Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

Brian Russin

Johns Hopkins
Whiting School
of Engineering

Module 8E
Shadow Maps

Shadow Z-Buffer or Shadow Mapping

- Lance Williams (1978) developed a shadow generation algorithm based on 2 passes through a Z-Buffer type algorithm
  - Lance Williams, “Casting Curved Shadows on Curved Surfaces,” SIGGRAPH 78
  - Image precision algorithm (operates in image space)
    - No knowledge of scene’s geometry is required
  - Pass1: for the light source(s)
  - Pass2: from the viewpoint
- A common **software** rendering technique
  - Pixar’s RenderMan uses the algorithm
    - Basic shadowing technique for Toy Story, etc.
- Hardware shadow mapping is now widely available
  - Uses texture mapping extensions
- Watt and Watt, Advanced Animation and Rendering Techniques, describes shadow Z-Buffer theory in extensive detail
  - Including pseudo-code and full anti-aliasing discussion
  - Haines and Moller discuss in section 7.4 (9.1.4 in 3rd Edition)

Shadow Z-Buffer: Creating the Shadow Map

- Shadow Z-Buffer is essentially depth testing from the light's point of view
  - Requires 2 passes
- Pass 1 produces the **shadow map** or **shadow Z-Buffer**
  - Render scene to store depth in the shadow Z-Buffer
    - 2D map indicating the depth of the closest pixels to the light
    - framebuffer object that can be used later as a texture
  - Uses the light source as the view point
  - This computes a depth from the light source
    - Disable writing to framebuffer
    - No lighting or texture mapping
  - The shadow map is used in the second pass
- Directional lights – orthographic projection
- Positional lights – perspective projection

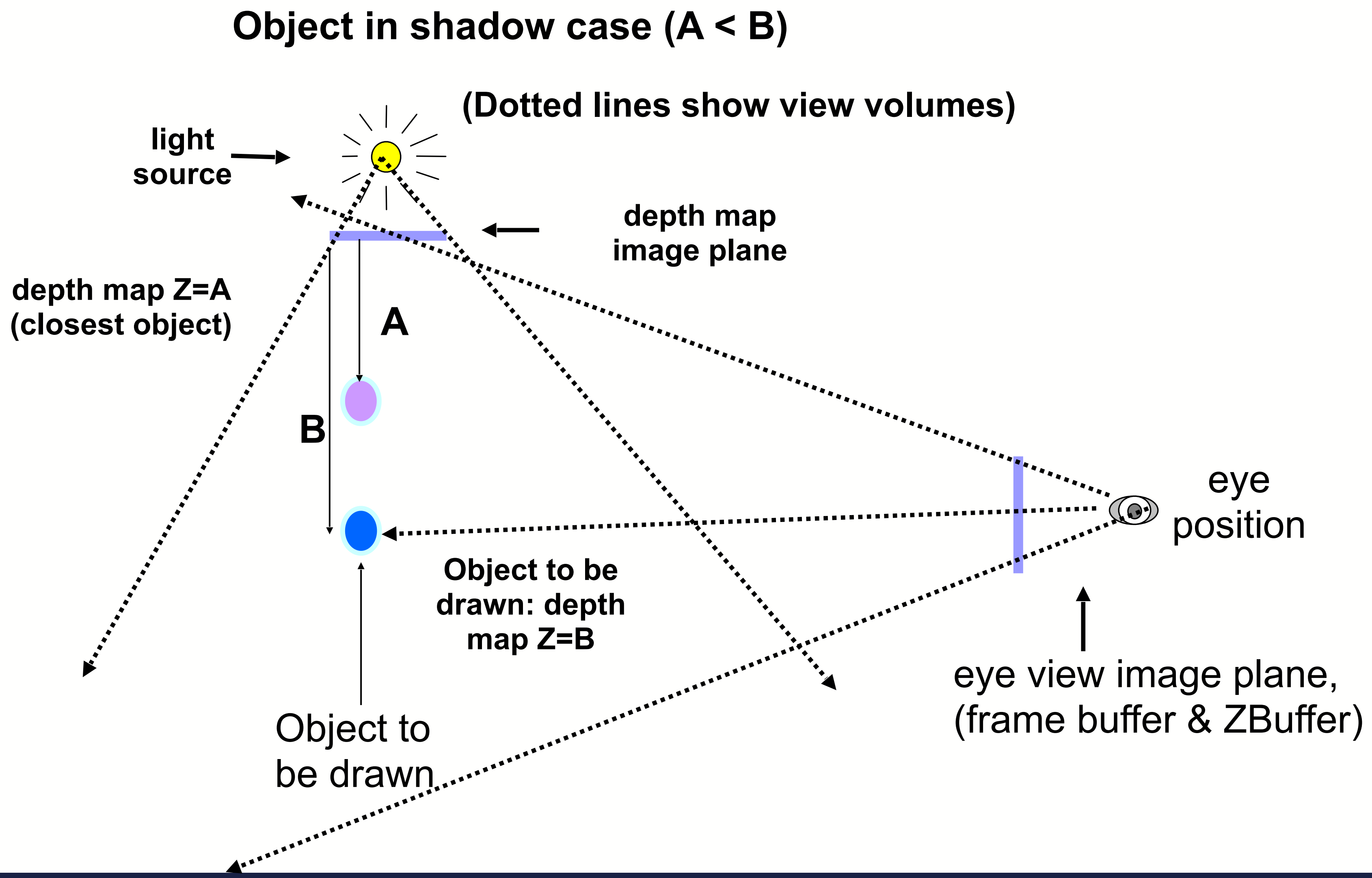
Shadow Z-Buffer: Shadow Determination

- Pass 2: render the scene from the viewer viewpoint using a Z-Buffer algorithm with a couple enhancements:
  - If a pixel is visible (from regular Z-Buffer), it is transformed from screen space to the light source 'screen' space (x', y', z')
- Back map into world coordinates and project (including view transformation) using light source as the viewpoint
  - x', y' provide an index into the shadow Z-Buffer
  - z' is compared to the value in the shadow Z-Buffer
    - If z' is greater, then another surface is nearer to the light source, point is in the shadow and a shadow intensity is assigned
      - Preferably the ambient intensity of the object being rendered
    - Otherwise, the point is on a surface nearest to the light source and is rendered normally
  - If z is approximately equal to z’ we may have a case of “self-shadowing
    - Object we are drawing is the one in the shadow Z-Buffer
    - Need to consider this case carefully

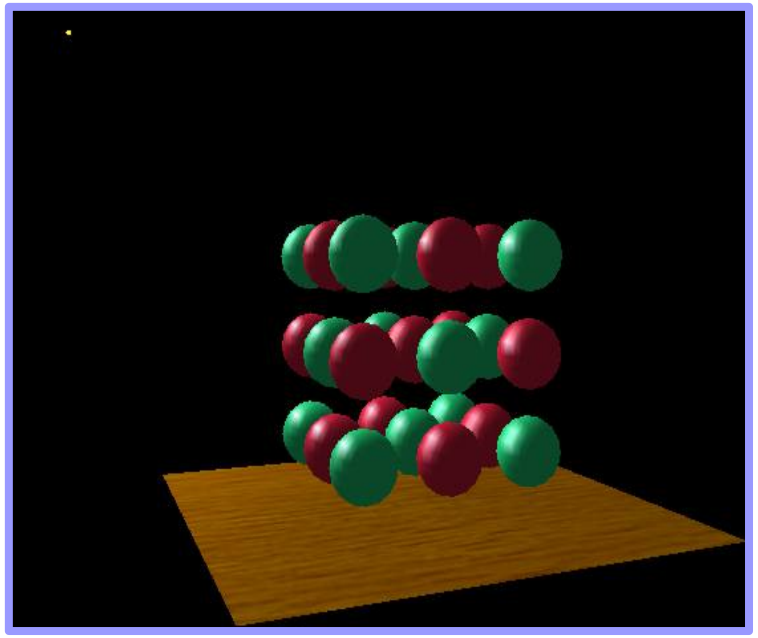
Shadow Z-Buffer (cont.)

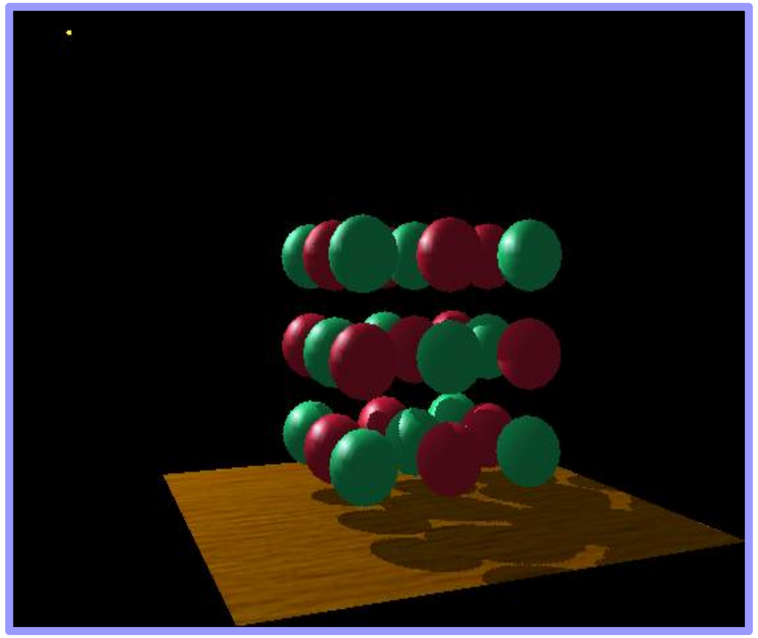
- Shadow Z-Buffers are reasonably simple for single light source
  - Requires a separate shadow Z-Buffers for each light source
    - Or a single shadow Z-Buffers with a separate pass for each light source
      - Would be somewhat inefficient and slow
- Some drawbacks include:
  - Additional memory requirements
  - Per pixel transformation from screen space to light source 'screen' space
  - Shadow calculations performed for a pixel may be subsequently overwritten
- Shadow Z-Buffer is susceptible to aliasing
  - Similar to texture aliasing
    - Effectively projecting a pixel extent onto the shadow Z-Buffers
      - Many shadow map pixels map to one screen pixel
      - Pixel may be partly in shadow and partly not-yet we are making a binary decision
  - Anti-aliasing can be performed at substantial cost
    - Williams used filtering and dithering to reduce aliasing effects

Shadow Mapping



Shadow Map Example

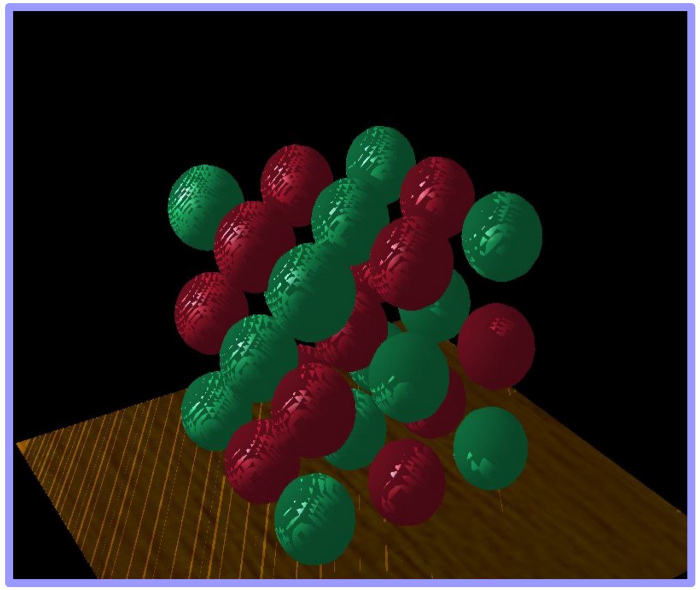




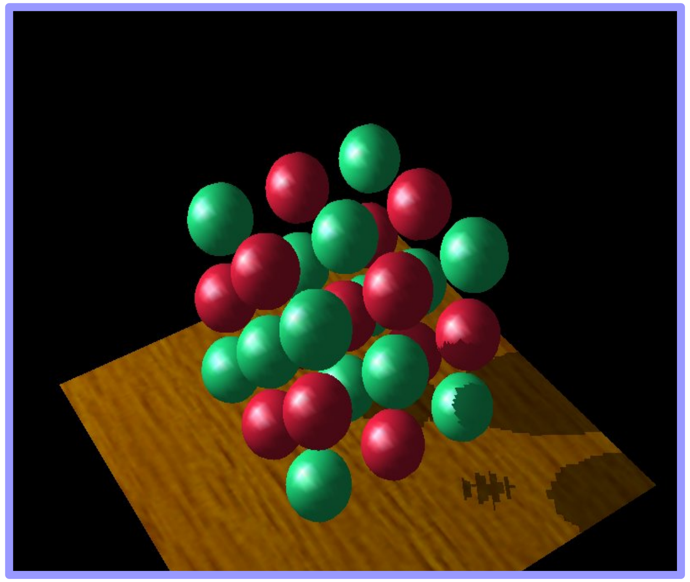
- Note the shadowing onto curved surfaces
  - Best technique for shadows onto curved surfaces
  - Notice how specular highlights never appear in shadows
- From Mark Kilgard presentation: “Shadow Mapping with Today’s OpenGL Hardware”.

Self Shadowing

- Regions where A approximately equal B are regions where self-shadowing is likely occurring
  - Fragment being drawn is the same as the closest to the light source
  - Round off in inverting projection/view as well as precision of buffers can lead to inequality
  - Want a slight “bias” when checking depth to prevent self-shadowing
  - Also known as “shadow acne”



Too little bias: everything self shadows



Too much bias: shadows can start too far away

Hardware Shadow Mapping Support

- Most hardware supports shadow mapping
  - Uses texture mapping resources
    - To store depth map
    - Perform filtering (for anti-aliasing)
    - Rely on projective texture
- NVidia GeForce, SGI, Xbox, and others
  - Performs the shadow test as a texture filtering operation
  - Modulate color with result
    - zero if fragment is shadowed or unchanged color if not
- OpenGL

```
glGenFramebuffers(1, &fbo);
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
glGenTextures(1, &depth_tex);
glBindTexture(GL_TEXTURE_2D, depth_tex);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT16, w, h, 0, GL_DEPTH_COMPONENT, GL_FLOAT, 0);
glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, depth_tex, 0);
```

Pass 1: Constructing the Depth Map

- First pass: render scene from point of view of the light
  - Construct the framebuffer object and association depth texture
    - Performed once
    - Framebuffer object is reused every frame or when scene objects have been updated
  - Enable depth testing to find closest object to the light
  - Use **glPolygonOffset** to create bias in depth from light
    - Help prevent self shadowing
      - Usually better to error on the side of too much bias
        - e.g., glPolygonOffset(factor = 1.1, bias = 4.0)
      - or render only back-facing polygons
    - Adjust to suit the shadows in your scene
      - Depends somewhat on shadow map precision
  - Setup transformation from world space to light source space
  - Bind the framebuffer object and render the scene

Pass 2: Render Scene and Access the Depth Texture

- Render scene from eye's point-of-view as follows
  - Fragment's light position can be generated using the inverse View-Projection matrix multiplied by the Light projection matrix.
    - Set shadow map texture as uniform sampler
    - Set light projection as uniform variable
    - Multiply fragment position to get x', y', z', w'
      - Sample the shadow map using x', y' to get the shadow depth
      - Compare z' to the shadow depth value
  - Compute lighting based on whether the fragment is in shadow, or not
- Tutorial for OpenGL
  - https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

Percentage Closer Filtering

- Percentage Closer Filtering - extends shadow maps to provide pseudo soft shadows
  - Use texture filtering (like when texture magnification occurs) to determine whether the 4 nearest texels are in shadow or not
  - Bilinear interpolation used to estimate the how much light actually at the shadow location (pixel)
  - http://graphics.pixar.com/library/ShadowMaps/paper.pdf
  - http://developer.download.nvidia.com/shaderlibrary/docs/shadow_PCSS.pdf
- Other filtering methods can be used
  - Variance Shadow Maps
  - http://developer.download.nvidia.com/SDK/10/direct3d/Source/VarianceShadowMapping/Doc/VarianceShadowMapping.pdf

Shadow Maps with GLSL

- GLSL has shadow map sampling support
  - http://www.opengl.org/wiki/GLSL_Sampler
  - Uses **depth textures**
  - **shadow2D**, using a **sampler2DShadow** texture sampler
  - **shadow2DProj** – divides by coord.q
- Several references show how to do simple shadow maps with GLSL
  - http://fabiensanglard.net/shadowmapping/index.php

Issues with Shadow Mapping

- Prone to aliasing artifacts
  - "Percentage closer" filtering helps this
  - Normal color filtering does not work well
- Depth bias is not completely foolproof
  - Other methods like "second-depth shadow mapping" can work
    - Render only backfaces into shadow map
- Requires extra shadow map rendering pass and texture loading
- Higher resolution shadow map reduces blockiness
  - Increases texture copying expense
- Shadows are limited to view frustums
  - Not "all-directional"
  - Problematic for light sources in middle of scenes
- Objects outside or crossing the near and far clip planes are not properly accounted for by shadowing
  - Move near plane in as close as possible
    - Not so close that too much depth map precision is lost