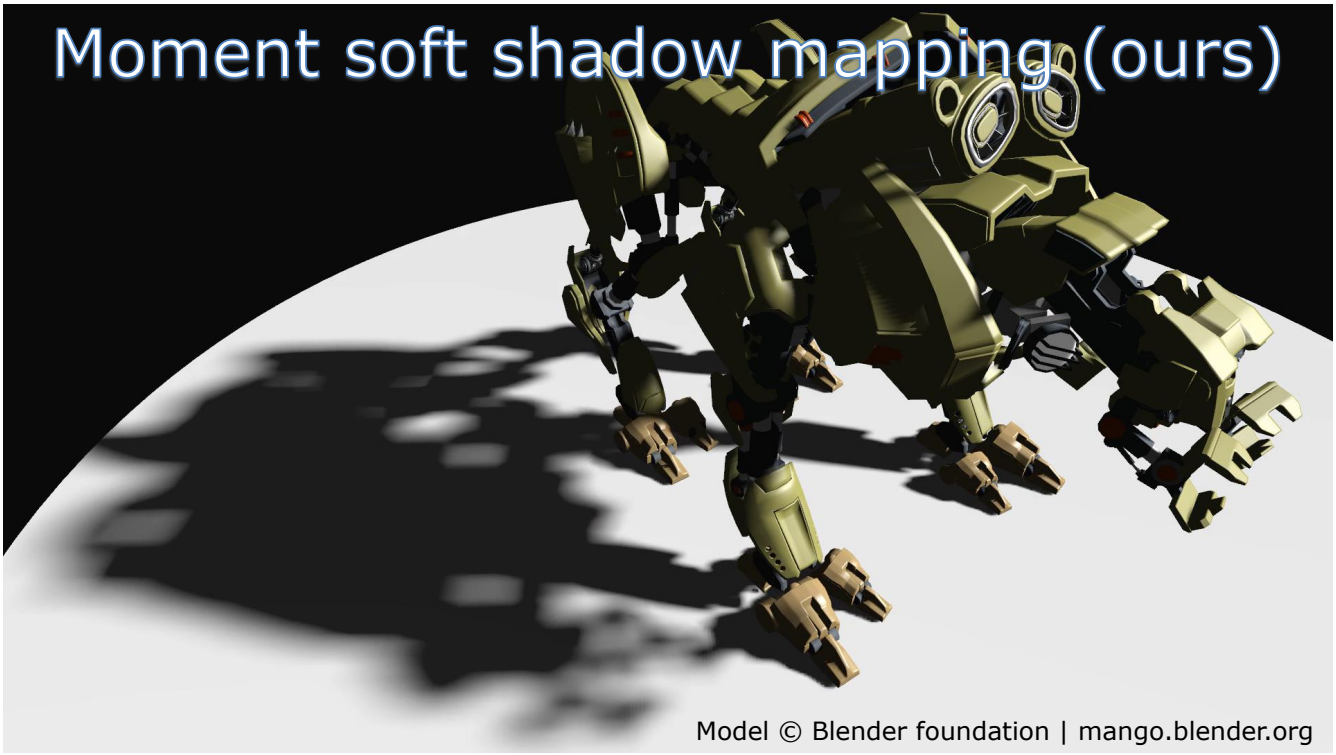
From these values we compute sums for the 3*3 rectangles shown here. CLICK

All of these rectangles are smaller than the maximal kernel size, so we can safely convert to float. CLICK

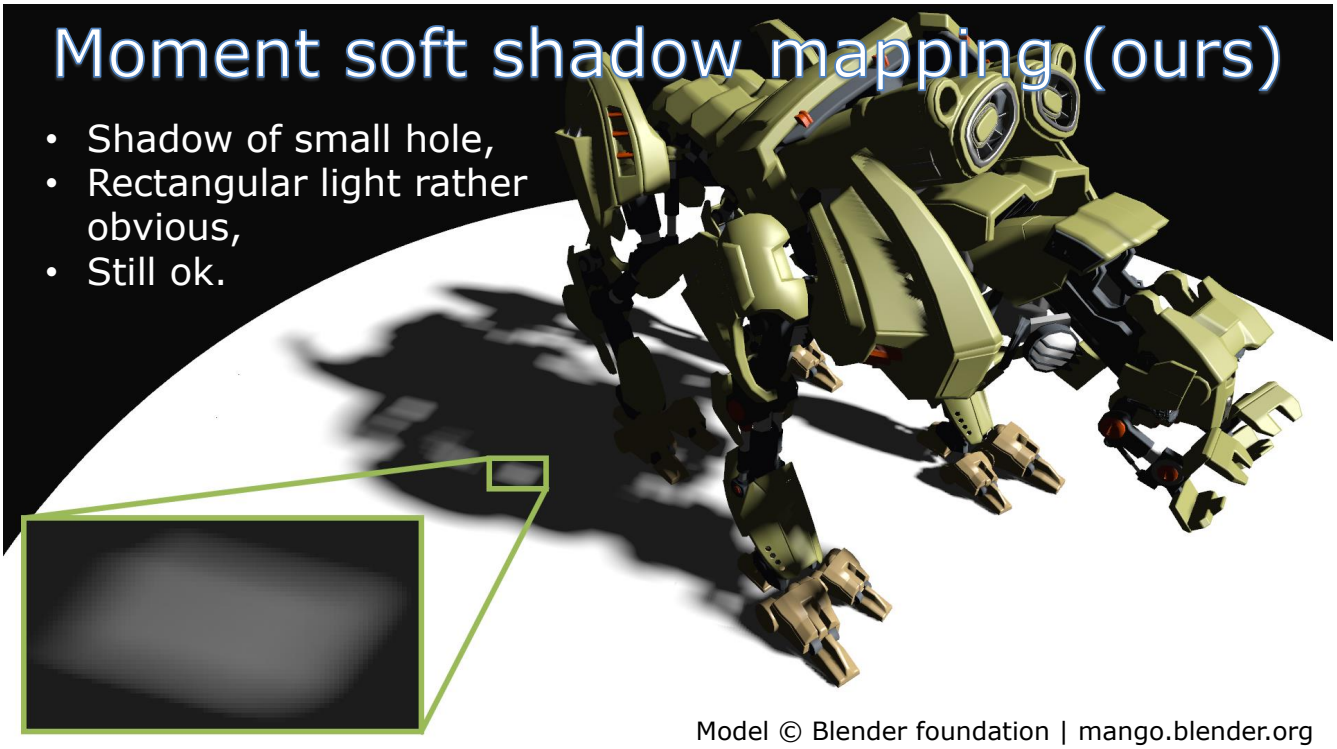
Finally, we combine the results with weights proportional to the area covered by the query. CLICK

This is quite costly but still better than brute-force sampling.

Lets take a look at the results.



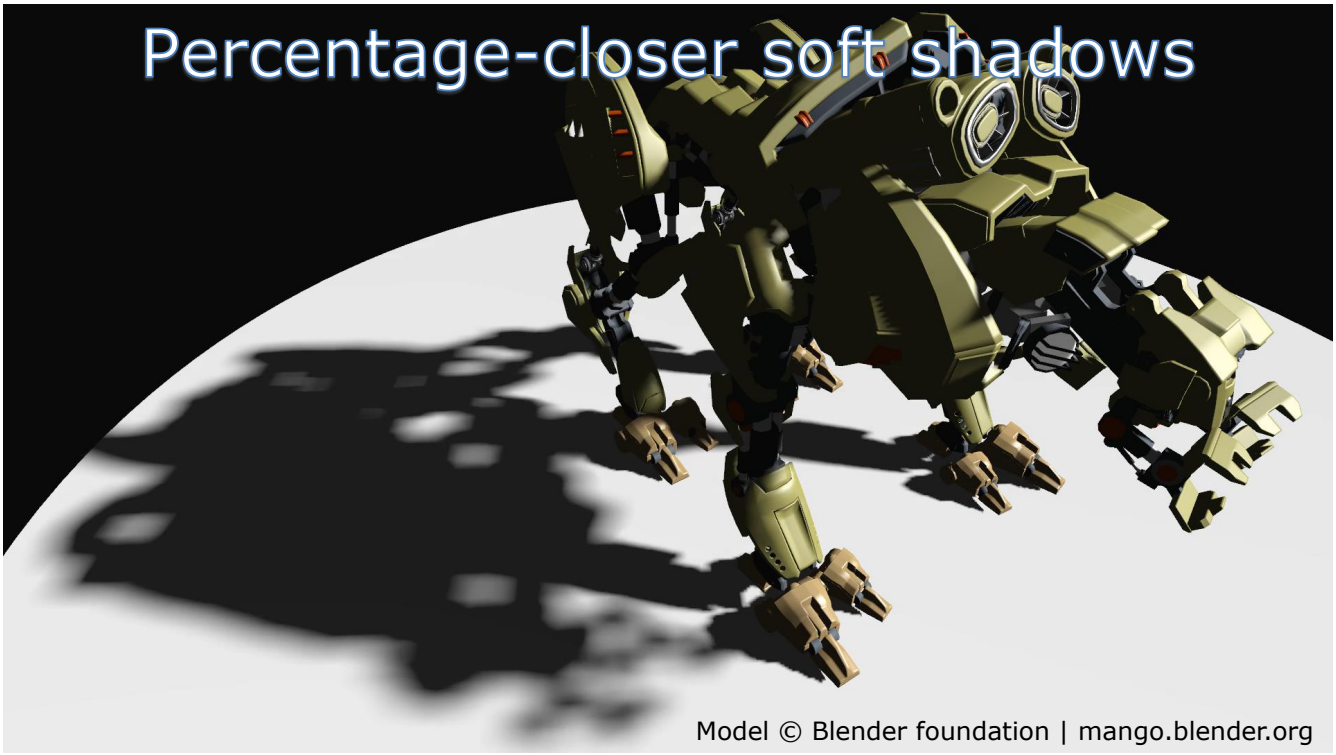
Here you see soft shadows computed by moment soft shadow mapping for a complex model. They are hard at contact points and then gradually soften as expected. The hard contact shadows exhibit some aliasing but it is not too bad thanks to 8x multisample antialiasing.



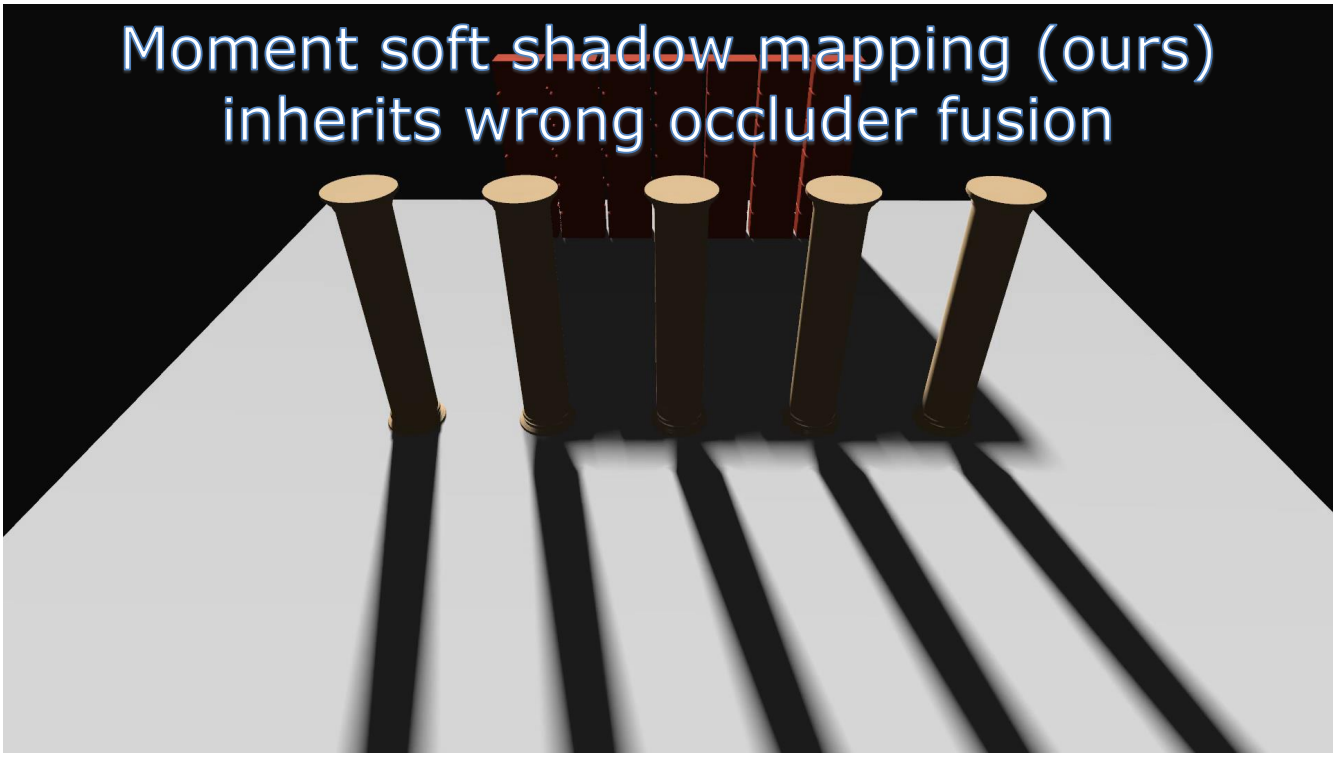
Here you see a still from this video. The magnified inset at the bottom left shows the shadow of a small hole. CLICK

You basically see the projection of the light source and the fact that it is rectangular is quite obvious. CLICK

However, this is not very distracting and in other situations it is nearly impossible to tell that the light is rectangular.

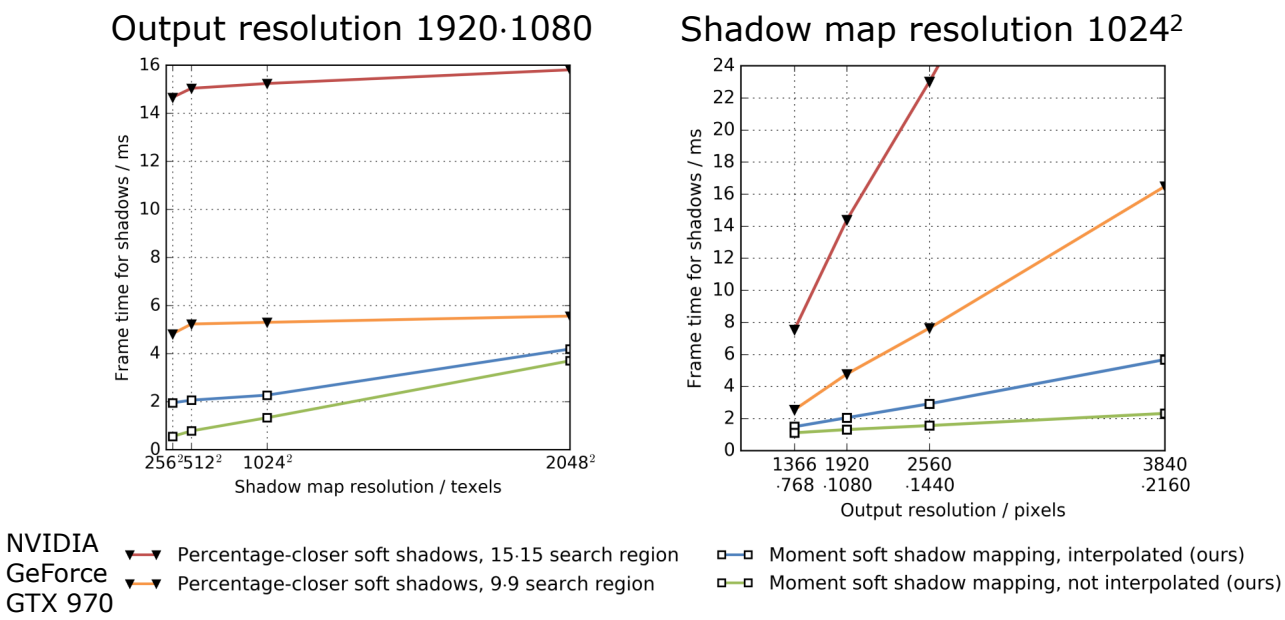


Here you see the result of percentage-closer soft shadows. It is quite similar but there are two noteworthy issues. The shadow does not get as soft as it should because we have limited the search region to 15*15 which is not quite enough but already slow. Besides aliasing for contact shadows is much worse because we cannot use multisample antialiasing for the common shadow map.



An artifact that moment soft shadow mapping inherits from percentage-closer soft shadows is demonstrated here. CLICK

The shadow of the distant wall makes the shadow of the nearby column softer. This is clearly an artifact but usually not too distracting. Every approach using the blocker search of percentage-closer soft shadows exhibits such artifacts. CLICK

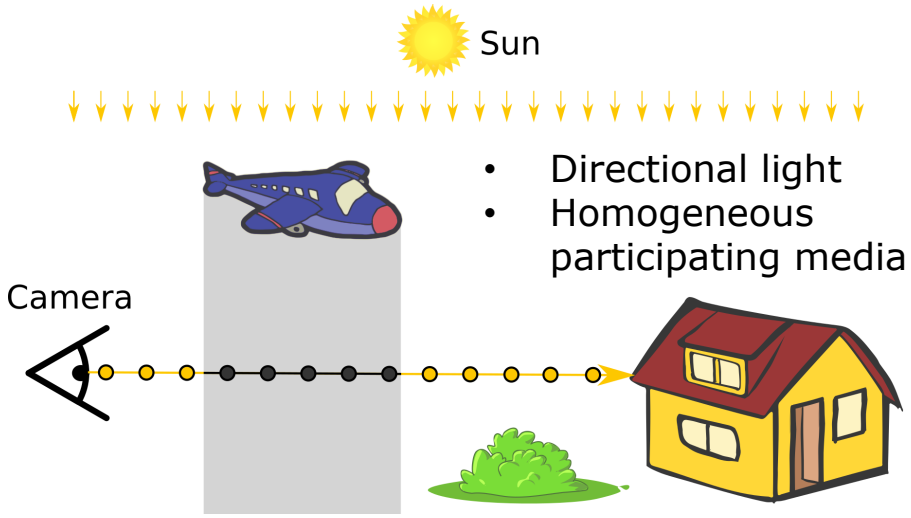


We take a look at the run time for shadows in a similar manner as before. To the left we vary the shadow map resolution, to the right we vary the output resolution. Even with a 9*9 search region, which is too small for realistic shadows, percentage-closer soft shadows is always slower than moment soft shadow mapping. Again we observe that approaches using moment shadow maps have a high cost per texel but a low cost per pixel. The cost per pixel is increased heavily when we enable interpolation. Optimizations for specific use cases or specific hardware may help to close this gap.

To conclude the talk lets move from surface shadows to volumetric single scattering.

We want single scattering because it is beautiful and helps to direct the attention of the viewer.

Integrate shadow along view ray



Here is a simple scene lit by a directional light. CLICK

If there is homoegenous participating media, the shadow volumes of objects become visible as they reduce single scattering. CLICK

To render this effect, we need to integrate the shadow term over each view ray, e.g. by using many shadow map samples along the view ray. Note that our technique is limited to the scenario described here, i.e. we do not support point lights or inhomogenous media but get a high performance for this special case.

Prefiltered single scattering [Klehm14]

- Ray marching = Percentage-closer filtering with view ray as kernel,
- Klehm et al. use a convolution shadow map,
- The convolution shadow map is filtered,
- Then each texel encodes the single scattering result for an entire light ray,
- 256 bits per texel ☹ .

Our technique is an extension of prefiltered single scattering by Oliver Klehm. The key observation in this technique is that ray marching is like percentage-closer filtering using the view ray as filter region. CLICK

Oliver Klehm uses a convolution shadow map to precompute this result. CLICK

Each texel encodes the single scattering result for an entire light ray. Thus, we no longer need to integrate per pixel. CLICK

On the other hand, we need 256 bits per texel for a reasonable quality. That‘s quite expensive.

Prefiltered single scattering with 6 moments (ours)

- Moment shadow maps yield better results with 64 bits per texel,
- Prefiltered moment shadow map:
  - Encodes single scattering result per light ray,
  - Using six moments.
- Interpolate between lower and upper bound to minimize artifacts.

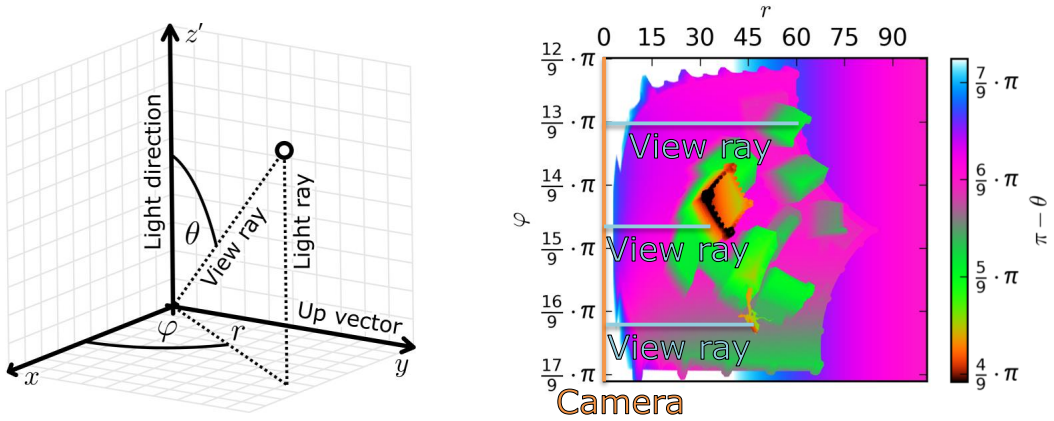
In our work we use a moment shadow map instead. Using four times less memory we get better results. CLICK

Each texel in the prefiltered moment shadow map encodes the single scattering result for one light ray. We no longer use four moments but six moments to a get a more faithful reconstruction. CLICK

With the same approach as before, we can compute lower and upper bounds for the single scattering. We then interpolate between the two in a manner that keeps objectionable artifacts to a minimum.

Resampling using epipolar geometry

- z=π-θ, u=r, v=φ $\longrightarrow$ View rays are rows.



To make this work, we transfer the moment shadow map into a new coordinate system based on epipolar geometry. This transform is nonlinear, so we have to apply resampling. For the coordinate conversion, we first put the camera into the origin and align the light direction with the z-axis. Then we compute spherical coordinates. The inclination is used as new shadow map depth. The azimuth is used as vertical shadow map coordinate. CLICK

In this coordinate system the whole left column of the shadow map corresponds to the camera. CLICK

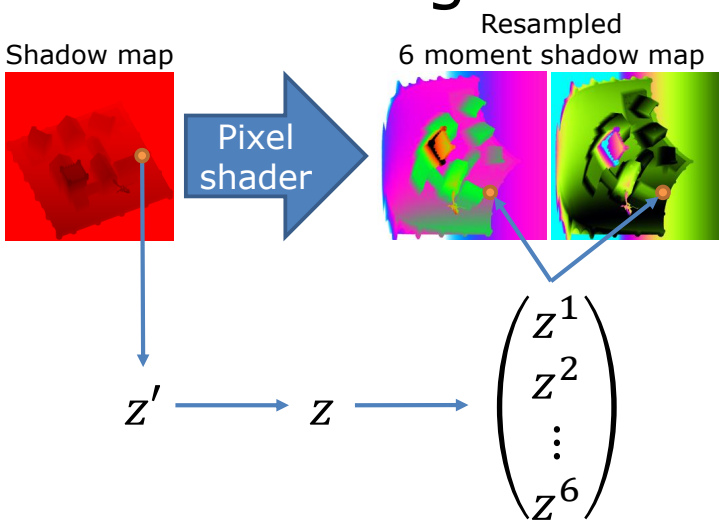
Each row corresponds to a view ray. CLICK

CLICK

Resampling without filtering

1. Convert coordinates,
2. Sample shadow map,
3. Convert depth,
4. Compute 6 moments.

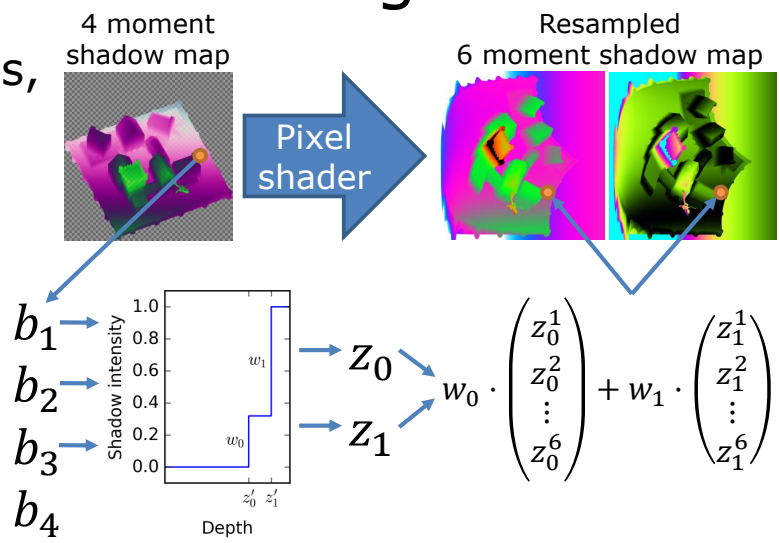
- Aliasing ☹ .



Here is one way to construct this resampled shadow map. We use a pixel shader. Suppose we want to compute the moments for the texel highlighted in the resampled 6 moment shadow map. CLICK
As a first step we compute the corresponding coordinate in the original shadow map coordinate system. CLICK
Then we take a sample from a common shadow map. CLICK
The sampled depth has to be converted into the new coordinate system. CLICK
Finally, we compute the six moments and write them to the output texture. CLICK
The problem with this approach is that we must not filter the common shadow map. Thus, we get bad aliasing.

Resampling with filtering

1. Convert coordinates,
2. Sample 4 moment shadow map,
3. Reconstruct 3 moments with 2 depths z'_0, z'_1,
4. Convert 2 depths,
5. Compute 6 moments.



Here's a more sophisticated approach. The first step is the same. We just convert the shadow map coordinates. CLICK

However, we then sample a moment shadow map with filtering. CLICK

We dispose the fourth moment and represent the other three moments using two depth values. The corresponding algorithm is closely related to what we've seen before. CLICK

Then we convert both depth values used in the reconstruction to the new coordinate system. CLICK

Finally, we construct two vectors of six moments and combine them into one. This gives us the final output.

Quantization

- 6 moments: 2×R10G10B10A2 normalized uint, i.e. 64 bits total (4 bit waste),
- Quantization transform:

$$\begin{pmatrix} R_0 \\ G_0 \\ B_0 \end{pmatrix} = \begin{pmatrix} 2.5 & -1.87499864 & 1.26583039 \\ -10 & 4.20757543 & -1.47644883 \\ 8 & -1.83257679 & 0.71061660 \end{pmatrix}^T \cdot \begin{pmatrix} z^1 \\ z^3 \\ z^5 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

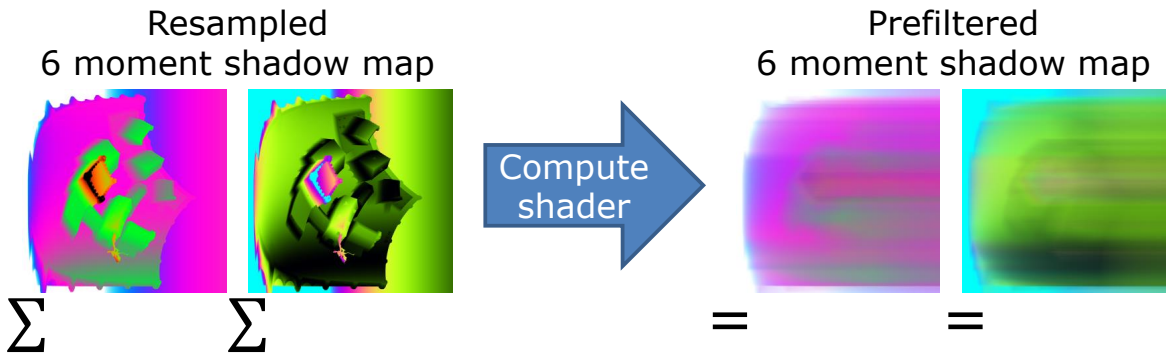
$$\begin{pmatrix} R_1 \\ G_1 \\ B_1 \end{pmatrix} = \begin{pmatrix} 4 & 9 & -0.57759806 \\ -4 & -24 & 4.61936648 \\ 0 & 16 & -3.07953907 \end{pmatrix}^T \cdot \begin{pmatrix} z^2 \\ z^4 \\ z^6 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0.018888946 \end{pmatrix}$$

As before, we need to ensure a sufficient precision of the moments. It turns out that rounding errors are less malicious for single scattering. Thus, we choose to use only 10 bits per moment. All six moments are stored in two textures where red, green and blue each use 10 bits. The four bits for alpha are unused. CLICK

We also need an optimized quantization transform for six moments which you can see here.

Prefix sums along rows

Precompute all single scattering results.



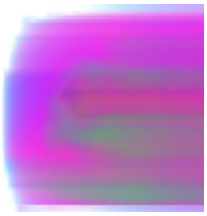
Next we precompute the single scattering integrals for each view ray by simply summing over rows. CLICK

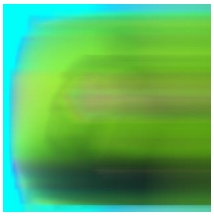
This is done by a compute shader. The sums should be normalized to avoid overflows.

Deferred pass for single scattering

Depth buffer

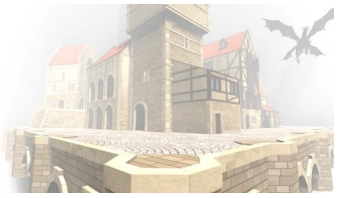
Prefiltered 6 moment shadow map





Additive deferred pass

Scene with single scattering



2 shadow map samples per pixel on screen.

Finally, we perform an additive deferrred pass. The depth buffer of the scene and the prefiltered six moment shadow map are used as input. For each pixel, we only need one lookup in each of the two textures for the six moment shadow map. Thus, the deferred pass is very efficient.

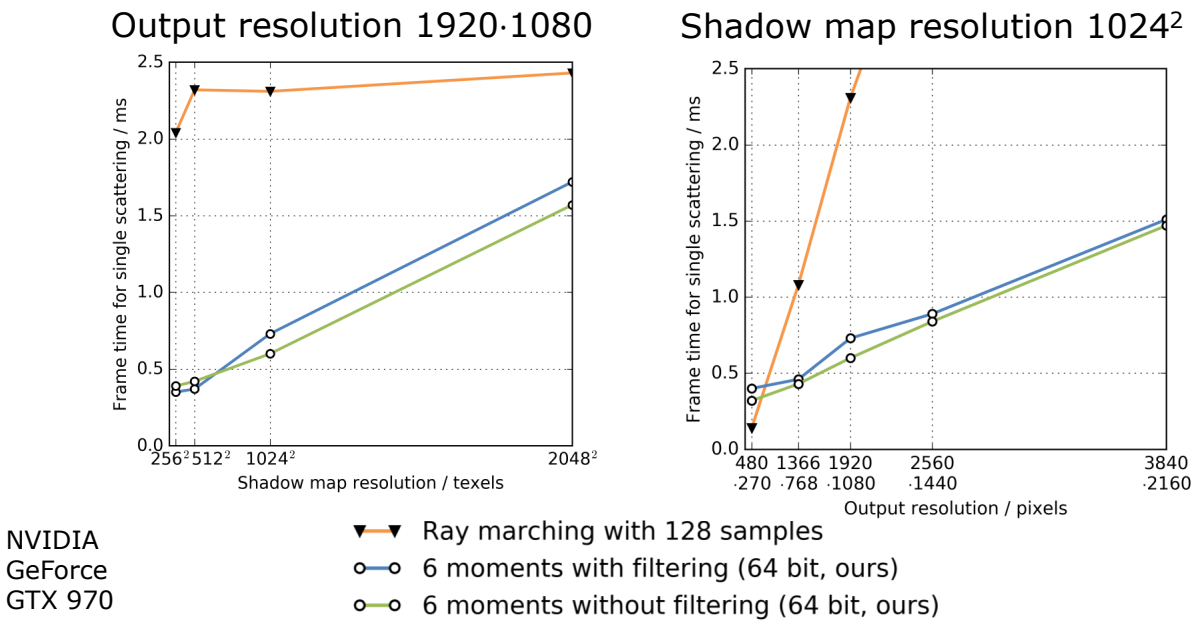
We will now take a look at the results.

Ray marching with 128 samples provides us with a ground truth. This result is what we are going for but computing it like this is very expensive.

Here you see the result that we get using prefiltered single scattering with six moments. As you can see, the appearance is very close to the ground truth. Throughout this animation there is just one obvious artifact, namely some leaking shadows at the roof to the left. If you do not look out for it, you probably won‘t notice it.

Here's a bad case of aliasing when we do not use filtering during resampling of the shadow map. The single scattering is extremely unstable.

If we instead filter a moment shadow map, this aliasing goes away almost entirely. The result is much smoother and more stable.



Looking at the run time, we make similar observations as before. The cost per texel is fairly high for prefiltered single scattering with moments. However, the cost per pixel is much lower than for ray marching because we only need two texture samples (instead of 128). It is impractical, to perform ray marching at full resolution. The most widely used solution is to compute it at a much lower resolution and to use a bilateral upscale afterwards. However, this approach will certainly fail in presence of detailed geometry. With prefiltered single scattering, you do not need this upscaling. We also note that the cost of filtering is fairly small.

Finally, we draw some conclusions.

Conclusions

- Moment shadow maps outperform other filterable shadow maps:
  - The memory requirements are much lower than for convolution shadow maps,
  - Less light leaking than variance shadow maps, exponential shadow maps and exponential variance shadow maps.

Moment shadow maps are better than all earlier types of filterable shadow maps, at least in some way. Compared to convolution shadow maps the memory and bandwidth requirements are substantially lower. Compared to all other earlier techniques, there is less light leaking.

Conclusions (continued)

- Moment shadow maps are faster than common shadow maps for:
  - Small shadow map resolutions (feasible thanks to antialiasing),
  - Large output resolutions (i.e. 4k and VR),
  - Large filter regions (great for soft shadows).
- Surface acne is hardly a problem.

Common shadow maps are still a valid option but there are many scenarios where they are slower. Moment shadow maps are faster if the shadow map resolution is small, which is feasible thanks to multisample antialiasing. They are also faster when the output resolution is large. In 4k and virtual reality rendering you should definitely use them. Even at 1080p, they are quite competitive. Moment shadow maps also scale better to large filter regions which makes them a good fit for soft shadows and single scattering. CLICK

Another major advantage over common shadow maps is that surface acne is much less of a problem. With moment shadow maps you can likely get away by using a small, globally constant depth bias. Tweaking is not required.

Code, demos, papers and more

MomentsInGraphics.de

If you want to implement any of these techniques, please take a look at my new blog. There you will find shader code, executable demos, papers, videos and more.

Acknowledgments

- I would like to thank:
  - Michiel van der Leeuw and Giliam de Carpentier at Guerrilla Games who gave valuable feedback during review,
  - My coauthors Cedrick Münstermann, Nico Wetzstein and Reinhard Klein who contributed to the two i3D papers.

I would like to thank Michiel van der Leeuw and Giliam de Carpentier at Guerilla Games who gave a lot of valuable feedback during review. I would also like to thank my coauthors Cedrick Münstermann, Nico Wetzstein and Reinhard Klein who contributed to the two i3D papers.

My job search

- I'm about to hand in my PhD thesis,
- Do you have an open position in graphics R&D?
- Please contact me: peters@cs.uni-bonn.de

And last but not least, I am about to hand in my PhD thesis and will be looking for a job after that. If you have an open position in graphics R&D, please let me know.

That's it. Thank you for your attention and I'm looking forward to your questions.

References

- [Annen07] Annen, T., Mertens, T., Bekaert, P., Seidel, H.-P., and Kautz, J. (2007). Convolution shadow maps. In EGSR07: 18th Eurographics Symposium on Rendering, pages 51–60. Eurographics Association.
- [Annen08] Annen, T., Mertens, T., Seidel, H.-P., Flerackers, E., and Kautz, J. (2008). Exponential shadow maps. In GI ’08: Proceedings of graphics interface 2008, pages 155–161. Canadian Information Processing Society.
- [Donnelly06] Donnelly, W. and Lauritzen, A. (2006). Variance shadow maps. In Proceedings of the 2006 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, i3D ’06, pages 161–165. ACM.

References (continued)

- [Fernando05] Fernando, R. (2005). Percentage-closer soft shadows. In ACM SIGGRAPH 2005 Sketches. ACM.
- [Klehm14] Klehm, O., Seidel, H.-P., and Eisemann, E. (2014). Filter-based real-time single scattering using rectified shadow maps. Journal of Computer Graphics Techniques (JCGT), 3(3):7–34.
- [Lauritzen08] Lauritzen, A. and McCool, M. (2008). Layered variance shadow maps. In Proceedings of graphics interface 2008, pages 139–146. Canadian Information Processing Society.

References (continued)

- [Lauritzen11] Lauritzen, A., Salvi, M., and Lefohn, A. (2011). Sample distribution shadow maps. In Proceedings of the 15th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, i3D '11, pages 97–102. ACM.
- [Martin04] Martin, T. and Tan, T.-S. (2004). Anti-aliasing and continuity with trapezoidal shadow maps. In EGSR04: 15th Eurographics Symposium on Rendering, pages 153–160. Eurographics Association.
- [Peters15] Peters, C. and Klein, R. (2015). Moment shadow mapping. In Proceedings of the 19th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, i3D '15, pages 7–14. ACM.

References (continued)

- [Peters16] Peters, C., Münstermann, C., Wetzstein, N., and Klein, R. (2016). Beyond hard shadows: Moment shadow maps for single scattering, soft shadows and translucent occluders. In Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, i3D ’16, pages 159–170. ACM.
- [Reeves87] Reeves, W. T., Salesin, D. H., and Cook, R. L. (1987). Rendering antialiased shadows with depth maps. In Proceedings of the 14th annual conference on Computer graphics and interactive techniques, SIGGRAPH ’87, pages 283–291. ACM.

References (continued)

- [Salvi08] Salvi, M. (2008). ShaderX 6 , chapter Rendering filtered shadows with exponential shadow maps, pages 257–274. Cengage Learning Inc.
- [Williams78] Williams, L. (1978). Casting curved shadows on curved surfaces. In Proceedings of the 5th annual conference on Computer graphics and interactive techniques, SIGGRAPH ’78, pages 270–274. ACM.
- [Zhang06] Zhang, F., Sun, H., Xu, L., and Lun, L. K. (2006). Parallel-split shadow maps for large-scale virtual environments. In Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications, pages 311–318. ACM.


GDC
Bonus slides
GAME DEVELOPERS CONFERENCE EUROPE
COLOGNE, GERMANY · 15–16 AUGUST 2016
UBM


GDC
Computing the bounds
in moment shadow mapping
GAME DEVELOPERS CONFERENCE EUROPE
COLOGNE, GERMANY · 15–16 AUGUST 2016
UBM

Computing the bounds

- Given: Moments $b_j = \mathbb{E}_Z(z^j)$ for $j = 1 \dots 4$, depth z_f
- Problem: Find z_1, z_2, w_0, w_1, w_2 such that

$$\begin{pmatrix} 1 & 1 & 1 \\ z_f & z_1 & z_2 \\ z_f^2 & z_1^2 & z_2^2 \\ z_f^3 & z_1^3 & z_2^3 \\ z_f^4 & z_1^4 & z_2^4 \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}.$$

Suppose we are given four moments and one depth. We want to compute two additional depth values and three weights such that all moments match. Additionally, the weights have to sum up to one.

Computing the bounds

- Solution: Compute roots z_1, z_2 of polynomial

$$\begin{pmatrix} 1 \\ z_{1/2} \\ z_{1/2}^2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 & b_1 & b_2 \\ b_1 & b_2 & b_3 \\ b_2 & b_3 & b_4 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 \\ z_f \\ z_f^2 \end{pmatrix} = 0.$$

Now we can either solve this equation by hand or we can use some clever tricks. Either way, we find that it implies the following equation. Note that the weights are gone. And the two depth values are now simply the two roots of a quadratic equation. We can easily solve that.

Computing the bounds

- Input: Depth $z_f \in \mathbb{R}$, moments $b_j = \mathbb{E}_z(\boldsymbol{z}^j)$ for $j = 1 \dots 4$
- Output: Approximate shadow intensity

1. Solve $\begin{pmatrix} 1 & b_1 & b_2 \\ b_1 & b_2 & b_3 \\ b_2 & b_3 & b_4 \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 \\ z_f \\ z_f^2 \end{pmatrix}$
2. Solve $c_3 \cdot z^2 + c_2 \cdot z + c_1 = 0$ for $z_1, z_2 \in \mathbb{R}$
3. Solve $\begin{pmatrix} 1 & 1 & 1 \\ z_f & z_1 & z_2 \\ z_f^2 & z_1^2 & z_2^2 \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ b_1 \\ b_2 \end{pmatrix}$
4. Return $\sum_{i=1, z_i < z_f}^{2} w_i$ (add w_0 for upper bound)

By putting the pieces together, we get our main algorithm. It computes an approximate shadow intensity from four moments and a fragment depth. CLICK

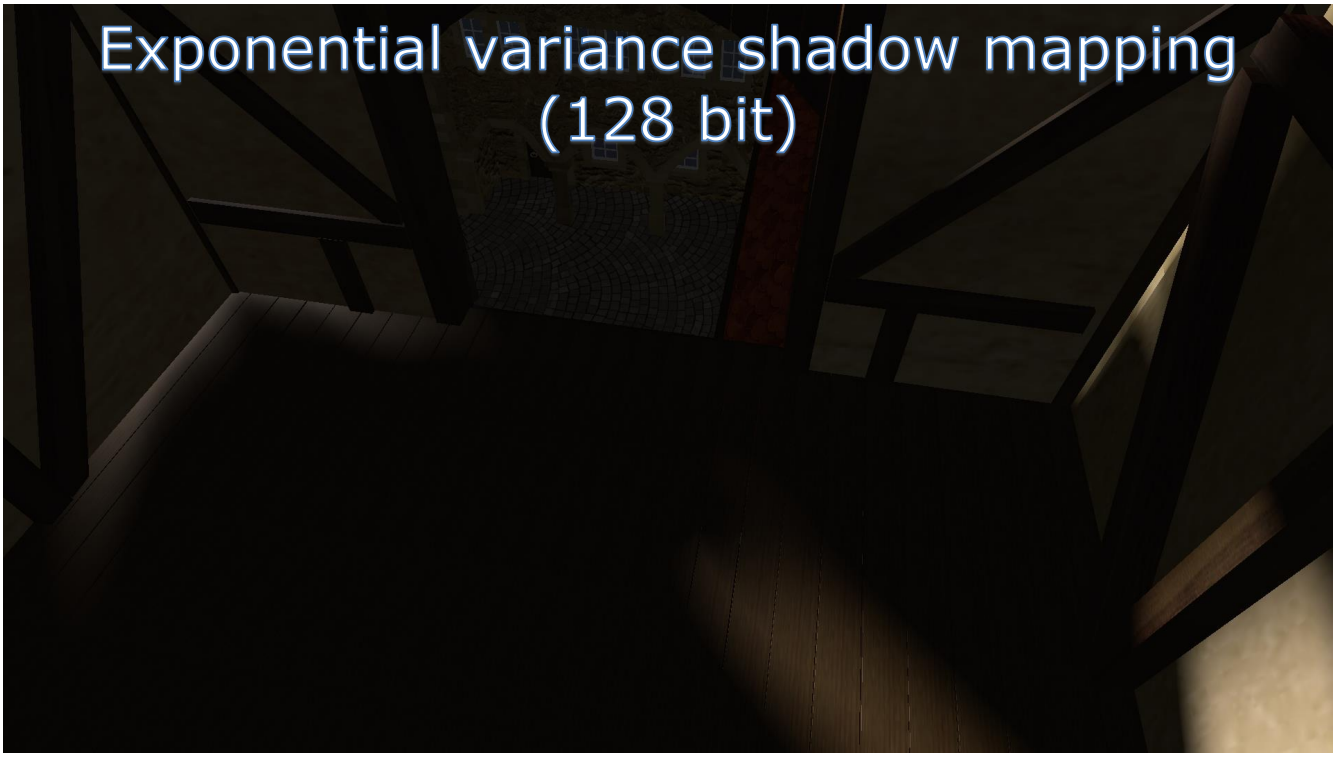
First, we have to compute the coefficients of the quadratic equation from the previous slide by solving a system of linear equations. CLICK

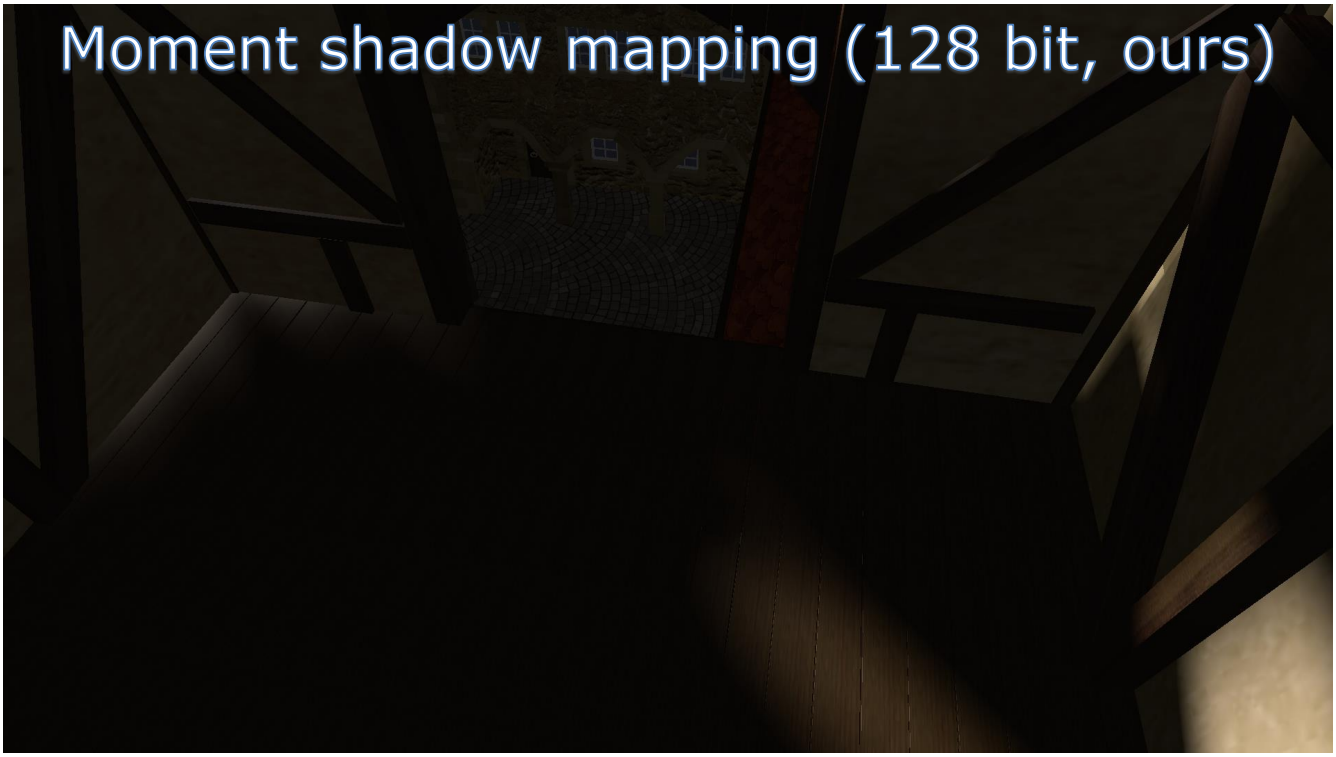
Then we solve the quadratic equation itself using the quadratic formula. CLICK

Now the three weights can be computed using only three of our five linear equations. CLICK

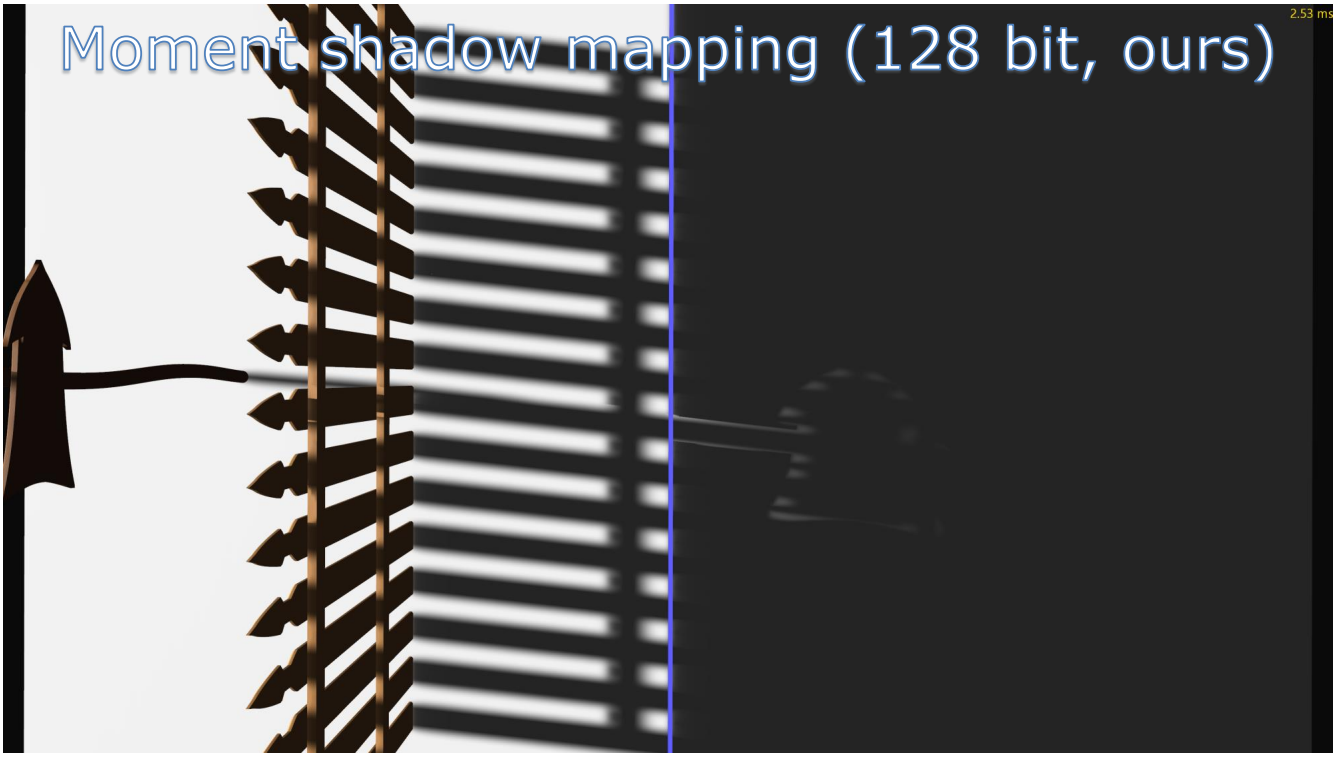
The lower bound is obtained by simply adding up the appropriate weights. That is the lower bound for our shadow intensity. We can also add weight zero to get the upper bound. When implementing this technique it is crucial to use numerically stable methods for every step. Most importantly, you have to use a Cholesky decomposition in step one.

Moment shadow maps at 128 bits


Exponential variance shadow mapping
(128 bit)


Moment shadow mapping (128 bit, ours)


Exponential variance shadow mapping
(128 bit)
2.48 ms


Moment shadow mapping (128 bit, ours)
2.53 ms


GDC
Use in production
GAME DEVELOPERS CONFERENCE EUROPE
COLOGNE, GERMANY · 15–16 AUGUST 2016
UBM

THE DANGER ZONE

99.9% genuine graphics programming

UPDATE FOR MY SHADOW SAMPLE UPDATE

WRITTEN BY MJP

JANUARY 24, 2016

Recently I was contacted by Christoph Peters (one of the authors of Moment Shadow Mapping) regarding a blog post where I compared EVSM to MSM using my sample app. He noticed that I was incorrectly clamping the maximum exponential warp to 10.0 for the 16-bit variant of EVSM, which can result in values that are greater than what can be stored in a 16-bit floating point texture. Doing this has 2 effects: it causes incorrect results during filtering (clamping the moments can lead to negative variance, causing a reconstruction that resembles a step function), and it reduces the amount of light bleeding. Unfortunately all of my comparisons were done without any additional prefiltering, and so I didn't notice my error. Even more unfortunate was that it misrepresented the extent of EVSM16's light bleeding issues in my comparison images, making the test unfairly skewed in favor of EVSM.

To help make things right, I've uploaded a new version of the sample app to GitHub that correctly clamps the exponential warp factor to 5.54. The releases page has a zip file containing code and a compiled executable. I've also updated the original blog post in question with new comparison images, as well as new commentary on those images. You'll notice that my thoughts on MSM have become quite a bit more positive now that I've taken a second look, and performed a more fair analysis. In fact, it now seems that MSM16 is hands-down the better option if you're looking to stay in an 64bpp footprint for the shadow map.

Finally, I'd like to apologize to Christoph and Reinhard for my unfair comparison. The work they've presented (including some new research being presented at I3D 2016) is very promising, and it's great to see research that's immediately usable and practical for games. I also need to call out that they release sample code alongside their work, which is extremely helpful for evaluating and understanding their techniques. Hopefully we see more great things from them in the future!

RECENT POSTS

- Bindless Texturing for Deferred Rendering and Decals
- Update For My Shadow Sample Update
- Stairway To (Programmable Sample Point) Heaven
- SIGGRAPH Follow-Up: 2015 Edition
- Mitsuba Quick-Start Guide

ARCHIVES

- March 2016
- January 2016
- September 2015
- August 2015
- April 2015
- March 2015
- February 2015
- March 2014
- February 2014
- September 2013
- July 2013
- June 2013
- November 2012
- October 2012
- August 2012
- April 2012
- March 2012


Home
Matt Pettineo
@MyNameIsMJP
Follow
After days of work I've finally decided that my sun shadows don't completely suck, which is really the pinnacle of real-time shadow tech.
RETWEETS
2
LIKES
38
2:28 PM - 14 May 2016
2
38
Leonard Ritter @paniq · May 14
@MyNameIsMJP what did you end up using?
Matt Pettineo @MyNameIsMJP · May 14
@paniq mostly changes to our cascade partitioning, by I also switched to MSM from EVSM. So far it's better quality with no increase in cost.
1
16
View other replies


GDC
More on prefiltered single scattering
GAME DEVELOPERS CONFERENCE EUROPE
COLOGNE, GERMANY · 15–16 AUGUST 2016
UBM

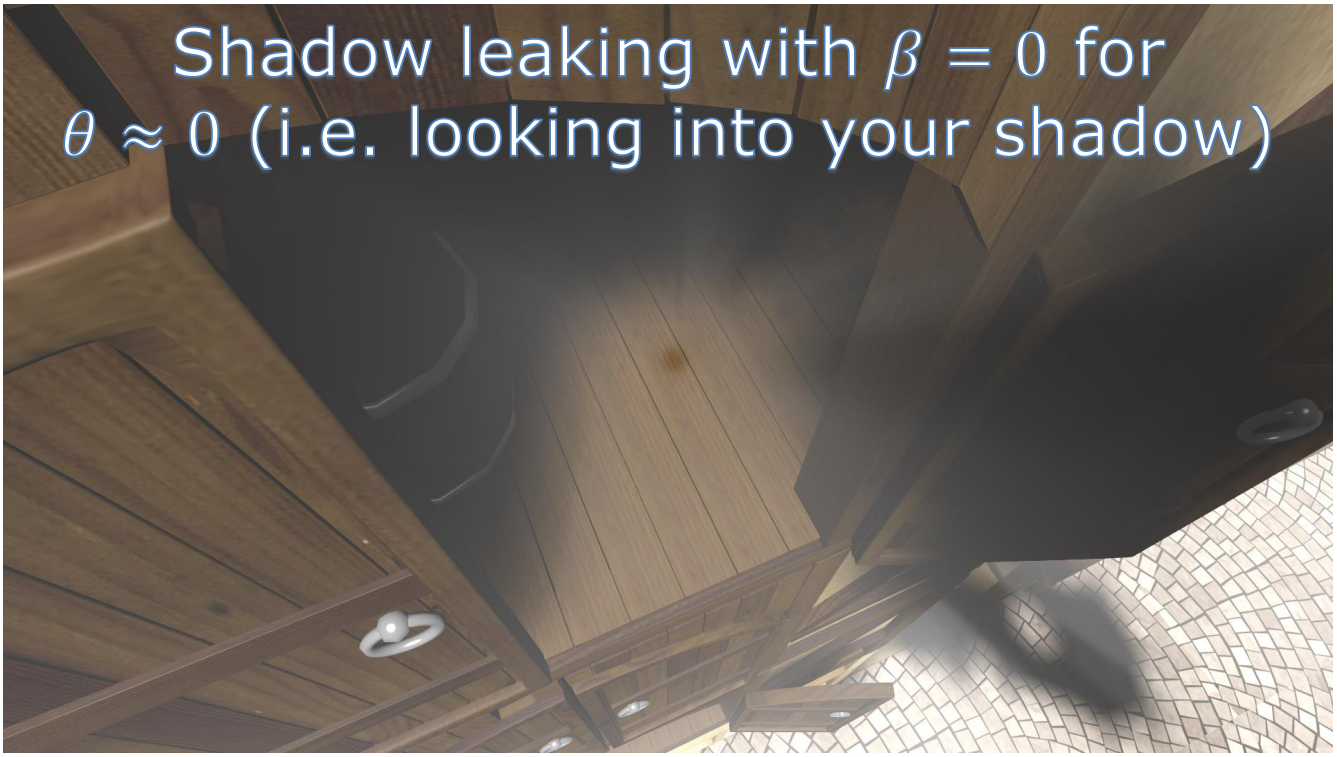
Interpolation between bounds

- Algorithm yields lower and upper bounds.
- We can interpolate arbitrarily:

$$\beta \cdot \text{lower bound} + (1 - \beta) \cdot \text{upper bound}$$

- $\beta = 0$: Single scattering always too dark,
- $\beta = 1$: Single scattering always too bright,
- Choose β intelligently per pixel.


Light leaking with $\beta = 1$ for
$\theta \approx \pi$ (i.e. looking into the light)


Shadow leaking with $\beta = 0$ for
$\theta \approx 0$ (i.e. looking into your shadow)

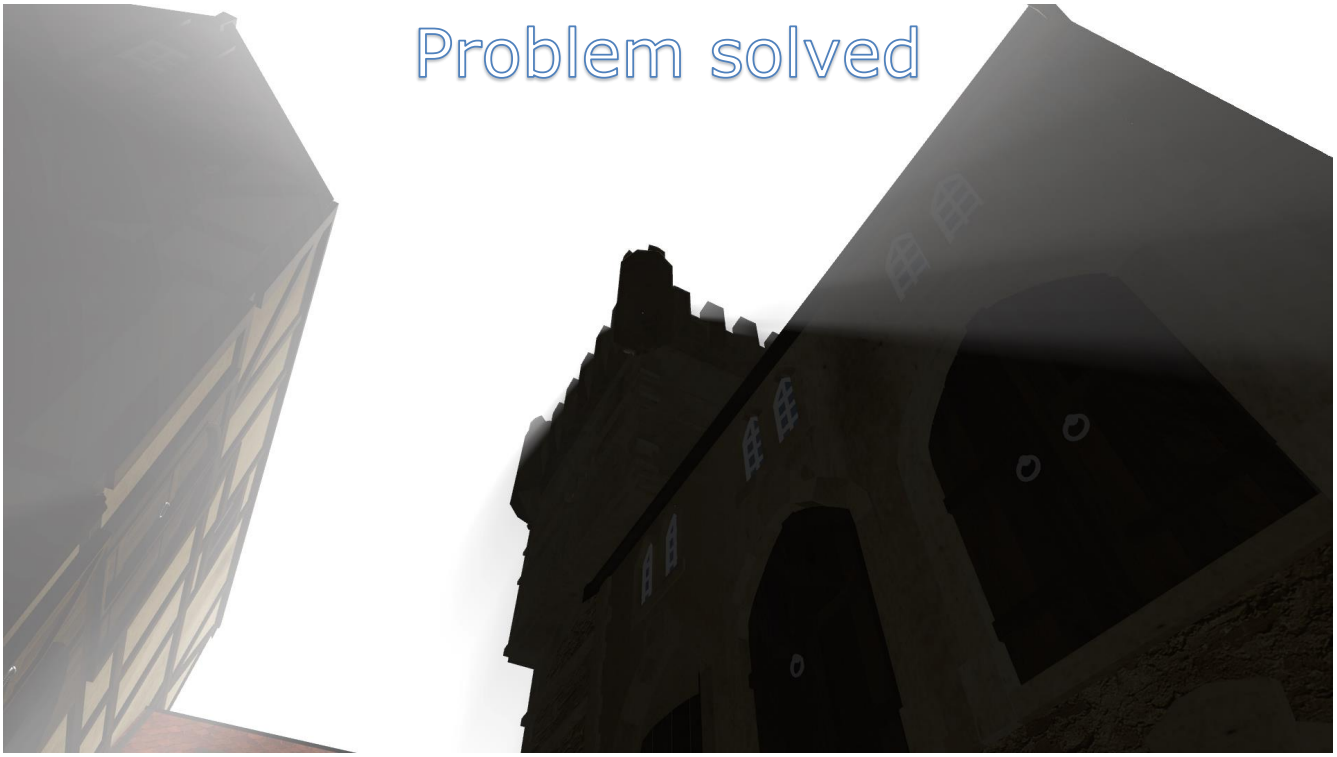
Adaptive overestimation

- We are looking for a smooth function that is 0 for $\theta = \pi$ and 1 for $\theta = 0$.
- The following looks good:

$$\beta := \frac{1 + \omega_l^T \cdot \omega_p}{2},$$

  - Where ω_l is the normalized light direction,
  - And ω_p is the normalized view ray direction.

Problem solved



Problem solved

4 moments suffer from leaking