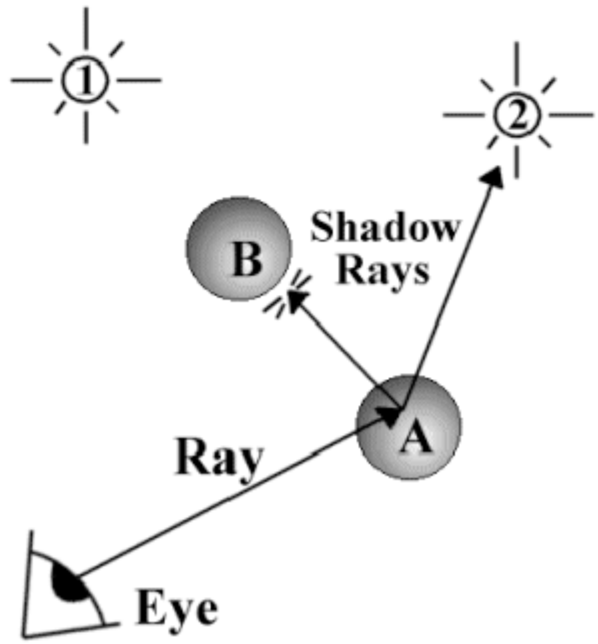
Shadows



Shadows

- Shadows is important in scenes, consolidating spatial relationships
- "Geometric shadows": the shape of an area in shadow
- Early days, just pasted into the scene as textures to fake the shadow (can you think of any other places of such usage? ☺)

Type of Shadows

- Sharp-edged or soft-edged?
- Umbra and penumbra
  - Umbra: the area completely cut off from the light source
  - Penumbra: receives some light from the light source (penumbra surrounds umbra)
- Depending on the types of light sources, may or may not get penumbra
  - Point source
  - Area source

A Simple Shadow on A Ground Plane (Blinn'88)

- Throwing shadows onto a flat plane
- Only works with scenes where objects don't cast shadows on each other
- Assuming single light source at infinite distance – parallel light rays $L(x_l, y_l, z_l)$
- Point on the object $P(x_p, y_p, z_p)$
- Shadow at $S(x_{sw}, y_{sw}, 0)$

Blinn's Algorithm

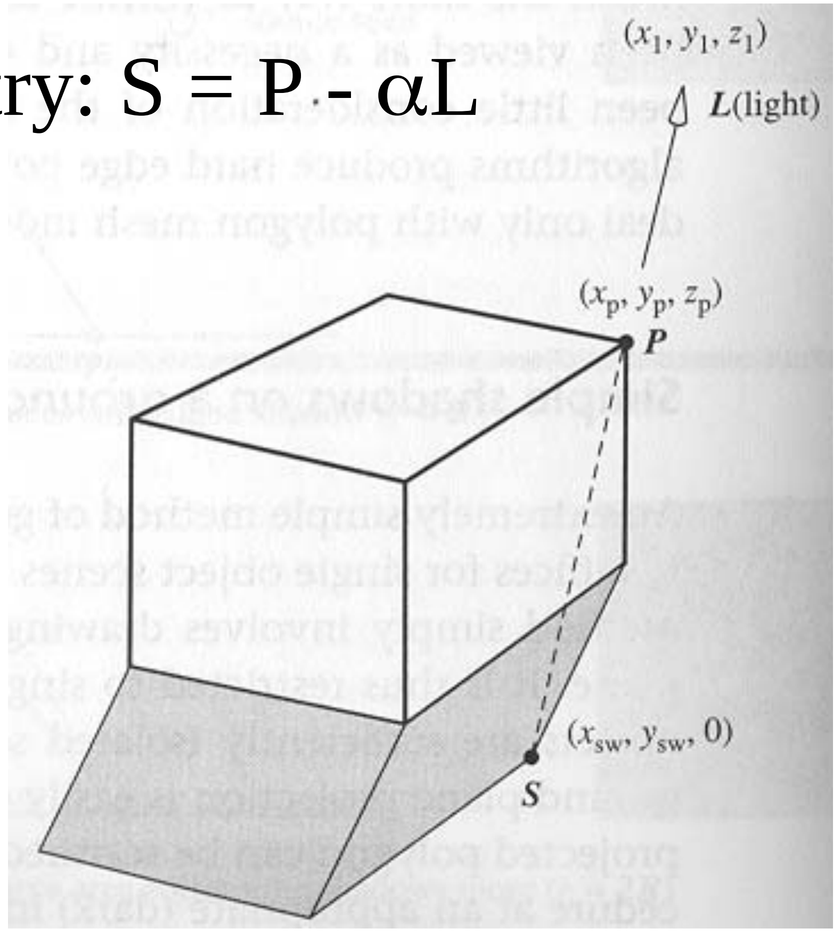
- Considering the geometry: S = P - αL

$$0 = z_p - \alpha z_l$$

$$\alpha = z_p / z_l$$

$$x_{sw} = x_p - (z_p/z_l)\ x_l$$

$$y_{sw} = y_p - (z_p/z_l)\ y_l$$



Blinn's Shadow Algorithm

- In matrix form:

$$\begin{bmatrix} x_{sw} \\ y_{sw} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -x_1/z_1 & 0 \\ 0 & 1 & -y_1/z_1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

- In fact, this is a form of projection, oblique projection

Shadow Algorithms

- Basic idea: Determine which surfaces can be "seen" from the light source
- Surfaces that can not be seen from the light are in shadow
- Shadows in the illumination model:

  $I = \text{ambient} + \Sigma\, S_i\,(\text{diffuse} + \text{specular})$

  $S_i = 0$ if light i is blocked (will cast a shadow)

  $S_i = 1$ if light i is not blocked (the point is lit)

General approach

- Main idea:
  - Point P is in shadow ⇔ P is not visible from light source
- 4 algorithms are discussed in the following:
  - Shadow z-buffer, two-pass z-buffer or (shadow map)
  - Shadow volume
  - Shadowing using Weiler-Atherton algorithm
  - Projecting Polygons/Span-line

Shadow Map

- 1st pass: create a z-buffer from light position, store distance from light source in shadow-buffer [x][y]. (only z-buffer, no color buffer)
- 2nd pass: do z-buffer from eye position. for each visible pixel (x,y,z) in 3D image space
    - inverse map to world space
    - map to screen space of shadow buffer
    - Compare z with that in the shadow buffer[x][y]
    - If shadow buffer[x][y] is smaller, pixel is in shadow!!

Shadow Map

- Advantage:
  - Simple
- Disadvantage:
  - Shadow distance from light position may appear blocky
  - Storage
  - Light source in the view volume?

Shadow Mapping References

- Important SIGGRAPH papers
  - Lance Williams, “Casting Curved Shadows on Curved Surfaces,” SIGGRAPH 78
  - 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 Map

- Depth testing from the light's point-of-view
  - Two pass algorithm
  - First, render depth buffer from the light's point-of-view
    - the result is a "depth map" or "shadow map"
    - essentially a 2D function indicating the depth of the closest pixels to the light
  - This depth map is used in the second pass

Shadow Map

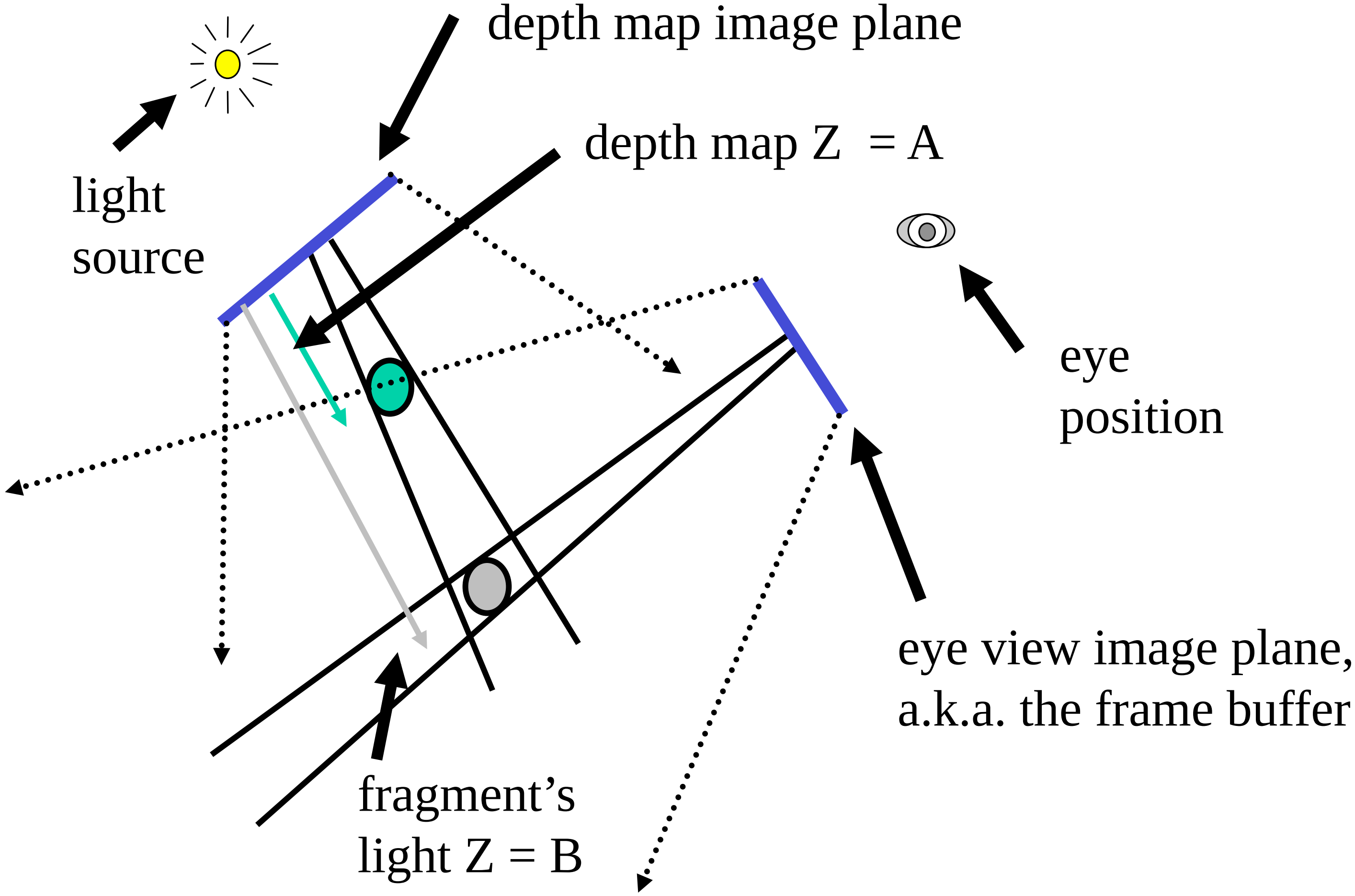
- Shadow determination with the depth map
  - Second, render scene from the eye's point-of-view
  - For each rasterized fragment
    - determine fragment's XYZ position relative to the light
    - this light position should be setup to match the frustum used to create the depth map
    - compare the depth value at light position XY in the depth map to fragment's light position Z

Shadow Map

- 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

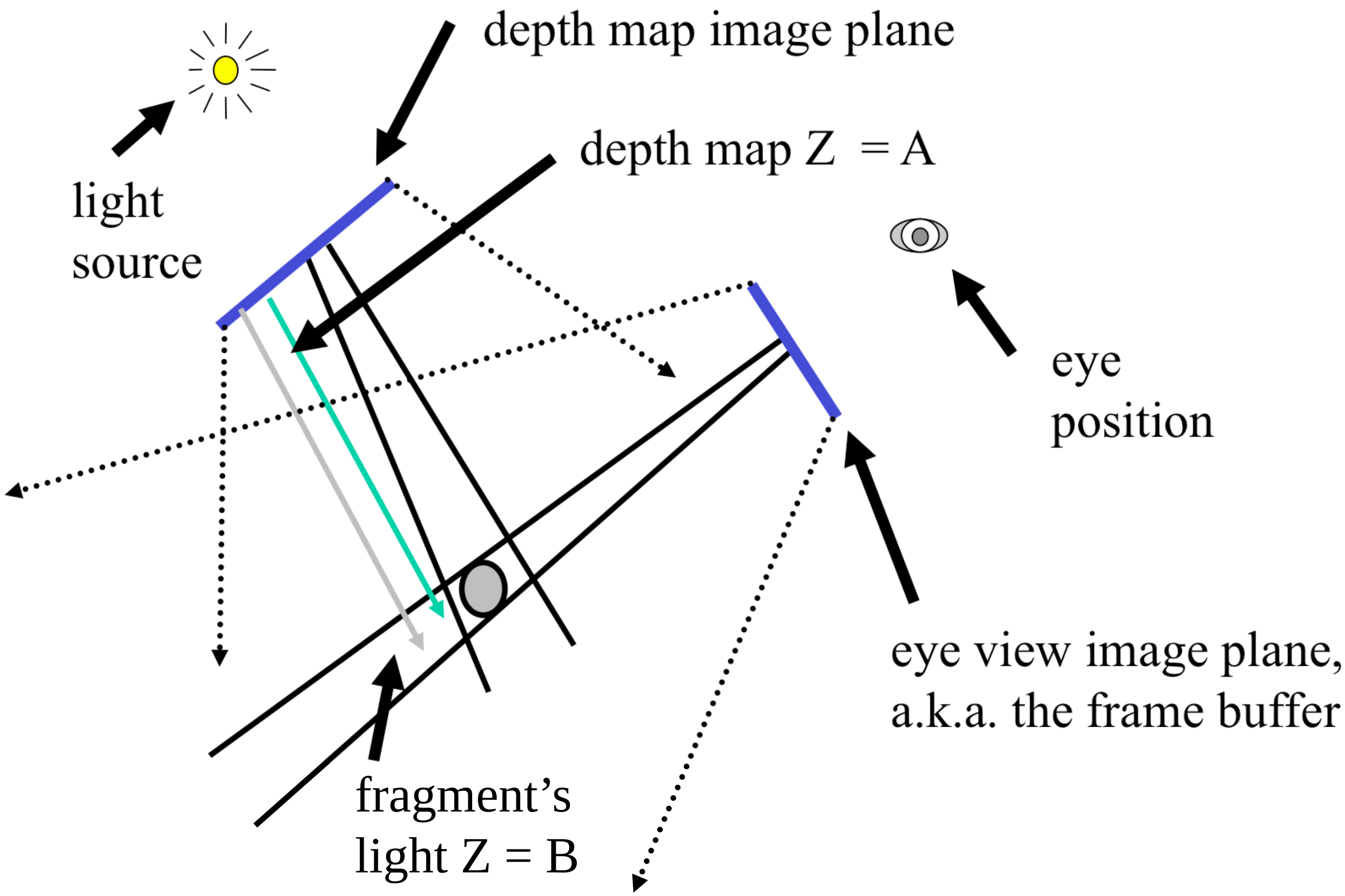
Shadow Mapping with a Picture in 2D (1)

The A < B shadowed fragment case



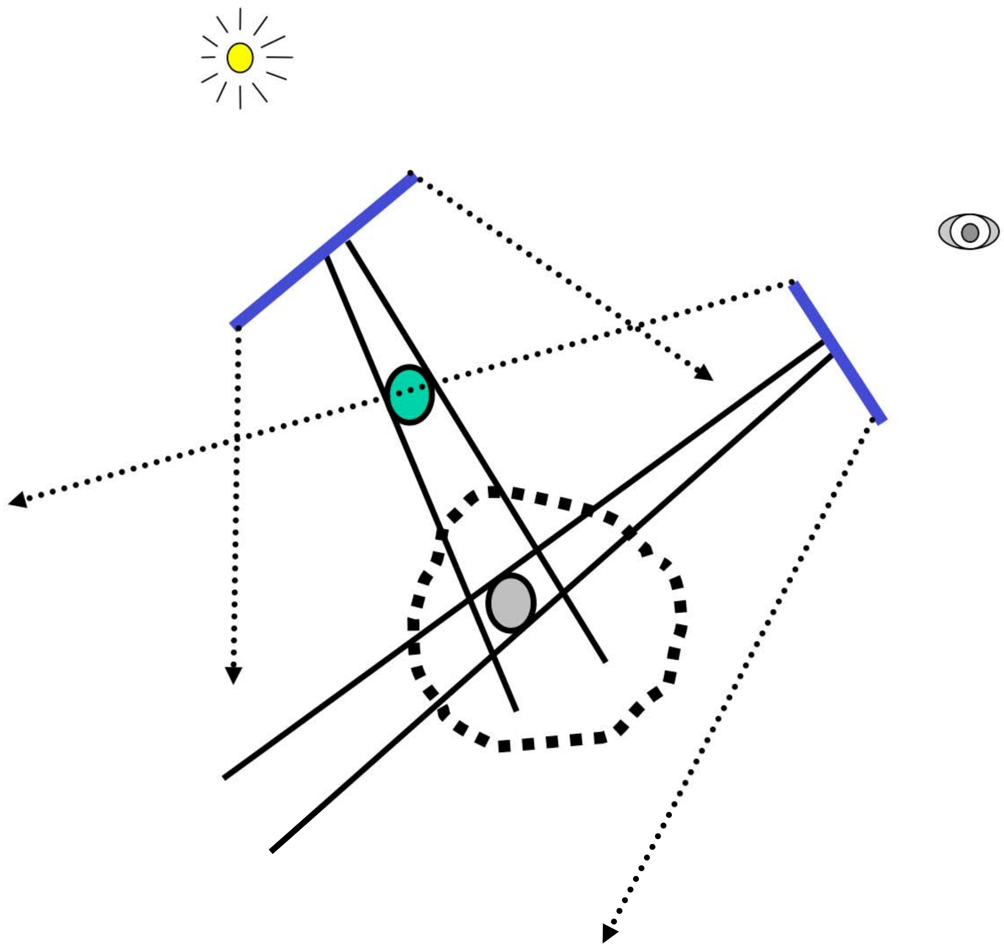
Shadow Mapping with a Picture in 2D (2)

The A ≅ B unshadowed fragment case



Shadow Mapping with a Picture in 2D (3)

Note image precision mismatch!



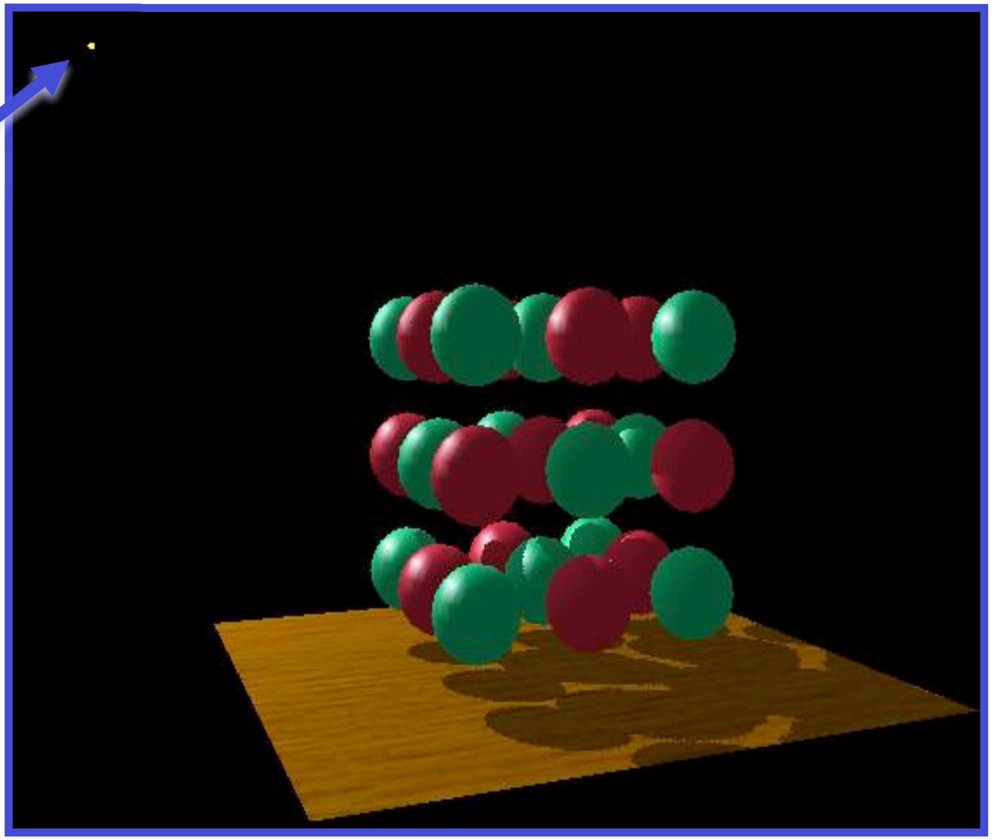
The depth map could be at a different resolution from the framebuffer

This mismatch can lead to artifacts

Visualizing the Shadow Mapping Technique (1)

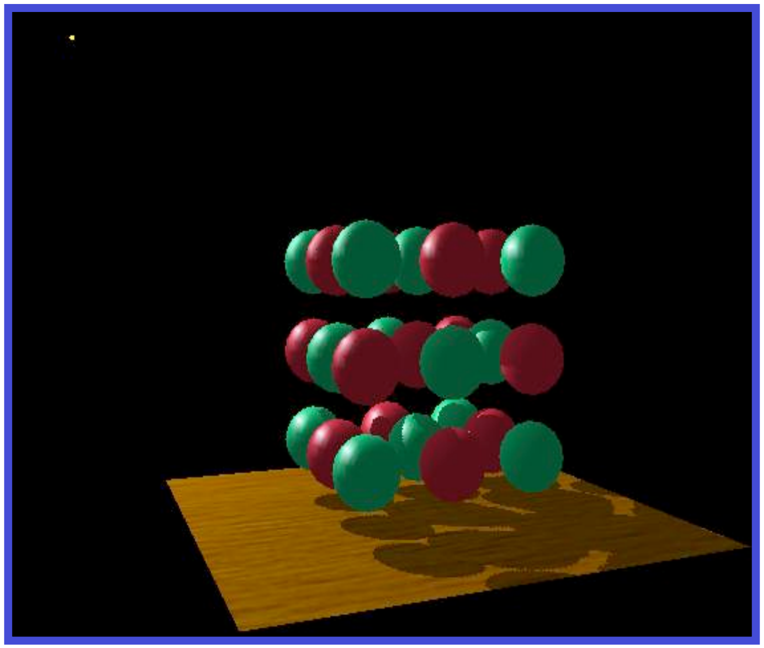
- A fairly complex scene with shadows

the point light source

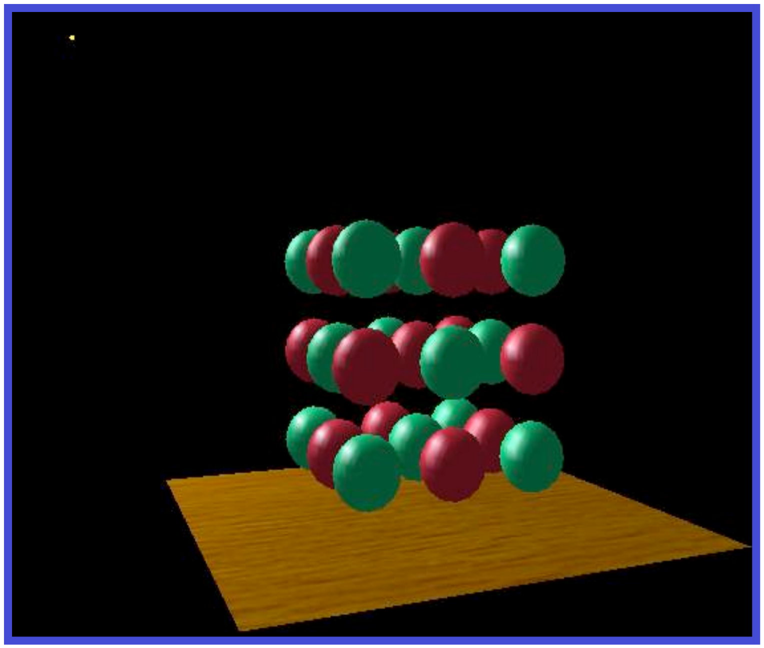


Visualizing the Shadow Mapping Technique (2)

- Compare with and without shadows



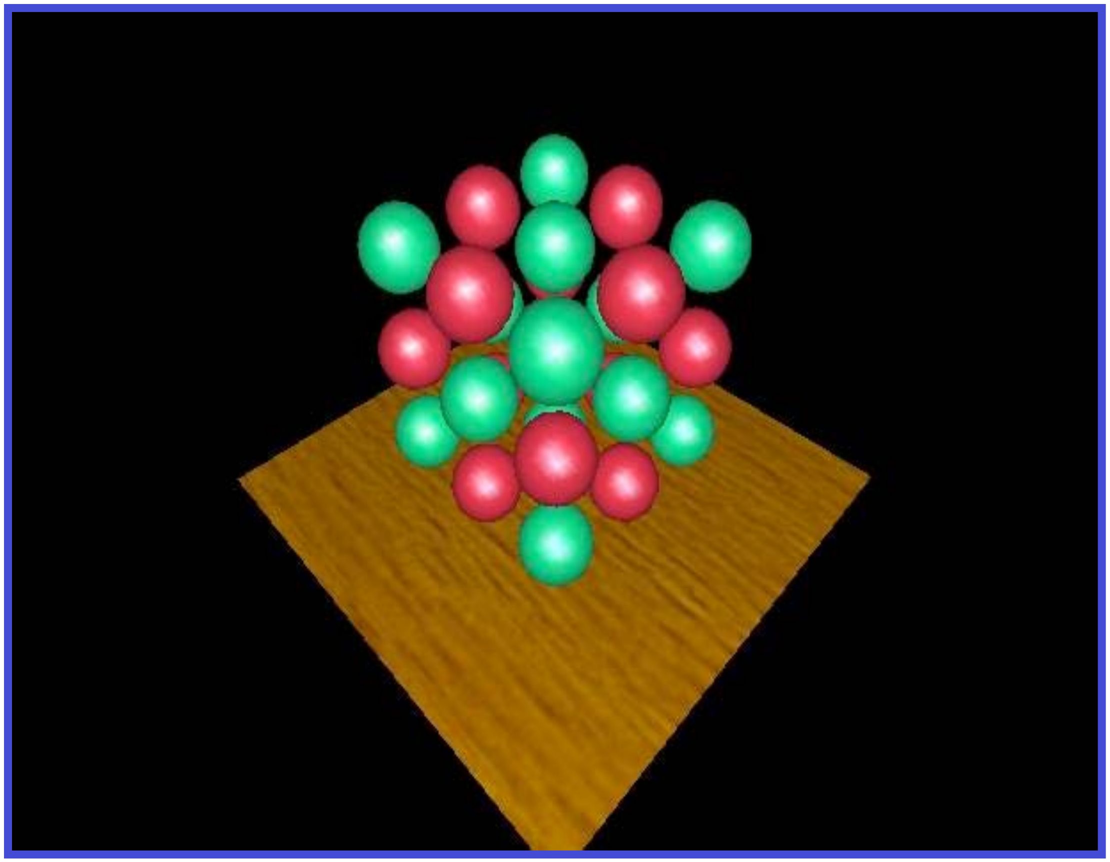
with shadows

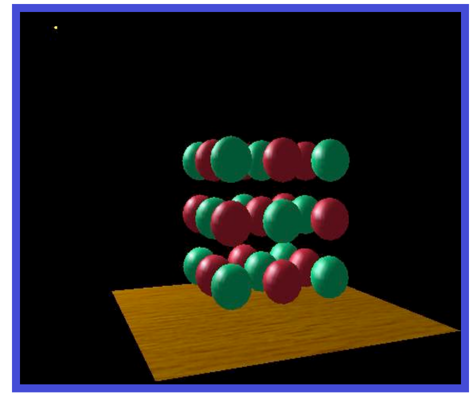


without shadows

Visualizing the Shadow Mapping Technique (3)

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

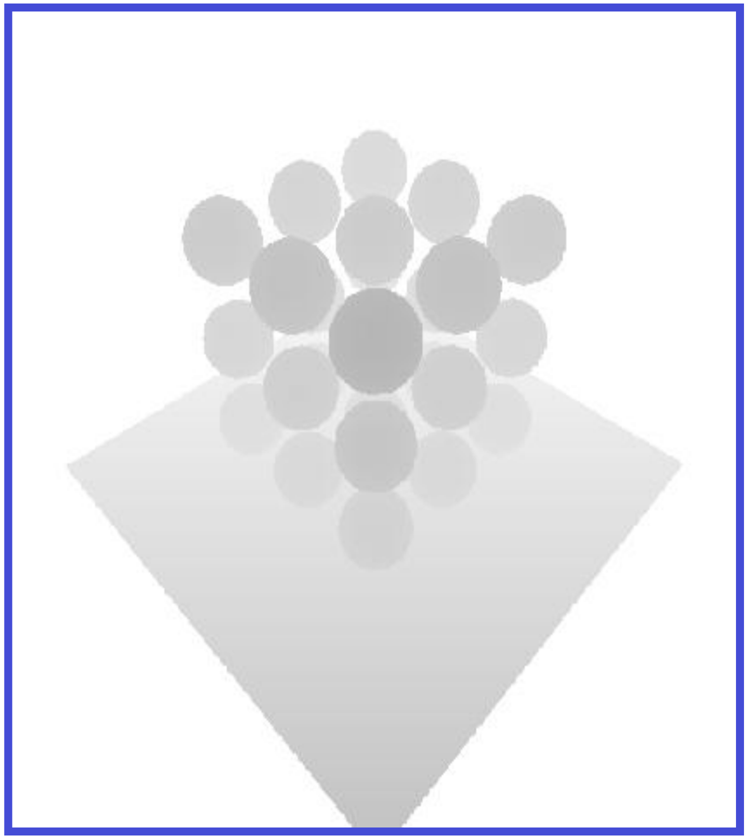


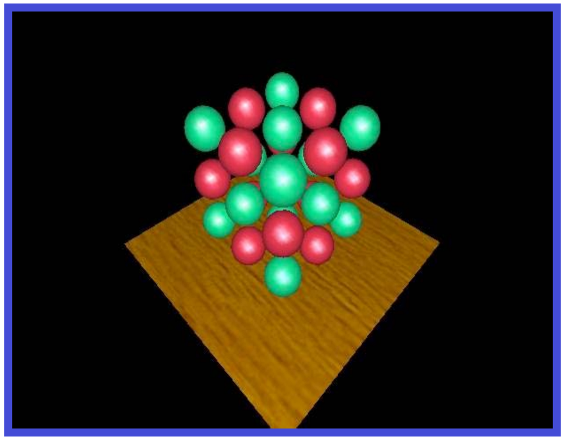


FYI: from the eye's point-of-view again

Visualizing the Shadow Mapping Technique (4)

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

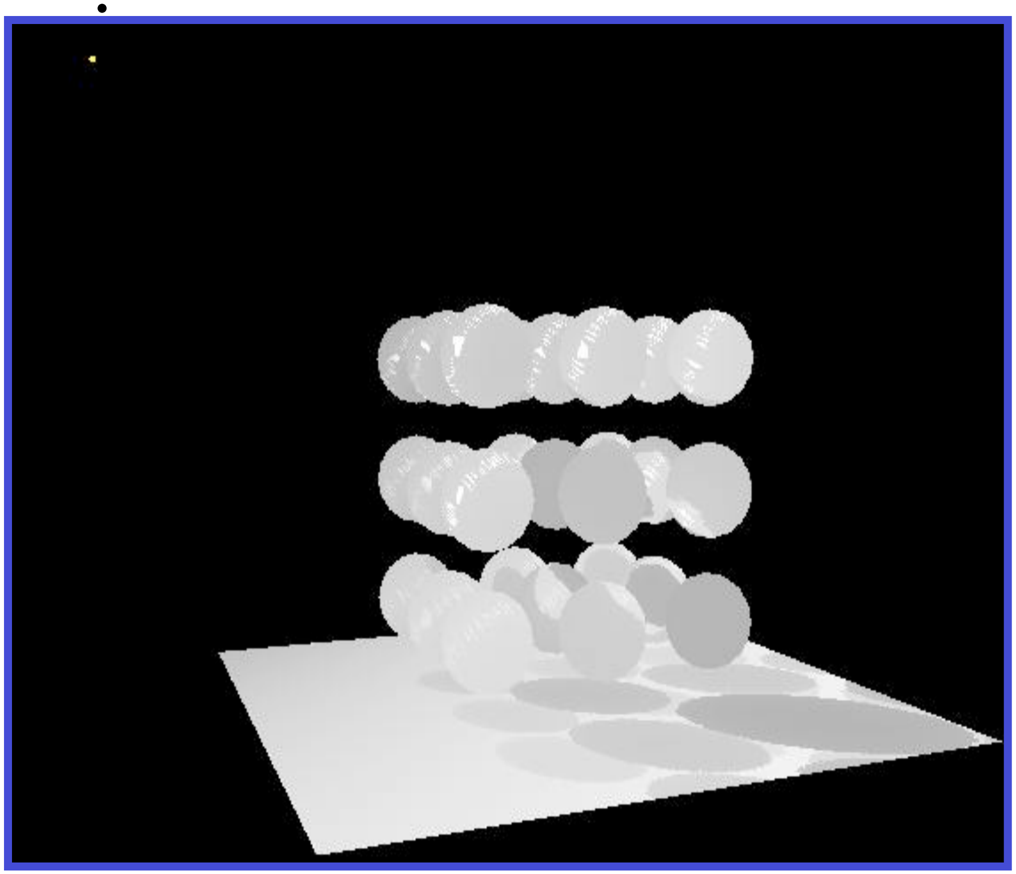


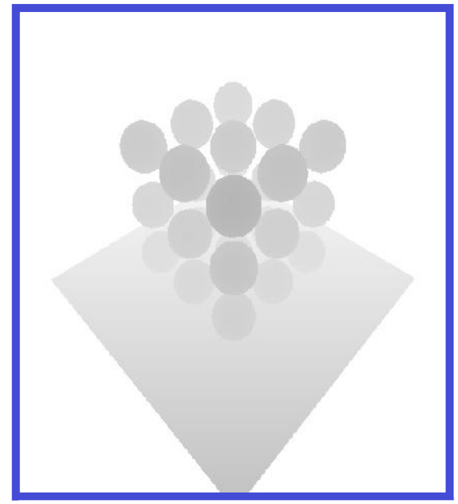


FYI: from the light's point-of-view again

Visualizing the Shadow Mapping Technique (5)

- Projecting the depth map onto the eye's

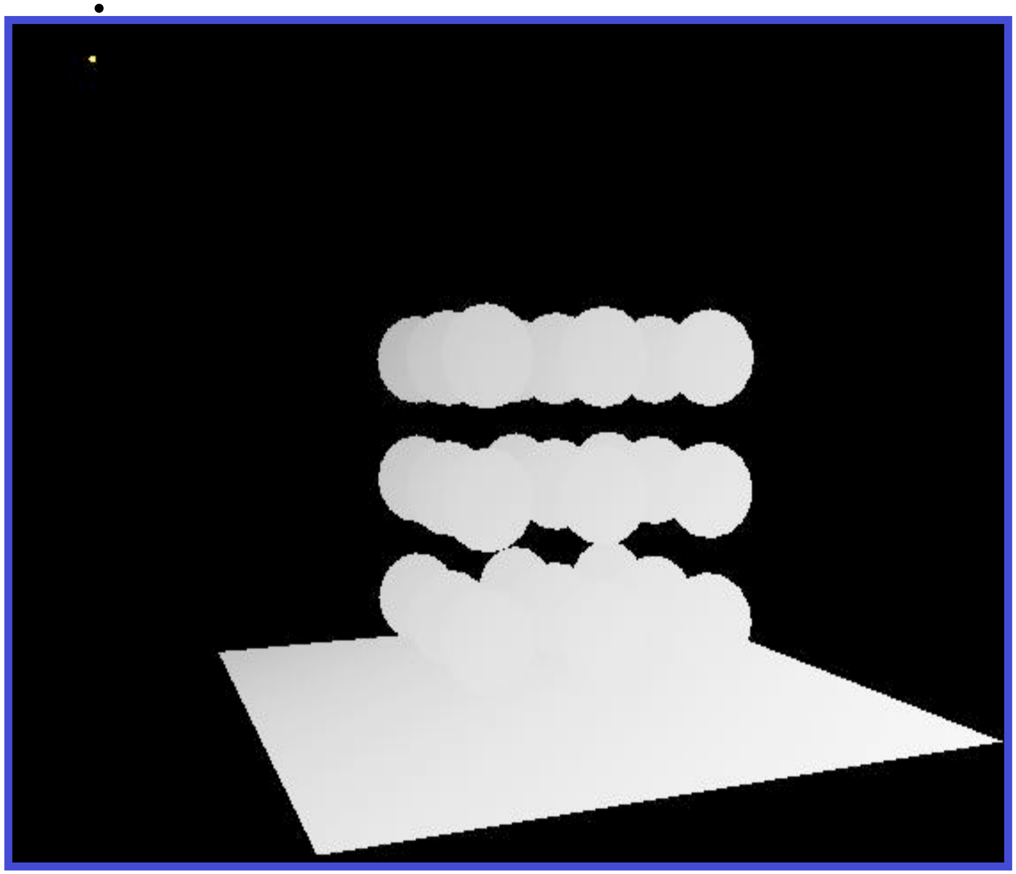




FYI: depth map for light's point-of-view again

Visualizing the Shadow Mapping Technique (6)

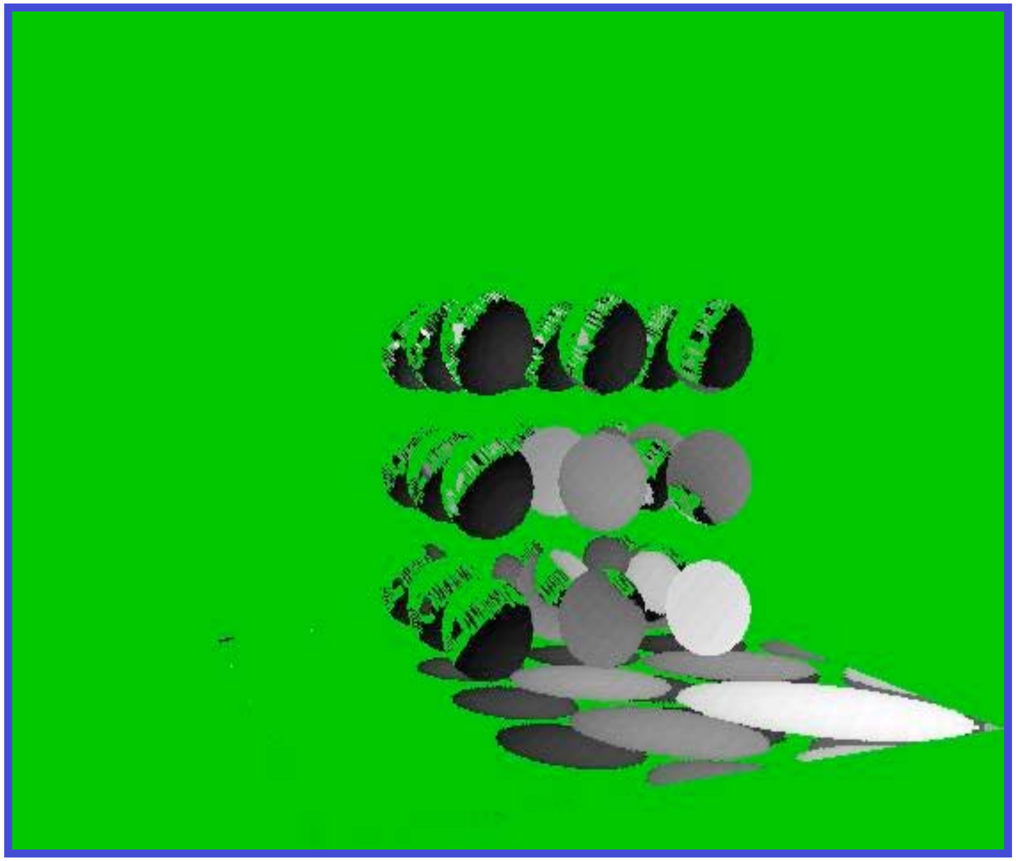
- Projecting light's planar distance onto eye's



Visualizing the Shadow Mapping Technique (6)

- Comparing light distance to light depth map

Green is where the light planar distance and the light depth map are approximately equal

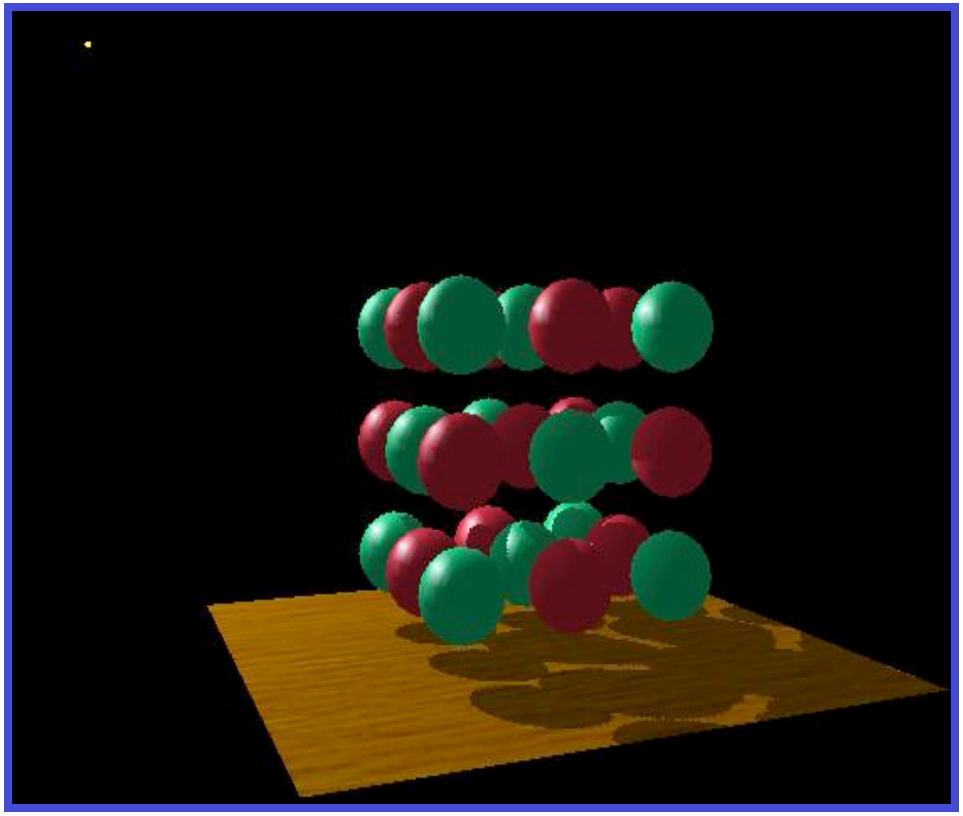


Non-green is where shadows should be

Visualizing the Shadow Mapping Technique (7)

- Scene with shadows

Notice how specular highlights never appear in shadows



Notice how curved surfaces cast shadows on each other

Construct Light View Depth Map

- Realizing the theory in practice
  - Constructing the depth map
    - use existing hardware depth buffer
    - use glPolygonOffset to offset depth value back
    - read back the depth buffer contents
  - Depth map can be copied to a 2D texture
    - unfortunately, depth values tend to require more precision than 8-bit typical for textures
    - depth precision typically 16-bit or 24-bit

Justification for glPolygonOffset When Constructing Shadow Maps

- Depth buffer of “window space” depth values
  - Post-perspective divide means non-linear distribution
  - glPolygonOffset is guaranteed to be a window space offset
- Doing a “clip space” glTranslatef is not sufficient
  - Common shadow mapping implementation mistake
  - Actual bias in depth buffer units will vary over the frustum
  - No way to account for slope of polygon