北京航空航天大學
人工智能研究院

Computer Graphics
Lecture 13: Shadow Mapping

潘成伟 (Chengwei Pan)

Email: pancw@buaa.edu.cn

Office: Room B1021, New Main Building

北京航空航天大学，人工智能研究院

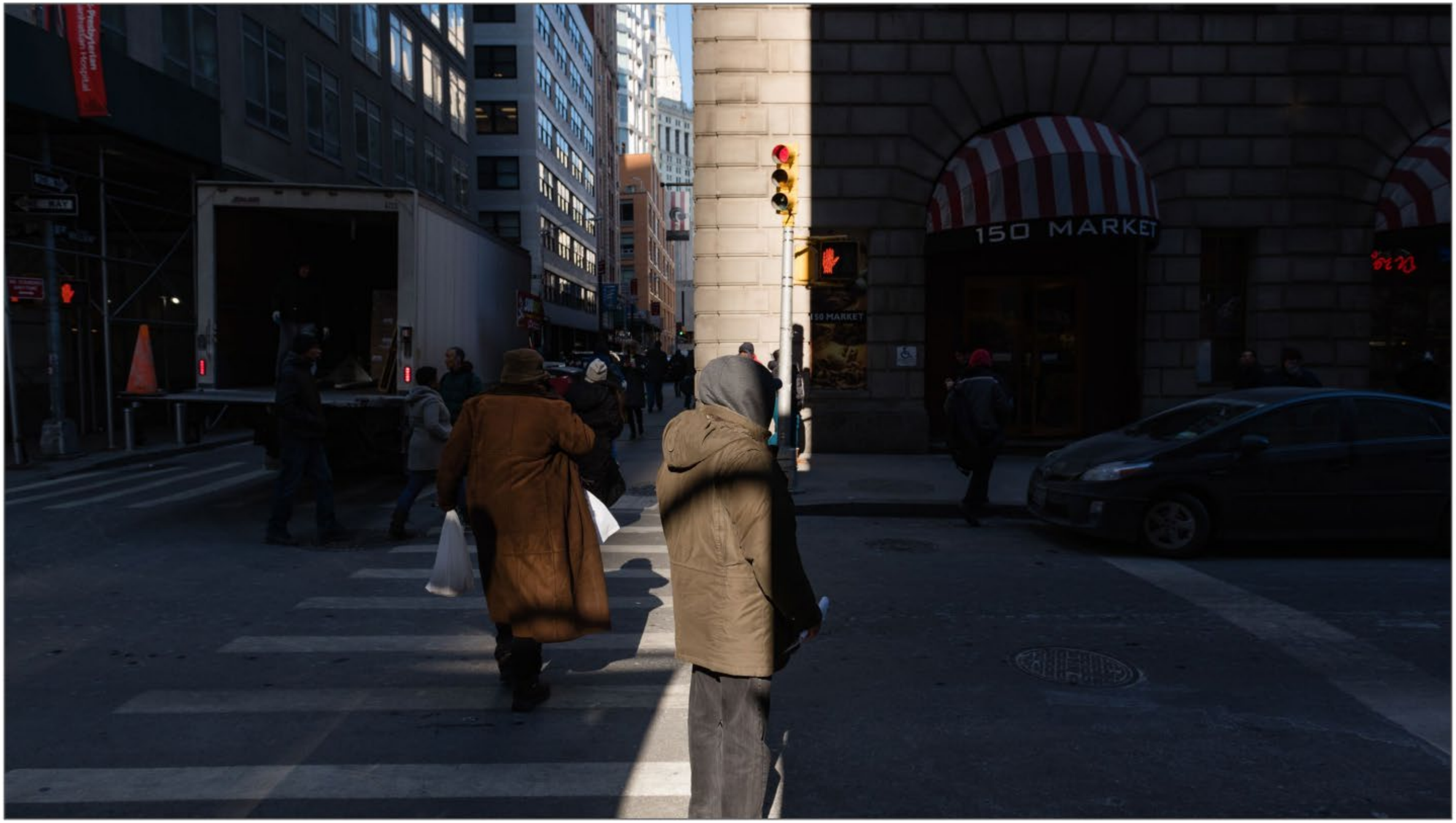
Institute of Artificial Intelligence, Beihang University

This Lecture

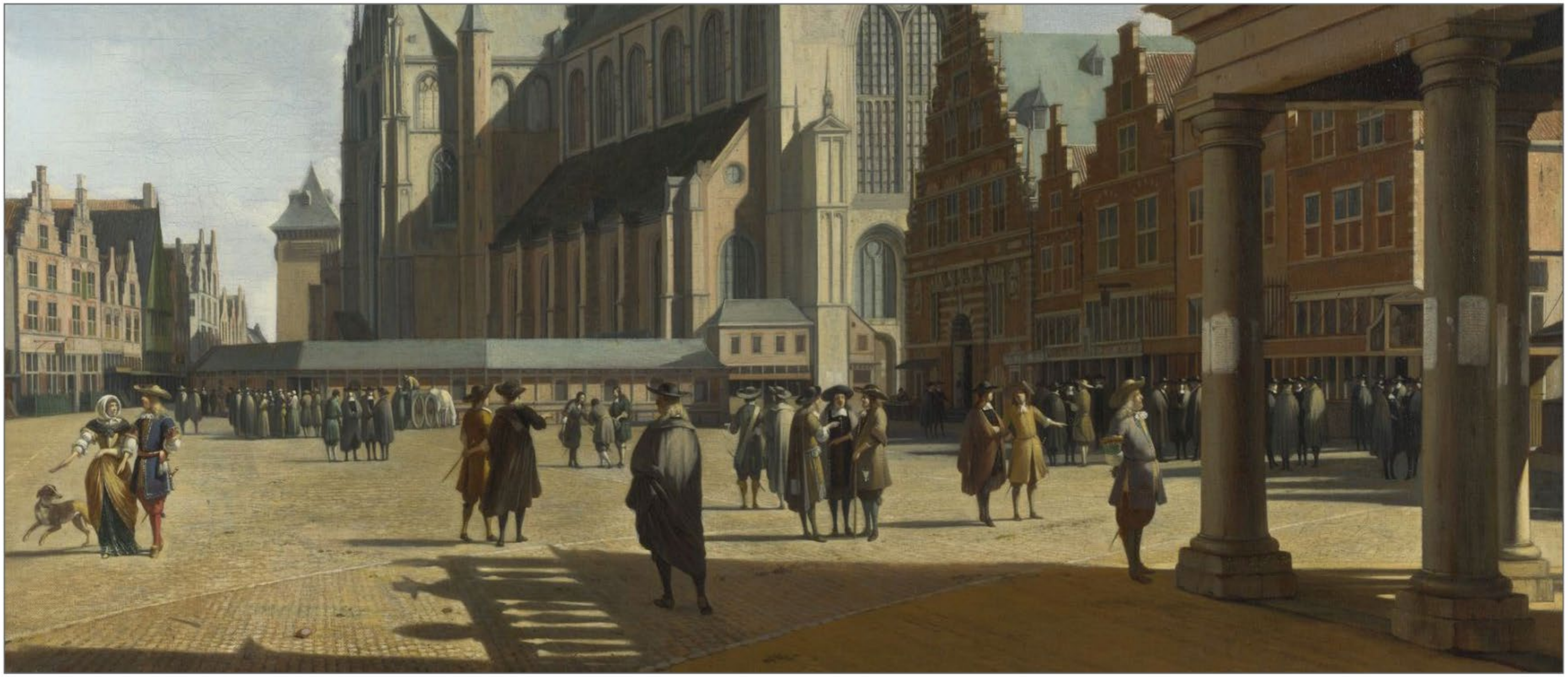
- Shadow
- Shadow Mapping
- Issues in Shadow Mapping

Shadows

Shadows



Shadows

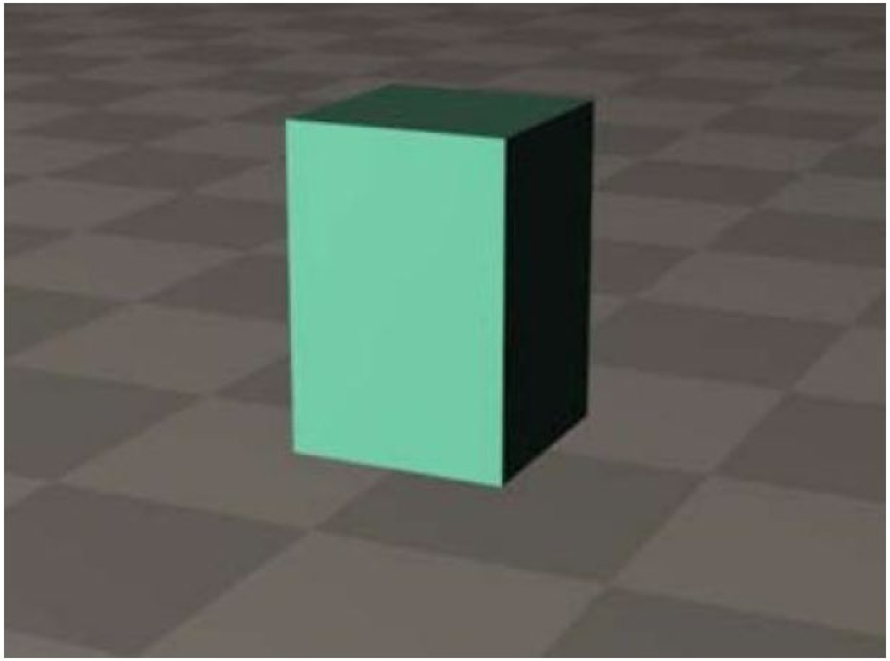


Shadows

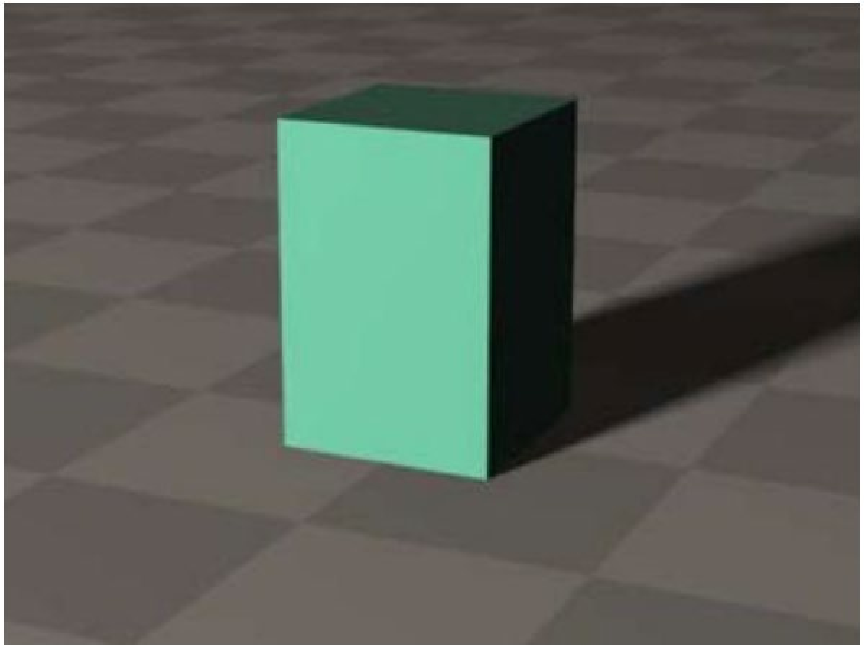
- Increase scene realism
  - real world has shadows
  - depth perception
  - immersive games
    - dramatic effects
    - spooky effects
  - Other art forms use effectively

Shadows

- Shadows indicate spatial relationships between objects e.g. contact with floor.

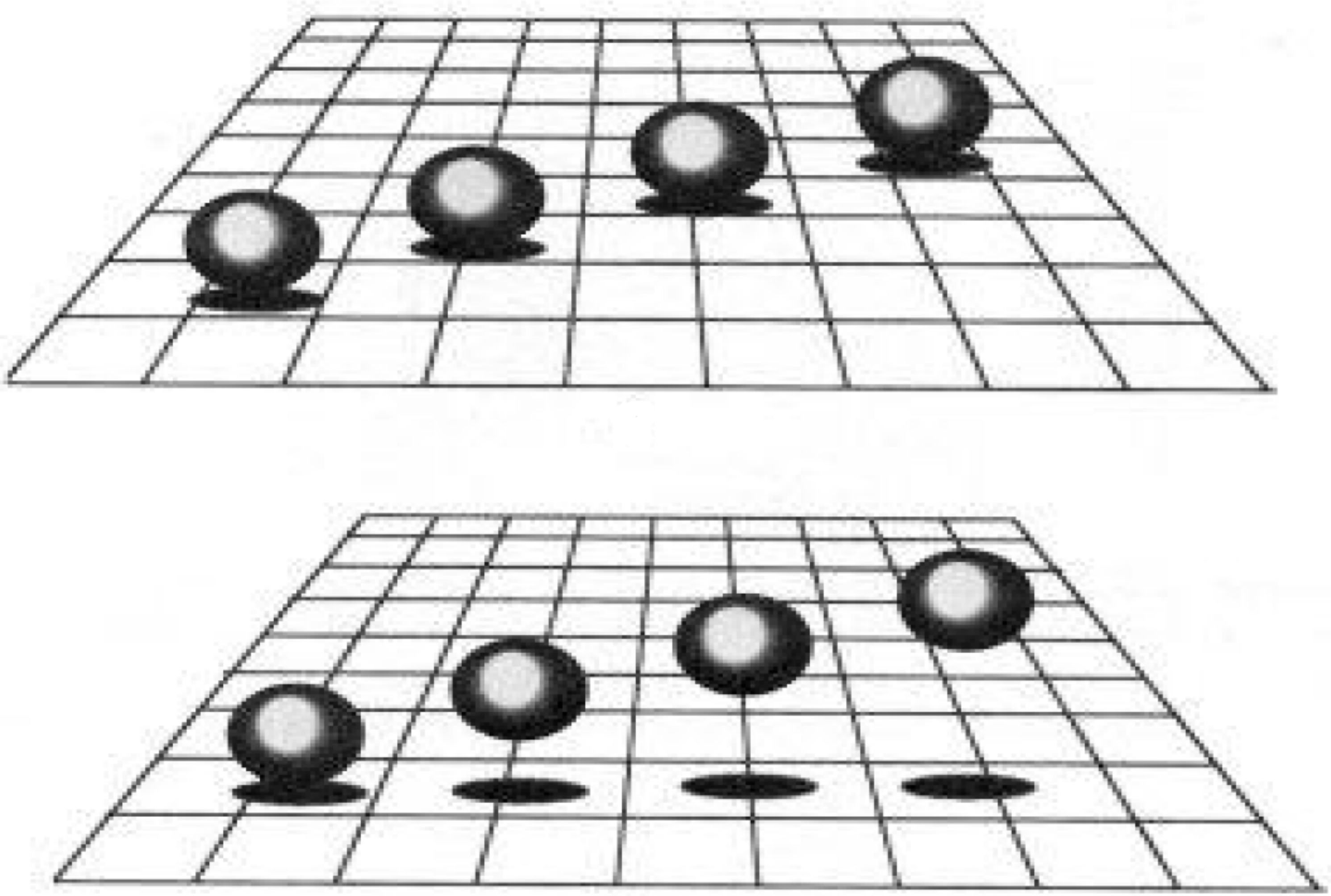


Without shadow



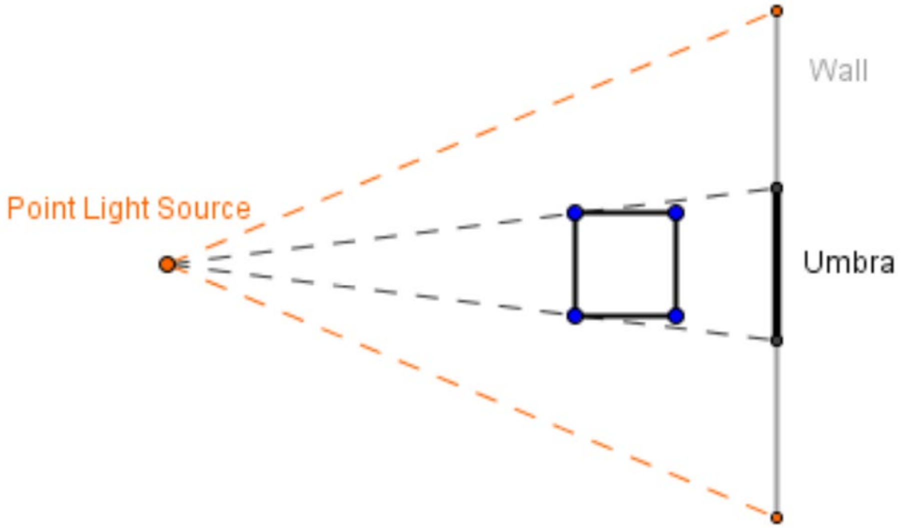
With shadow

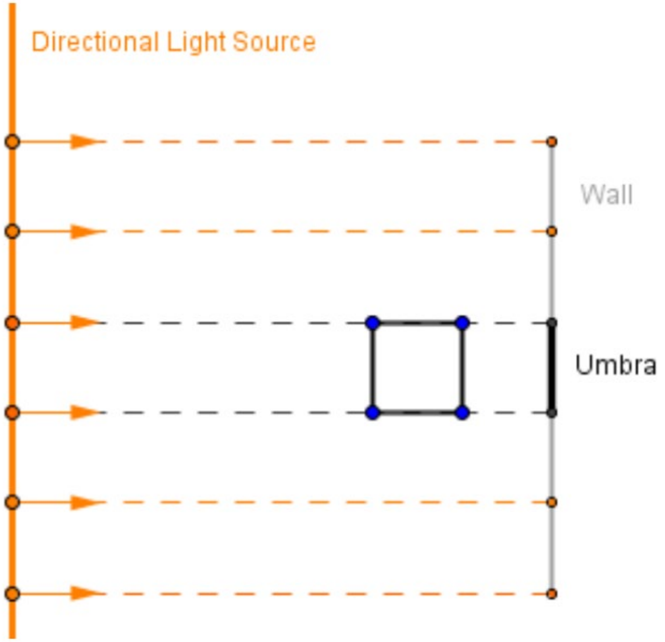
Shadows

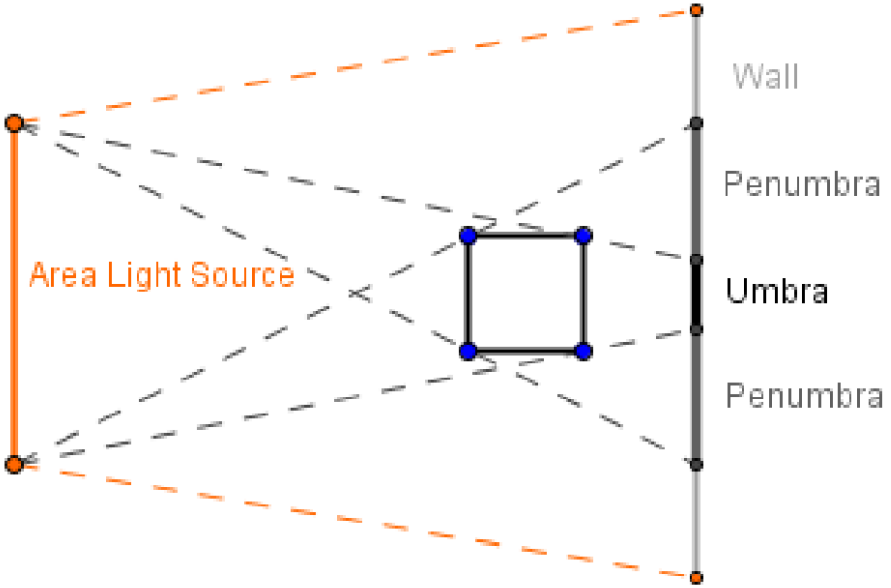


Shadow components

- Umbra (本影)
- Penumbra (半影)

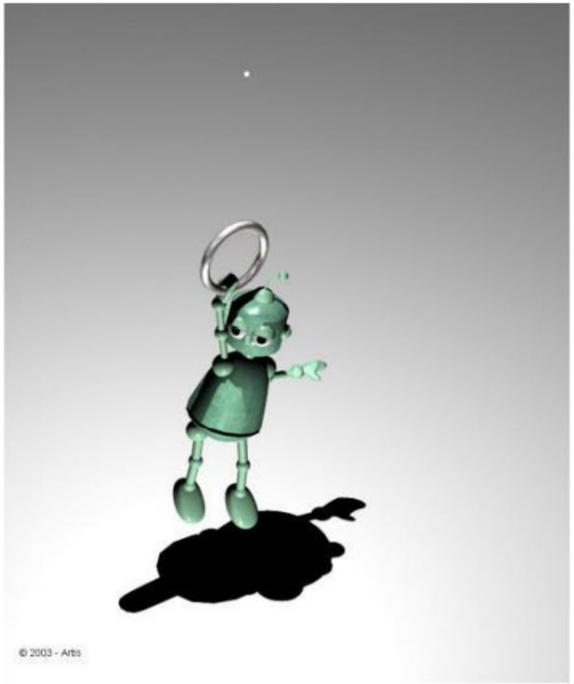


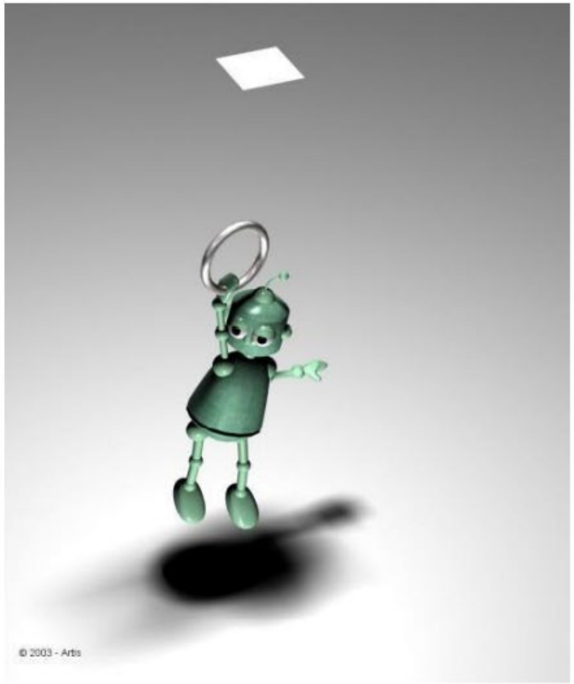




Lights and shadows

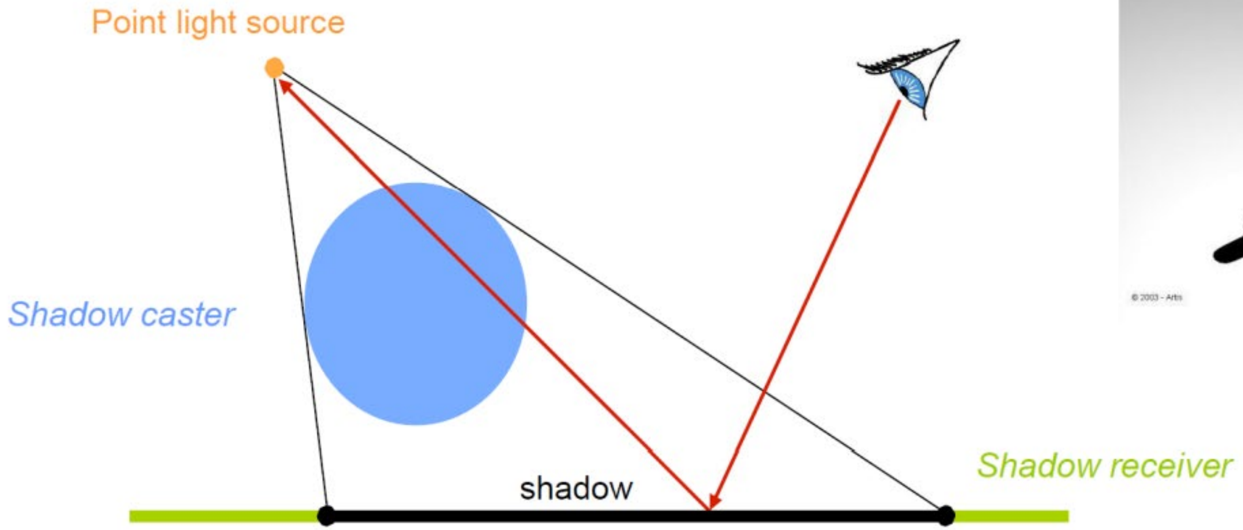
- Two basic approaches
  - Hard shadows – only point light sources
  - Soft shadows – area light sources





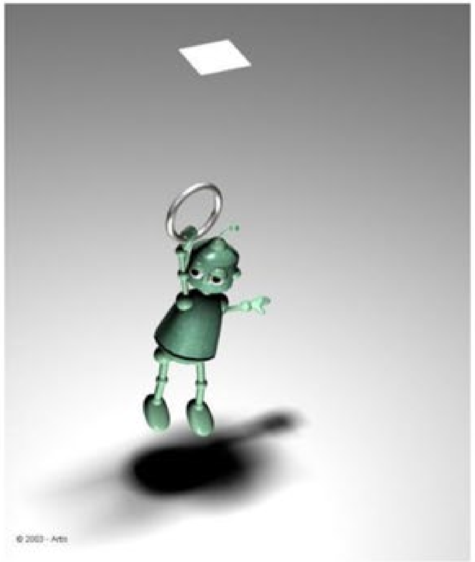
Hard shadow

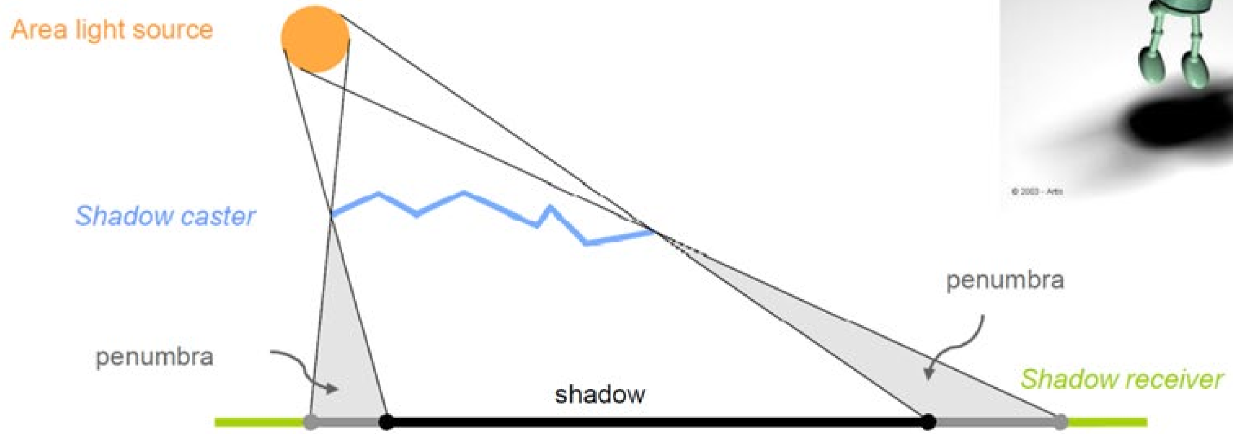
- Point light source
  - A point is in a shadow if it is not visible from the light source



Soft shadow

- Three types of surface:
  - Shadow: light source completely hidden
  - Penumbra: light source partially hidden
  - Lit: light source completely visible

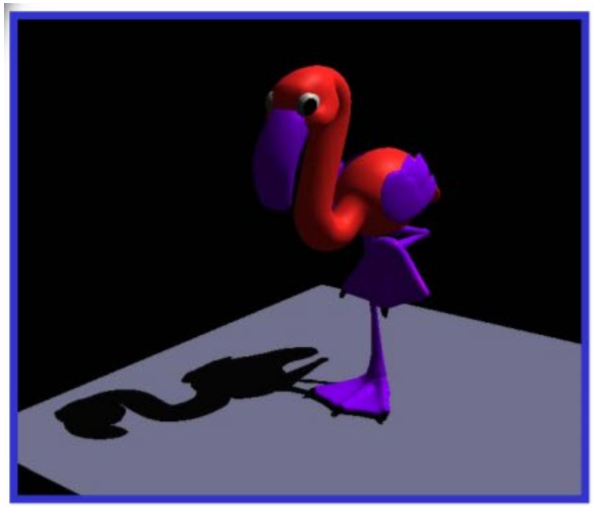




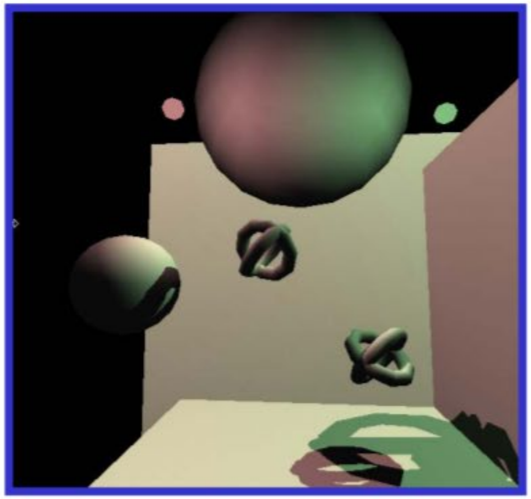
This Lecture

- Shadow
- Shadow Mapping
- Issues in Shadow Mapping

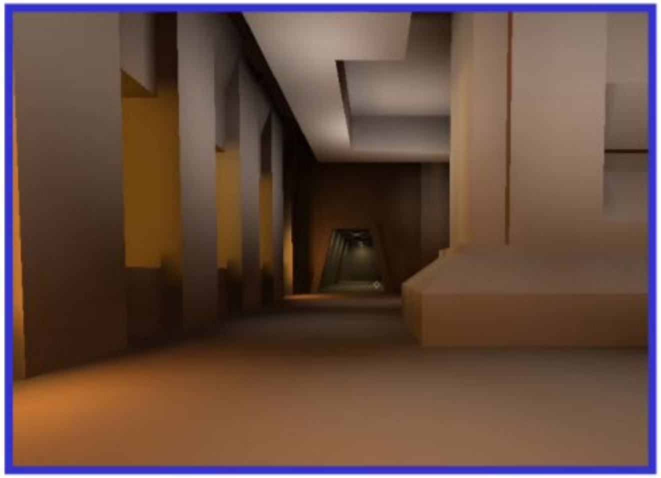
Common Real-time Shadow Techniques



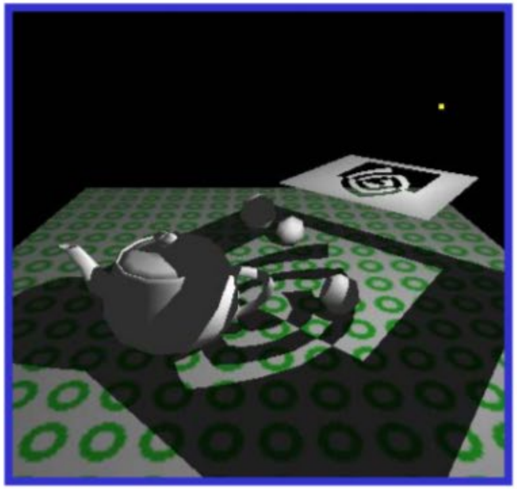
Projected planar shadows



Shadow volumes



Light maps



Hybrid approaches

Shadow Mapping

- Image-space shadow determination
  - Two pass algorithm
  - Fast on today's GPUs
  - relatively easy to implement

- Important papers
  - William Reeves, David Salesin, and Robert Cook (Pixar), “Rendering antialiased shadows with depth maps,” SIGGRAPH 87
  - Mark Segal, et. al. (SGI), “Fast Shadows and Lighting Effects Using Texture Mapping,” SIGGRAPH 92

Shadow Mapping Overview

- 1st pass: create a z-buffer from light position
  - assume light source has a “view frustum” (like a camera)
  - render scene from light source’s position
  - save depth values only (shadow map)

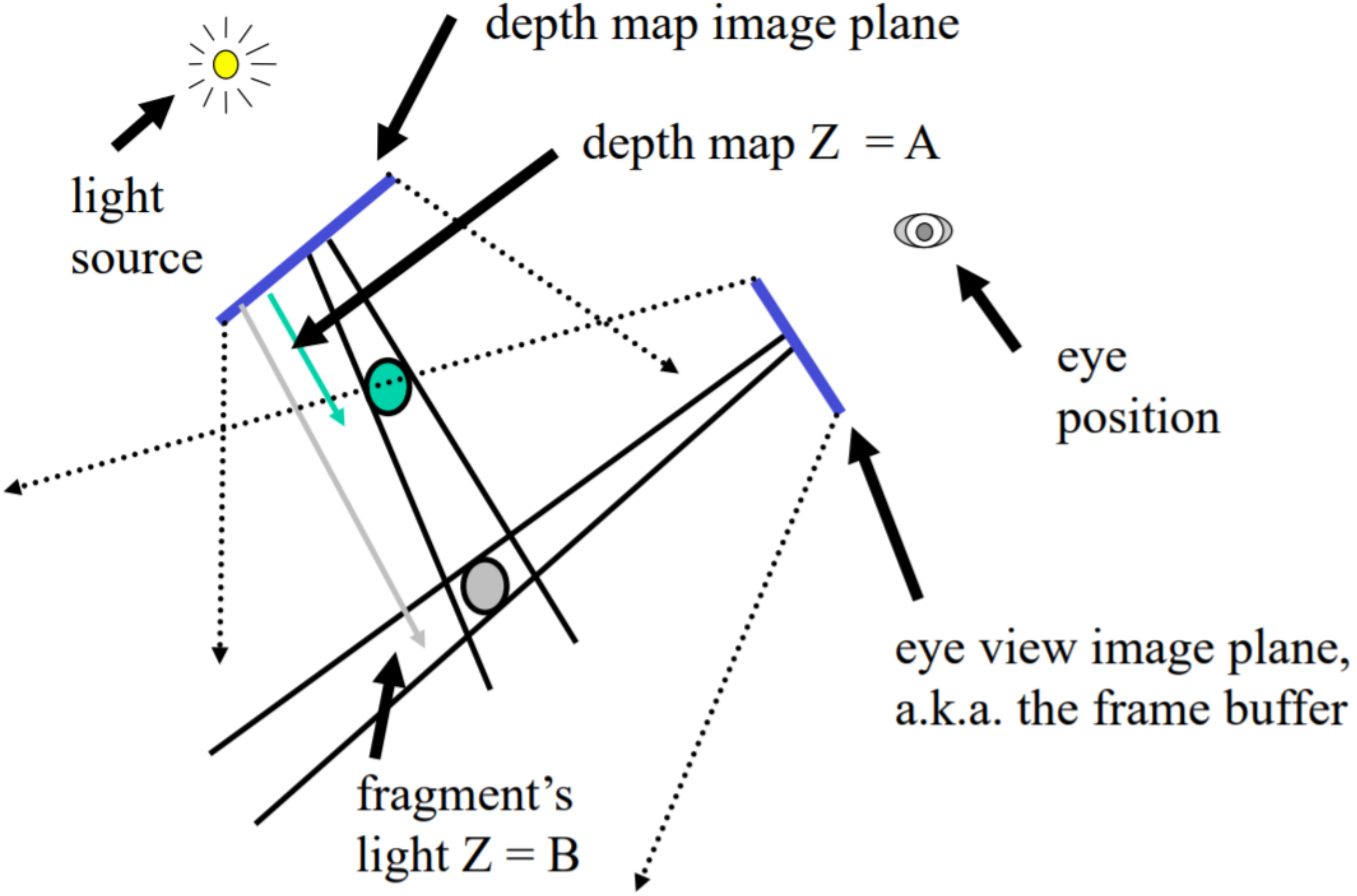
- 2st pass: do rendering from eye position
  - render scene as usual
    - inverse map to world space
    - transform vertices to light space, too
  - for each fragment, compare depth
    - $z_{fragment} > z_{shadow\_map}$ => fragment lies in shadow
    - fragment must be in light space!!!

Shadow Mapping

- The Shadow Map Comparison
  - Two values
    - A = Z value from depth map at fragment's light XY position
    - B = Z value of fragment's XYZ light position
  - If B is greater than A, then there must be something closer to the light than the fragment
    - then the fragment is shadowed
  - If A and B are approximately equal, the fragment is lit

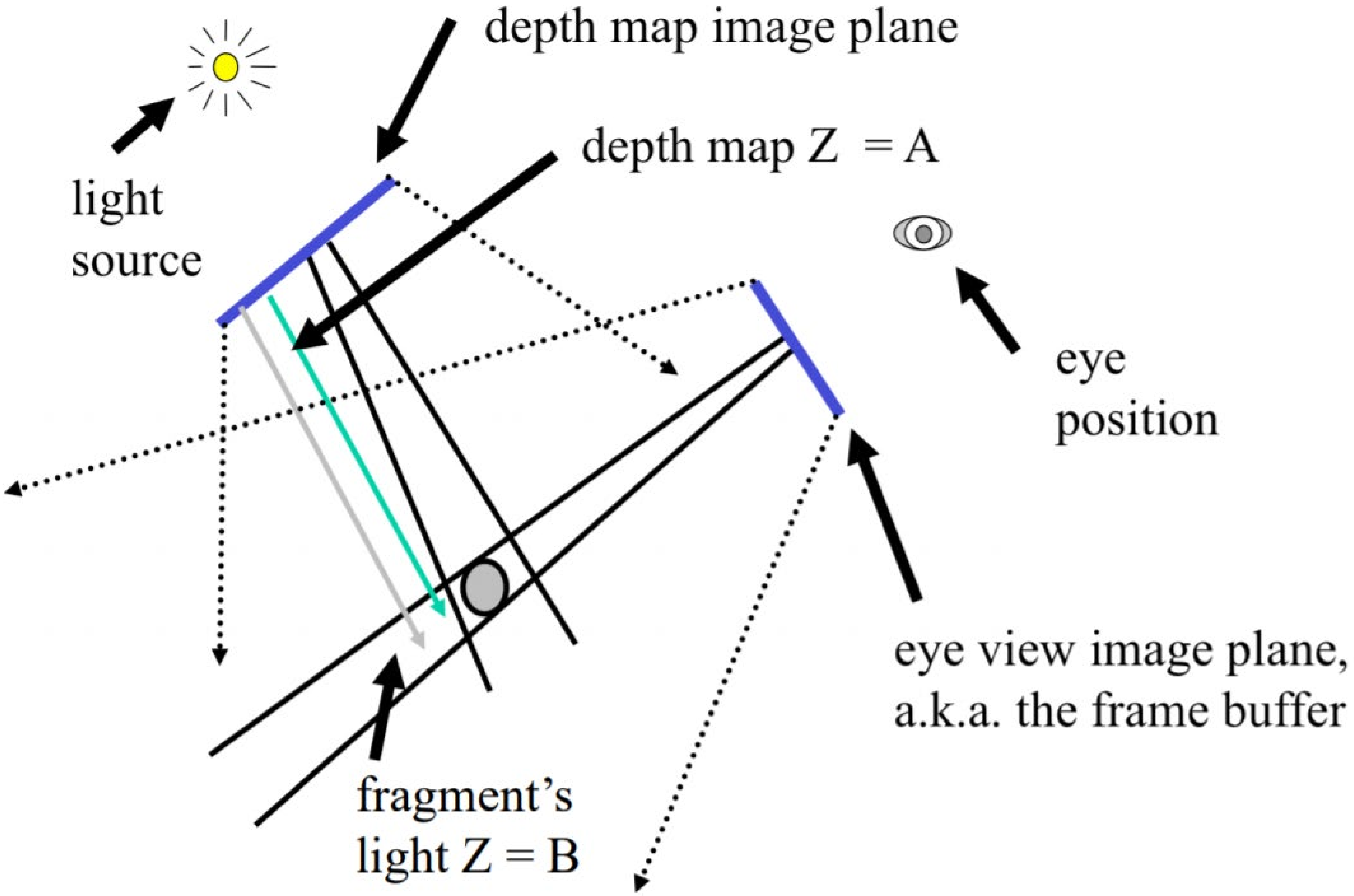
Shadow Mapping with a Picture in 2D

- The A < B shadowed fragment case

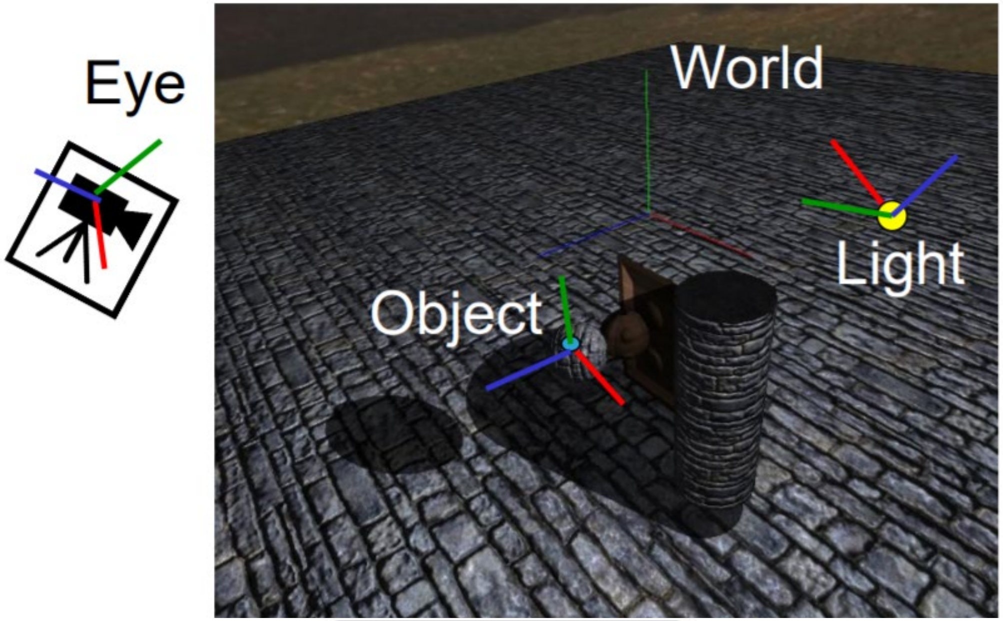


Shadow Mapping with a Picture in 2D

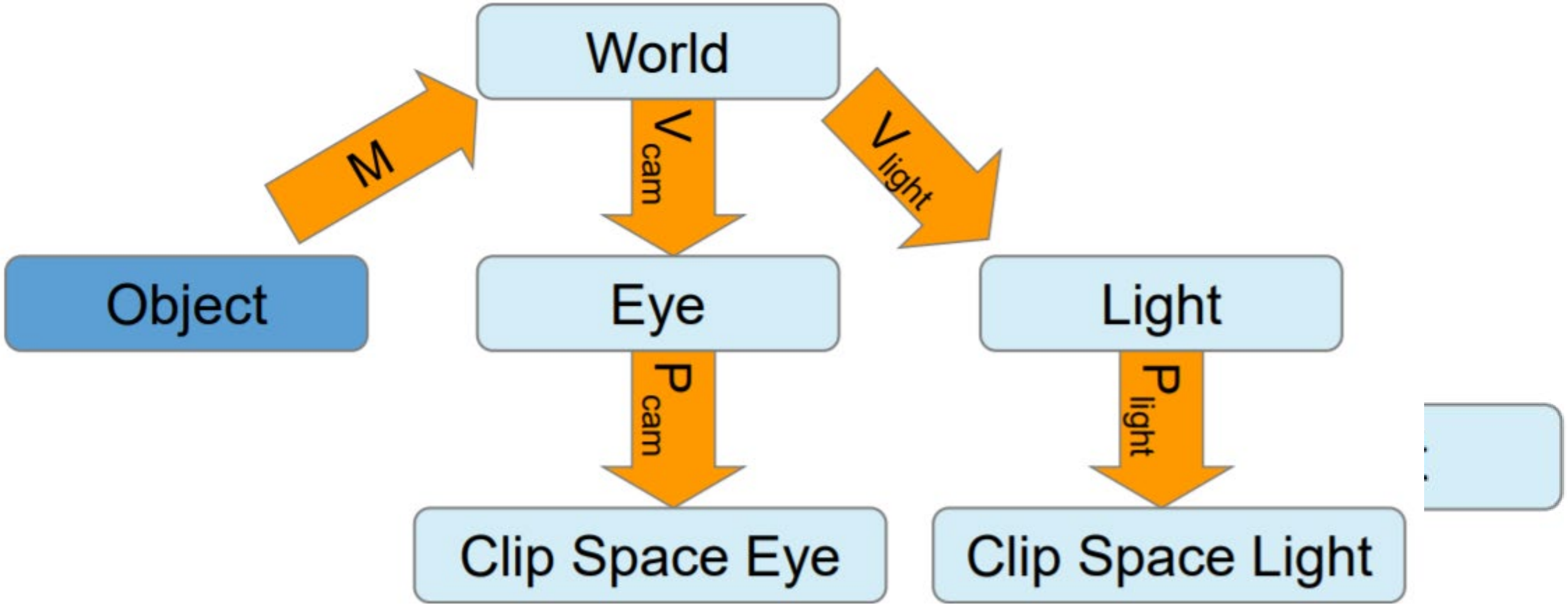
- The A ≅ B unshadowed fragment case



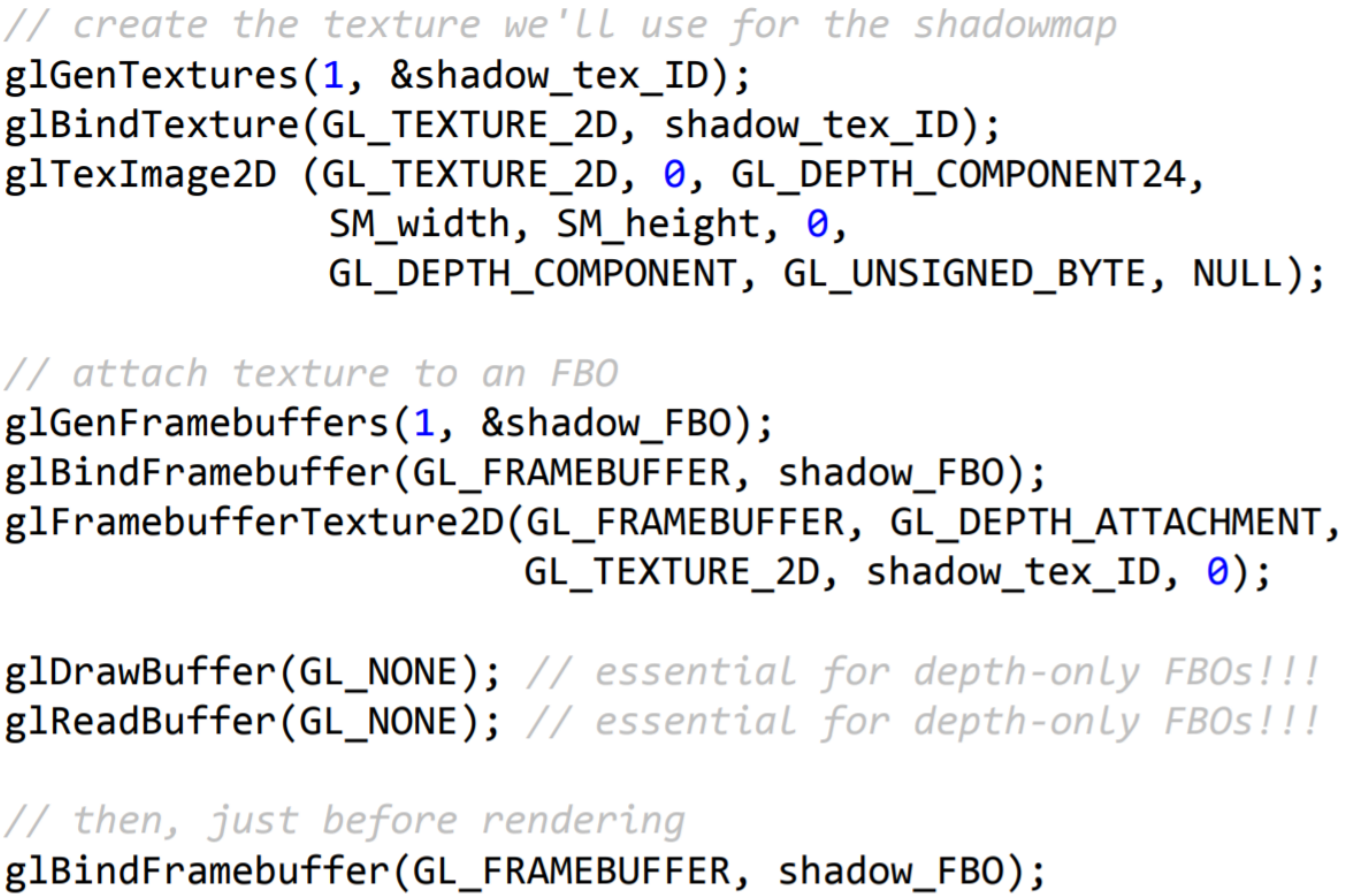
Involved Coordinate Systems



M …
V_{cam} …
V_{light} …



1st pass: Create Shadow Map



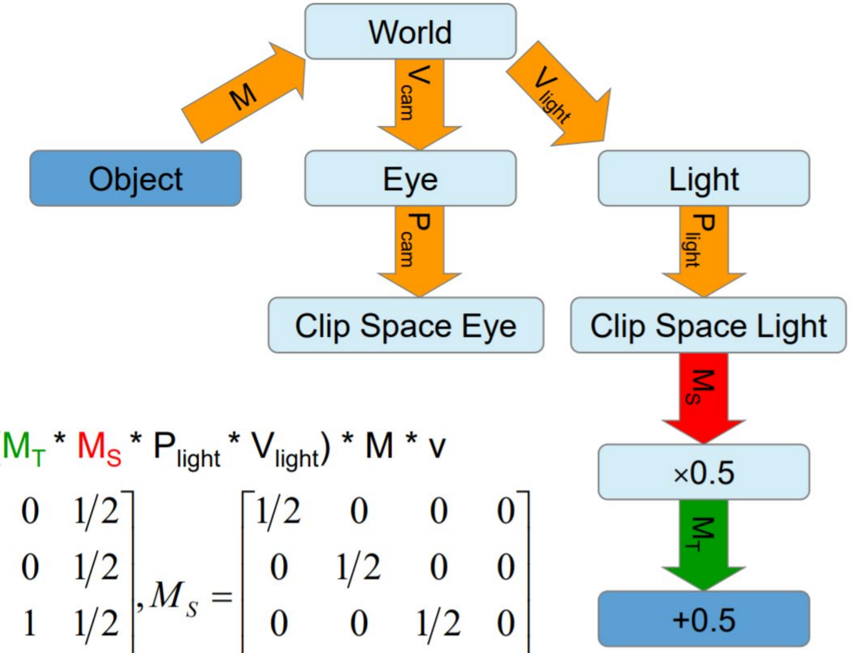
```
// create the texture we'll use for the shadowmap
glGenTextures(1, &shadow_tex_ID);
glBindTexture(GL_TEXTURE_2D, shadow_tex_ID);
glTexImage2D (GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT24,
              SM_width, SM_height, 0,
              GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);

// attach texture to an FBO
glGenFramebuffers(1, &shadow_FBO);
glBindFramebuffer(GL_FRAMEBUFFER, shadow_FBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
                       GL_TEXTURE_2D, shadow_tex_ID, 0);

glDrawBuffer(GL_NONE); // essential for depth-only FBOs!!!
glReadBuffer(GL_NONE); // essential for depth-only FBOs!!!

// then, just before rendering
glBindFramebuffer(GL_FRAMEBUFFER, shadow_FBO);
```

1st pass: Create Shadow Map

- Set view matrix of Light
- Set projection matrix of Light
- Turn off all effects when rendering to the shadow map
    - No textures, lighting, etc.

2st pass: Render from Eye’s POV

- Transform vertices to eye space and project as usual
- transform vertices to projected light space
  - Calculate texture coords (shadow map)



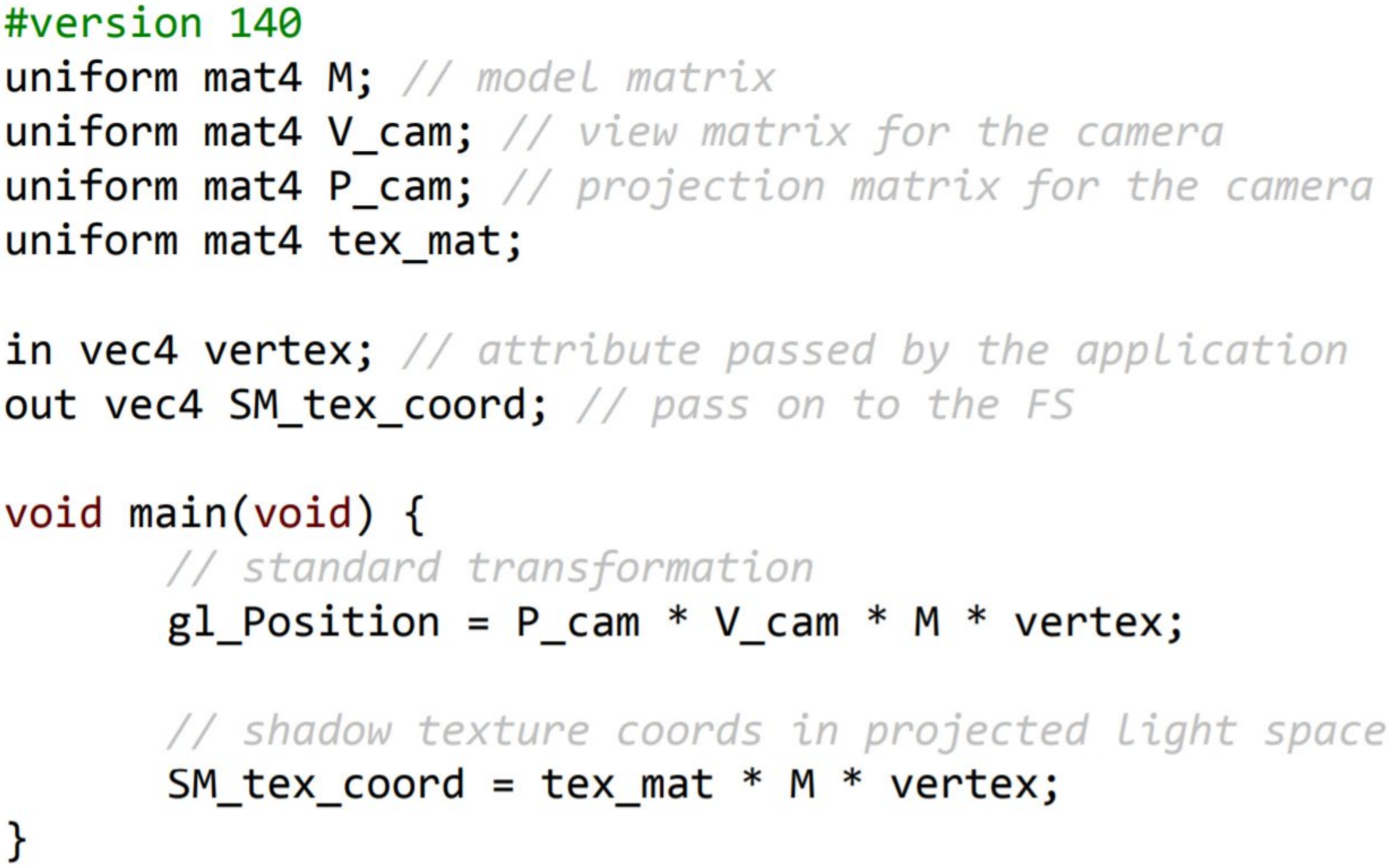
$$SM_{texcoord} = (M_T * M_S * P_{light} * V_{light}) * M * v$$

$$M_T = \begin{bmatrix} 1 & 0 & 0 & 1/2 \\ 0 & 1 & 0 & 1/2 \\ 0 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_S = \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Compare depth => determine whether in shadow
  - Shading

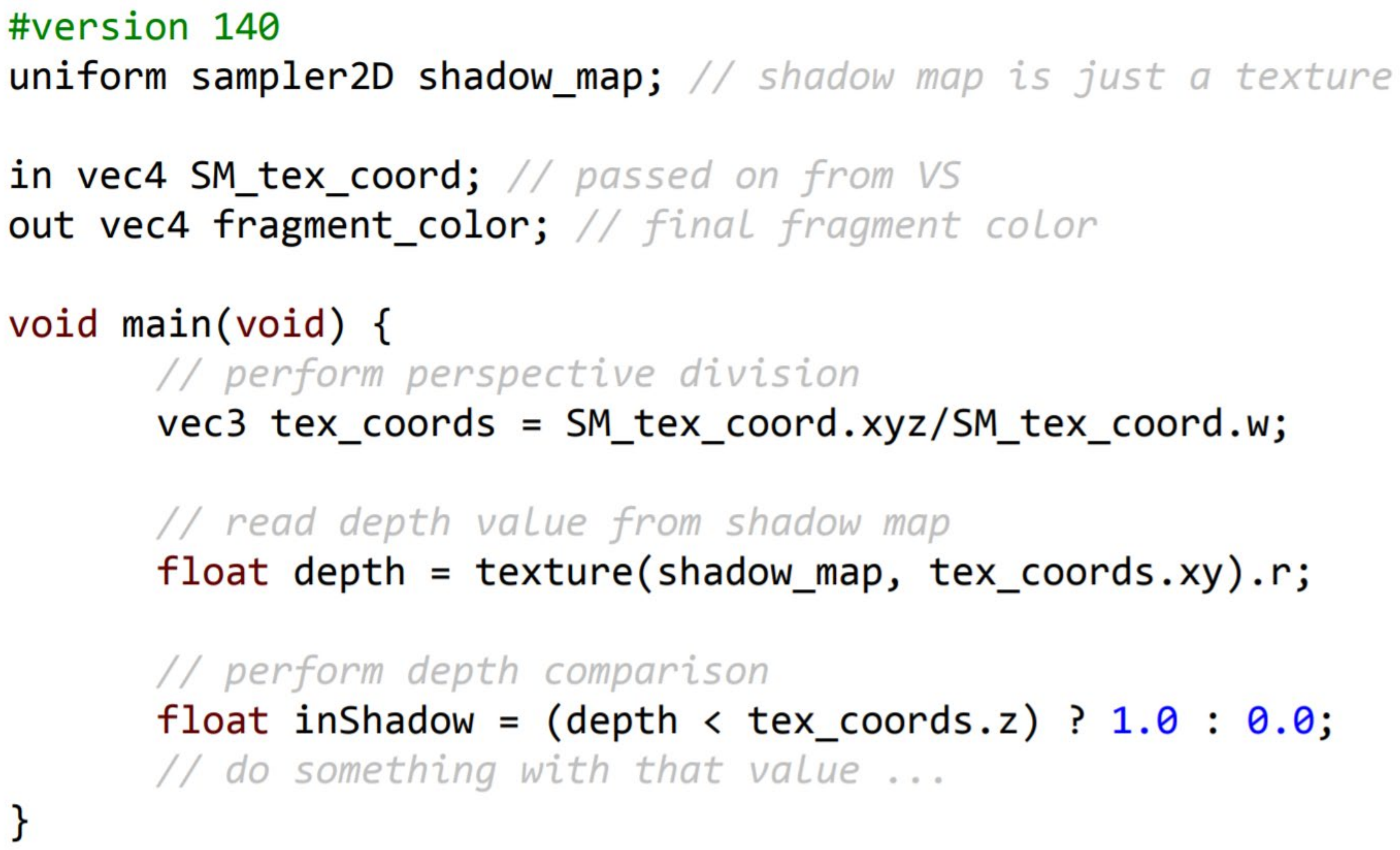
Shadow Mapping – Vertex Shader

- tex_mat = $M_T * M_S * P_{light} * V_{light}$



```
#version 140
uniform mat4 M; // model matrix
uniform mat4 V_cam; // view matrix for the camera
uniform mat4 P_cam; // projection matrix for the camera
uniform mat4 tex_mat;

in vec4 vertex; // attribute passed by the application
out vec4 SM_tex_coord; // pass on to the FS

void main(void) {
      // standard transformation
      gl_Position = P_cam * V_cam * M * vertex;

      // shadow texture coords in projected light space
      SM_tex_coord = tex_mat * M * vertex;
}
```

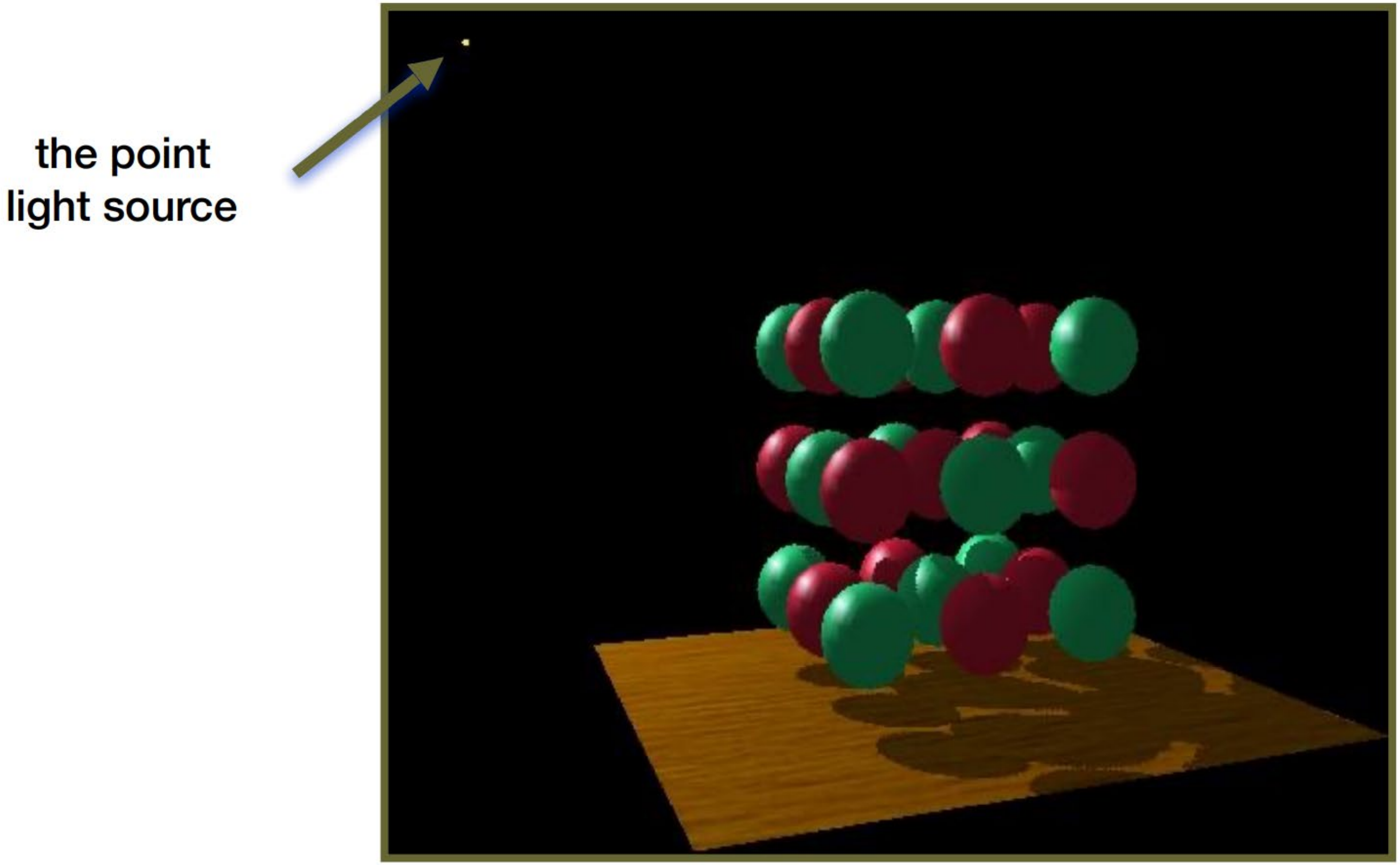
Shadow Mapping – Fragement Shader



```
#version 140
uniform sampler2D shadow_map; // shadow map is just a texture

in vec4 SM_tex_coord; // passed on from VS
out vec4 fragment_color; // final fragment color

void main(void) {
        // perform perspective division
        vec3 tex_coords = SM_tex_coord.xyz/SM_tex_coord.w;

        // read depth value from shadow map
        float depth = texture(shadow_map, tex_coords.xy).r;

        // perform depth comparison
        float inShadow = (depth < tex_coords.z) ? 1.0 : 0.0;
        // do something with that value ...
}
```

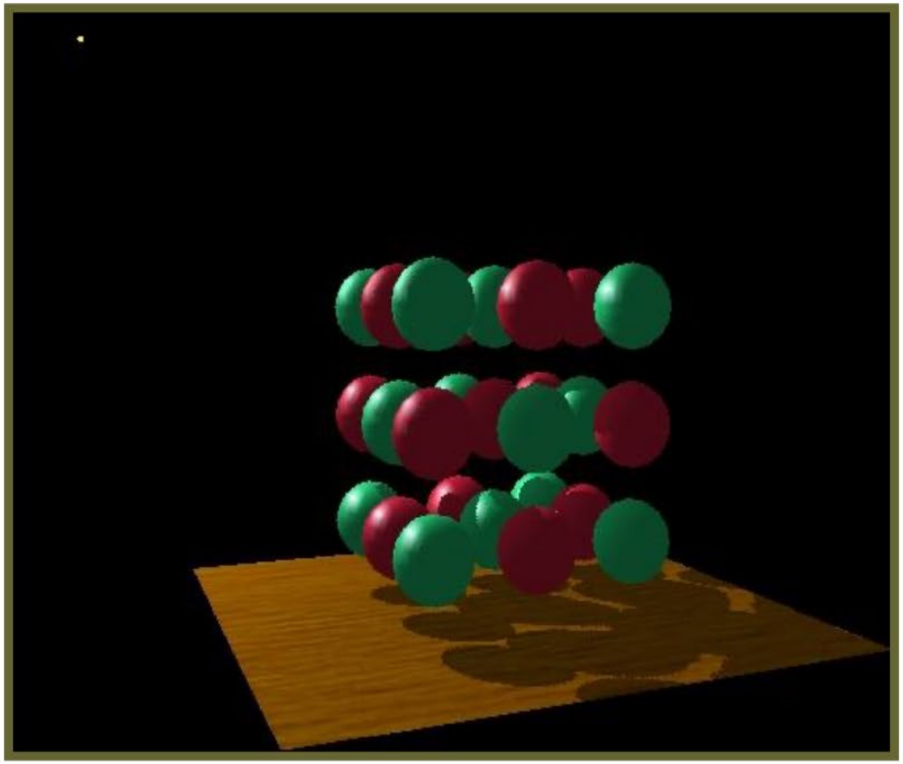
Visualizing Shadow Mapping

- A fairly complex scene with shadows

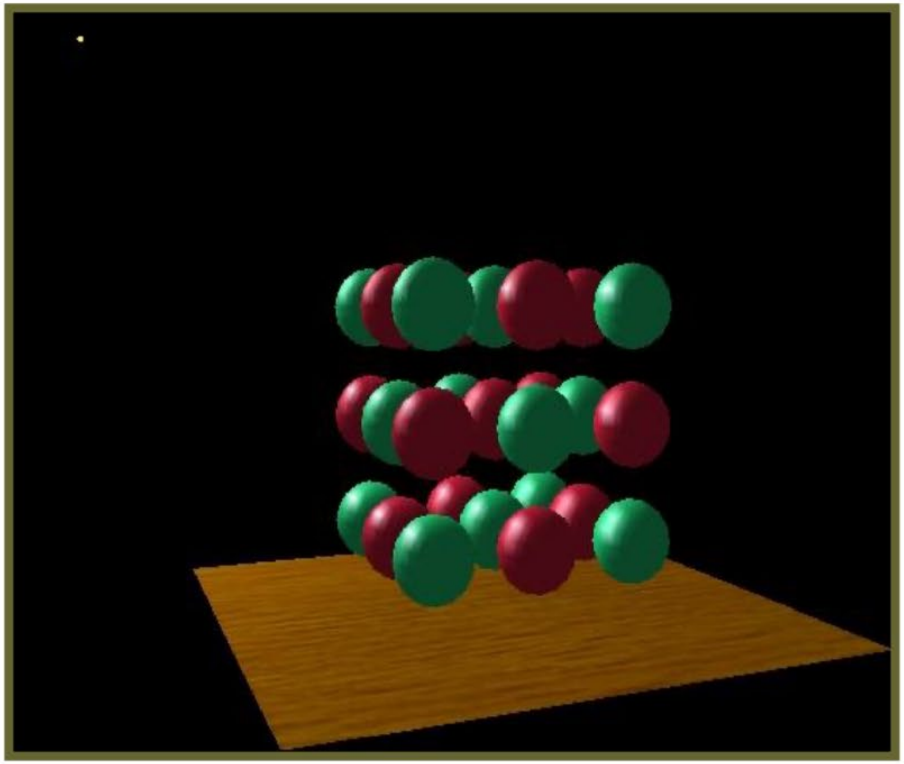


Visualizing Shadow Mapping

- Compare with and without shadows



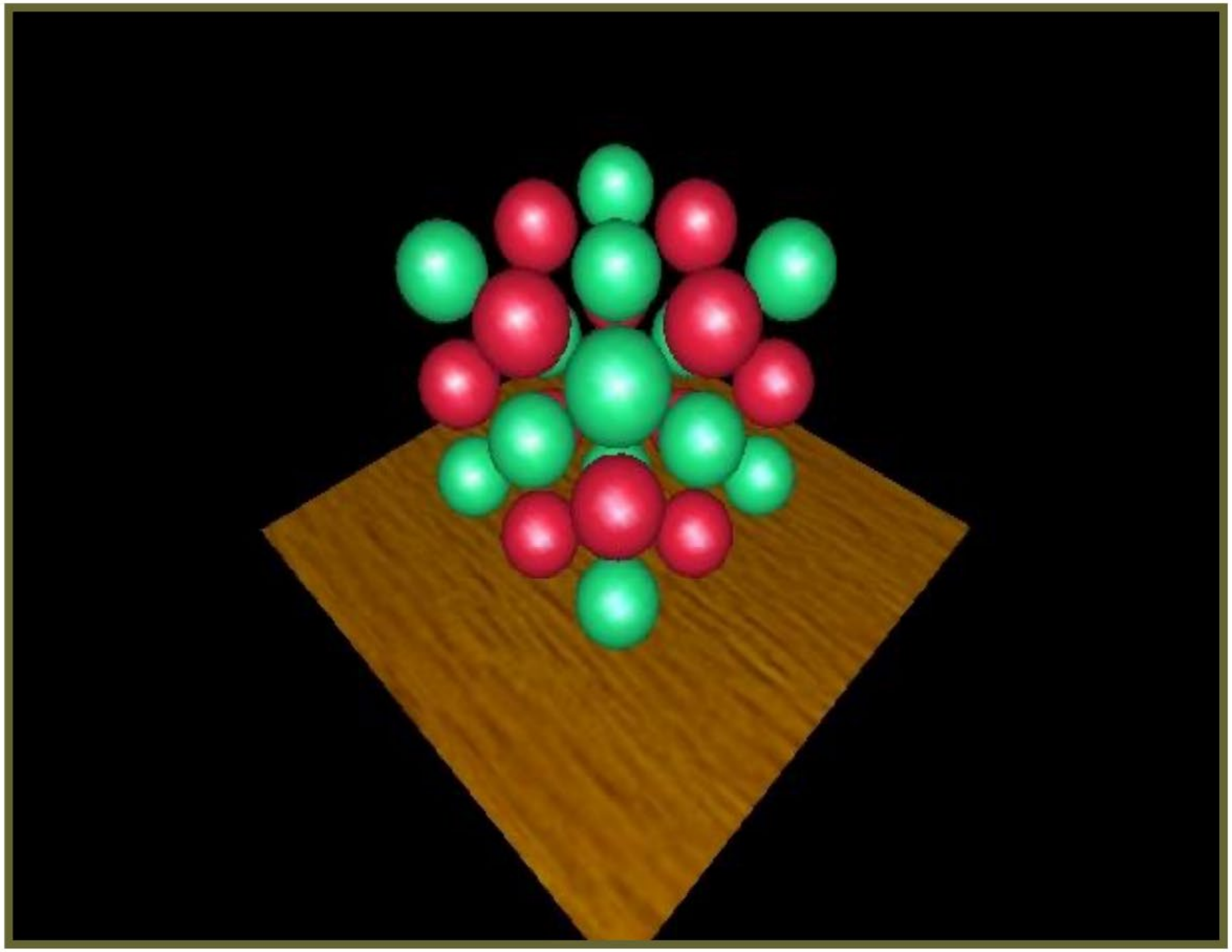
with shadows

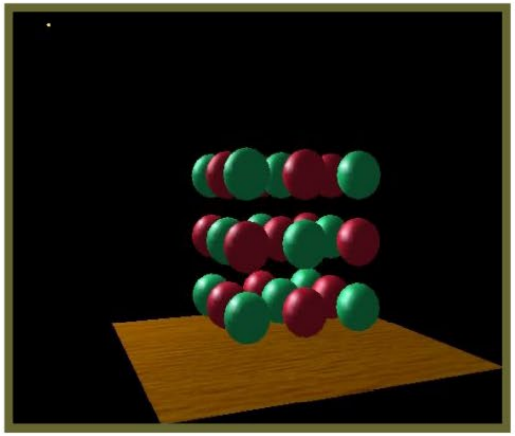


without shadows

Visualizing Shadow Mapping

- The scene from the light’s point-of-view

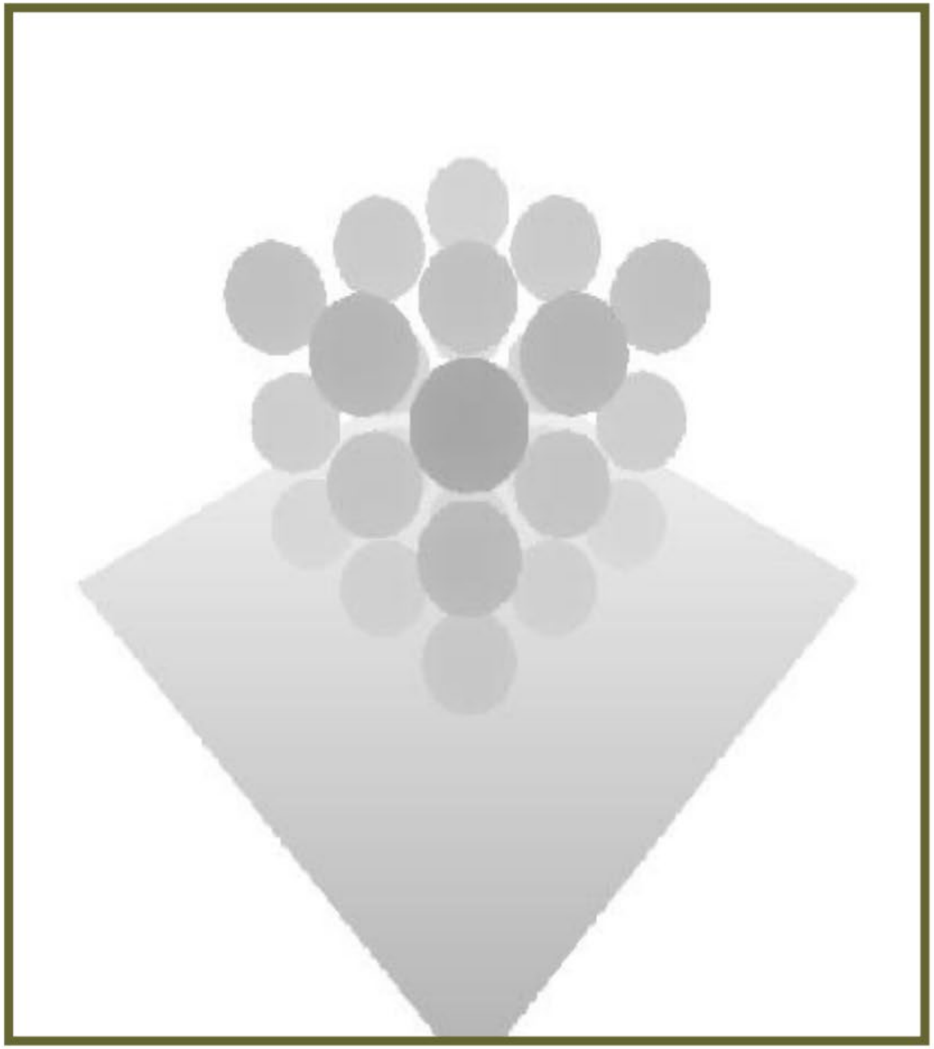


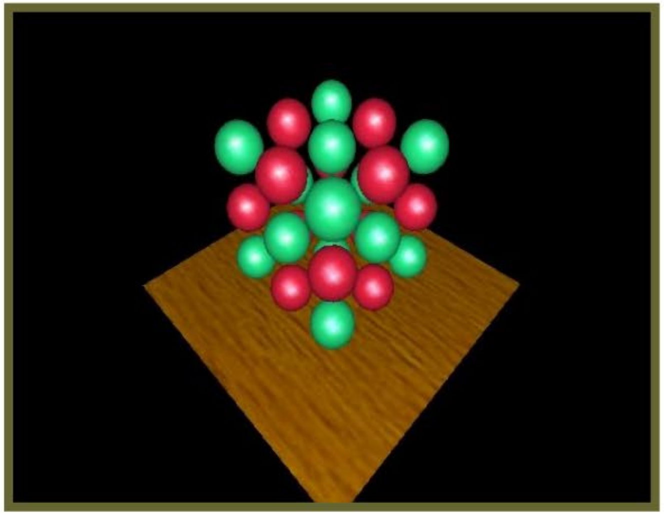


FYI: from the eye’s point-of-view again

Visualizing Shadow Mapping

- The depth buffer from the light's point-of-view

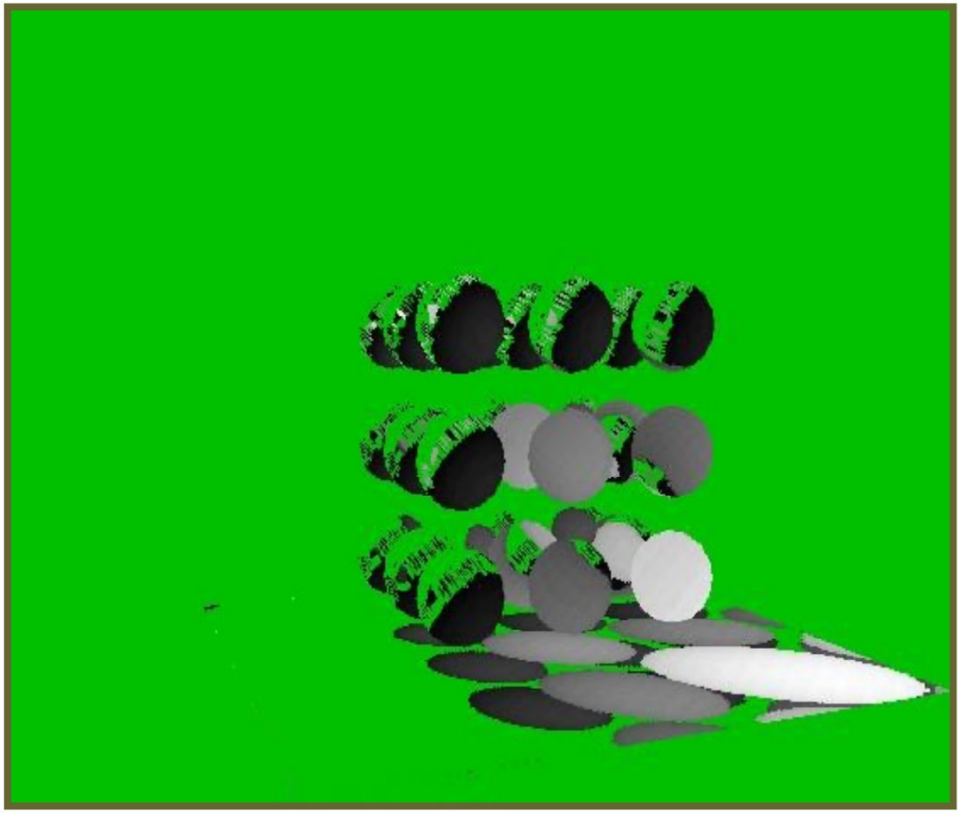




FYI: from the light's point-of-view again

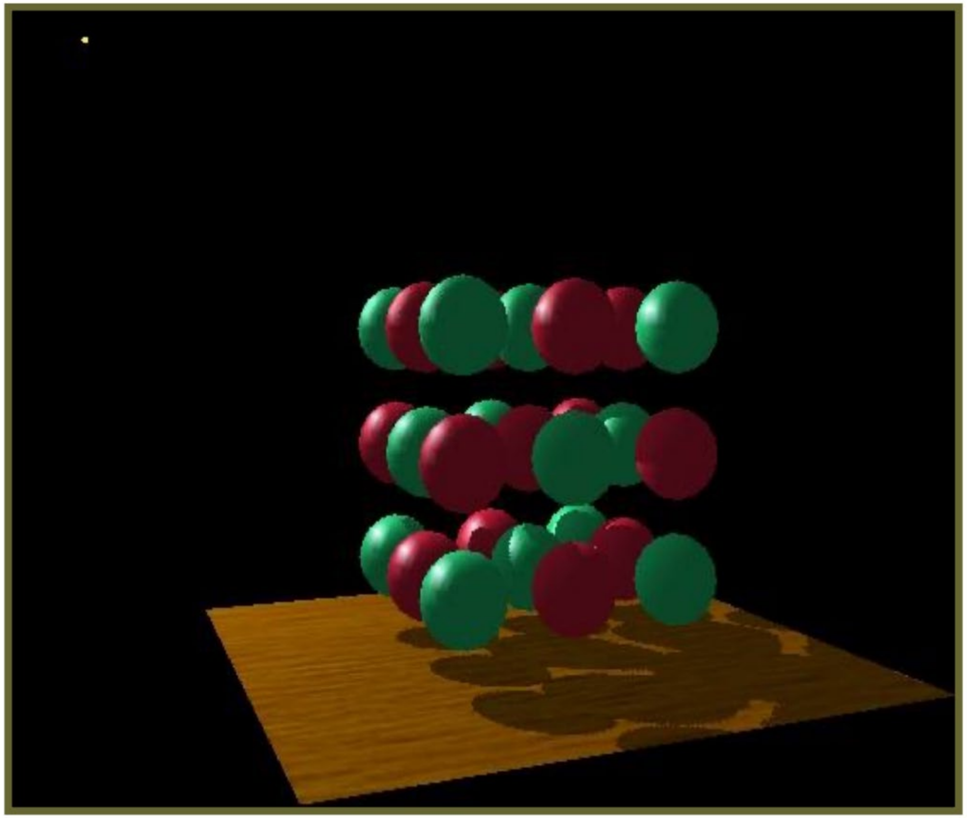
Visualizing Shadow Mapping

- Comparing Dist(light, shading point) with shadow map
  - Green is where the distance(light, shading point) ≈ depth on the shadow map
  - Non-green is where shadows should be



Visualizing Shadow Mapping

- Scene with shadows
  - Notice how specular highlights never appear in shadows
  - Notice how curved surfaces cast shadows on each other



This Lecture

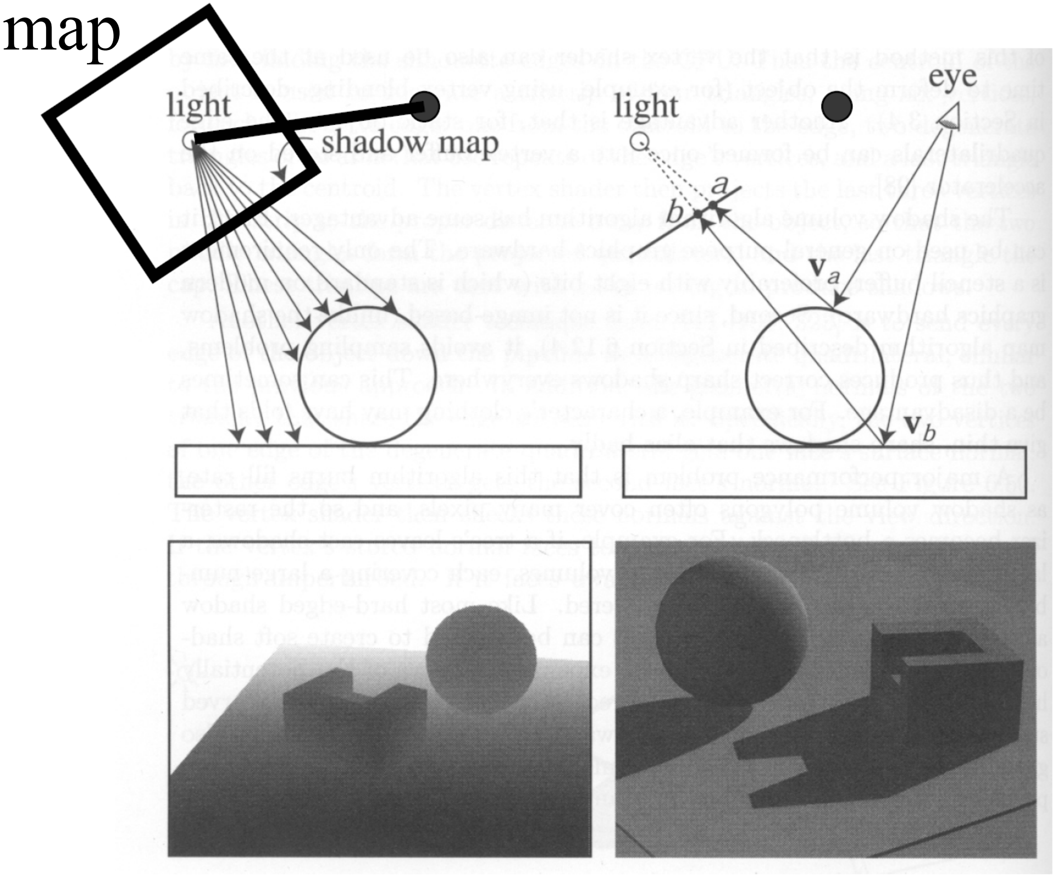
- Shadow
- Shadow Mapping
- Issues in Shadow Mapping

Issues in Shadow Mapping

- Field of View
- Self occlusion – Resolution in Z coordinates
- Aliasing – Resolution in XY coordinates.

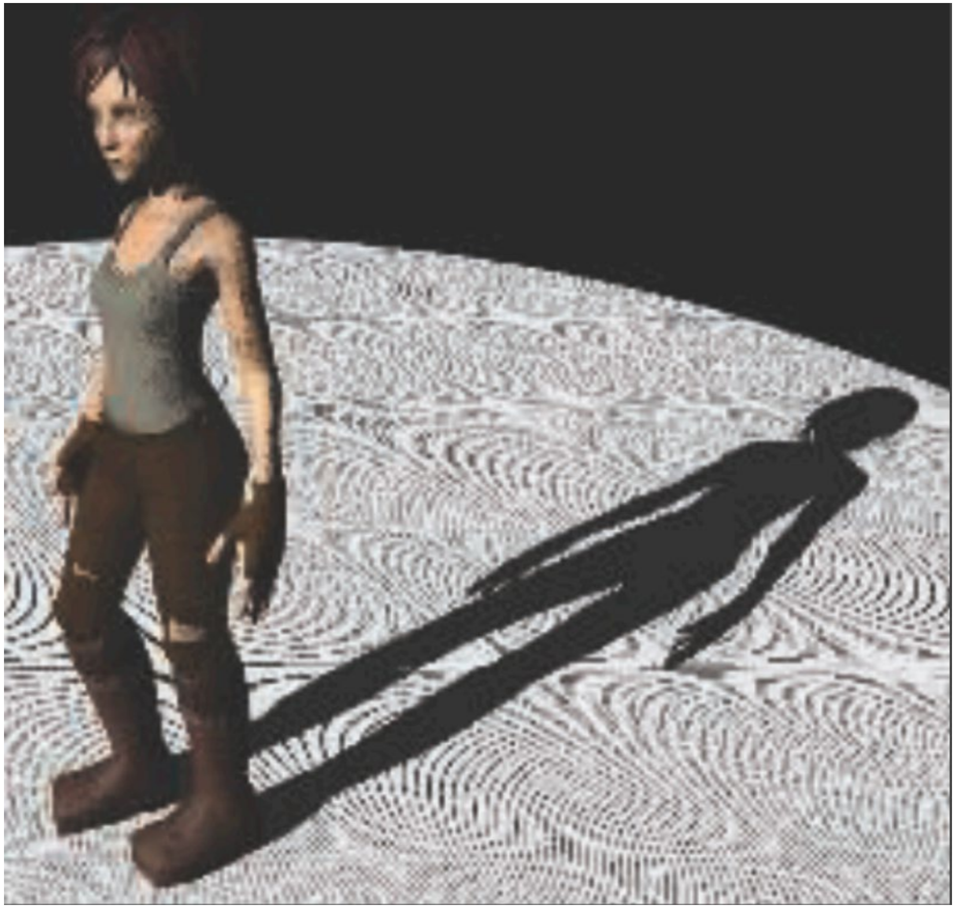
Field of View

- What if point to shadow is outside field of view of shadow map?
  - Use cubical shadow map
  - Use only spot lights!



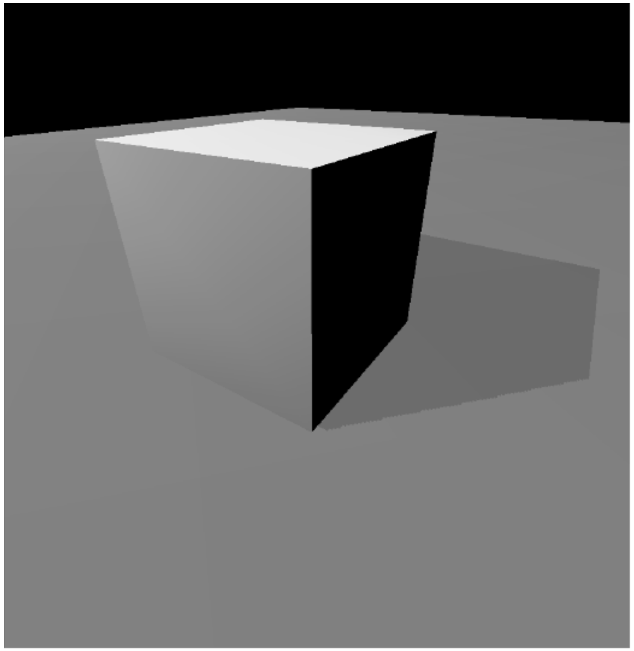
Self occlusion

- comparison of floating point depth
- shadow map resolution (the sampling)

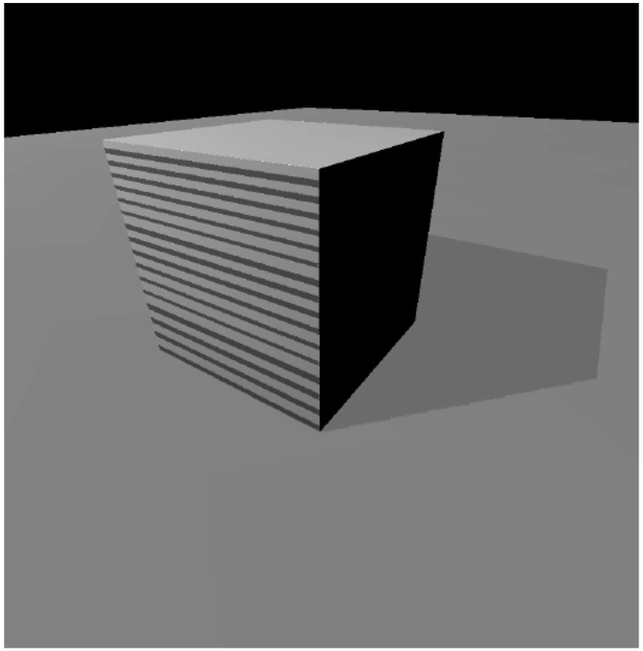


Self occlusion

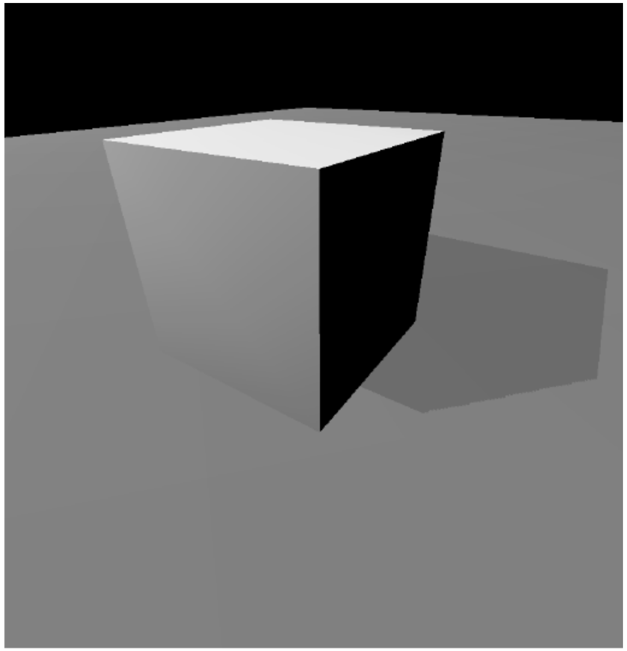
- Adding a (variable) bias to reduce self occlusion
  - Choosing a good bias value can be very tricky
  - Using the midpoint between first and second depths in Shadow Map



Correct image



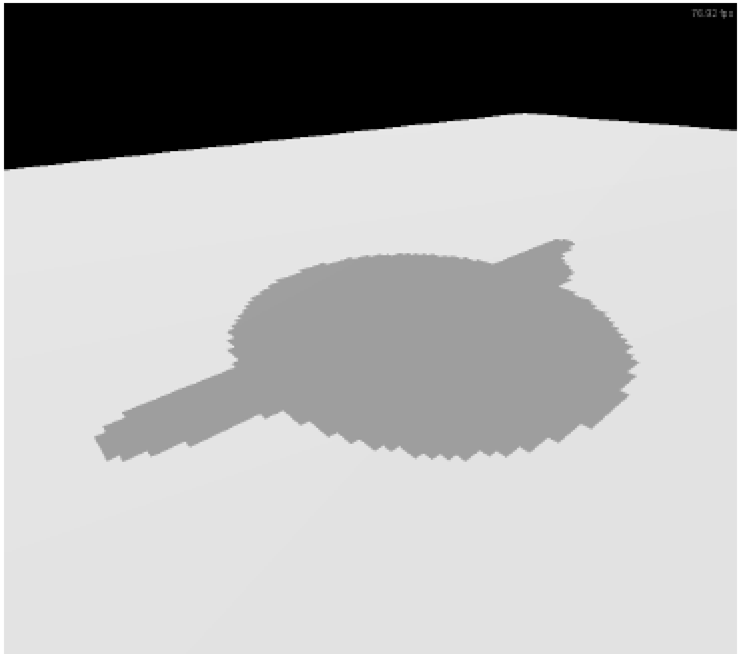
Not enough bias

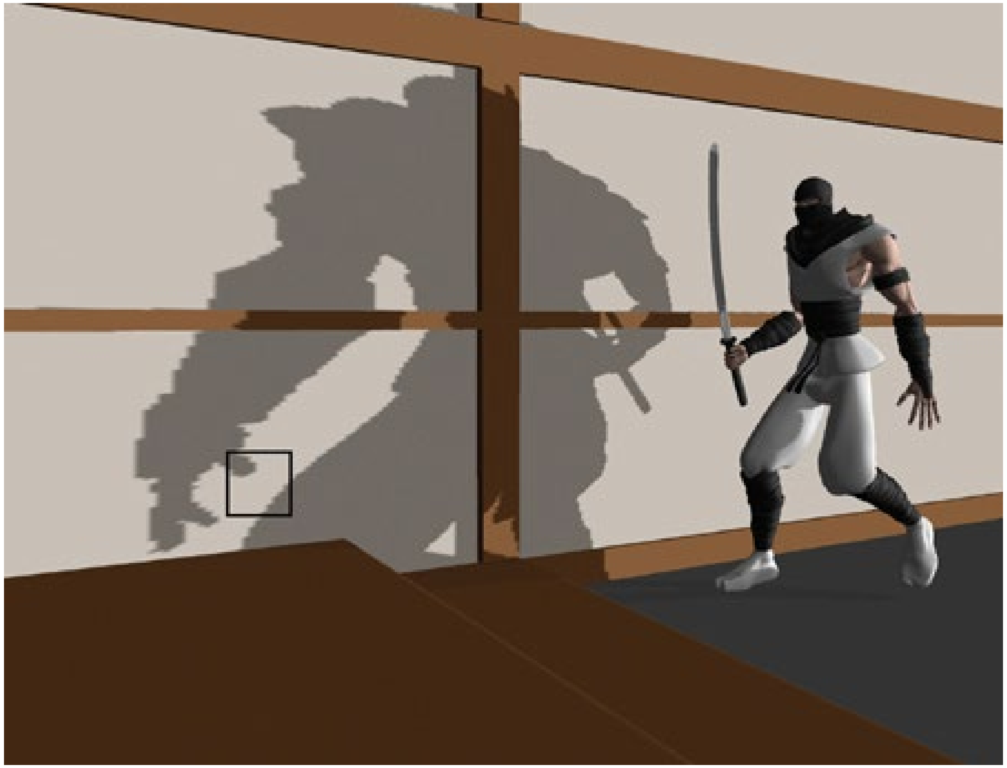


Way too much bias

Aliasing

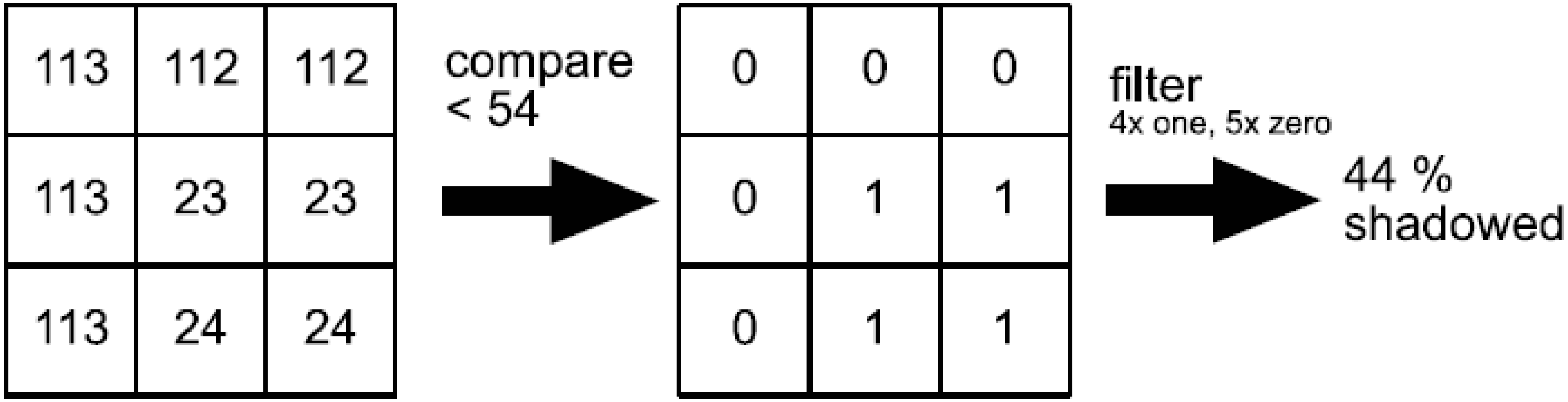
- Under-sampling of the shadow map
  - aliasing on the border of the shadow
  - one pixel is white, the next is black, without a smooth transition inbetween





Shadow Map Filtering

- Percentage closer filtering
  - Filtering depth values makes no sense
  - Perform shadow test before filtering



113	112	112
113	23	23
113	24	24

compare
< 54

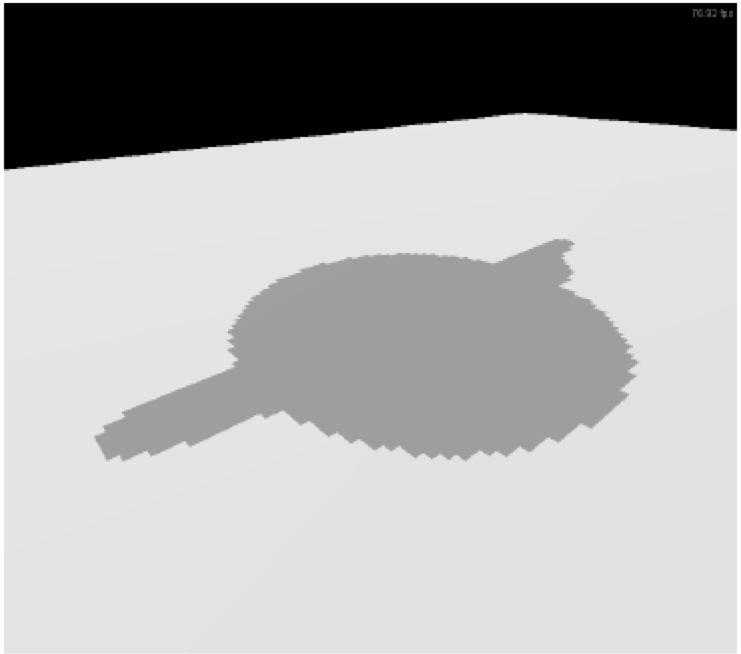
0	0	0
0	1	1
0	1	1

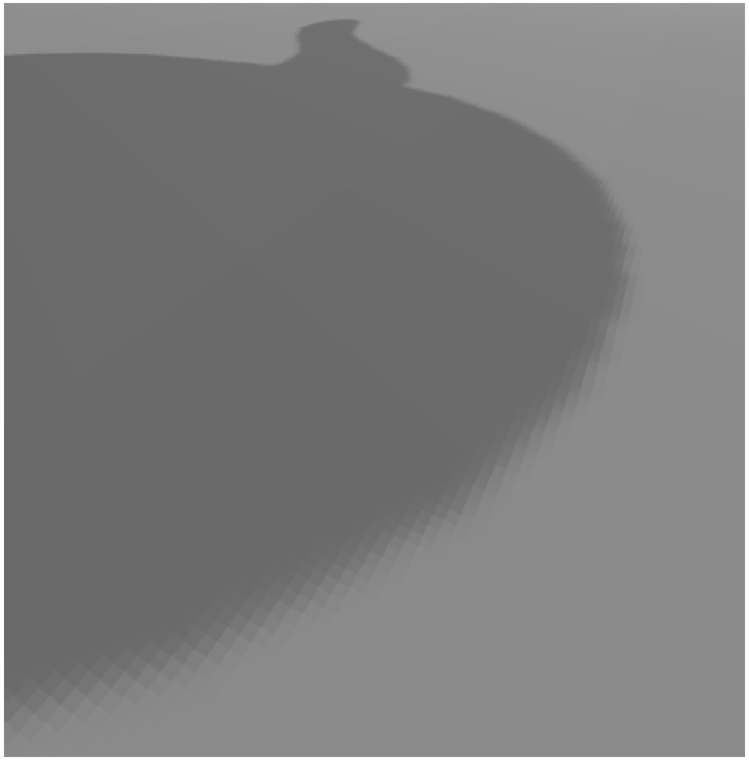
filter
4x one, 5x zero

44 %
shadowed

Percentage Closer Filtering

- Provides anti-aliasing at shadows’ edges
  - 5x5 samples
  - Using a bigger filter produces fake soft shadows
  - Not for soft shadows (PCSS, Percentage Closer Soft Shadow)



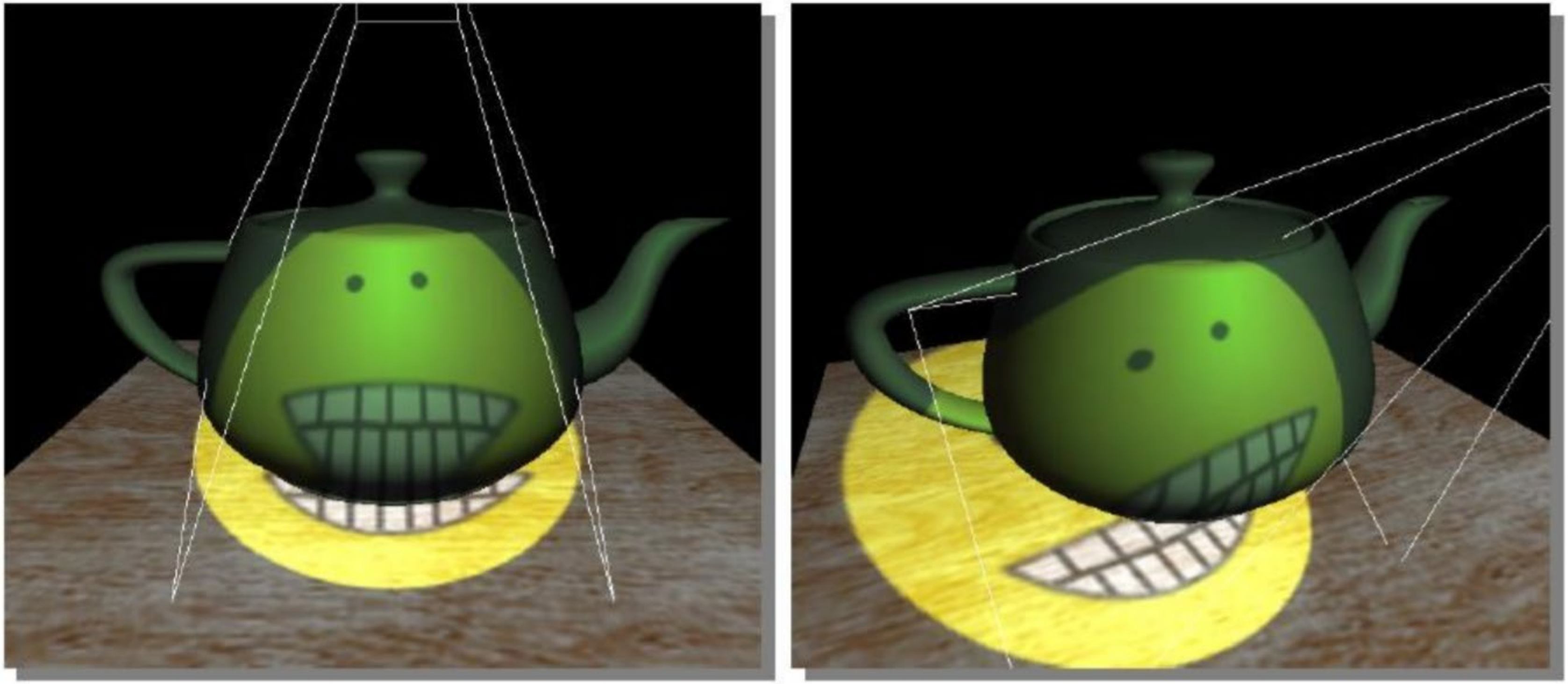


Shadow Map

- Advantages
  - Very flexible: can handle arbitrary shadow casters
  - Maps well to hardware
  - It's fast

- Disadvantages
  - Aliasing problems (image based method)
  - Quality depends on the resolution of the shadow map and the numerical precision of the Z-buffer

Projective Texturing - Example



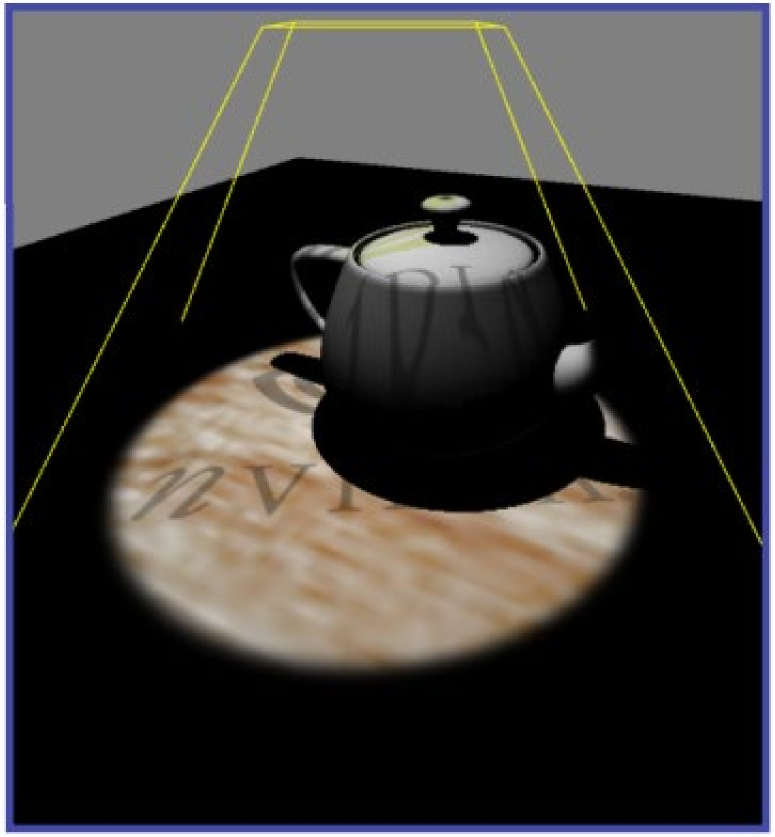
http://developer.nvidia.com/object/Projective_Texture_Mapping.html

Texture matrix

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathit{Light} \\ \mathit{Frustrum} \\ (\mathit{projection}) \\ \mathit{Matrix} \end{bmatrix} \begin{bmatrix} \mathit{Light} \\ \mathit{View} \\ (\mathit{lookat}) \\ \mathit{Matrix} \end{bmatrix} \begin{bmatrix} \mathit{Modeling} \\ \mathit{Matrix} \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix}$$

Projective Texturing

- Use a spotlight-style projected texture to give shadow maps a spotlight falloff



谢谢

北京航空航天大学人工智能研究院

INSTITUTE OF ARTIFICIAL INTELLIGENCE, BEIHANG UNIVERSITY

北京航空航天大學

人工智能研究院