Basic Shadow and Reflection Techniques in Real-Time

Shadow Maps and Shadow Volumes

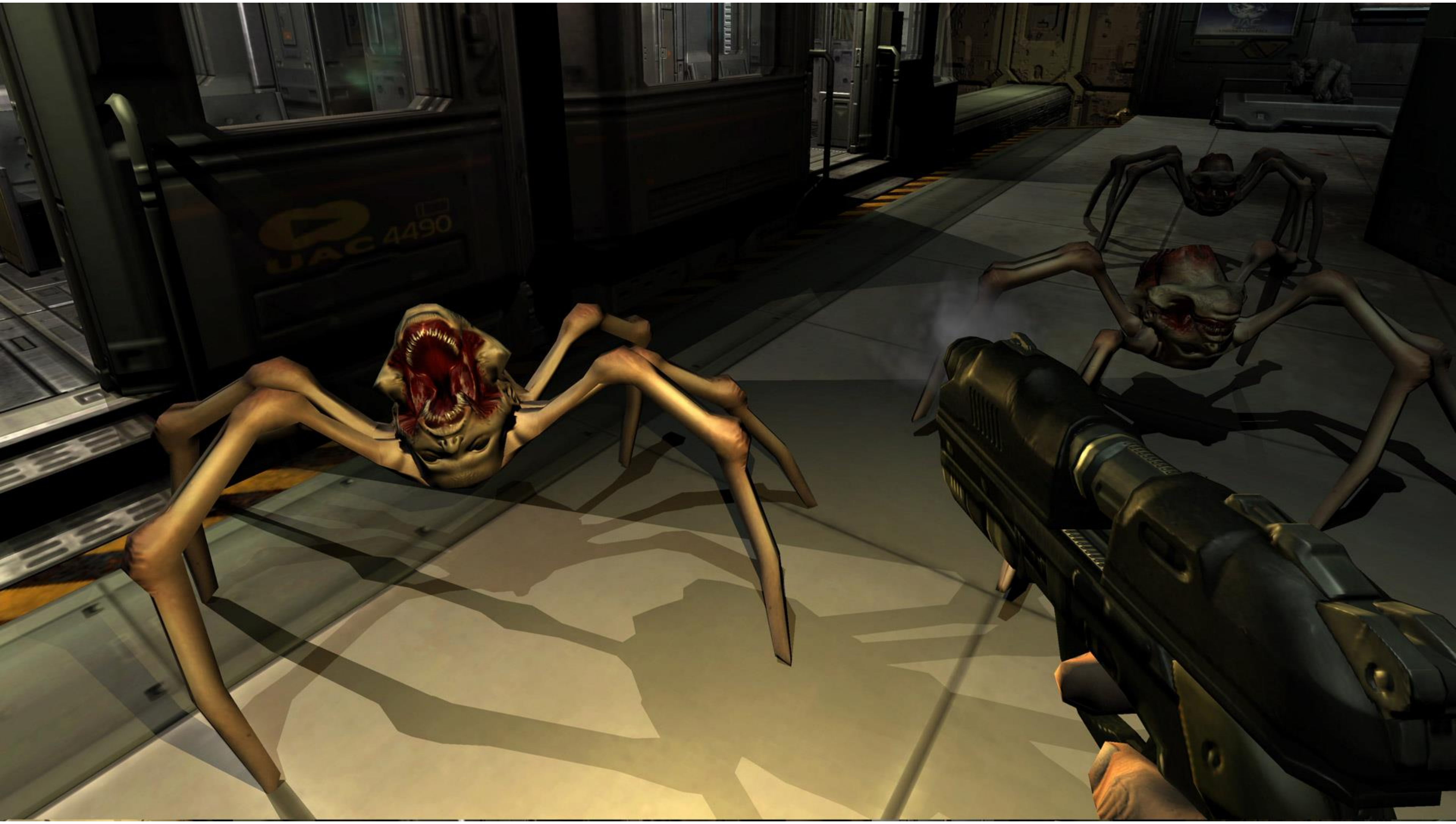
Ulf Assarsson

Why shadows?

- More realism and atmosphere

Doom Eternal – shadow maps

Another example

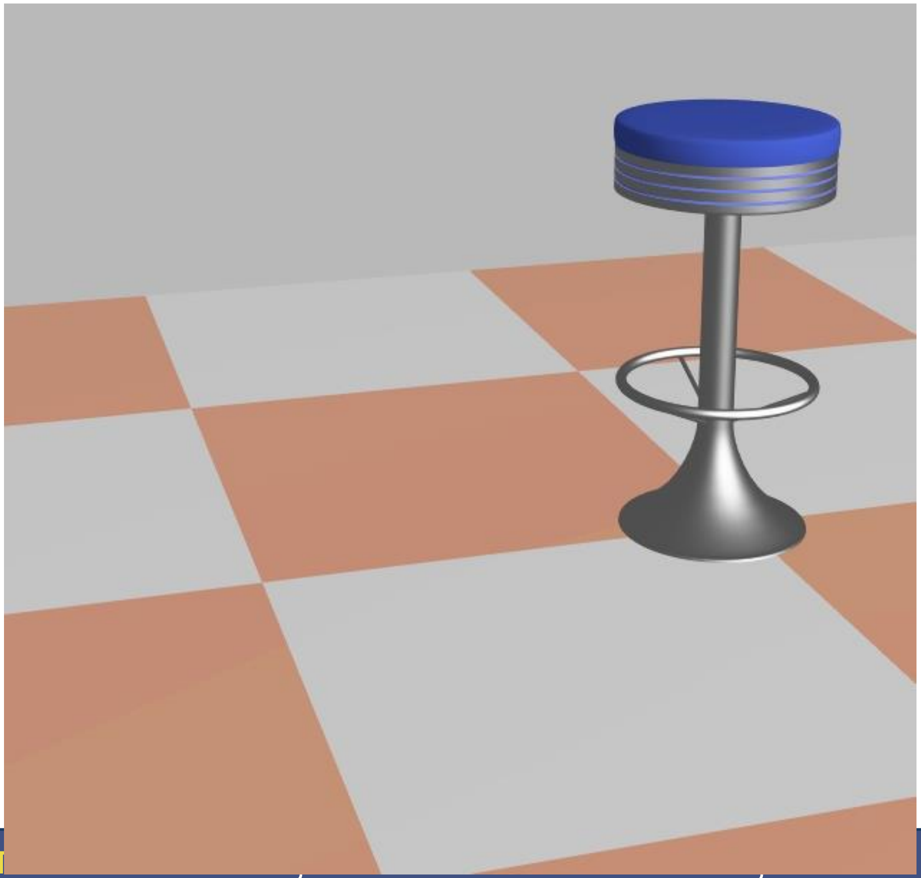


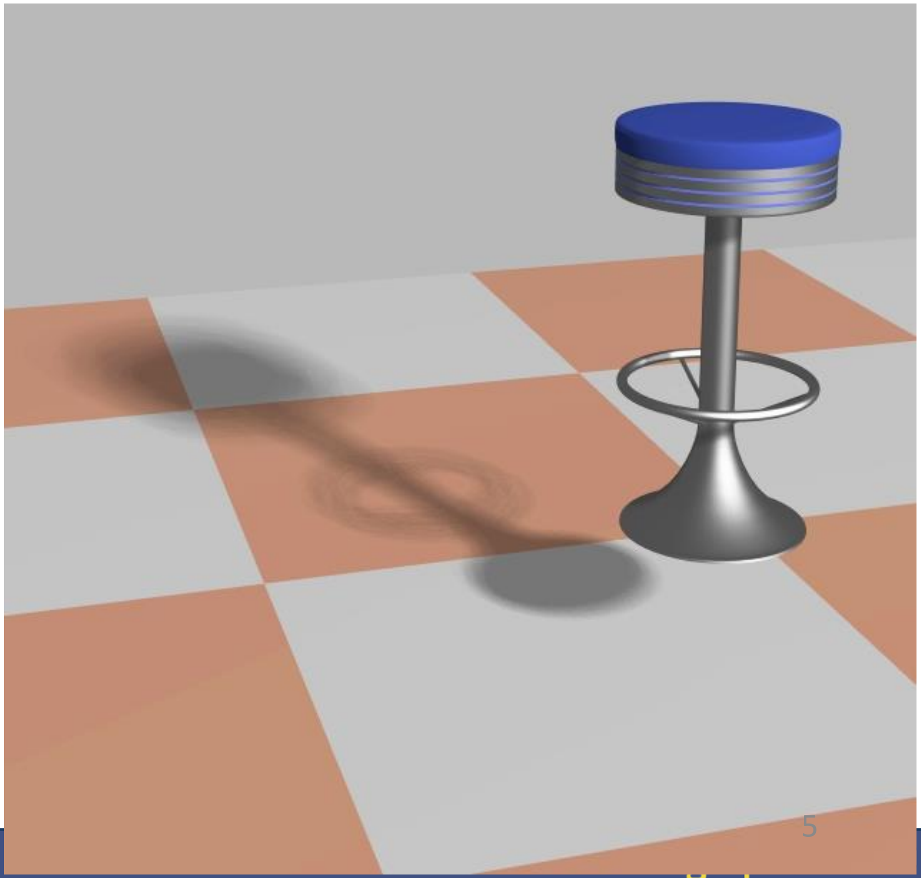
Doom 3 – Shadow volumes

POV
FLAKES
Pov is great!
The Kitchen - Jaime Vives Piqueres - POVCOMP 2004

Why shadows?

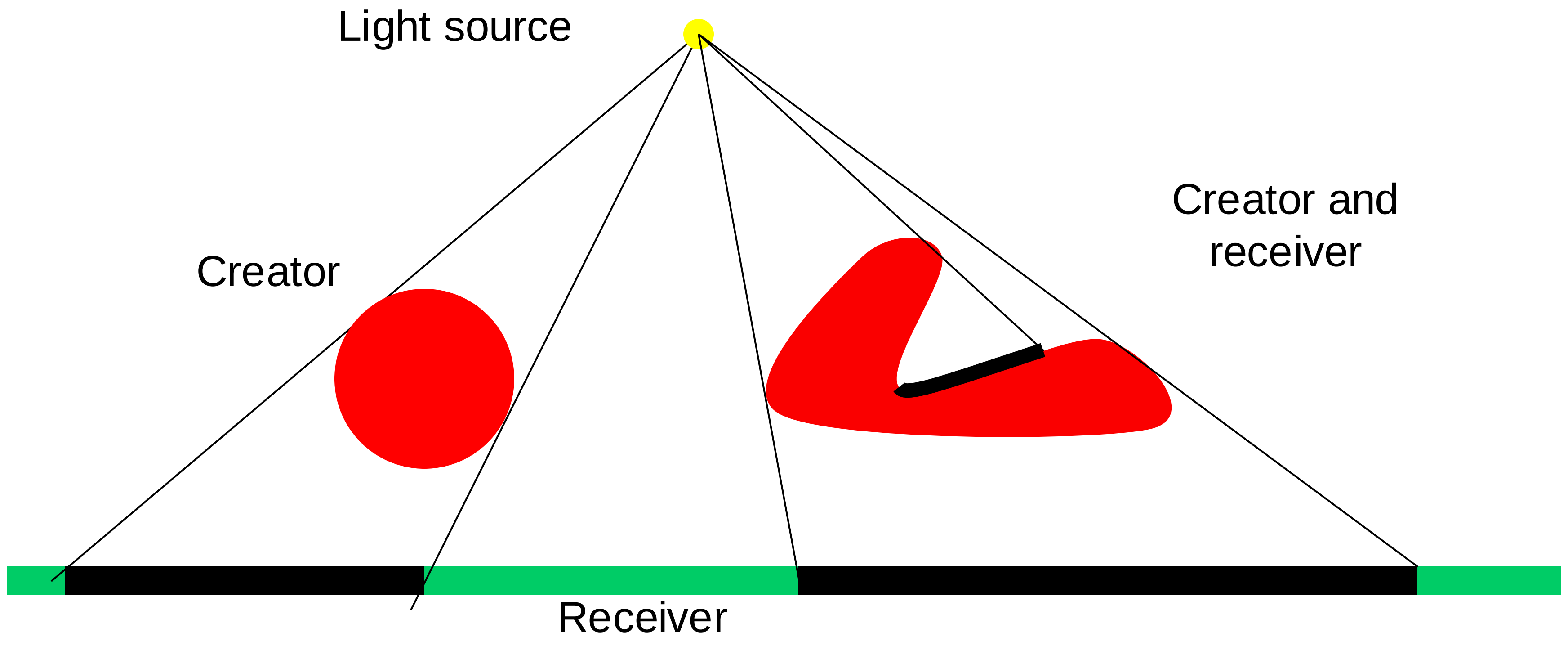
- More clues about spatial relationships
- Orientation & gameplay





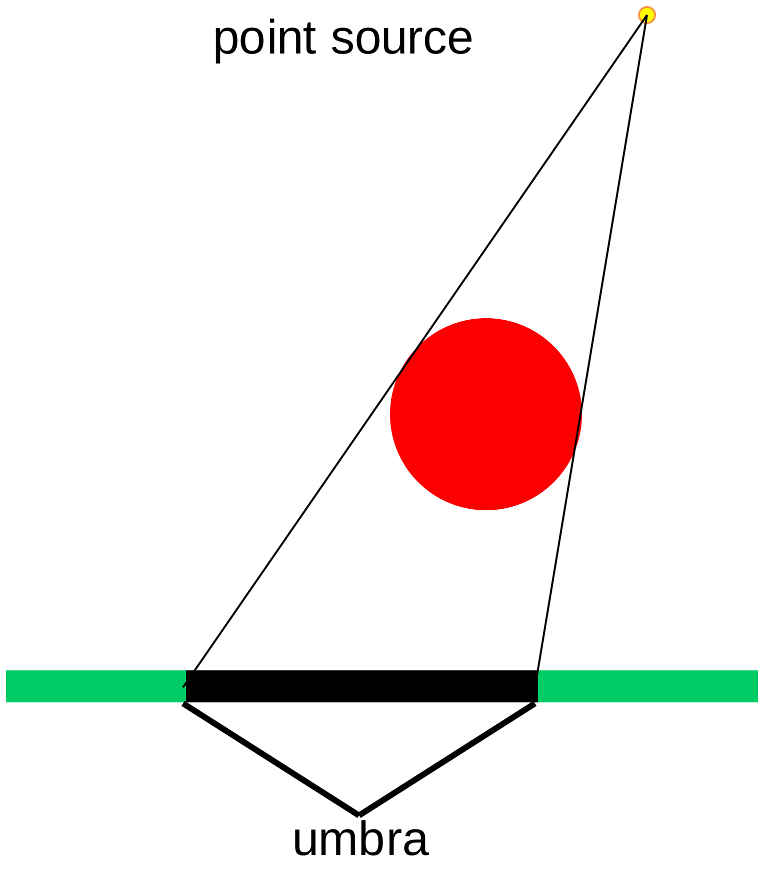
Definitions

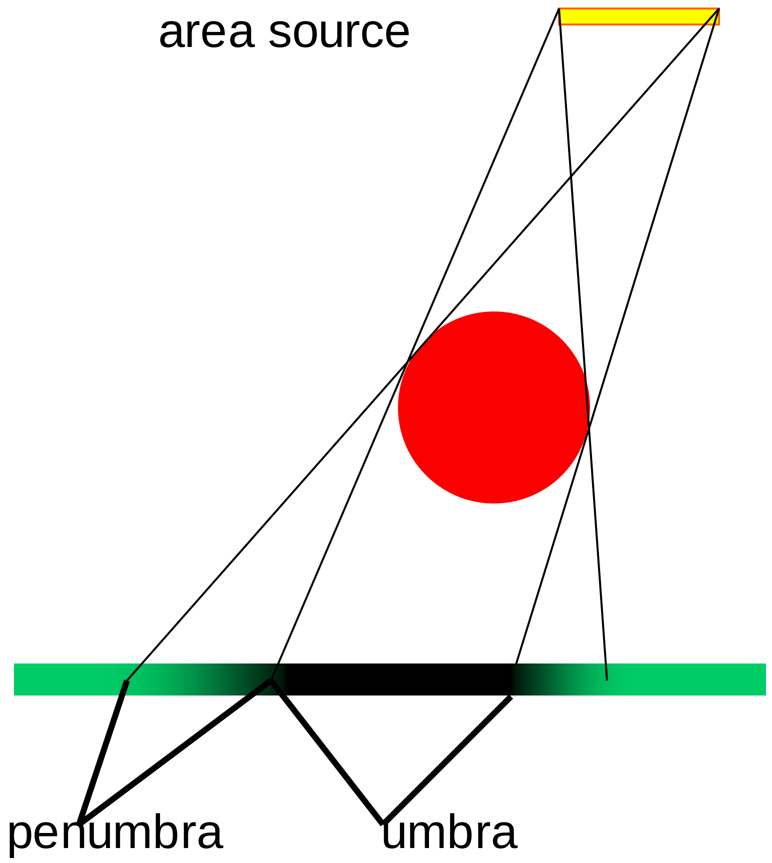
- Light sources
- Shadow creators and receivers



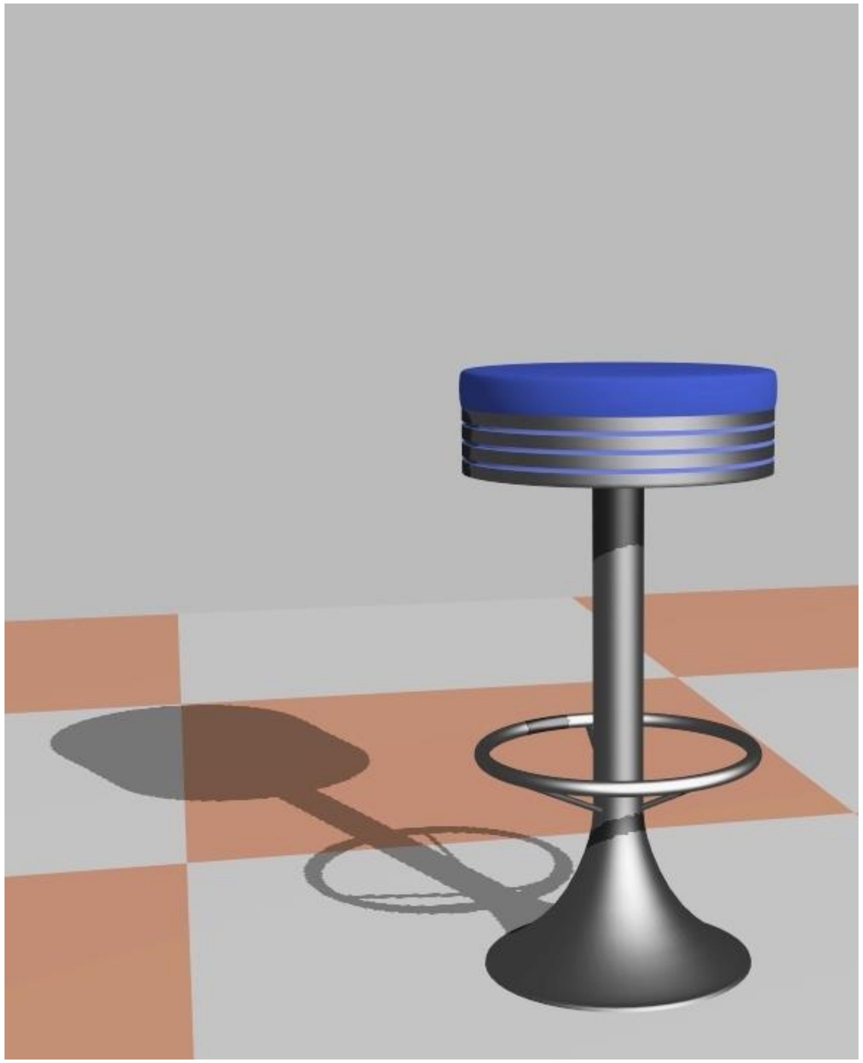
Definitions

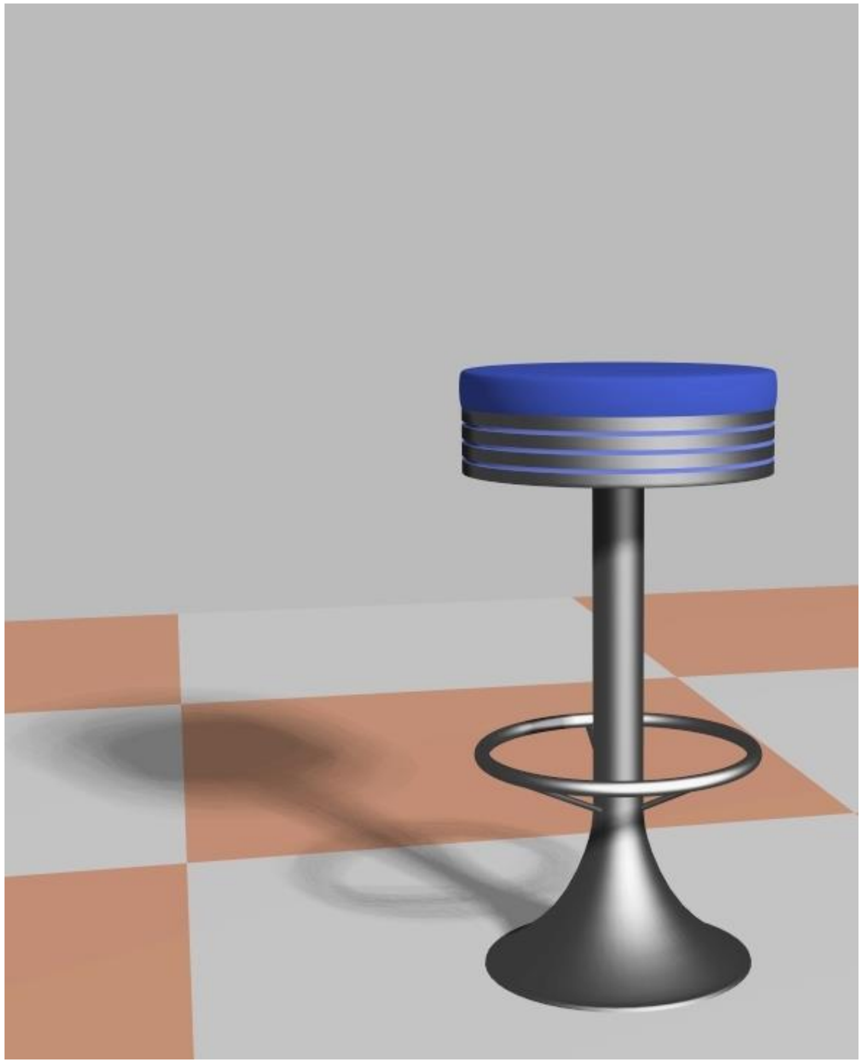
- Light source types



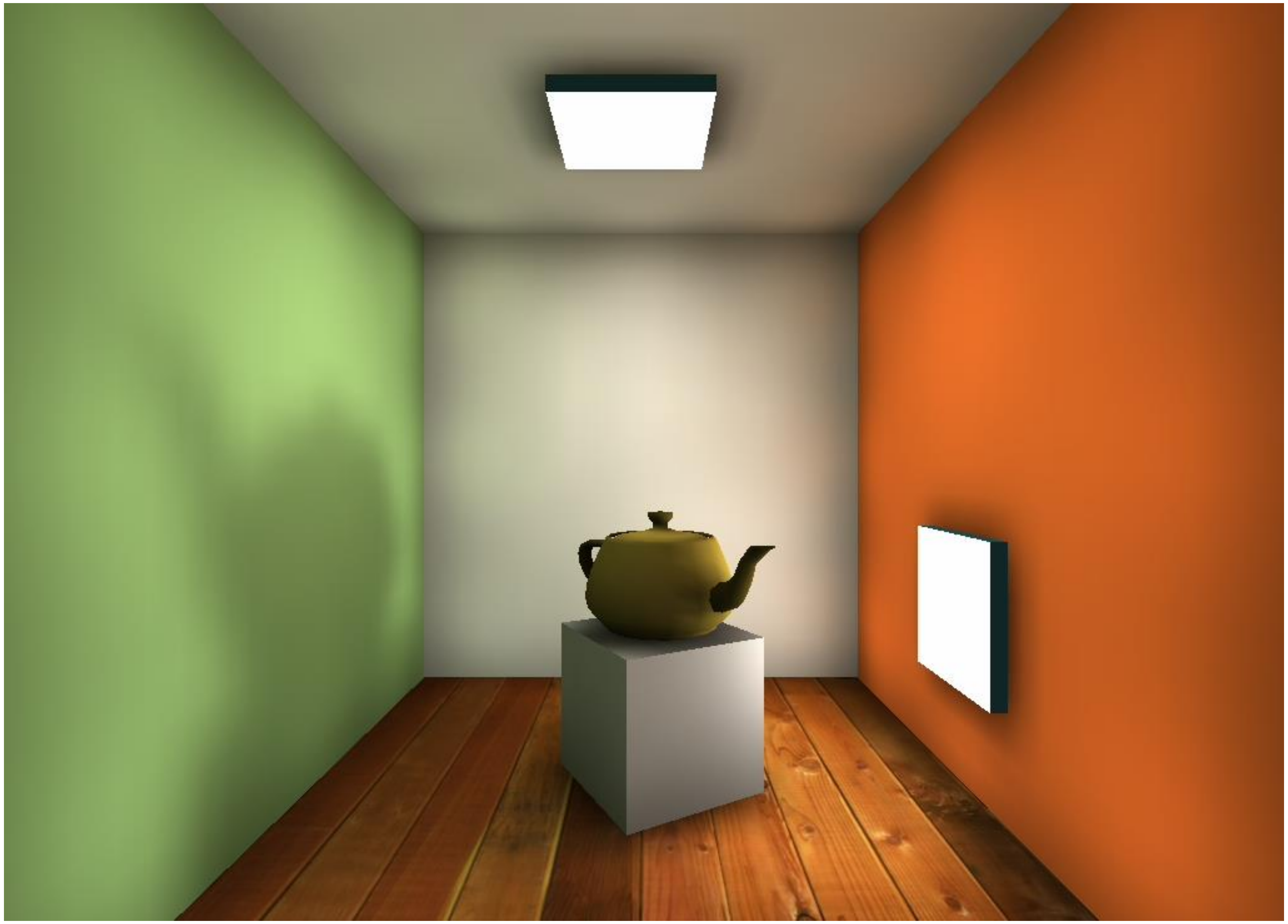


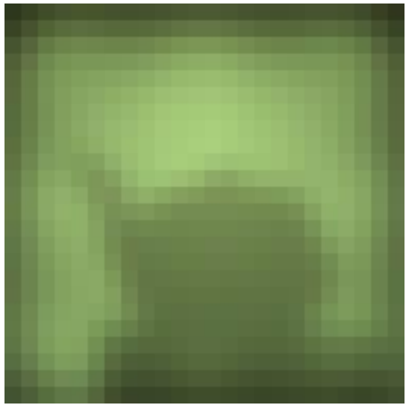
Example: hard vs soft shadows





Store precomputed shadows in textures

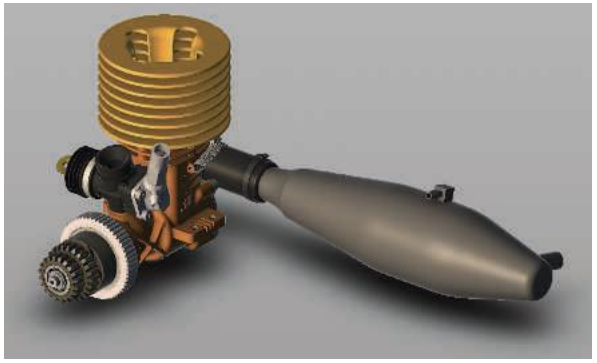




Images courtesy of Kasper Høy Nielsen.

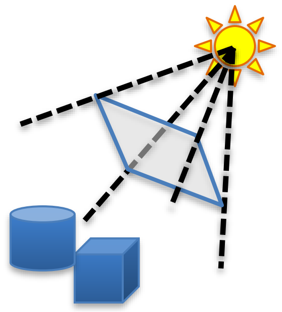
Ways of thinking about shadows

- As separate objects (like Peter Pan's shadow)
  - E.g., a drop shadow:



- As volumes of space that are dark
  - Shadow Volumes [Franklin Crow 77]
- As places not seen by a light source looking at the scene
  - Shadow Maps [Lance Williams 78]

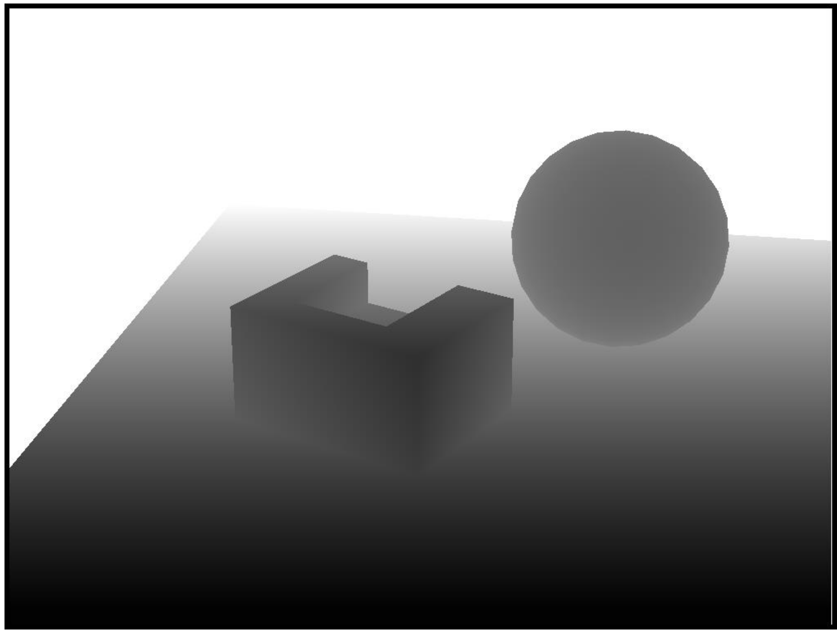
Shadow Maps



Basic Algorithm – the simple explanation:

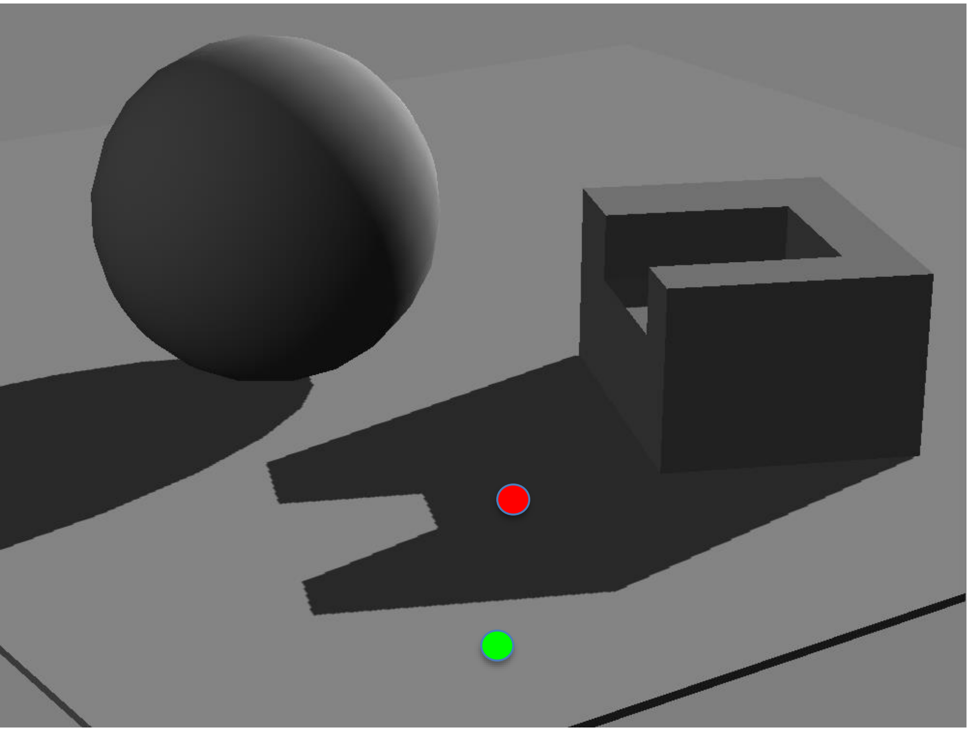
Idea:

- Render image from light source
  - Represents geometry in light
- Render from camera
  - Test if rendered point is visible in the light's view
    - If so -> point in light
    - Else -> point in shadow

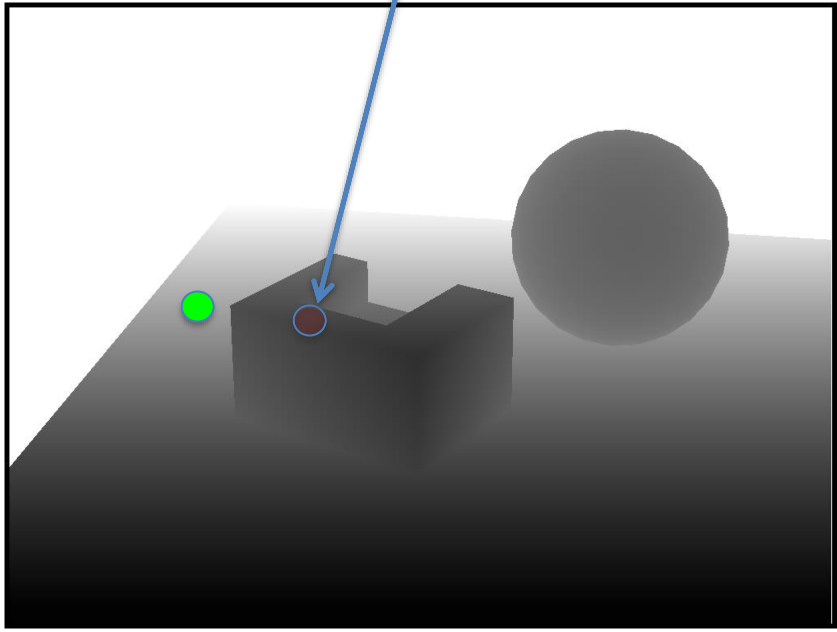


Shadow Map (light's view)

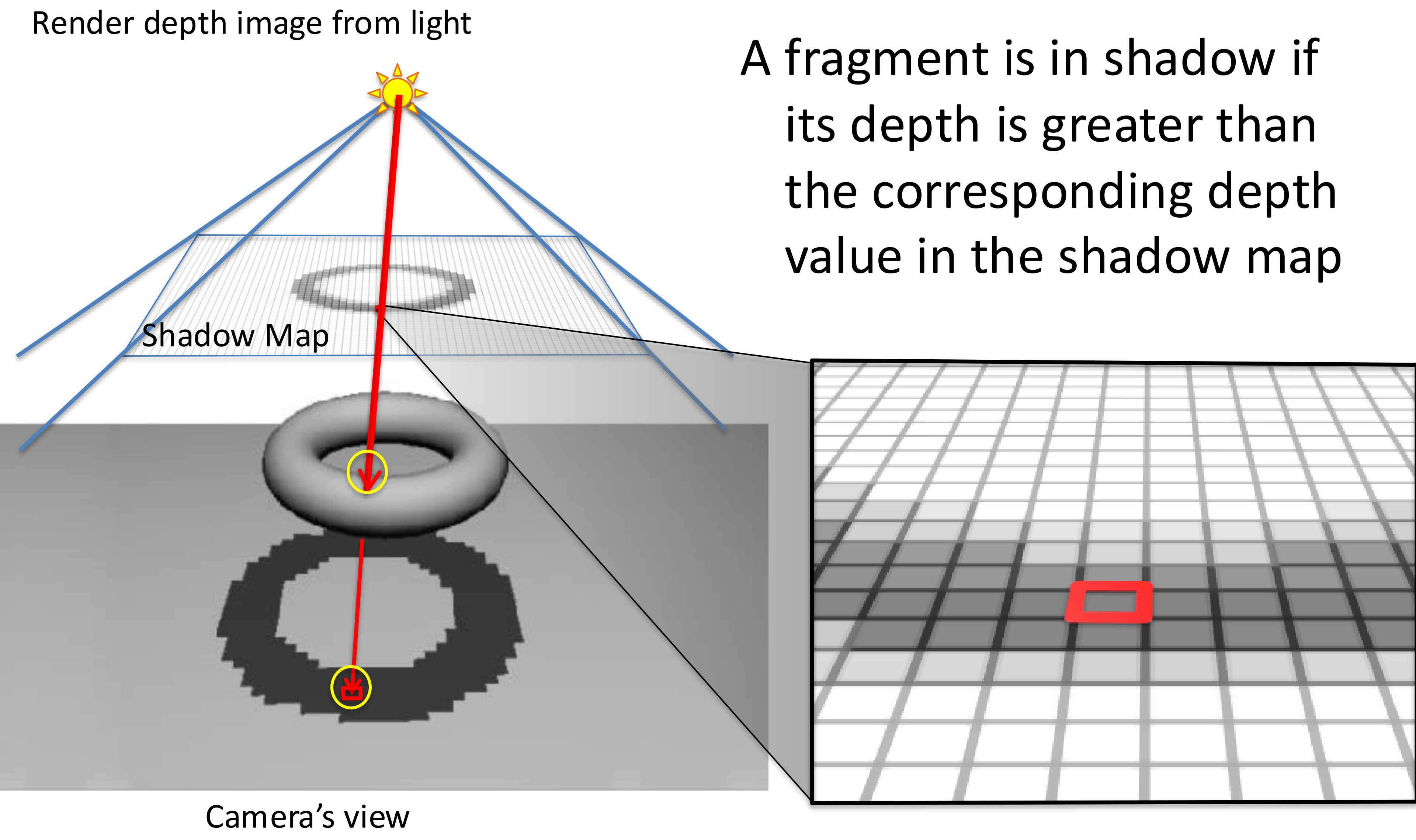
Shadow Maps


Camera's view

Point not represented in shadow map (point is behind box)


Light's view (Shadow Map)

Depth Comparison



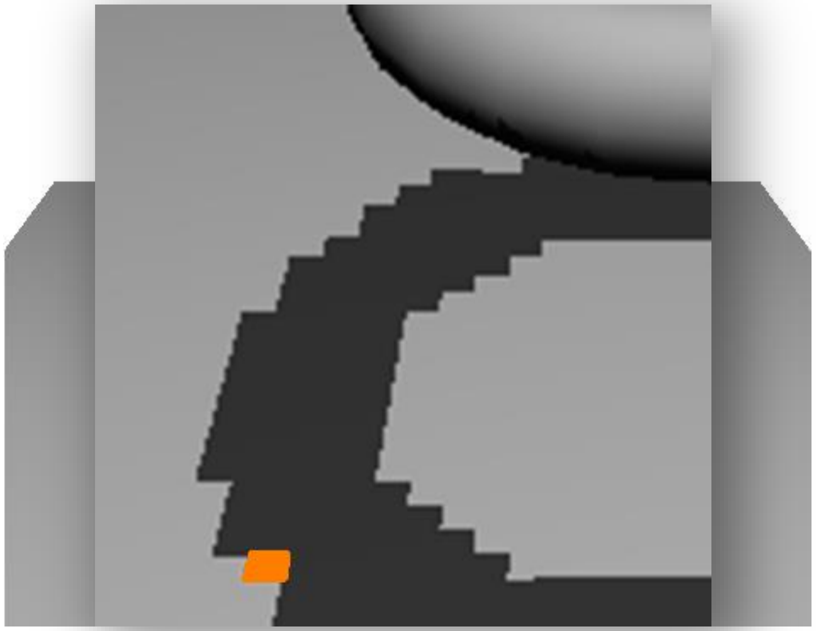
A fragment is in shadow if its depth is greater than the corresponding depth value in the shadow map

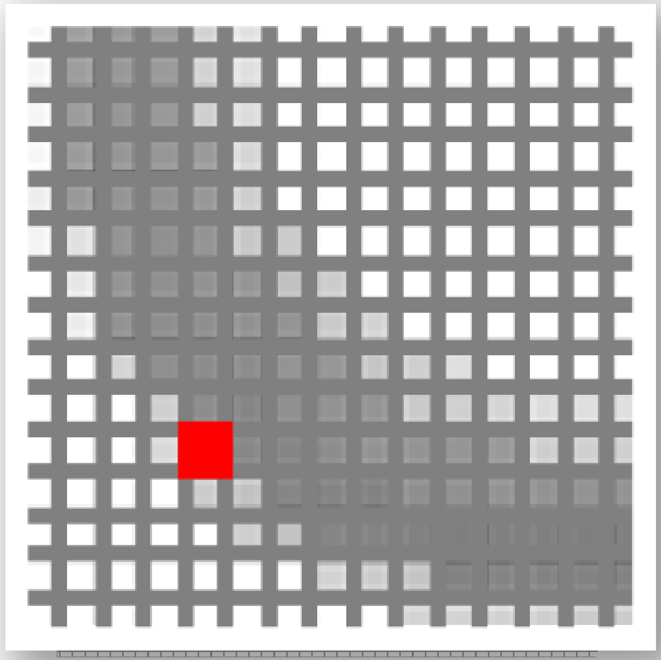
Shadow Maps

- Pros
  - Very efficient: "This is as fast as it gets"
- Cons…

Shadow Maps - Problems

- Low Shadow Map resolution results in jagged shadows


from viewpoint

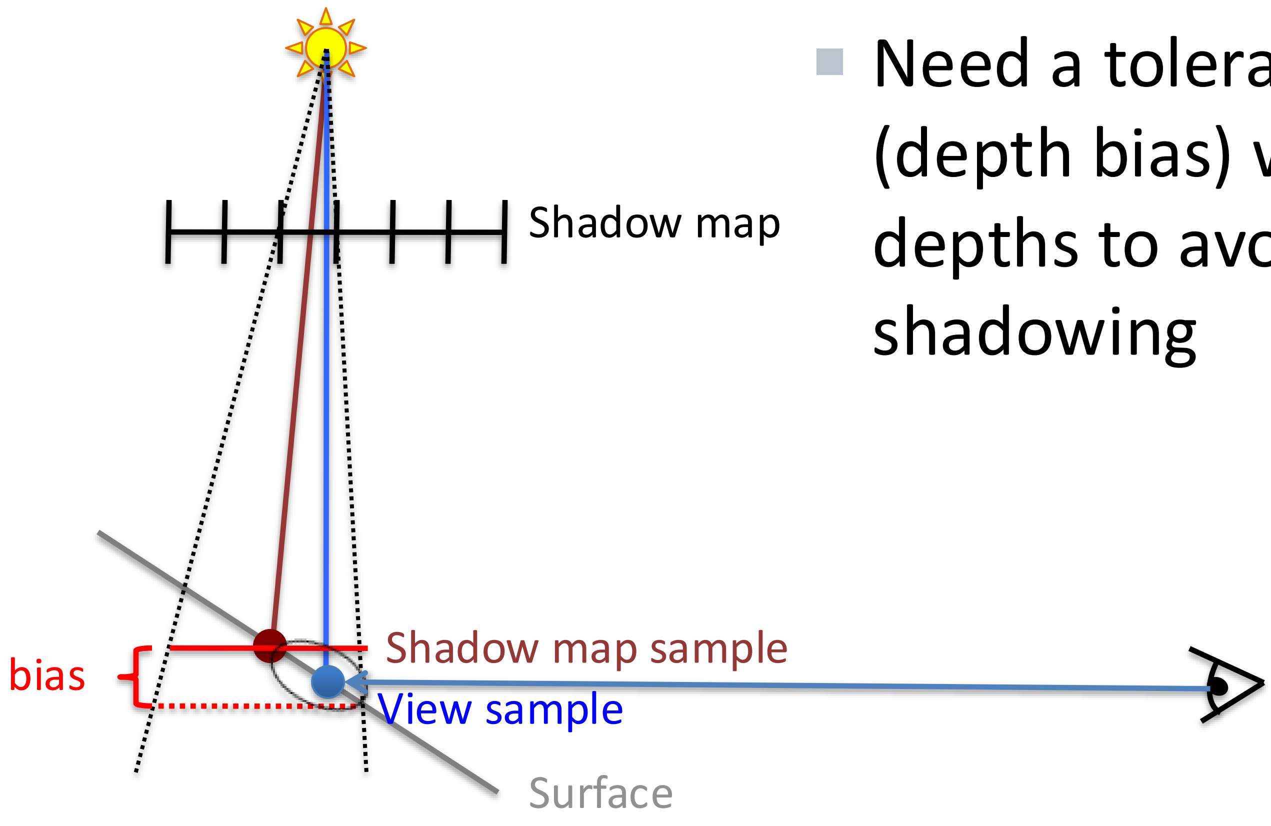

from light

Shadow Maps - Problems

In addition:

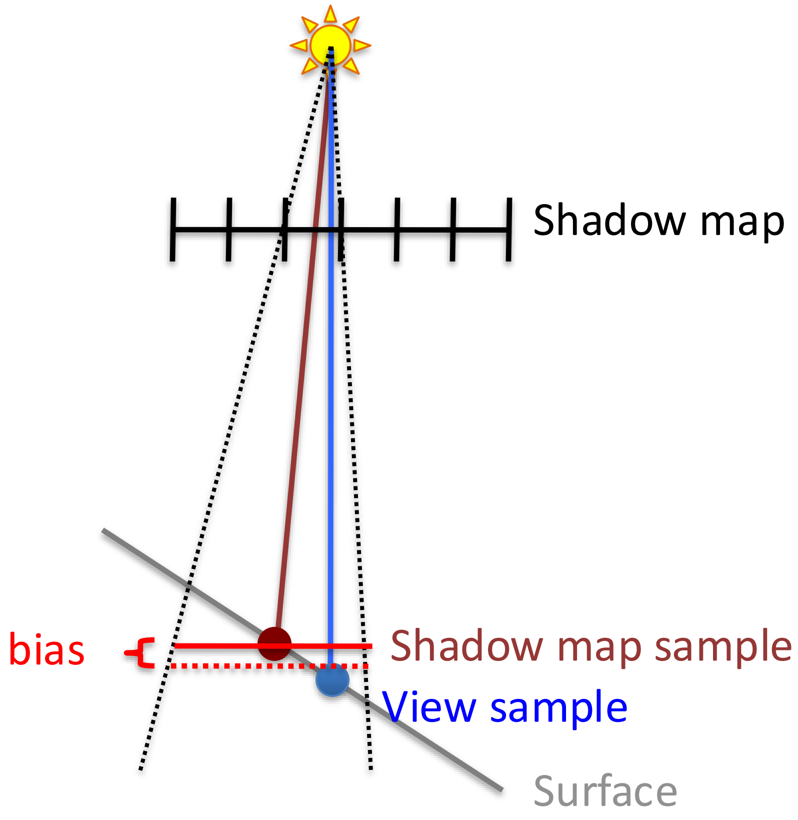
- A tolerance threshold (bias) needs to be tuned for each scene for the depth comparison

Bias

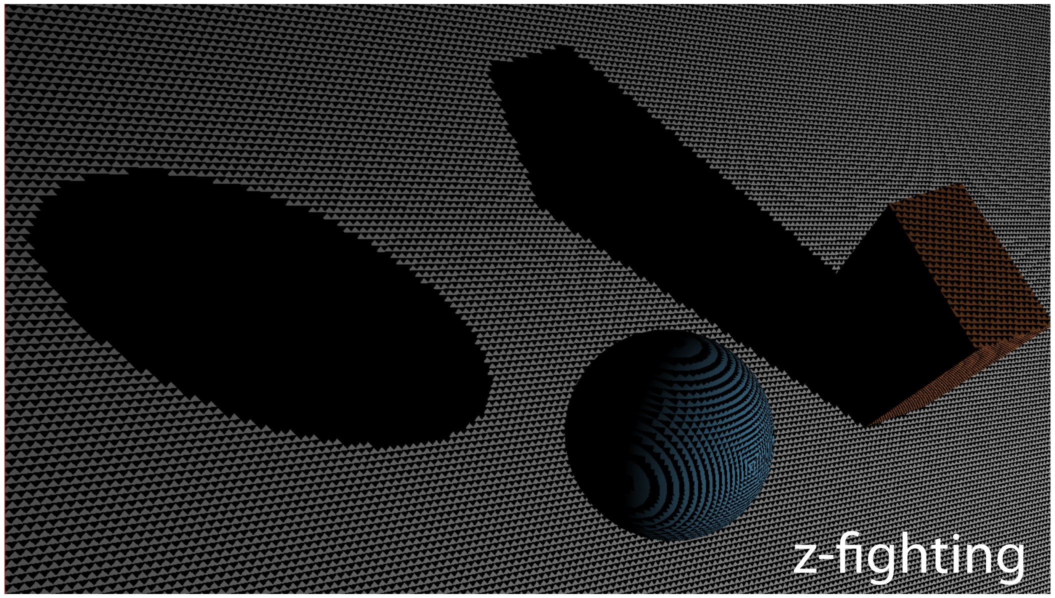


- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing

Bias

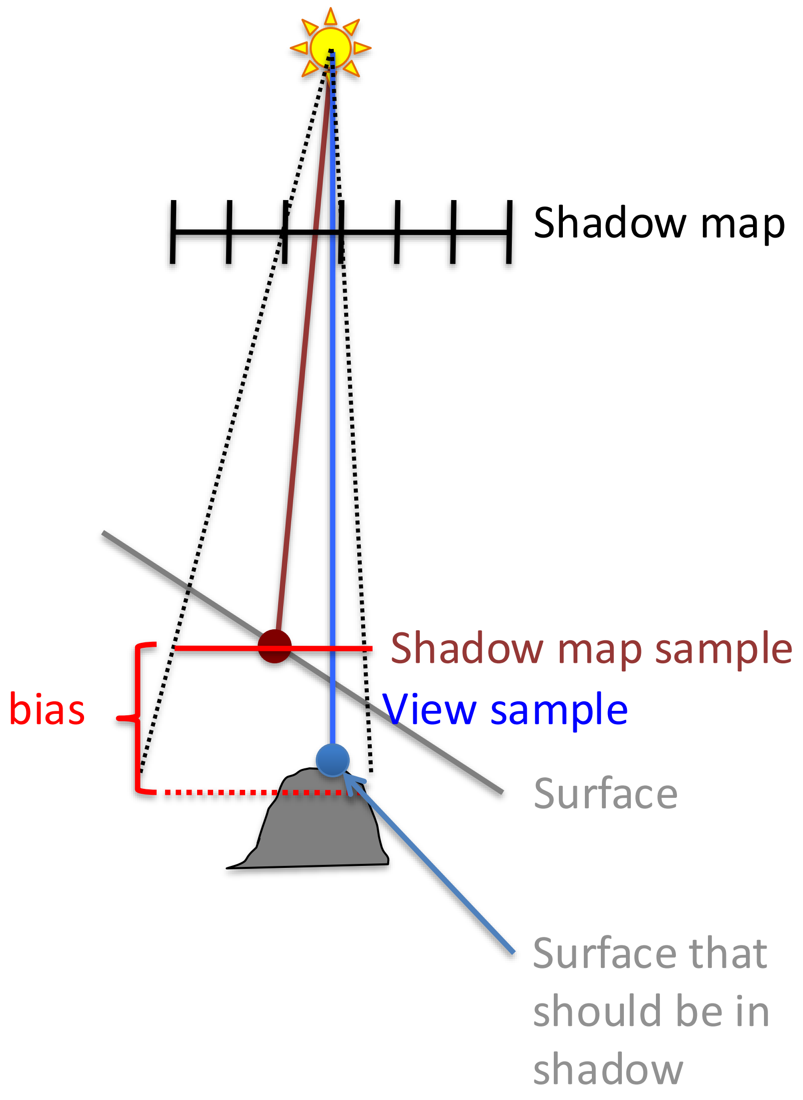


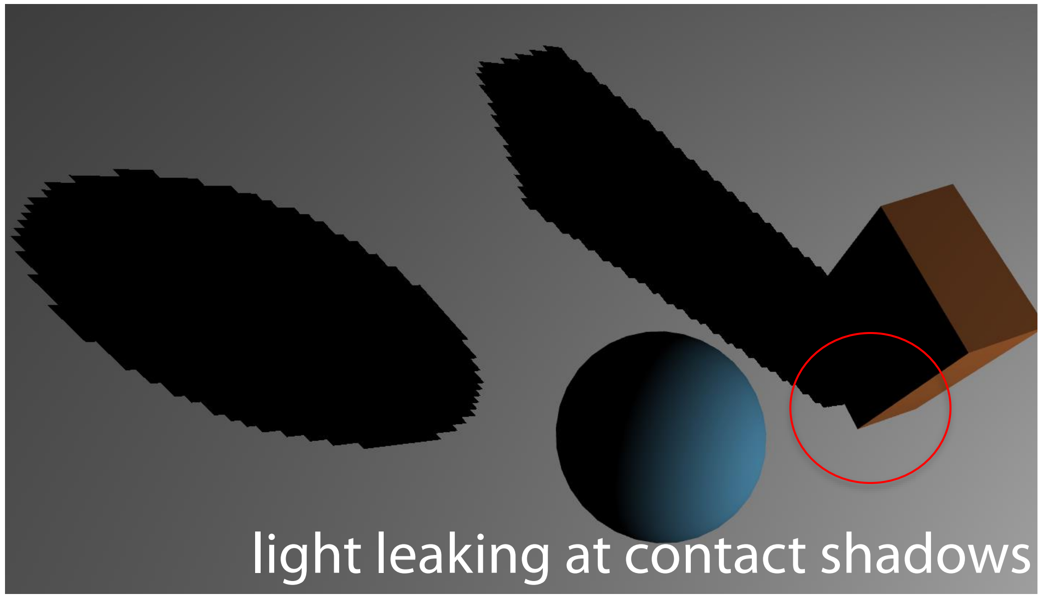
- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



Bias

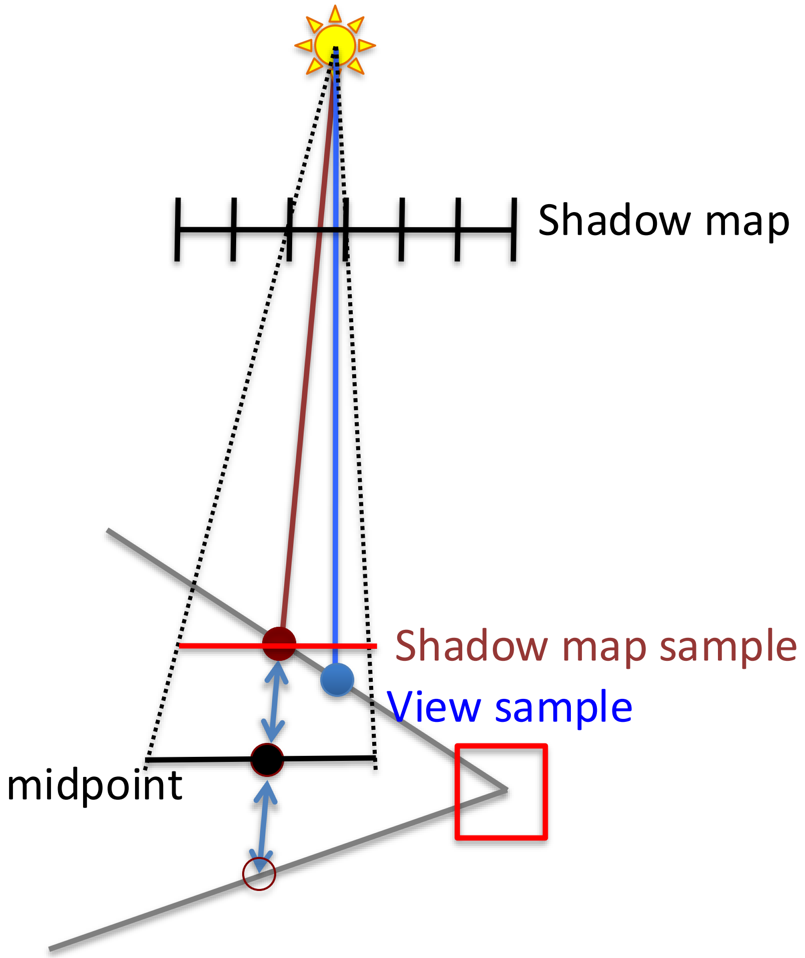
- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing





Ameliorating the Bias

- Midpoint Shadow Maps [Woo 92]
  - For closed objects, just 1 pass is needed
  - http://www.codersnotes.com/notes/midpoint/

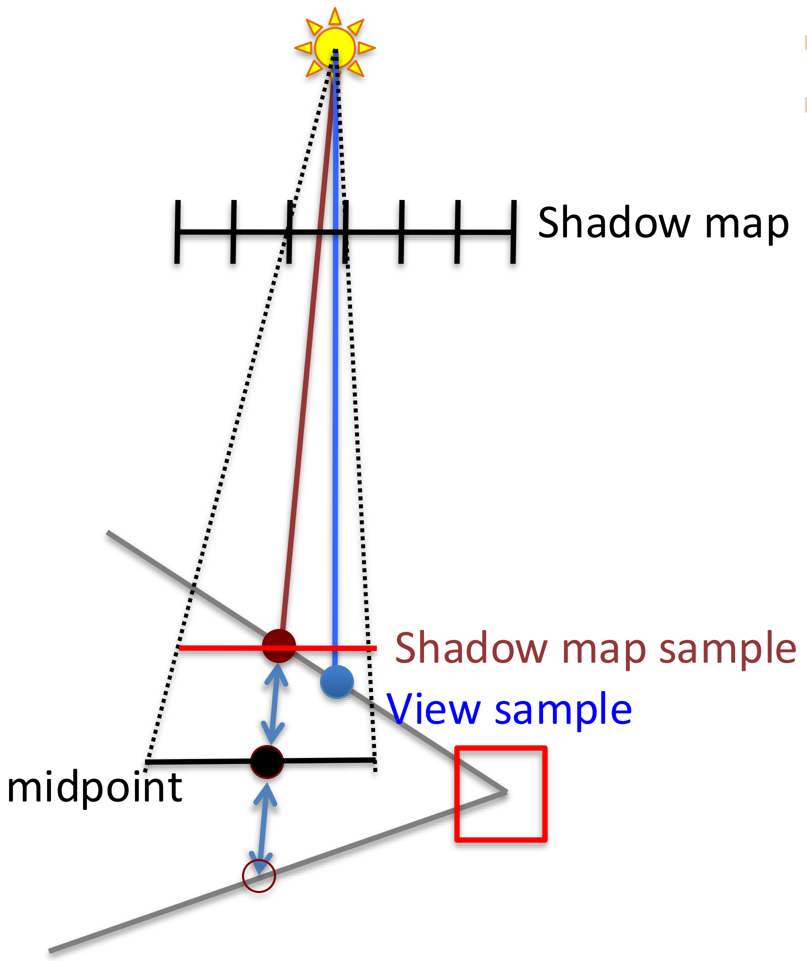


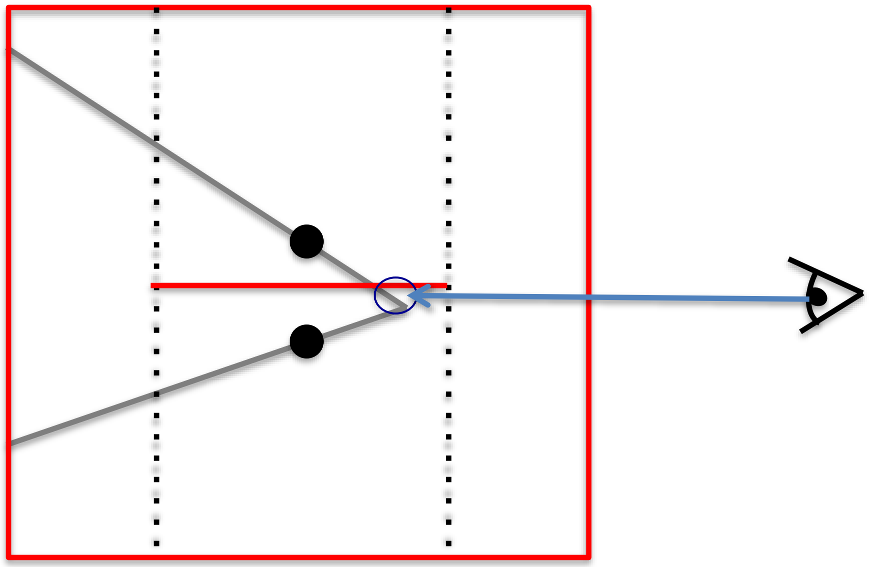
Further methods (even more accurate):

- Second Depth Shadow Mapping [Wang and Molnar94]
- Dual Depth Layer [Weiskopf and Ertl 04]
- Neither solves the problem completely but both improve a lot!
- But need depth peeling of 1st & 2nd layer!

Ameliorating the Bias

- Midpoint Shadow Maps [Woo 92]
  - For closed objects, just 1 pass is needed
  - http://www.codersnotes.com/notes/midpoint/

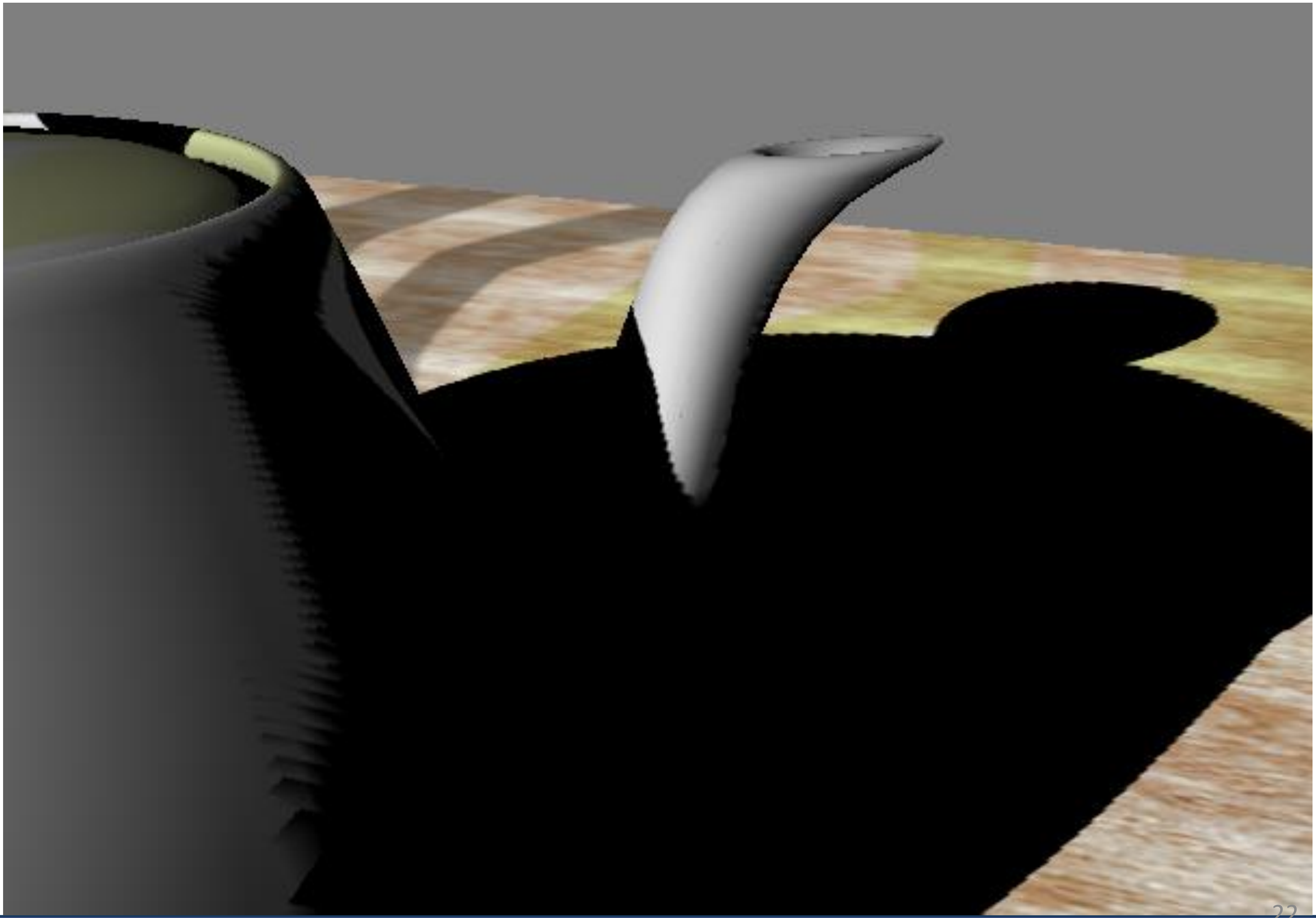




Further methods (even more accurate):

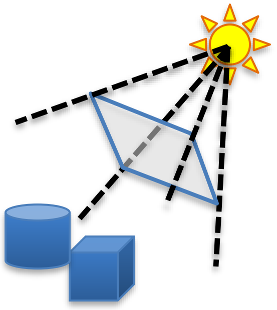
- Second Depth Shadow Mapping [Wang and Molnar94]
- Dual Depth Layer [Weiskopf and Ertl 04]
- Neither solves the problem completely but both improve a lot!
- But need depth peeling of 1st & 2nd layer!

Shadow Maps

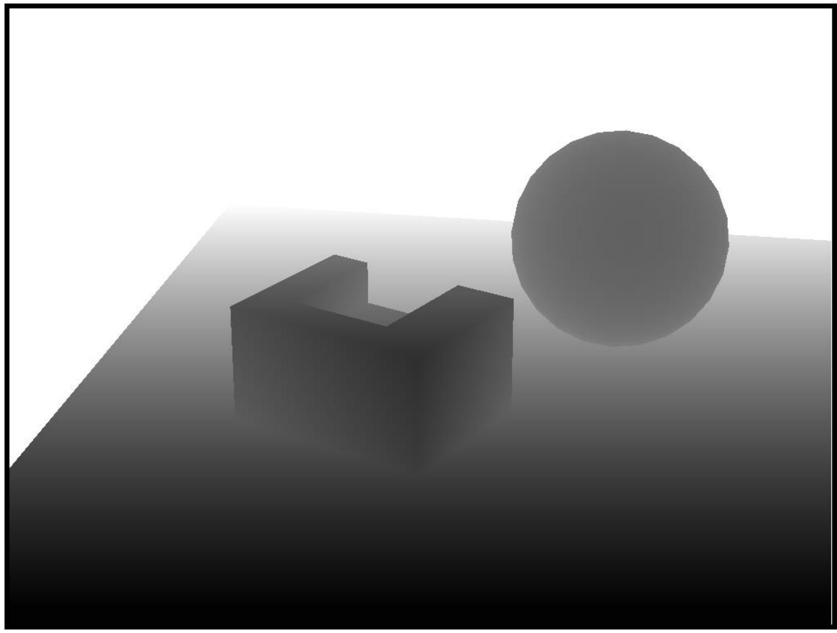


Shadow Maps - Summary

Shadow Map Algorithm:

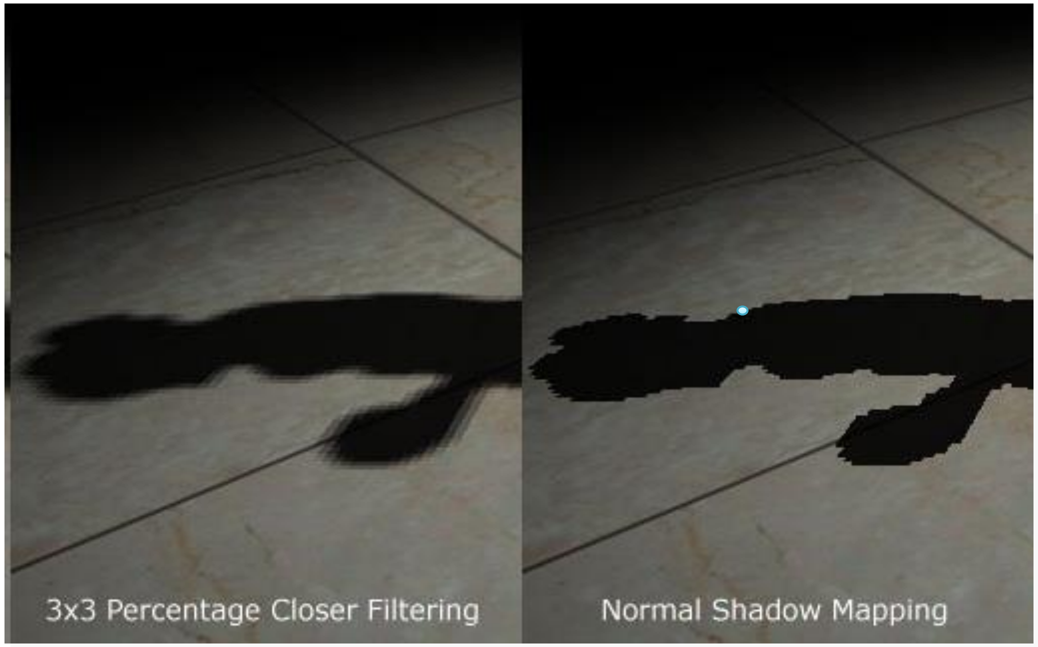


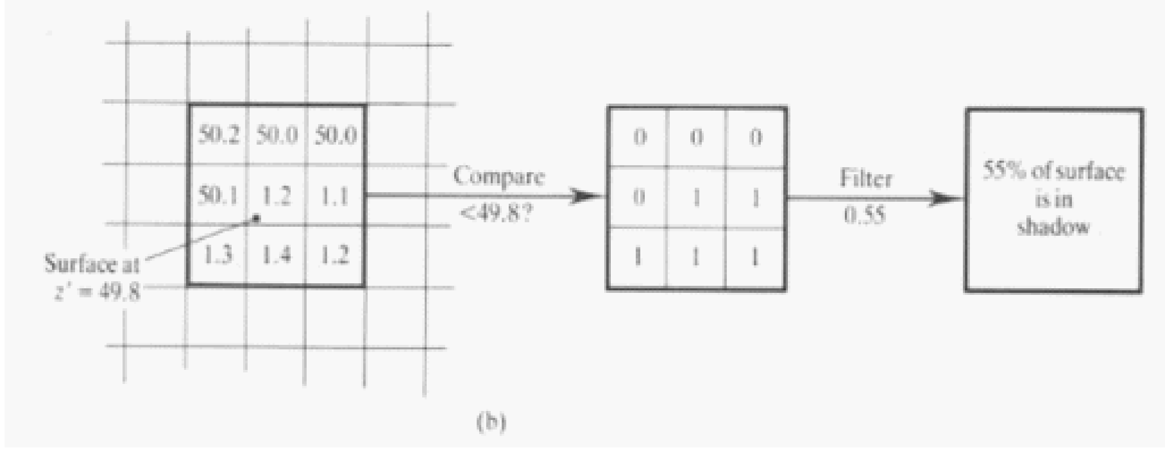
- Render a z-buffer from the light source
  - Represents geometry in light
- Render from camera
  - For every fragment:
    - transform its 3D-pos into shadow map (light space)
    - If depth greater-> point in shadow
    - Else -> point in light
    - Use a bias at the comparison



Shadow Map (=depth buffer)

Percentage Closer Filtering





Cascaded Shadow Maps

- You need high SM resolution close to the camera and can use lower further away. So create a separate SMs per depth region of the view frustum, with higher spatial resolution closer to camera.

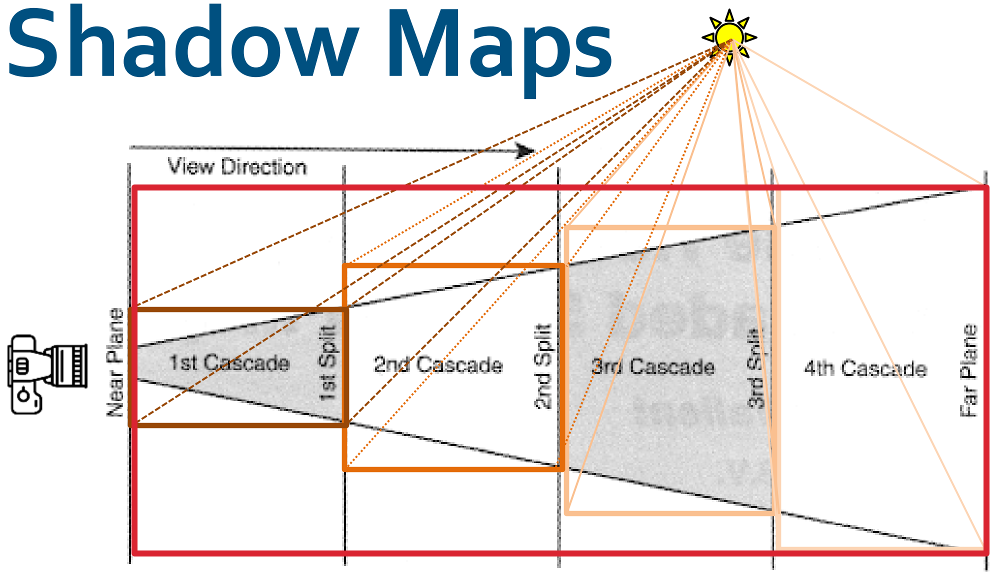
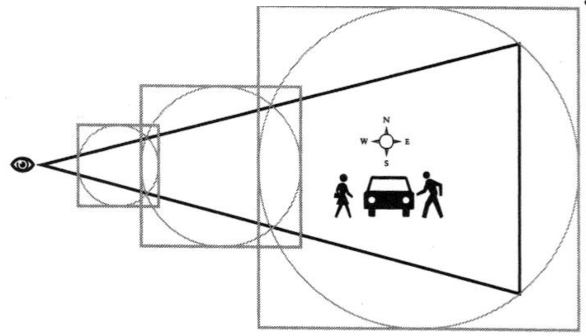
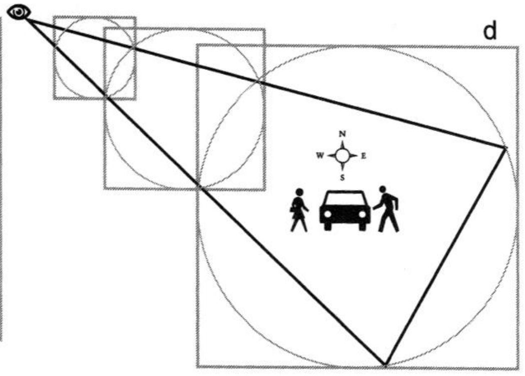


FIGURE 4.1.1 2D visualization of view frustum split (uniformly) into separate cascade frustums.

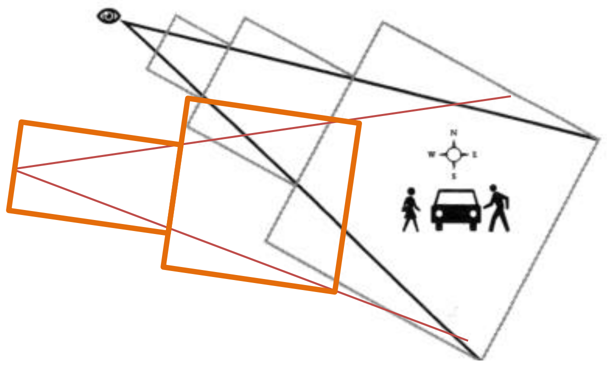
- Optionally:



To hide discrete resolution changes, let SMs overlap a bit and blend result from both at overlap.



Aligned SMs allow resuse between frames for small cam movements...



... as opposed to non-aligned SM (if the scene is static).

Super high resolution shadow maps and fast 9x9 tap (pcf) filtering using Sparse Voxel DAGs:

Compact Precomputed Voxelized Shadows

Erik Sintorn* Viktor Kämpe* Ola Olsson* Ulf Assarsson*
Chalmers University of Technology

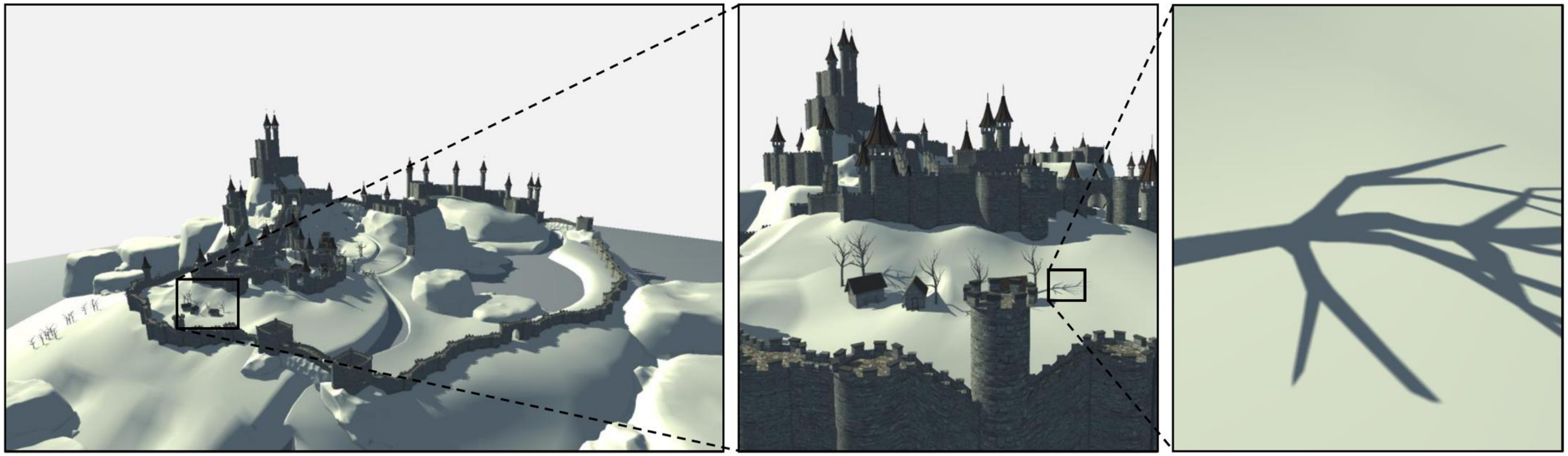


Figure 1: *An example of using our algorithm to evaluate precomputed shadows from the sun when viewing the scene at varying scales. Our compact data structure occupies 100MB of graphics memory and is equivalent to a* $256k \times 256k$ *(i.e.* 262144^2*) shadow map. With a filter size of* 9×9 *taps, shadow evaluation is done in* $< 1ms$ *at 1080p resolution.*

SM, 165 lights, 4ms/frame, ~300x lossless compression

Fast, Memory-Efficient Construction of Voxelized Shadows

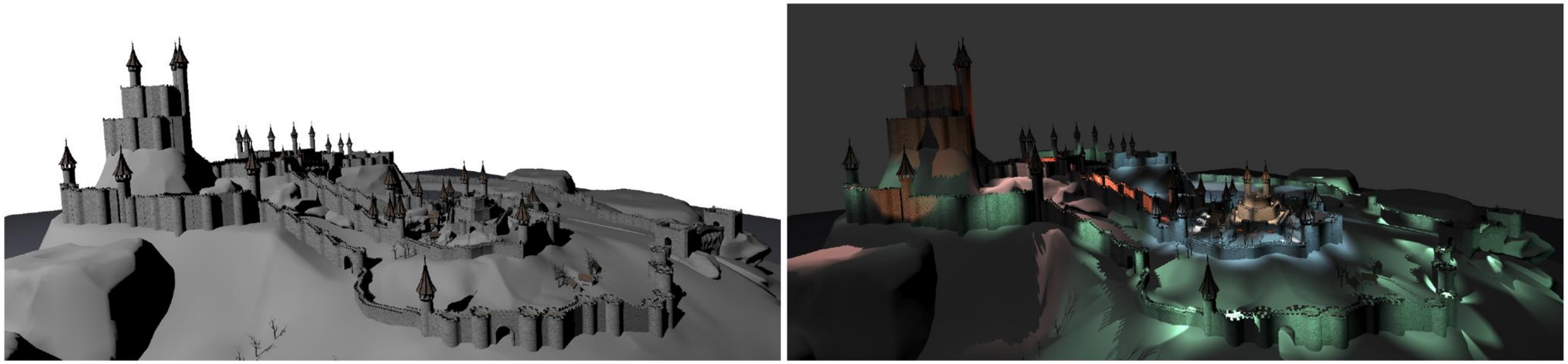


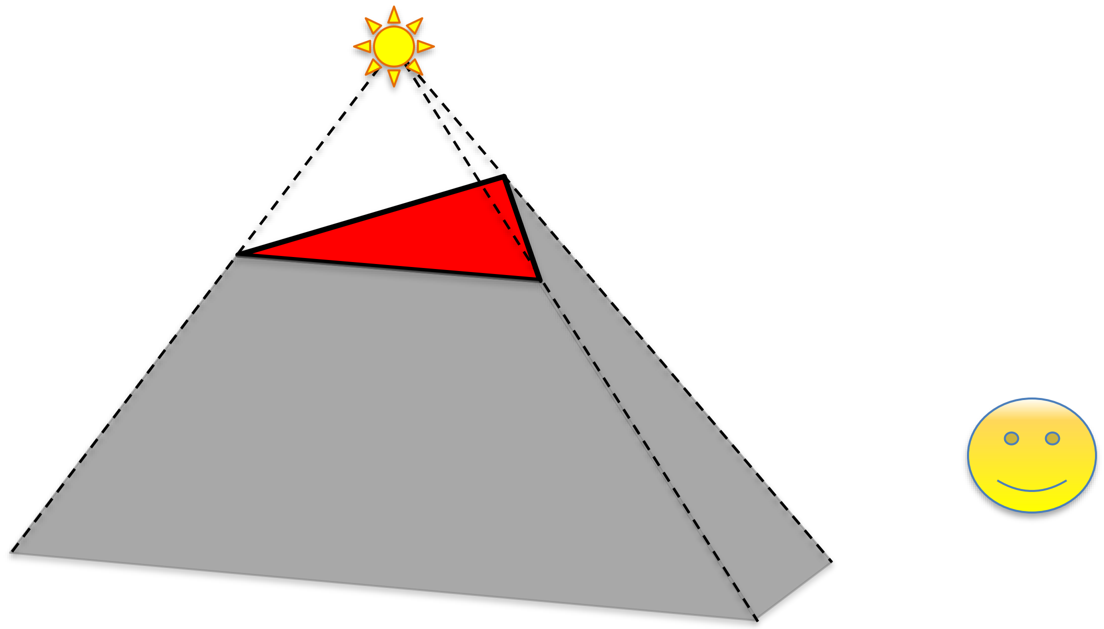
Figure 1: *The left image shows a scene lit by the sun with precomputed voxelized shadows of resolution 262144^3. Our novel algorithm generates this shadow information in 38 seconds and compresses it to 48MB (vs. 100MB for the previous CPVS method). To the right is the same scene lit by 165 spotlights with precomputed shadows, each with a resolution of 8192^3. The average build time for these CPVSs is 114ms, and the average size is 0.5MB (vs. 128MB for a 16-bit shadow map). Evaluating shadows for all lights at 1920×1080 takes 3.2ms.*

Or using Sparse Voxel Quadtrees:

- see *“Compressed Multiresolution Hierarchies for High-Quality Precomputed Shadows”*, by Leonardo Scandolo, Pablo Bauszat, and Elmar Eisemann

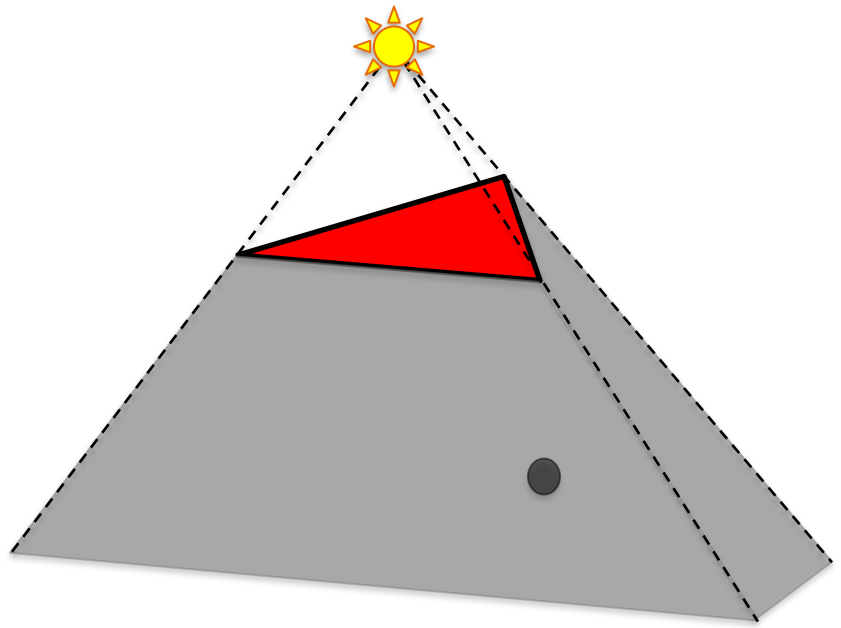
Shadow Volumes

- Concept
  - Create volumes of "space in shadow" from each triangle
    - Each triangle creates 3 quads that extends to infinity



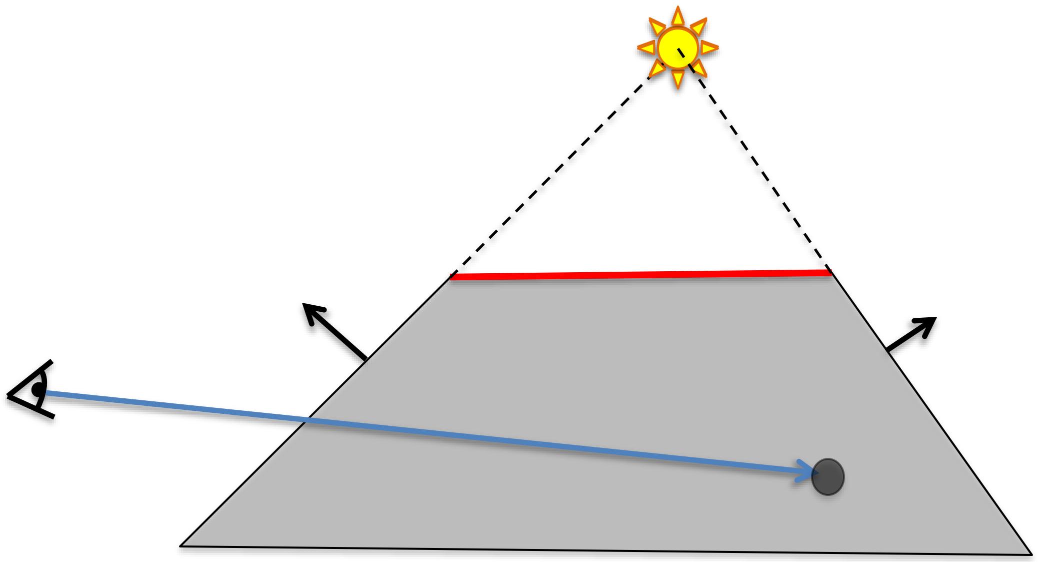
Shadow Volumes

- To test a point, count how many shadow volumes it is located within. One or more means the point is in shadow



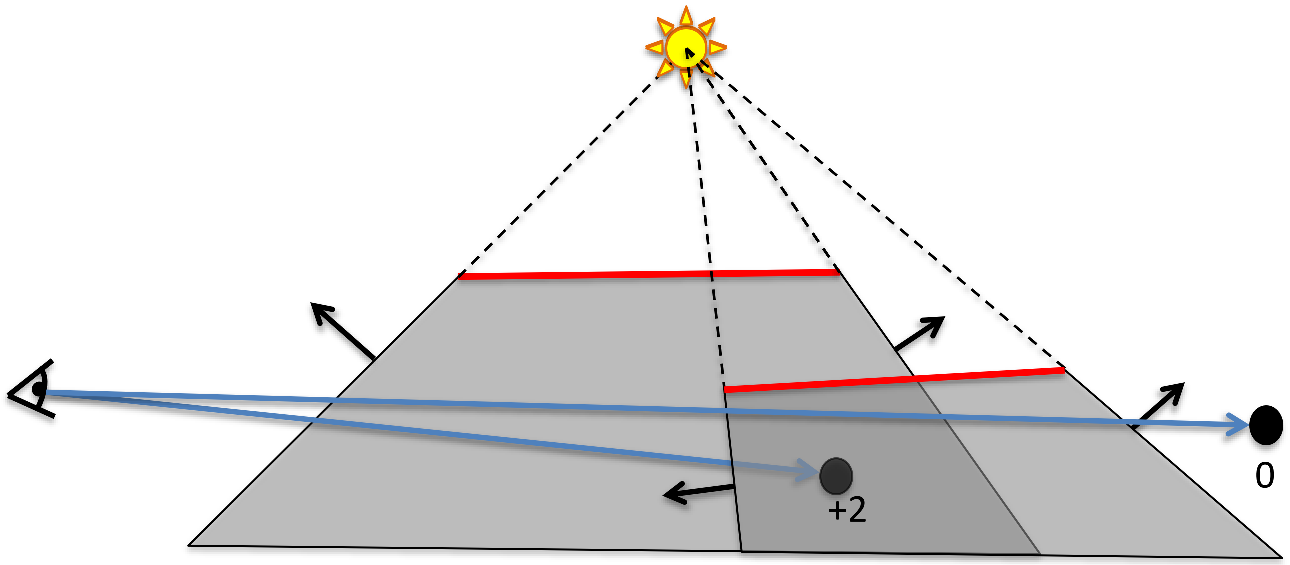
Shadow Volumes

- To test a point, count how many shadow volumes it is located within. One or more means the point is in shadow



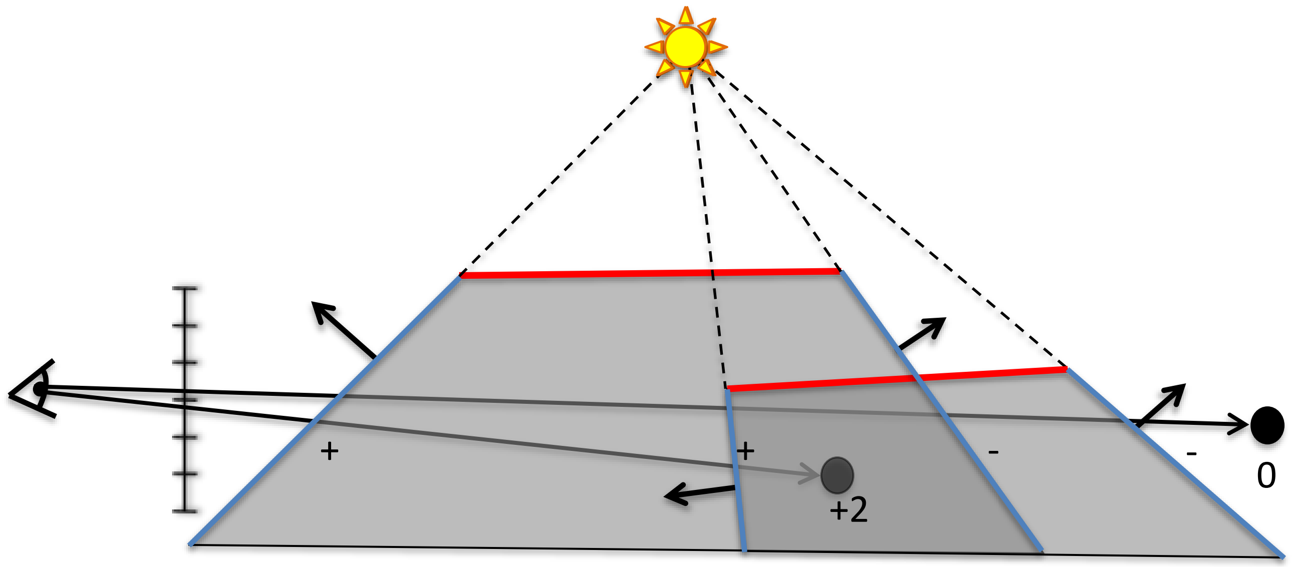
Shadow Volumes

- To test a point, count how many shadow volumes it is located within. One or more means the point is in shadow



Shadow Volumes - concept

- A counter per pixel
- If we go through more frontfacing than backfacing polygons, then the point is in shadow



Shadow volume algorithm uses stencil buffer

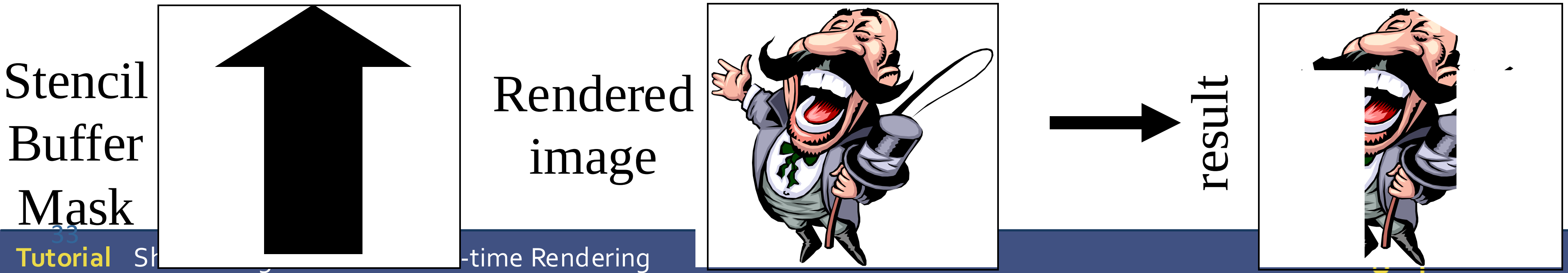
- Stencil what?
- Is just another buffer (often 8 bits per pixel)
- When rendering to it, we can add, subtract, etc
- Then, the resulting image can be used to mask off subsequent rendering

To create stencil values:

```
glStencilFunc(GL_ALWAYS, 0, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR):
```

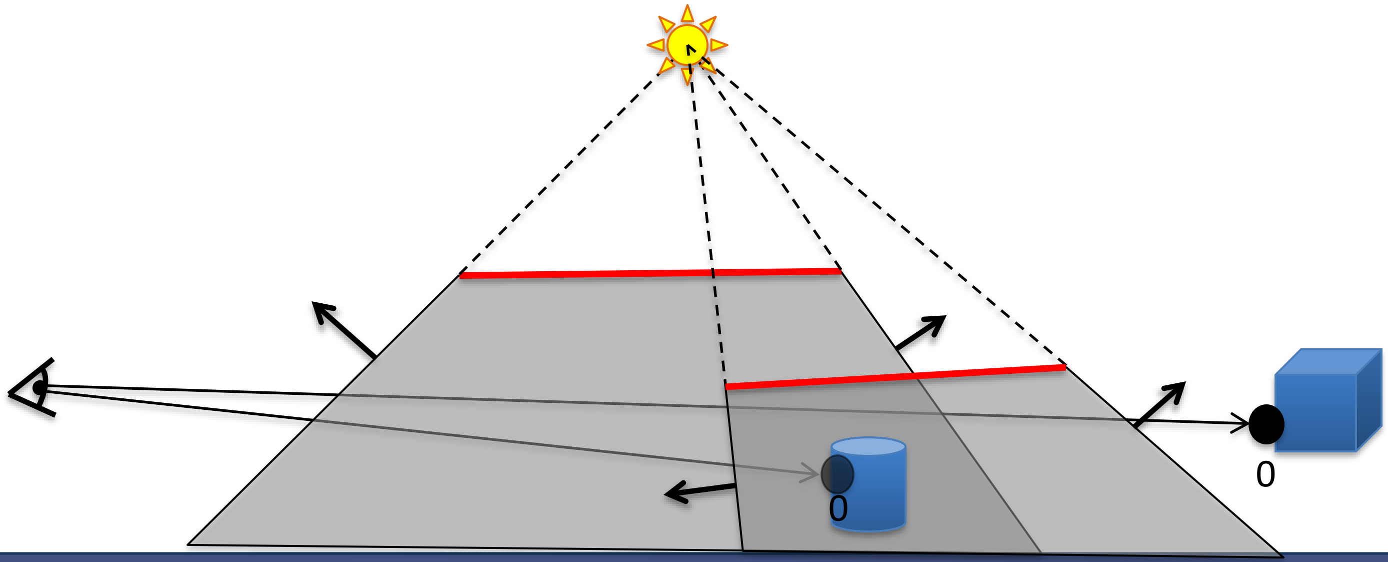
To use stencil buffer as mask:

```
glStencilFunc(GL_GREATER, 0, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP):
```



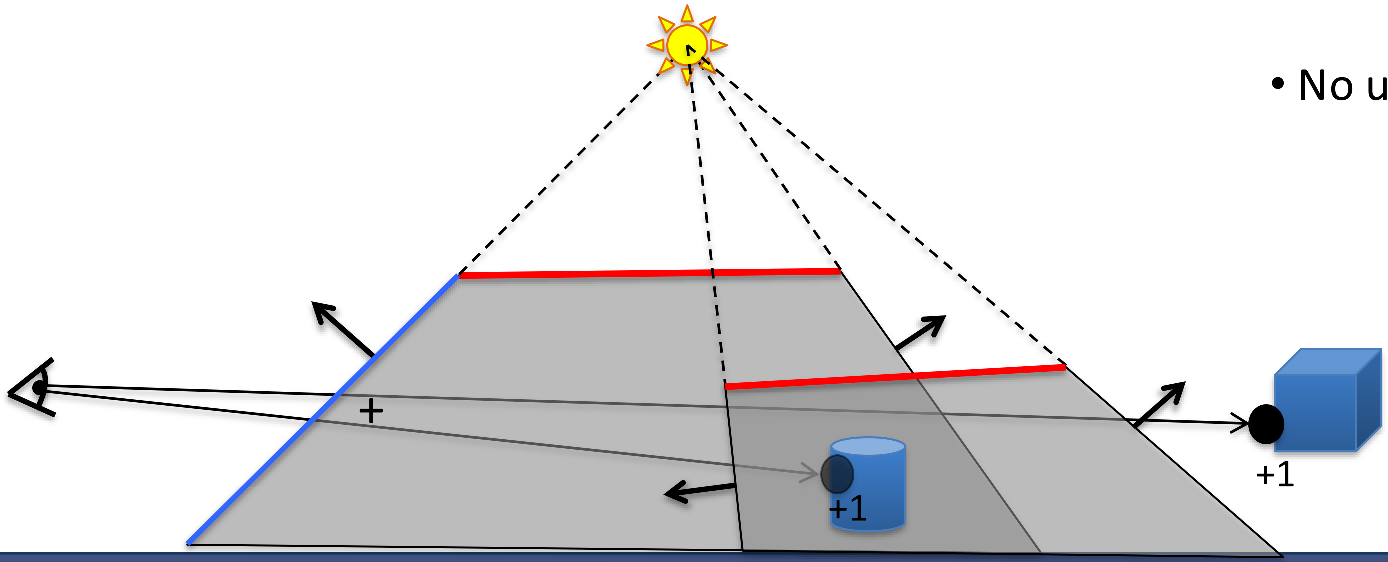
Shadow Volumes - concept

- Perform counting with the stencil buffer
  - Render front facing shadow quads to the stencil buffer
    - Inc stencil value, since those represents entering shadow volume
  - Render back facing shadow quads to the stencil buffer
    - Dec stencil value, since those represents exiting shadow volume



Shadow Volumes - concept

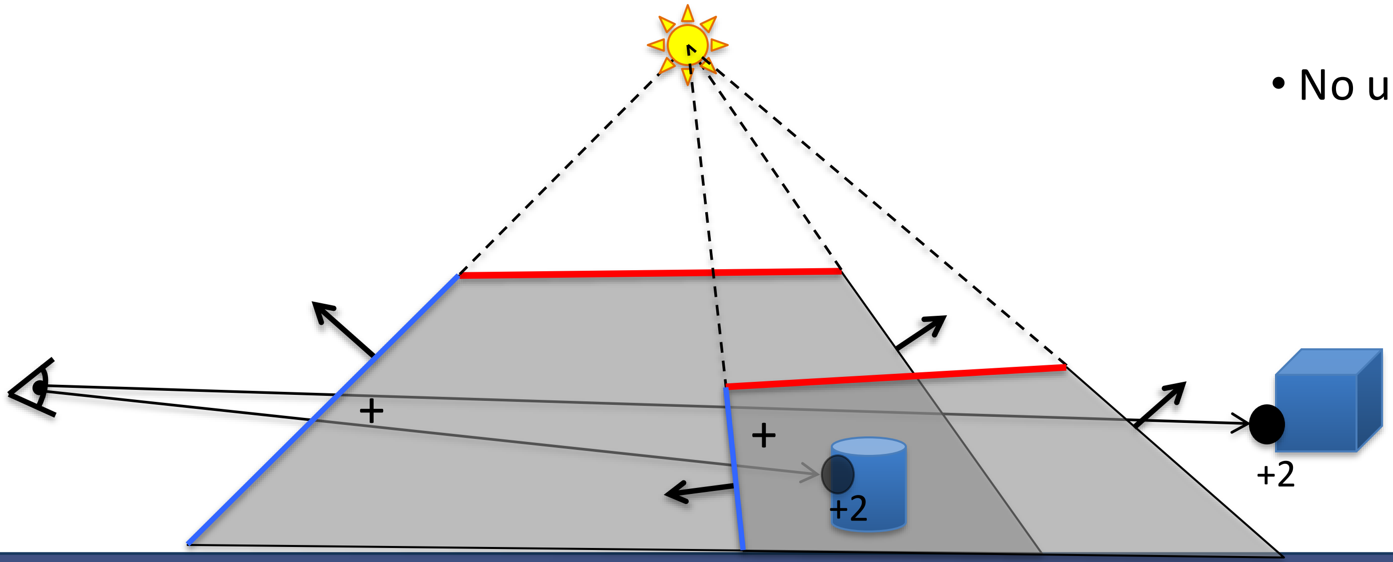
- Perform counting with the stencil buffer
  - Render front facing shadow quads to the stencil buffer
    - Inc stencil value, since those represents entering shadow volume
  - Render back facing shadow quads to the stencil buffer
    - Dec stencil value, since those represents exiting shadow volume



- No updating of z-buffer

Shadow Volumes - concept

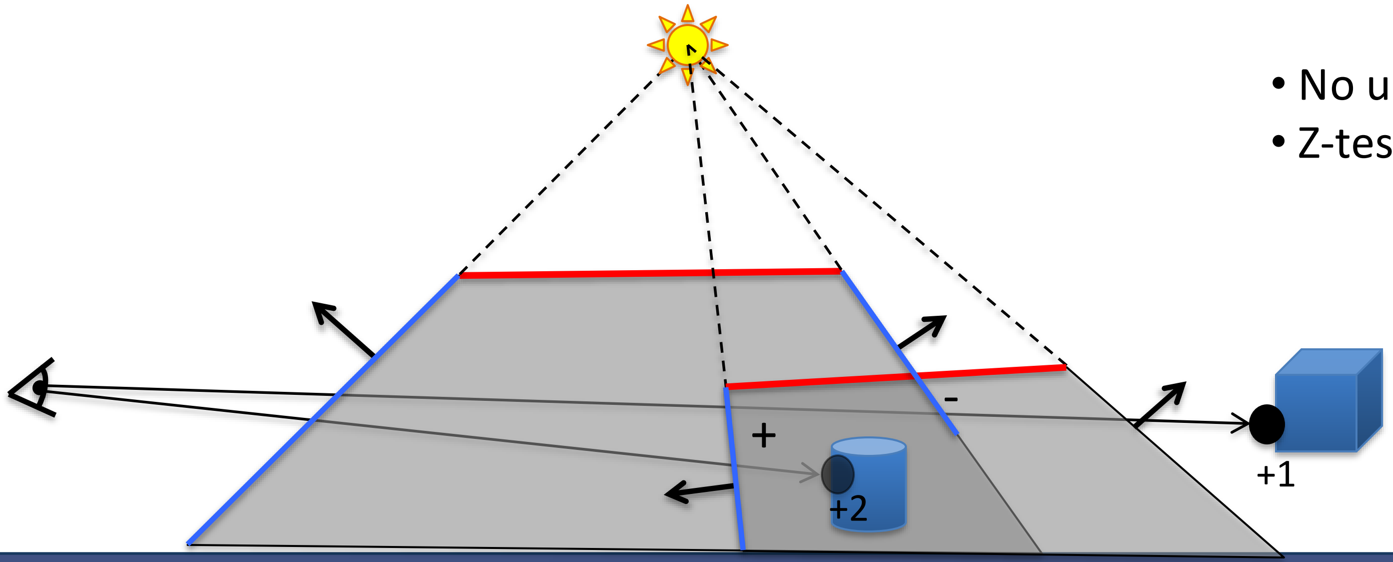
- Perform counting with the stencil buffer
  - Render front facing shadow quads to the stencil buffer
    - Inc stencil value, since those represents entering shadow volume
  - Render back facing shadow quads to the stencil buffer
    - Dec stencil value, since those represents exiting shadow volume



• No updating of z-buffer

Shadow Volumes - concept

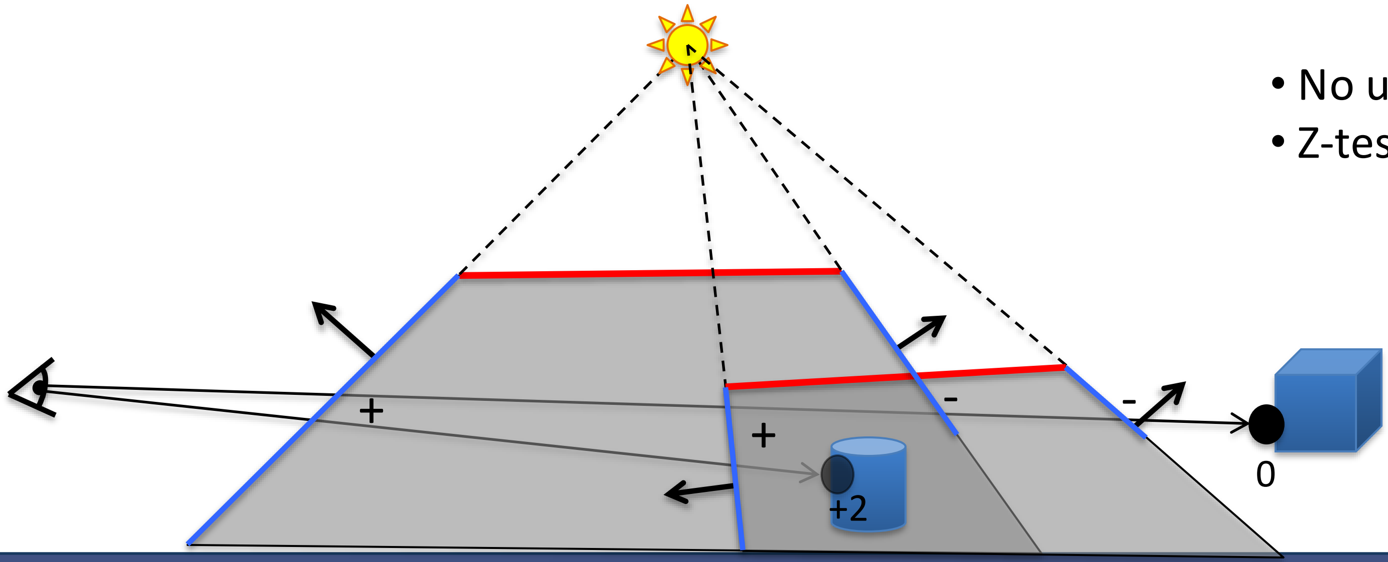
- Perform counting with the stencil buffer
  - Render front facing shadow quads to the stencil buffer
    - Inc stencil value, since those represents entering shadow volume
  - Render back facing shadow quads to the stencil buffer
    - Dec stencil value, since those represents exiting shadow volume



- No updating of z-buffer
- Z-test is enabled as usual

Shadow Volumes - concept

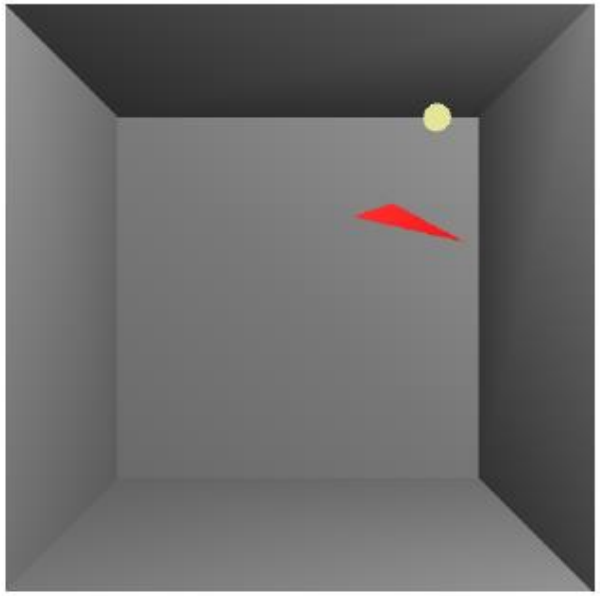
- Perform counting with the stencil buffer
  - Render front facing shadow quads to the stencil buffer
    - Inc stencil value, since those represents entering shadow volume
  - Render back facing shadow quads to the stencil buffer
    - Dec stencil value, since those represents exiting shadow volume



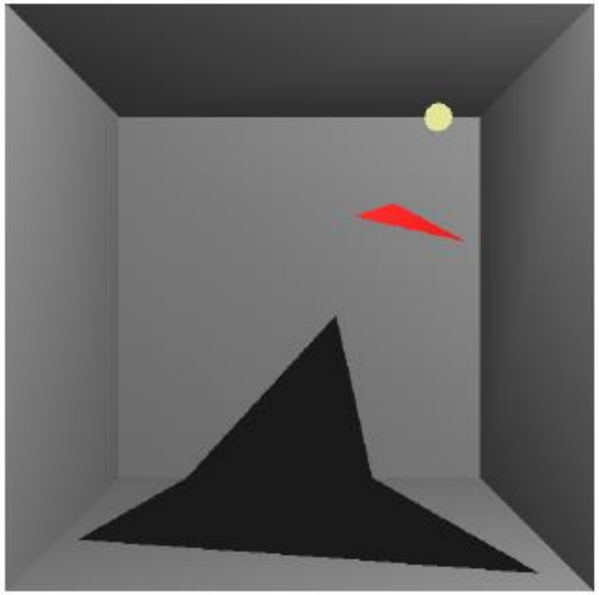
- No updating of z-buffer
- Z-test is enabled as usual

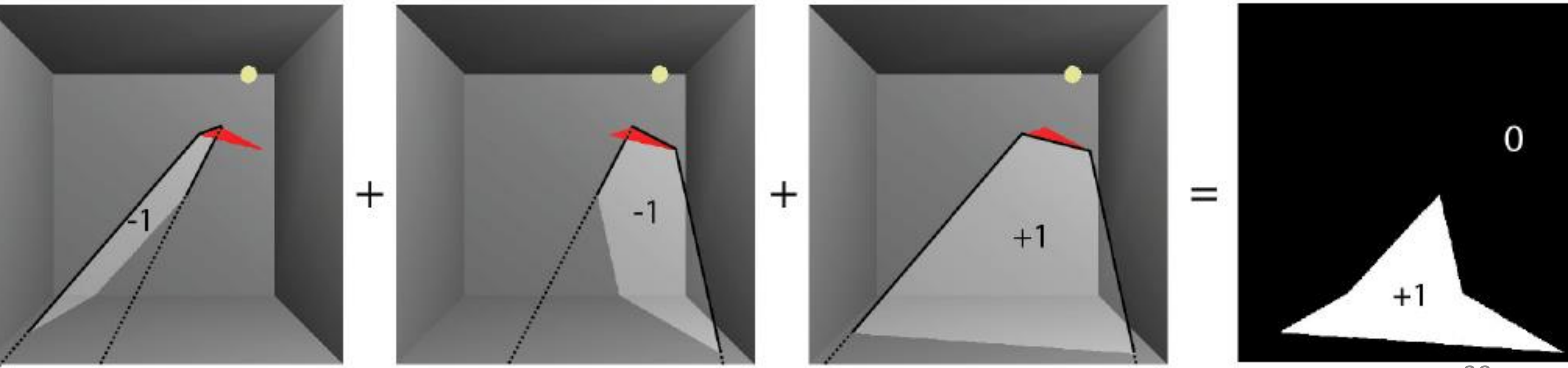
Z-pass by example: how the stencil buffer is used

What we have...



What we wnat...





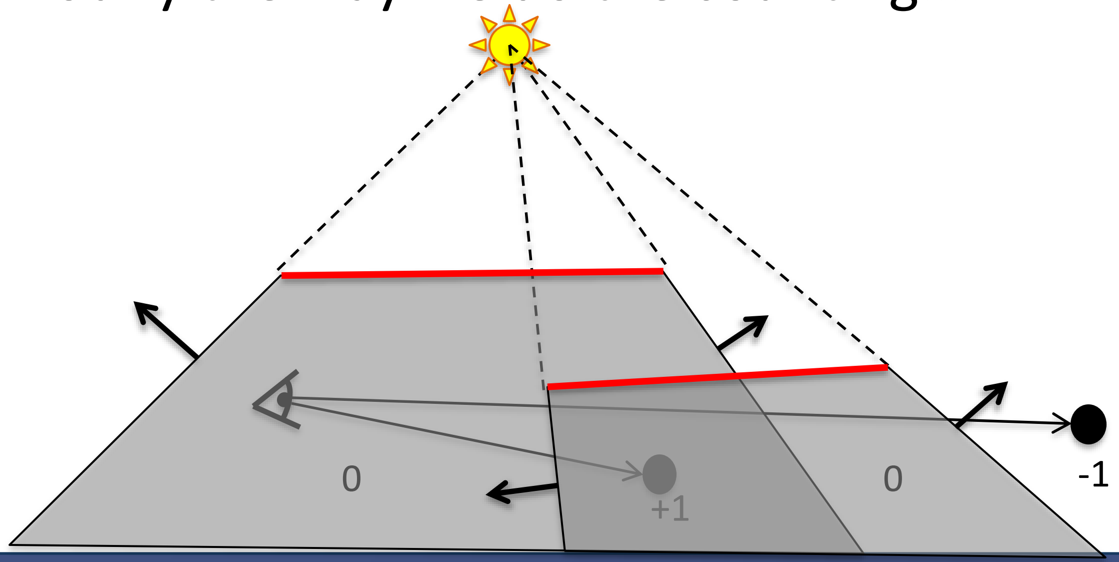
Shadow Volumes with the Stencil Buffer

- A three pass process:
  - **1st pass:** Render *ambient* lighting
  - **2nd pass:**
    - Draw to stencil buffer only
      - Turn off updating of z-buffer and writing to color buffer but still use standard depth test
      - Set stencil operation to
        - » *incrementing* stencil buffer count for *frontfacing* shadow volume quads, and
        - » *decrementing* stencil buffer count for *backfacing* shadow volume quads

        use **glStencilOpSeparate(...)**
  - **3rd pass:** Render *diffuse and specular* where stencil buffer is 0.

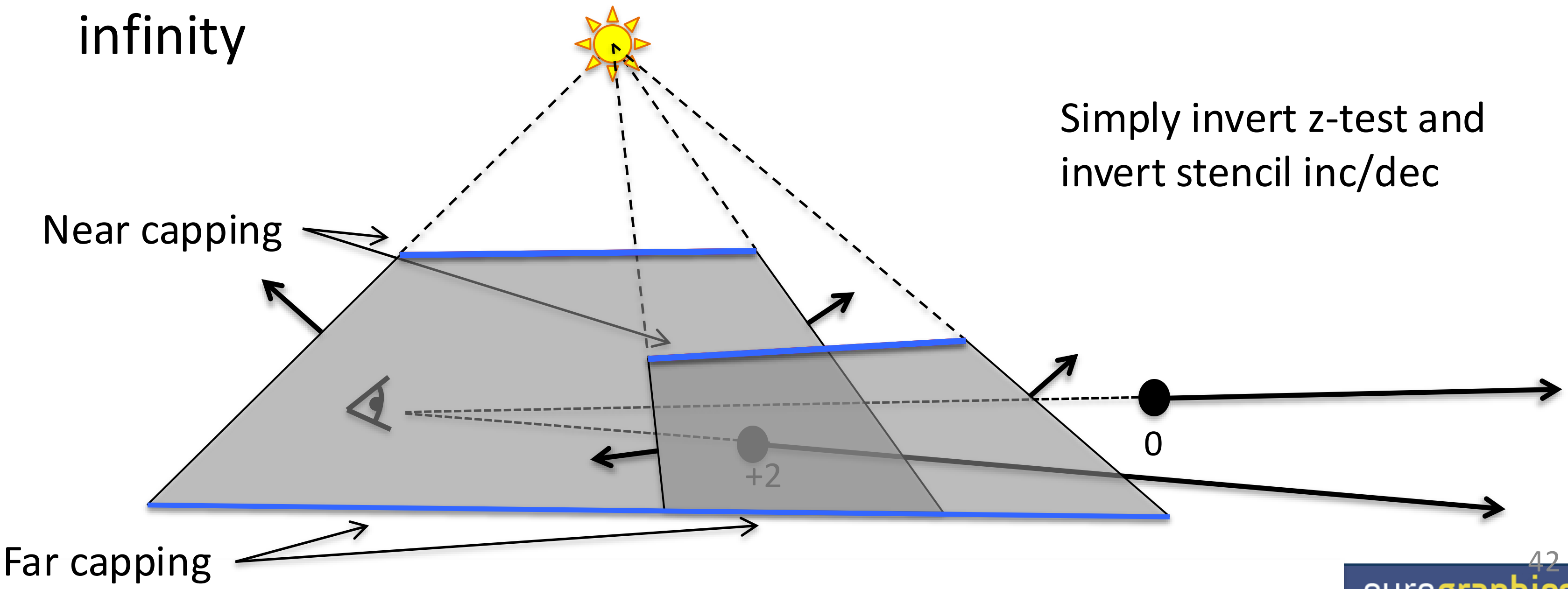
Eye Location Problem

- If the eye is located inside one or more shadow volumes, then the count will be wrong
- Solution:
  - Offset stencil buffer with the #shadow volumes that the eye is located within
  - Or modify the way we do the counting…

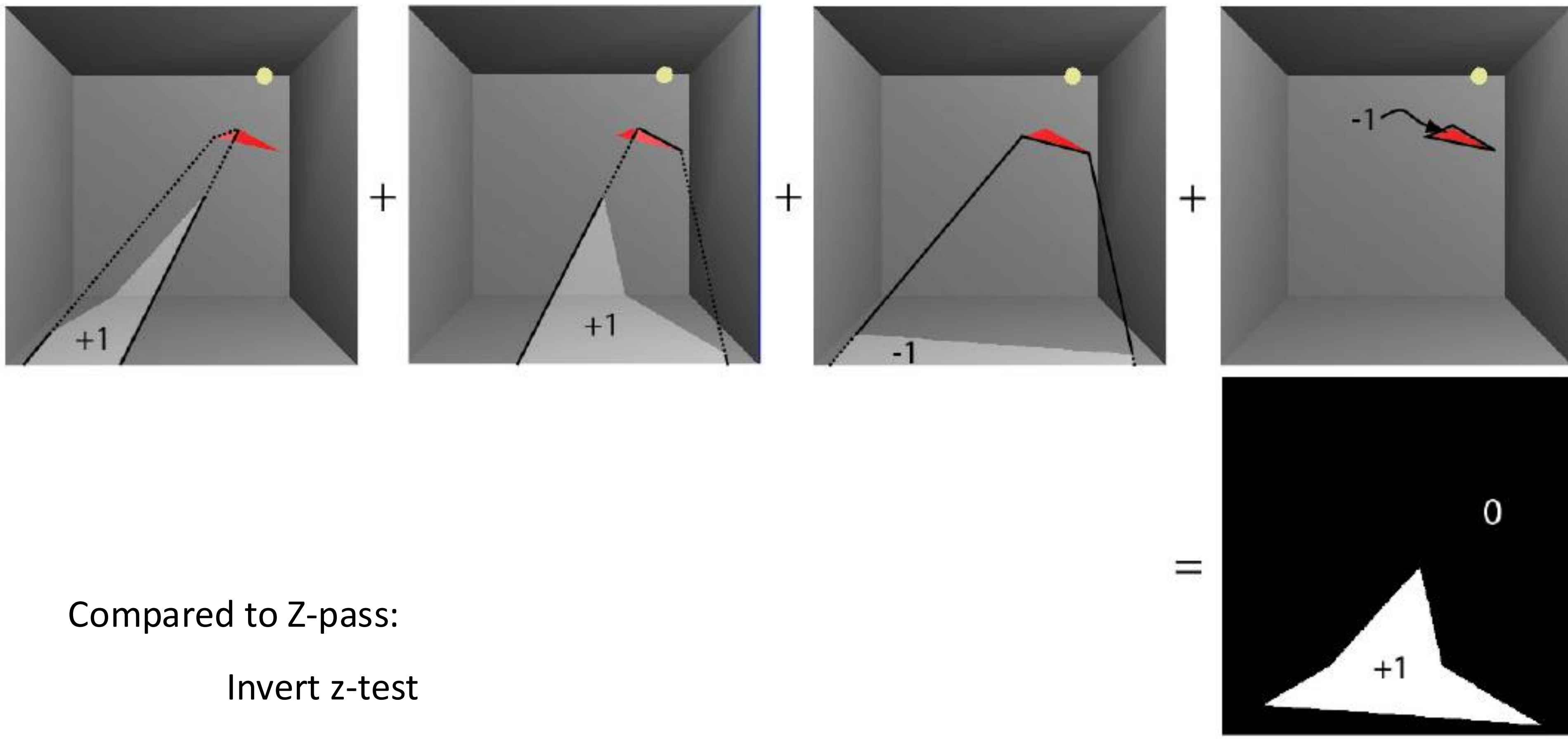


The Z-fail Algorithm

- By [Carmack00] and [Bilodeau and Songy 99]
  - “Carmacks Reverse”
- Count to infinity instead of to the eye
  - We can choose any reference location for the counting
  - A point in light avoids any offset
  - Infinity is always in light – if we cap the shadow volumes at infinity



Z-fail by example



Compared to Z-pass:

Invert z-test

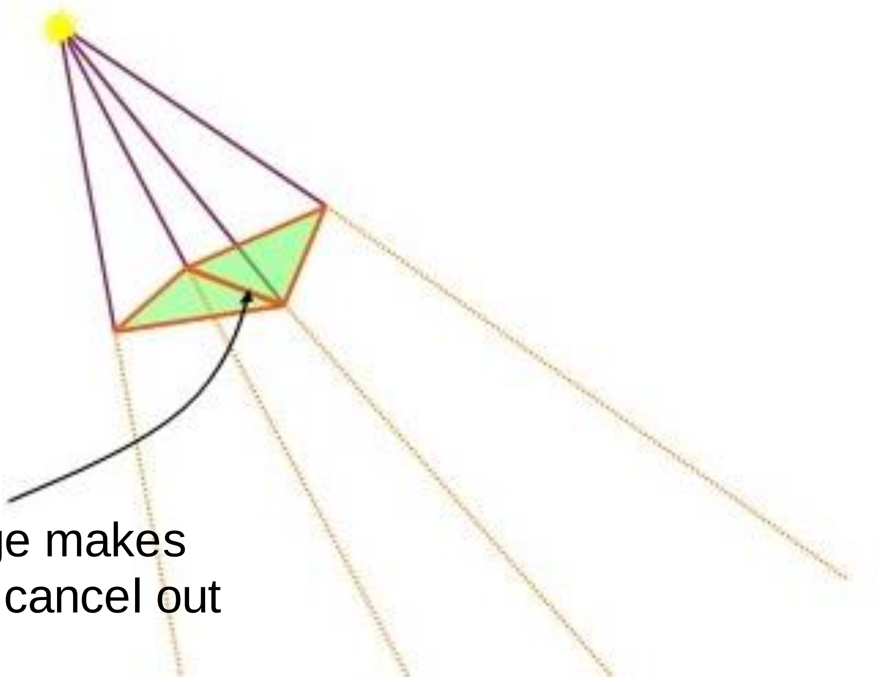
Invert stencil inc/dec

I.e., count to infinity instead of from eye.

Shadow Volumes from Silhouette Edges

Merging shadow volumes:

- An interior edge (non-silhouette edge as seen from the light position) creates two shadow quads that cancel each other out:

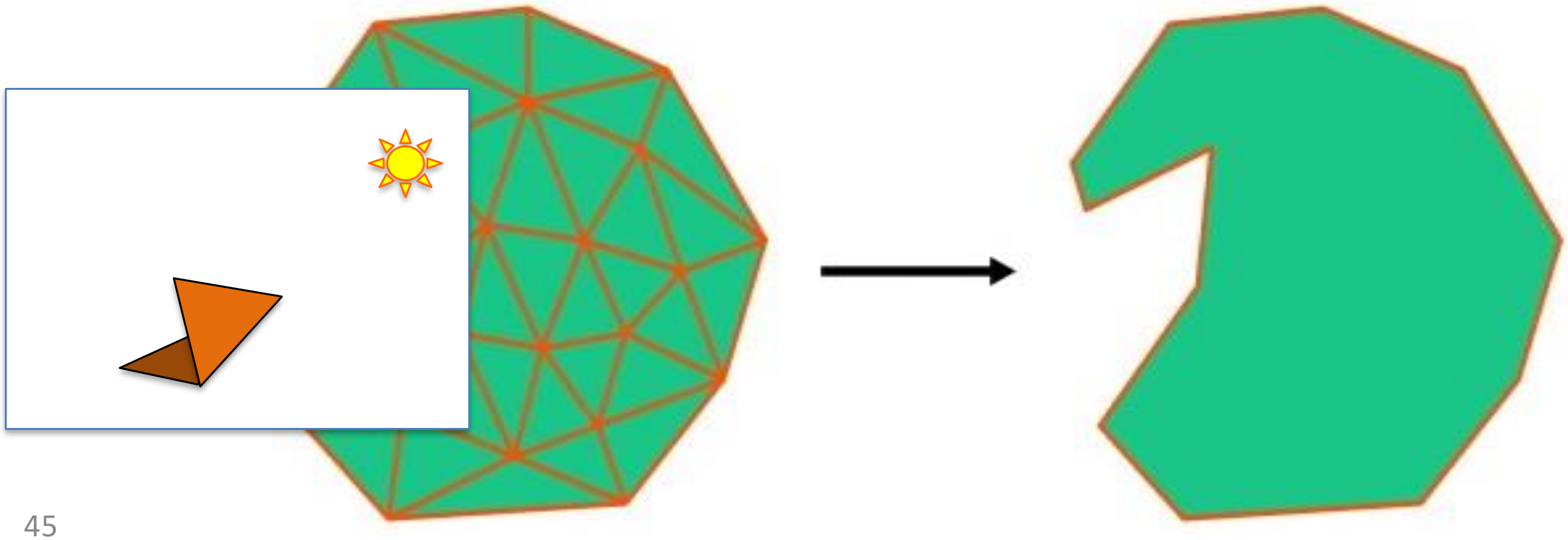


This interior edge makes two quads, which cancel out

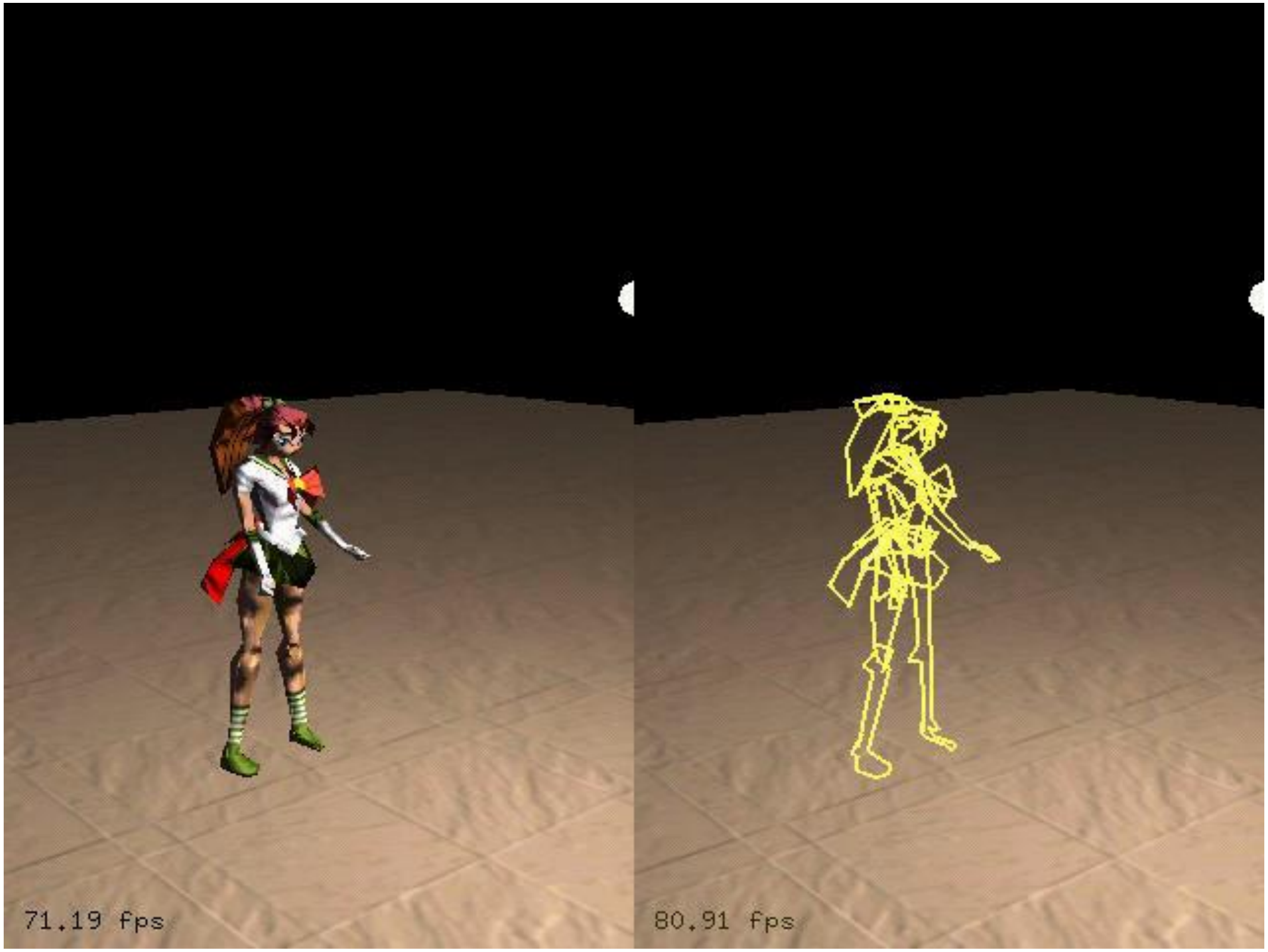
Shadow Volumes from Silhouette Edges

Merging shadow volumes:

- An interior edge (non-silhouette edge as seen from the light position) creates two shadow quads that cancel out each other:
- Thus, popular to create a shadow quad only per silhouette edge as seen from the light source.
  - (Slightly more care needed for non-closed objects…)
  - Avoids rendering of many useless shadow quads



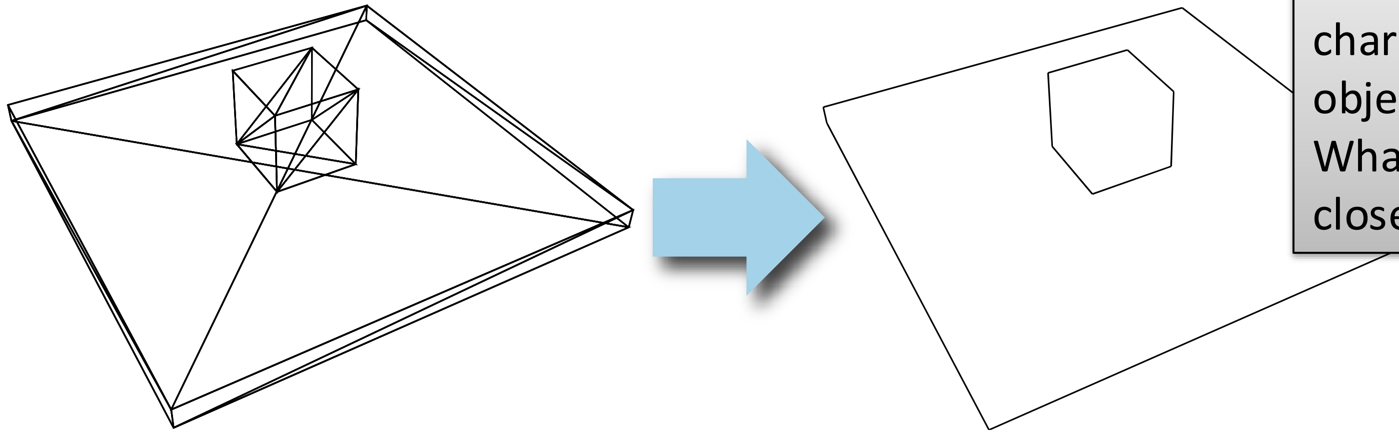
Example of silhouettes from light position



Shadow Volumes from Silhouette Edges

Merging shadow volumes:

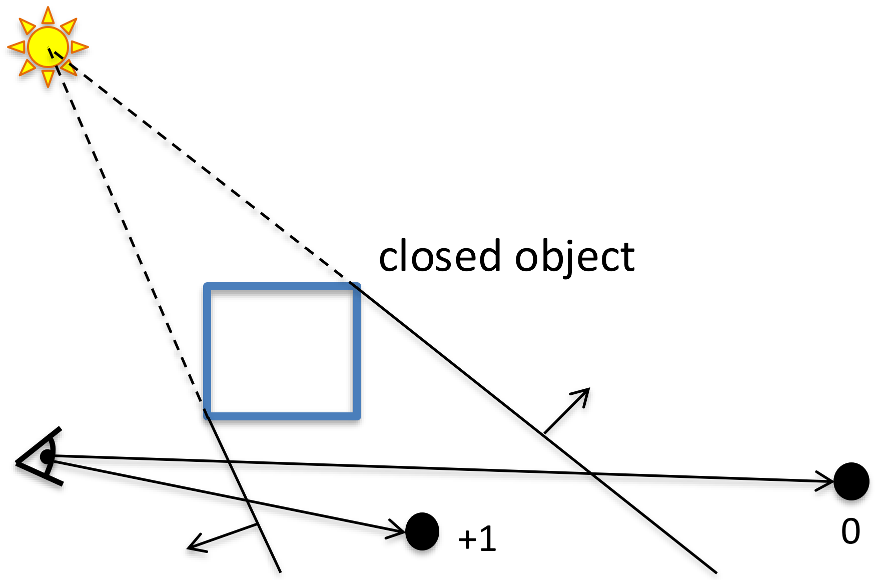
- An interior edge (non-silhouette edge as seen from the light position) creates two shadow quads that cancel out each other:
- Thus, popular to create a shadow quad only per silhouette edge as seen from the light source.
  - (Slightly more care needed for non-closed objects…)
  - Avoids rendering of many useless shadow quads
  - A real example:



This works like a charm for closed objects.
What about non-closed objects?

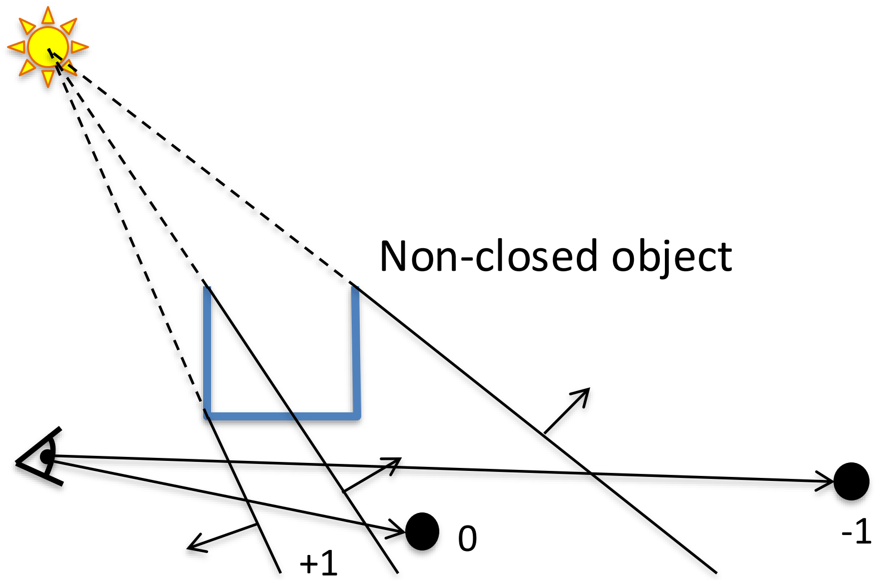
Shadow Volumes from Silhouette Edges

It is a misconception that objects **need** to be closed to remove non-silhouette edges.



Shadow Volumes from Silhouette Edges

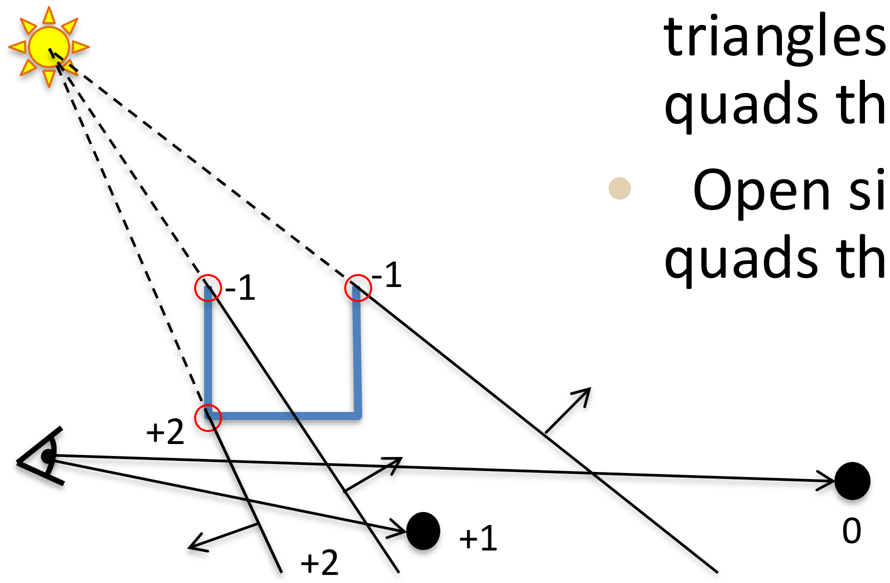
It is a misconception that objects **need** to be closed to remove non-silhouette edges.



Shadow Volumes from Silhouette Edges

It is a misconception that objects **need** to be closed to remove non-silhouette edges.

Fixed by [Bergeron 86]



Observation:

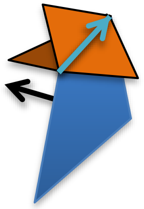
- Silhouette edges with two adjacent triangles should actually create shadow quads that inc/dec count by 2
- Open silhouette edges create shadow quads that inc/dec count by one

Stencil value >0 means shadow

Works identically for Z-fail

Shadow Volumes from Silhouette Edges

For general objects with edges that can be shared by many triangles:



Preprocess (or in geometry shader):

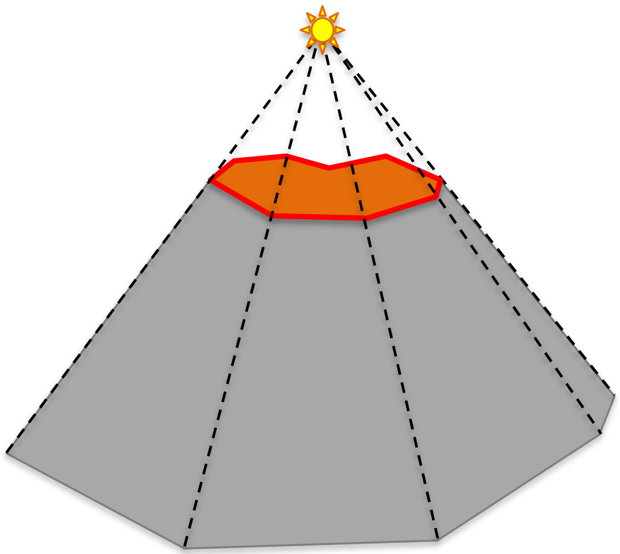
- For each triangle edge ***e*** in scene:
  - Choose edge ***e***'s direction
    - Create ***e***'s shadow volume quad
    - Let ***e*** have a counter $c_e = 0$
    - For each adjacent triangle, t:
      - Inc/dec c_e depending on if triangle t's created shadow volume quad would have same/opposite facing of ***e***'s quad.
    - Add quad {***e,*** c_e} to list L, if c_e != 0.

At rendering:

- Render all quads in L, and inc/dec stencil by the quad's c_e depending on if quad is front/back-facing eye.
- For 100% robustness, see our *book Real-Time Shadows*

Shadow Volumes - Summary

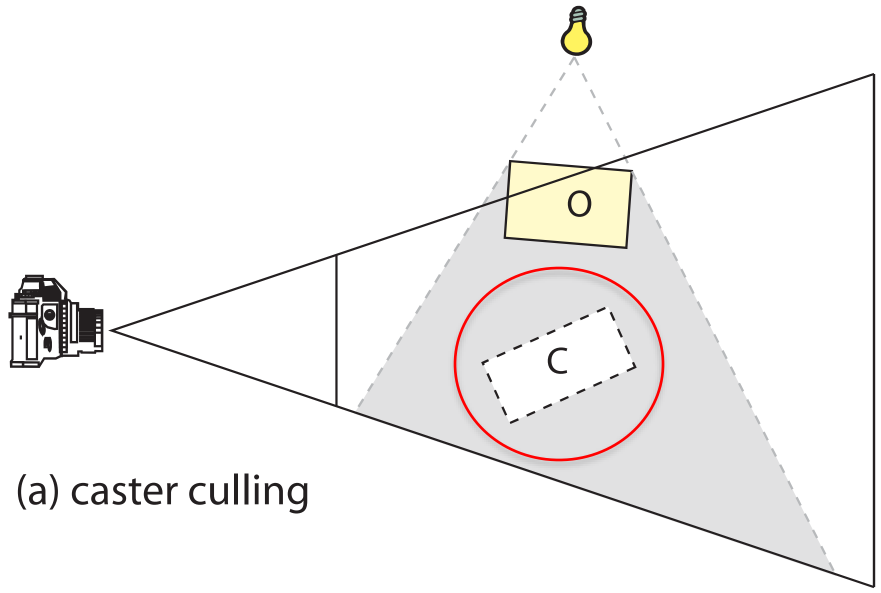
- Pros:
  - High quality
- Cons:
  - OVERDRAW



Bonus

Culling and Clamping

- **Culling of Shadow Volumes** [Lloyd et al. 2004][Stich et al. 2007]
  - **Culling of Shadow Casters** if it is located totally within shadow
    - Tested against a shadow depth map

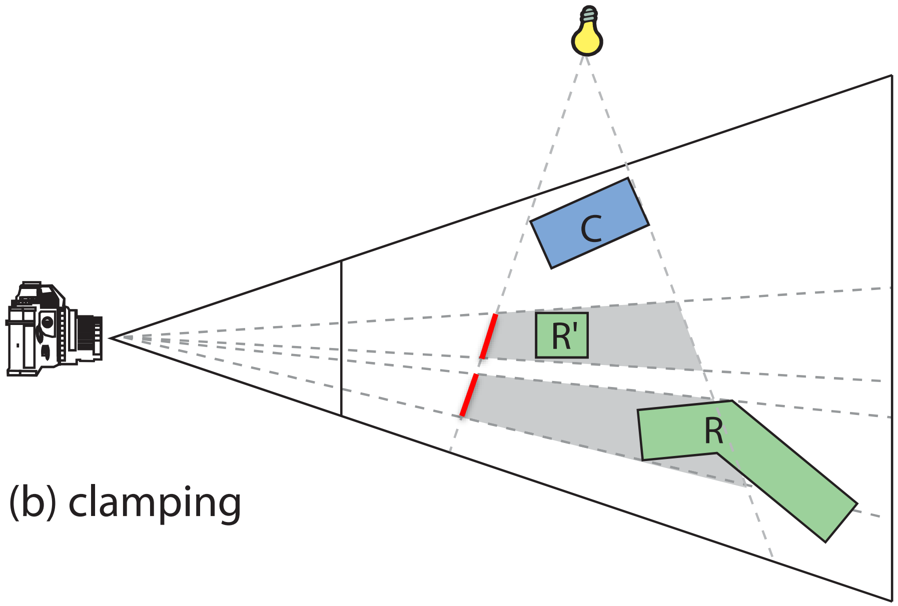


(a) caster culling

Bonus

Culling and Clamping

- **Clamping of Shadow Volumes** [Lloyd et al. 2004][Eisemann and Decoret 2006]
  - Idea: Only render parts of shadow quads that affects a shadow receiver
    - Tested against AABB around shadow receivers

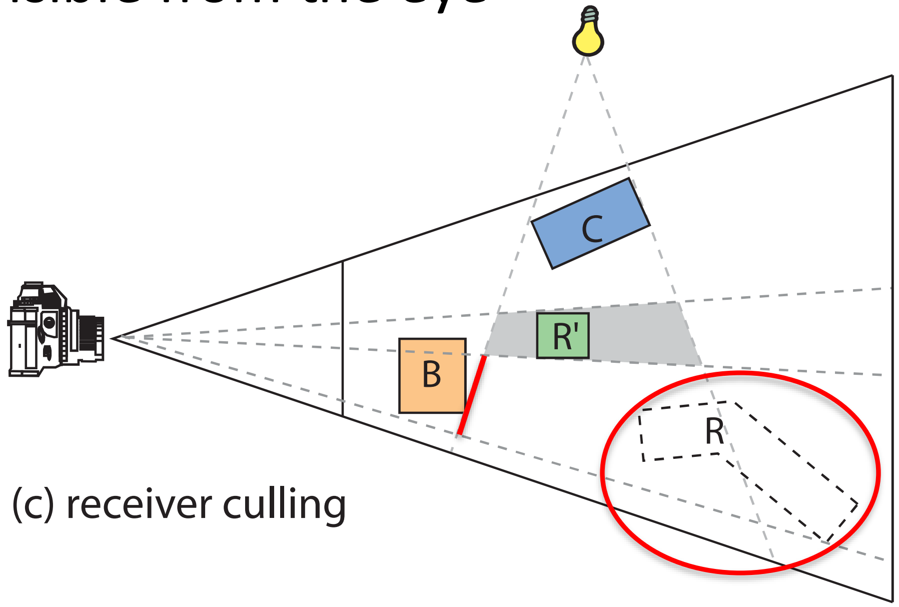


(b) clamping

Bonus

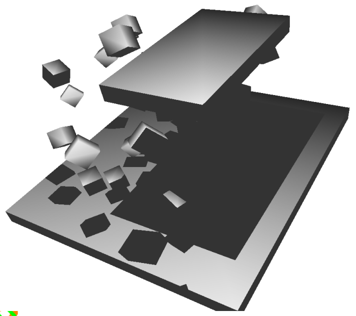
Culling and Clamping

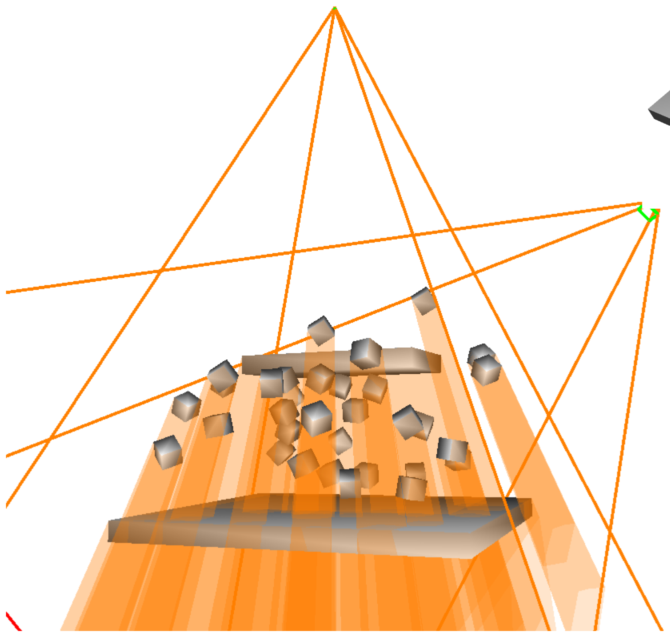
- **Culling of Shadow Volumes** [Lloyd et al. 2004][Eisemann and Decoret 2006]
  - **Receiver Culling**
    - Idea: Cull part of shadow volumes where shadow receivers are not visible from the eye



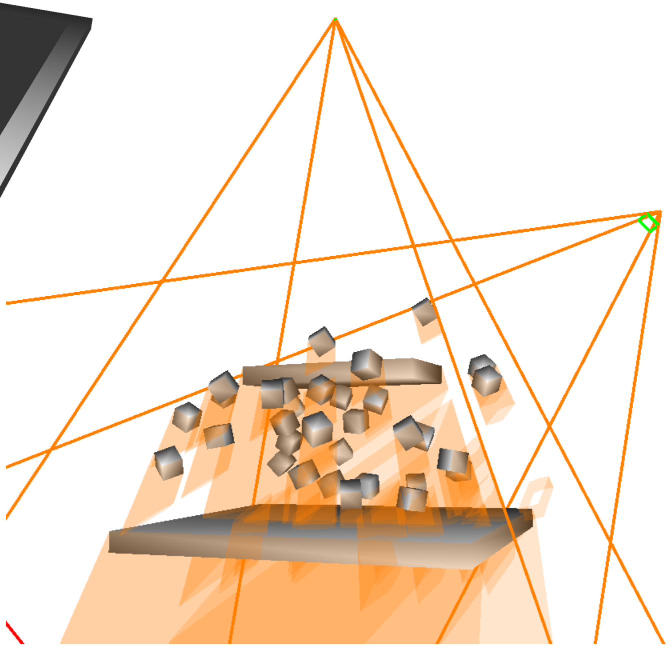
(c) receiver culling

Culling and Clamping





Shadow volumes

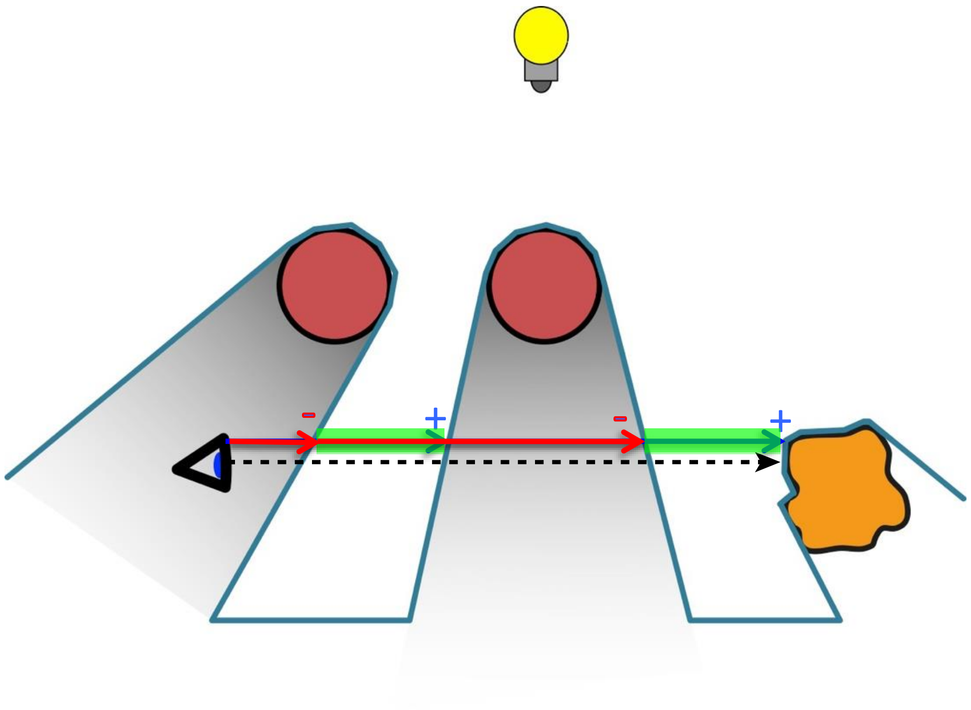


CC Shadow volumes

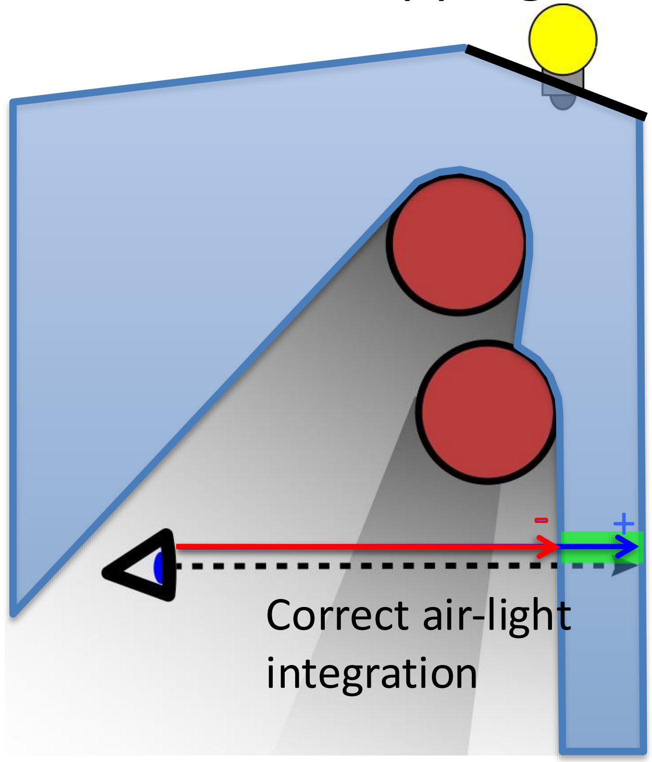
Illustrates reduced depth complexity when using Culling and Clamping

Volumetric Lighting

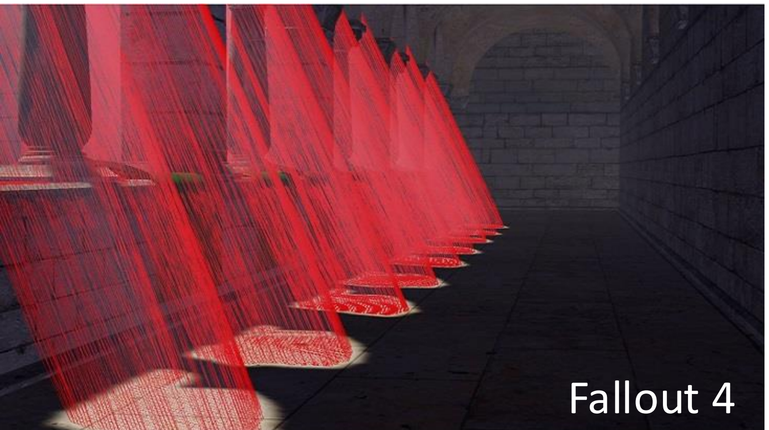
- Shadow volumes can be used for "God rays"/Shafts of light/volumetric lighting/participating media.
  - *Volumetric Shadows using Polygonal Light Volumes,* Billeter et al, 2010.
  - Part of NVIDIA Volumetric Lighting SDK

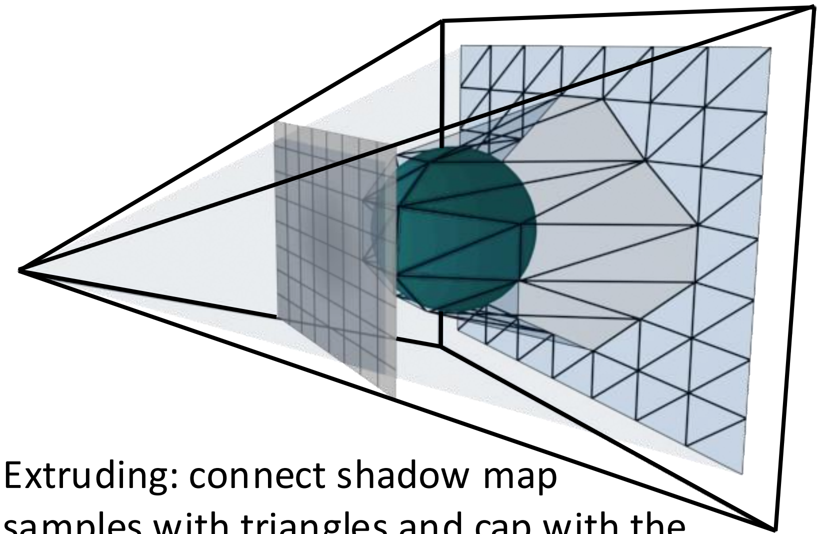


- For correctness, extrude light volumes from the shadow map, to avoid overlapping volumes:



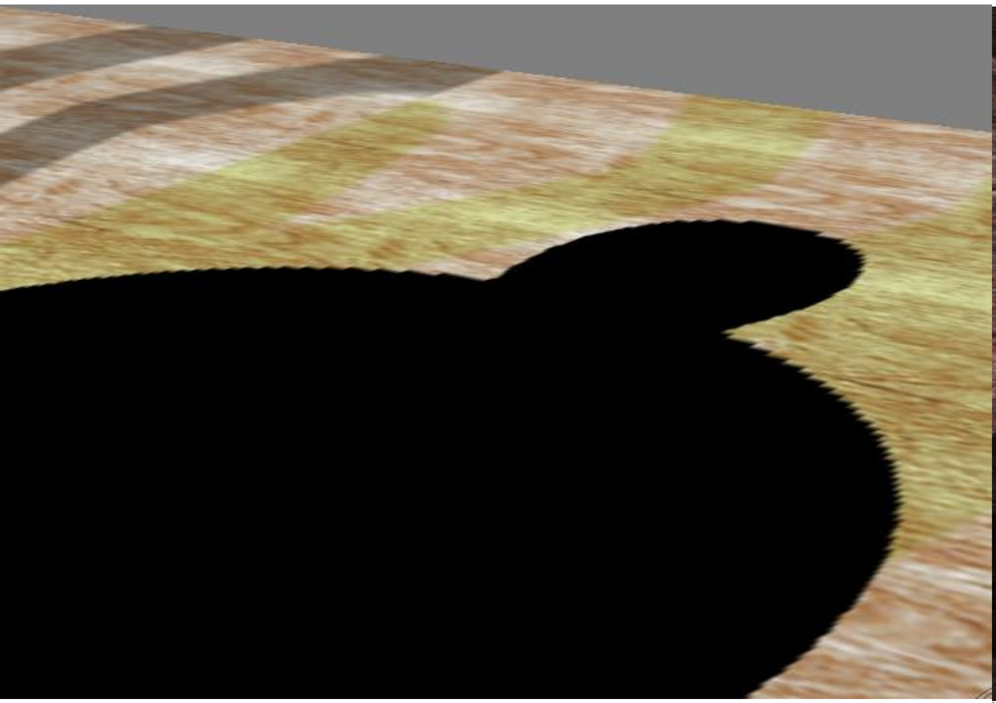
Correct air-light integration


Fallout 4



Extruding: connect shadow map samples with triangles and cap with the left+right+top+bottom frustum planes
→ encloses volume in light

Shadow Maps vs Shadow Volumes


Shadow Maps

- *Good*: Handles any rasterizable geometry, **constant cost** regardless of complexity, map can sometimes be reused. **Very fast**.
- *Bad*: Frustum limited. **Jagged shadows** if res too low, **biasing** headaches.
  - Solution:
  - 6 SM (cube map), high res., use filtering (huge topic)

Shadow Volumes

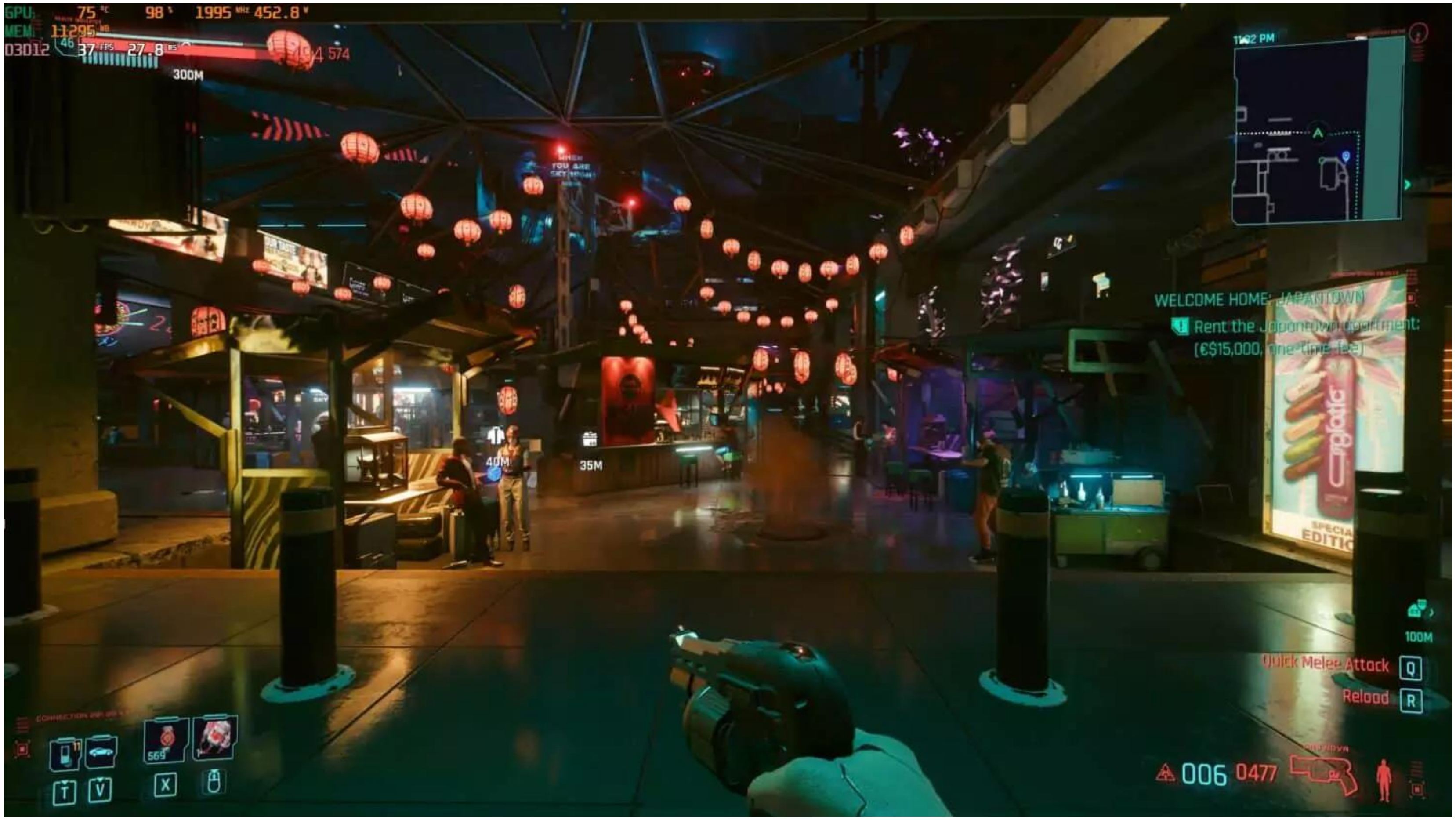
- *Good*: shadows are **sharp**. Handles omni-directional lights.
- *Bad*: **3 passes**, shadow polygons must be generated and rendered → lots of polygons & **fill**
  - Solution: culling & clamping (or per-triangle SV using hierarchical shadow buffer)

Shadow Maps vs Shadow Volumes

- Shadow volumes: popular in games up to ~2005, e.g.,
  - DOOM 3, 2004.
  - Far Cry (shadow volumes are used indoors, shadow maps - outdoors), 2004.
  - The Chronicles of Riddick: Escape from Butcher Bay. 2004.
  - Spiderman 3 (Activision), 2007.
- Shadow maps are more popular today due to speed and ease of filtering for soft-shadows.
  - E.g., DOOM (5) Eternal, 2020. 4096x8196px 24-bit shadow map.

The future – ray traced shadows?

- For only few point lights, shadow maps are attractive due to speed.
- For many lights or area/volumetric lights, tracing shadow rays + AI denoising is attractive.
  - E.g., for thousands/millions of lights with a few shadow rays per pixel, see versions of the "*ReSTIR*" method. Trick: weight shadow-ray samples smarter (importance sampling), incl. for adjacent pixels and frames. See for instance: https://www.youtube.com/watch?v=gsZiJeaMO48&list=LL&index=1

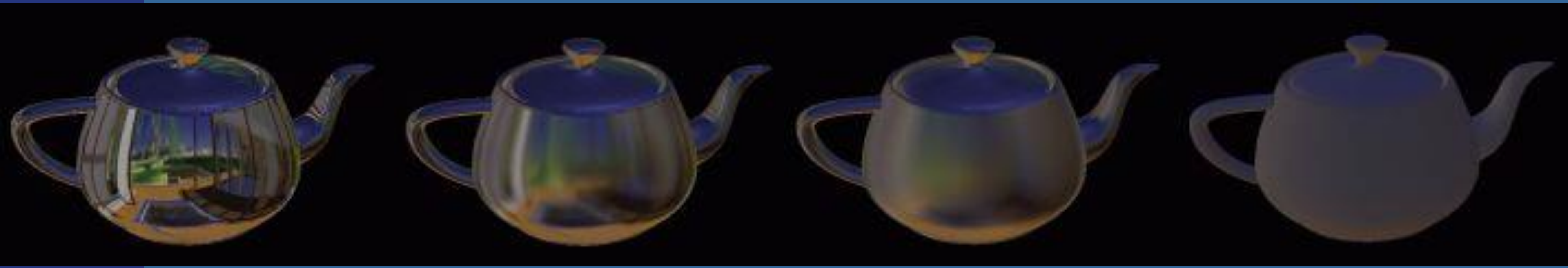


- But GPU ray tracing is still expensive for real time, so…

Reflections

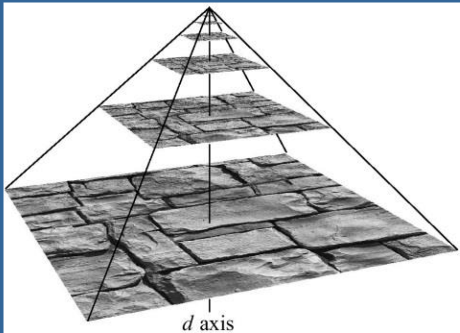
Misc

- Clamp the minimum (finest) lod level to the amount of blur you need.



E.g., via

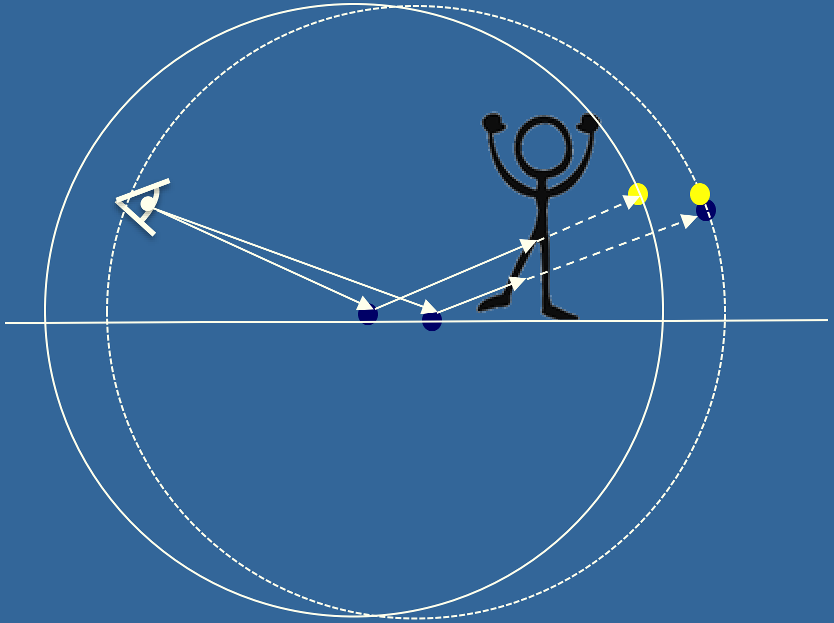
- **glTexParameterf(GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_MIN_LOD, lambda);**
- or control it manually in your shader:
  - lod=max(min_lod, lod_level);
  - textureLod(tex, uv, lod);

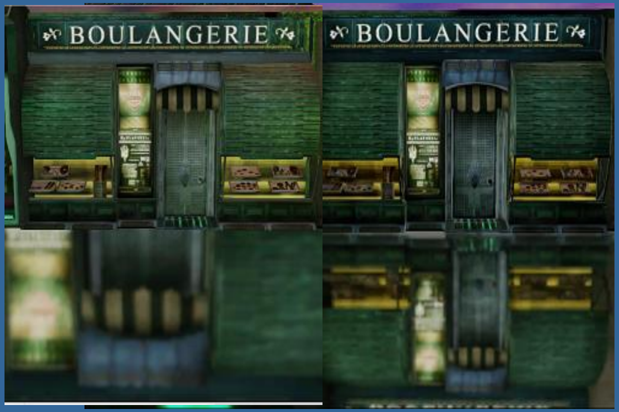


Planar reflections

- We’ve already done reflections in curved surfaces with environment mapping. But the env.map is assumed to have an infinite radius, such that only the reflection ray’s direction (not origin) matters. Hence…
- …Environment maps does not work well for reflections in planar surfaces:

For two adjacent screen pixels, the cube map returns a too small uv change. Hence the reflection will be smeared out.





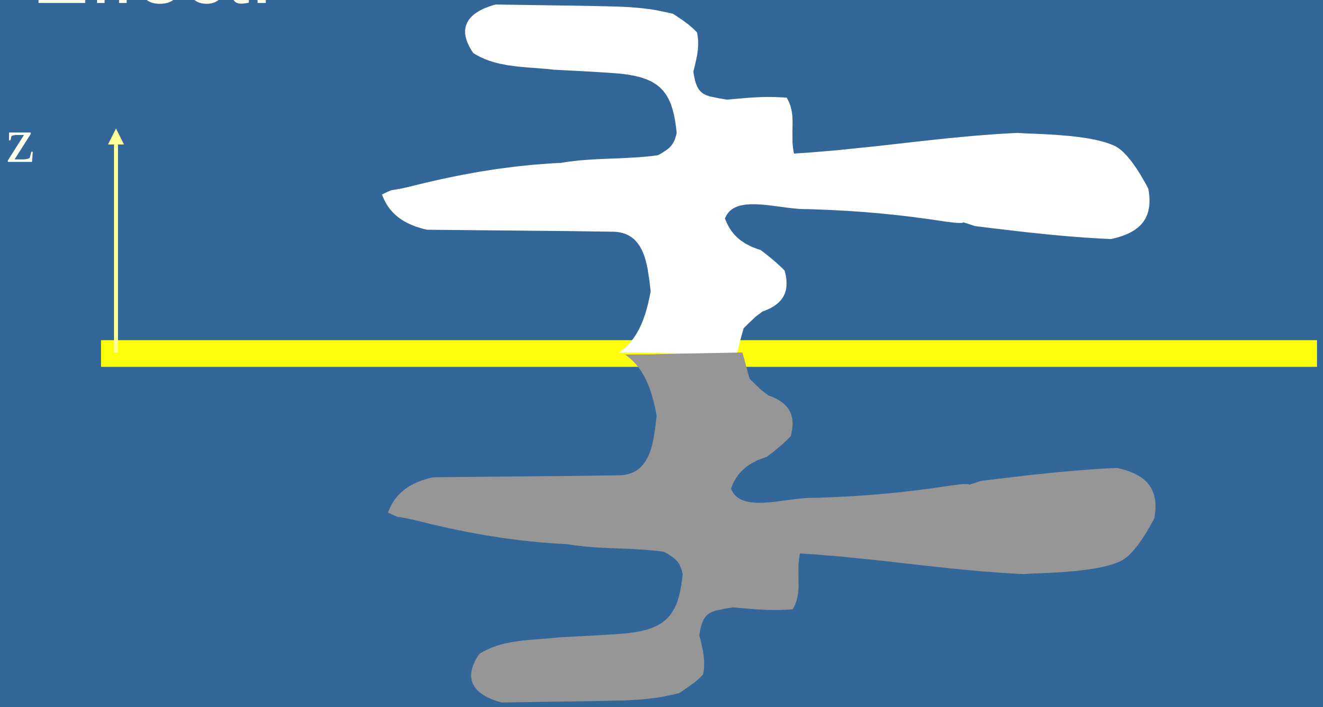
Standard cube map (smear in xy)

Parallax corrected (no smear)

- Parallax corrected cube maps fix this, but has its own problems. Ray tracing solves all but is slower. Purely planar reflections are actually easy to get by reflecting the geometry or camera as we will see on the next slide…

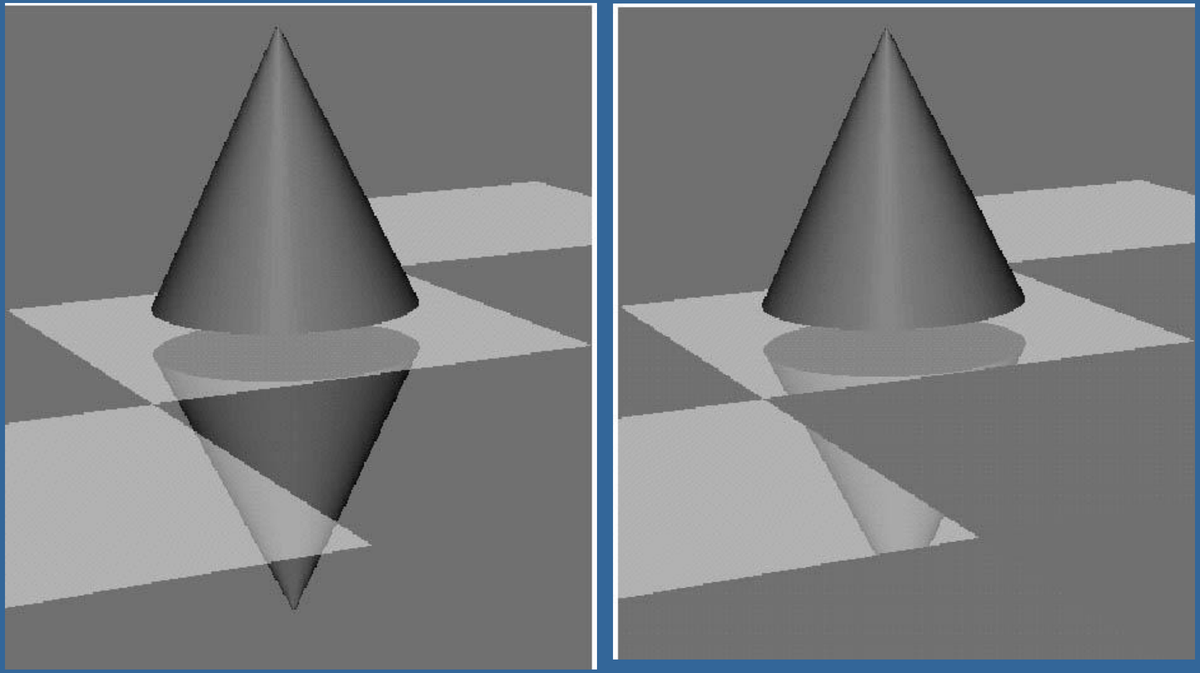
Planar reflections

- Assume plane is z=0
- Then apply a scaling matrix **S**(1,1,-1);
- Effect:

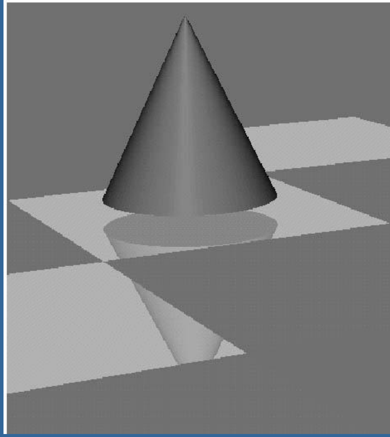


Planar reflections

- Backfacing becomes front facing!
- Lights should be reflected as well
- (May need to clip using stencil buffer)
- See example on clipping:



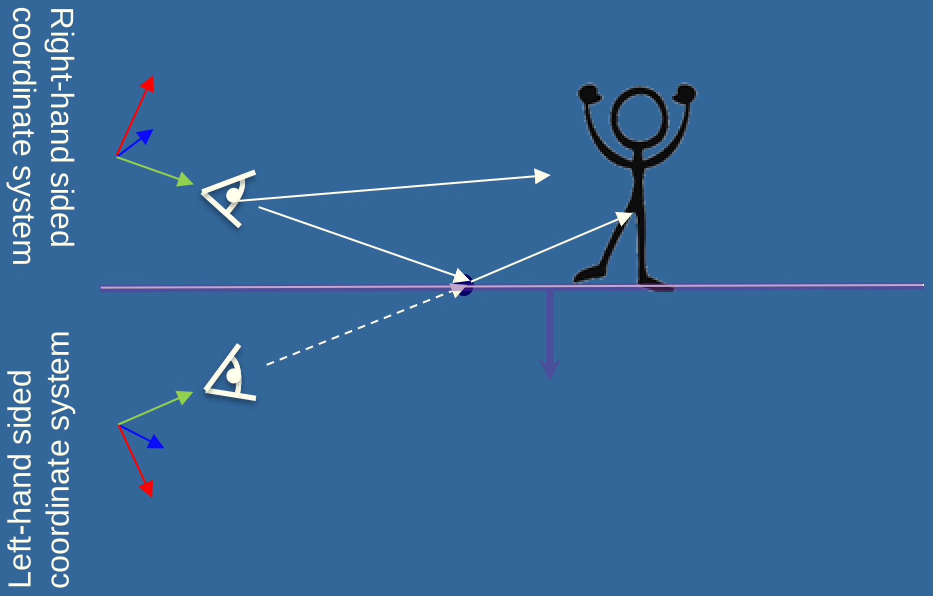
Planar reflections

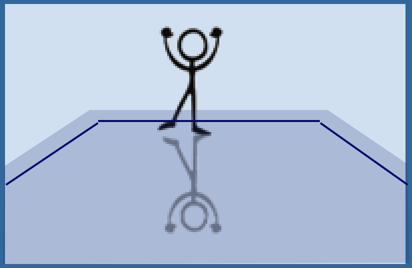


- How should you render?
- 1) the reflective ground plane polygons into the stencil buffer
- 2) the scaled (1,1,-1) model, but mask with stencil buffer
  - Reflect light pos as well
  - Use front face culling
- 3) the ground plane (semi-transparent)
- 4) the unscaled model

Important:
- render scaled (1,1,-1) model
- with reflected ligh pos.
- using front face culling

Or reflect camera position instead of the object:


Right-hand sided coordinate system
Left-hand sided coordinate system



- Render reflection:
  1. Render reflective plane to stencil buffer
  2. Reflect camera including camera axes ← The important part!
  3. Set user clip plane in mirror plane to cull anything between mirror and reflected camera
  4. Render scene from reflected camera.
- Render scene as normal from original camera

A real example:

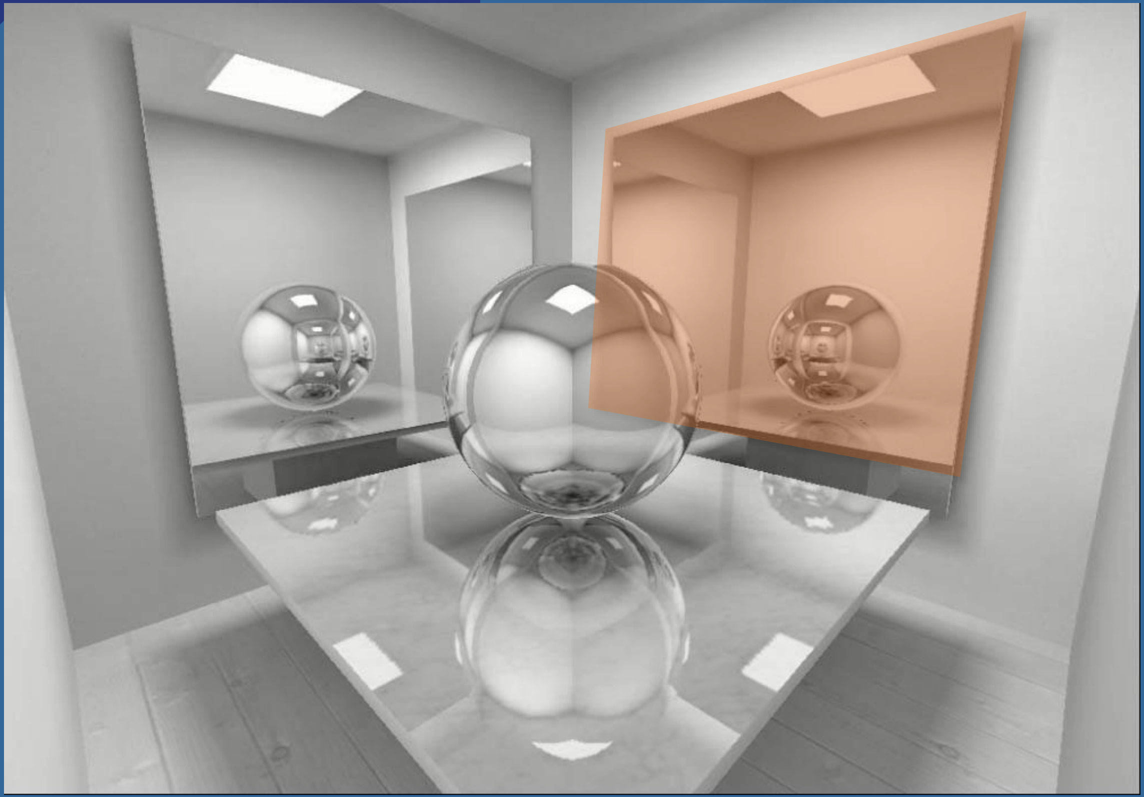


Image courtesy of Kasper Hoey Nielsen

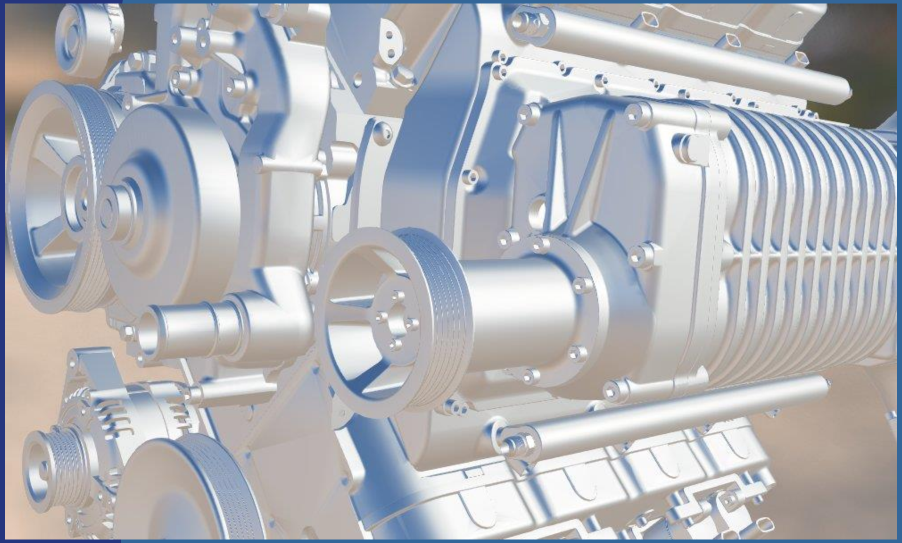
1. Render mirror to stencil buffer
2. Reflect camera (including cam axes)
3. Set user clip plane in mirror plane to cull anything between mirror and reflected camera
4. Render scene to screen.

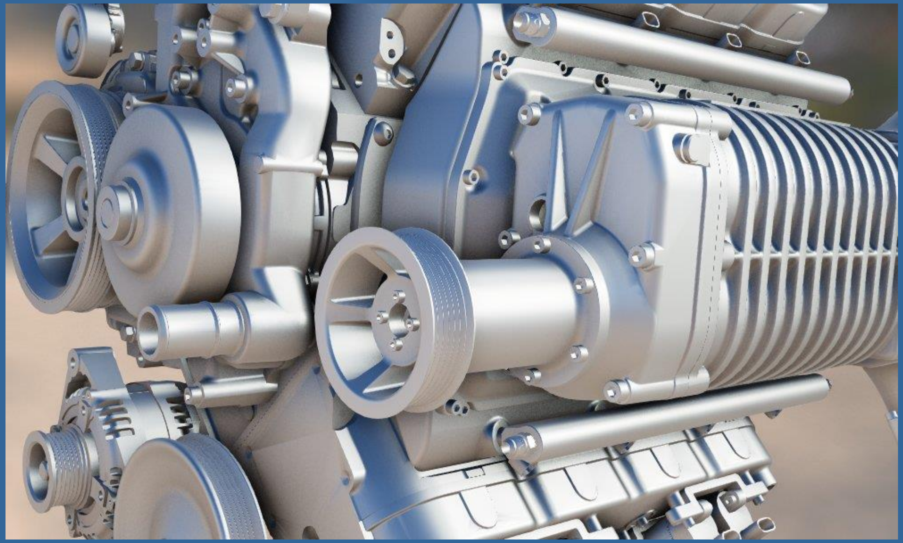
Study Questions

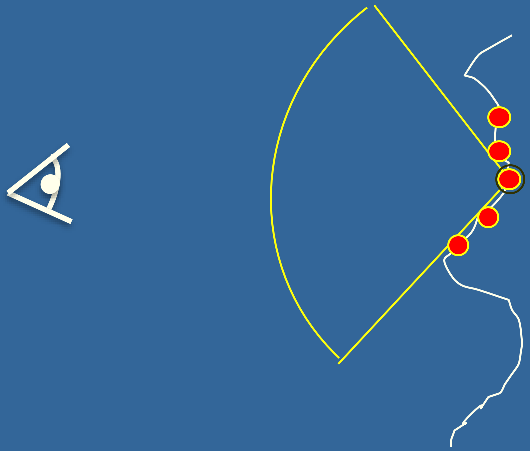
- What is “Planar shadows”
  - Answer: you project the objects’ triangles onto the plane and draw them with dark color.
- Explain shadow maps
- Explain shadow volumes
  - Both z-pass and z-fail
- What are the pros and cons of shadow maps vs. shadow volumes?
- Why are environment maps problematic for planar reflections?
- How can you render planar reflections?

Bonus slides…

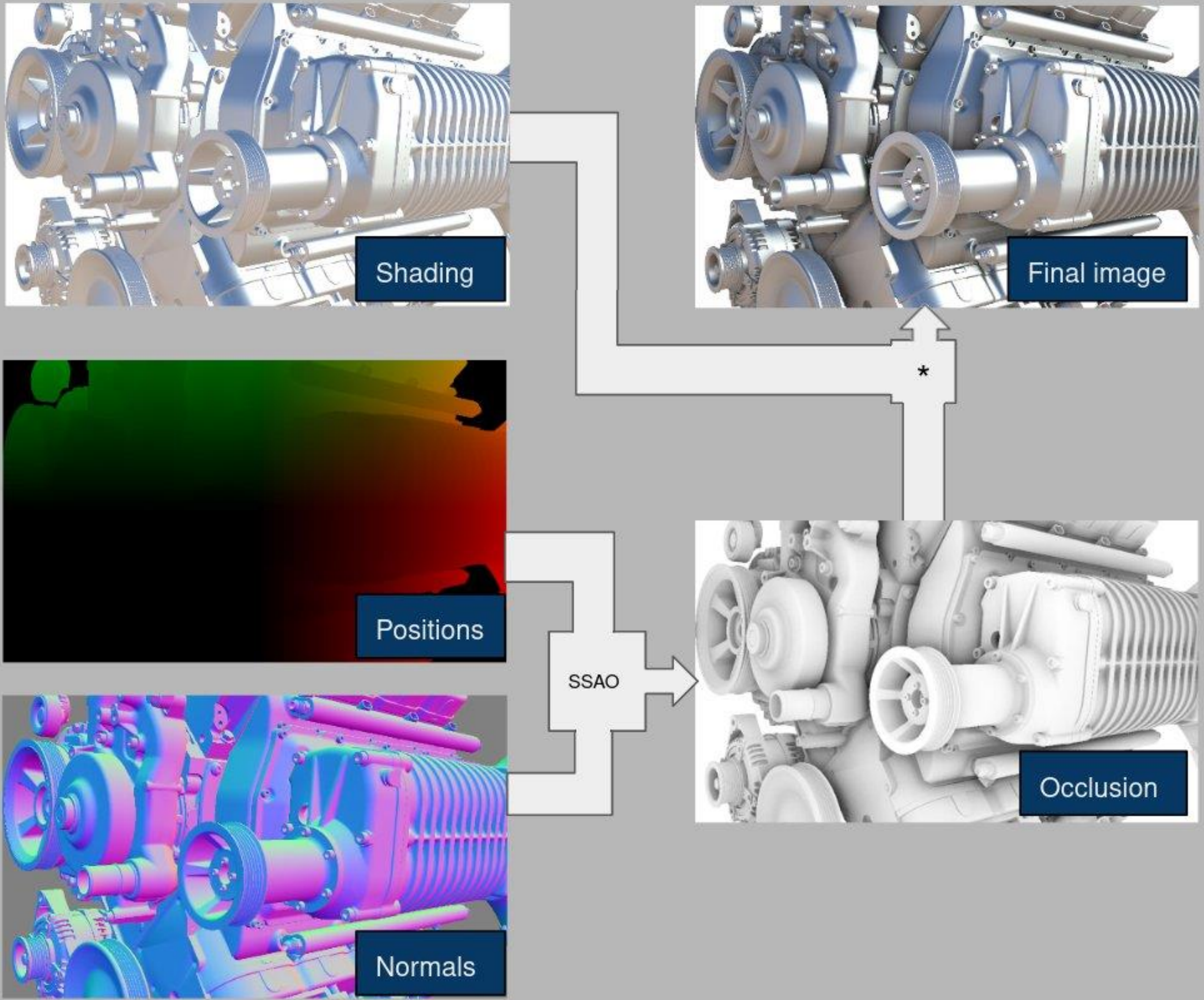
Screen-space Ambient Occlusion


Without SSAO


With SSAO

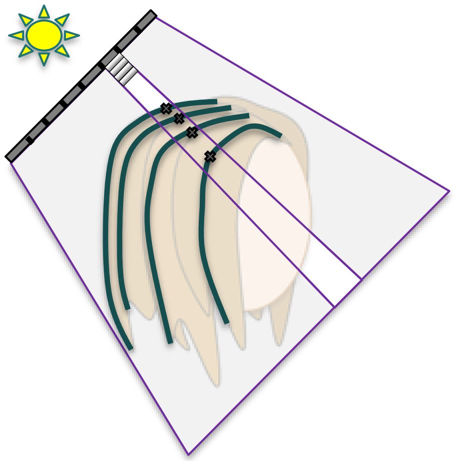


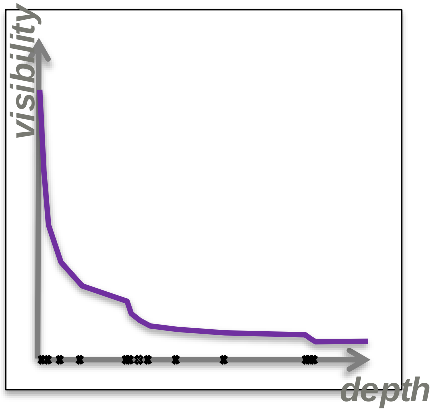
Use the z buffer to, for each pixel, estimate how much of the hemisphere that is non-blocked for incoming light. (See Labs – SSAO Project)


Shading
Final image
*
Positions
SSAO
Occlusion
Normals

Deep Shadow Maps

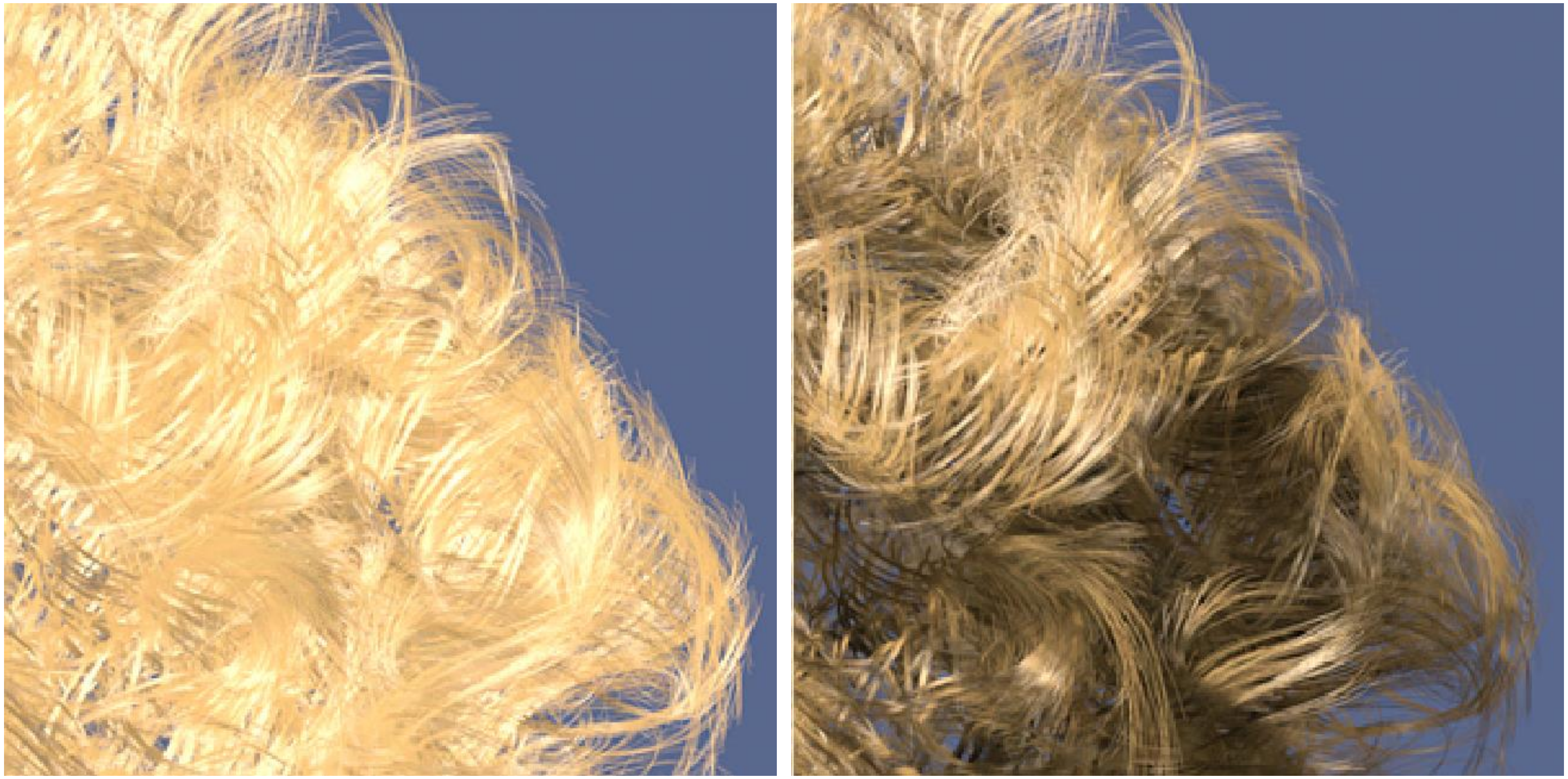
- Pixar
  - Lokovic and Veach, Siggraph 2000.
  - Minutes per frame
  - Monster's Inc, 2001





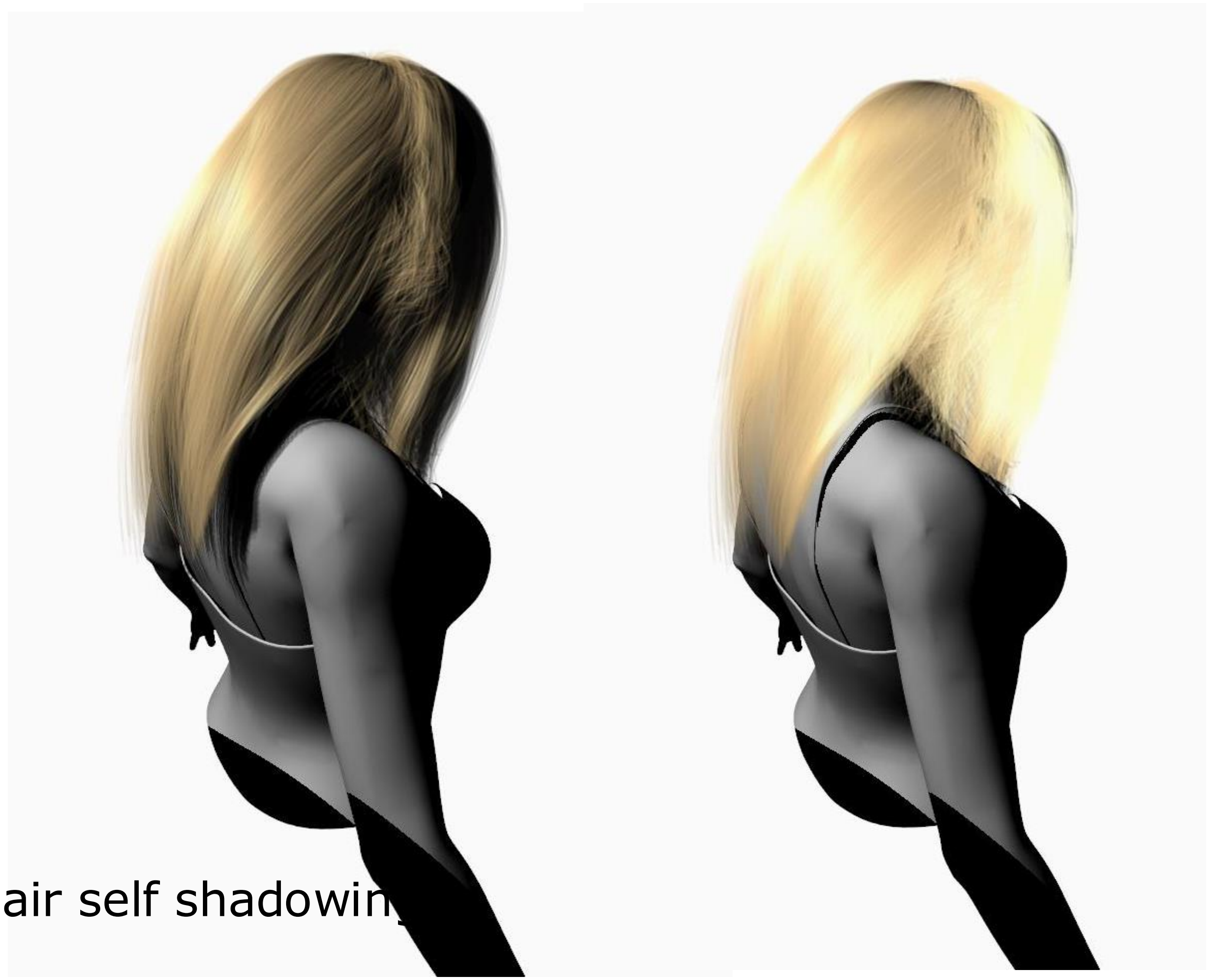
Each shadow-map texel holds a shadow/visibility function of depth from light.

Importance of Shadows



Images from: Tom Lokovic and Erich Veach, “*Deep Shadow Maps*”*, pp 385-392, Siggraph 2000.*

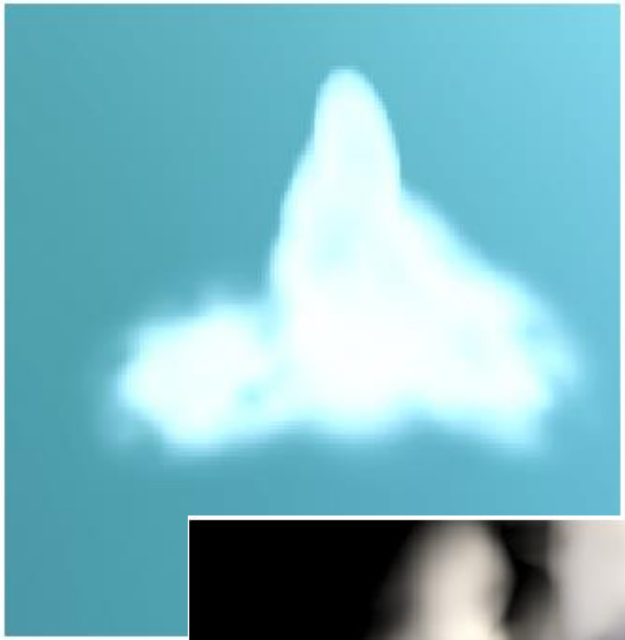
Importance of Shadows



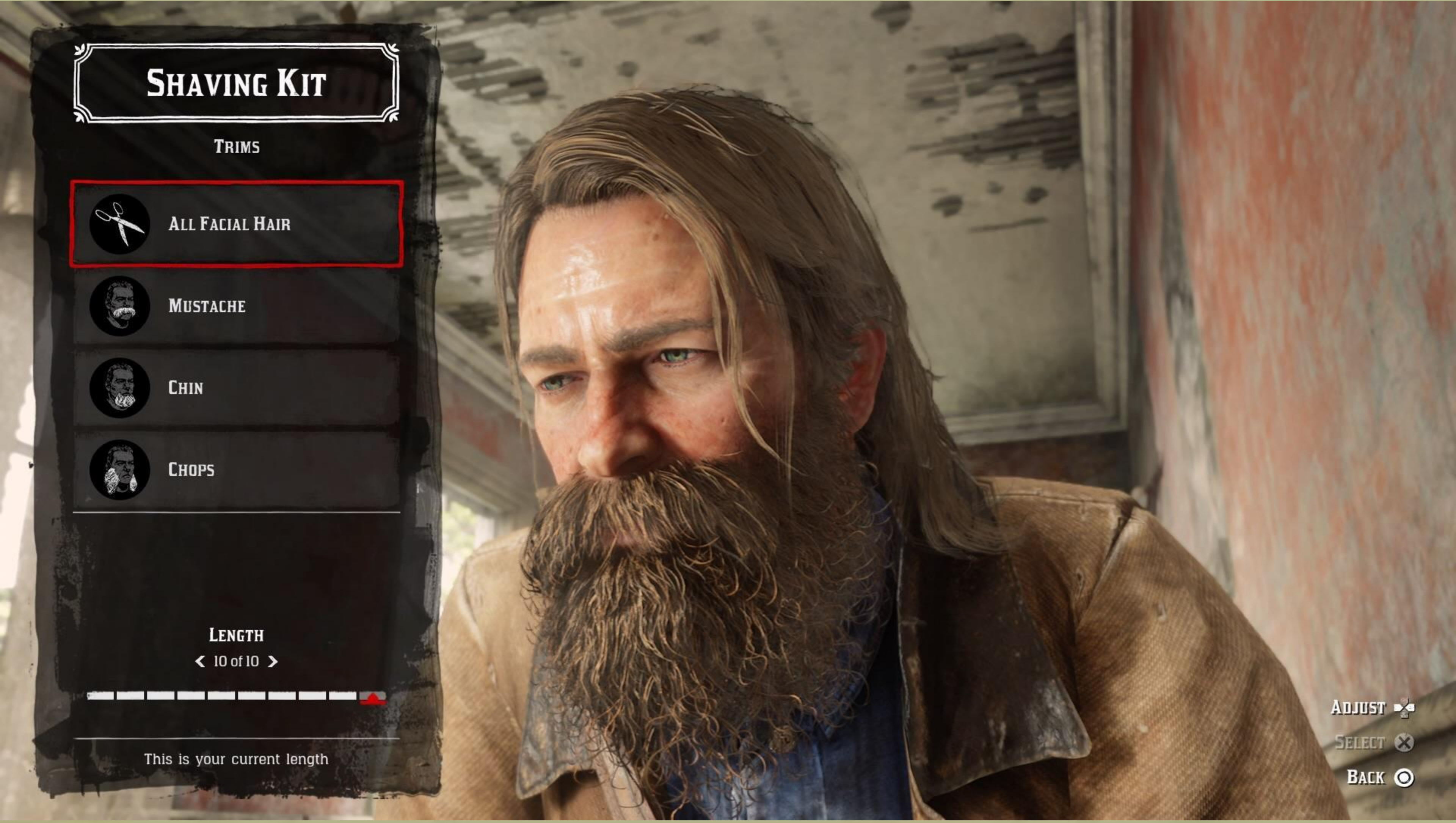
With hair self shadowing

Without hair self shadowing

Deep shadow maps
More examples

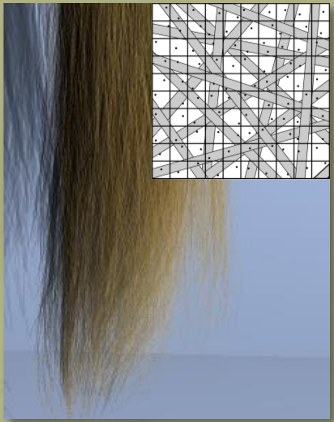


Red Dead Redemption 2 (2018)



Real time hair rendering

Two main challenges



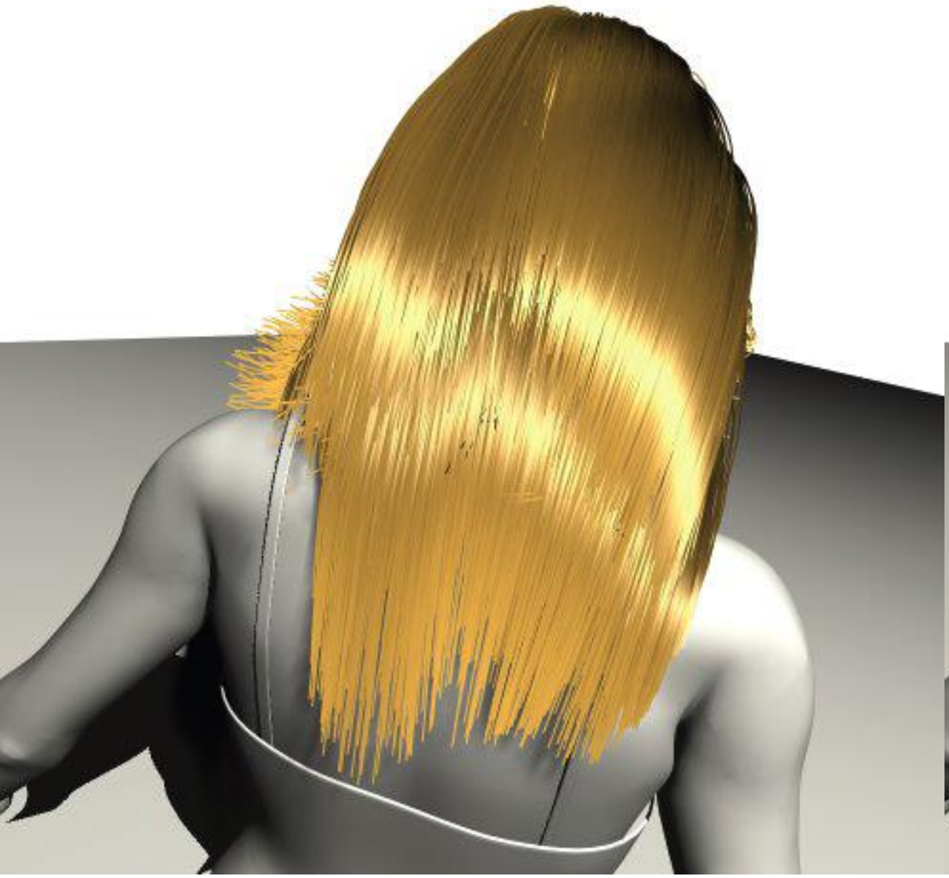
Self shadowing

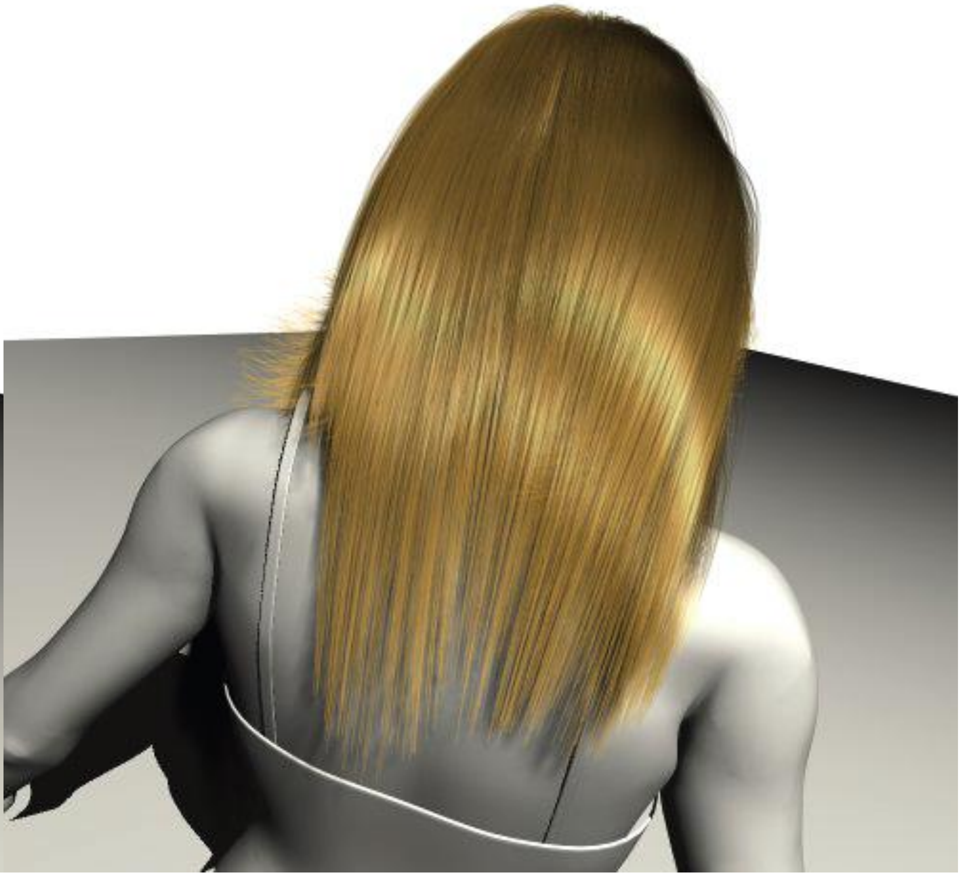
- Standard shadowing techniques fail
  - Shadow Maps => aliasing at sillhouette edges
  - Shadow Volumes => overdraw proportional to the number of sillhouette edges
  - Hair is **ALL** sillhouette edges
- Neither technique handles transparency

Transparency

- Each strand should contribute very little to a pixel (~1%)
- Hair strands are actually refractive and at least some transparency effect is required
- Alpha blending works very well to handle this

Importance of Transparency

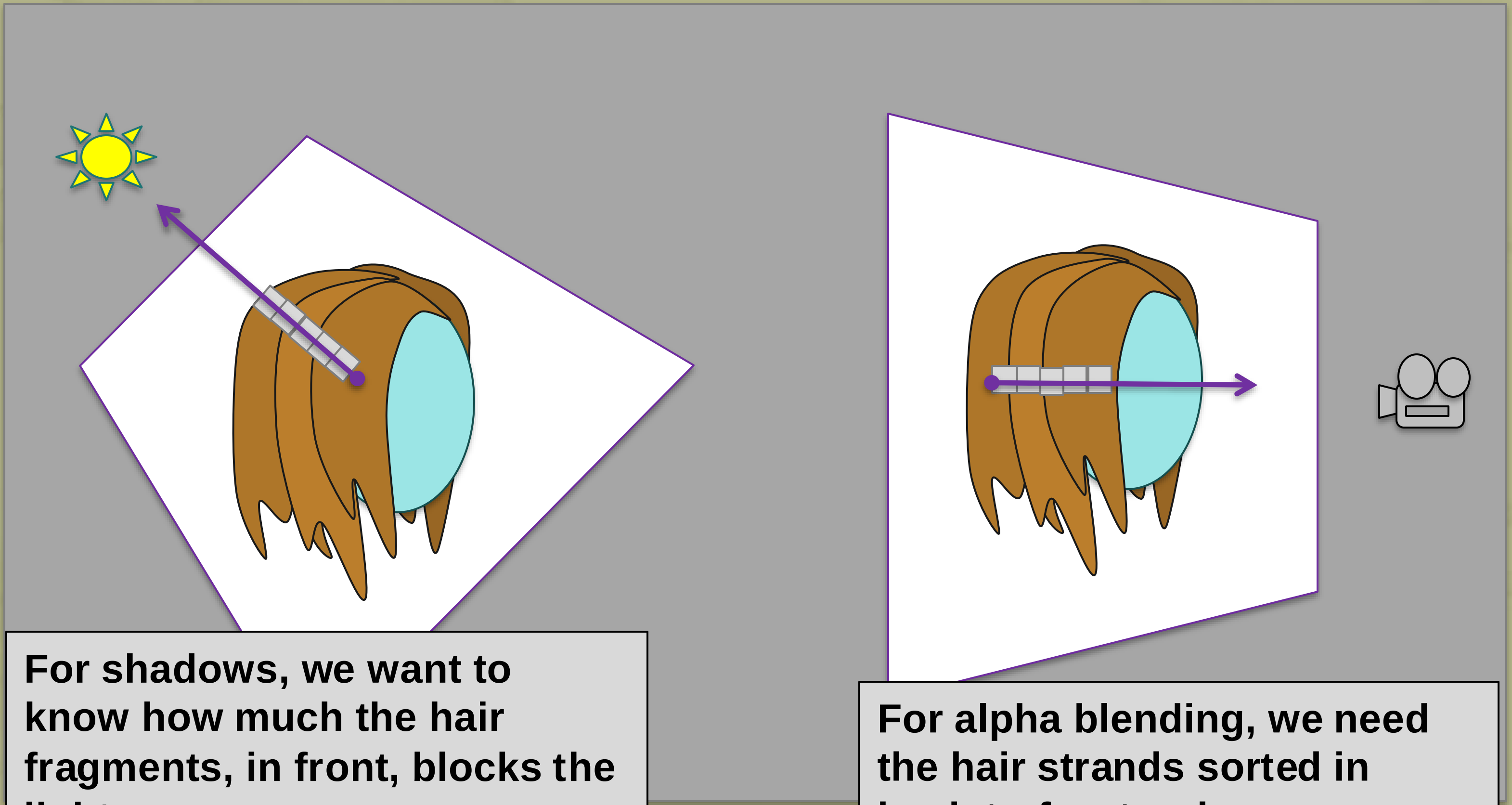




Hair rendered without alpha blending. Hair rendered with alpha blending (= 0.2).

Real time hair rendering

The two problems are quite similar



For shadows, we want to know how much the hair fragments, in front, blocks the light
- Can be solved by sorting

For alpha blending, we need the hair strands sorted in back-to-front order

Transparent Media

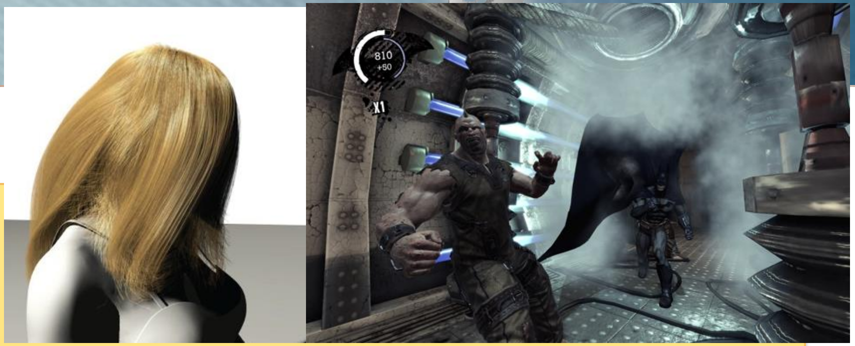
Transp

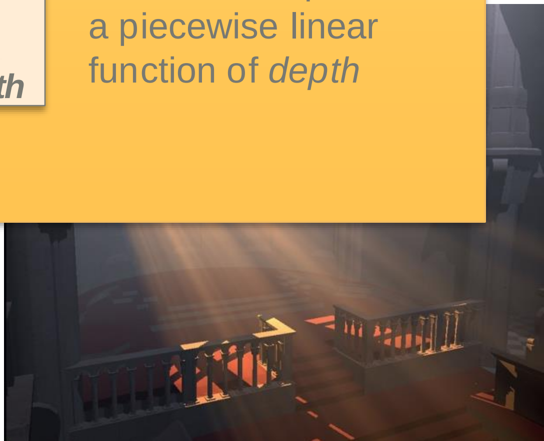
Hair ar

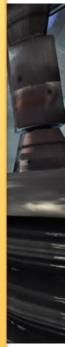
Shado

Shadow Volume based approaches

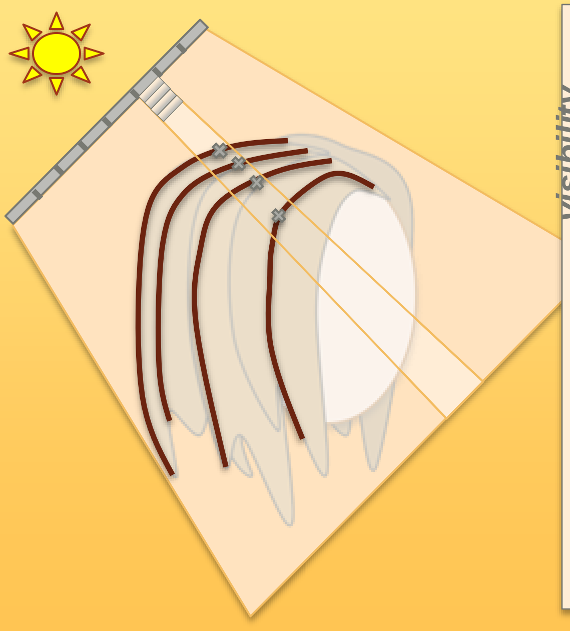
- Real-Time **Multiple** Scattering in Homogeneous Participating Media

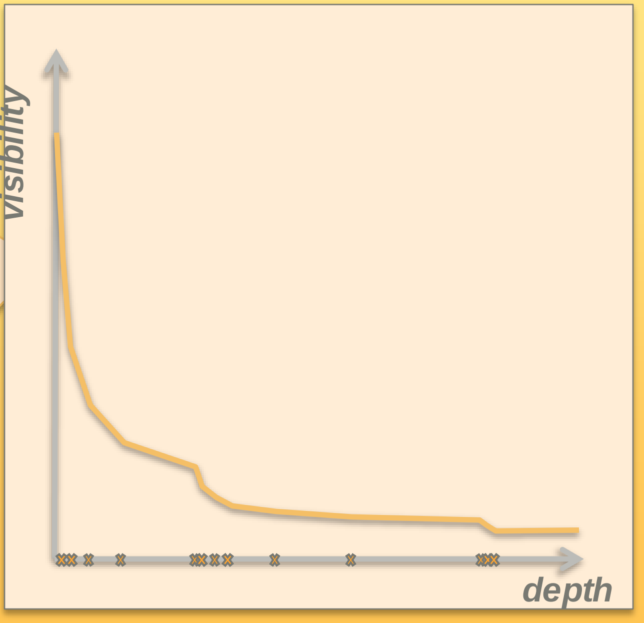






Deep Shadow Maps [Lokovic and Veach 2000]

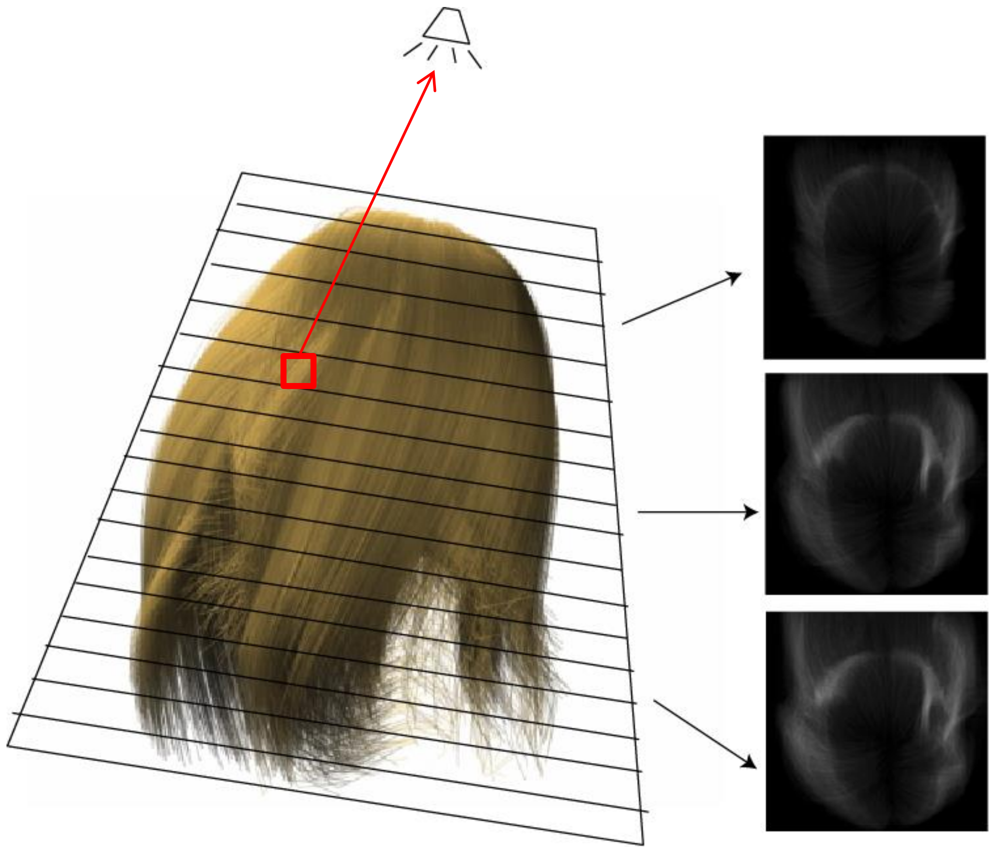




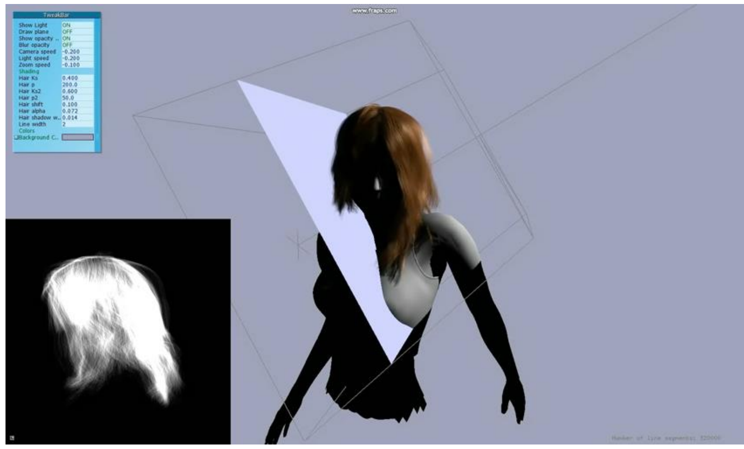
- Draw all hair strands from lights viewpoint
- Compute and store a visibility function per shadow map pixel.
- The visibility function represents how much the shadow increases with distance from light
- …and is compressed to a piecewise linear function of *depth*

Opacity Maps

- Build a 3d texture (=3D grid = 3D lookup table) where each cell represents the amount of shadow at a certain distance from light



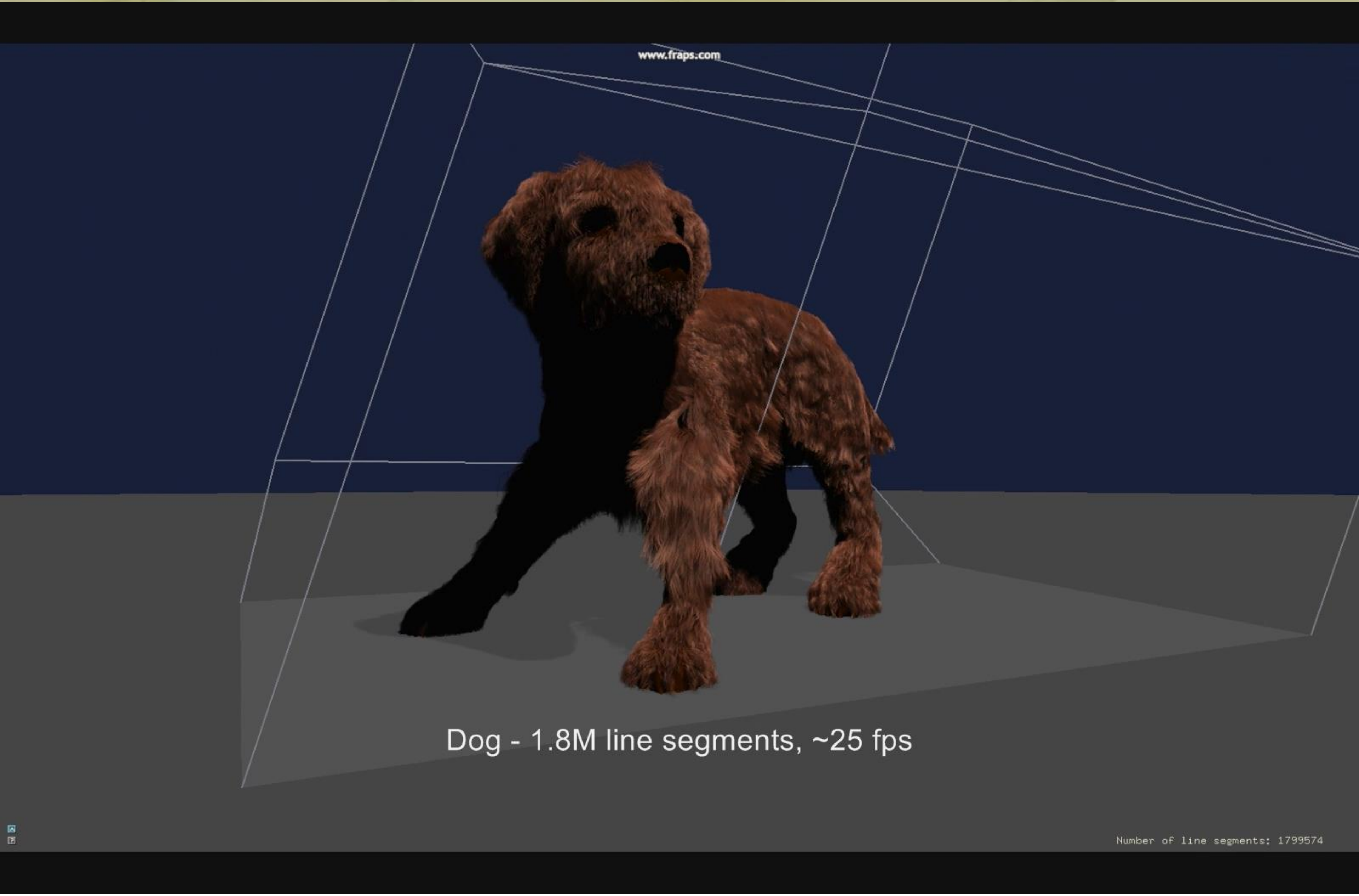
- Sort hair into 256 slices.
- Render each slice as 512x512 texels
- For each texel -> count shadowing strands in front of light source



TweakBar

Show Light	ON
Draw plane	OFF
Show opacity ..	ON
Blur opacity	OFF
Camera speed	-0.200
Light speed	-0.200
Zoom speed	-0.100
Shading	
Hair Ks	0.400
Hair p	200.0
Hair Ks2	0.600
Hair p2	50.0
Hair shift	0.100
Hair alpha	0.072
Hair shadow w..	0.014
Line width	2
Colors	
Background C..	

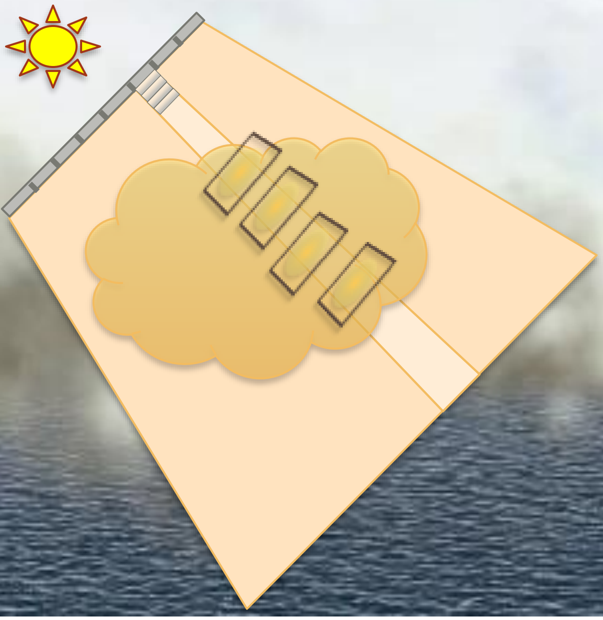
Essentially a 3D-grid with shadow values.
Each slice: 512x512 texels
256 slices

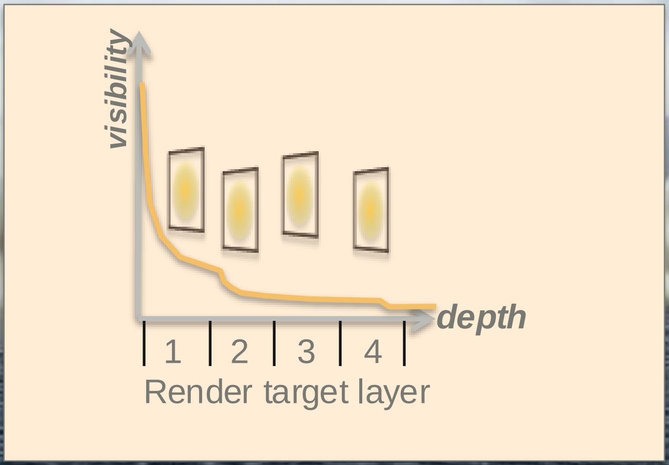


Dog - 1.8M line segments, ~25 fps

Shadows from semi-transparent objects
Particle Shadow Mapping

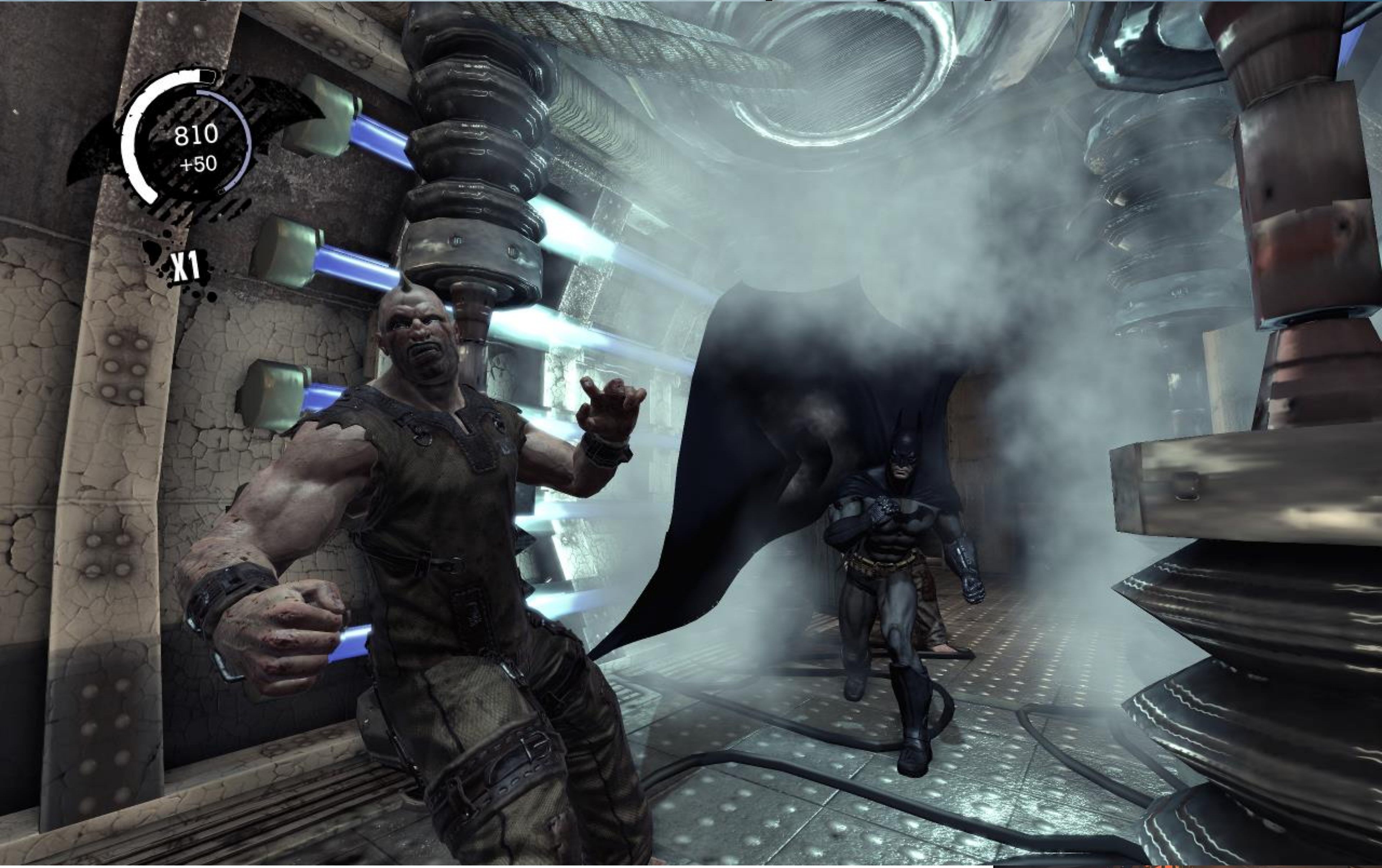
- Particle Shadow Mapping
  - by Jansen and Bavoil, GDC March 2013
    - Using SV_RenderTargetArrayIndex

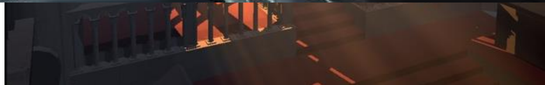




Per SM texel, precompute a visibility-function on depth (in separate real-time rendering pass). Then query this function when adding shadow-value for fragment.

Transparent Media – Fourier Opacity Maps





Transparent Media

E.g., one method for all:

- Moment Shadow Maps for Single Scattering, Soft Shadows and Translucent Occluders, Christoph Peters et al., 2016.

Or more streamlined methods:

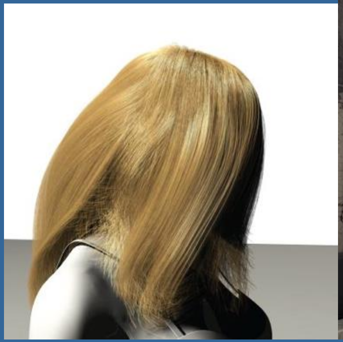
Transparent solid objects:

- **Shadow Volumes:**
  - Textured transparency: Per-triangle shadow volumes [Sintorn et al. '11]
- **Shadow Maps**
  - Layered Shadow Maps
  - Stochastic transparency [Sintorn et al.]

Volumetric Shadows:

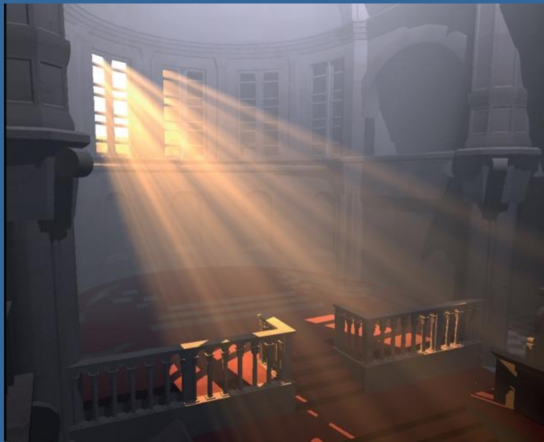
Hair and Smoke:

- **Deep Shadow Maps [Lokovic and Veach 2000]**
  - Opacity Shadow Maps [Kim and Neumann '01]
  - Occupancy Maps [Sintorn and Assarsson, '09]
  - Fourier Opacity Mapping [Jansen and Bavoil '10]



Shadows from scattering in Participating Media

- Real-Time **Single** Scattering in Homogeneous Participating Media
  - Ray-Marching based approaches
  - Shadow-Volume based approaches
    - Our version is part of NVIDIA SDK
- Real-Time **Multiple** Scattering in Homogeneous Participating Media



But AI is rapidly approaching for most of these tasks and there is lot's of new cool things to do!

Transparent Media

E.g., one method for all:

- Moment Shadow Maps for Single Scattering, Soft Shadows and Translucent Occluders, Christoph Peters et al., 2016.

Or more streamlined methods:

Transparent solid objects:

- **Shadow Volumes:**
  - Textured transparency: Per-triangle shadow volumes [Sintorn et al. '11]
- **Shadow Maps**
  - Layered Shadow Maps
  - Stochastic transparency [Sintorn et al.]

Volumetric Shadows:

Hair and Smoke:

- **Deep Shadow Maps [Lokovic and Veach 2000]**
  - Opacity Shadow Maps [Kim and Neumann '01]
  - Occupancy Maps [Sintorn and Assarsson, '09]
  - Fourier Opacity Mapping [Jansen and Bavoil '10]

However, several of these methods are relatively old, and there is an apparent lack of newer faster more general real-time methods.

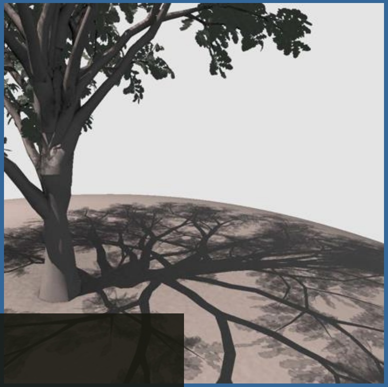
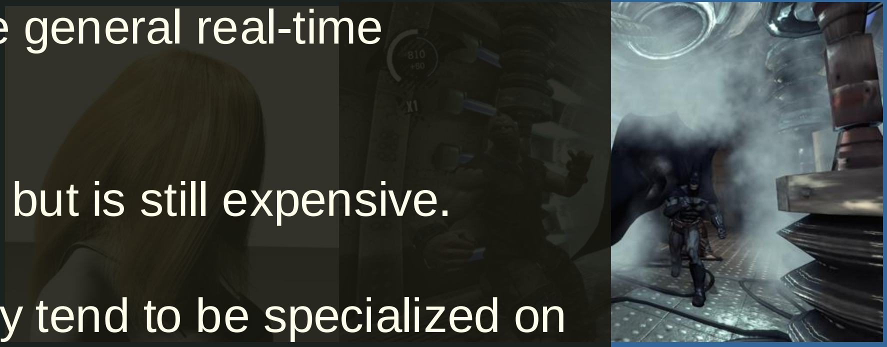
RTX ray tracing can be made general but is still expensive.

AI methods can be made fast, but they tend to be specialized on individual effects, to be both fast and accurate.

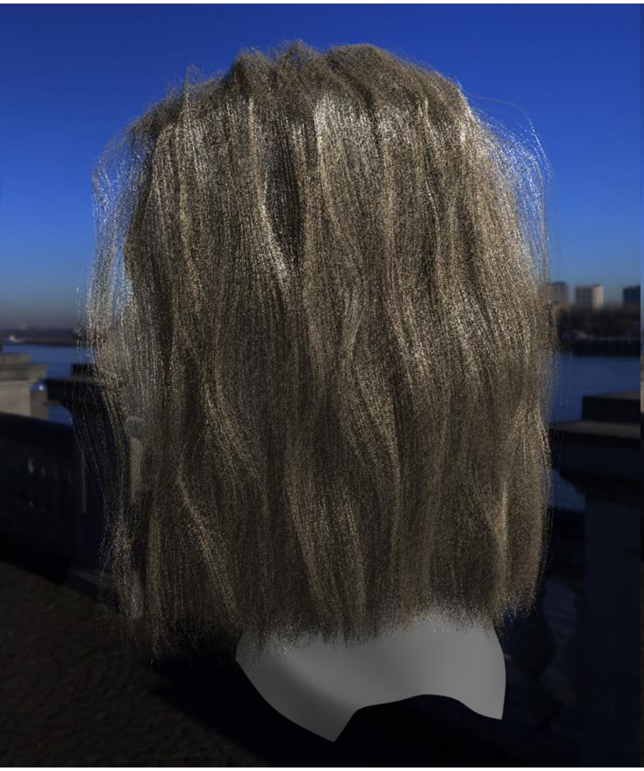
Shadows from scattering in Participating Media

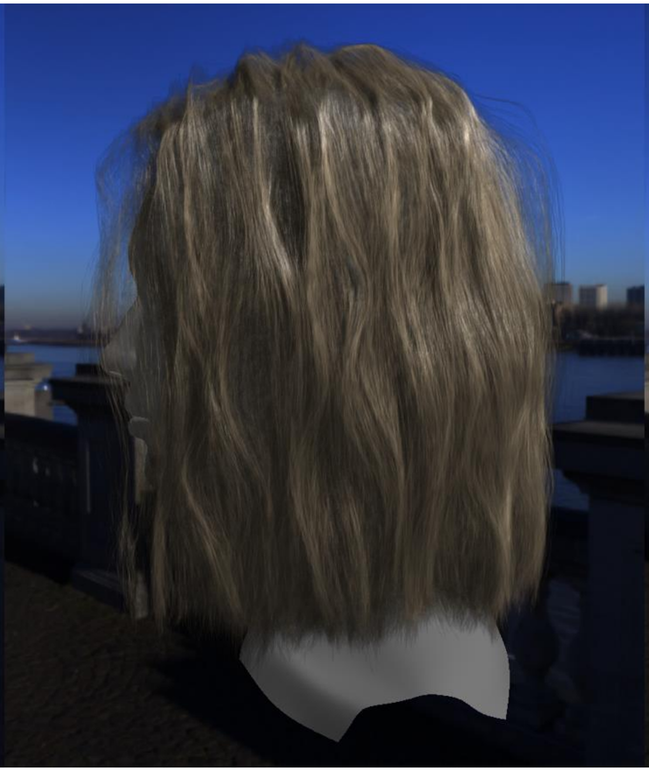
- Real-Time **Single** Scattering in Homogeneous Participating Media
  - Ray-Marching based approaches
  - Shadow-Volume based approaches
    - Our version is part of NVIDIA SDK
- Real-Time **Multiple** Scattering in Homogeneous Participating Media

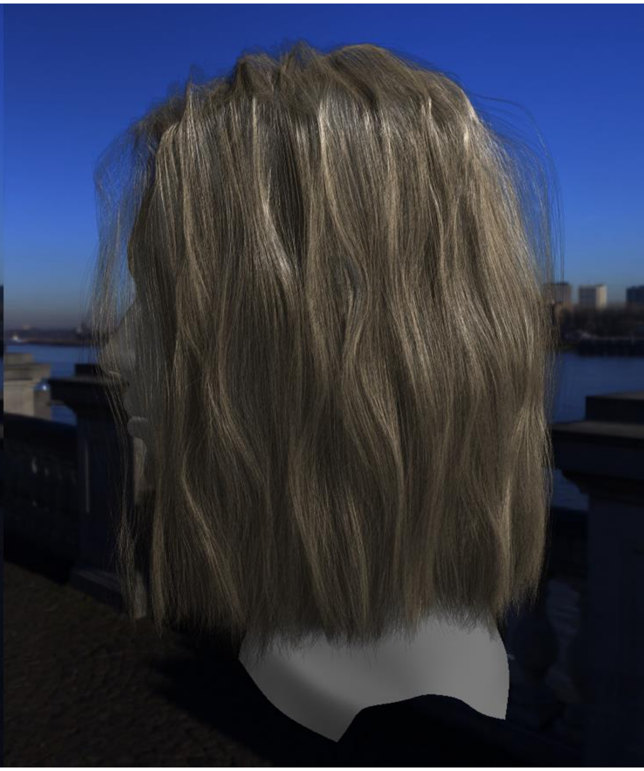
But AI is rapidly approaching for most of these tasks and there is lot's of new cool things to do!

Real-time Hair using AI


Stochastic transparency + stochastic shadow map


Our real-time DNN


“Ground truth”

Stochastic sampling resembles dithering in a way. Transparency or depth is sampled stochastically, storing 1 sample per pixel, where probability is proportional to the sample’s occlusion by closer semi-transparent layers.