VGP353 – Week 2

- Agenda:
  - Introduce shadow maps
    - Differences / similarities with shadow textures
    - Added benefits
    - Potential problems

Shadow Textures

- Shadow textures have a number of faults:
  - Separate texture for each caster / light pair
  - No self-shadowing
  - Difficulty with casters / receivers that are nearly the same distance from the light
- What is the fundamental limitation at the root of all these problems?

Shadow Textures

- Shadow textures have a number of faults:
  - Separate texture for each caster / light pair
  - No self-shadowing
  - Difficulty with casters / receivers that are nearly the same distance from the light
- What is the fundamental limitation at the root of all these problems?
  - Each shadow texel is a simple on-or-off. The remaining information must be inferred.

Light Visibility

- Calculating shadows is just like view visibility
  - Along a particular ray, can a point, p, see the light?

    vs.
  - Along a particular ray, which point, p, can the camera see?

Light Visibility

- Several ways to calculate visibility:
  - Geometric – BSP trees, etc.
  - Image-space – Depth buffer, etc.

Light Visibility

- Remember the multi-pass rendering problem:
  - Draw an object
  - Draw the object again, but combine (blend) the new rendering with the old rendering
    - How can we only draw the second pass to pixels where the first pass was visible?

Light Visibility

- Remember the multi-pass rendering problem:
  - Draw an object
  - Draw the object again, but combine (blend) the new rendering with the old rendering
    - How can we only draw the second pass to pixels where the first pass was visible?
    - Change the depth test function to `GL_EQUAL` or `GL_LEQUAL` and take steps to ensure the vertices are transformed in an identical manner.

Light Visibility

- Note the similarity with the shadow texture problem!
  - A pixel is not in shadow if it's the point in space as the point in the shadow map
  - All other pixels along that light ray are occluded and are in shadow

Shadow Maps

- Use the depth buffer from the shadow texture generation pass
  - Compare the distance read from the shadow map to the distance between the object and the light

$$\begin{cases} d_{object} \leq d_{shadow} & \text{Not in shadow} \\ \text{otherwise} & \text{In shadow} \end{cases}$$

  - The color buffer from the shadow texture generation pass is no longer needed

Shadow Textures vs. Shadow Maps

- Shadow texture:
  - Draw either light color or shadow color to a color texture
  - Read light color directly from shadow texture
  - Color fragment based on light color
- Shadow map:
  - Draw distance to nearest object to a depth texture
  - Compare occluder distance to object distance
  - Color fragment based on result of comparison

Shadow Maps

- Advantages:

Shadow Maps

- Advantages:
  - Objects can self-shadow!
  - Near-by objects can shadow each other correctly

Shadow Maps

- Advantages:
  - Objects can self-shadow!
  - Near-by objects can shadow each other correctly
- Disadvantages:

Shadow Maps

- Advantages:
  - Objects can self-shadow!
  - Near-by objects can shadow each other correctly
- Disadvantages:
  - Aliasing problems
    - Even more than shadow textures
  - More memory usage
  - Omni-directional lights inside the view frustum

Shadow Maps

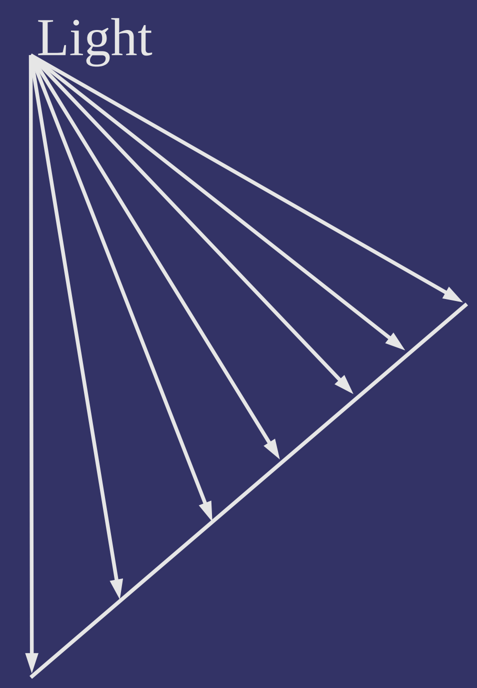
- Algorithm:
  - Group potential casters
  - Calculate frustum that encompasses all objects within a group
  - Render objects using calculated frustum
    - Store depth buffer in a texture (shadow map)
  - Render objects from the camera's PoV with appropriate shadow map
    - Use comparison previously described

Shadow Map Problems

- Four big problems with shadow maps:
  - Sampling differences between shadow map rendering and reading...the dreaded “shadow acne”
  - Aliasing
  - Lack of depth precision
  - Omni-directional lights inside the view frustum

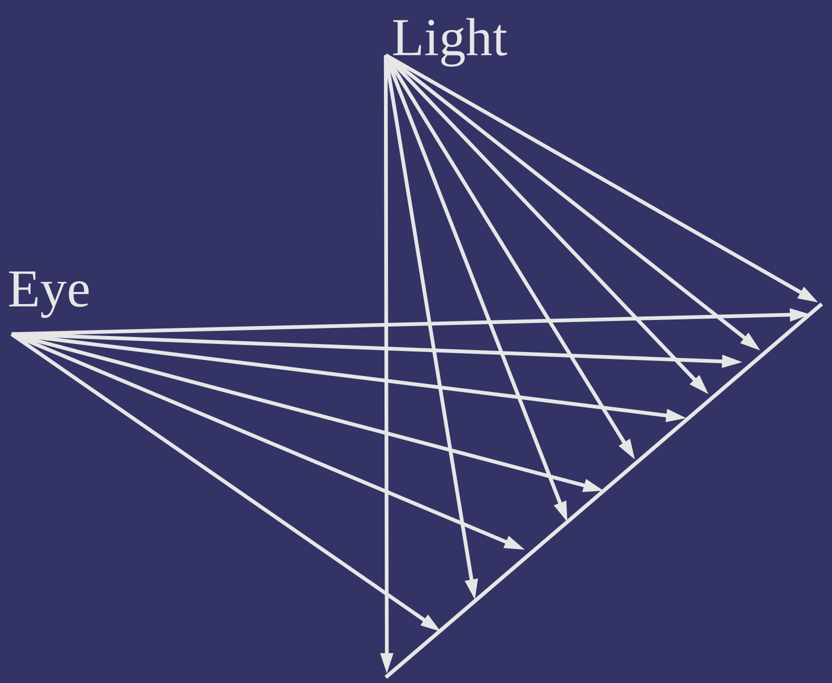
Shadow Acne

- Light and camera sample object at different positions
  - Drawing from the light's PoV samples one set of positions



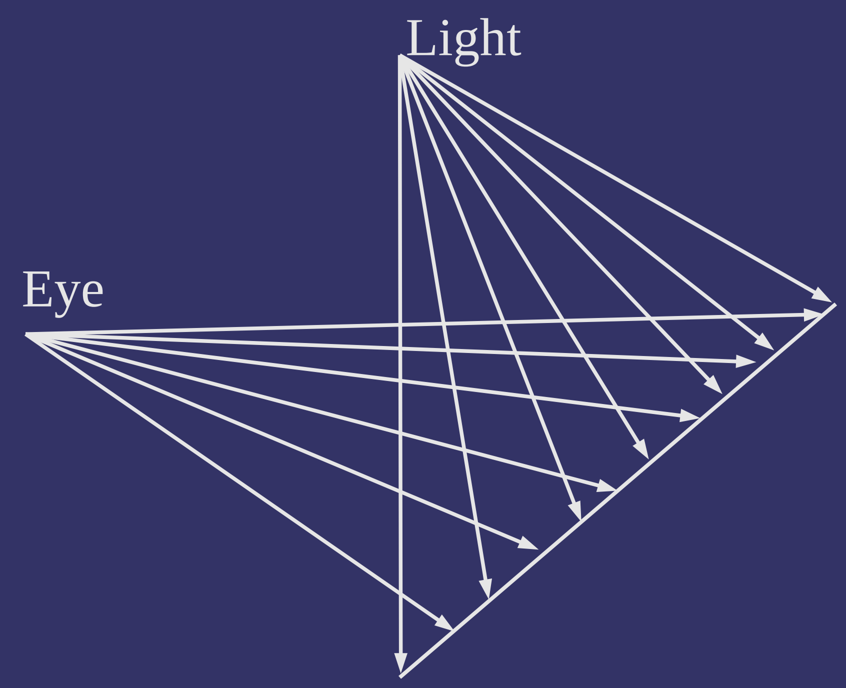
Shadow Acne

- Light and camera sample object at different positions
  - Drawing from the light's PoV samples one set of positions
  - Drawing from the camera's PoV samples a different set of positions

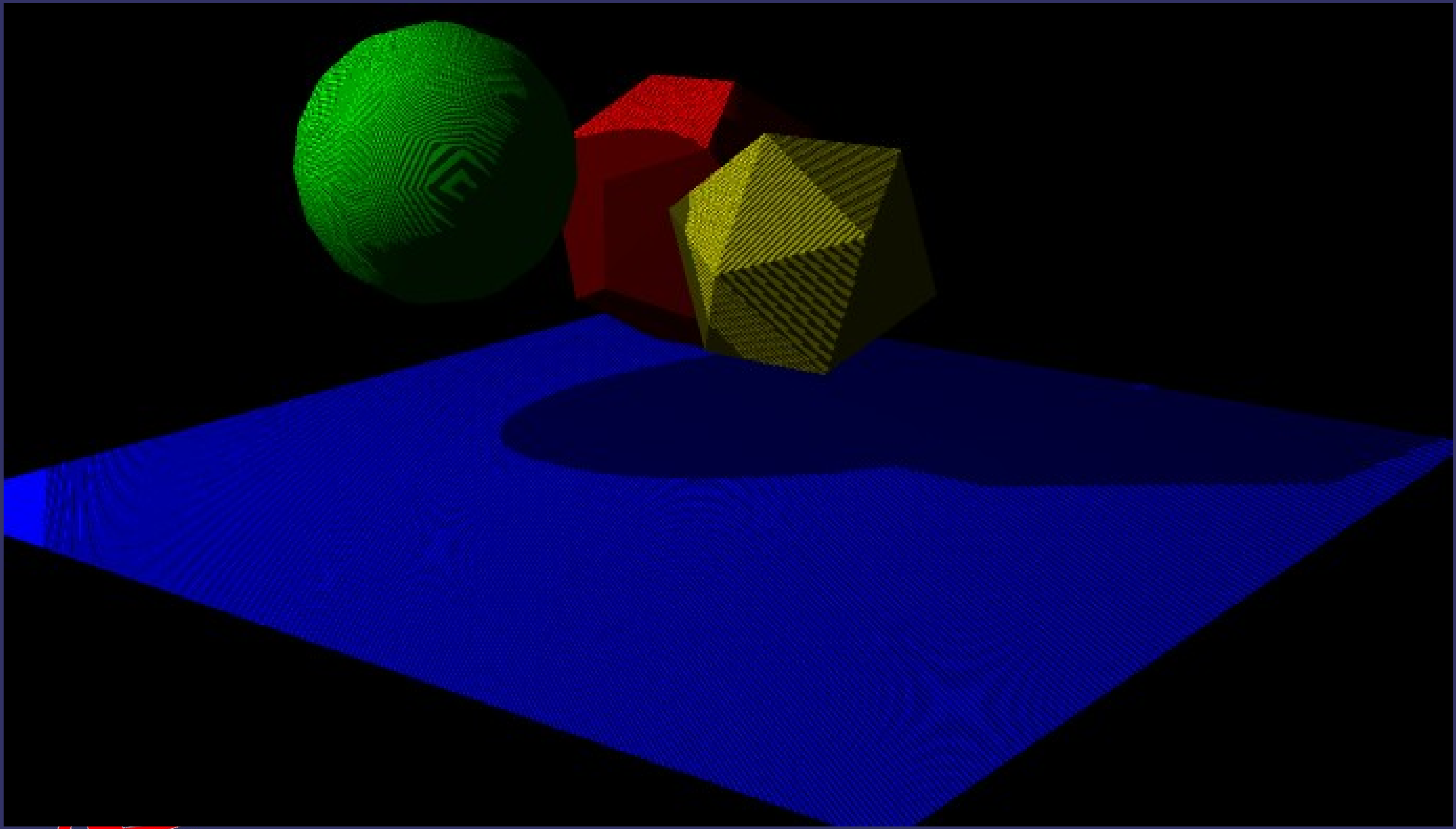


Shadow Acne

- Light and camera sample object at different positions
  - Drawing from the light's PoV samples one set of positions
  - Drawing from the camera's PoV samples a different set of positions
  - Result: incorrect values are used to determine if a surface shadows itself



Shadow Acne



Shadow Acne

- Two common solutions:

Shadow Acne

- Two common solutions:
  - Render back faces to shadow map
    - Front faces aren't drawn to shadow map, so they won't self-shadow
    - Back faces aren't lit: depth comparison result is irrelevant

Shadow Acne

- Two common solutions:
  - Use polygon offset
    - Bias fragment depth by small factor to ensure $d_{shadow} \geq d_{object}$

```
glPolygonOffset(1.1f, 1.0f);
```

    - Very tricky to get right! Movie fx companies spend *lots* of time tweaking every frame to eliminate artifacts[1]

[1] G. King, “Shadow Mapping Algorithms.” NVIDIA. 2004.
ftp://download.nvidia.com/developer/presentations/2004/GPU_Jackpot/Shadow_Mapping.pdf

Shadow Acne

- Two common solutions:
  - Render back faces to shadow map
    - Front faces aren't drawn to shadow map, so they won't self-shadow
    - Back faces aren't lit: depth comparison result is irrelevant
    - Can still have acne when normal map causes a polygon facing away from the light to be lit

Shadow Map Aliasing

- Several sources of aliasing in shadow maps
  - Must use nearest-neighbor sampling
    - Bilinear or mipmap sampling would average depth values together for use in comparison
  - Depth maps are typically small, so fine details may get lost
    - Shadows from thin objects (telephone wires, chain link fence, etc.) may disappear
    - Small gaps between objects may fill-in
  - Objects distant from light may be too small in shadow map
    - If the object's shadow is near the camera, it will appear very blocky

Shadow Map Precision

- Every Z-buffer has potential precision problems

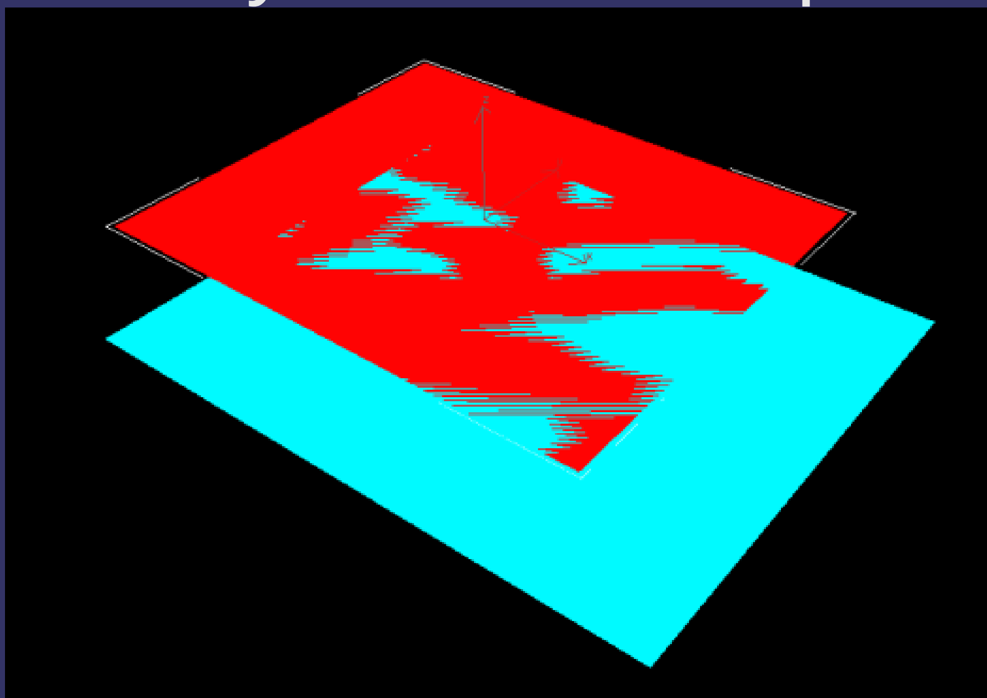
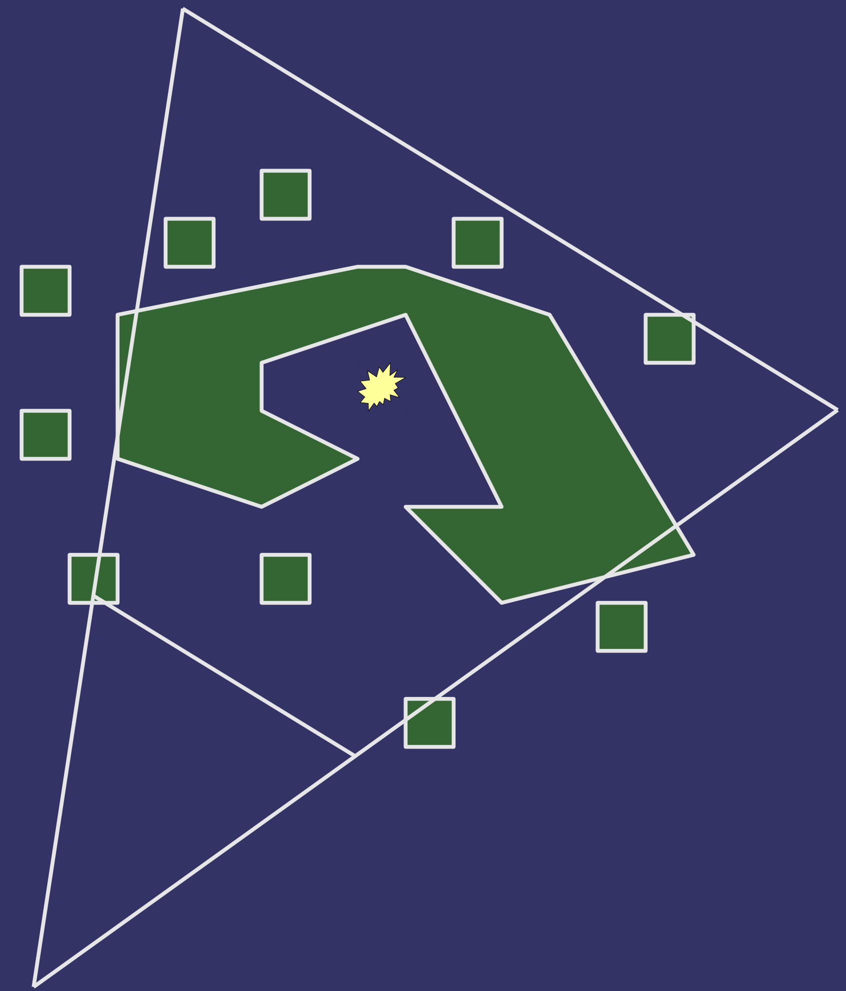


Image from http://en.wikipedia.org/wiki/Z-fighting

  - Objects distant from near-plane get fewer significant bits to store depth
    - May not be noticeable far from the near plane
  - Due to viewing differences, lack of Z precision far from *light's* near-plane may result in artifacts close to *camera's* near-plane

Omni-directional Lights

- Consider this scene...
  - What frustum do we pick for the light and the large object?
  - We'd need a 360° field-of-view!



Shadow Maps in GLSL

- New sampler types:
  - `sampler1DShadow`
  - `sampler2DShadow`
  - `sampler2DRectShadow`

Shadow Maps in GLSL

- New sampler functions:

```
vec4 shadow2D(sampler2DShadow, vec3)
vec4 shadow2DProj(sampler2DShadow, vec4)
```

  – 3rd component of texture coordinate is the distance used for comparison
  – There are also `1D` and `2DRect` versions
  – Value returned depends on comparison mode and `GL_DEPTH_TEXTURE_MODE` setting of texture unit

– As with projective textures, use shadow sampler types and functions instead of doing comparisons by hand

Shadow Maps in GLSL

- OpenGL 3.0 and GLSL 1.30 change things:
  - `GL_DEPTH_TEXTURE_MODE` is deprecated
    - You don't want it.
    - It's removed completely in 3.1
  - GLSL texture functions change name and return type:

```
float texture(sampler2DShadow, vec3)
float textureProj(sampler2DShadow, vec4)
```

- - `1D` and `2DRect` versions get similar changes

Shadow Maps in GLSL

- For GLSL 1.20 and earlier:
  - Leave `GL_DEPTH_TEXTURE_MODE` in the default state
    - `GL_LUMINANCE`
  - Wrap 1.20 API to look like 1.30 API:

```
float texture(sampler2DShadow s, vec3 c)
{
    return shadow2D(s, c).x;
}
```

Shadow Maps in GLSL

- Each texture has a depth comparison mode
  - Mode is set by calling `glTexParameteri` with *name* of `GL_TEXTURE_COMPARE_FUNC`
  - Sets mode used for comparison in `sampler[12]D` functions
  - Comparison mode is one of the “usual” `GL_LEQUAL`, etc. modes.
- Sampler function returns 1.0 if the test passes or 0.0 if the test fails

Depth Textures

- Store single component, normalized value used for depth (shadow) comparisons
  - Use one of three internal formats:
    - `GL_DEPTH_COMPONENT16`
    - `GL_DEPTH_COMPONENT24`
    - `GL_DEPTH_COMPONENT32`
  - Only format that can be used with GLSL shadow samplers
    - Can be also use with non-shadow samplers as a luminance, intensity, or alpha texture

Depth Textures

- Create just like any other texture:

```
glBindTexture(GL_TEXTURE_2D, my_shadow_tex);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT24,
             0, 0, width, height, GL_DEPTH_COMPONENT,
             GL_UNSIGNED_INT, NULL);
```

Depth Textures

- Create just like any other texture:

```
glBindTexture(GL_TEXTURE_2D, my_shadow_tex);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT24,
             0, 0, width, height, GL_DEPTH_COMPONENT,
             GL_UNSIGNED_INT, NULL);
```

- To use as false-color texture:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
                GL_NONE);
```

  - Color returned is `vec4(d, d, d, 1)`

Depth Textures

- Create just like any other texture:

```
glBindTexture(GL_TEXTURE_2D, my_shadow_tex);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT24,
             0, 0, width, height, GL_DEPTH_COMPONENT,
             GL_UNSIGNED_INT, NULL);
```

- To use as false-color texture:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
                GL_NONE);
```

  - Color returned is `vec4(d, d, d, 1)`

- To use as a shadow map:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
                GL_COMPARE_R_TO_TEXTURE);
```

Depth Textures

- Set comparison function similarly:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC,
                GL_LESS);
```

  - In OpenGL 1.4 only `GL_LEQUAL` and `GL_GEQUAL` were available
  - In OpenGL $\geq$ 1.5 all 8 functions are available

Depth Textures and FBOs

- Attach the depth-component texture to the depth attachment:

```
glFramebufferTexture2D(GL_FRAMEBUFFER,
                       GL_DEPTH_ATTACHMENT,
                       GL_TEXTURE_2D, tex, 0);
```

  - If there are no mipmaps (likely), as usual, be sure to set non-mipmap minification mode
  - If there is no color output (likely), be sure to disable all color buffer access:

```
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
```

Next week...

- Advanced shadow map techniques
  - Quiz #1
  - Assignment #1, part 2... due *next week*
  - Read:

W. Reeves, D. Salesin, and R. Cook, "Rendering Antialiased Shadows with Depth Maps." In Proceedings of SIGGRAPH ’87. 1987. http://graphics.pixar.com/ShadowMaps/

R. Fernando, "Percentage-Closer Soft Shadows." In Proceedings of SIGGRAPH 2005. 2005. http://developer.nvidia.com/object/siggraph_2005_presentations.html

  - Reducing shadow map aliasing
  - Percentage closer soft shadows (PCSS)
  - Depth range optimizations

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.