Shadow Volumes on GPUs

- Agenda:
  - Assignment #3
    - Discuss / ~~hand in~~
  - ~~Reading presentation~~ We'll do 2 next week
  - Quiz #3
  - Shadow volumes on GPUs
    - Generating the shadow volume
  - Lab time:
    - Work on assignment #3

Shadow volume geometry recap

- Two passes over object geometry are required:
  - Each edge that is shared by a front-facing polygon and a back-facing polygon, it is on the silhouette.
  - Project each edge on the silhouette away from the light to “infinity”. Create a new quad using these two edges. Add this quad to the shadow volume.
  - Add each front-facing polygon to the volume.
  - Project each back-facing polygon away from the light to infinity and add it

Shadow volume creation problems

- New volume must be created each time the object or the light move
- Time consuming and must be performed on the CPU
  - Re-upload data to the GPU *each frame!*
- Bad interactions with vertex shaders
- We'll see how to resolve these issues next week!

What really happens?

- Each edge either becomes a quad extended to infinity, or it becomes nothing
  - OpenGL treats a quad with two identical edges (i.e., points A, B, B, A) as nothing
- Can this be exploited so that shadow volume geometry can be created in a vertex shader?

Creating shadow volume geometry, take 2

- Augment the geometry with degenerate quads.
  - Each edge, (A, B), becomes a quad, (A, B, B, A).
  - Two quad points, (A, B), have normals equal to the surface normal of one of the shared polygons.
  - The other two quad points, (B, A), have normals equal to the surface normal of the other shared polygon.

Creating shadow volume geometry, take 2

- In the vertex shader:
  - If the normal of a point faces towards the light, transform the position normally.
  - If the normal of a point faces away from the light, transform the position and project it away from the light towards infinity.

Creating shadow volume geometry, take 2

- Results:
  - If all 4 points face toward the light, the quad remains degenerate and is not drawn.
  - If all 4 points face away from the light, the quad is projected to infinity, remains degenerate, and is not drawn.
  - If the edge is a silhouette, one edge of the quad remains in place, and the other is projected to infinity. Exactly what is needed!

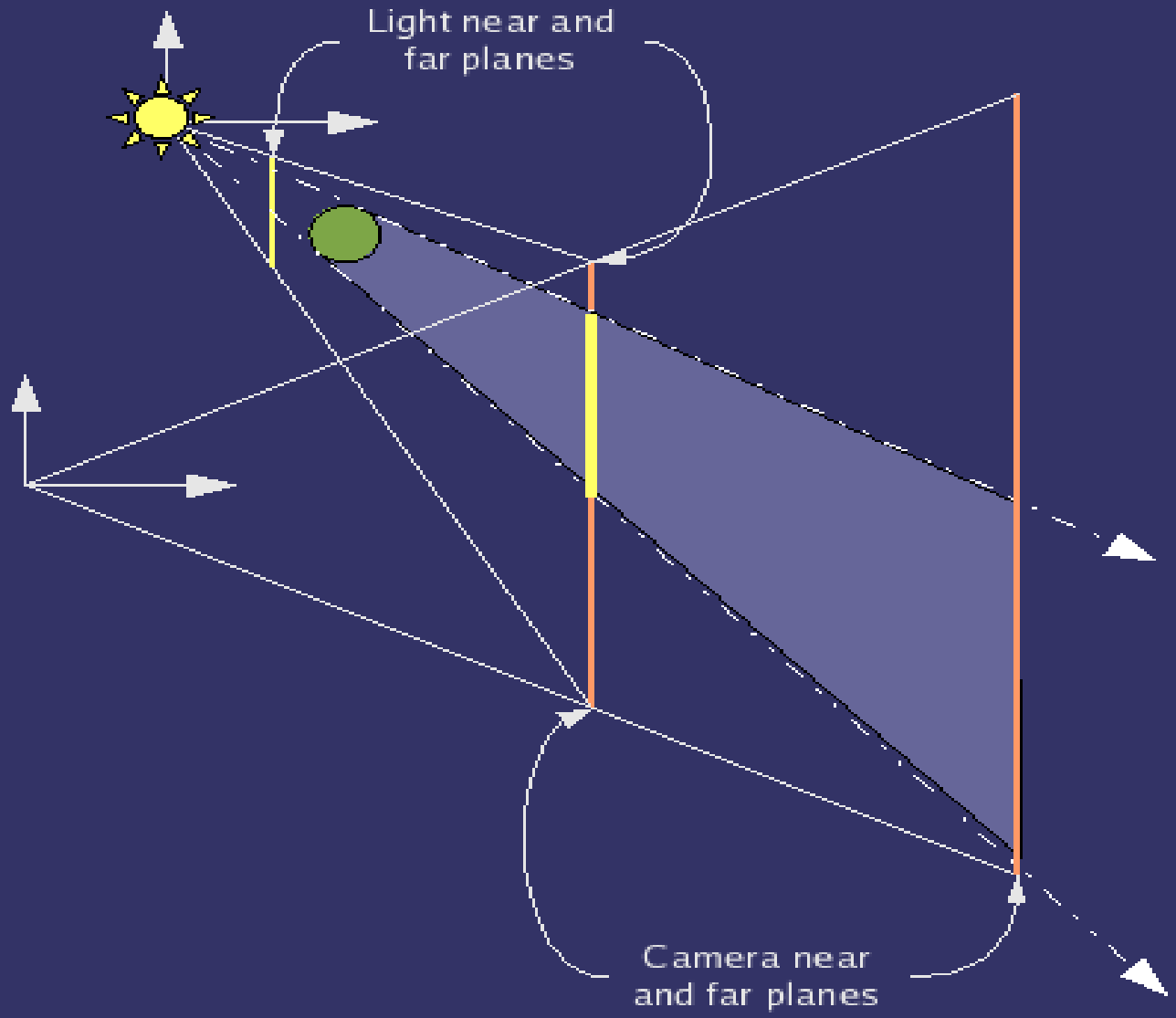
What about volume caps?

- Z-pass still has problems when the light and occluders are outside the camera frustum
  - Shadow volume geometry that is clipped by the near plane is the source of all the z-pass problems
- The Nvidia paper assigned for reading this week covered the various problems with generating cap geometry
  - The authors punt on the issue and use z-fail.

Shadow volume projection

- Ultimately, this geometry “just” needs to be projected from the light onto the near plane
- We can do just that!
  1. Position eye at light
  2. Orient view frustum parallel (or antiparallel) to camera frustum
  3. Set far-plane to match camera's near-plane
  4. Draw front facing geometry into stencil buffer
  5. Continue with regular z-pass

Shadow volume projection (cont.)



Shadow volume projection (cont.)

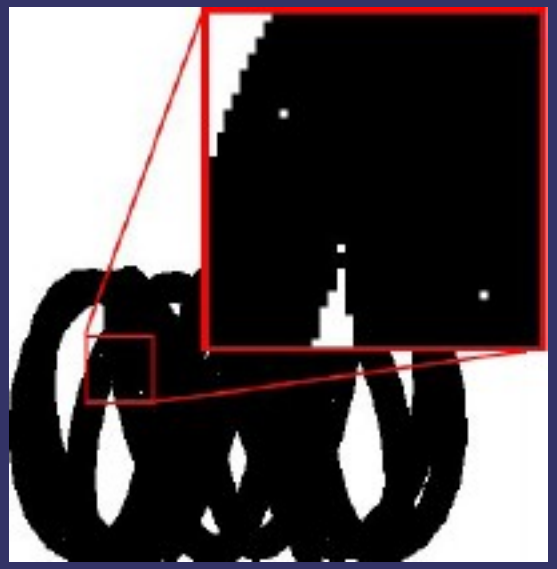
- The matrix to project from light onto the camera's near plane is:

$$P_l = \begin{pmatrix} \frac{2\alpha l_{far}}{c_{width}} & 0 & -2\frac{\Delta_x}{c_{width}} & 0 \\ 0 & \frac{2 l_{far}}{c_{height}} & -2\frac{\Delta_y}{c_{height}} & 0 \\ 0 & 0 & \frac{l_{near}+l_{far}}{l_{near}-l_{far}} & \frac{2 l_{near}+l_{far}}{l_{near}-l_{far}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

  - Δ is vector from camera to light
  - α is 1 if light and camera are on same side of near plane, -1 otherwise

But there's still a (small) problem!

- Because geometry is draw with different projections, slight cracks can appear!



  - We'll talk about the solution *next week*...

Extensions to optimize shadows

- Several useful extensions exist:
  - Two-sided stencil
  - Depth clamping
  - Depth bounds testing

Two-sided stencil

- Exposed three ways:
  - `GL_EXT_stencil_two_side`
  - `GL_ATI_separate_stencil`
  - OpenGL 2.1
- Functionally similar, but different interfaces.
  - `GL_ATI_separate_stencil` is missing some functionality

GL_EXT_stencil_two_side

- Adds a single new entry-point `glActiveStencilFaceEXT`
  - Conceptually similar to `glActiveTexture`

GL_EXT_stencil_two_side

```
glDisable(GL_CULL_FACE);
glEnable(GL_STENCIL_TEST);
glEnable(GL_STENCIL_TEST_TWO_SIDE_EXT);

glActiveStencilFaceEXT(GL_BACK);
glStencilOp(GL_KEEP, GL_KEEP, GL_DECR_WRAP_EXT);
glStencilMask(~0);
glStencilFunc(GL_ALWAYS, 0, ~0);

glActiveStencilFaceEXT(GL_FRONT);
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR_WRAP_EXT);
glStencilMask(~0);
glStencilFunc(GL_ALWAYS, 0, ~0);
```

GL_ATI_separate_stencil

- Adds two new entry-points, `glStencilOpSeparateATI` and `glStencilOpSeparateATI`.
  - Doesn't support separate reference values or masks.

GL_ATI_separate_stencil

```
glDisable(GL_CULL_FACE);
glEnable(GL_STENCIL_TEST);

glStencilOpSeparateATI(GL_BACK, GL_KEEP, GL_KEEP,
                       GL_DECR_WRAP_EXT);
glStencilOpSeparateATI(GL_FRONT, GL_KEEP, GL_KEEP,
                       GL_INCR_WRAP_EXT);
glStencilFuncSeparateATI(GL_ALWAYS, GL_ALWAYS, 0, ~0);
```

OpenGL 2.1

- Adds two new entry-points, `glStencilOpSeparate` and `glStencilOpSeparate`.
- Hybrid approach that provides full functionality.
  - Compromise FTW. :(

OpenGL 2.1

```
glDisable(GL_CULL_FACE);
glEnable(GL_STENCIL_TEST);

glStencilOpSeparate(GL_BACK, GL_KEEP, GL_KEEP,
                    GL_DECR_WRAP_EXT);
glStencilOpSeparate(GL_FRONT, GL_KEEP, GL_KEEP,
                    GL_INCR_WRAP_EXT);


// Could do as single call w/ GL_FRONT_AND_BACK.

glStencilFuncSeparate(GL_FRONT, GL_ALWAYS, 0, ~0);
glStencilFuncSeparate(GL_BACK, GL_ALWAYS, 0, ~0);
```

Depth clamping

- Caused fragments that would be clipped by the near or far plane to be rendered with a depth of 0.0 or 1.0.
- Exposed via `GL_NV_depth_clamp` since Geforce3.
  - Part of DX10 and (likely) next version of OpenGL.
- Useful for shadow volumes
  - Mostly for z-fail. Eliminates the need for volume end capping.

Depth bounds testing

- Add extra per-fragment test before alpha test.
- Discards fragment if the existing depth value is outside a predefined range.
- If the Z range of an attenuated light can be calculated, fillrate can be reduced by skipping stencil updates outside it's range.
  - Scissor test can be used in X and Y.
- Exposed via `GL_EXT_depth_bounds_test` since Geforce FX 5700.

Depth bounds testing

```
calculate_light_screen_space_volume(light,
                                    &x_min, &x_max,
                                    &y_min, &y_max,
                                    &z_min, &z_max);

glEnable(GL_DEPTH_BOUNDS_TEST_EXT);
glEnable(GL_SCISSOR_TEST);

glDepthBoundsEXT(z_min, z_max);
glScissor(x_min, y_min, x_max - x_min, y_max - y_min);

do_shadows(light, objects);
```

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.