



**SIGGRAPH**2004

**Shadow Mapping**

Marc Stamminger, University of Erlangen-Nuremberg

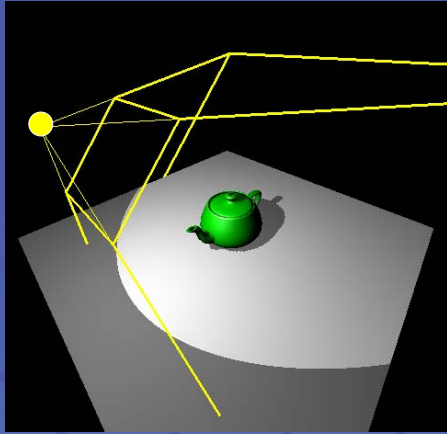
## Idea



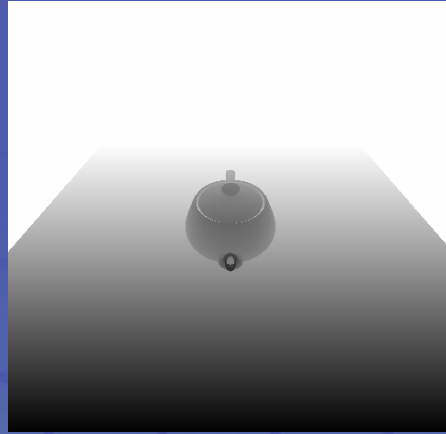
- assumption:  
spot light with spot angle  $\ll 180^\circ$
- render scene from the view of the light
  - camera position = light position
  - camera direction = spot direction
  - opening angle = spot angle
  - light frustum
- shadow map = resulting depth buffer

Mostly, shadow maps are generated for spot lights. All scene points illuminated by such a spot light are also visible by a perspective camera in the light source. We thus render the scene from the view of the light source and store the depth buffer as „shadow map“.

# Idea



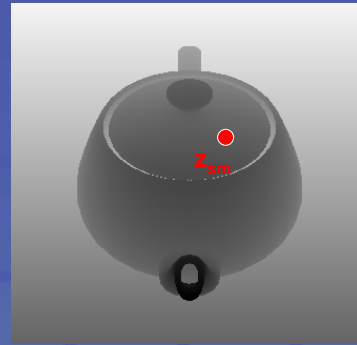
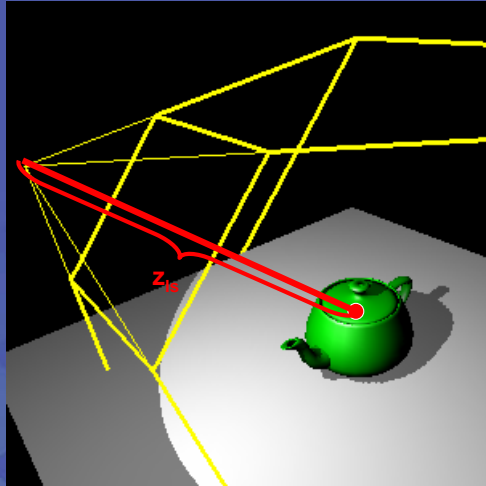
observer's view with light frustum



shadow map

On the left one can see a simple scene, illuminated by a light source from the left. The corresponding shadow map is on the right.

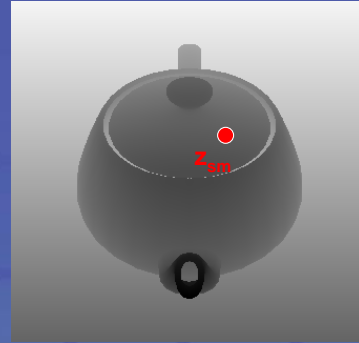
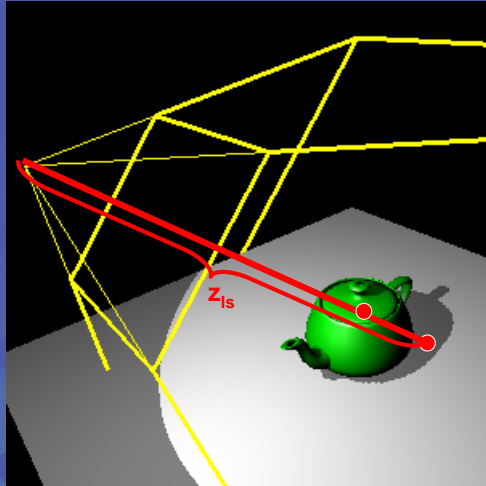
# Example



$z_{ls} = z_{sm} \rightarrow$  point lit

In order to determine whether a point is in shadow, we compute its distance to the light source  $z_{ls}$  and compare it with the corresponding depth value  $z_{sm}$  in the shadow map. If  $z_{ls} = z_{sm}$ , the point is visible from the light and thus lit.

# Example



$z_{ls} > z_{sm} \rightarrow$  point in shadow

Another point behind the teapot has a larger distance to the light  $z_{ls}$ . Its projection in the shadow map is at the same position, but this time  $z_{ls} > z_{sm}$ . Thus the point is in shadow.

## Idea



- per pixel  $(x,y)$  of the camera view
  - point in camera space is  $(x,y,\text{depth}(x,y))$
  - map back to world space
  - map to light source space  $\rightarrow (x_{ls}, y_{ls}, z_{ls})$
  - compare  $z_{ls}$  with shadow map value at  $(x_{ls}, y_{ls})$

This is what we have to do per pixel more precisely.

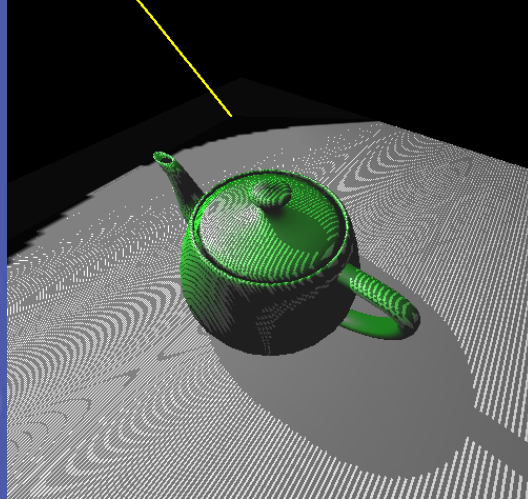
# Idea



- demo „shadowmaps“

# Bias

- bias needed
  - if point is lit
$$z_{ls} \approx z_{sm}$$
  - if we test for
$$z_{ls} > z_{sm},$$
also lit points are shadowed
- test for
$$z_{ls} > z_{sm} + \text{bias}$$

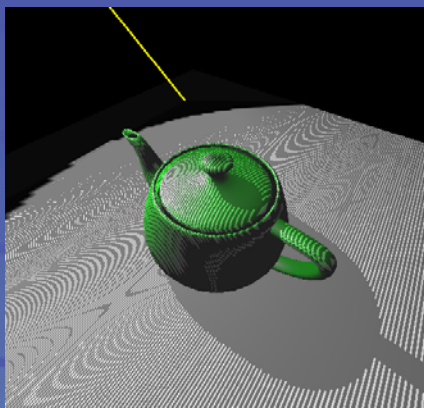


The „lit“ condition  $z_{ls} = z_{sm}$  is usually not fulfilled for lit points, because of numerical inaccuracies. So we have to define a corridor around  $z_{sm}$  and classify all points as lit, as long as  $|z_{ls} - z_{sm}| < \text{epsilon}$ . Because the case  $z_{ls} < z_{sm}$  cannot appear (except for numerical reasons), we can replace the test by  $z_{ls} > z_{sm} + \text{bias}$ .

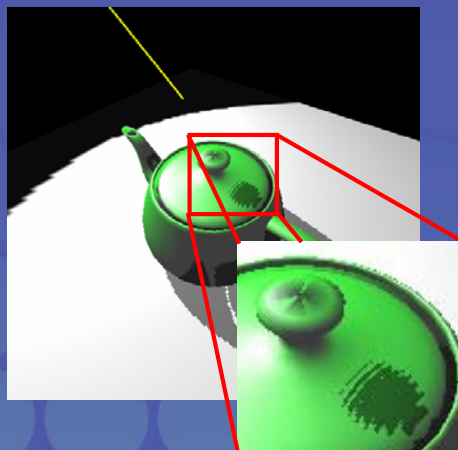


# Bias

- bias too small → surface acne



- bias too large → shadow leaks



The bias has to be chosen carefully. Too small bias results in surface acne, where surfaces shadow themselves. If the bias is too big, shadows get lost if the occluder and shadow receiver are close.

# Bias

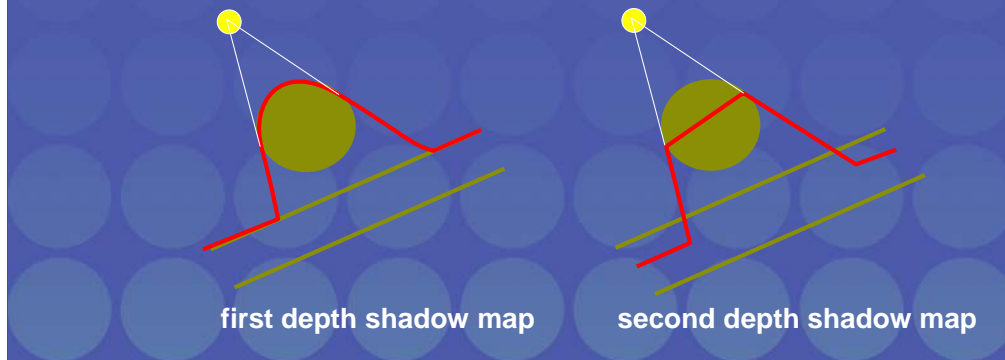


- better solution:  
use PolygonOffset when rendering the shadow map  
→ offset is adapted to surface slope

polygon offset as supported by normal hardware is an even better means for biasing, because it also considers surface slope.

# Bias

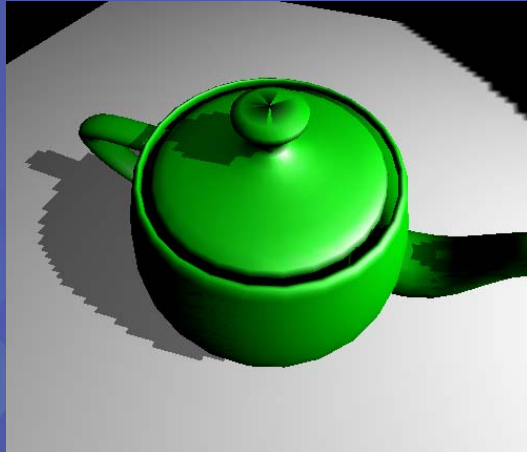
- even more robust:  
Second Depth Shadow Maps
  - store in shadow map average depth of first and second visible surface



Second depth shadow maps use the average between first and second visible surface from the view of the light source. By this, the shadow map depth values have maximum distance to the lit and the first shadowed surfaces. On the downside, the generation is more expensive and requires to generate the first and second visible layer from the view of the light source.

# Aliasing

- finite shadow map resolution  
→ pixelized shadows



The major problem of shadow maps is aliasing. Because of the image based nature of the shadow map, shadows appear pixelized, when shadow map and camera image resolution mismatch.

# Aliasing

- percentage closer filtering
  - filter shadow test result 0/1 instead of depth



When filtering shadow maps, it is important not to filter the depth values, but the comparison results.

# Aliasing

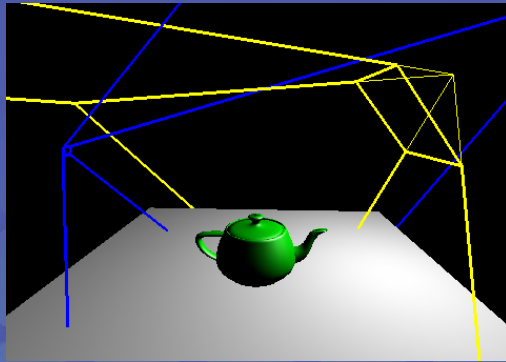


- bad aliasing cases
  - large scenes
    - high shadow map resolution required
  - close-ups to shadow boundaries
  - shadow stretches along receiver

Aliasing is particularly apparent for several cases. In particular for large outdoor scenes, standard shadow maps are almost useless. Furthermore, when we zoom into a shadow boundary, we can always recognize the pixelized shadow structure at some point.

# Aliasing

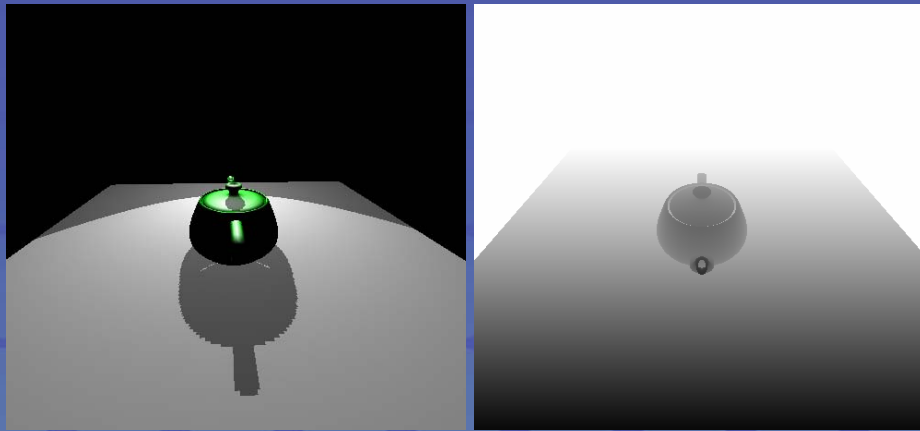
- duelling frusta
  - light shines opposite to viewing direction



A typical bad case are duelling frusta.

# Aliasing

- duelling frusta  
– resolution mismatch

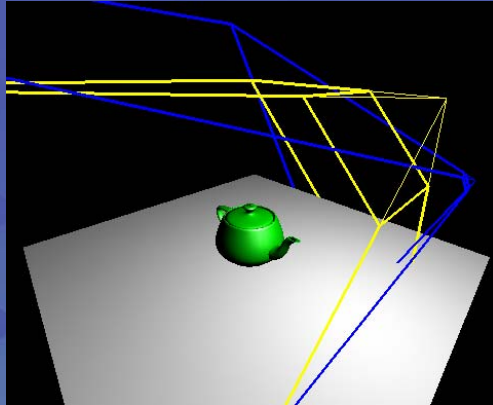


For duelling frusta, the resolution mismatch is maximal. The regions that are large in the camera view, are small in the shadow map and vice versa. As a result, the shadow of the handled is clearly pixelized.



# Aliasing

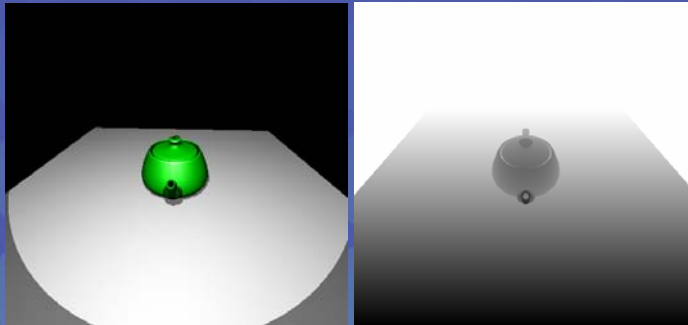
- best case: miner's headlamp  
– light position close to camera



The optimal case appears if camera and light source are at similar positions.

# Aliasing

- miner's headlamp
  - similar frusta
  - similar sampling densities
  - „one shadow map pixel per image pixel“



In this case, the resolutions are similarly distributed over the scene, and we get a one-to-one match between shadow map and image pixels.

# Hardware Support



- OpenGL 1.4
  - `GL_ARB_DEPTH_TEXTURE`  
internal texture format to store depth values
  - `glTexGen`  
generation of light space coordinates as texture coordinates
  - `GL_ARB_SHADOW`  
special texture mode:  
`return texture(s,t) < r ? Black : White;`

The big advantage of shadow maps is that they are fully supported by hardware. In order to apply them efficiently, two extensions are necessary, which became part of OpenGL 1.4, as well as the `glTexGen` command.

# Hardware Support



- **P-Buffers**
  - offscreen rendering of shadow map
  - large shadow maps sizes
- **GL\_ARB\_RENDER\_TEXTURE and WGL\_NV\_RENDER\_TEXTURE**
  - bind P-Buffer depth buffer as depth texture
  - no copy needed

Further improvements are possible using P-Buffers and the render to texture functionality.

# Hardware Support



- Register Combiners / Fragment Programs
  - for shadow application
  - if point is in shadow:
    - leave ambient light unchanged
      - ambient light has no origin, thus cannot be shadowed
    - dim diffuse light
      - some shading remains to show surface orientation
    - remove specular light
      - no highlights in shadow

Register combiners or fragment programs can be used to apply the shadows.

# Pros and Cons



- + general
  - everything that can be rendered can cast and receive a shadow
  - works together with vertex programs
- + fast
  - full hardware support
  - (almost) no overhead for static scenes
  - two passes needed for dynamic scenes
- + robust
- + easy to implement
- aliasing

## Extensions



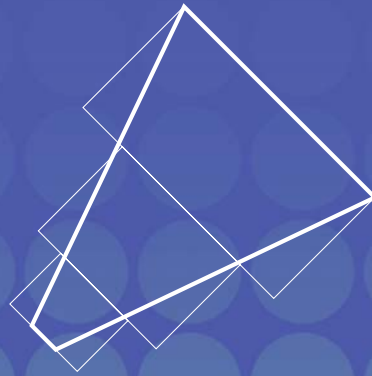
- concentrating on visible part [Brabec02]
  - restrict shadow map to visible part of scene
  - increased shadow map resolution
    - less aliasing
  - shadow map view dependent
    - two rendering passes needed, also for static scenes

Brabec describes an efficient method to restrict the shadow map to these parts of the scene that are visible. No shadow map resolution is wasted for invisible objects. However, by this the shadow map becomes view dependent and must be regenerated per frame, which needs to be done for dynamic scenes anyhow.

## Extensions



- outdoor scenes [Tadamura01]
  - multiple shadow maps along view frustum
  - not interactive



Tadamura use multiple depth buffers to cover the trapezoidal view frustum. The method is not suited for interactive rendering.



## Extensions



- Adaptive Shadow Maps [Fernando01]
  - shadow map as hierarchical quadtree
  - generate only required part of tree
  - many rendering passes needed
  - evaluation of shadow map in software

Adaptive Shadow Maps store a quadtree representation of a very high resolution shadow map. Only part of the quadtree is stored that is needed for the current view. For every frame it is determined, whether a node of the hierarchy is still sufficient or needs to be replaced by higher resolution children. The approach requires several rendering passes per frame, and the evaluation of the shadow map is not possible in hardware (yet?).

## Extensions



- Perspective Shadow Maps [Stamminger02]
  - shadow map sees world after perspective transformation
  - reduced aliasing
  - very large scenes possible
  - see later

Perspective shadow maps exploit a remaining degree of freedom in the generation of shadow maps. When rendering the scene from the view of the light source, previously symmetric frusta have been used. However, by using asymmetric frusta, aliasing effects can be largely reduced in several cases. Other cases, however, remain critical and prone to aliasing. We will discuss perspective shadow maps in detail in the next talk.

## Extensions



- omni-directional light sources
  - use cubical shadow maps
  - six passes needed to generate shadow map
- dual-paraboloid shadow maps [Brabec02]
  - use two parabolic shadow maps for omnidirectional light sources
  - only two passes needed to generate shadow map
  - non-projective mapping
    - fine tessellation assumed

As mentioned in the beginning, shadow maps are usually generated for spot lights, because their illumination solid angle can be covered by a perspective camera frustum. For omnidirectional lights, multiple shadow maps or non-projective mappings must be used.

## Extensions



- Soft Shadows from shadow maps
  - [Agrawala00]
  - [Brabec00]
  - ...
  - see later lesson

Shadow maps can also be used to generate soft shadows. However, this requires rather involved additional computations, and the performance penalty is significant. Later, we will describe one such approach in detail [Brabec00]