Shadow Maps, part 2

- Agenda:
  - Assignments:
    - Hand in assignment #1
    - Discuss assignment #2...anyone started?
  - Finish basic shadow map techniques
  - Begin advanced shadow map techniques
  - Work on second programming assignment

Shadow map projections in GLSL

- Calculate texture coordinates that correspond to the object's position in *projected* light-space
  - In other words, the texture coordinate is the object's world space coordinate transformed by the light's modelview-projection matrix
  - There is also an offset to convert the resulting [-1, 1] to the correct [0, 1] range for texture sampling
    - This is the *B* matrix below

$$T_n = B P_{L_n} M_{L_n} M_{object} V$$

Shadow map projections in GLSL (cont.)

- In fixed-function, this is done using `EYE_LINEAR` texgen and planes that correspond to the rows of the $B P_{L_n} M_{L_n}$ matrix
  - `EYE_LINEAR` texgen computes $M_{object} V$
- We can replicate this *exactly* in GLSL
  - Do a matrix multiply of `gl_Vertex` with `gl_EyePlaneS[n]`, etc. as the rows
  - Could also compute the matrix and put it in a texture matrix or other uniform matrix

Light projection matrix

- The light's projection matrix is just like the camera's projection matrix
- The view frustum corresponds to the area covered by the spotlight
  - If the spotlight covers an angle of θ, the view plane covers $\pm\tan(\theta)$:

```
glFrustum(-tan(θ) * near, tan(θ) * near,
          -tan(θ) * near, tan(θ) * near,
          near, far);
```

Calculating near and far?

- How can we calculate the `near` and `far`?
  - Getting these values as “tight” as possible makes better use of available depth precision
- If the light is *outside* the camera's view frustum, set `far` to match the farthest part of the camera frustum that intersects the light's frustum
- Can't just set `near` to the nearest part of the camera frustum
  - Why?

Calculating near

- There may be objects between the light and the camera frustum that cast shadows on objects in the camera's view
  - One “easy” way is to use the distance of the object nearest the light

Shadow map problems

- Classic shadow maps of long, thin objects alias *horribly*
  - Since shadow maps are typically sampled with `GL_NEAREST`, aliasing is unavoidable
  - Can't use other modes because blending the shadow depth values is wrong
- Classic shadow maps also can't do soft shadows
- Omnidirectional point lights are hard

Percentage closer filtering

- Reeves created percentage closer filtering (PCF) as a method to antialias shadow maps
  - Perform multiple shadow comparison operations per sample
  - Blend the results of the comparisons
    - This is like `GL_LINEAR` blending, but the result of the comparison is blended instead of the raw texel values.
    - Nvidia does this in hardware when `GL_LINEAR` is used with a shadow map.

PCF (cont.)

- Basic PCF uses a fixed size filter kernel (usually 2x2)
- Fernando observed that as the size of the filter kernel increases the shadows become softer
  - Percentage closer soft shadows extends PCF by setting the kernel size based on the distance between the light and caster and the distance between the light and the receiver
  - Fernando's paper is one of the reading assignments this week

Omnidirectional lights

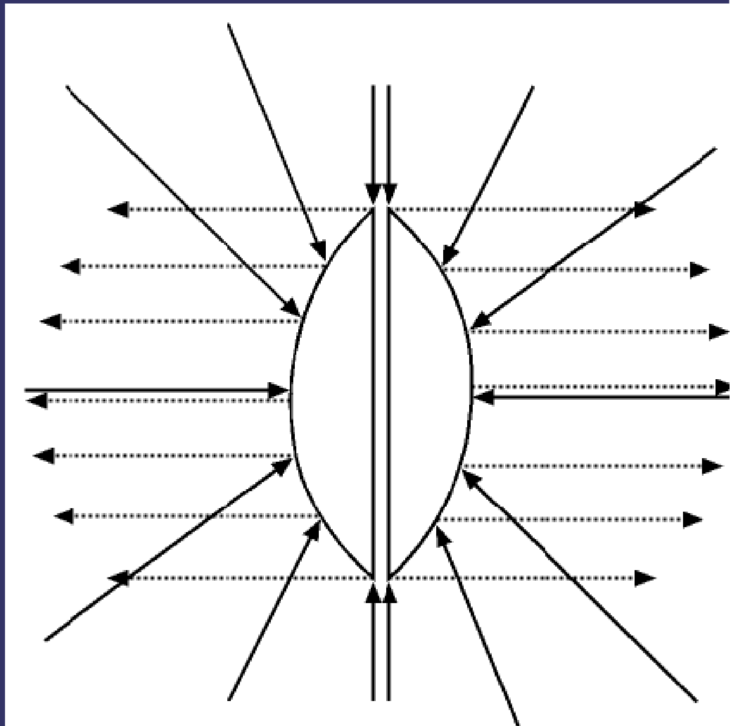
- Omnidirectional lights are hard
  - Can't just use a single map with a view frustum: θ is 90°, and tan(90°) is ∞
- Obvious technique is to render 6 views to the sides of a cubemap
  - Six passes to create the shadow map for *a single light???* Ouch!
  - Can slightly optimize this if the light is outside the view frustum, but might have upto 5 passes

Omnidirectional lights

- We really want a different environment map that requires fewer passes than a cubemap
  - Sphere maps (and related techniques) are right out because the edges of the sphere have a *lot* of area mapped onto them. Same problem as using sphere maps for environment mapping.

Paraboloid mapping

- Paraboloid mapping models a mirrored parabola instead of a sphere
  - Maps 180° into the map instead of 360°
  - Still compresses a lot of data into the edges, but not nearly as bad as a sphere map
  - Image from Brabec, et. al.



Paraboloid mapping (cont.)

- We can map the geometry onto the parabola in the vertex shader
  - This only maps the verticies to the parabola and uses linear interpolation between
  - If the geometry is sufficiently tessellated this is probably good enough
- This allows an omnidirectional light in *at most* 2 passes instead of 6

Questions?

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.