These are the annotated slides of the “real-time” part of the Many-Lights Rendering course at SIGGRAPH ‘12.

This part covers techniques for making many lights methods suitable for (high-quality) real-time rendering.

You can find some of the papers (list of references on the last slide) and accompanying videos on our webpage: http://cg.ibds.kit.edu

Real-time Many-light Rendering

Outline

- main difference to offline-methods is visibility computation
  - rasterization instead of raycasting
  - VPL generation
  - lighting and shadowing from VPLs
- high-quality rendering
  - bias compensation in screen-space
  - approximate compensation in participating media rendering

As we will see, it is basically all about visibility computation.

Whereas offline methods often simply use ray casting to determine visibility – between arbitrary points in the scene or between VPLs and surfaces - real-time rendering still (and almost exclusively) means using rasterization hardware, which is best suited to render scenes from pinhole cameras and similar projections.

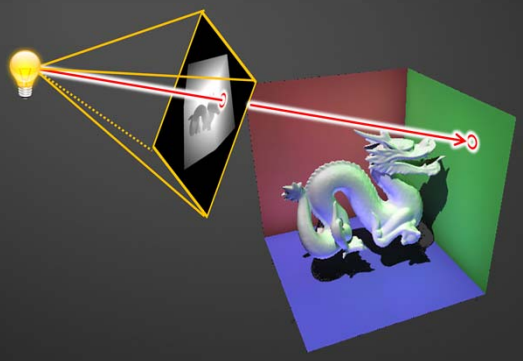
We will have a look at the consequences of this for VPL generation, for lighting and shadowing from VPLs.

The second part will be on high-quality rendering with bias compensation and participating media rendering.

Real-time Many-light Rendering

Visibility Computation for VPL Generation

- real-time rendering ↔ mostly diffuse scenes ↔ relatively few VPLs (~10^3)
- if acceleration structure available use ray casting
- VPL generation with rasterization
  - render scene from light
  - observation: visible surfaces = first intersection of light path



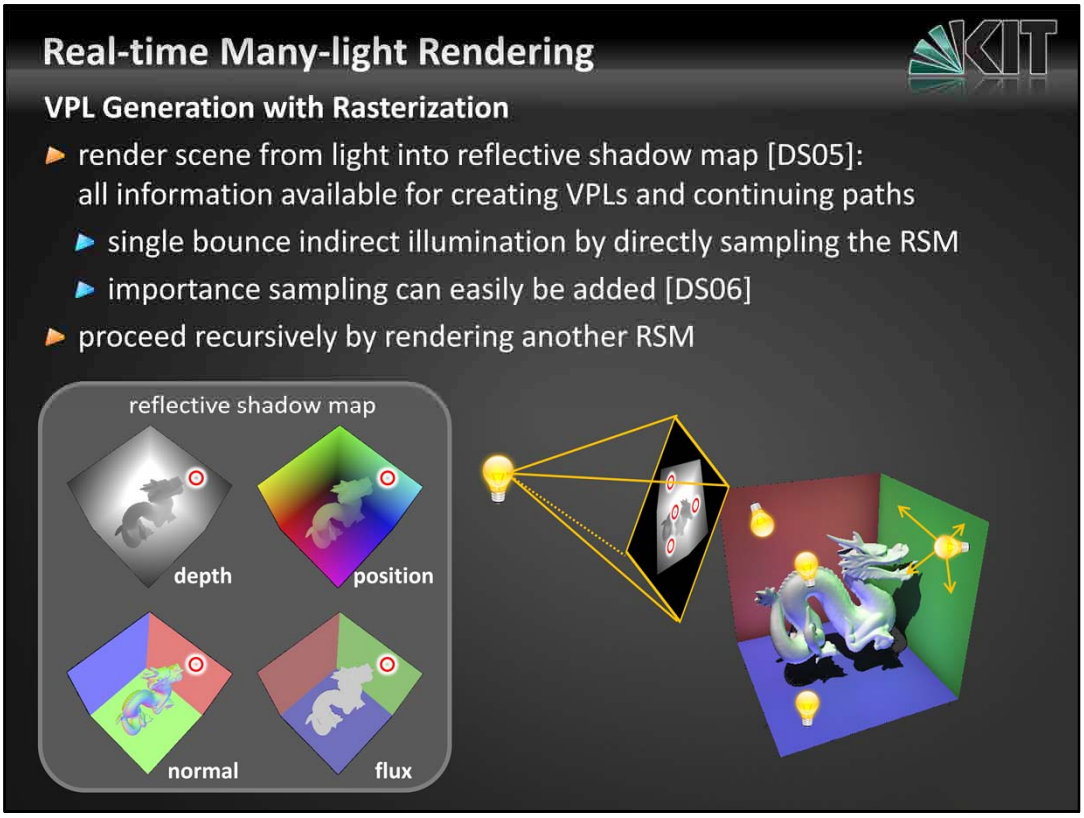
For VPL generation, we should keep one thing in mind: as we‘re targetting real-time performance, the number of VPLs that we can handle is limited, typically to a few thousand VPLs. This in turn means that the scenes we can handle are mostly diffuse or very moderately glossy.

One thing is obvious for the VPL generation: if you happen to have an acceleration structure for your scene, then use ray casting for the random walks.

If you don‘t, maybe because your scene is dynamic, then we can still generate VPLs using rasterization only.

It’s based on a simple observation: imagine that you render the scene from the primary light source (here a spot light and its depth buffer is shown).

…then the visible surfaces are those that are connected by a single light path segment to the light source.



In order to create a VPLs we need more information about those surfaces, e.g. the position (which can be computed from depth), the surface normal and the material, but all this can be created altogether in a single render pass.

The reflective shadow map paper [DS05] uses this information to compute single-bounce indirect illumination by directly sampling these textures.

In our case, we would choose a random subset of pixels, and use the available information to create VPLs at the respective locations.

It is also possible to importance sample the RSM [DS06] in order to generate more VPLs on glossy surfaces for examples. This can easiest be done by warping an initially uniform sampling pattern according to an importance function.

And lastly, we need to continue a subset of the light paths, which is done by rendering another RSM, again picking a subset of pixels, and so forth.

Of course it is expensive to render the scene multiple times, but a few slides further we will address exactly this problem (with micro-rendering).

Rendering with VPLs

Lighting and Shadowing

- many lights can be handled with deferred shading
  - interleaved sampling (problem: detailed normals/geometry) [Seg06]
  - hierarchical shading [NW10]
  - accumulate and filter incident light [SW09]
  - clustered deferred and forward shading [OBA12]

- **bottleneck: shadow computation**

For now let‘s assume that we generated the VPLs and want to use them.

When rendering with many light sources, we usually use deferred shading to decouple shading cost from geometric complexity of the scene and restrict shading to visible surfaces only.

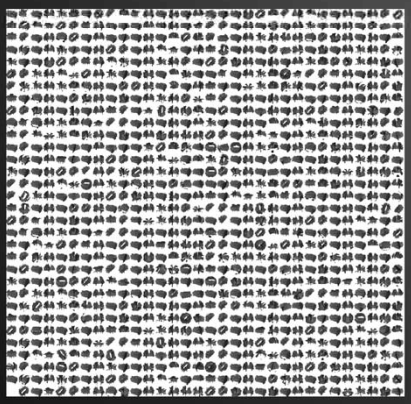
A lot of ideas have been developed for improving the performance with deferred shading, e.g. interleaved sampling, hierarchical shading in image space, filtering incident light to handle detailed geometry and glossy surfaces, and clustering of VPLs for image tiles.

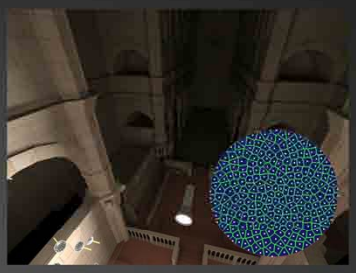
All these works are great as they push the number of VPLs that we can handle further, but the real bottleneck is visibility or shadow computation for the VPLs.

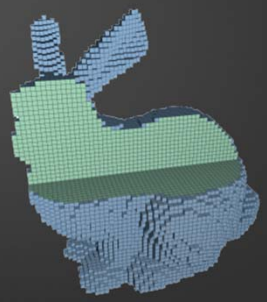
Rendering with VPLs

Shadow Computation

- …is the real bottleneck with instant radiosity / many lights methods
  - exploit temporal coherency [LSKLA07]
  - sampled visibility
    - voxelization, e.g. [SS10]
    - faster shadow maps



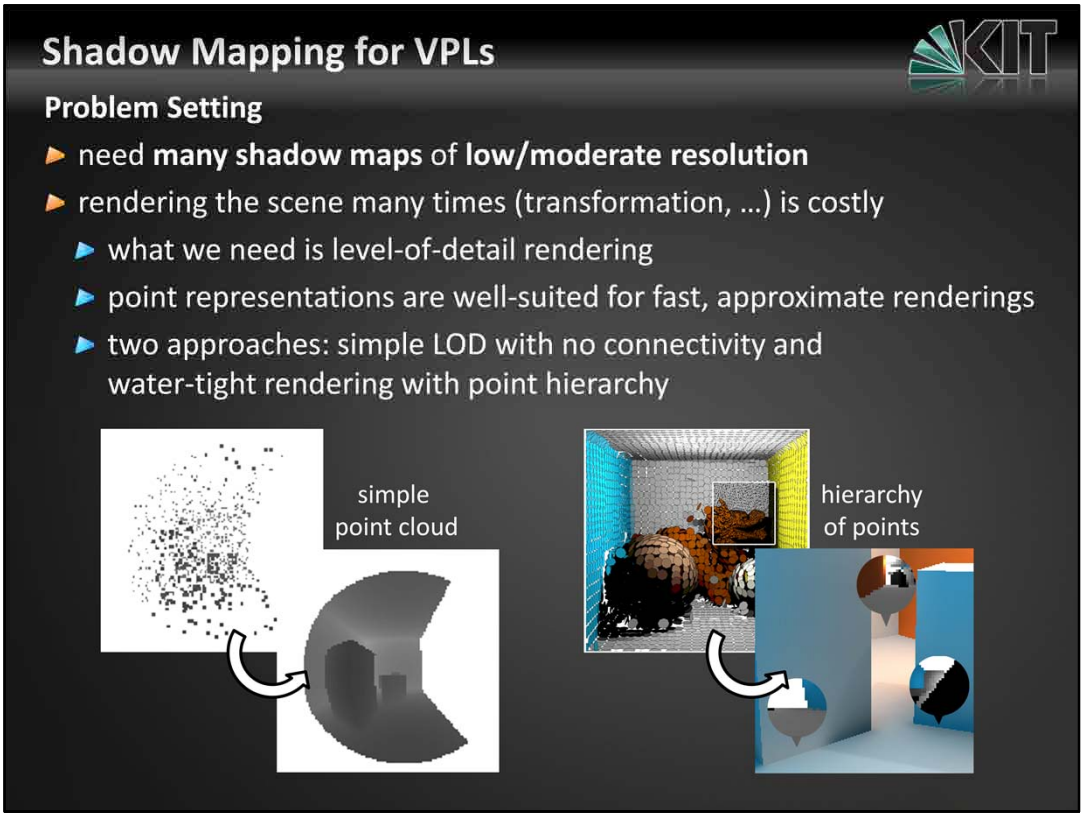




This problem has been addressed from very different directions.

Laine et al., simply speaking create VPLs and reuse them, as long as they remain directly lit – and thus reduce the number of shadow maps that have to be generated each frame. Unfortunately, maintaining a list of active VPLs is only feasible for single bounce illumination.

Another way which is becoming more and more popular is to use voxelization and ray marching to compute visibility, but what this presentation covers is how to compute shadow maps fast.



… because we need one shadow map for every VPL.

As we have many VPLs, the resolution of each individual shadow map does not have to be high, as the soft indirect illumination by VPLs does not show sharp features.

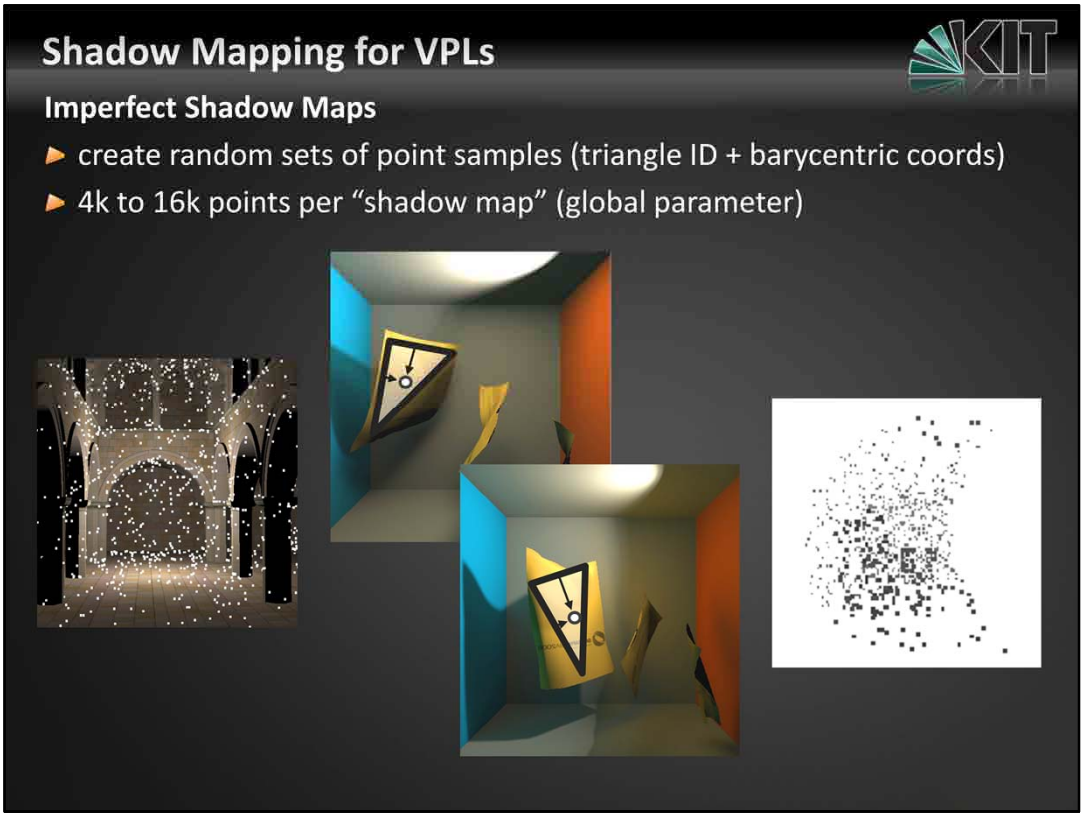
However, the situation is quite unfortunate, as the major cost in rendering tiny shadow maps is independent of the resolution: geometry processing, transformation, rasterizer setup and so on.

So what we ultimately would like to have is means to efficiently render a low detail version of the scene.

Of course there exist triangle mesh reduction algorithms which could be used to generate such versions, but they are not so well suited for rendering many tiny images (rather few high-quality ones).

However, point-based rendering is very well-suited for fast and approximate rendering with level-of-detail.

In particular we can use very simple point cloud rendering for computing approximate shadow maps, or techniques which use point hierarchies for efficiently creating many high-quality renderings or shadow maps of a scene.



The first one is the “imperfect shadow maps” method.

The idea is very simple: first of all we create a set of points randomly distributed on the surfaces of your scene.

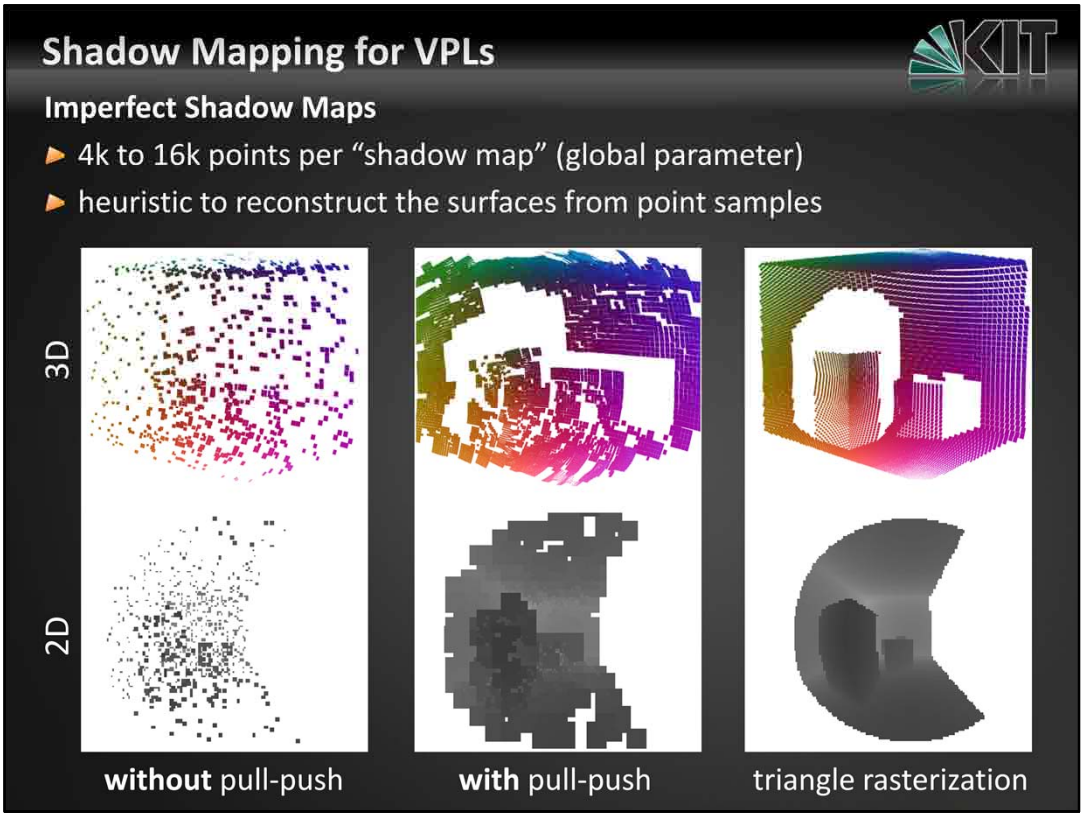
In order to reuse these points in dynamic scenes, we do no store there absolute position, but instead the triangle on which they reside together with the two barycentric coordinates.
Once the triangle moves, we can compute the absolute position of the point sample from the vertex positions.

Then we use a few thousand of these points and render the shadow map (typically with a paraboloid projection for hemispherical shadow maps).

And you get this beautiful result on the right.

The problem is that the number of point samples was obviously not enough to reasonably reconstruct the surfaces in the projection.

And in fact, this is a problem with simple point-based rendering: the number of point samples has to be quite large…



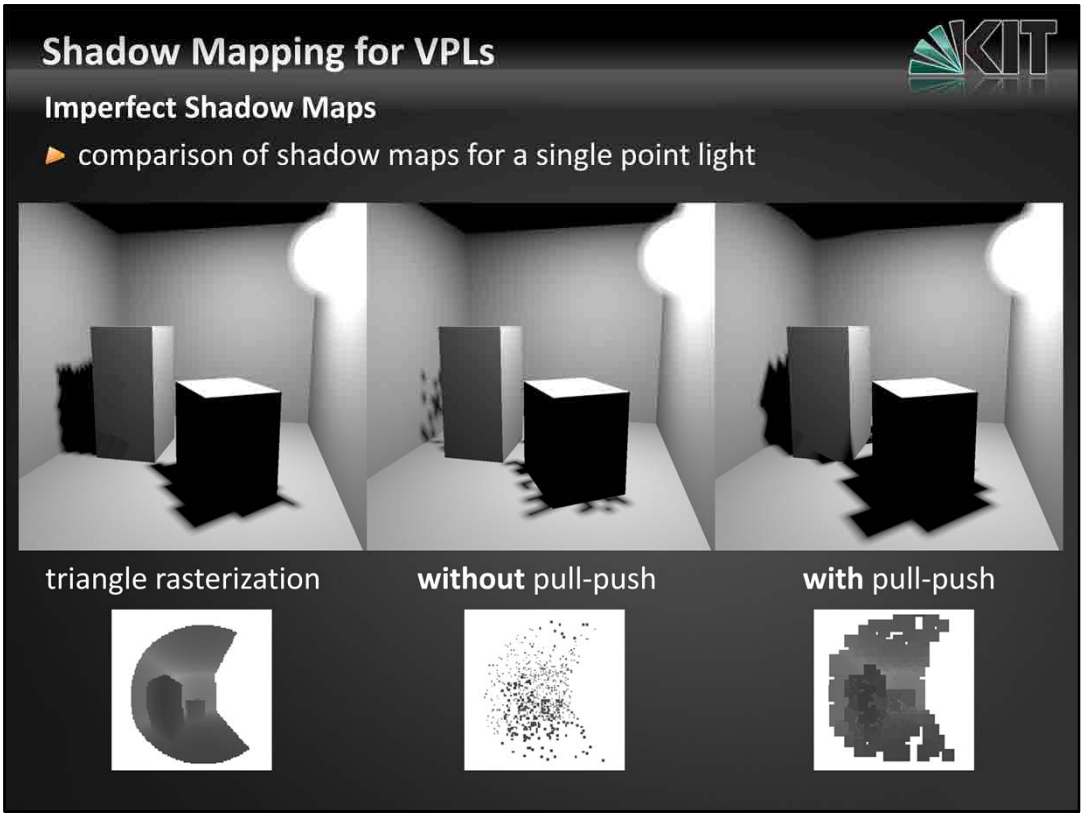
Here we see the same shadow map again and also the visible surfaces projected back into 3D.

The right image shows the shadow map rendered with the original geometry.

The question when developing ISMs was whether these coarse point renderings make sense at all…

Fortunately there are quite simple heuristics to reconstruct the surfaces even from this incomplete information.

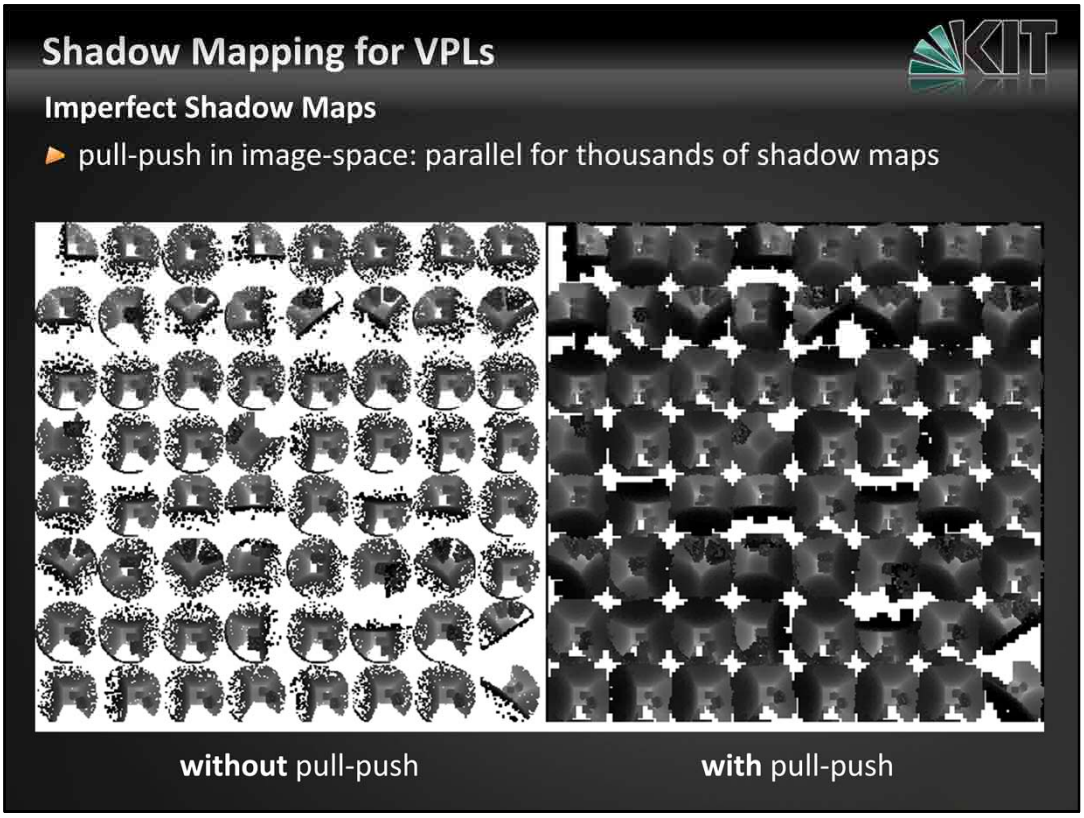
It basically works like this: first you create an image pyramid of the depth image where only valid pixels are used to compute average pixels in coarser levels.
And in a second phase, holes are filled by interpolation of pixels from the coarser levels.



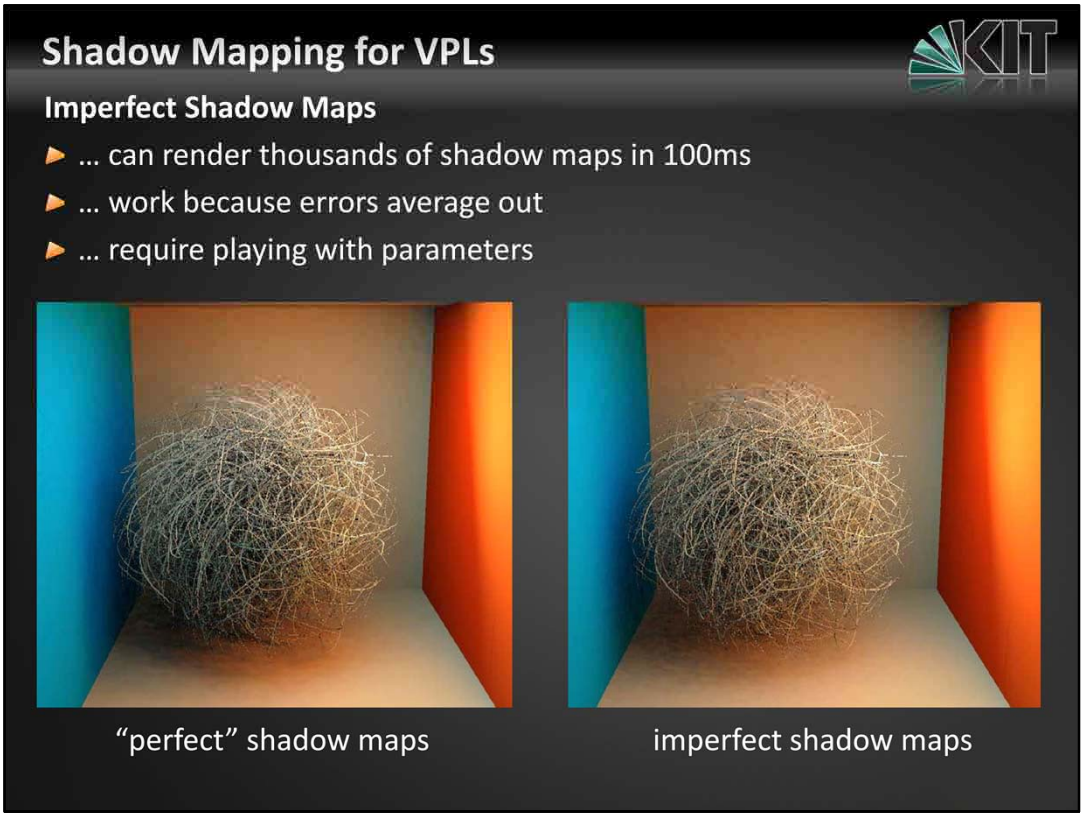
Here you can see an imperfect shadow map used for a single point light.

If you compare these three images then the depth map reconstructed using this pull-push heuristic does not seem to be that bad.

Of course the result is not absolutely accurate, but as we will use them for a large number of VPLs these errors will basically average out.



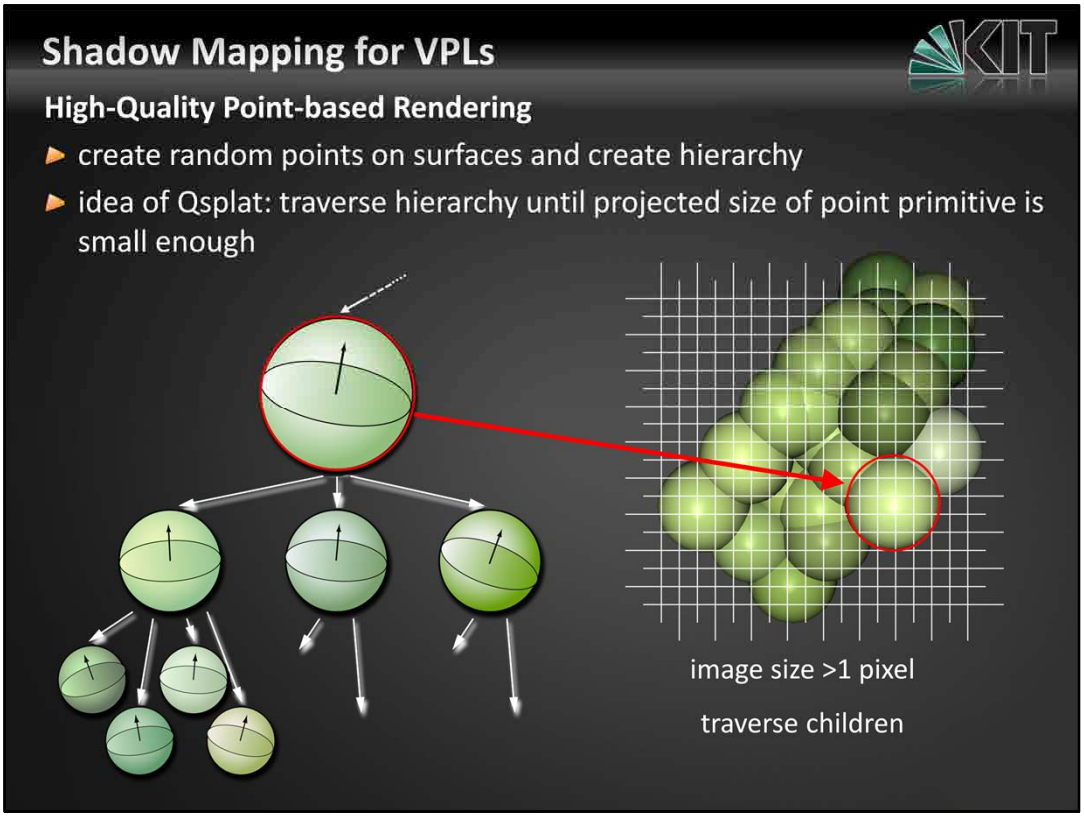
And another good thing is that we can perform this pull-push step on all depth maps in parallel when they are stored in one large texture – it's just an operation on a 2D image which is independent of the geometric complexity of the scene.



With this method we can easily create several thousands of ISMs in a few hundred milliseconds.

And the results for many-lights rendering are almost indistinguishable from real shadow maps, simply because these errors all average out.

However, in addition to playing with the shadow map parameters (e.g. for bias), we have even more to tweak for the pull-push heuristic.



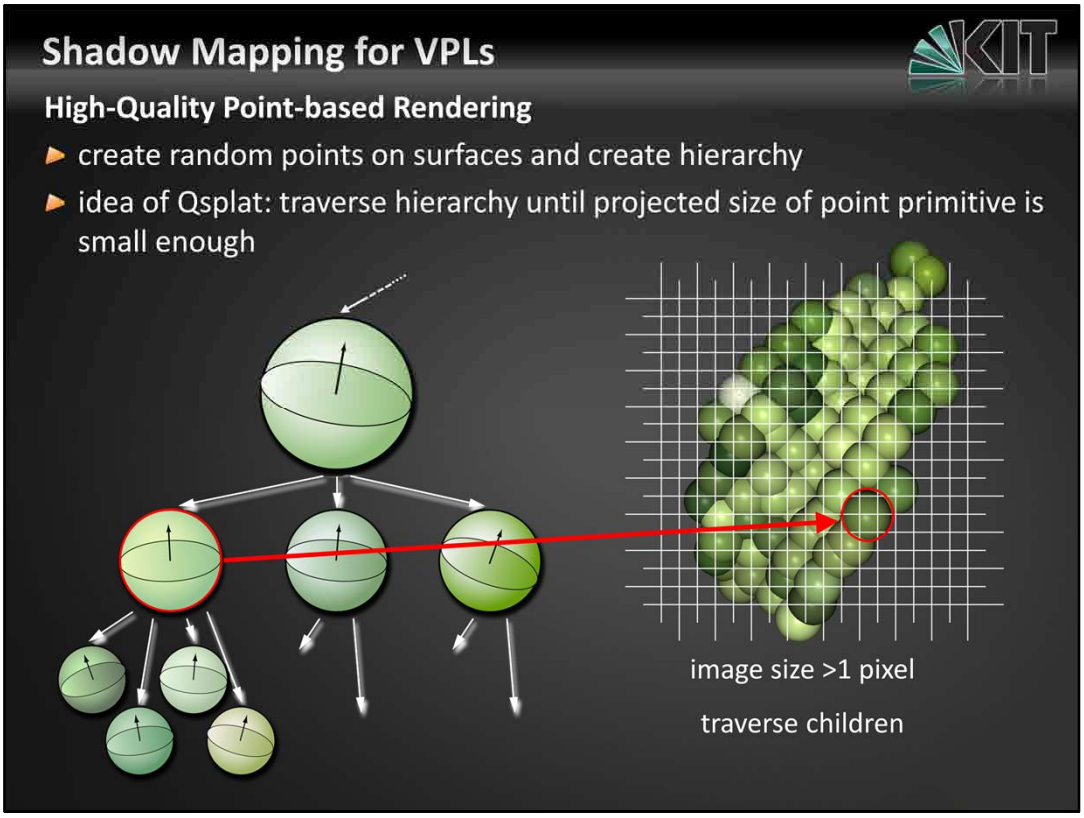
One way to avoid this is to use high-quality point-based rendering.

In principle you start by distributing points on the surfaces again where each points represents a part of the surfaces.

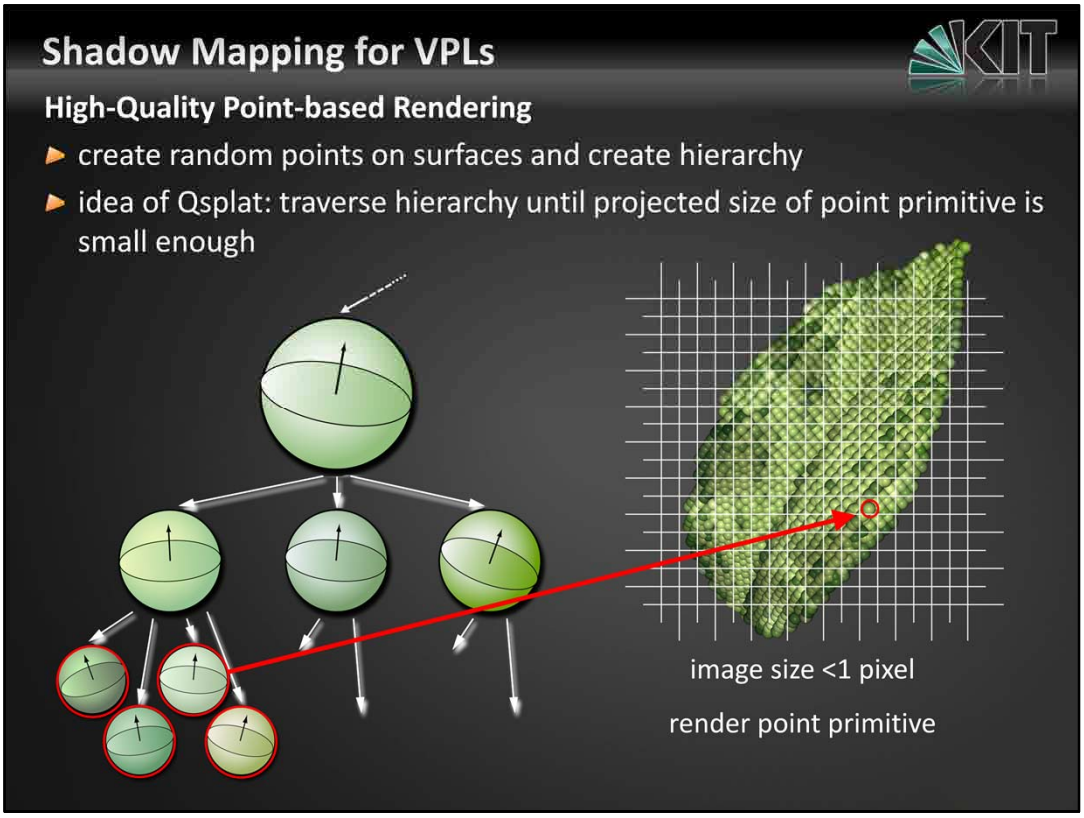
Then you build a hierarchy on top of them, e.g. by recursively merging nearby point samples to larger ones.

To render an image you basically want to get away using a few point samples as possible. And you do this by traversing the hierarchy staring from the root node to find the cut, where the projected size of a point sample is just below a certain threshold, e.g. the size of one pixel.

So here we would decide that the cut has to be refined…



…refined again…

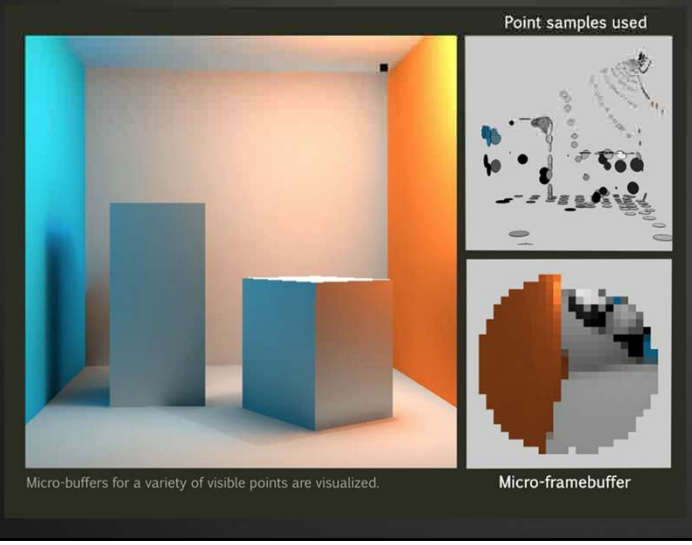


and here we finally found the point sample we would like to render as a single pixel, of course combined with depth buffering.

Shadow Mapping for VPLs

Micro-Rendering

- renders accurate environment maps / depth buffers from point hierarchy
- actually developed for final gathering, using CUDA/OpenCL
- can be used to create (R)SMs (in 2009: ~16k in 100 ms, each 24^2 pixels)

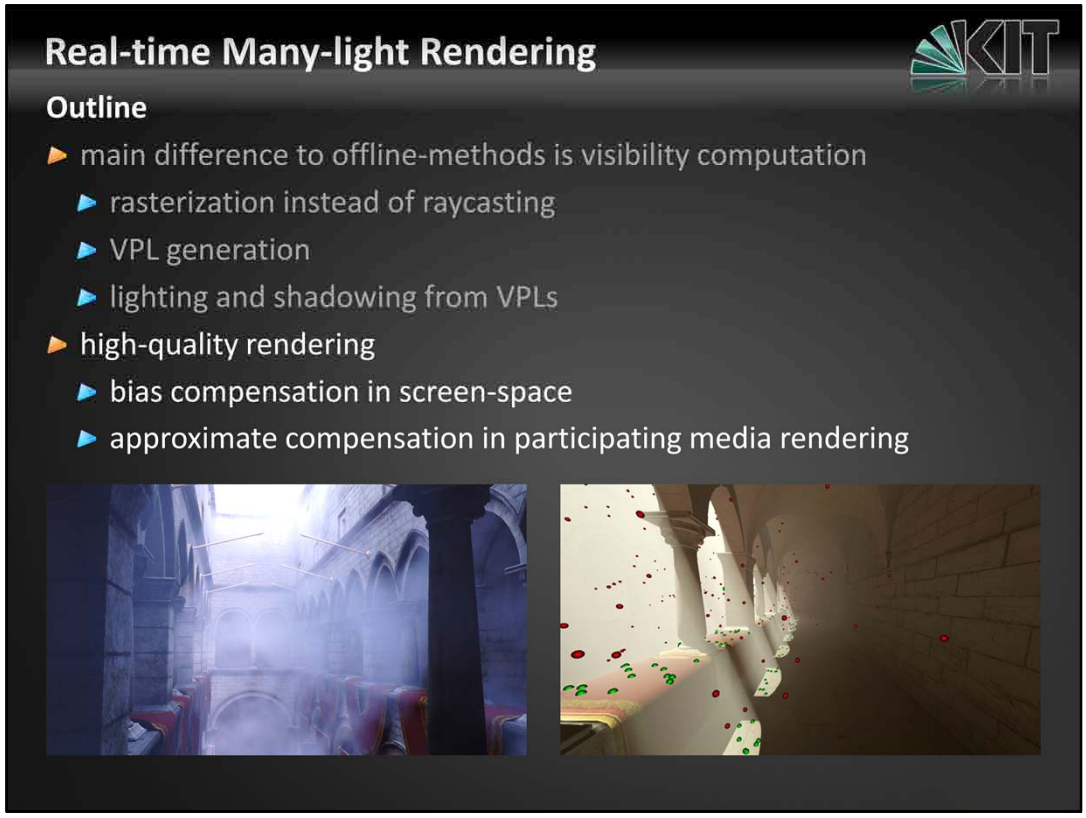


The micro-rendering paper uses exactly this idea to render small environment maps (with depth buffers) from point hierarchies using CUDA.

As it was initially meant for final gathering, it supports warped projections for importance sampling and things which are not important for VPLs.

But we can use it to render a large number of low-resolution shadow maps and reflective shadow maps (to create VPLs) very efficiently and with high accuracy.

Real-time Many-light Rendering



With the ideas that I mentioned so far we have a pretty good set of techniques for VPL generation, shading and shadow computation.

In the following we focus more on the aspects of high-quality rendering.

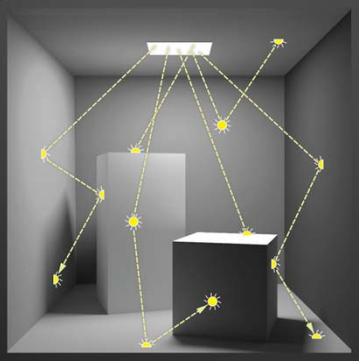
Singularities and Bias Compensation

- so far: VPL generation, shading and shadowing
- we assume to use VPLs to approximate indirect illumination $\hat{L}$ only

$$L = L_e + \mathbf{T}L$$

$$L = L_e + \mathbf{T}L_e + \mathbf{T}\hat{L}$$

direct emission | direct illumination | indirect illumination



And as this is related to indirect illumination only, we will – from now on – assume that we use VPLs to approximate indirect illumination only

This makes perfect sense if you keep in mind that direct illumination in real-time applications is typically computed using some special methods for area lights and soft shadows.

We also don't care how these VPLs have been created, we only know that their joint contribution makes up the indirect light in the scene.

In terms of the operator notation we have 3 summands: emission, direct illumination and indirect illumination, and no recursion anymore.

T L-hat is the transport from the collection of VPLs that represent indirect illumination.

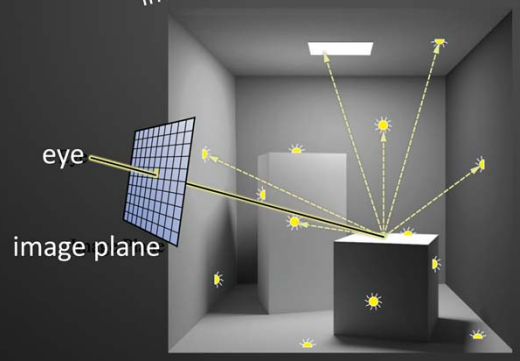
Singularities and Bias Compensation

- so far: VPL generation, shading and shadowing
- we assume to use VPLs to approximate indirect illumination $\hat{L}$ only

$$L = L_e + \mathbf{T}L$$

$$L = L_e + \mathbf{T}L_e + \mathbf{T}\hat{L}$$

direct emission
direct illumination
indirect illumination



Let‘s have a quick look at the problems which might occur when computing the indirect light…

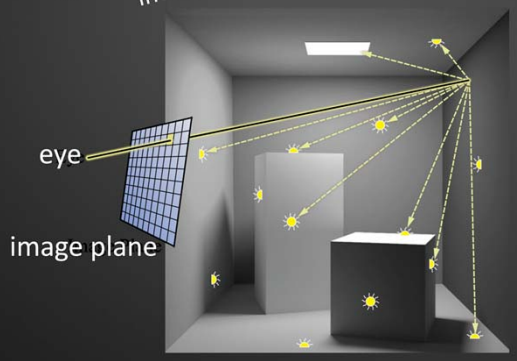
In this case everything is fine, we just accumulate the contributions from all VPLs.

Singularities and Bias Compensation

- so far: VPL generation, shading and shadowing
- we assume to use VPLs to approximate indirect illumination $\hat{L}$ only

$$L = L_e + \mathbf{T}L$$

$$L = \underbrace{L_e}_{\text{direct emission}} + \underbrace{\mathbf{T}L_e}_{\text{direct illumination}} + \underbrace{\mathbf{T}\hat{L}}_{\text{indirect illumination}}$$



… but for this surface location that‘s not the case… because a nearby VPLs will create bright splotch.

Singularities and Bias Compensation

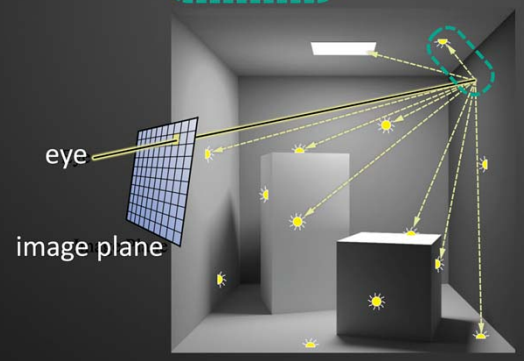
$$L = L_e + \mathbf{T}L_e + \mathbf{T}\hat{L}$$

transport operator:

$$(\mathbf{T}\hat{L})(\mathbf{x}\leftarrow\mathbf{y}) = \sum_{i=1}^{N} f_r(\mathbf{x}\leftarrow\mathbf{y}\leftarrow\mathbf{z}_i)\,G(\mathbf{y}\leftrightarrow\mathbf{z}_i)\,V(\mathbf{y}\leftrightarrow\mathbf{z}_i)\,\hat{L}(\mathbf{y}\leftarrow\mathbf{z}_i)$$

geometry term:

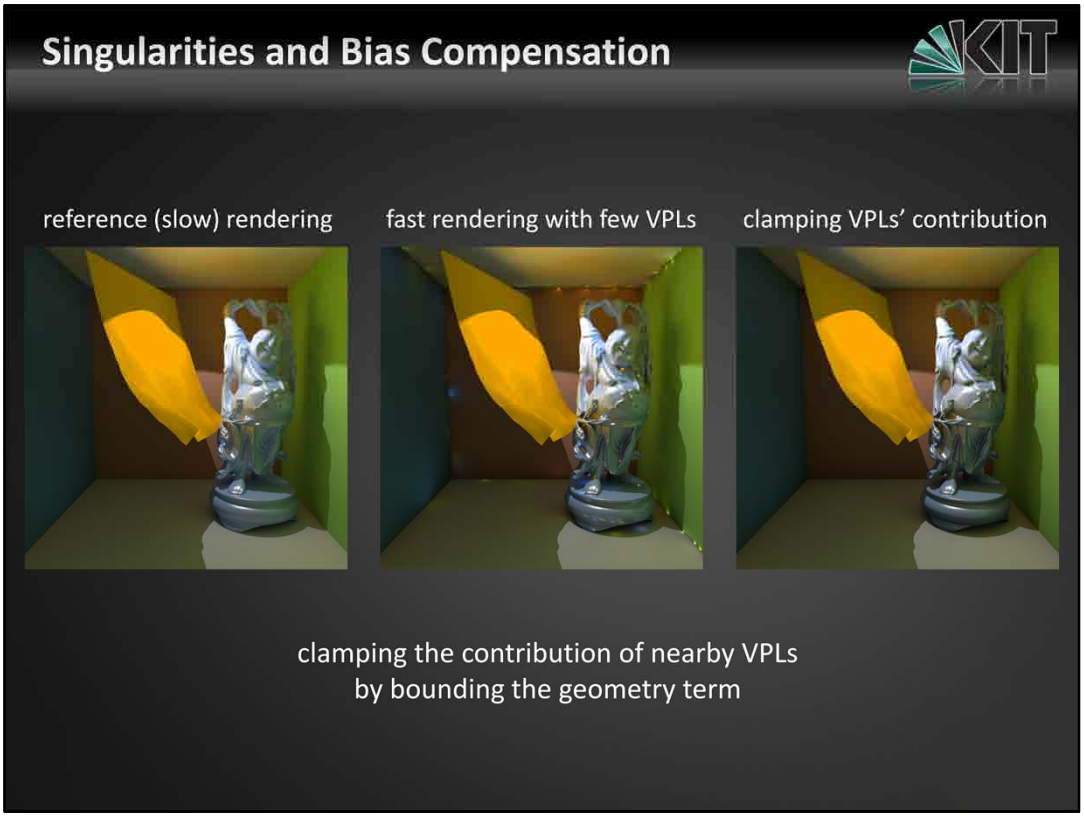
$$G(\mathbf{y}\leftrightarrow\mathbf{z}_i) = \frac{\cos^+(\theta_\mathbf{y})\cos^+(\theta_{\mathbf{z}_i})}{\|\mathbf{y}-\mathbf{z}_i\|^2}$$



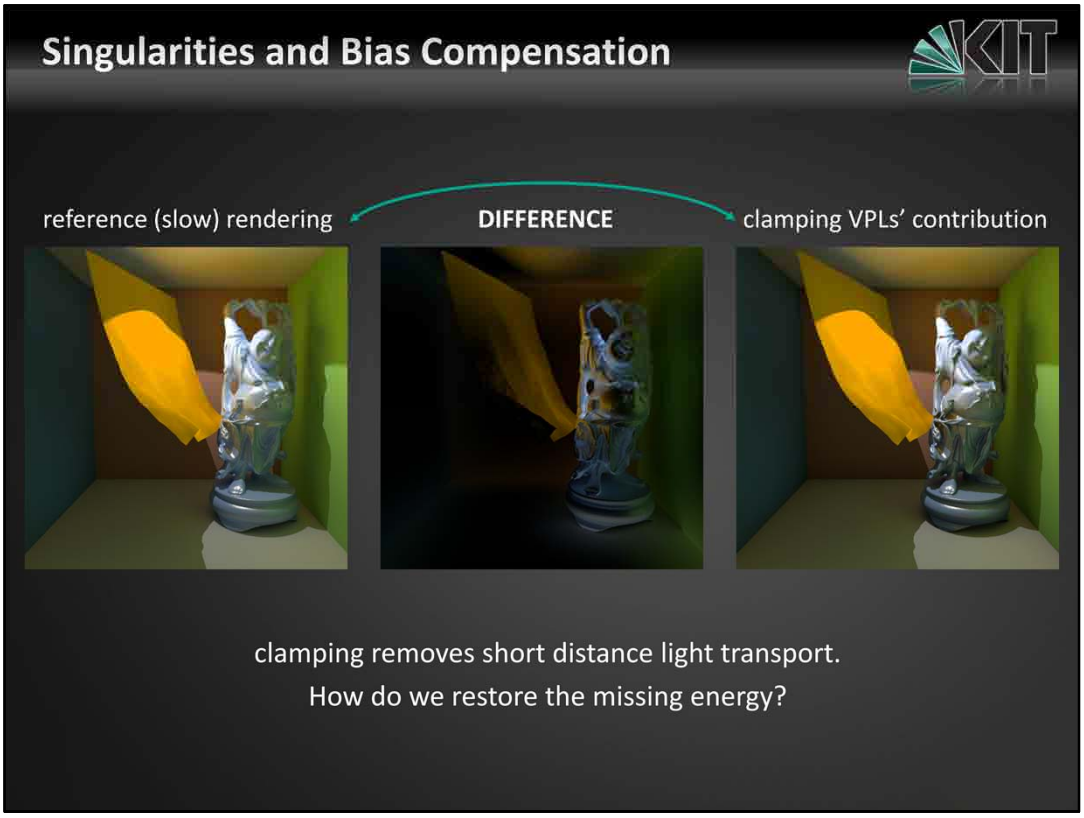
The reason for this can be observed when looking at the transport for VPL lighting.

T-L-hat is nothing but a sum over all VPLs, taking account for the mutual visibility and the geometry term between the VPL and the surface point.

And the geometry term can become arbitrarily large, as we divide by the squared distance.

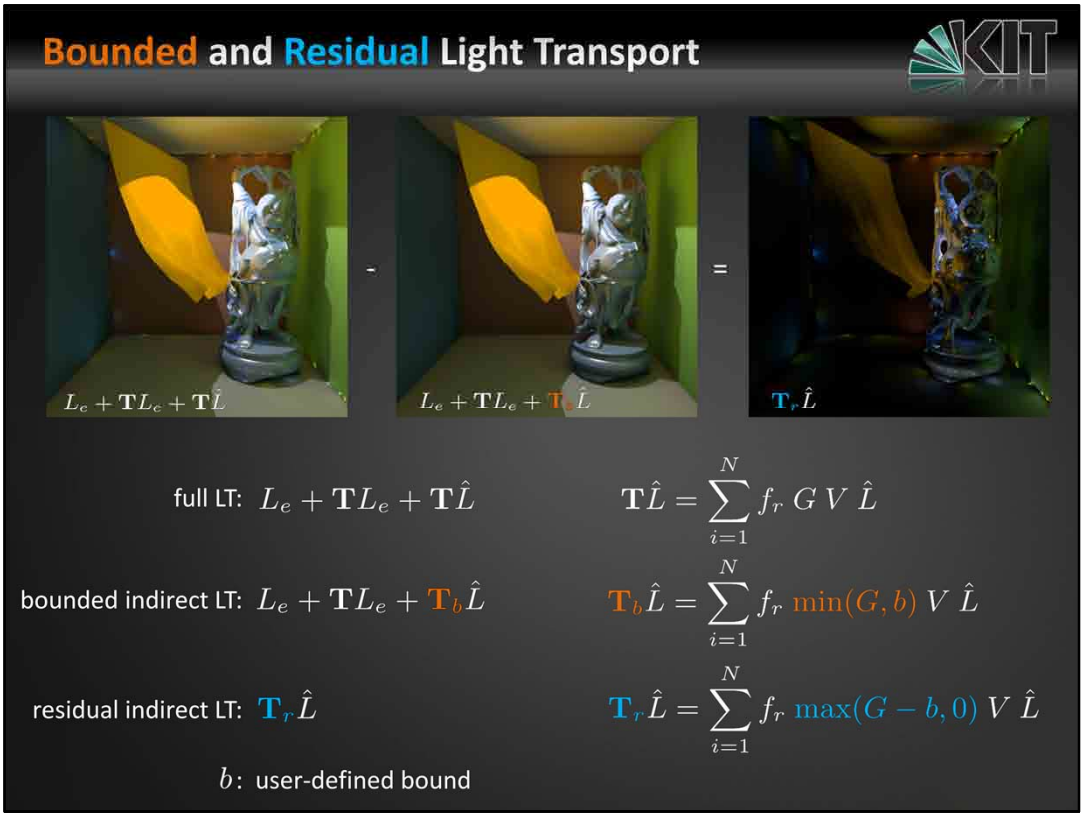


The naïve solution is to clamp the VPLs' contributions by clamping the geometry term … and seemingly the artifacts disappear.



But unfortunately, not only artifacts go away: we also remove a lot of energy from the light transport, i.e. we introduce a systematic error.

This loss of energy basically affects short distance light transport visible in concave regions (e.g. the corners of the room) where the geometry term becomes large.



Let's now have a closer look what's actually going on here.

The first line shows the rendering equation with full light transport computed from VPLs, which suffers from artifacts.

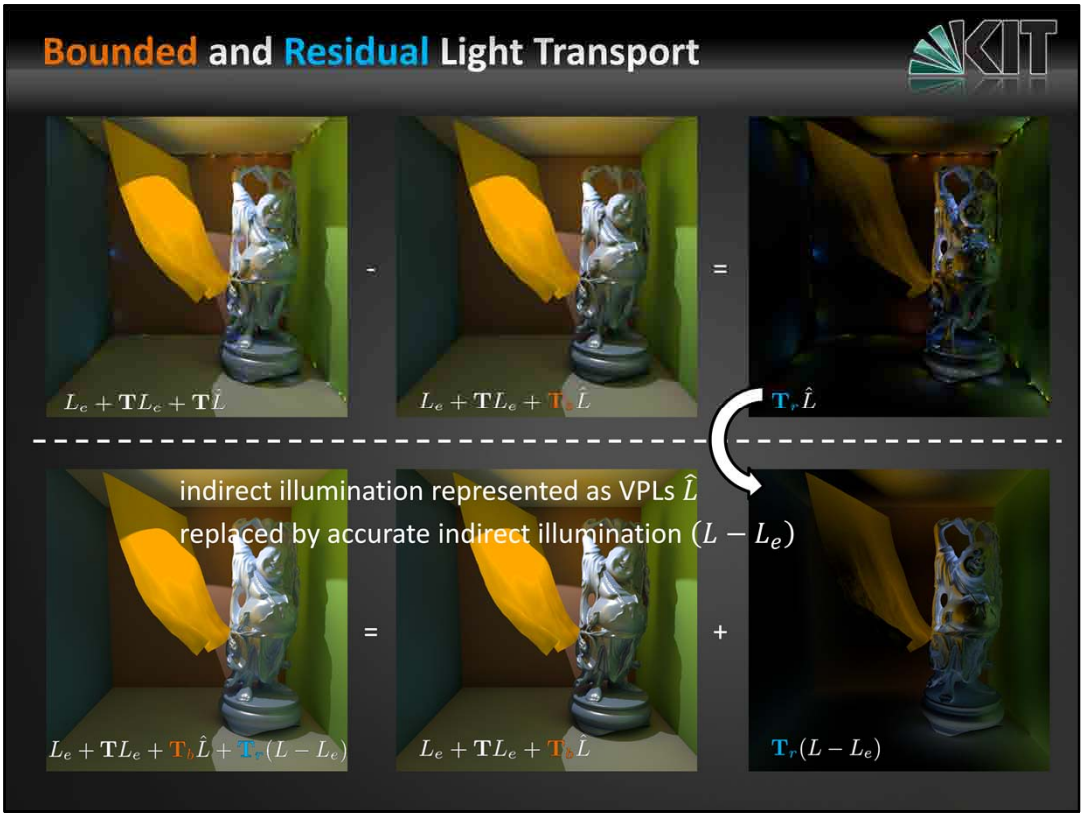
We can now define a transport operator T_b (for bounded transport) which clamps the VPL contribution when the geometry term becomes too large.

You can see that we clamp the geometry term to a user-defined bound b.

And we can also define a transport operator T_r (for residual transport) that describes the energy that we clamped away.

And in the top-right image, you can see that this is exactly the part of the lighting that contains the artifacts.

However, we don't want to remove the residual energy, but instead compute it without artifacts.

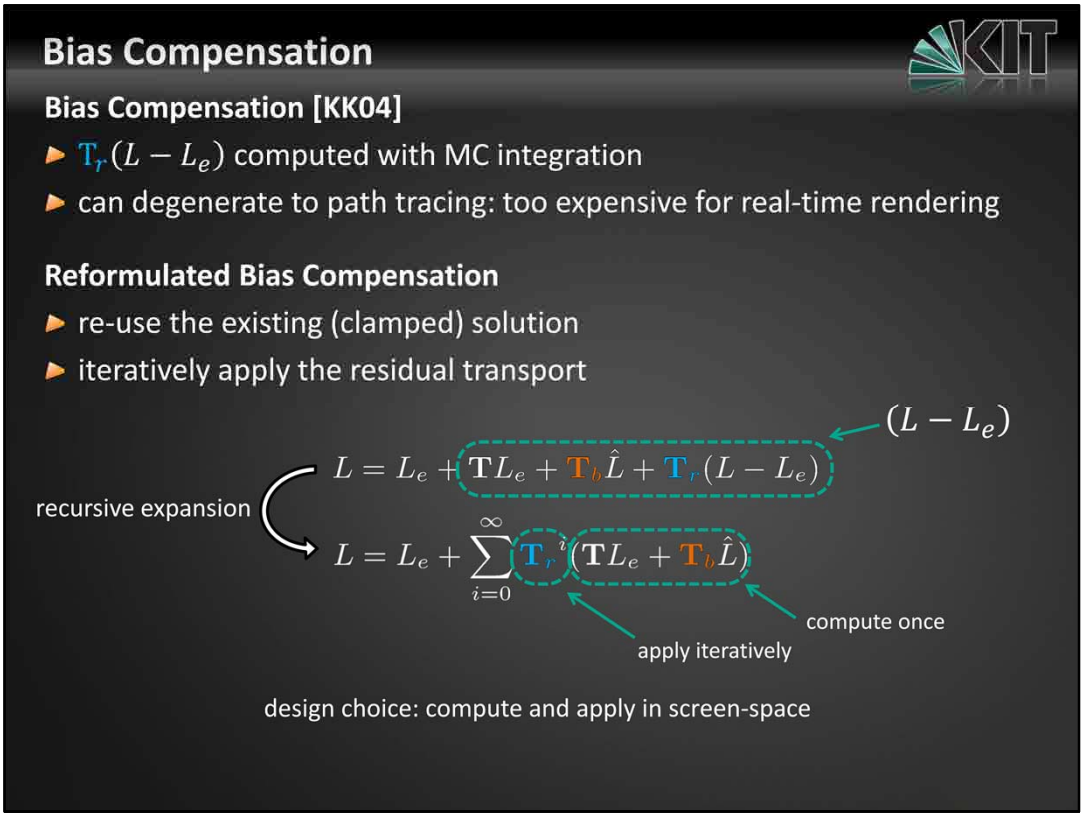


… and for this we need to do a smart reformulation…

We defined L-hat as the collection of VPLs that represent the indirect illumination.

The trick is to replace the approximation L-hat in the residual part by the accurate indirect illumination which is (L-Le) (i.e. all light except for direct emission).

If we add this accurate residual part to the bounded VPL contribution we obtain an artifact free and correct image.



Bias compensation as described by Kollig and Keller basically computes the residual part using Monte Carlo integration, which means that it can degenerate to path tracing in the worst case, and thus it is too computationally expensive for real-time applications.

Fortunately, our reformulation trick allows us to do something smart:

If you look at our rendering equation with the two transport operators, then we can observe that (L-Le) is exactly these 3 terms in the topmost green box.

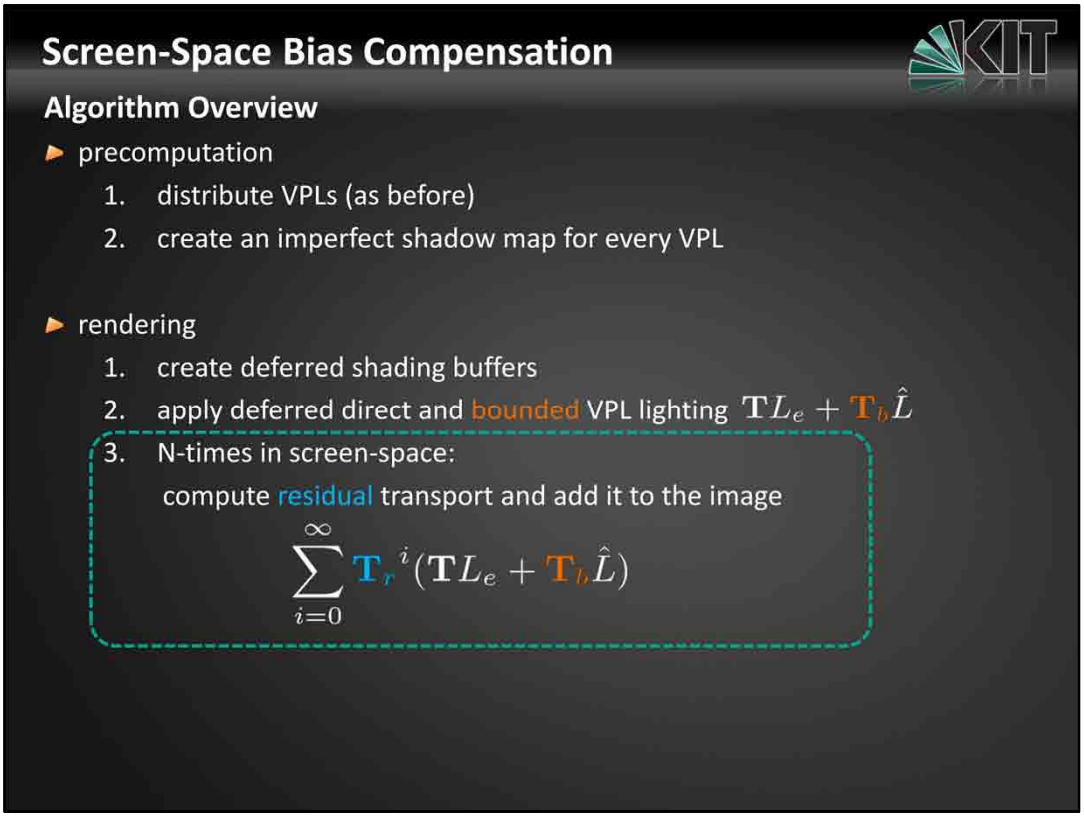
We can do a recursive expansion and we end up with this interesting result in the second equation.

Interesting because all summands in this version contain the same factor: direct light T Le plus bounded VPL lighting.

This means we can obtain all summands by applying the residual transport operator iteratively and that we can compute direct + bounded indirect light once, and recover the full light transport rom that.

However, we have to make a design choise: we need a basis to store the direct + bounded indirect light.

For real-time rendering we chose the image pixels as basis, i.e. we do the entire computation in screen-space.



This is the algorithm overview…

precomputation works as usual: VPL and shadow map generation.

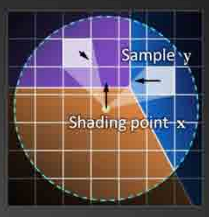
The image is then rendered with deferred shading with direct and bounded VPL lighting first, and only based on this information, we compute the residual transport.

Screen-Space Bias Compensation

Residual Transport Integration (1 iteration)

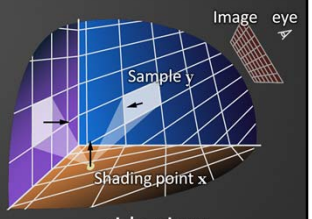
- **FOR EACH** pixel:
  - iterate over neighboring pixels
  - **IF** $G(\mathbf{x}\leftrightarrow\mathbf{y}) > b$
    - add contribution (with information in G-buffer)

$$G(\mathbf{x}\leftrightarrow\mathbf{y}) = \frac{\cos^+(\theta_\mathbf{x})\cos^+(\theta_\mathbf{y})}{\|\mathbf{x}-\mathbf{y}\|^2}$$



camera view

- clamping occurs in a close neighborhood only: close in world space = close in screen-space
- we can conservatively estimate a bounding radius and restrict the integration to it



side view

To compute one of these iterations, for every pixel in the image we compute the residual transport from all other surfaces.

Fortunately, the clamped away energy can only come from nearby pixels and we can estimate a bounding radius in screen space.

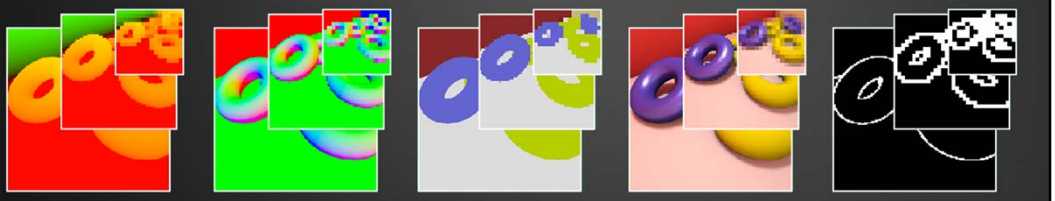
Then we go over all of these pixels, and see if the geometry term with those is larger than our bound, if it this is the case,
the we compute the transport using the information which is stored in the deferred shading buffers and the clamped image.

Screen-Space Bias Compensation

Hierarchical Integration

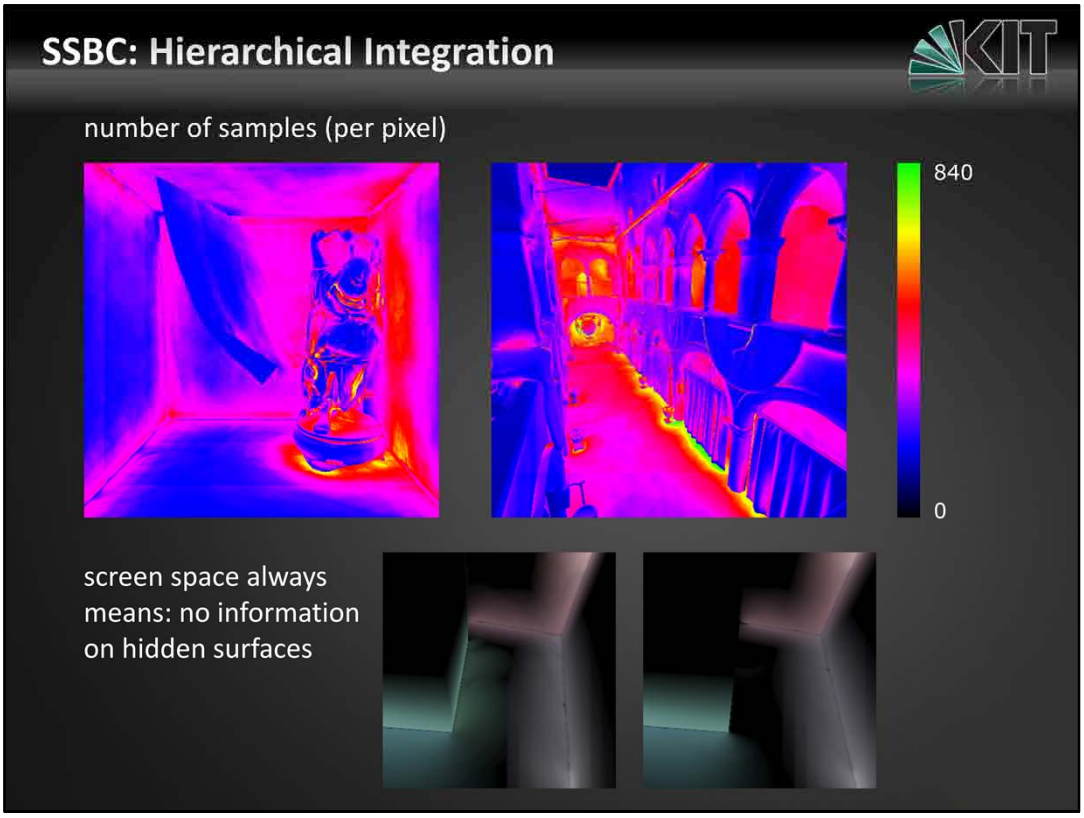
- still too many samples (even with the bounding radius)
- multi-resolution top-down integration (in spirit of [NW09])

- hierarchical approach requires
  - mip-map chain of the G-Buffer and bounded illumination
  - discontinuity buffer



deferred shading buffers | clamped solution | discontinuity buffer

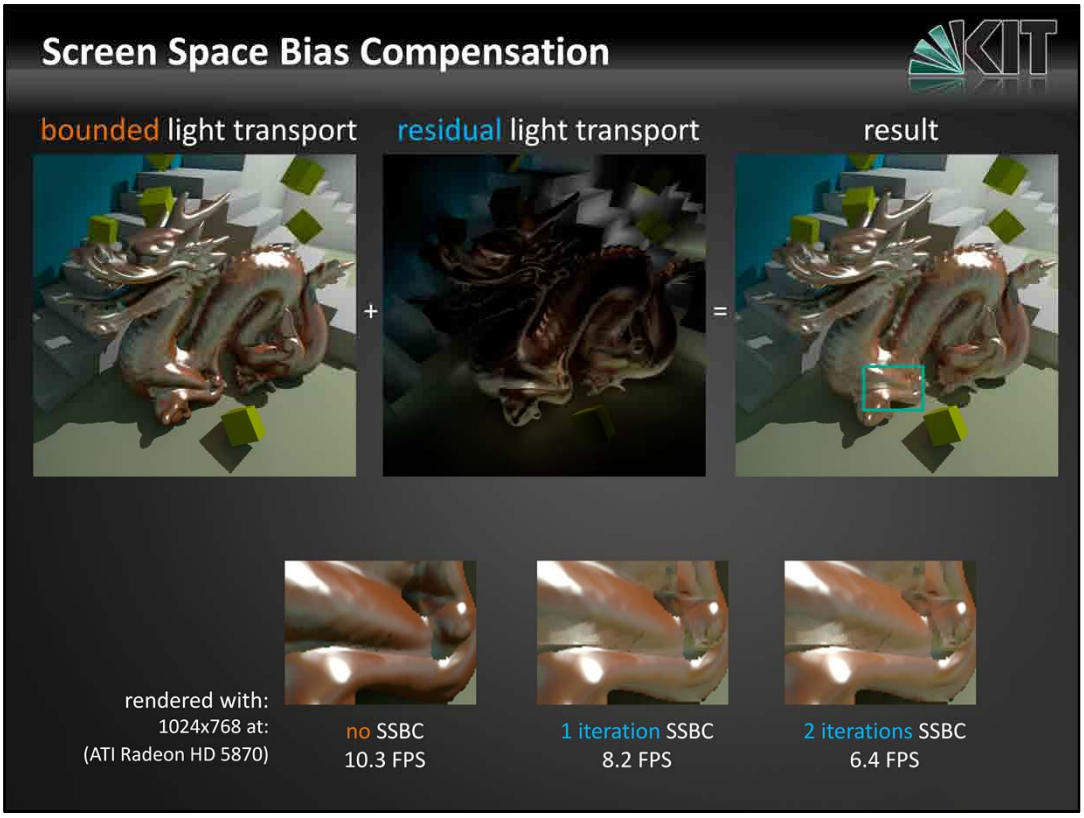
However, this integration would still be too slow and thus we used a hierarchical approach inspired by multi-resolution splatting idea of Nichols and Wyman, which works on a resolution pyramid of the G-buffer, the illumination and a discontinuity buffer.



You can see that the hierarchical integration computes the residual transport with an acceptable number of samples (i.e. queried pixels).

And as we would expect: more samples in corners and less, where there is nothing to compensate.

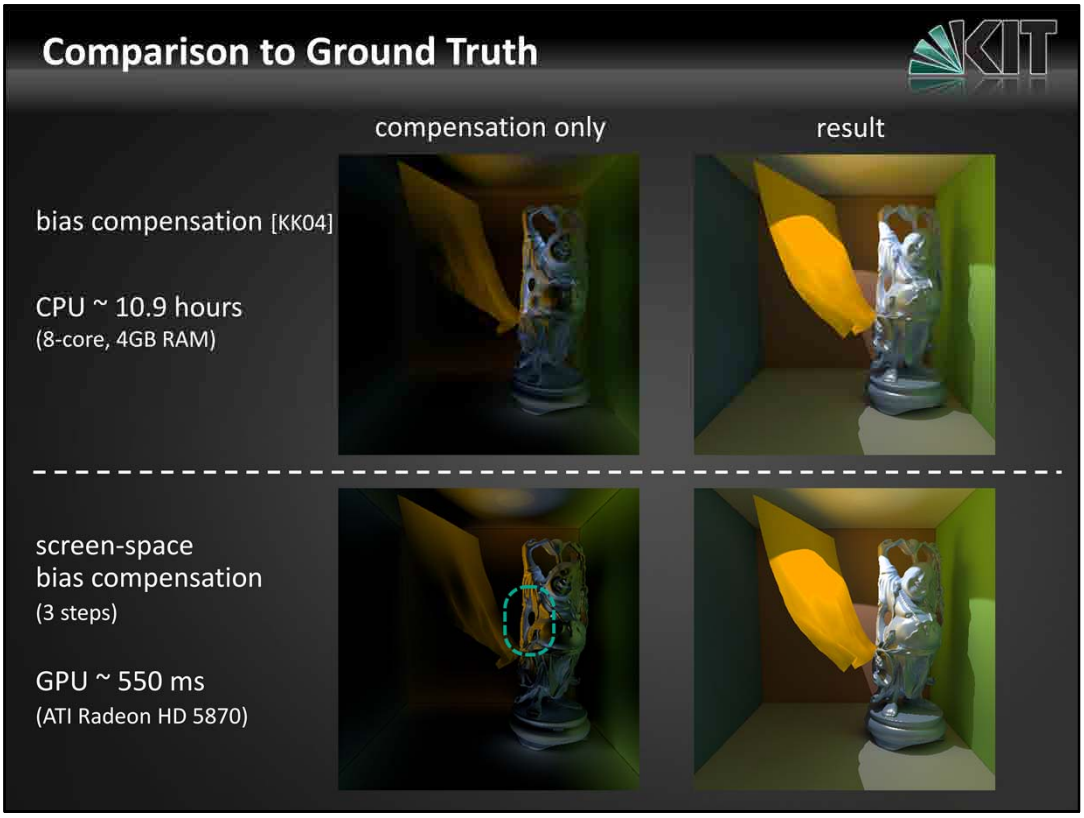
However, this design choice shares one problem with all screen space approaches: the image does not contain information about hidden surfaces, and thus the compensation from those is simply missing in the image.



Here you can see some results and two things are important here:

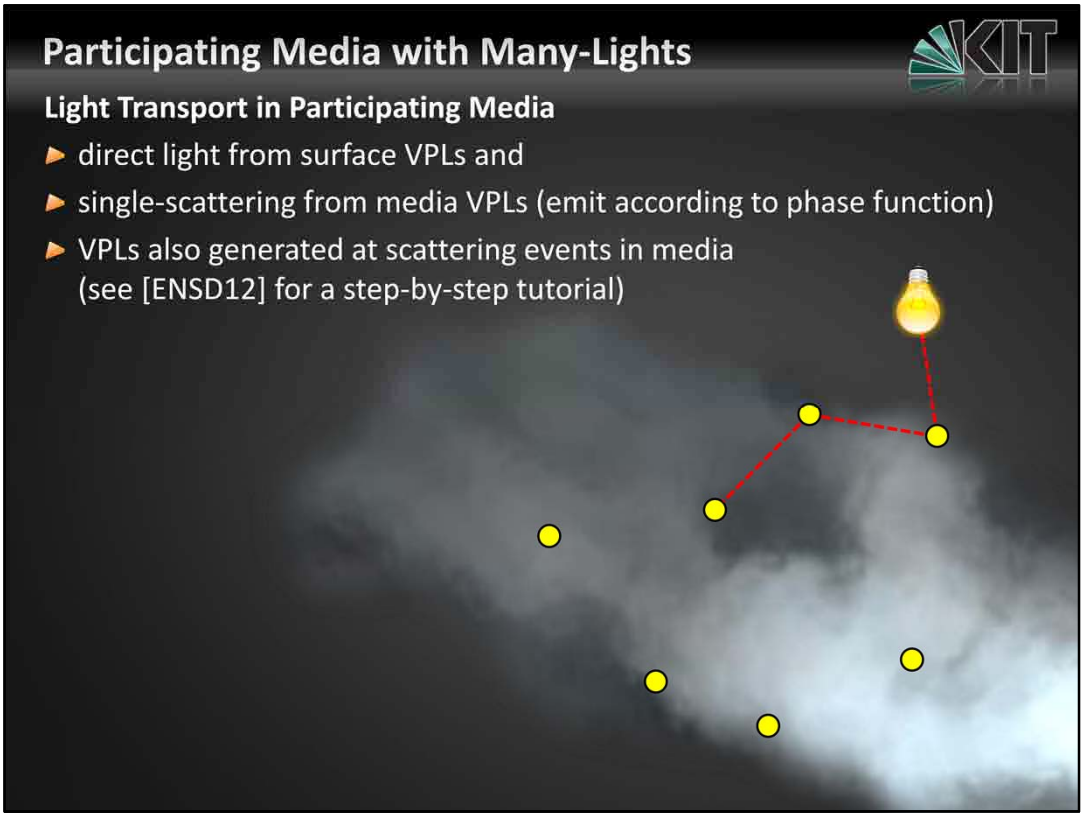
1) the first iteration adds a lot of energy, the 2nd significantly less. And more iterations are typically not really necesssary. This is because in every compensation step, we convolve with the BRDF

2) The screen space bias compensation is not very expensive compared to the clamped VPL rendering itself.



Well compared to ground-truth bias compensation it is of course orders of magnitude faster (and this is of course no fair comparison),

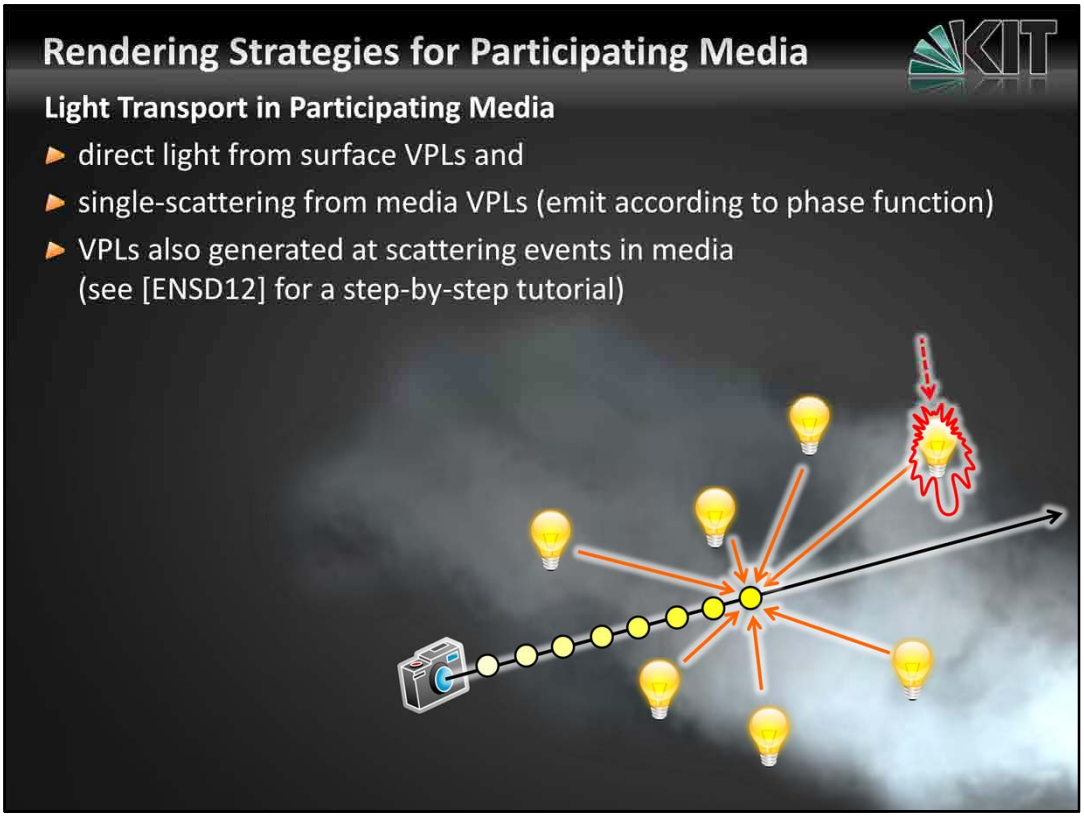
These images again shows a case where some surfaces appear too dark, simply because there is missing energy/compensation from hidden surfaces that are not sampled.



One nice thing about many lights rendering is that it directly transfers to participating media as well.

For surface transport, we created VPLs on surfaces and computed direct illumination thereof.

To account for scattering we create random walks which also generate VPLs where interactions in the media happen.

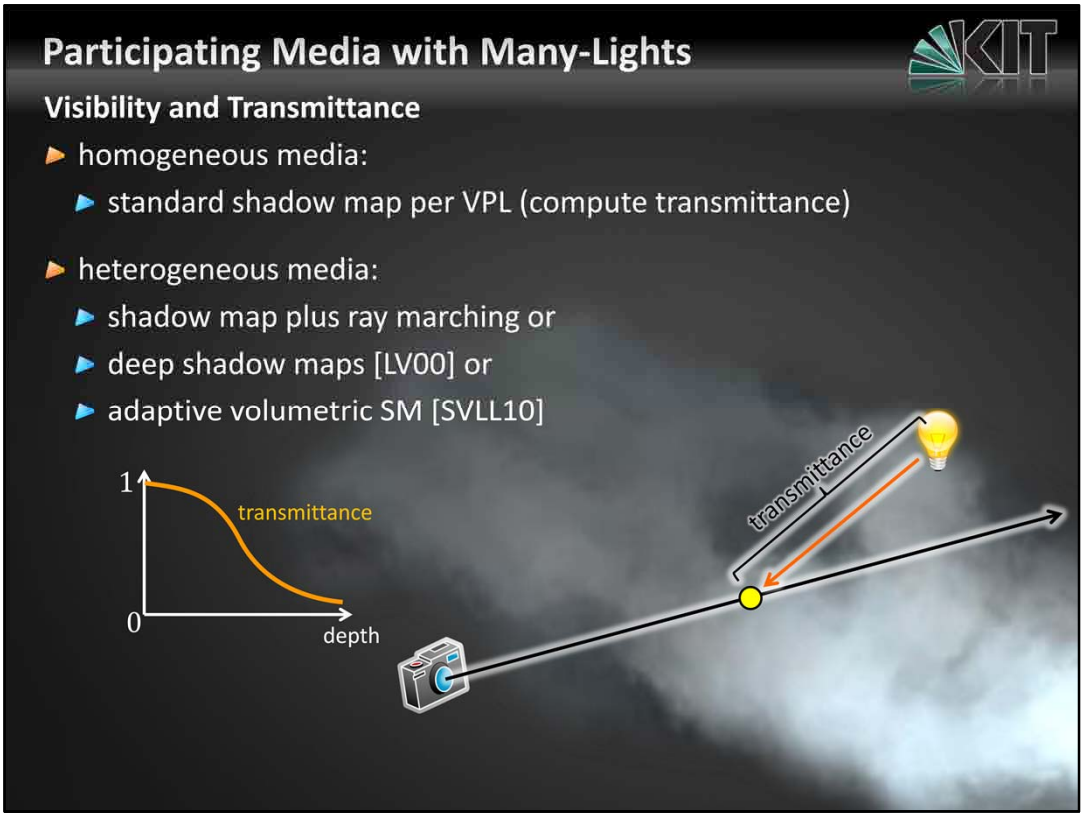


These VPLs emit light according to incident direction of the path (indicated by the small red arrow) and the phase function of the media.

We then have to compute the contribution due to single-scattering from these VPLs to a camera ray.

For this we typically ray march along the ray (to compute the transmittance along it) and connect to some or all VPLs at every step.

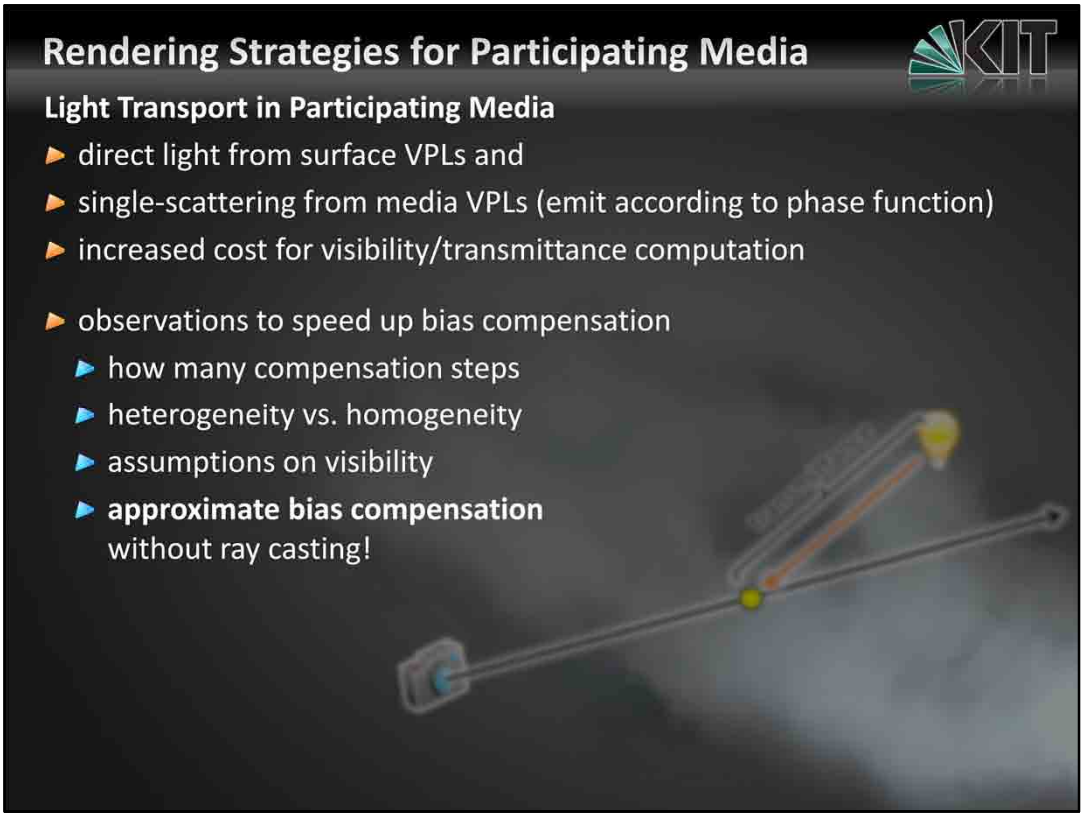
I strongly recommend to read [ENSD12] which explains the entire process and contains much more detail than this presentation.



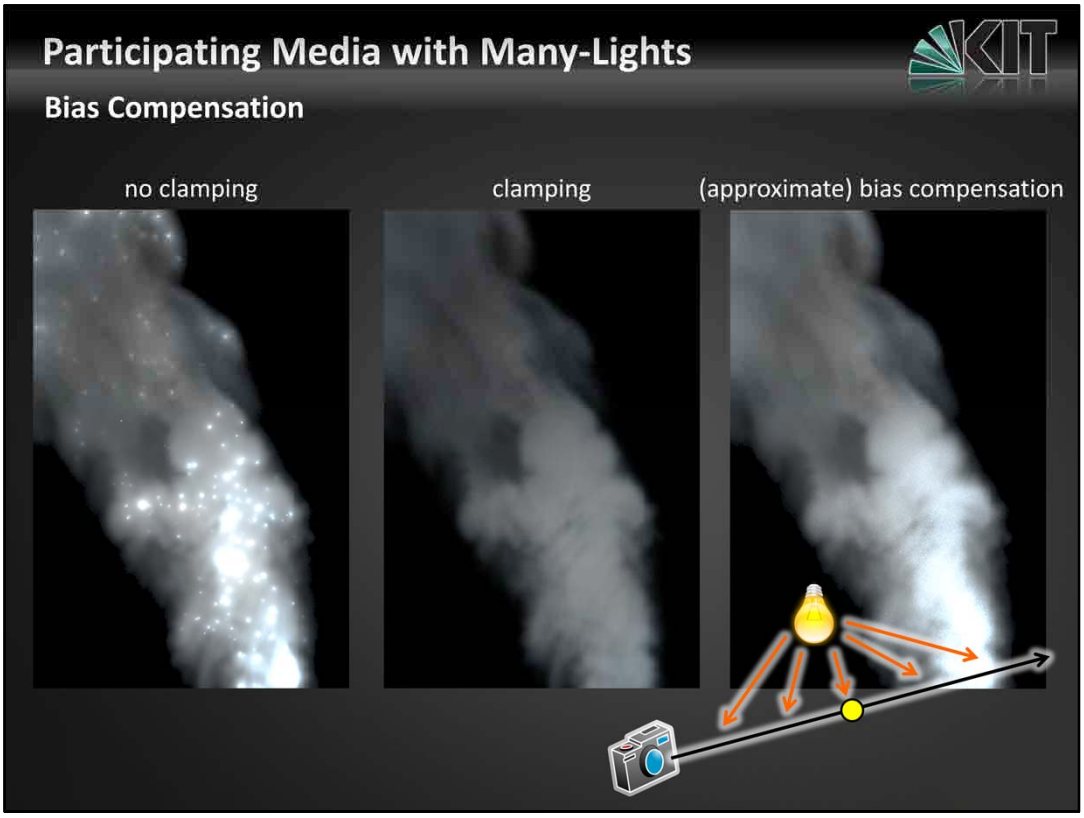
One important difference now is that in addition to binary visibility in vaccum we now have to account for outscattering and absorption along the connection of a VPL to a point on the camera ray.

For homogeneous media it is sufficient to create a standard shadow map which takes care of the binary part, and compute the transmittance along the paths analytically.

This, however, is not possible for heterogeneous media and has a strong impact on the render time:
We either have to ray march along every connecting path segment, or compute deep shadow maps, which essentially sample and store the transmittance function for all pixels in a shadow map. There is a recent hardware-friendly version (the adaptive volumetric SM) which can easily be used for our purpose.



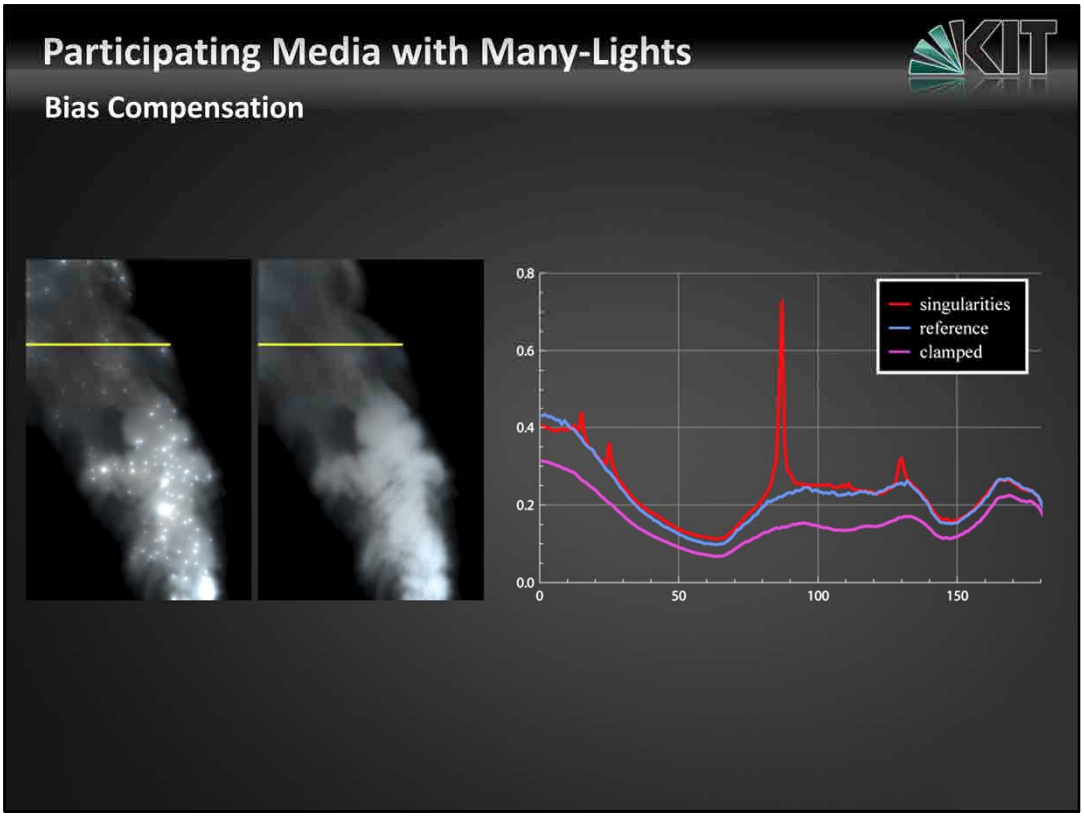
As we will see, clamping and artifacts are even worse in the case of participating media. Fortunately, we can derive an approximate bias compensation technique which does not require ray casting and is thus feasible on a GPU.



Here you see some smoke, and the bright splotches come from VPL lighting without clamping, whenever a VPL is too close to the camera ray.

The clamped image looks ok-ish, but we removed a lot of energy in this case.

On the right side you can see the result with our bias compensation for media, which can be even be done in interactive speed.



Here‘s a plot along a scan line in this image illustrating the radiance values of the pixels in a clamped, non-clamped, and correct solution.

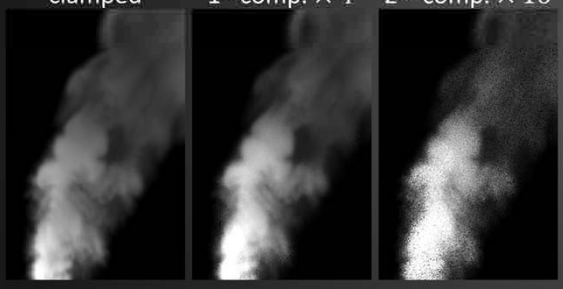
You can see how bright these splotches actually are.

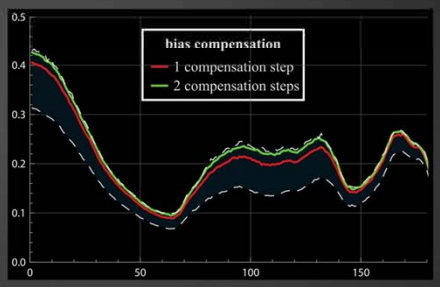
Participating Media with Many-Lights

Bias Compensation

- classic bias compensation [RSK08] if prohibitively expensive
- similar to surface case: magnitudes of compensation steps drop quickly

clamped | 1st comp. × 4 | 2nd comp. × 16





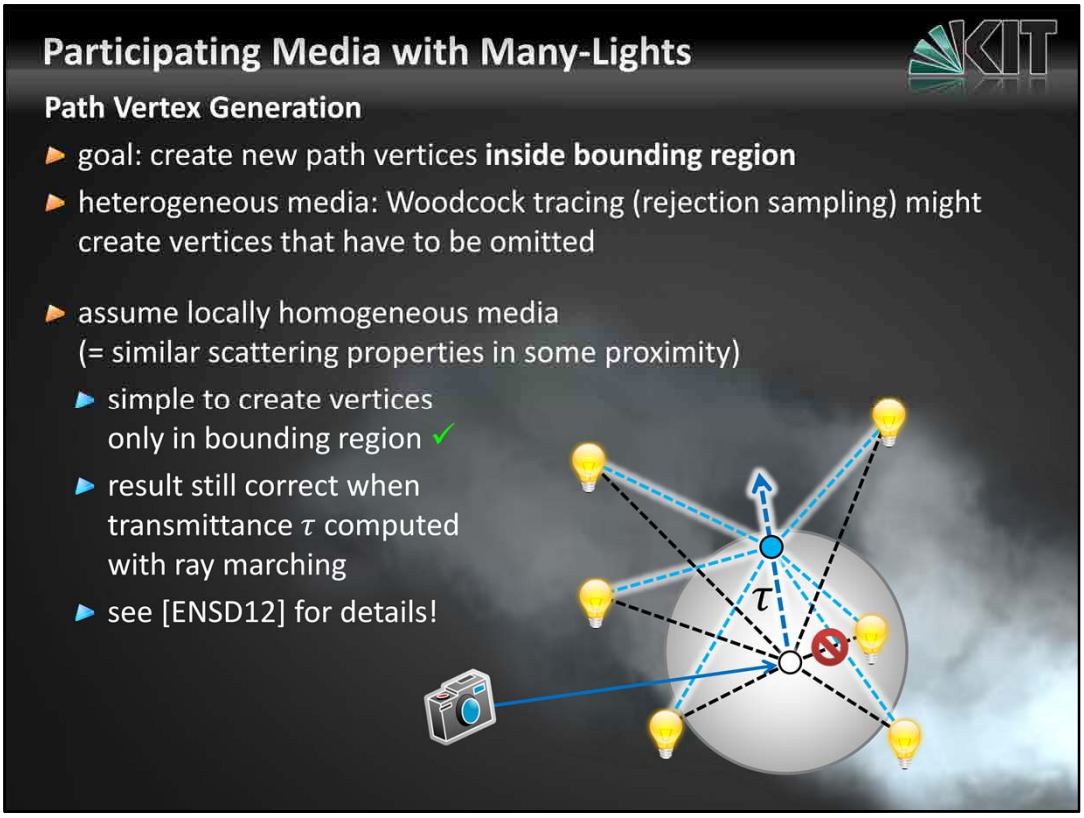
computed with path tracing (Raab et al.'s method)

As before, we could compensate the bias introduced by clamping using Monte Carlo integration, but this is incredibly expensive for participating media – remember it degenerates to path tracing, and path tracing in participating media is a very bad idea.

So we started looking at the bias compensation to figure out if there is potential good approximations and to find out how Raab et al's method has to be modified to be feasible in interactive speed (but the observations are also beneficial for offline rendering, see [ENSD12]).

The first aspect we were looking at is the recovered energy which drops fast with every iteration.

We found that it drops so quickly that 2 iterations were always enough (as long as we do not have highly anisotropic phase functions, but then many lights methods would not be a good idea anyway)



Another difficulty appears when doing bias compensation in heterogeneous media:

This figure is meant to explain Raab et al.'s bias compensation:

In this example one VPL is too close to our shading point and what bias compensation does is it omitts the VPL, and we have to sample a new direction to create a new path vertex and a scattering event along this direction (direction is easy, simply sample the phase function).

To figure out where this scattering event happens in an unbiased way, one typically uses Wookcock tracking (which is essentially a rejection sampling technique – and thus a bad idea by definition).

The problem is that it can generate locations which are outside the bounding region and thus have to be omitted and then we would sample again.

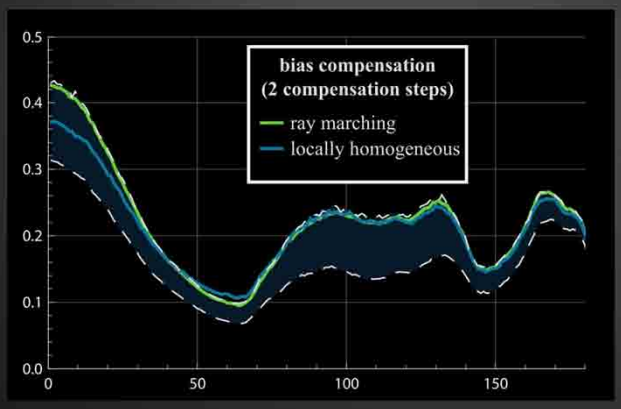
Once we created a new path vertices in the bounding region, we connect it to the VPLs and sum up the contributions. And of course, a recursive omission of VPLs can happen.

For a GPU-friendly algorithm we don't want to omit vertices, and to do so, we assume a locally homogeneous media in the bounding region for sampling the distance (i.e. we take the average extinction coefficients there) because allows us to sample scattering events in that region without the need of rejection sampling. Note that the result is still correct if we compute the true transmittance along the segments.

Participating Media with Many-Lights

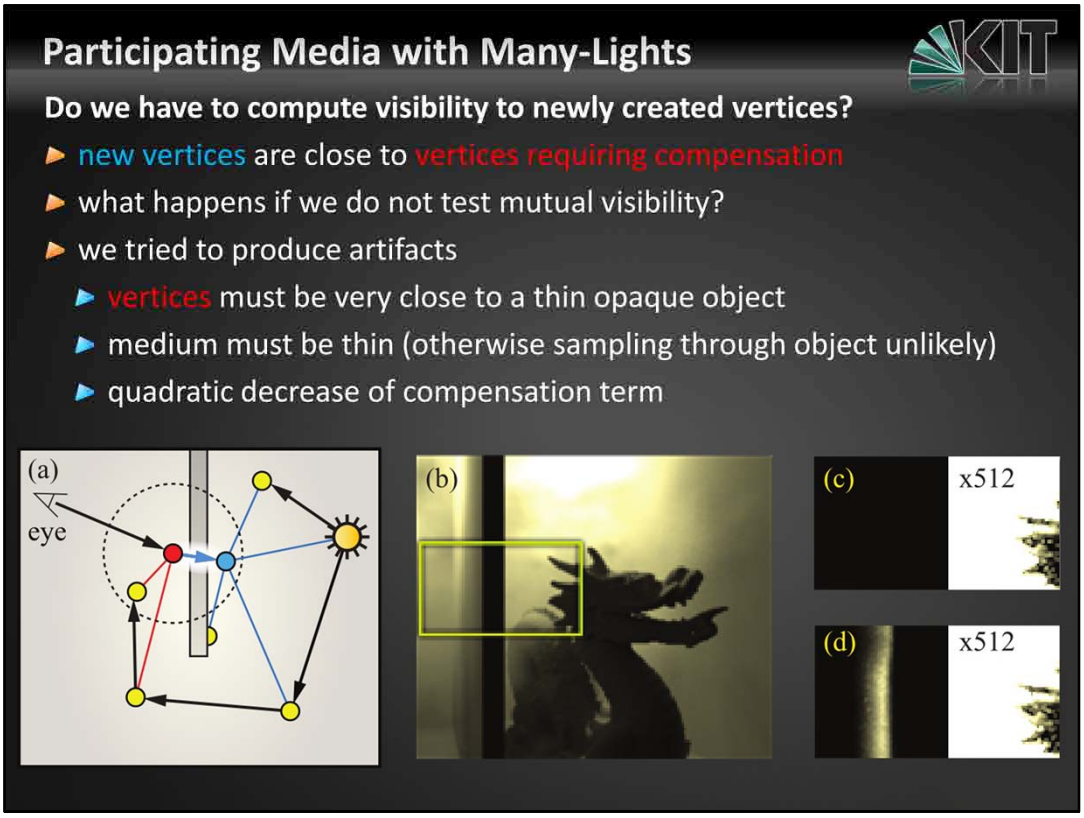
Path Vertex Generation

- assume media to be locally homogeneous
  - simple to create vertices only in bounding region ✓
- also compute transmittance using averaged scattering coefficients
  - not correct but very close



We also tried to compute the transmittance analytically from the locally homogenous medium (and thus avoid ray marching).

Still the differences to the exact solution are quite small.



And there‘s another very interesting aspect in bias compensation for participating media:

strictly speaking, when creating a new path vertex (blue) for a location that requires compensation (red) we have to account not only for transmittance, but also for visibility.

We tried what happens if we always assume mutual visibility (i.e. do not check for it).

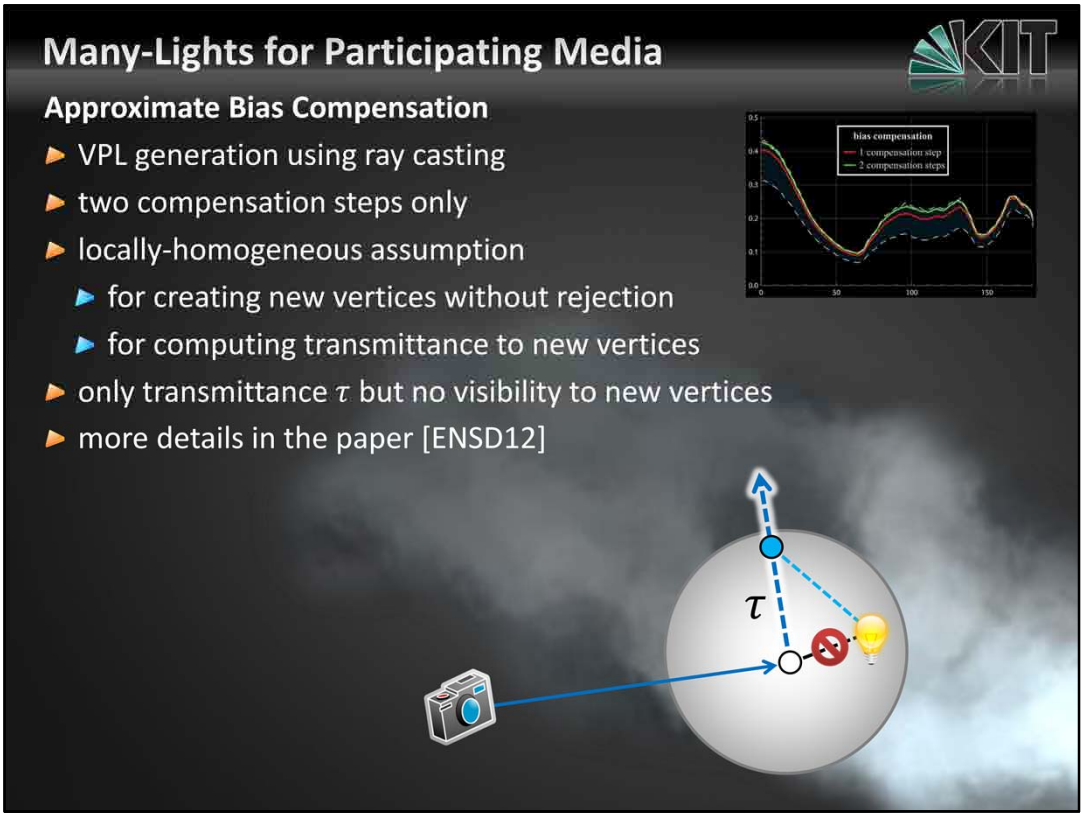
Interesting is that we had to try hard to produce visible artifacts. The reasons are as follows:

- first of the location that requires compensation has to be close to a surface (closer than the bounding radius) otherwise occlusion can‘t happen
- second: the medium must not be too thick, otherwise sampling larger distances is unlikely
- and lastly there‘s also a quadratic decrease of the compensation term with the distance

Finally we succeeded: we create a scene consisting of two rooms full of smoke and separated by a wall.

We turned of the light in one room and after scaling the brightness by a factor of 512 we discovered the splotch on the bottom right!

We learned that artifacts from omitting visibility in participating media bias compensation are basically invisible.



This method is not fundamentally different from Raab et al's method, but it uses some approximations with make it much fast and suited for GPUs.

In our implementation we used ray casting for VPL generation, although you could come up with some "deep reflective shadow maps" to do that on the GPU as well.

This slide shows an offline rendering taken from [ENSD12] with 118k VPLs.

Please also have a look at the video which can be found on our webpage:
http://cg.ibds.kit.edu

Important to note is that a significant portion of the render time in the interactive/GPU version for participating media is spent on ray marching along camera rays, which is also required for single scattering from the primary light source, i.e. this is cost you have to pay even without many-lights rendering…

Conclusions

Famous Last Words…

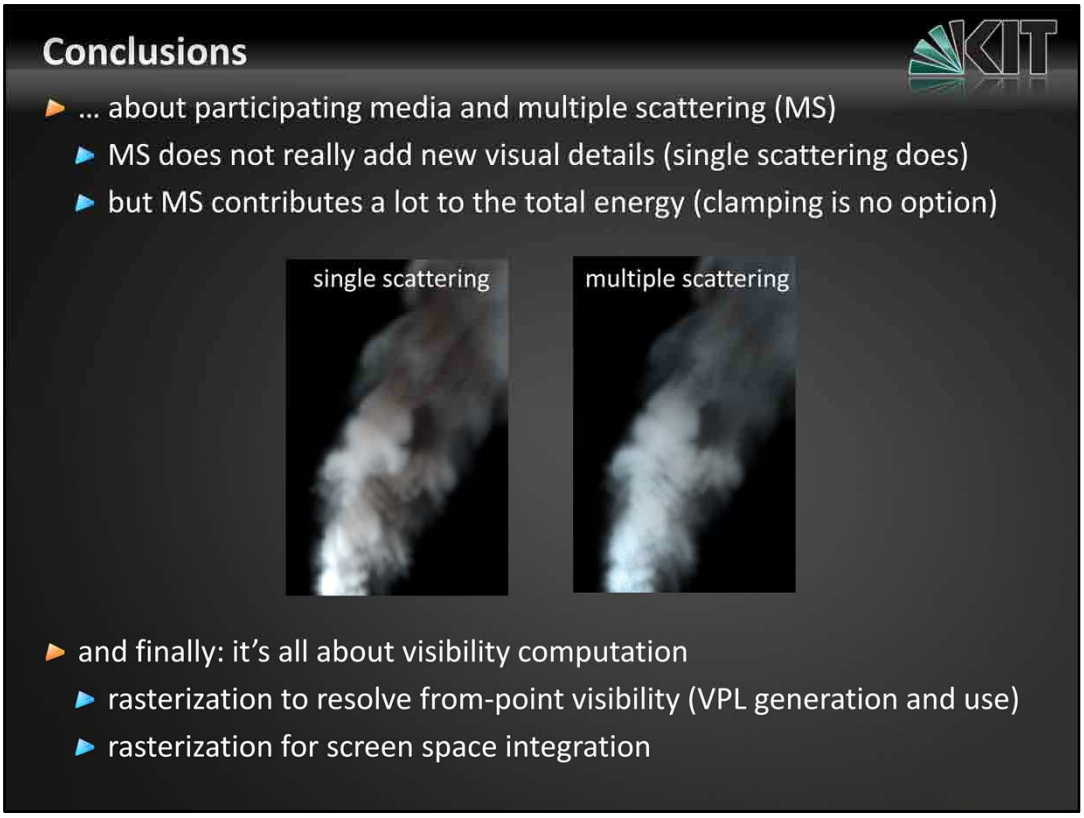
- many-lights methods work quite well in real-time
  - bias compensation is feasible for surfaces and media
  - glossiness for surfaces ↔ anisotropic phase functions for media
  - for mostly diffuse scenes, for scenes with moderate anisotropic media

isotropic | moderate anisotropic | strong anisotropic

I hope to have you convinced that many lights methods are suitable for real-time rendering, also with bias compensation for high quality rendering.

Their use is rather limited by the materials in a scene, where glossy surfaces require more VPLs to capture the light transport accurately.

As glossy BRDFs for surface, highly anisotropic phase functions cause problems when rendering participating media, as you can guess from the images on this slide.



What we also found – which is not surprising actually – that multiple scattering does not really add new crispy visual details, and this is why is can be well rendered using a moderate number of VPLs.

Note, VPLs for participating media make sense for thin media. Thick media would create many interactions and thus many VPLs and is not sufficient in this case.

The detail in thin media comes from computing transmittance along the camera ray and single scattering into the direction of the camera.

However, multiple scattering can contribute a lot to the overall energy, i.e. it should not be left out, and clamping should be avoided.

And to conclude with what this presentation started -- it's all about visibility computation:

We've seen that rasterization can be efficiently used to resolve or approximate the visibility between a moderate number of points (the VPL locations) and the surfaces in our scene,

And it is also possible to replace ray casting in bias compensation by screen space integration or to use well-working approximations/well-justified assumptions for bias compensation in participating media.

Optimizing Realistic Rendering with Many-Light Methods

Real-Time Many-Light Rendering

Acknowledgements:
Some slides on SSBC have been created by Jan Novak. Tobias Ritschel provided images for ISM/MR.

References

- [LV00] Lokovic and Veach, Deep Shadow Maps, SIGGRAPH 2000
- [KK04] Kollig and Keller, Illumination in the Presence of Weak Singularities, 2004
- [DS05] Dachsbacher and Stamminger, Reflective Shadow Maps, I3D 2005
- [Seg06] Segovia et al., Non-interleaved Deferred Shading of Interleaved Sample Patterns, GH 2006
- [DS06] Dachsbacher and Stamminger, Splatting of Indirect Illumination, I3D 2006
- [LSKLA07] Laine et al., Incremental Instant Radiosity for Real-Time Indirect Illumination, EGSR 2007
- [RSK08] Raab et al., Unbiased global illumination with participating media, Monte Carlo and Quasi-Monte Carlo Methods, 2008
- [RGKSDK09] Ritschel et al., Imperfect Shadow Maps for Efficient Computation of Indirect Illumination, SIGGRAPH Asia 2008
- [SW09] Segovia and Wald, Screen Space Spherical Harmonics Filters for Instant Global Illumination, TechReport Intel, 2009
- [ED10] Engelhardt and Dachsbacher, Epipolar Sampling for Shadows and Crepuscular Rays in Participating Media with Single Scattering, I3D 2010
- [REGSKD10] Ritschel et al., Micro-Rendering for Scalable, Parallel Final Gathering, SIGGRAPH Asia 2009
- [SS10] Schwarz, Seidel, Fast Parallel Surface and Solid Voxelization on GPUs, SIGGRAPH Asia 2010
- [NW10] Nichols and Wyman, Interactive Indirect Illumination Using Adaptive Multiresolution Splatting, IEEE Transactions on Visualization and Computer Graphics 16(5), 2010
- [SVLL10] Salvi et al., Adaptive Volumetric Shadow Maps, EGSR 2010
- [NED11] Novak et al., Screen-Space Bias Compensation for Interactive High-Quality Global Illumination with Virtual Point Lights, I3D 2011
- [NNDJ12a] Novak et al., Virtual Ray Lights for Rendering Scenes with Participating Media, SIGGRAPH 2012
- [NNDJ12b] Novak et al., Progressive Virtual Beam Lights, EGSR 2012
- [ENSD12] Engelhardt et al., Approximate Bias Compensation for Rendering Scenes with Heterogeneous Participating Media, Pacific Graphics 2012
- [OBA12] Olsson et al., Clustered Deferred and Forward Shading, High Performance Graphics 2012