CMSC427
Advanced shading – getting global illumination by local methods

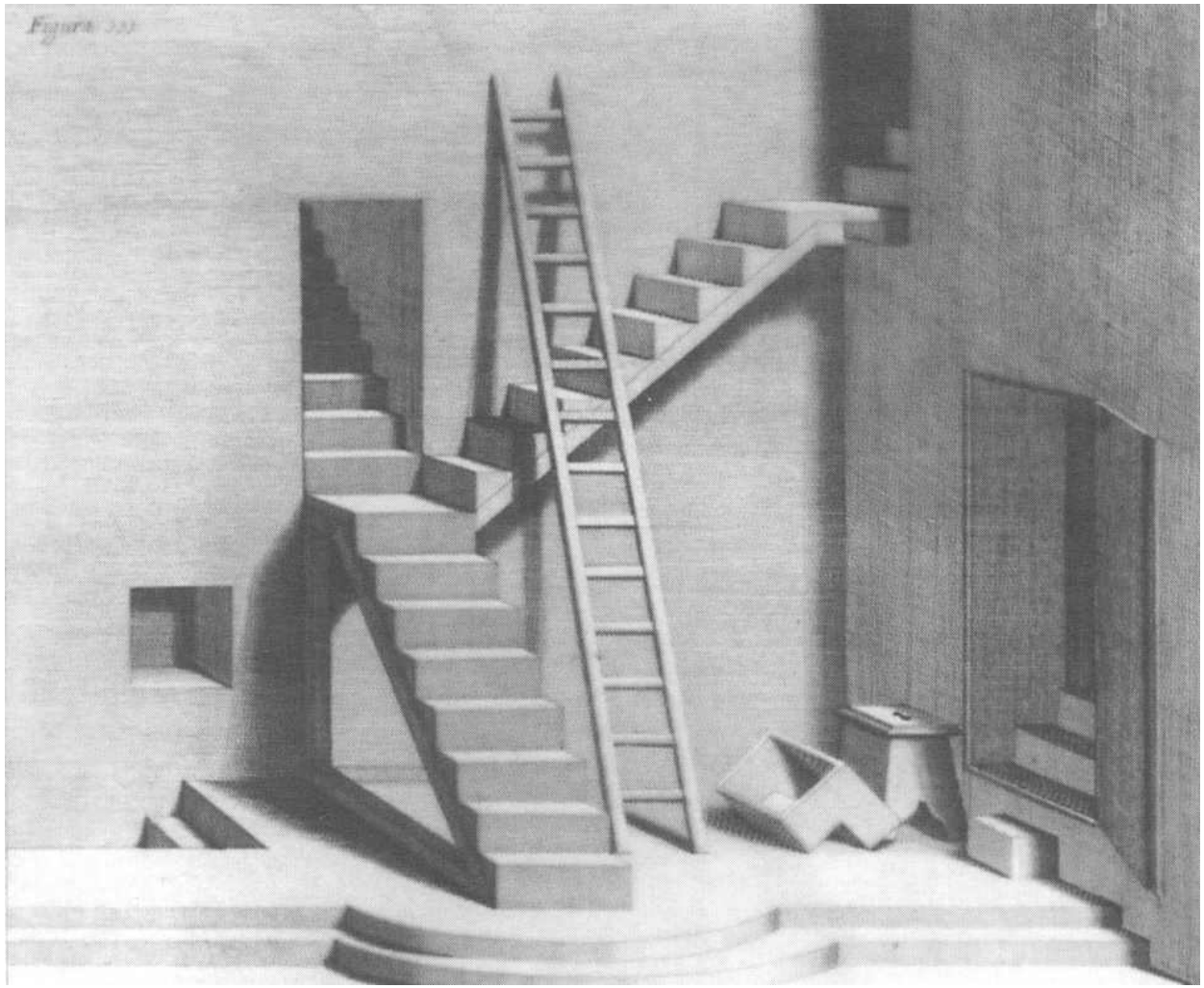
Credit: slides Prof. Zwicker

Topics

- Shadows
- Environment maps
- Reflection mapping
- Irradiance environment maps
- Ambient occlusion
- Reflection and refraction
- Toon shading

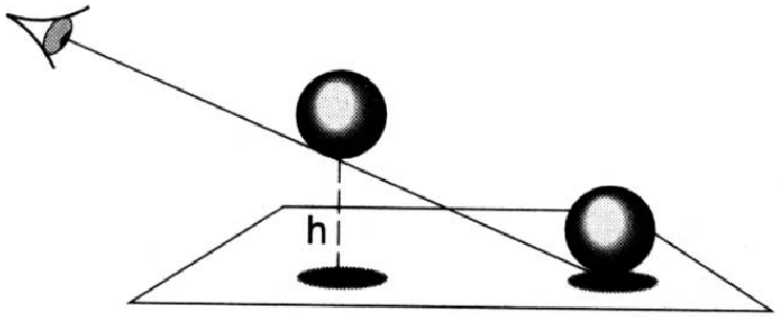
Why are shadows important?

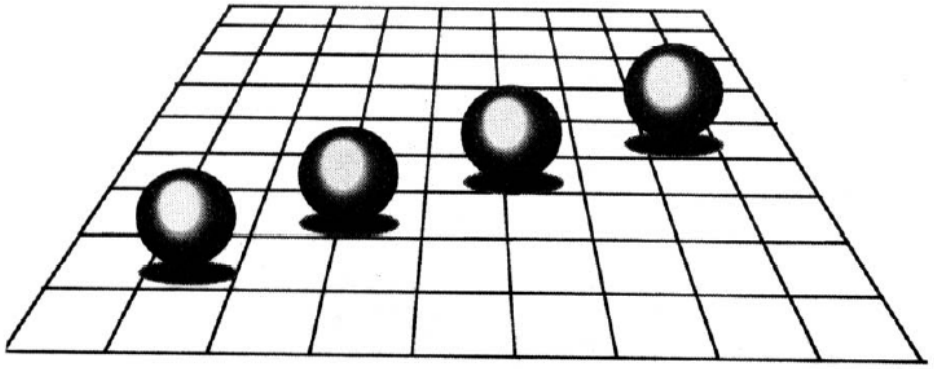
- Cues on scene lighting

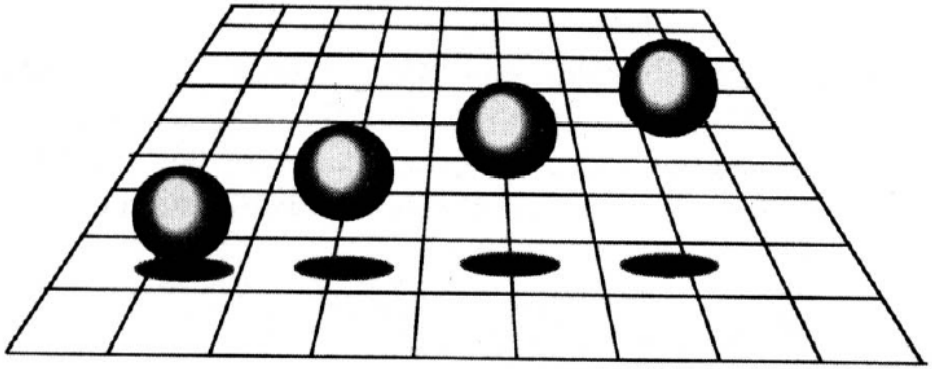


Why are shadows important?

- Contact points
- Depth cues

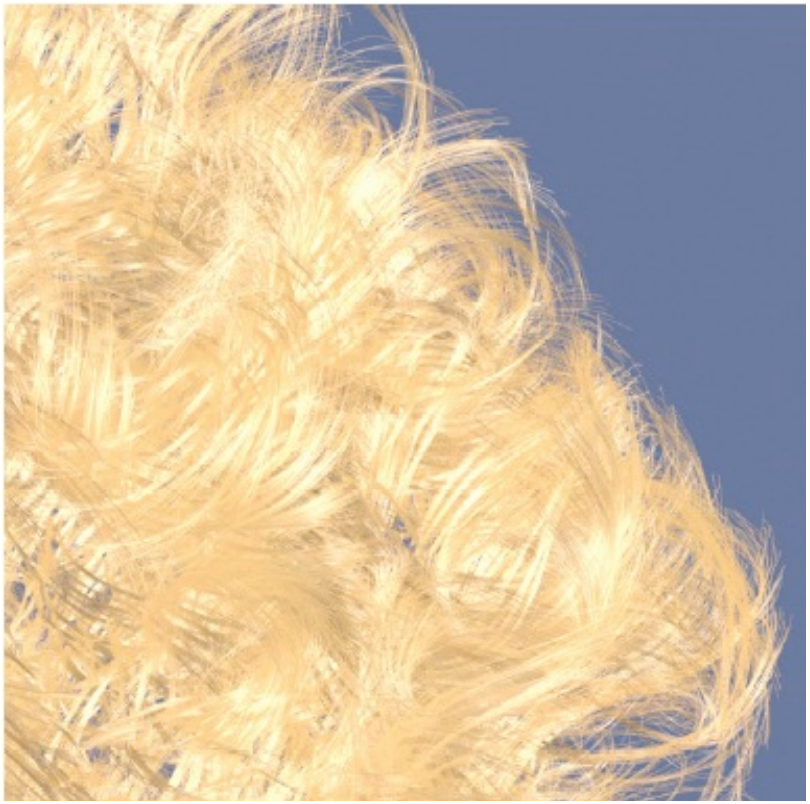






Why are shadows important?

- Realism

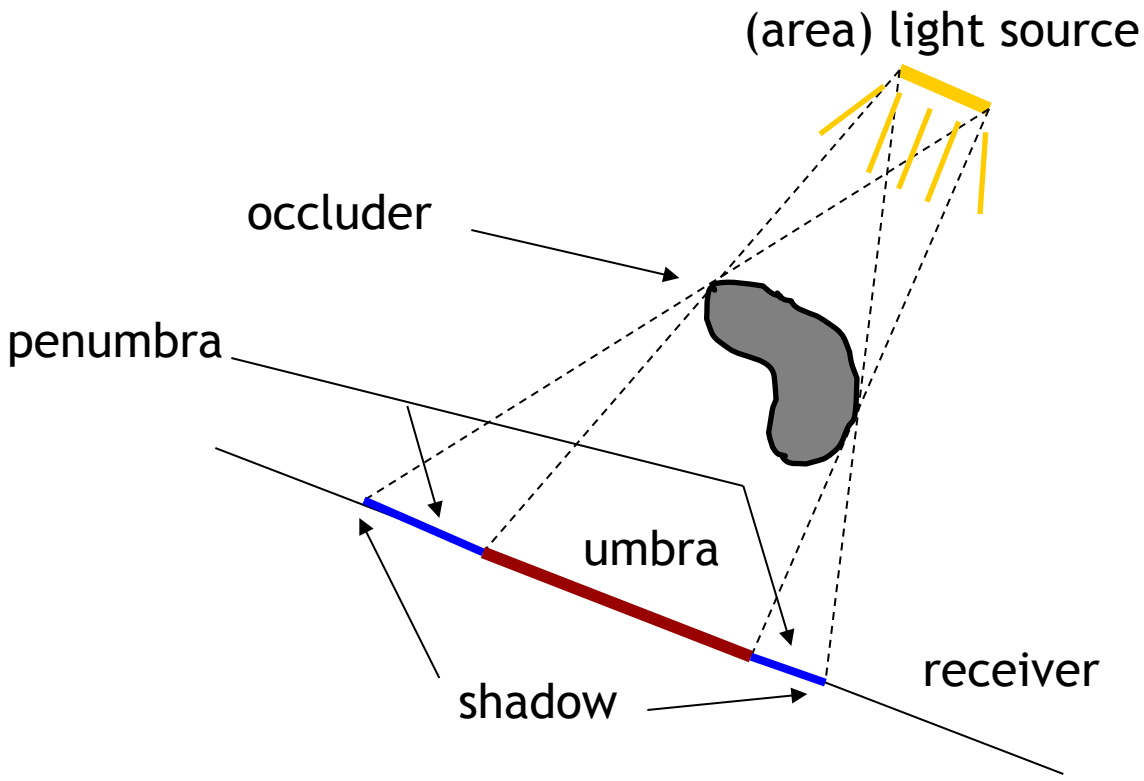


Without self-shadowing

Without self-shadowing

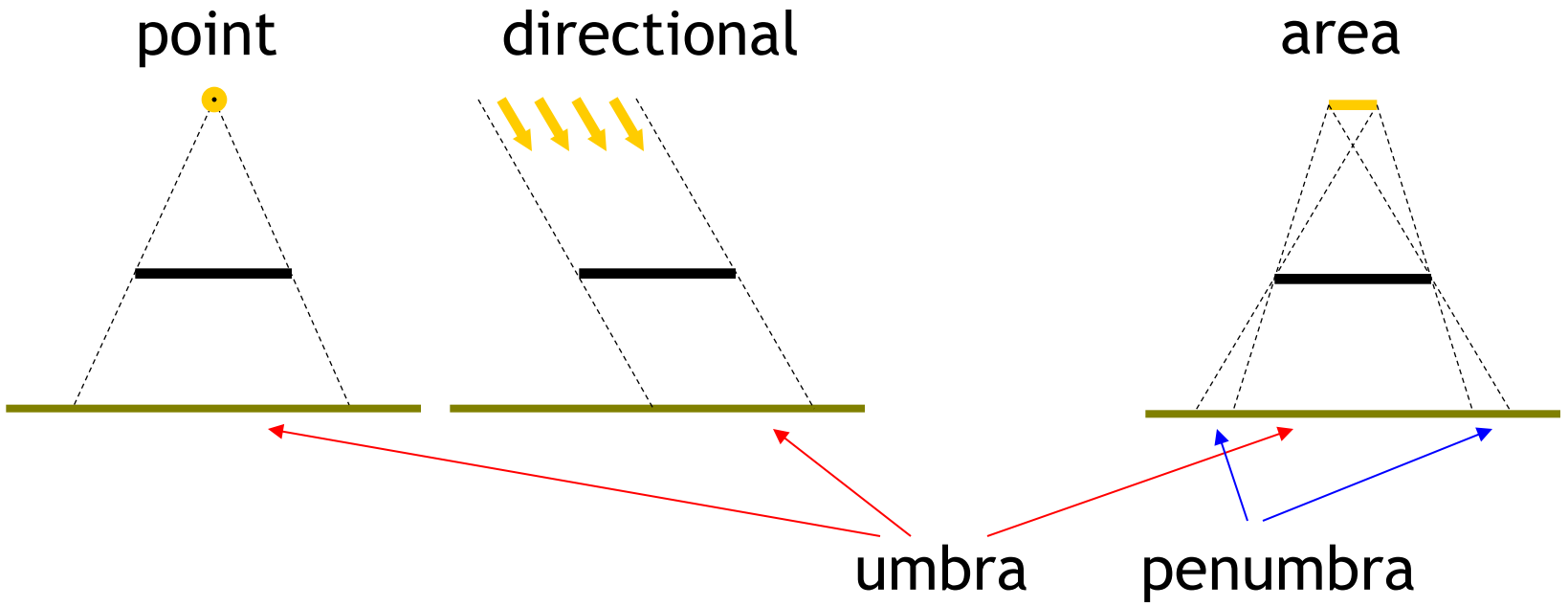
Terminology

- Umbra: fully shadowed region
- Penumbra: partially shadowed region



Hard and soft shadows

- Point and directional lights lead to hard shadows, no penumbra
- Area light sources lead to soft shadows, with penumbra



Hard and soft shadows

Hard shadow,
point light source

Soft shadow,
area light source

Shadows for interactive rendering

- Focus on hard shadows
  - Soft shadows often too hard to compute in interactive graphics
- Two main techniques
  - Shadow mapping
  - Shadow volumes
- Many variations, subtleties
- Still active research area

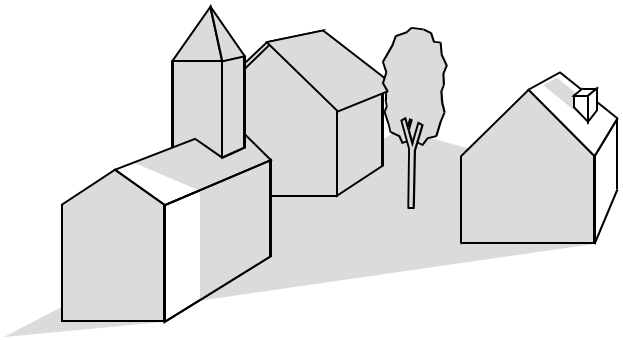
Shadow mapping

http://en.wikipedia.org/wiki/Shadow_mapping

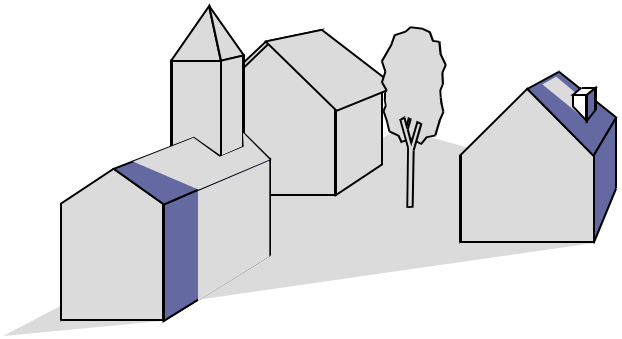
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

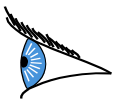
Main idea

- Scene point is lit by light source if it is visible from light source
- Determine visibility from light source by placing camera at light source position and rendering scene



Scene points are lit if visible from light source



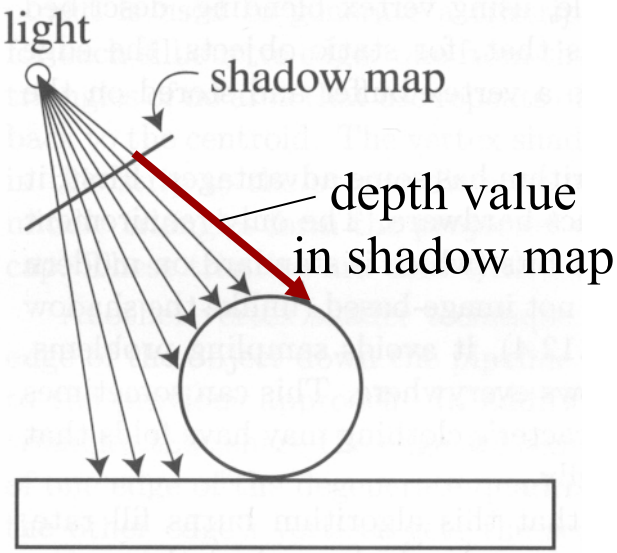


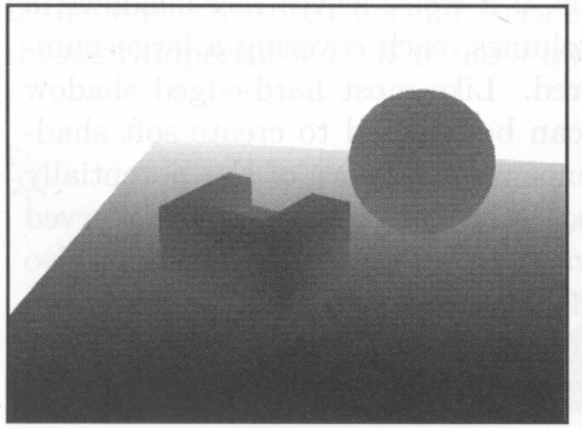
Determine visibility from light source by placing camera at light source position

Two pass algorithm

First pass

- Render scene by placing camera at light source position
- Store depth image (*shadow map*)



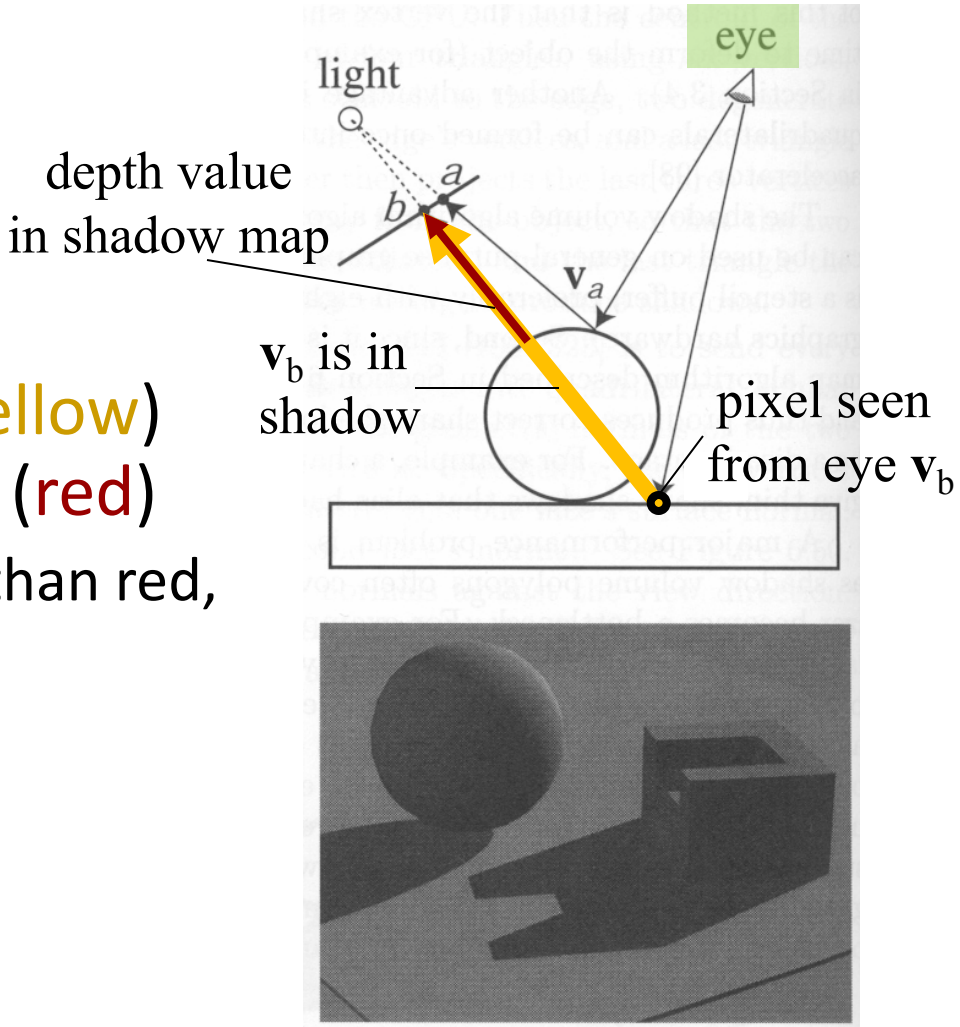


Depth image seen from light source

Two pass algorithm

Second pass

- Render scene from camera (eye) position
- At each pixel, compare distance to light source (yellow) with value in shadow map (red)
  - If yellow distance is larger than red, we are in shadow
  - If distance is smaller or equal, pixel is lit



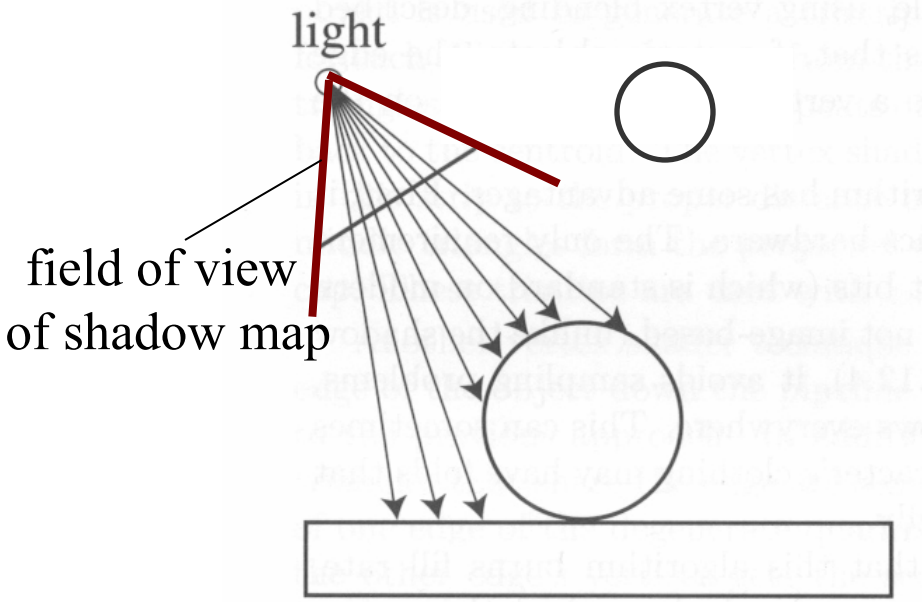
Final image with shadows

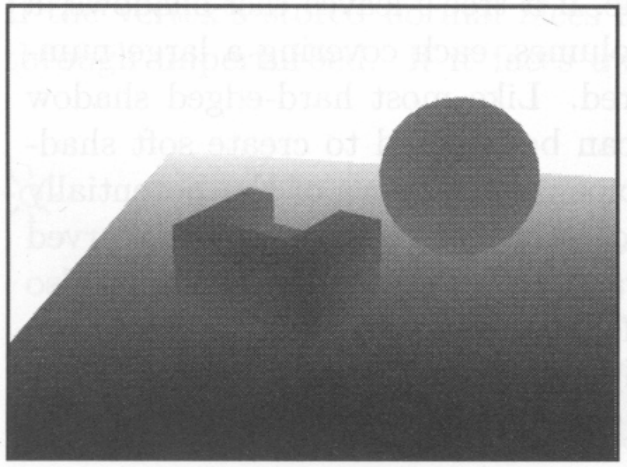
Issues

- Limited field of view of shadow map
- Z-fighting
- Sampling problems

Limited field of view

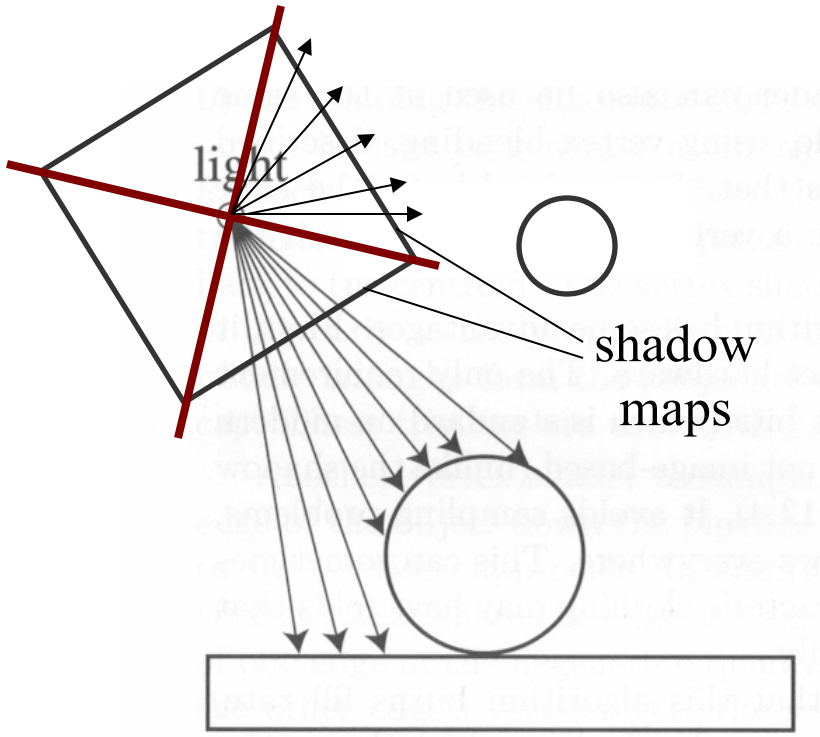
- What if a scene point is outside the field of view of the shadow map?

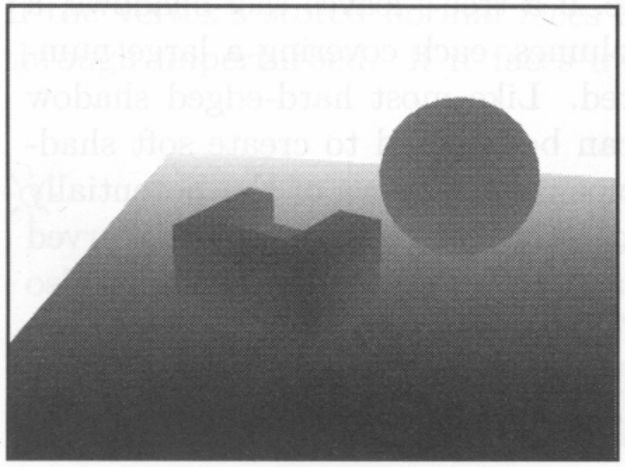




Limited field of view

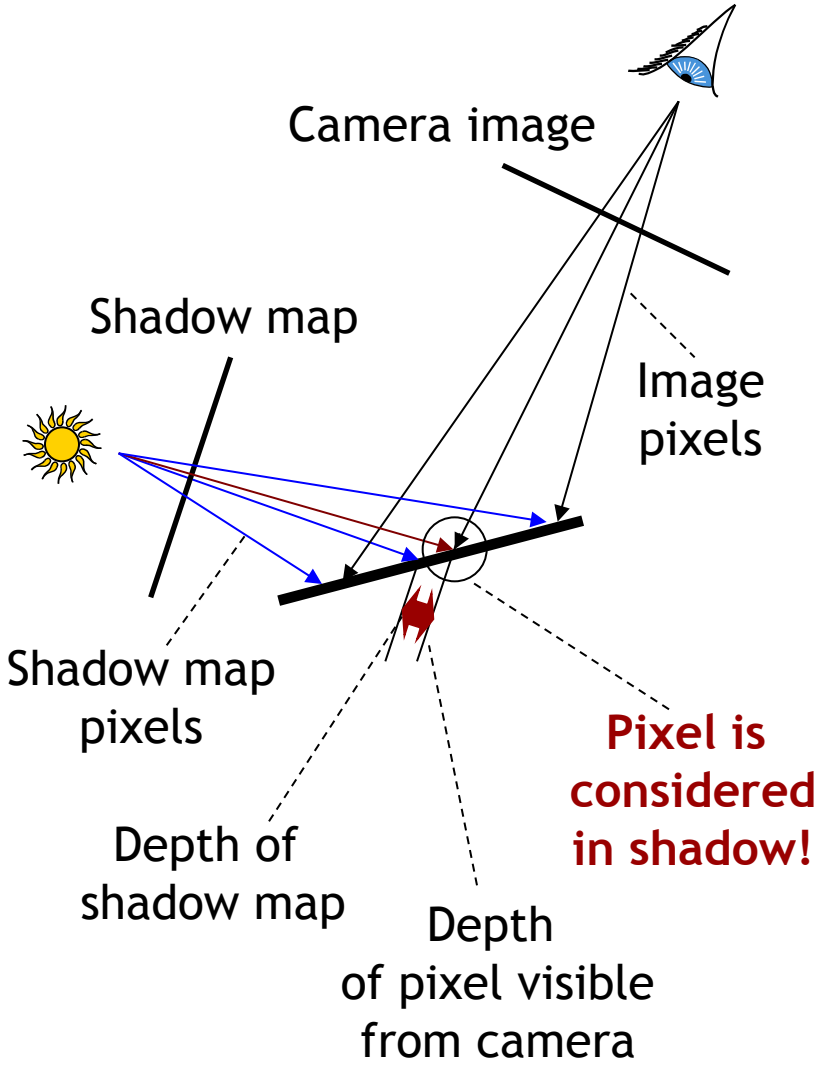
- What if a scene point is outside the field of view of the shadow map?
- Use six shadow maps, arranged in a cube
- Requires rendering pass for each shadow map!





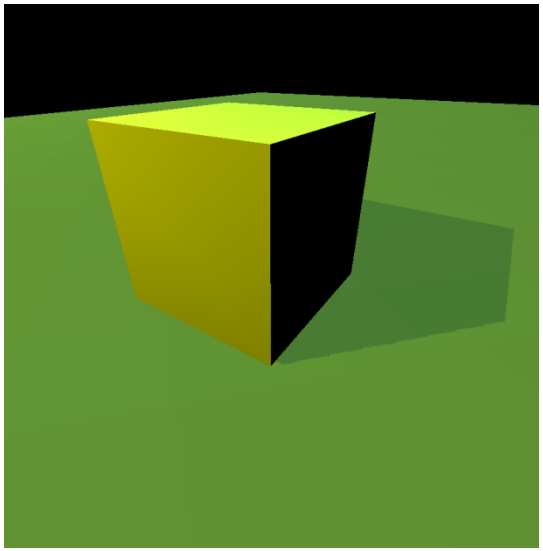
z-fighting

- In theory, depth values for points visible from light source are equal in both rendering passes
- Because of limited resolution, depth of pixel visible from camera could be larger than shadow map value
- Need to add bias in first pass to make sure pixels are lit

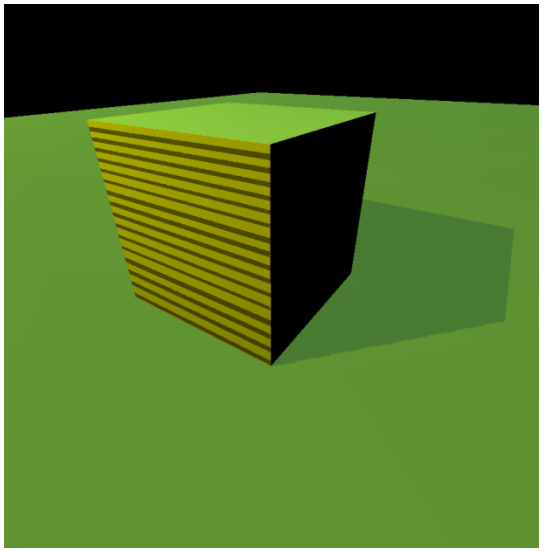


Solution

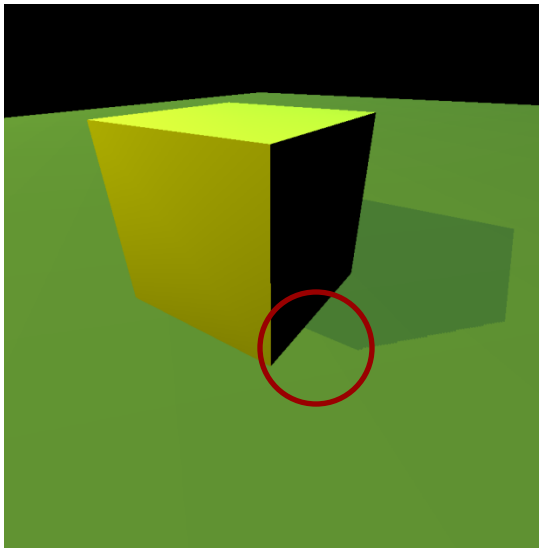
- Add bias when rendering shadow map
  - Move geometry away from light by small amount
- Finding correct amount of bias is tricky



Correct bias



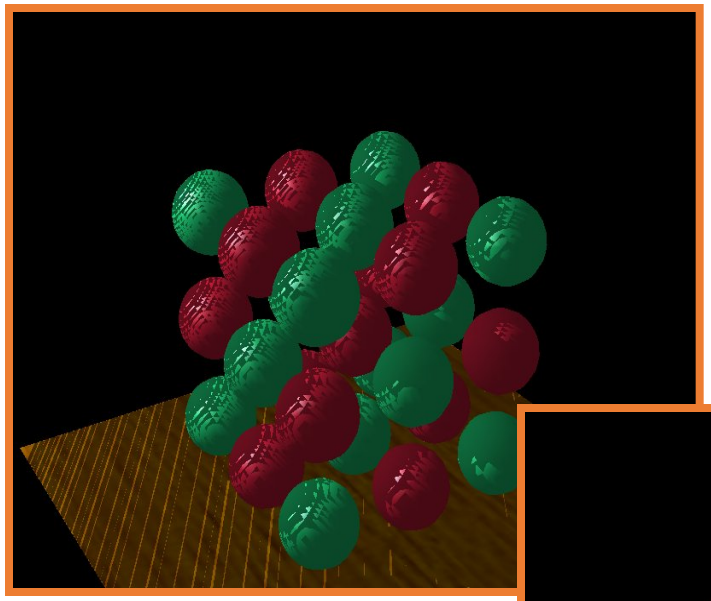
Not enough bias



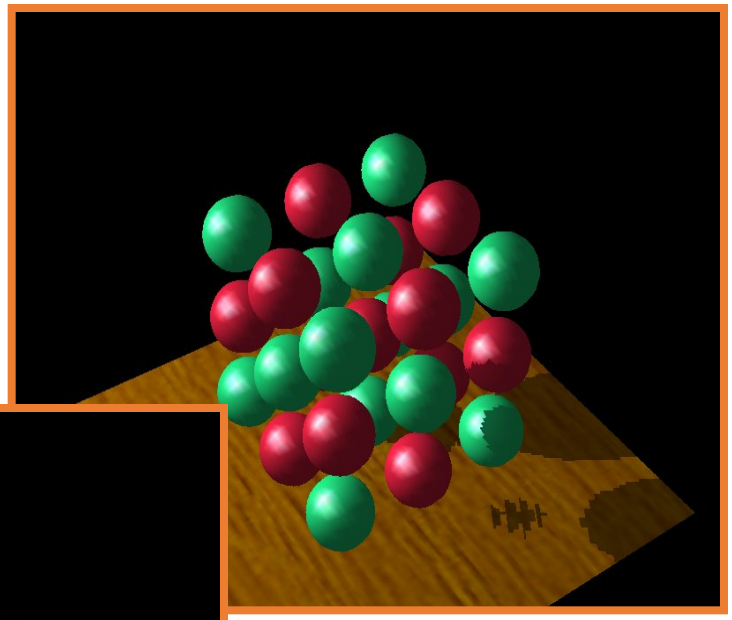
Too much bias

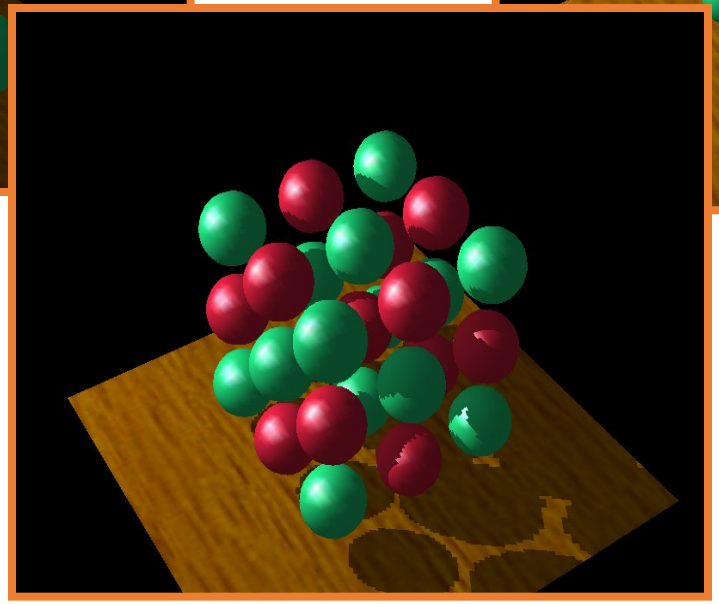
Bias

Not enough



Too much

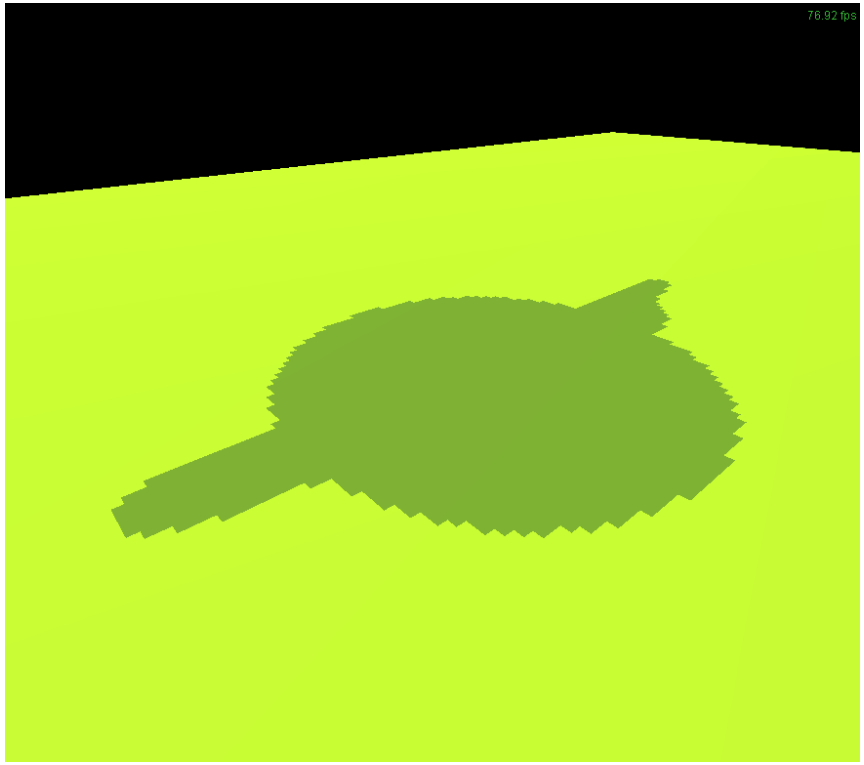




Correct

Sampling problems

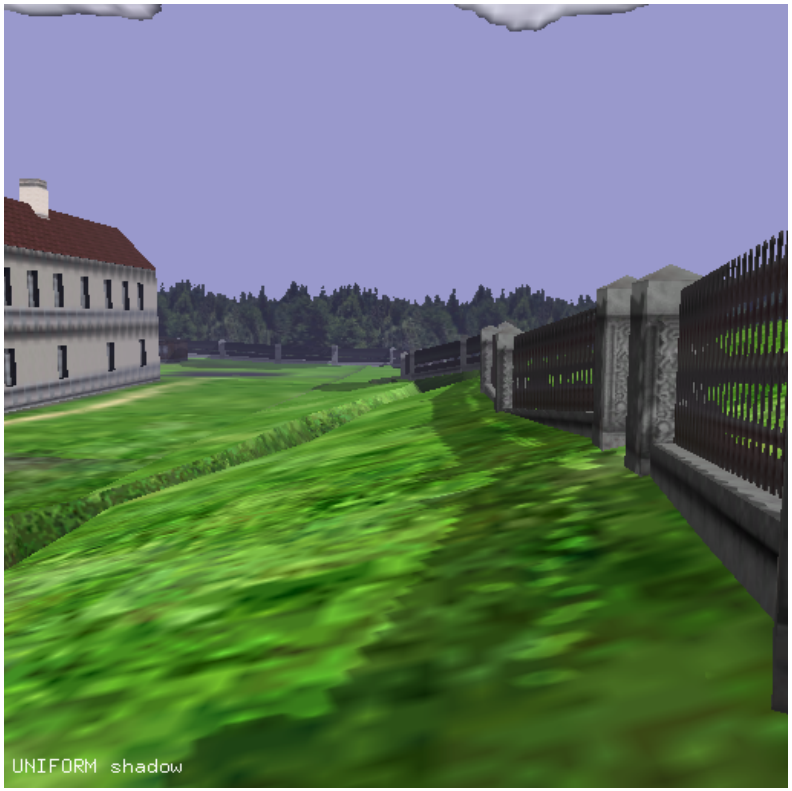
- Shadow map pixel may project to many image pixels
- Ugly stair-stepping artifacts



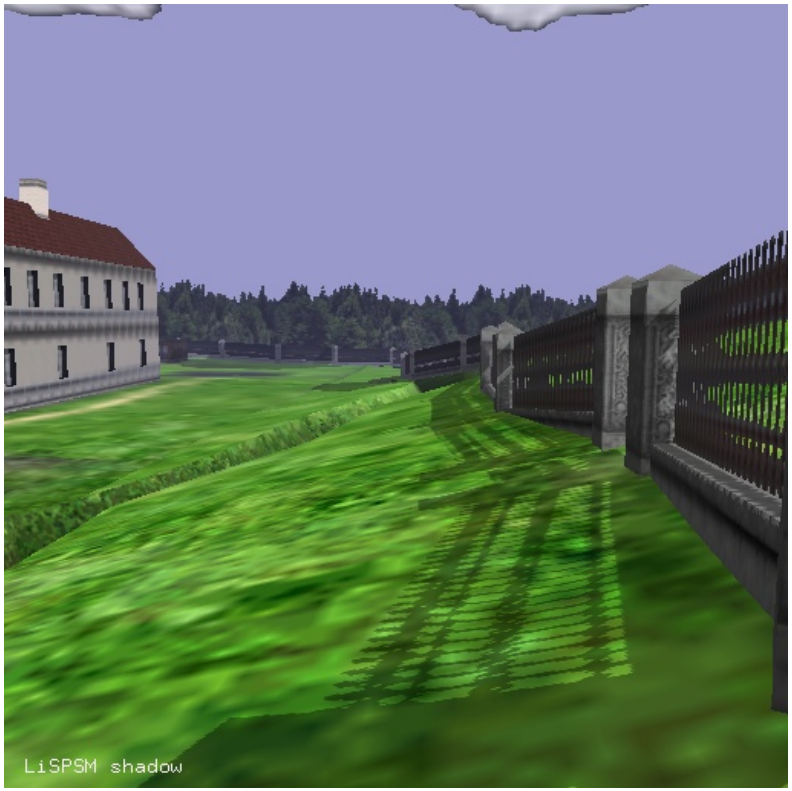
Solutions

- Increase resolution of shadow map
  - Not always sufficient
- Split shadow map into several slices
- Tweak projection for shadow map rendering
  - Light space perspective shadow maps (LiSPSM)
    http://www.cg.tuwien.ac.at/research/vr/lispsm/
  - With GLSL source code!
- Combination of splitting and LiSPSM
  - Basis for most serious implementations
  - List of advanced techniques see
    http://en.wikipedia.org/wiki/Shadow_mapping

LiSPSM



Basic shadow map

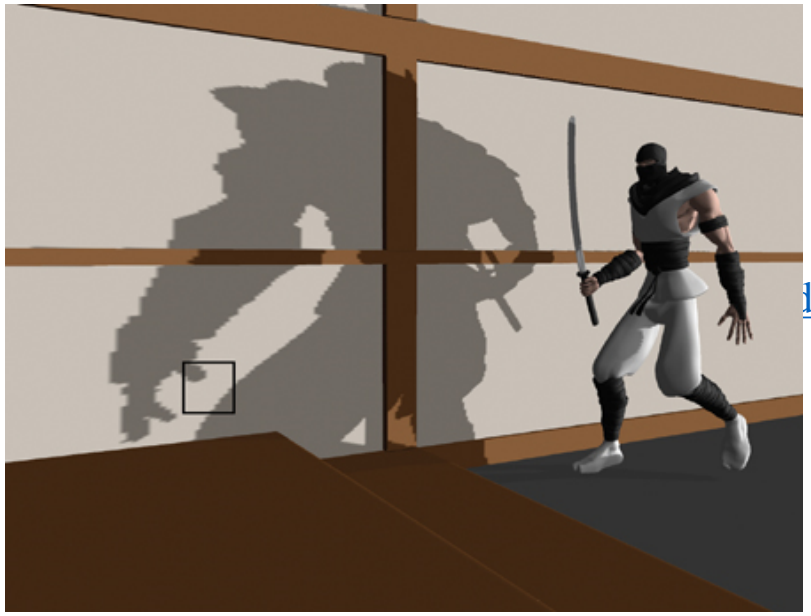


Light space perspective shadow map

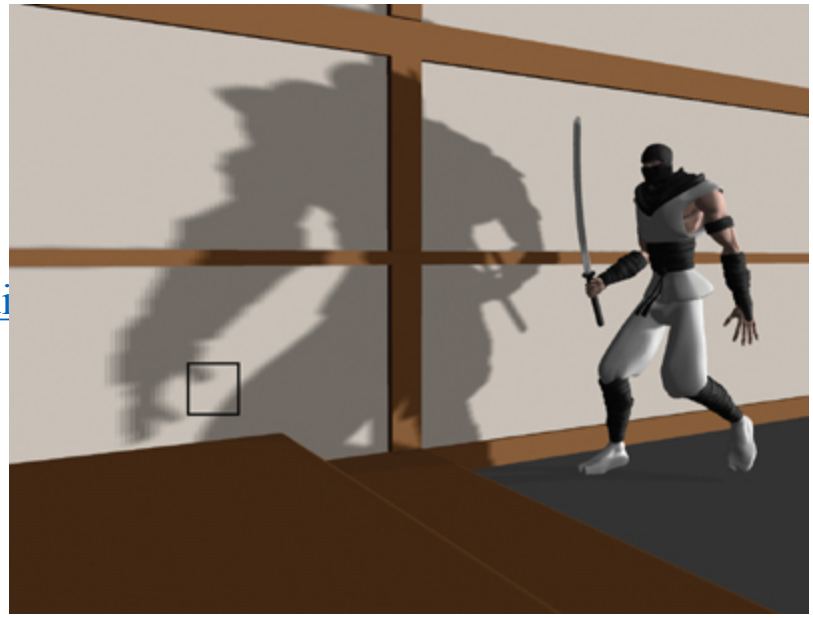
Percentage closer filtering

- Goal: avoid stair-stepping artifacts
- Similar to texture filtering, but with a twist

Simple shadow mapping

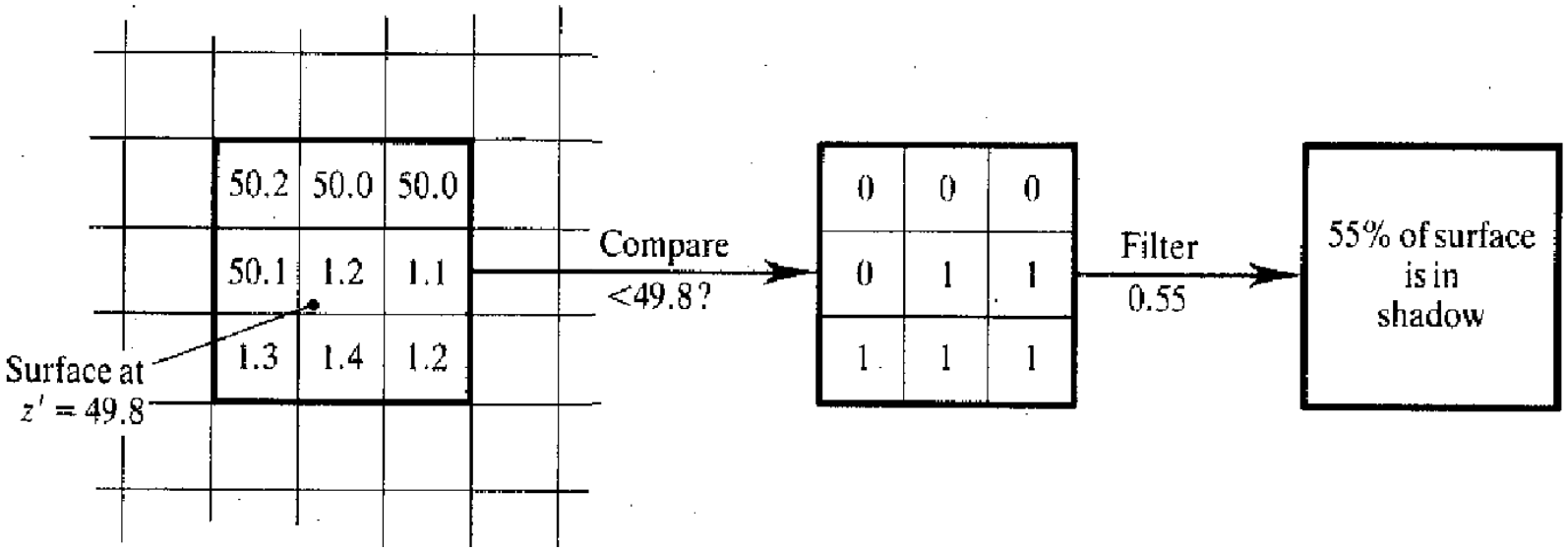


Percentage closer filtering



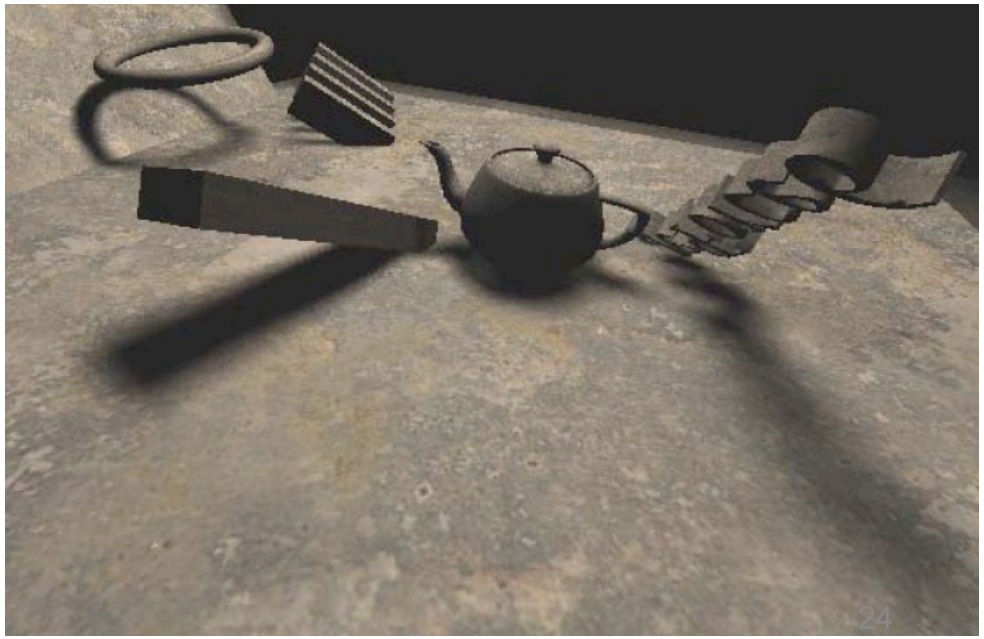
Percentage closer filtering

- Instead of looking up one shadow map pixel, look up several
- Perform depth test for each shadow map pixel
- Compute percentage of lit shadow map pixels

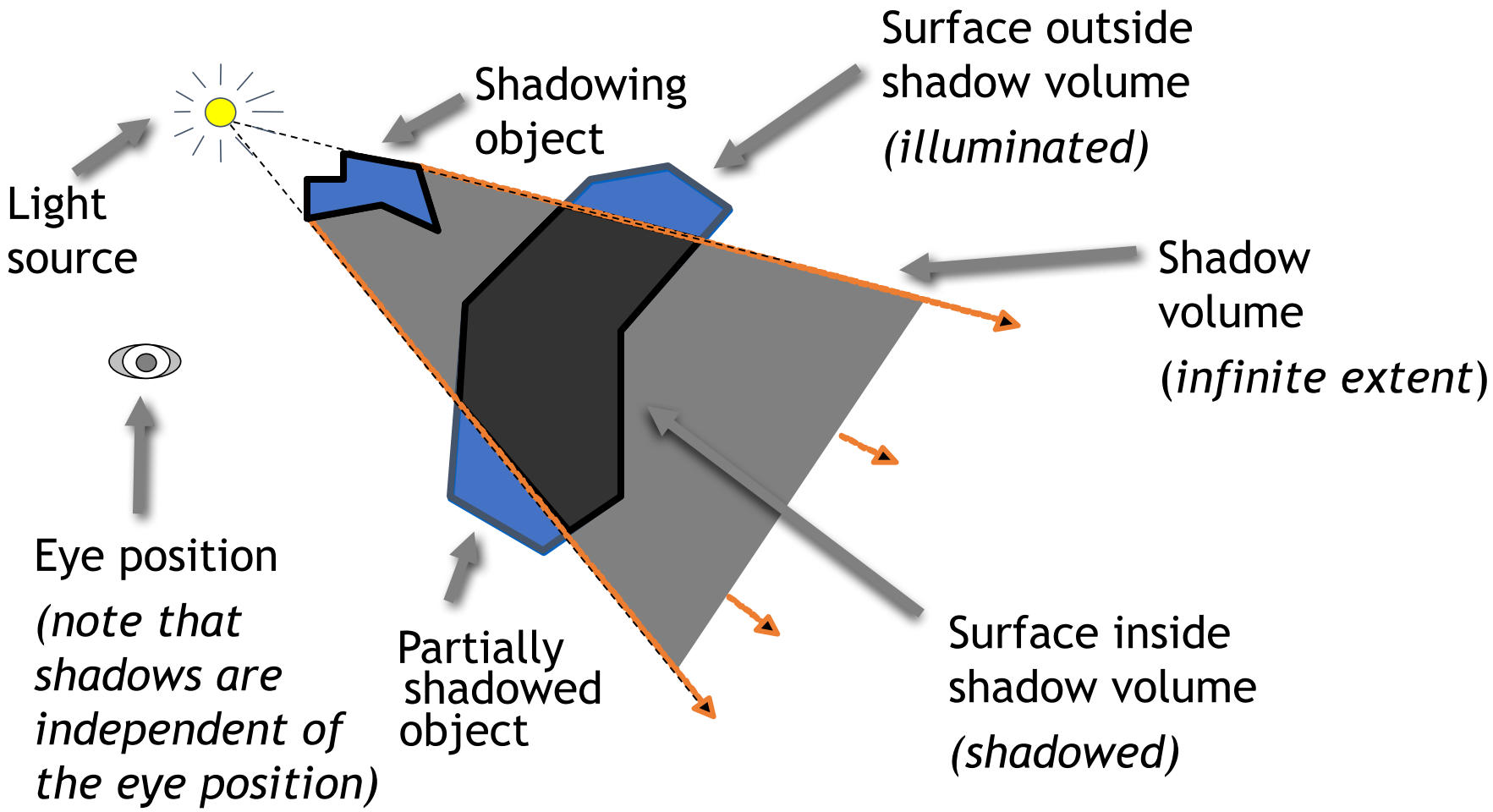


Percentage closer filtering

- Supported in hardware for small filters (2x2 shadow map pixels)
- Can use larger filters (look up more shadow map pixels) at cost of performance penalty
- Fake soft shadows
  - Larger filter, softer shadow boundary



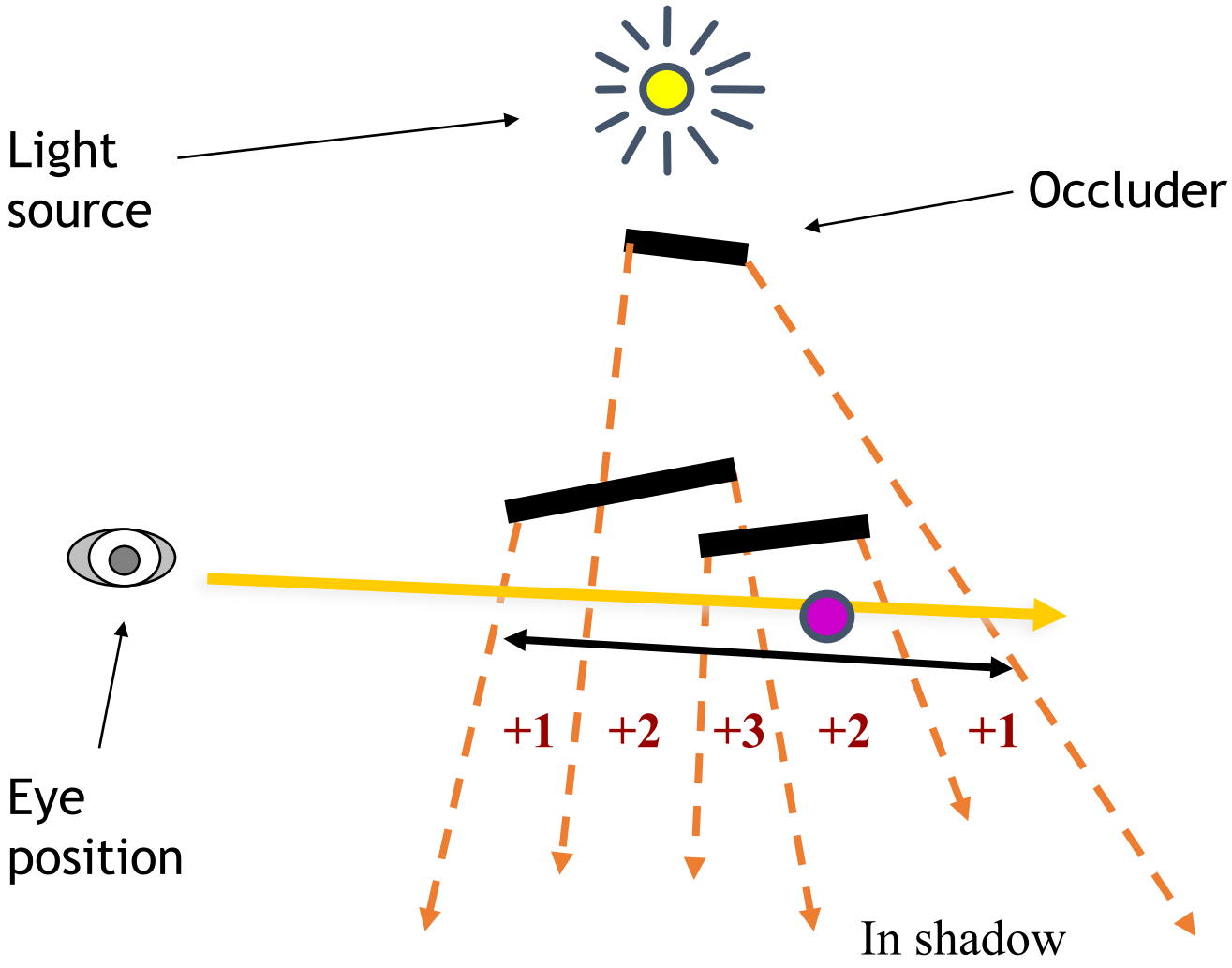
Shadow volumes



In shadow or not

- Test if surface visible in given pixel is inside or outside shadow volume
  1. Allocate a counter per pixel
  2. Cast a ray into the scene, starting from eye, going through given pixel
  3. Increment the counter when the ray enters the shadow volume
  4. Decrement the counter when the ray leaves the shadow volume
  5. When we hit the object, check the counter.
     - If counter > 0, in shadow
     - Otherwise, not in shadow

In shadow or not

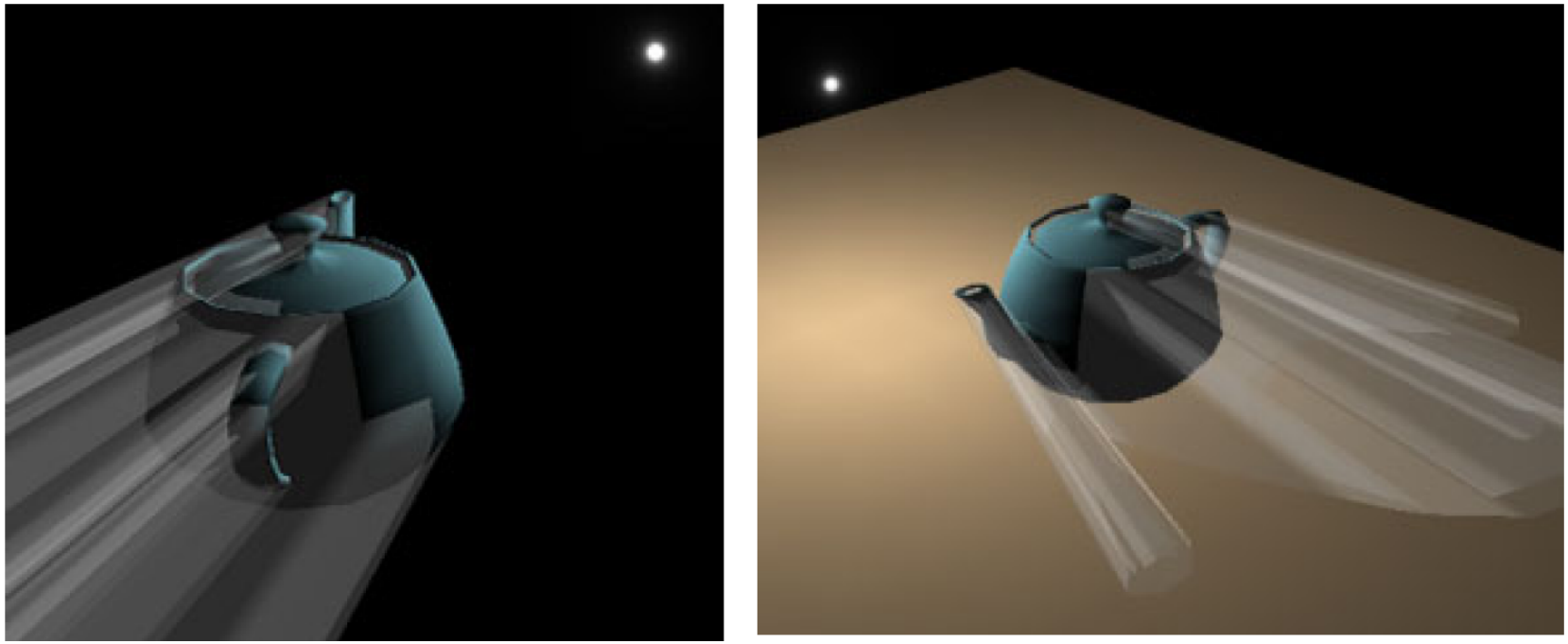


Implementation in rendering pipeline

- Ray tracing not possible to implement directly
- Use a few tricks...

Shadow volume construction

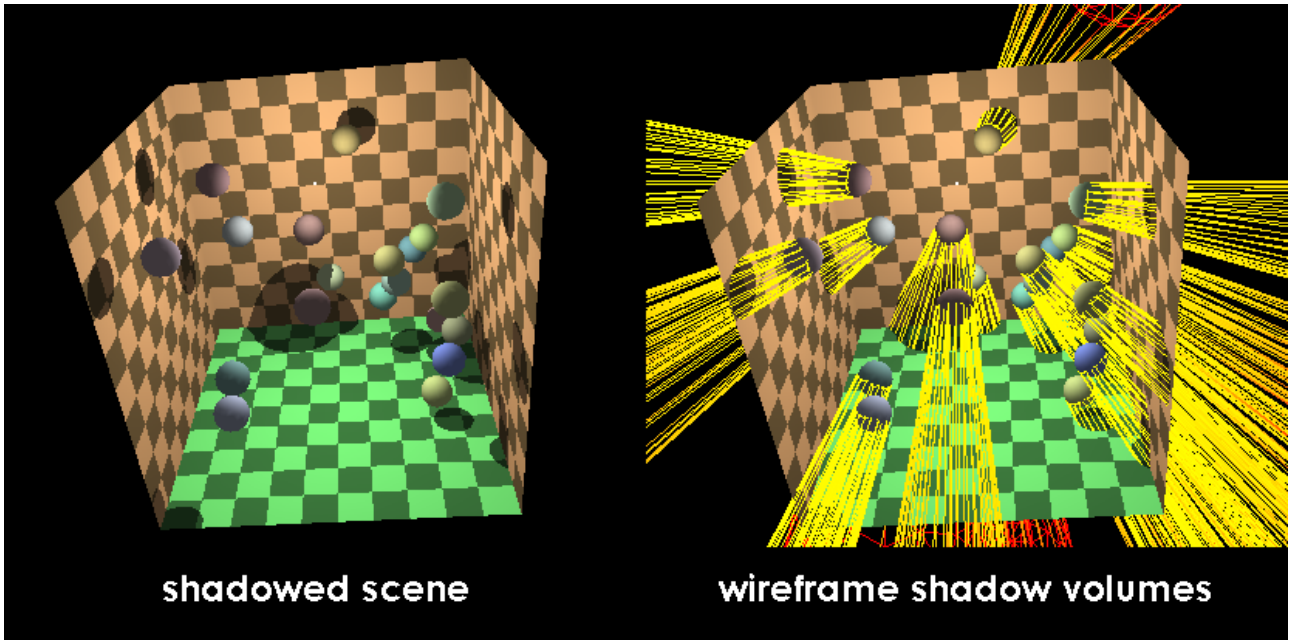
- Need to generate shadow polygons to bound shadow volume
- Extrude silhouette edges from light source



Extruded shadow volumes

Shadow volume construction

- Needs to be done on the CPU
- Silhouette edge detection
  - An edge is a silhouette if one adjacent triangle is front facing, the other back facing with respect to the light
- Extrude polygons from silhouette edges



Shadow test without ray tracing

Using the stencil buffer

- A framebuffer channel (like RGB colors, depth) that contains a per-pixel counter (integer value)
- Available in OpenGL
- Stencil test
  - Similar to depth test (z-buffering)
  - Control whether a fragment is discarded or not
  - Stencil function: is evaluated to decide whether to discard a fragment
  - Stencil operation: is performed to update the stencil buffer depending on the result of the test

Shadow volume algorithms

Z-pass approach

- Count leaving/entering shadow volume events as described
- Use stencil buffer to count number of visible (i.e. not occluded from camera) front-facing and back facing shadow volume polygons for each pixel
- If equal, pixel is not in shadow

Z-fail approach

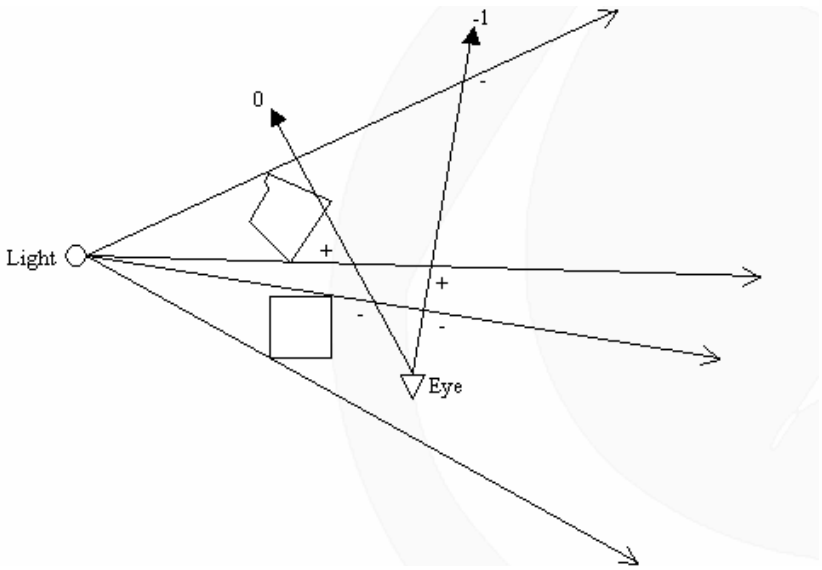
- Count number of invisible (i.e. occluded from camera) front-facing and back-facing shadow volume polygons
- If equal, pixel is not in shadow

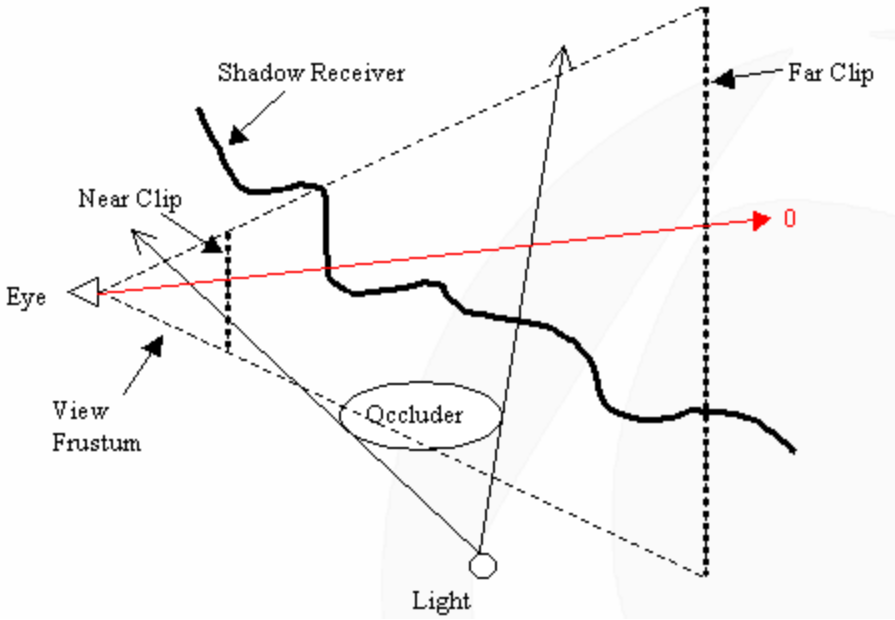
Z-pass approach: details

- Render scene with only ambient light
  - Update depth buffer
- Turn off depth and color write, turn on stencil, keep the depth test on
- Init stencil buffer to 0
- Draw shadow volume twice using face culling
  - 1st pass: render front faces and increment stencil buffer when depth test passes
  - 2nd pass: render back faces and decrement when depth test passes
- At each pixel
  - Stencil != 0, in shadow
  - Stencil = 0, lit
- Render the scene again with diffuse and specular lighting
  - Write to framebuffer only pixels with stencil = 0

Issues

- Z-pass fails if
  - Eye is in shadow
  - Shadow polygon clipped by near clip plane





Shadow volumes

- Pros
  - Does not require hardware support for shadow mapping
  - Pixel accurate shadows, no sampling issues
- Cons
  - More CPU intensive (construction of shadow volume polygons)
  - Fill-rate intensive (need to draw many shadow volume polygons)
  - Expensive for complex geometry
  - Tricky to handle all cases correctly
  - Hard to extend to soft shadows

Shadow maps

- Pros:
  - Little CPU overhead
  - No need to construct extra geometry to represent shadows
  - Hardware support
  - Can fake soft shadows easily
- Cons:
  - Sampling issues
  - Depth bias is not completely foolproof
- Shadow mapping has become more popular with better hardware support

Resources

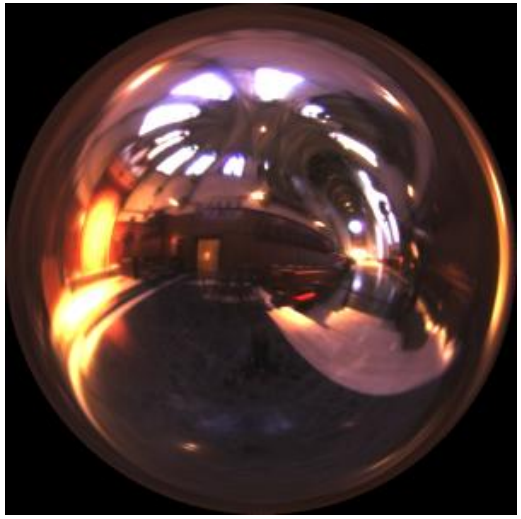
- Overview, lots of links
  http://www.realtimerendering.com/
- Basic shadow maps
  http://en.wikipedia.org/wiki/Shadow_mapping
- Avoiding sampling problems in shadow maps
  http://www.comp.nus.edu.sg/~tants/tsm/tsm.pdf
  http://www.cg.tuwien.ac.at/research/vr/lispsm/
- Faking soft shadows with shadow maps
  http://people.csail.mit.edu/ericchan/papers/smoothie/
- Alternative: shadow volumes
  http://en.wikipedia.org/wiki/Shadow_volume

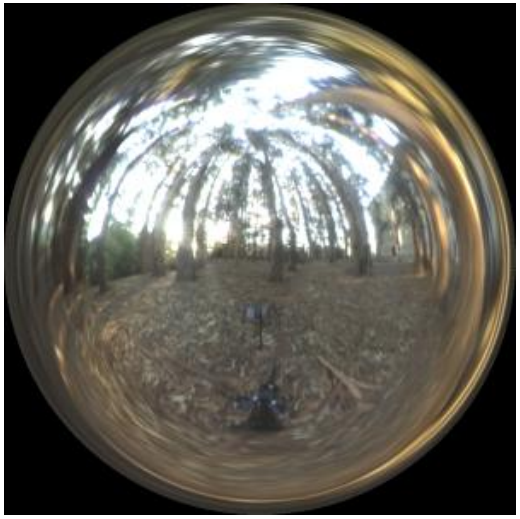
More realistic illumination

- In real world, at each point in scene light arrives from all directions
  - Not just from point light sources
- Environment maps
  - Store “omni-directional” illumination as images
  - Each pixel corresponds to light from a certain direction

Capturing environment maps

- “360 degrees” panoramic image
- Instead of 360 degrees panoramic image, take picture of mirror ball (light probe)





Light probes
[Paul Debevec, http://www.debevec.org/Probes/]

Environment maps as light sources

Simplifying assumption

- Assume light captured by environment map is emitted infinitely far away
- Environment map consists of directional light sources
  - Value of environment map is defined for each direction, independent of position in scene
- Use single environment map as light source at all locations in the scene
- Approximation!

Environment maps as light sources

- How do you compute shading of a diffuse surface using an environment map?
- What is more expensive to compute, shading a diffuse or a specular surface?

Environment maps applications

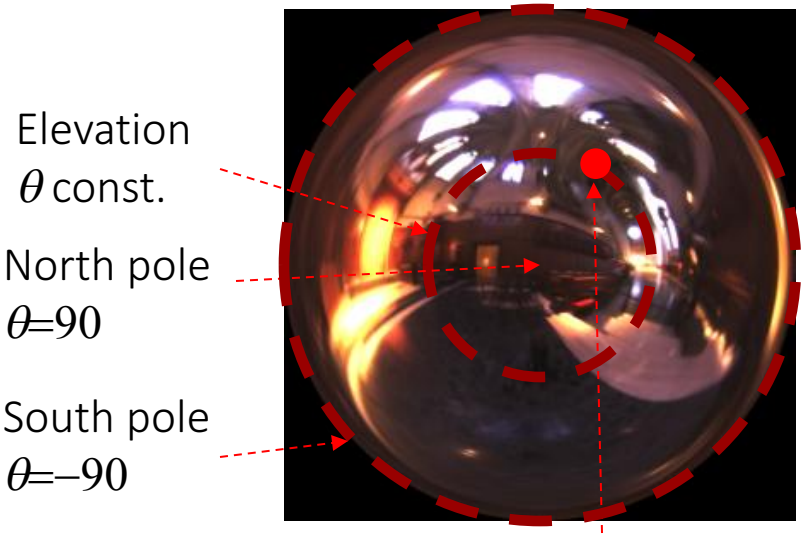
- Use environment map as “light source”

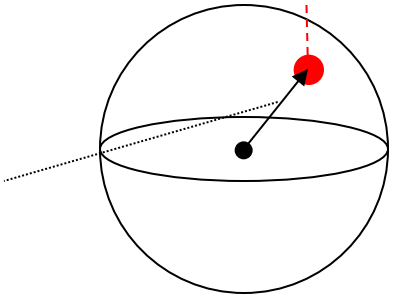
Global illumination
[Sloan et al.]

Reflection mapping

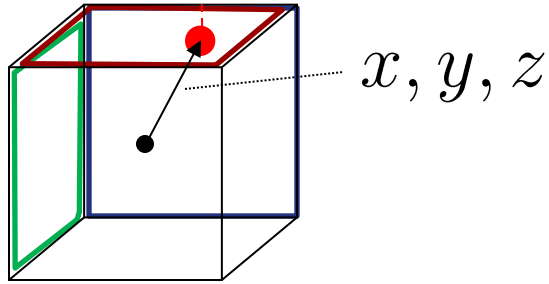
Sphere & cube maps

- Store incident light on sphere or on six faces of a cube





Spherical map



Cube map

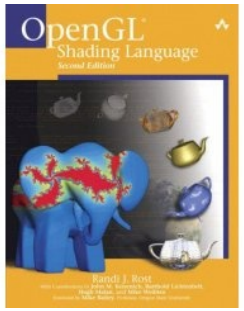
Cube maps in OpenGL

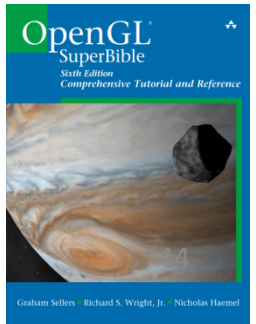
Application setup

- Load, bind a cube environment map

```
glBindTexture(GL_TEXTURE_CUBE_MAP, …);
// the six cube faces
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X,…);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X,…);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y,…);
…
glEnable(GL_TEXTURE_CUBE_MAP);
```

- More details
  - "OpenGL Shading Language", Randi Rost
  - "OpenGL Superbible", Sellers et al.
  - Online tutorials





Cube maps in OpenGL

Look-up

- Given direction (x,y,z)
- Largest coordinate component determines cube map face
- Dividing by magnitude of largest component yields coordinates within face
- Look-up function built into GLSL
  - Use (x,y,z) direction as texture coordinates to `samplerCube`

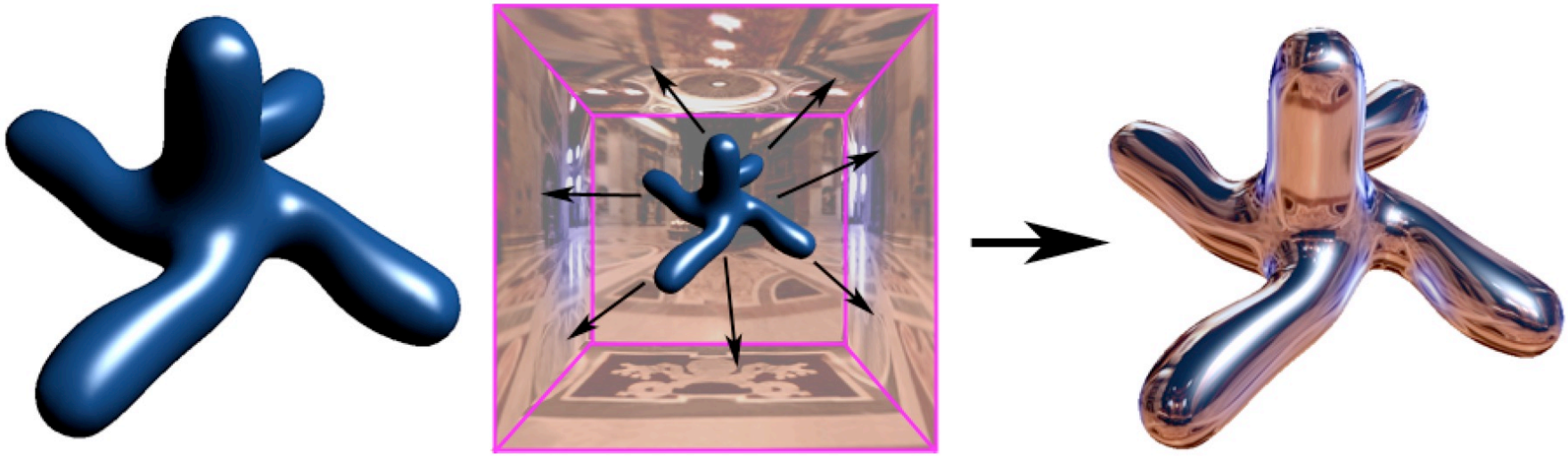
Environment map data

- Also called „light probes“
  http://www.debevec.org/Probes/
- Tool for high dynamic range data (HDR)
  http://projects.ict.usc.edu/graphics/HDRShop/
- Pre-rendered light probes for games
  http://docs.unity3d.com/Manual/LightProbes.html

Light probes (http://www.debevec.org/Probes/)

Reflection mapping

- Simulate mirror reflection
- Compute reflection vector at each pixel using view direction and surface normal
- Use reflection vector to look up cube map
- Rendering cube map itself is optional



Reflection mapping

Reflection mapping in GLSL

Vertex shader

- Compute viewing direction for each vertex
- Reflection direction
  - Use GLSL built-in `reflect` function
- Pass reflection direction to fragment shader

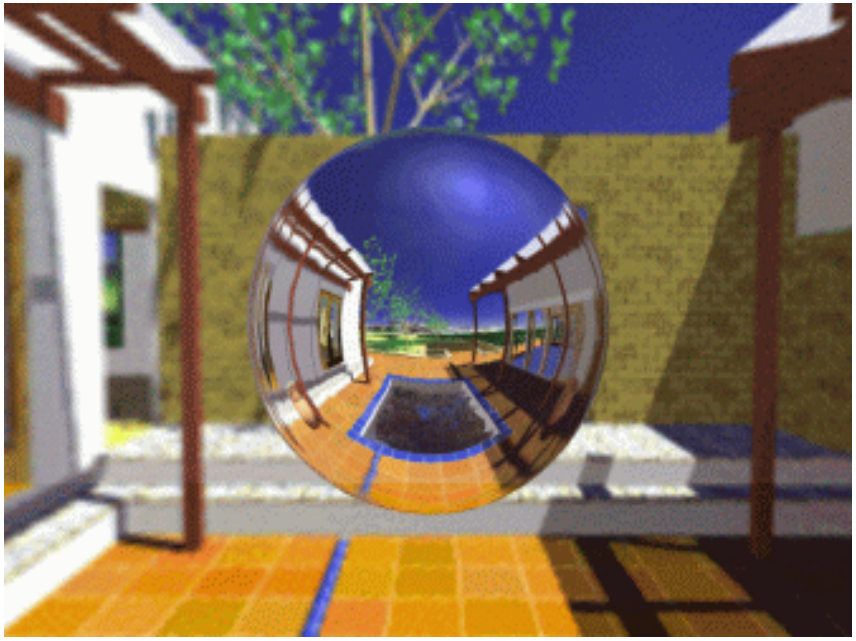
Fragment shader

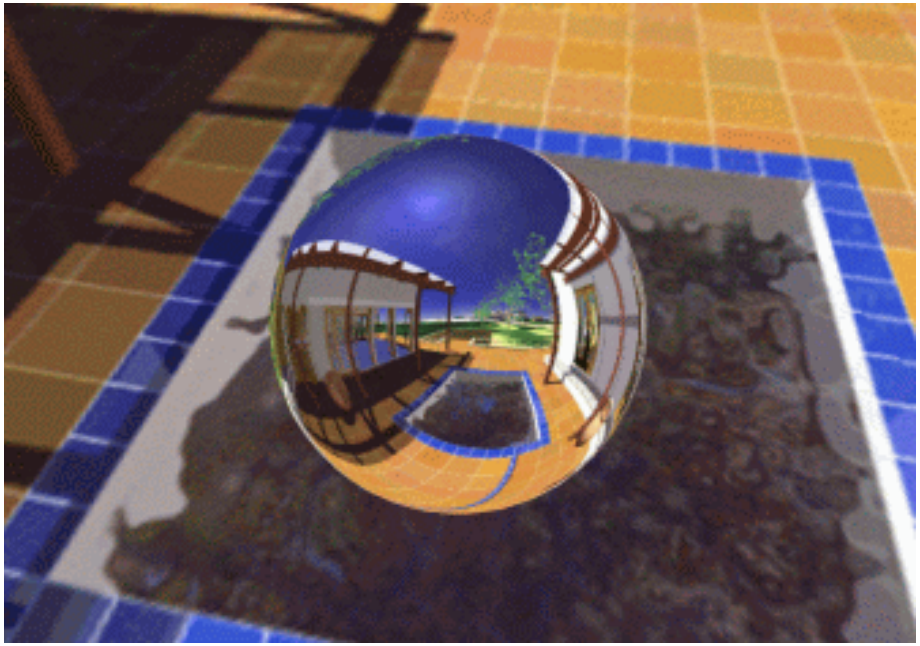
- Look-up cube map using interpolated reflection direction

```
in float3 refl;
uniform samplerCube envMap;
texture(envMap, refl);
```

Reflection mapping examples

- Approximation, reflections are not accurate



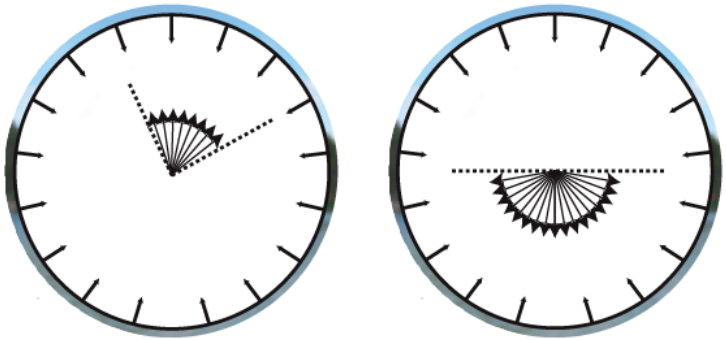


[NVidia]

Shading using environment map

- Assumption: distant lighting
  - Incident light is a function of direction, but not position
- Realistic shading requires
  - Take into account light from all directions
  - Include occlusion

Illumination from environment



Same environment map for both points
"Illumination is a function of direction, but not position"

Mathematical model

- Assume Lambertian (diffuse) material, BRDF k_d
  - Ignore occlusion for now
- Illumination from point light sources

$$c = k_d \sum_i c_{l_i} (\mathbf{L}_i \cdot \mathbf{n})$$

- Illumination from environment map using hemispherical integral

$$c = k_d \int_\Omega c(\omega)(\omega \cdot \mathbf{n}) d\omega$$

  - Directions ω
  - Hemisphere of directions Ω
  - Environment map, radiance from each direction $c(\omega)$

Irradiance environment maps

- Precompute irradiance as a function of normal

$$E(\mathbf{n}) = \int_{\Omega} c(\omega)(\omega \cdot \mathbf{n}) d\omega$$

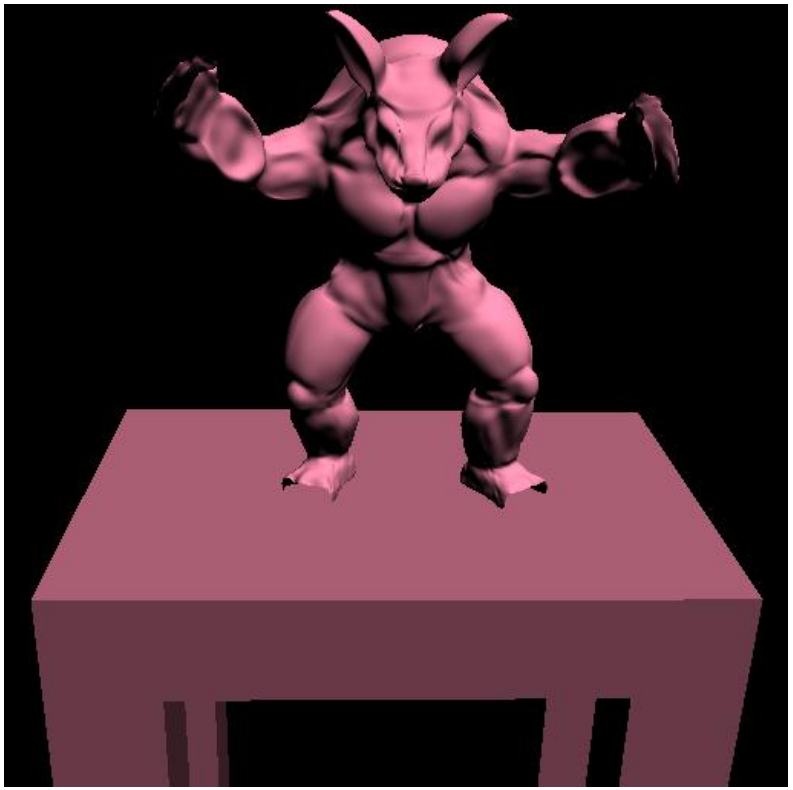
  - Store as irradiance environment map
- Shading computation at render time
  - Depends only on normal, not position

$$c = k_d E(\mathbf{n})$$

Environment map

Irradiance map

Irradiance environment maps



Directional light

Environment illumination

Images from http://www.cs.berkeley.edu/~ravir/papers/envmap/

Implementation

- Precompute irradiance map from environment
  - HDRShop tool, “diffuse convolution” http://projects.ict.usc.edu/graphics/HDRShop/
- At render time, look up irradiance map using surface normal
  - When object rotates, rotate normal accordingly
- Can also approximate glossy reflection
  - Blur environment map less heavily
  - Look up blurred environment map using reflection vector

Today

More shading

- Environment maps
- Reflection mapping
- Irradiance environment maps
- Ambient occlusion
- Reflection and refraction
- Toon shading

Including occlusion

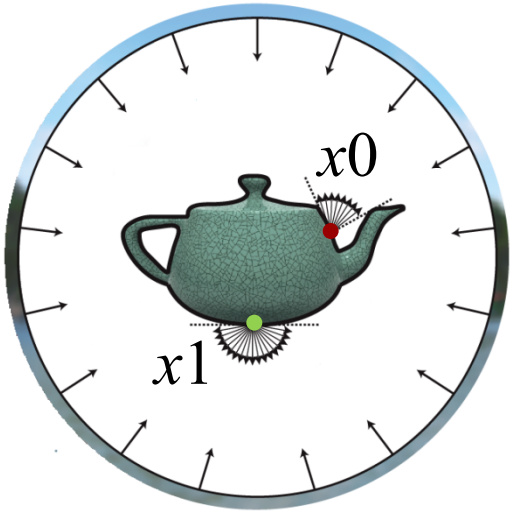
- At each point, environment is partially occluded by geometry
- Add light only from un-occluded directions

Visualization of un-occluded directions

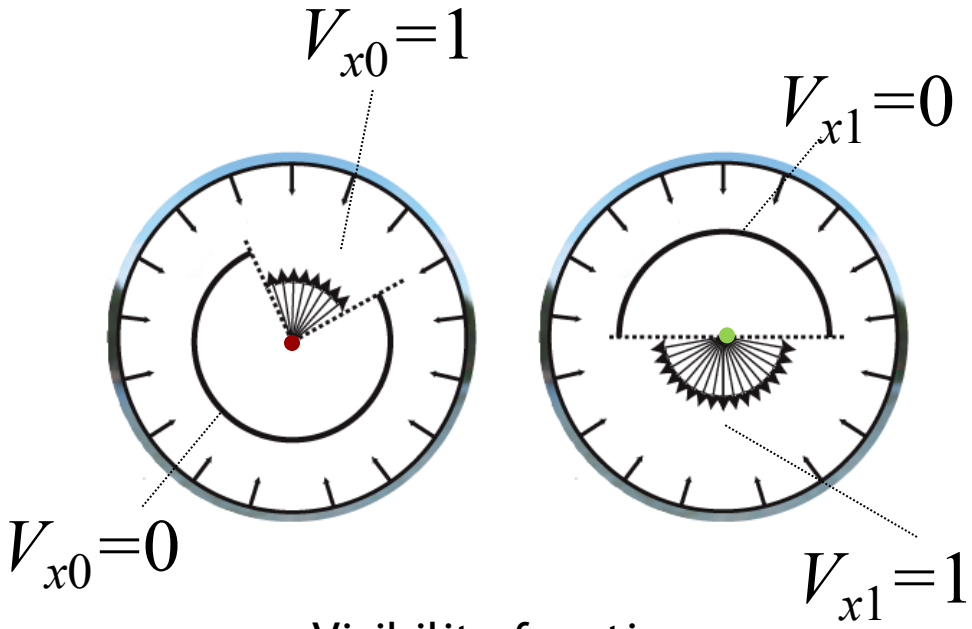
Including occlusion

Visibility function $V_x(\omega)$

- Binary function of direction ω
- Indicates if environment is occluded
- Depends on position x



Environment map



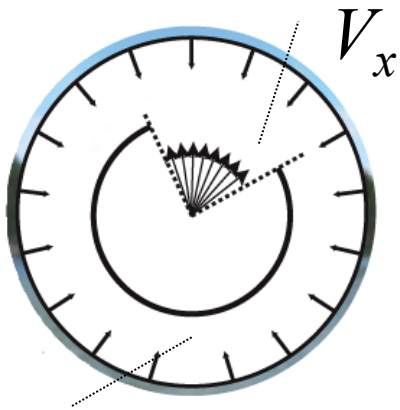
Visibility functions

Mathematical model

- Diffuse illumination with visibility

$$c = k_d \int_\Omega V_x(\omega) c(\omega)(\omega \cdot \mathbf{n}) d\omega$$

- Ambient occlusion
  - "Fraction" of environment that is not occluded from a point x
  - Scalar value

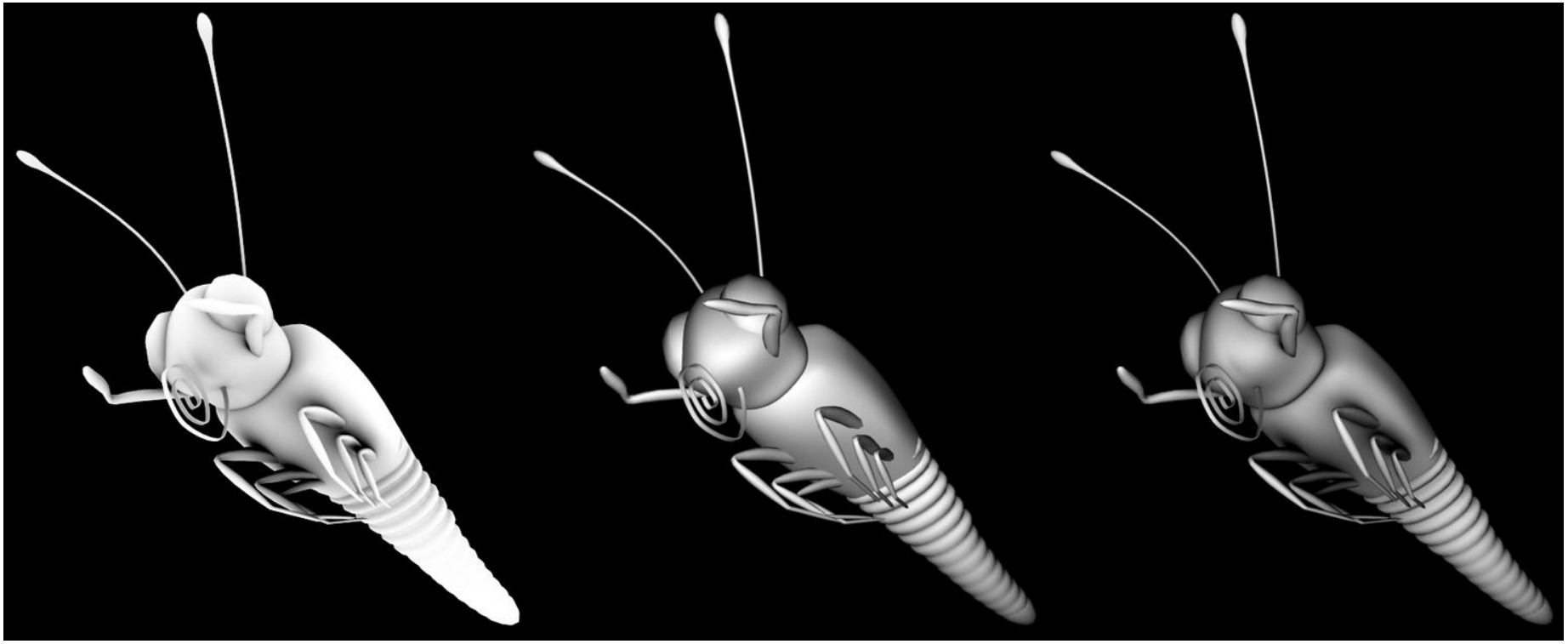


- Approximation: diffuse shading given by irradiance weighted by ambient occlusion

$$a_x = \int_\Omega V_x(\omega)(\omega \cdot \mathbf{n}) d\omega$$

$$c = k_d a_x E(\mathbf{n}) \qquad E(\mathbf{n}) = \int_\Omega c(\omega)(\omega \cdot \mathbf{n}) d\omega$$

Ambient occlusion



Ambient occlusion

Diffuse shading

Ambient occlusion combined (using multiplication) with diffuse shading

http://en.wikipedia.org/wiki/Ambient_occlusion

Implementation

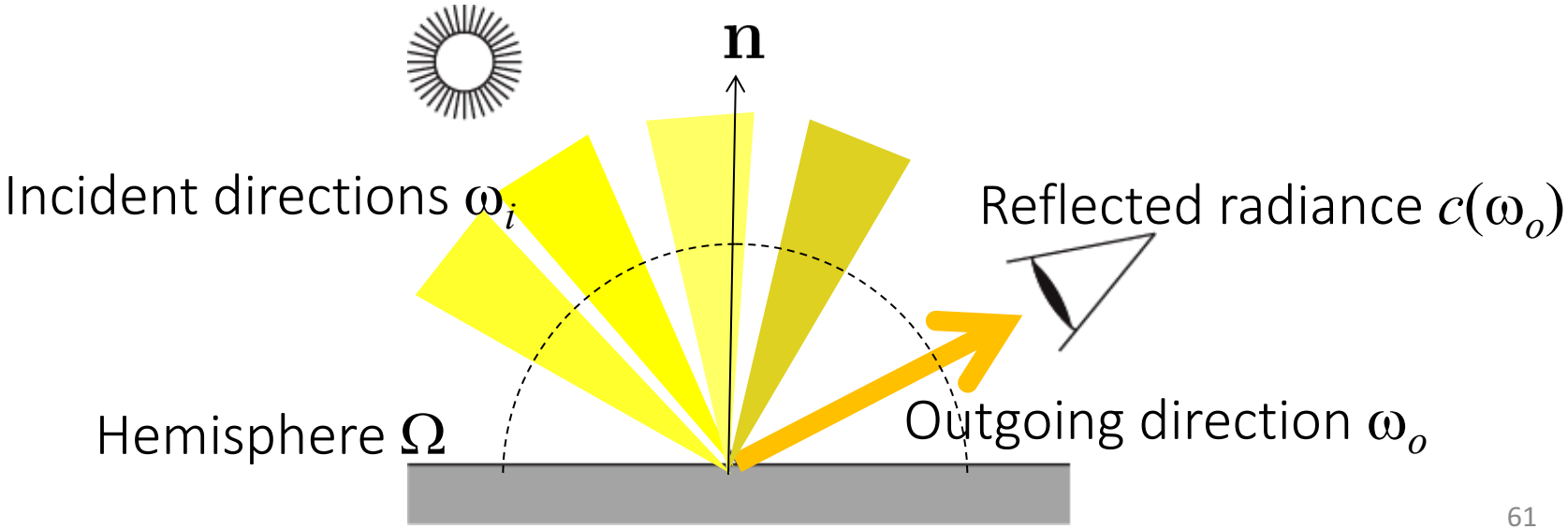
- Precomputation (off-line, before rendering)
  - Compute ambient occlusion on a per-vertex basis
  - Using ray tracing
  - Free tool that saves meshes with per-vertex ambient occlusion
    http://www.xnormal.net/
- Caution
  - Basic pre-computed ambient occlusion does not work for animated objects

Shading integral

- Ambient occlusion with irradiance environment maps is crude approximation to general shading integral

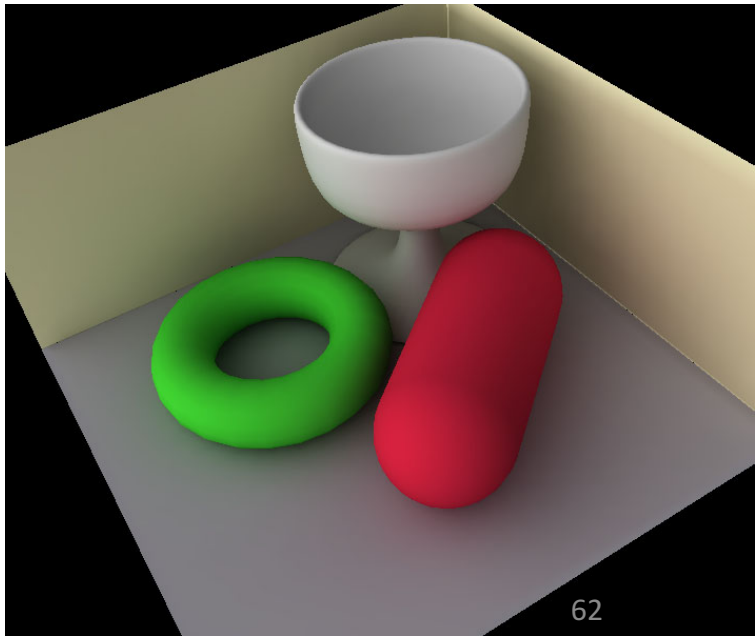
$$c(\omega_o) = \int_\Omega V_x(\omega_i) c(\omega_i) f(\omega_o, \omega_i)(\omega_i \cdot \mathbf{n}) d\omega_i$$

  - BRDF for (non-diffuse) material $f(\omega_o, \omega_i)$



Shading integral

- Accurate evaluation is expensive to compute
  - Requires numerical integration
- Many tricks for more accurate and general approximation than ambient occlusion and irradiance environment maps exist
  - Spherical harmonics shading
    http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.pdf



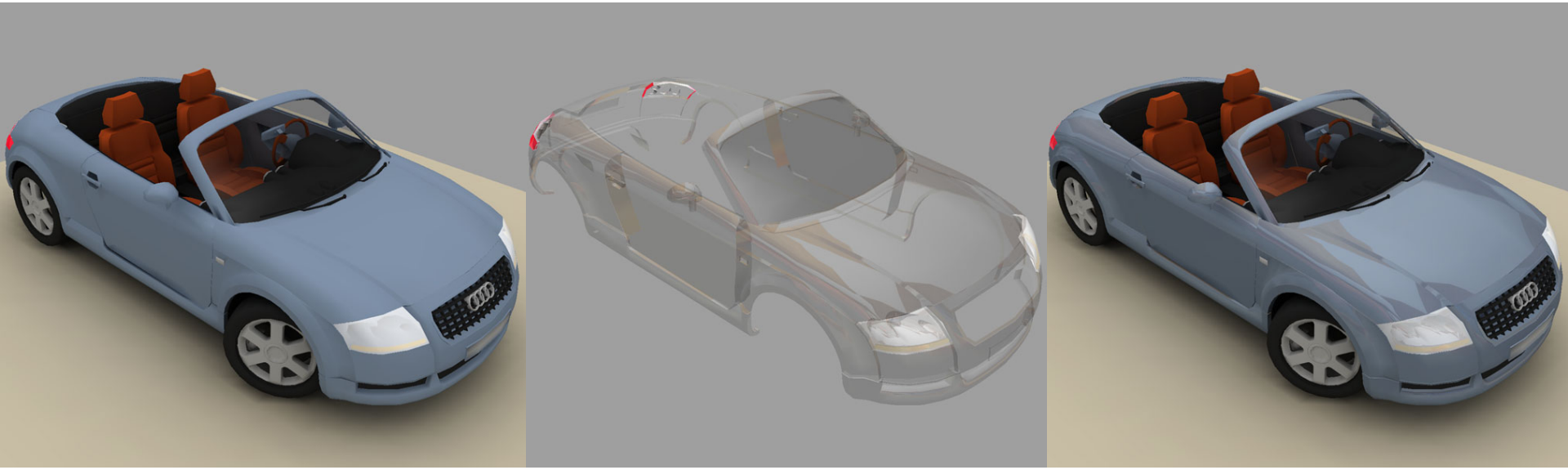
Note

- Visually interesting results using combination (sum) of diffuse shading with ambient occlusion and reflection mapping

Diffuse shading with ambient occlusion

Reflection mapping

Combination (sum)



http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.pdf

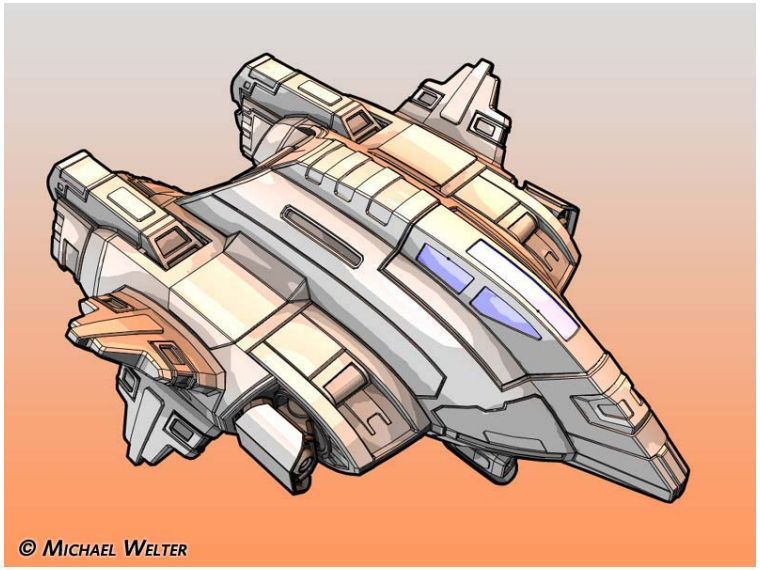
Today

More shading

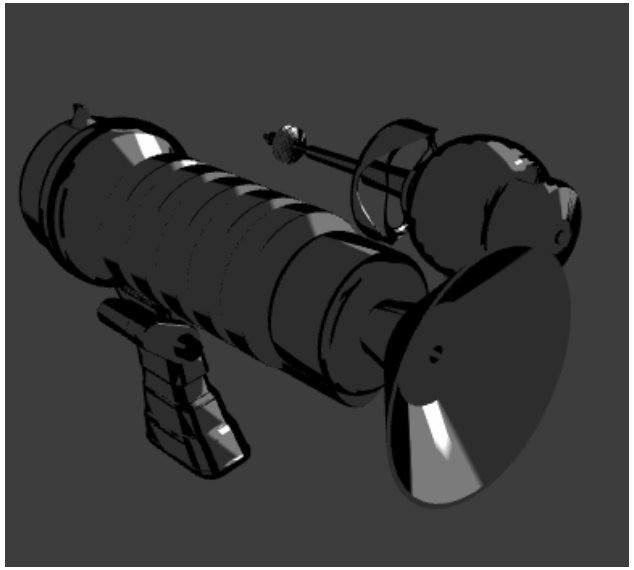
- Environment maps
- Reflection mapping
- Irradiance environment maps
- Ambient occlusion
- Reflection and refraction
- Toon shading

Toon shading

- Simple cartoon style shader
- Emphasize silhouettes
- Discrete steps for diffuse shading, highlights
- Sometimes called CEL shading
  http://en.wikipedia.org/wiki/Cel-shaded_animation



Off-line toon shader


GLSL toon shader

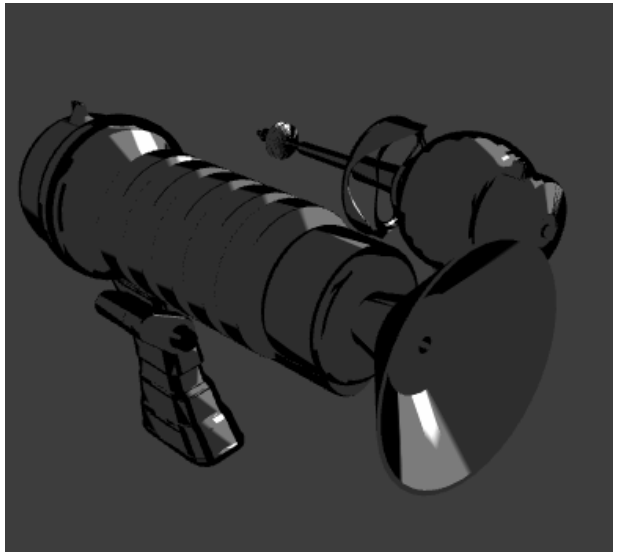
Toon shading

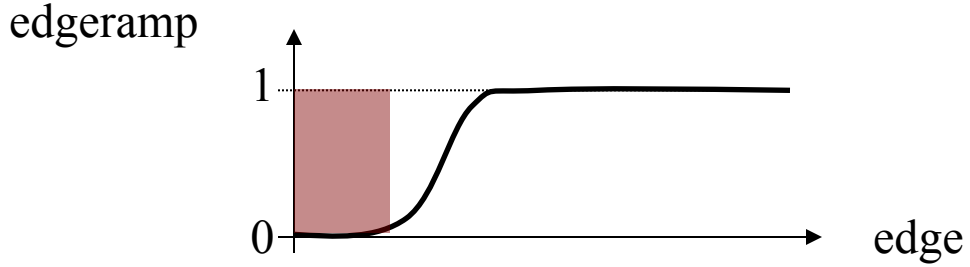
- Silhouette edge detection
  - Compute dot product of viewing direction **v** and normal **n**

$$\text{edge} = \max(0, \mathbf{n} \cdot \mathbf{v})$$

  - Use 1D texture to define edge ramp

```
uniform sample1D edgeramp;
e=texture1D(edgeramp,edge);
```





Toon shading

- Compute diffuse and specular shading

$$\text{diffuse} = \mathbf{n} \cdot \mathbf{L} \qquad \text{specular} = (\mathbf{n} \cdot \mathbf{h})^s$$

- Use 1D textures diffuseramp, specularramp to map diffuse and specular shading to colors
- Final color

```
uniform sampler1D diffuseramp;
uniform sampler1D specularramp;
c = e * (texture(diffuse,diffuseramp)+
         texture(specular,specularramp));
```