VGP353 – Week 2

- Agenda:
  - Assignment #1 due
  - Introduce shadow maps
    - Differences / similarities with shadow textures
    - Added benefits
    - Potential problems

Shadow Textures

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

Shadow Textures

- As discussed last week, shadow textures have a number of faults
  - Separate texture for each caster / light pair
  - No self-shadowing
  - Difficulty with casters / receivers that a 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.

Shadow Textures

- To determine whether a position in 3-space is in shadow, what information is needed?

Shadow Textures

- To determine whether a position in 3-space is in shadow, what information is needed?
  - Is there something *closer* to the light in the direct line-of-sight
    - The shadow texture only tells whether there is something in the line of sight, *not* whether that something is closer to the light

Shadow Maps

- Instead of storing boolean “shadow” / “not shadow”, store the distance from the light to the closest shadow caster
  - This is a *shadow map*
  - Compare the distance read from the shadow map to the distance between the object and the light
    - If $distance_{shadow} < distance_{object}$, the fragment is in shadow
    - If $distance_{shadow} \geq distance_{object}$, the fragment is not in shadow

Shadow Maps

- Shadow map stores "distance to nearest shadow caster."
  - *Remind* you of anything?

Shadow Maps

- Shadow map stores "distance to nearest shadow caster."
  - *Remind* you of anything?
  - A depth buffer!
    - Depth buffer (typically) stores the per-pixel distance to the object nearest to the eye
    - When rendering from the light's PoV, the distance stored in the depth buffer is the distance to the object nearest to the light

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 base 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:
  - Separate texture for each caster / light pair

Shadow Maps

- Advantages:
  - Objects can self-shadow!
  - Near-by objects can shadow each other correctly
- Disadvantages:
  - Separate texture for each caster / light pair
    - *Is this necessary? NO!*

Shadow Maps Revised

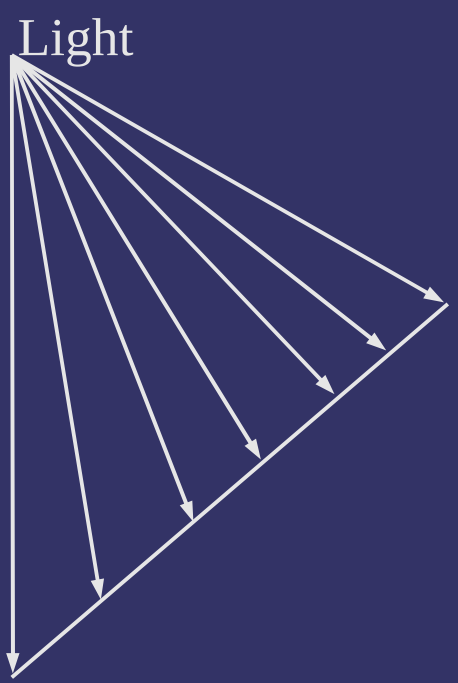
- Algorithm:
  - Group potential casters and receivers
  - Calculate frustum that encompasses all objects within a group
  - Render objects using calculate frustum. Store depth buffer in a texture (shadow map)
  - Render objects from the camera's PoV with appropriate shadow map. Use comparison previously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

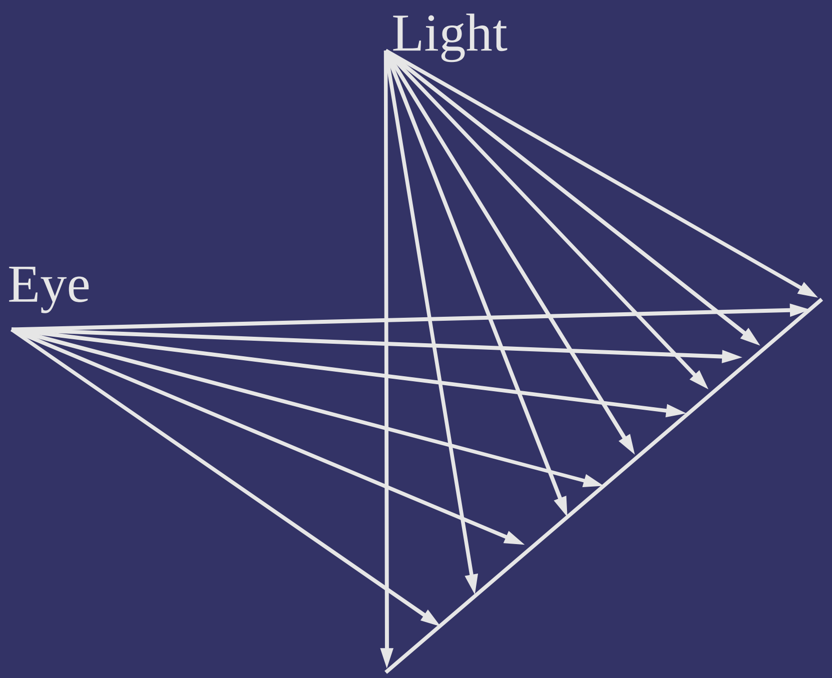
Shadow Acne

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



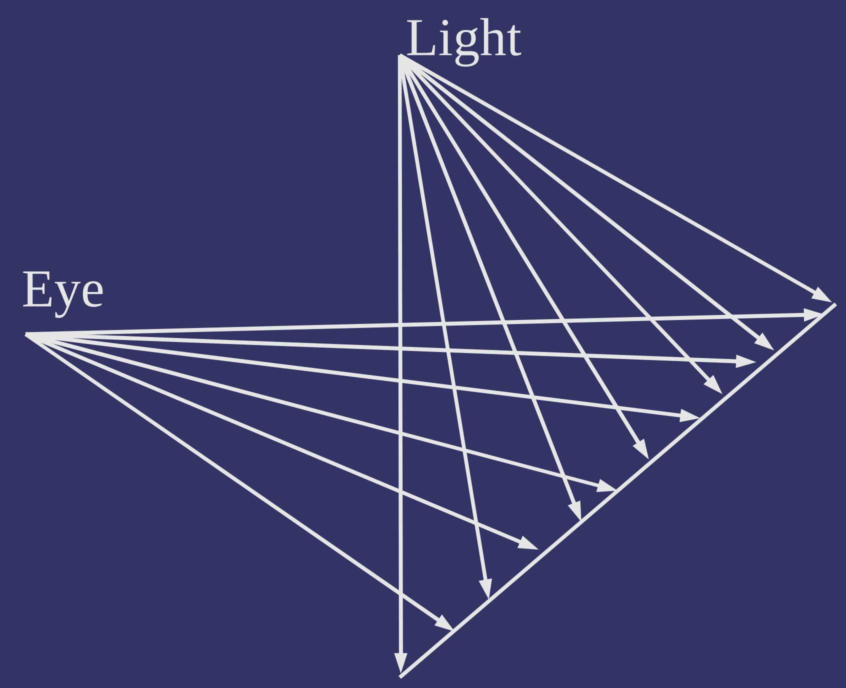
Shadow Acne

- 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

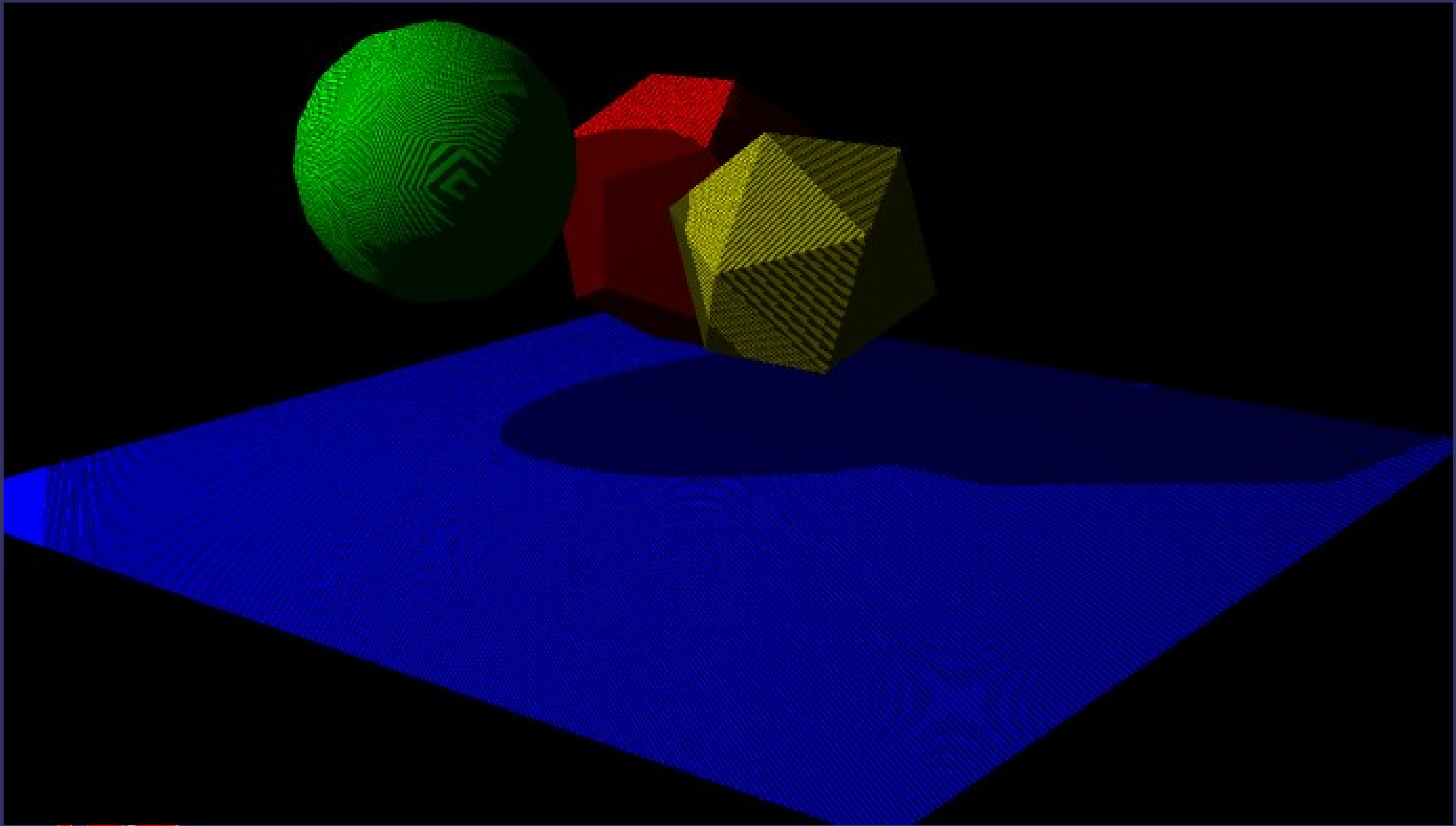


Shadow Acne

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

```
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 Map Aliasing

- Several sources of aliasing in shadow maps
  - Must use nearest-neighbor sampling
    - Straightforward bi-linear 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

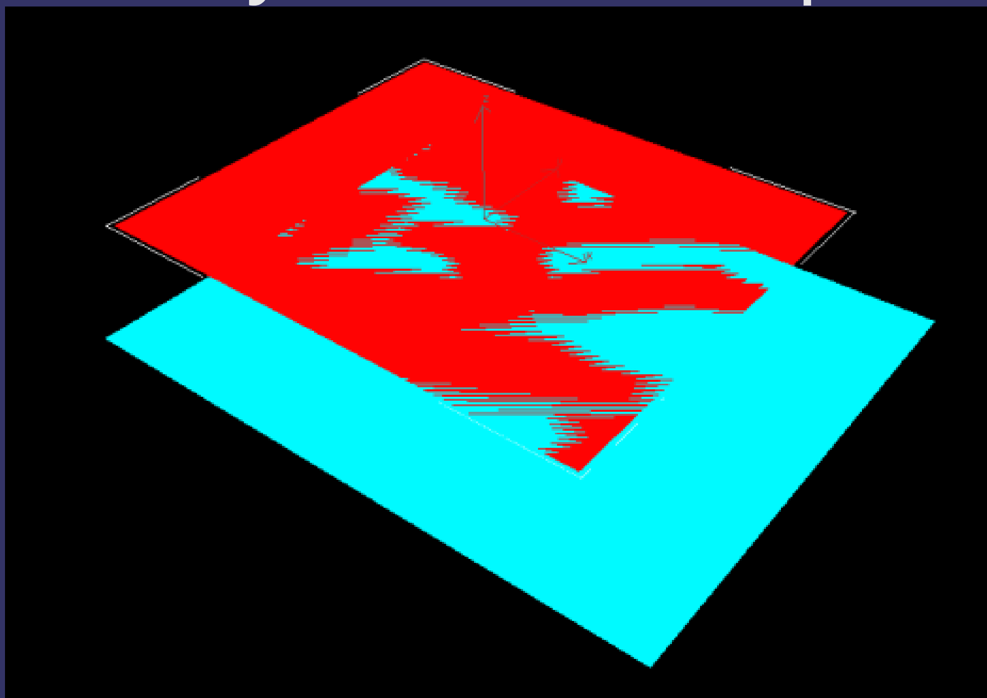
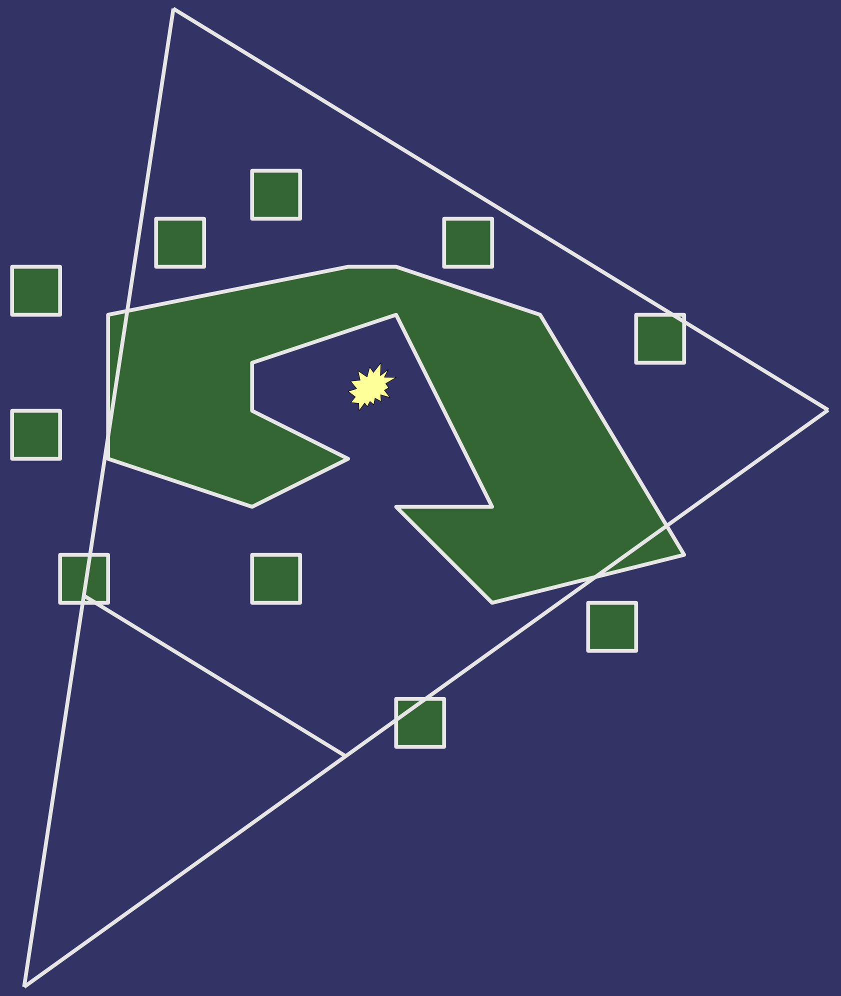


Image from http://en.wikipedia.org/wiki/Z-fighting

  - 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!



21-July-2009

Shadow Maps in GLSL

- New sampler types:
  - `sampler1DShadow` and `sampler2DShadow`
- New sampler functions:
  - `shadow1D` and `shadow1DProj`
  - `shadow2D` and `shadow2DProj`
    - 3rd component of texture coordinate is the distance used for comparison
  - As with projective textures, use shadow sampler types and functions instead of doing comparisons by hand

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
- 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_COMPONENT24,
             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_COMPONENT24,
             GL_UNSIGNED_INT, NULL);
```

- To use as false-color texture:

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

  - Or `GL_LUMINANCE` or `GL_ALPHA`

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_COMPONENT24,
             GL_UNSIGNED_INT, NULL);
```

- To use as false-color texture:

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

  - Or `GL_LUMINANCE` or `GL_ALPHA`

- 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 1.5 and later all of the comparison functions are available

Depth Textures and FBOs

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

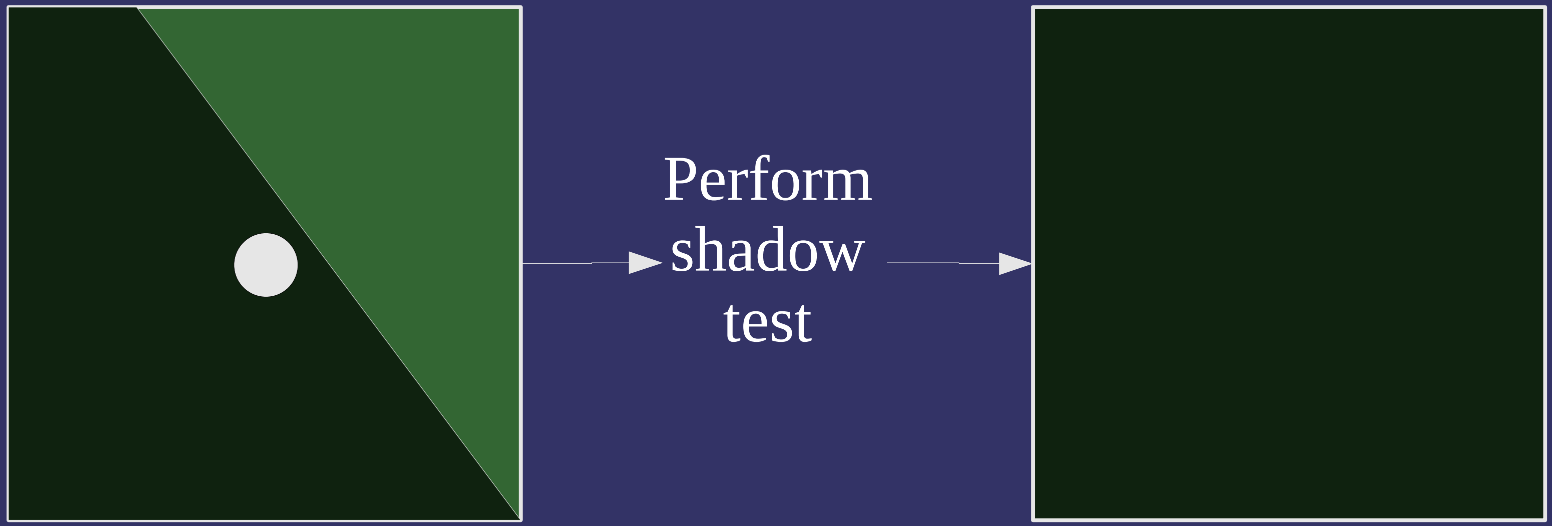
```
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
                          GL_DEPTH_ATTACHMENT_EXT,
                          GL_TEXTURE_2D, tex, 0);
```

  – 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);
```

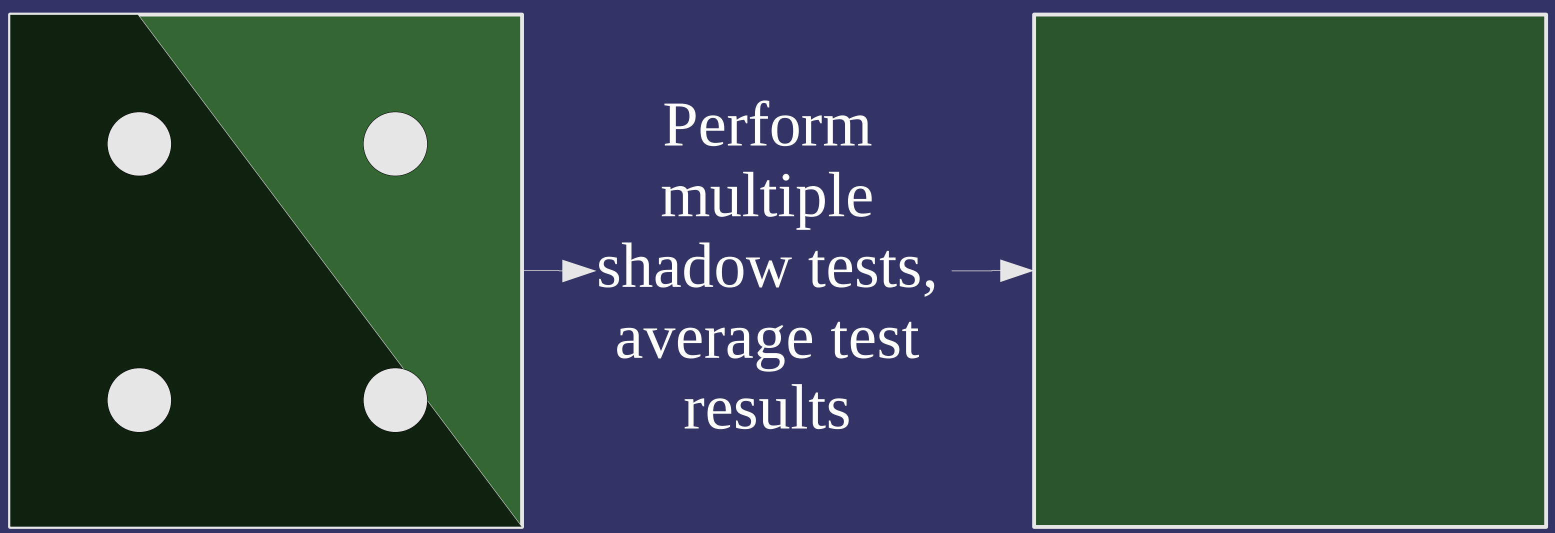
Shadow Map Texture Filtering

- Shadow test samples shadow map once per fragment
  - Results in two possible light levels: fully lit or fully shadowed



Shadow Map Texture Filtering

- Percentage closer filtering (PCF) reads multiple samples, performs one test per sample, averages test results
  - Results in n+1 possible light levels, where n is the number of samples



Shadow Map Texture Filtering

- Straight forward implementation in GLSL:

```
uniform vec2 bias;
uniform sampler2DShadow map;

void main()
{
    vec3 proj = coord.xyz / coord.w;
    vec3 p0 = proj – vec3((0.5 * bias.xy), 0.0);

    vec4 shadow = shadow2D(map, p0);
    shadow += shadow2D(map, p0 + vec3(bias.x, 0.0, 0.0));
    shadow += shadow2D(map, p0 + vec3(0.0, bias.y, 0.0));
    shadow += shadow2D(map, p0 + vec3(bias.x, bias.y, 0.0));

    shadow /= 4.0;

    ...
}
```

Percentage Closer Filtering

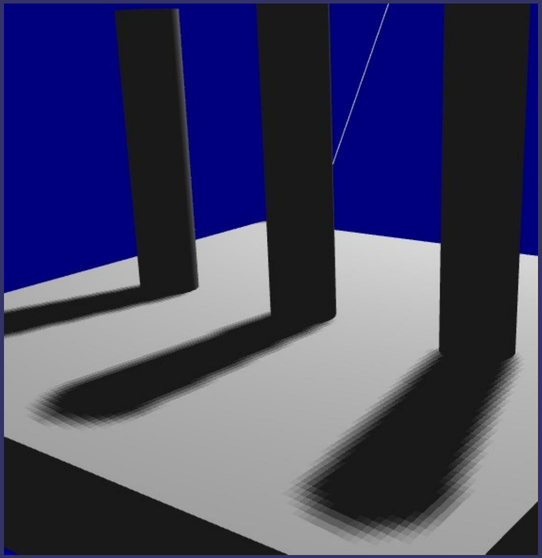
- The good news:
  - Improves quality
  - Larger filter kernels can be used to enable soft shadows
  - Some hardware can do 2x2 PCF nearly for free
    - Just enable `GL_LINEAR` filter on Nvidia hardware

Percentage Closer Filtering

- The good news:
  - Improves quality
  - Larger filter kernels can be used to enable soft shadows
  - Some hardware can do 2x2 PCF nearly for free
    - Just enable `GL_LINEAR` filter on Nvidia hardware
- The bad news:
  - Larger filter kernels are expensive
  - Grid-based sampling has artifacts

Grid-Based Sampling

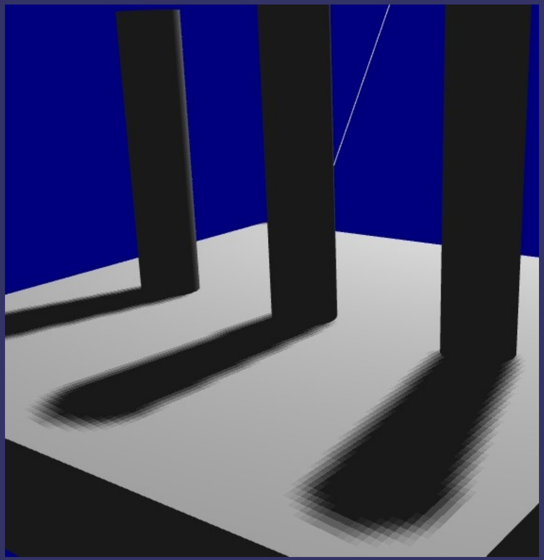
- Grid-based sampling artifacts have regular shape and are easily noticed by the eye

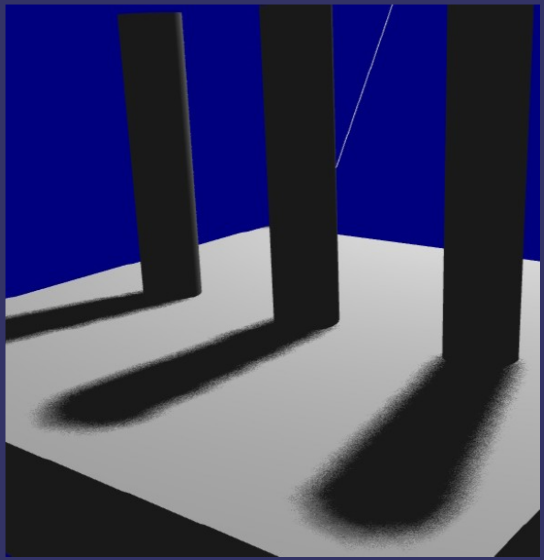


Images from http://ati.amd.com/developer/SIGGRAPH05/ShadingCourse_ATI.pdf

Grid-Based Sampling

- Grid-based sampling artifacts have regular shape and are easily noticed by the eye
- Irregular sample patterns are more easily accepted by the eye
  - Can even use fewer samples in the same size area

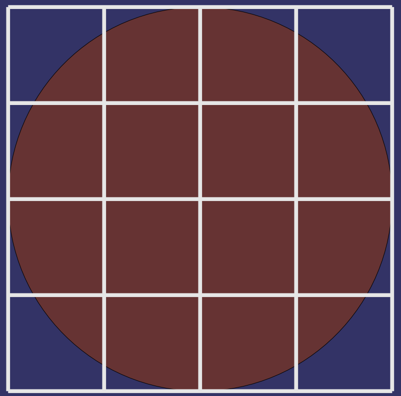




Images from http://ati.amd.com/developer/SIGGRAPH05/ShadingCourse_ATI.pdf

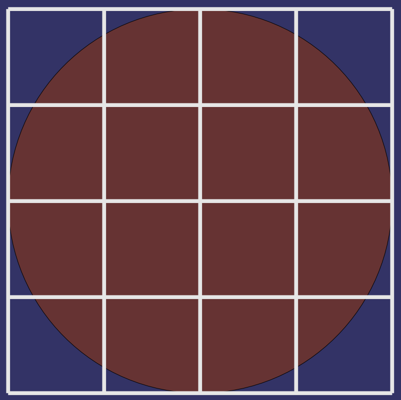
Irregular Sampling

- Select filter area

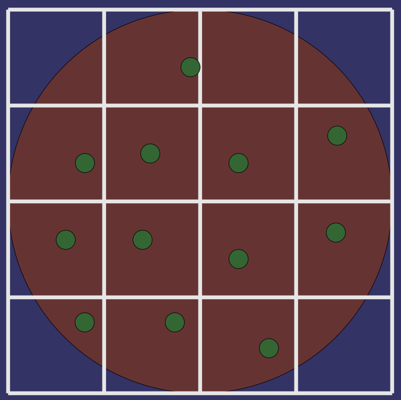


Irregular Sampling

- Select filter area



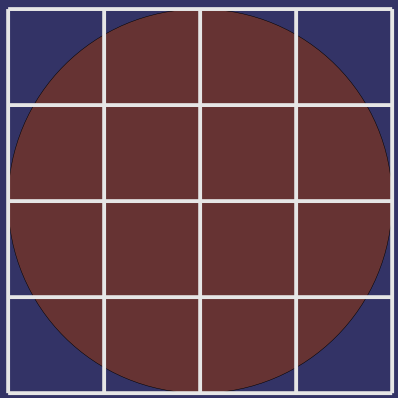
- Select random sample locations within area



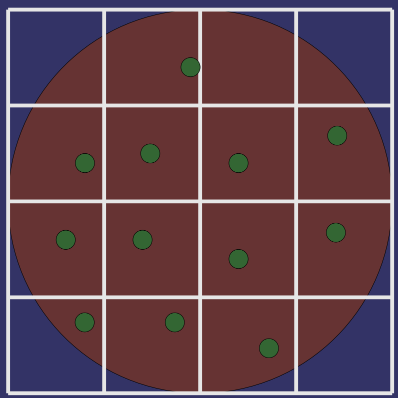
21-July-2009

Irregular Sampling

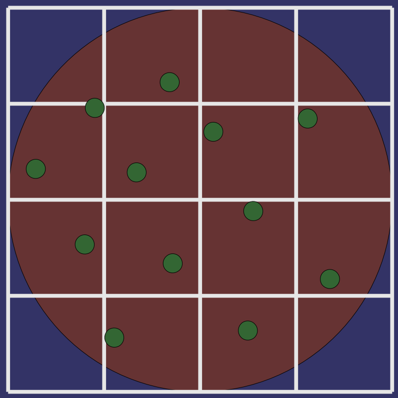
- Select filter area



- Select random sample locations within area



- Randomly rotate sample locations
  - Rotation based on screen location



Filter Cost

- 12 or 16 samples per fragment is expensive

Filter Cost

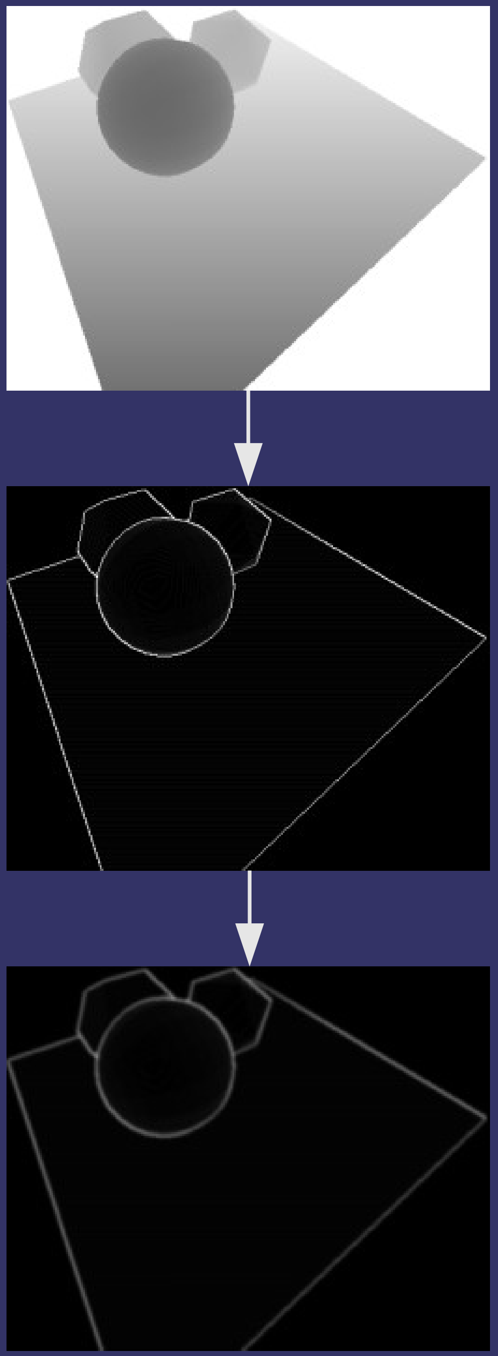
- 12 or 16 samples per fragment is expensive
  - In most of the final image, expensive sampling is unnecessary
  - Nyquist–Shannon sampling theorem tells us that areas with only low-frequency information need fewer samples than areas with high-frequency information

Filter Cost

- 12 or 16 samples per fragment is expensive
  - In most of the final image, expensive sampling is unnecessary
  - Nyquist–Shannon sampling theorem tells us that areas with only low-frequency information need fewer samples than areas with high-frequency information
    - The only high-frequency information is near the shadow boundaries!

Shadow Boundary Map

- Find boundaries in shadow map using edge detection filter
  - The edges in the map are the regions where the expensive filter should be applied
- Blur edge map using a blur kernel equal in size to the shadow map sample filter
  - This increases the area where the expensive filter will be applied and ensures that it will be applied everywhere that it needs to be



Shadow Boundary Map

- Use the shadow boundary map to determine whether to use one or many shadow samples

```
if (texture2DProj(boundary_map, proj) > 0.0) {
    shadow = pcf_shadow_filter(shadow_map, proj);
} else {
    shadow = shadow2DProj(shadow_map, proj)
}
```

  - On hardware that support dynamic flow control, this can be a *big* win
    - DFC is a required part of DX 9.0c Shader Model 3.0
      - Geforce6 and later
      - Radeon X1xxx (R500) and later
      - Intel GMA X3000 (G965) and later

References

Sander, P. and Isidoro, J. *Explicit Early-Z Culling and Dynamic Flow Control on Graphics Hardware*. ATI Corporation, 2005, accessed 20 April 2008; available from http://ati.amd.com/developer/techpapers.html

Next week...

- Advanced shadow map techniques
  - Quiz #1
  - Assignment #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 IBM 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.