March 28th, 2013

Real-Time Volumetric Shadows

using 1D Min-Max Mipmaps

By: Bart van Greevenbroek
Simon Rosman

Overview

- Problem Description
- Problem Solution
- Volumetric Shadows
- Method Description
- Epipolar Rectification
- 1D Min-Max Mipmaps
- Strenghts & Drawbacks
- Conclusion
- Questions

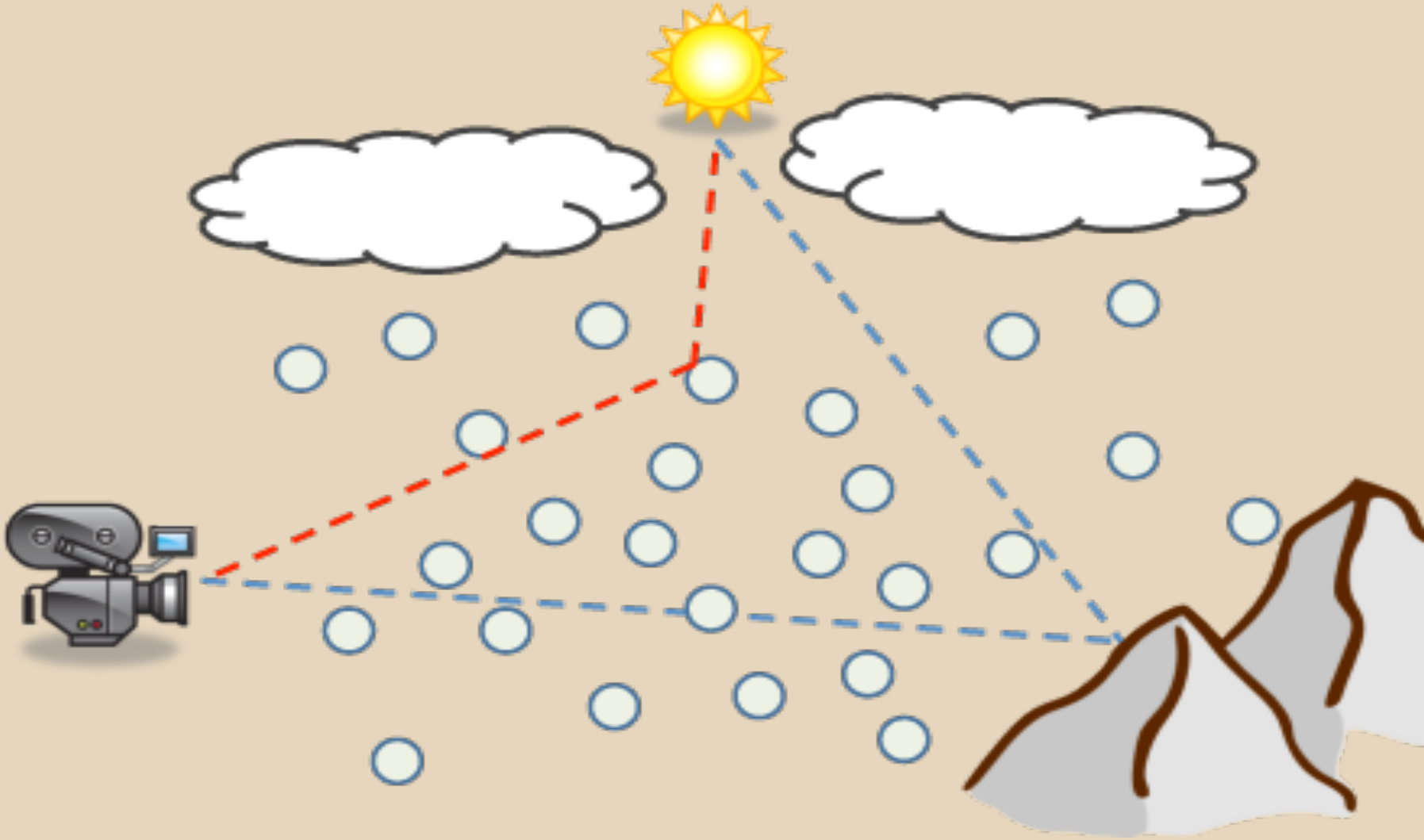
Problem Description

How do we approximate the scattering integrals efficiently to render crepuscular (God) - rays in real time?

Problem solution

- Ignore complex light transport techniques (caustics, multiple scattering and transparent blockers)
- Treat every ray shot through each pixel independently and in parallel
- Use an efficient data structure (1D min-max mipmap) to speed up computation of the scattering integral

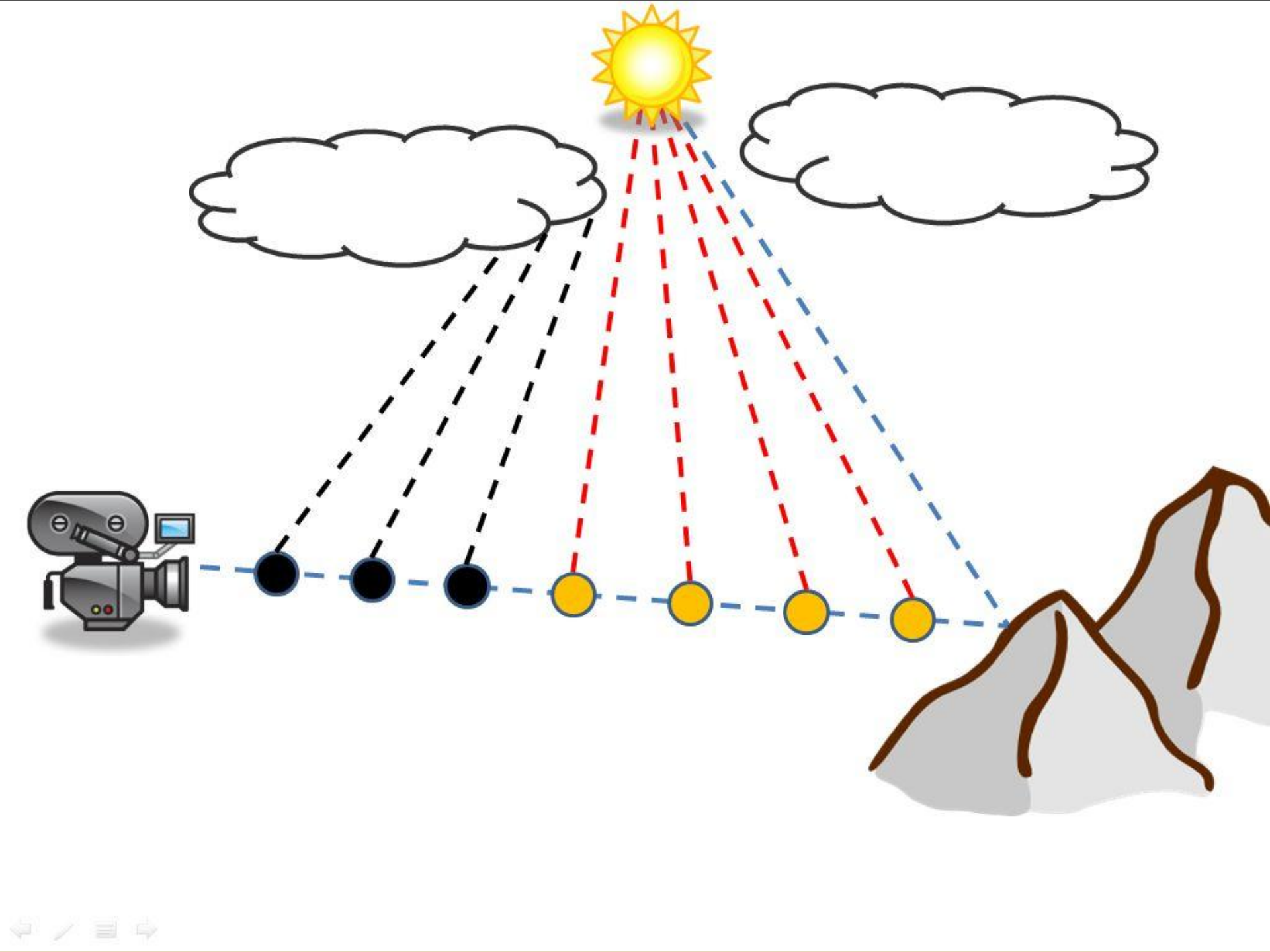
Scattering Light



Ray Marching

Recall from the 3rd lecture (slide 65):

- Sample along the ray
- Blend the results



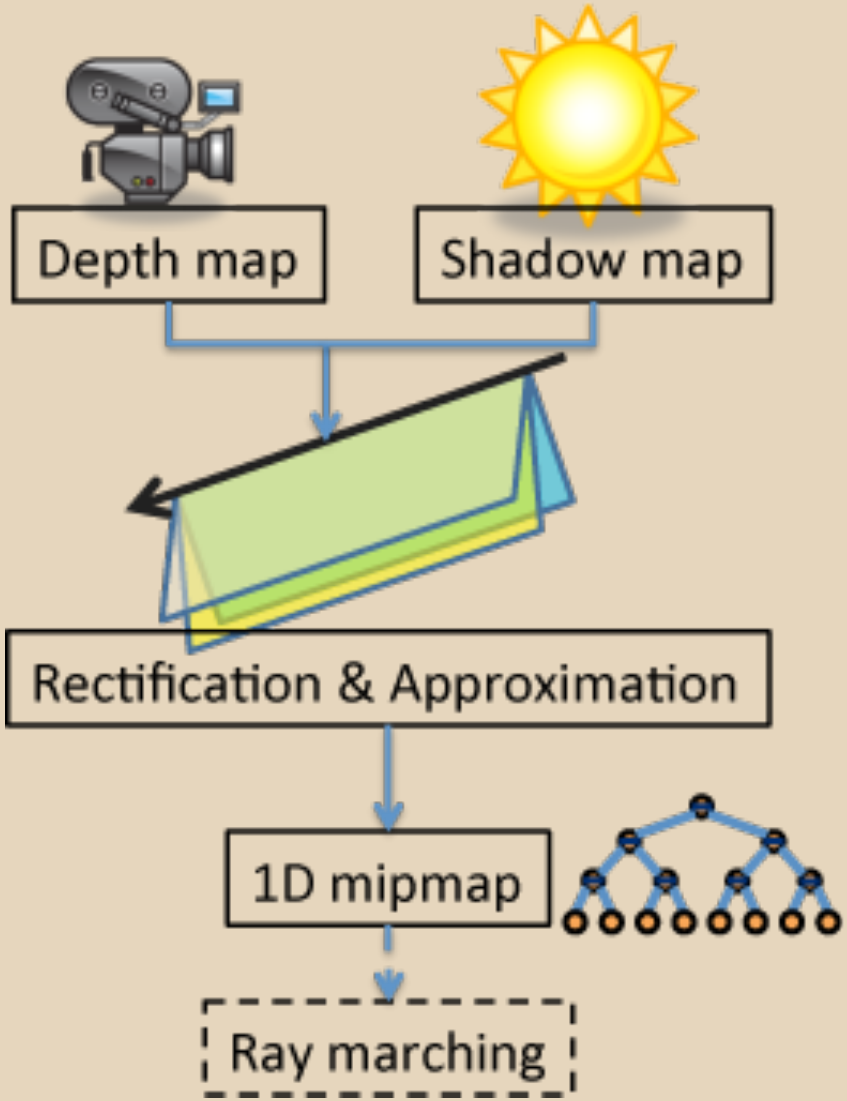
Single scattering using ray marching

- From the camera eye, cast a ray through each pixel
- For each of those rays, sample points along the ray (ray marching)
- For each sample, check if it is lit or unlit using a shadowmap
- Needs alot of samples to get a good scattering approximation

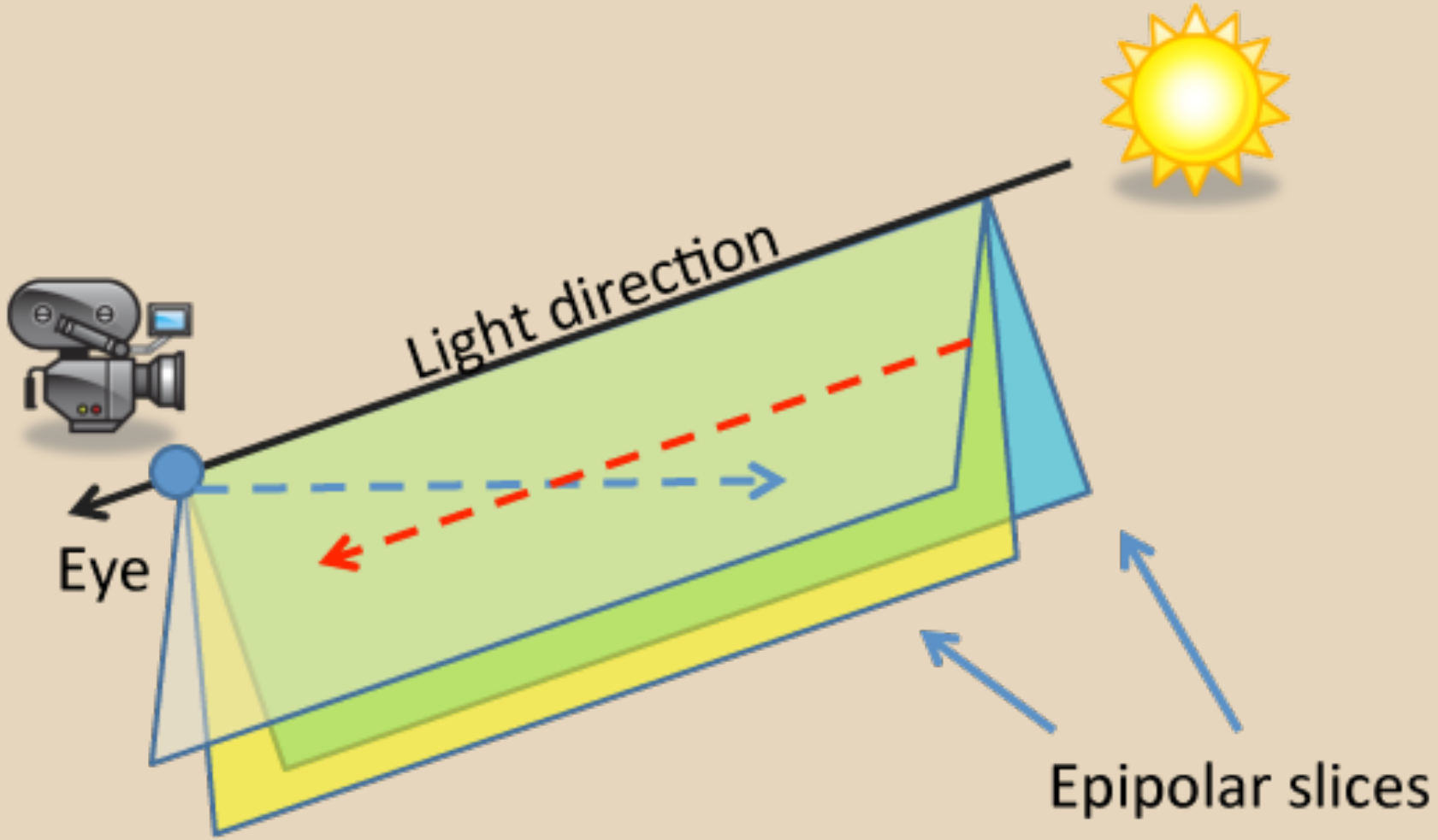
Method

Per frame:

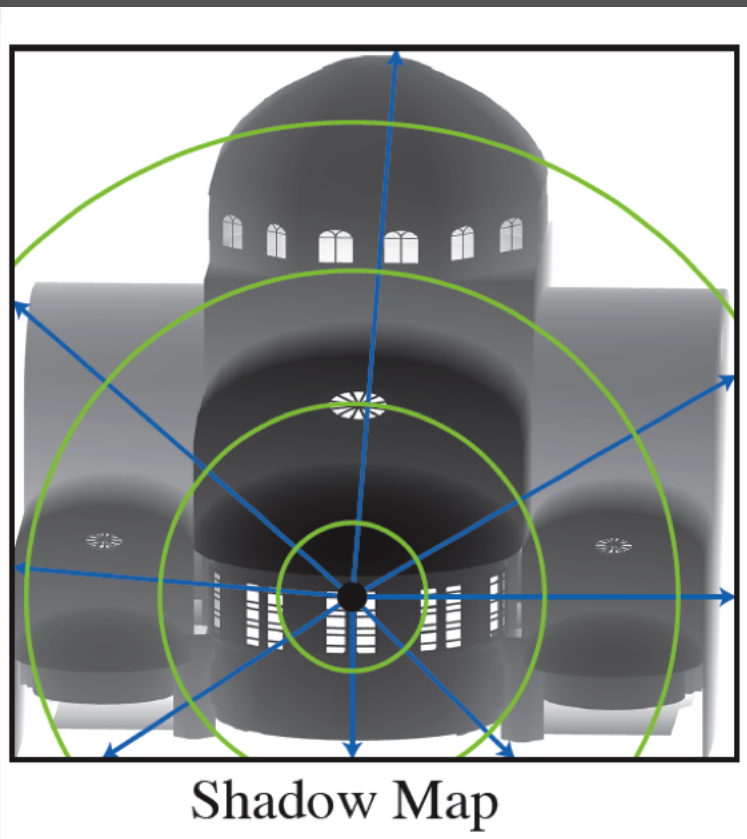
1. Render depth map + shadow map
2. Perform epipolar rectification
3. Compute low-rank approximation
4. Compute 1D min-max mipmap
5. (Compute scattering)



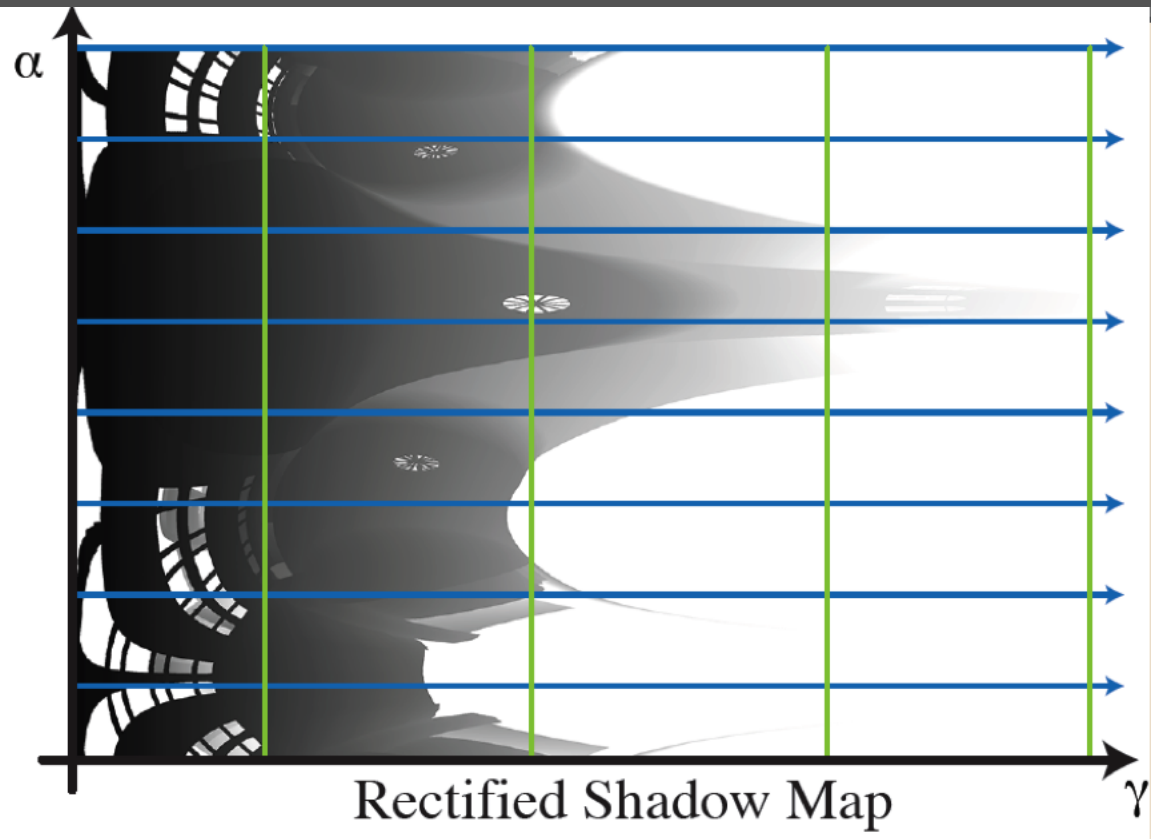
Retification & Approximation 1



Retification & Approximation 2



Shadow Map

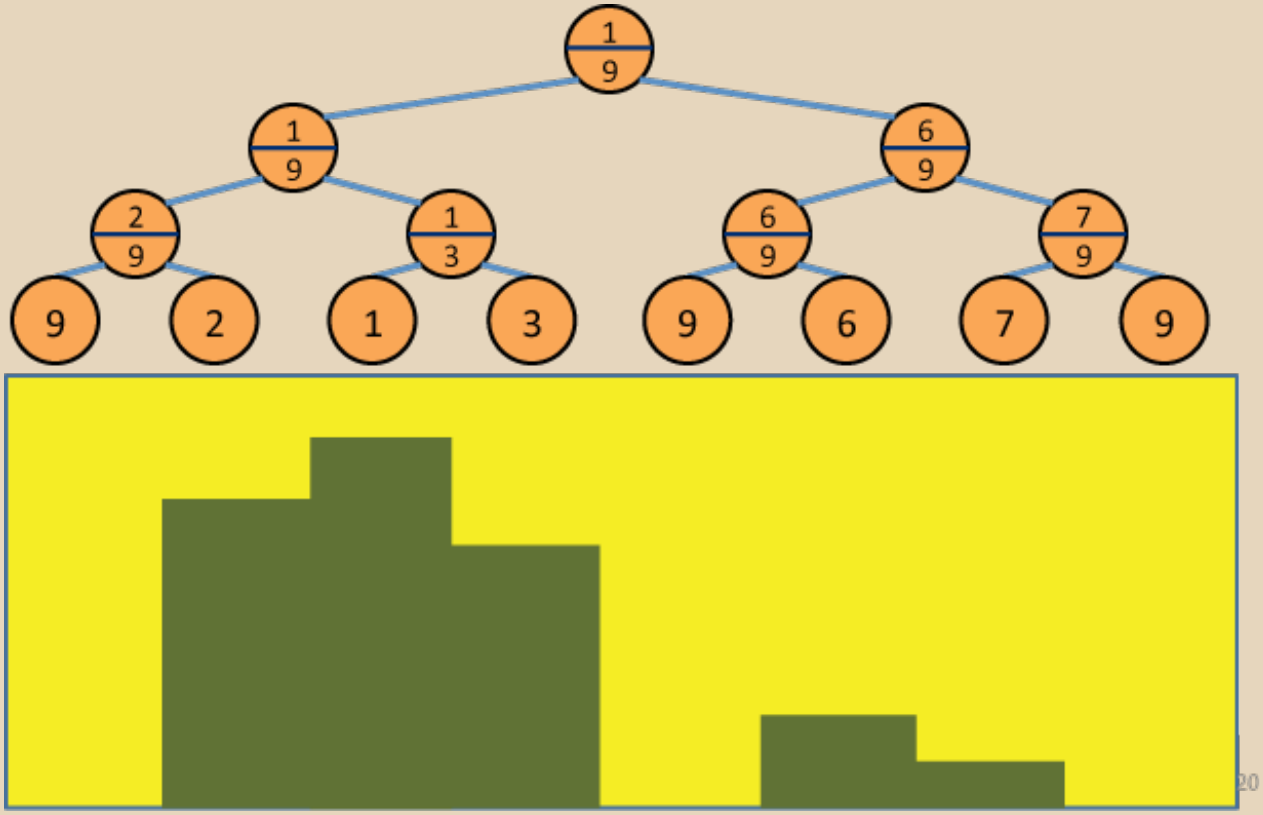


Rectified Shadow Map

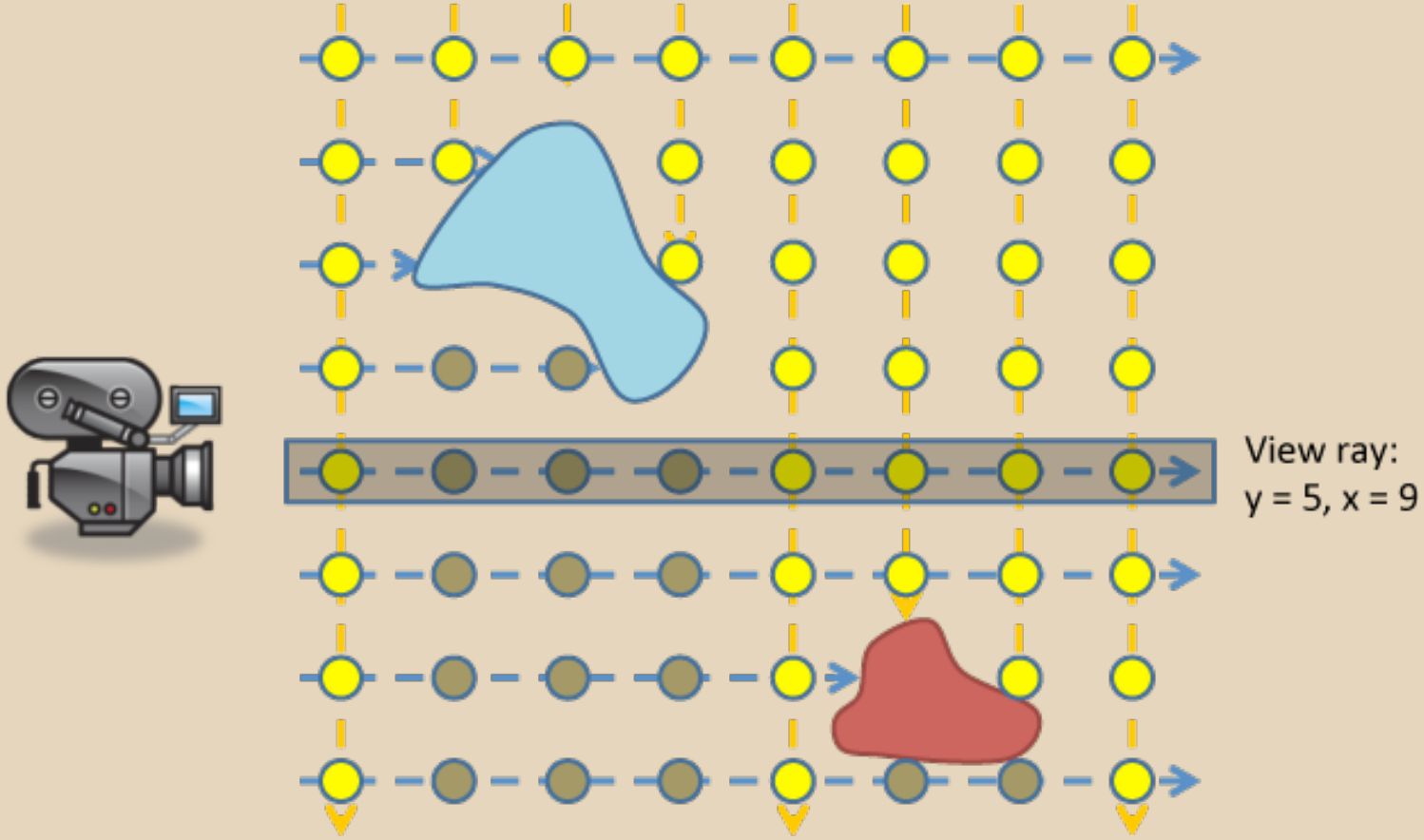
- Partition world space in slices
- Planes contain the eye, and are parallel to the light

1D min-max mipmap - 1

- Basically a BVH, but only in 1D
- 1 mipmap for each row in the shadowmap



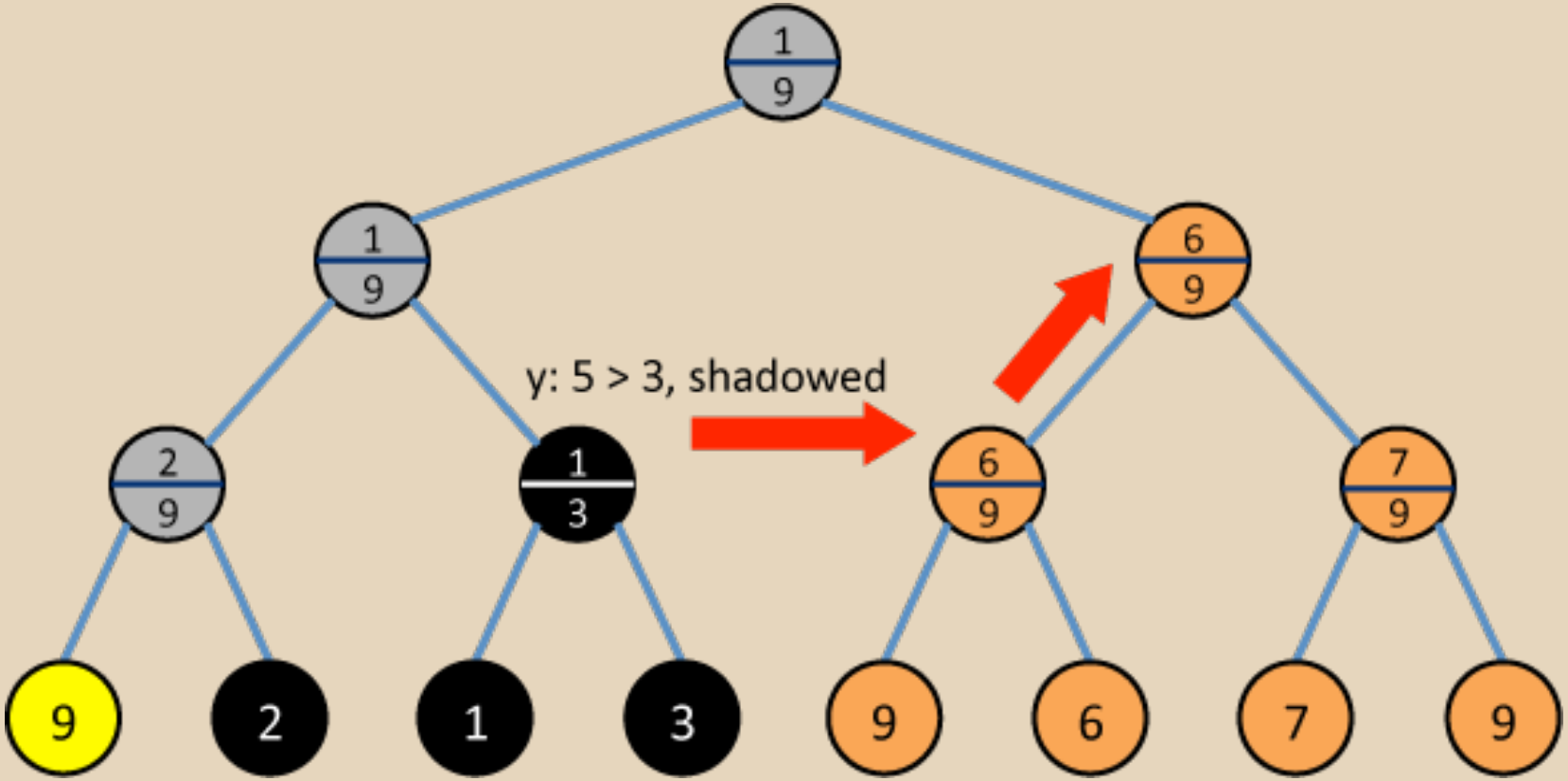
1D min-max mipmap - 2



Horizontal: Depth map

Vertical: Shadow map

1D min-max mipmap - 3



- Travelsal for each camera ray
- Implemented in pixel shader, but without recursion

Compute scattering

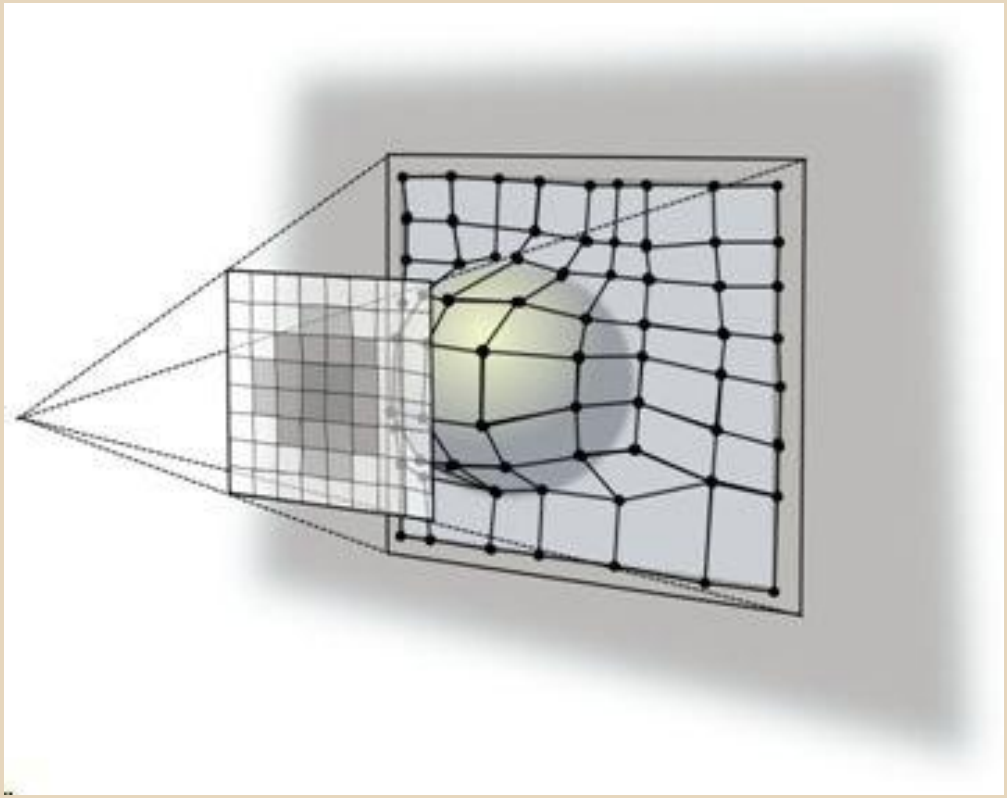
- For scattering around epipolar center, use Ray-marching
- Blend with other lighting (direct, GI)
- Textured light
  - Plot coordinates in the rectified light texture

Related work

Bileter et al.[2010]

- Interpret shadow map as a 2D heightfield
- Rasterize height field and accumulate the scattering integral analytically
- Introduces Vertex and fragment
- overhead
- Fast for low resolution shadow maps, scales badly with higher resolution shadow maps

Related Work

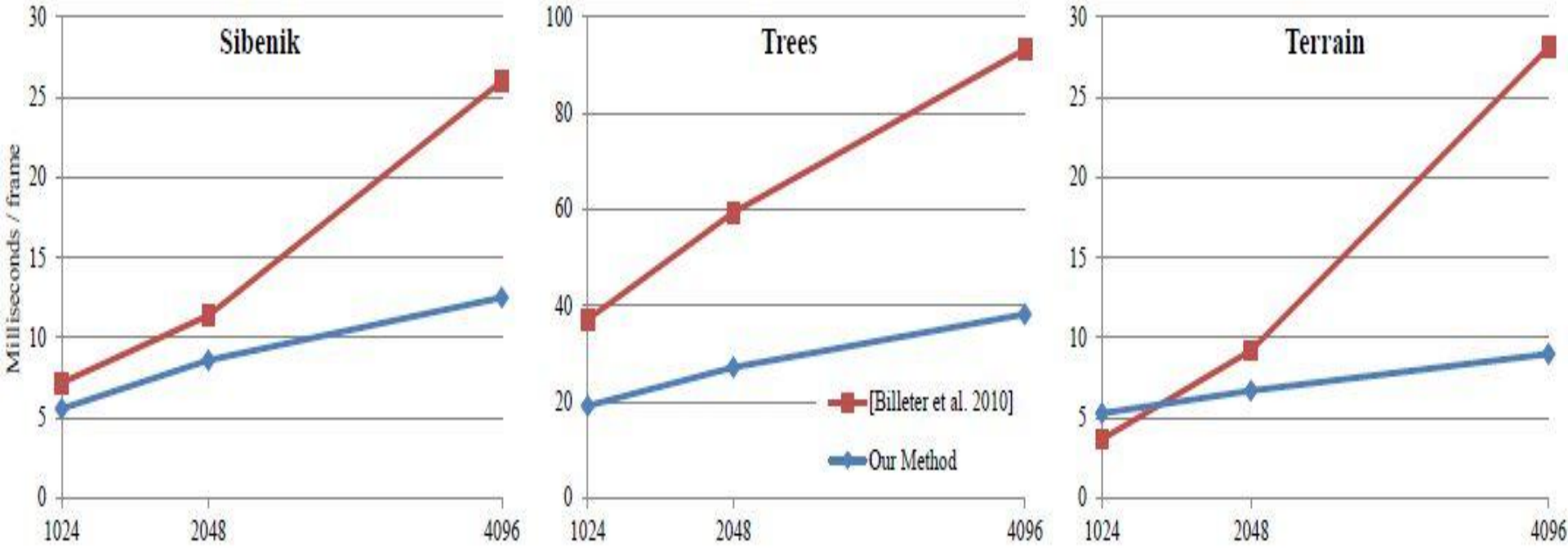


Results

Video

http://groups.csail.mit.edu/graphics/mmvs/mmvs.mp4

Results



	Sibenik			Trees			Terrain		
	1K	2K	4K	1K	2K	4K	1K	2K	4K
Our method	5.6	8.6	12.5	19.2	27.2	38.2	5.3	6.7	9.0
[Billeter et al. 2010]	7.2	11.4	26.0	37.2	59.4	93.2	3.7	9.2	28.1
Our speedup	1.3×	1.3×	2.1×	1.9×	2.2×	2.4×	0.7×	1.4×	3.1×

Strenghts

- Avoids the need for a specific order of ray sample traversal -> more parallellism
- Avoids the need for CUDA or OpenCL, making this technique viable for current game consoles that don't support GPGPU API's
- Supports textured lights, meaning God rays coming through stained glass ("glas in lood")

Strengths (2)

- Scales well with high-resolution shadow maps because of the 1D min-max mipmap data structure
- Uses pixel shaders only

Drawbacks

Flickering or Temporal Aliasing - As a polygon get rasterised differently over time, the jump is propagated through the scattering integation. This is solved using a higher resolution shadowmap, with which this method scales well with.

Drawbacks (2)

The real time results (45-70 fps in the church scene) are achieved using the method in combination with direct illumination only. Is the method still real time using contemporary global illumination techniques?

Conclusion

Fast technique for rendering volumetric shadows, using technology found in current gen consoles

However, artifacts and aliasing still occur unless high resolution shadow maps are used

Questions?