VGP353 – Week 6

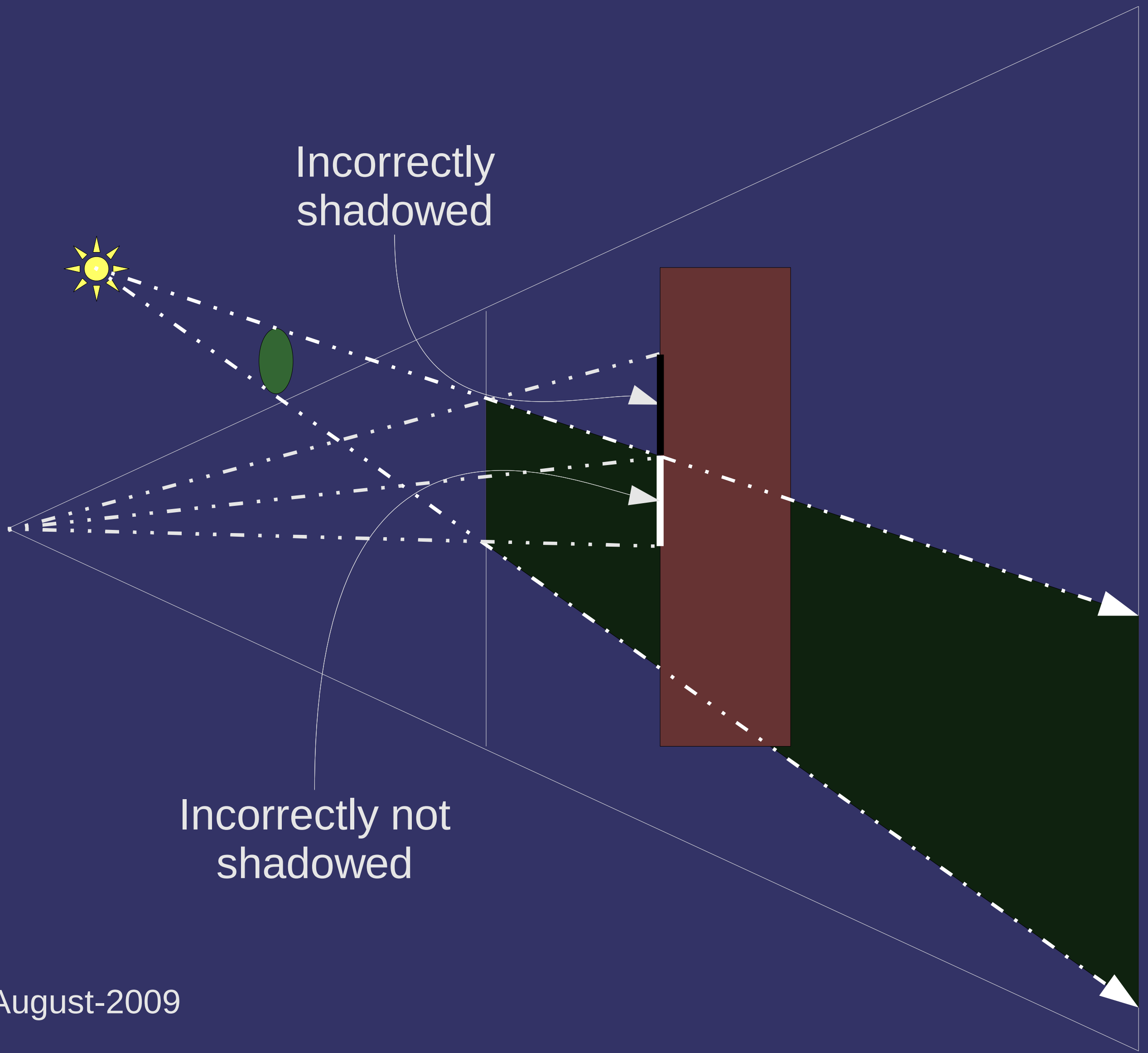
- Agenda:
  - Quiz #2
  - Fixing z-pass and z-fail with ZP+
  - Hardware based optimizations:
    - Depth clamping
    - Depth bounds testing

25-August-2009

Z-pass Problems

- Z-pass has problems when the light and occluders are outside the view frustum
  - This *includes* the case where the camera is inside a shadow volume
  - Shadow volume geometry that is clipped by the near plane is the source of all the z-pass problems
- Partially solved by generating front-cap geometry
  - Generating this geometry is hard
  - This difficulty led to the invention of z-fail

Z-pass Problems



25-August-2009

Z-pass Problems

- At a high level, what *is* the front-cap geometry?

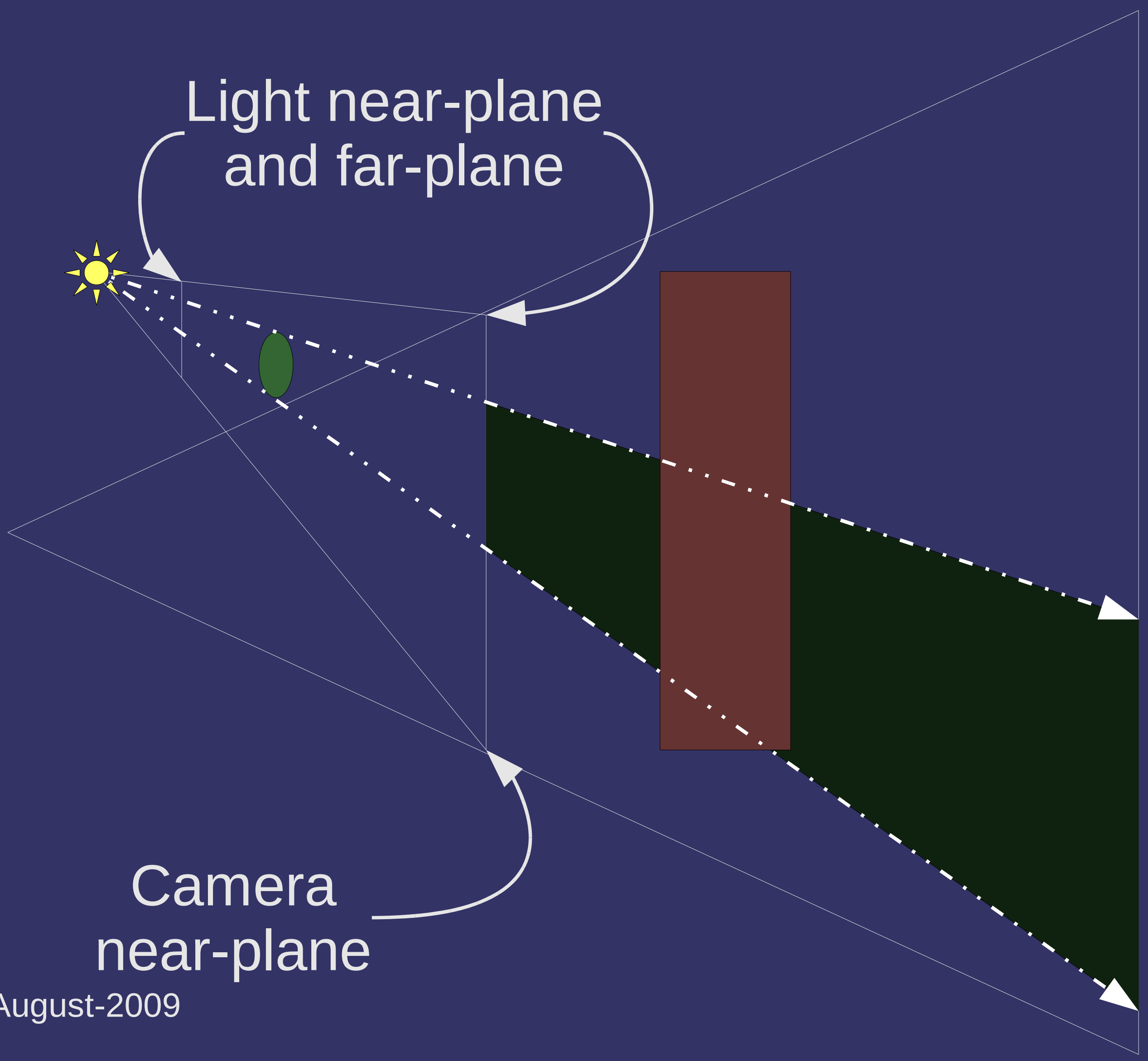
Z-pass Problems

- At a high level, what *is* the front-cap geometry?
  - Front-facing (w.r.t. the light) occluder geometry projected onto the camera's near-plane
  - Why not *do* just that: project front-facing occluder geometry on the the camera's near-plane

Z-pass Problems

- At a high level, what *is* the front-cap geometry?
  - Front-facing (w.r.t. the light) occluder geometry projected onto the camera's near-plane
  - Why not *do* just that: project front-facing occluder geometry on the the camera's near-plane
- ZP+ Algorithm:
  1. Position eye at light
  2. Orient view frustum parallel (or anti-parallel) to the camera frustum
  3. Set far-plane to match the camera's near-plane
  4. Draw front facing geometry into stencil buffer

ZP+



25-August-2009

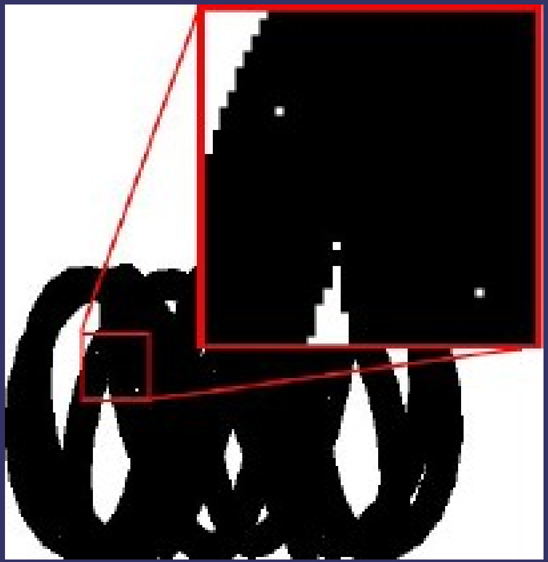
ZP+

- Projection matrix is:

$$P_l = \begin{pmatrix} \frac{2\alpha f}{c_{width}} & 0 & -2\frac{\Delta_x}{c_{width}} & 0 \\ 0 & \frac{2f}{c_{height}} & -2\frac{\Delta_y}{c_{height}} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

ZP+

- Since geometry is drawn with different projections, rounding errors can cause slight cracks to appear:



  - Not a significant problem in practice
  - Can be solved, see paper for details

References

Hornus, Samuel; Hoberock, Jared; Lefebvre, Sylvain; Hart, John C., "ZP+: Correct Z-Pass Stencil Shadows." In *Proceedings of ACM Symposium on Interactive 3D Graphics and Games*. ACM Press, April 2005. http://artis.imag.fr/Publications/2005/HHLH05/

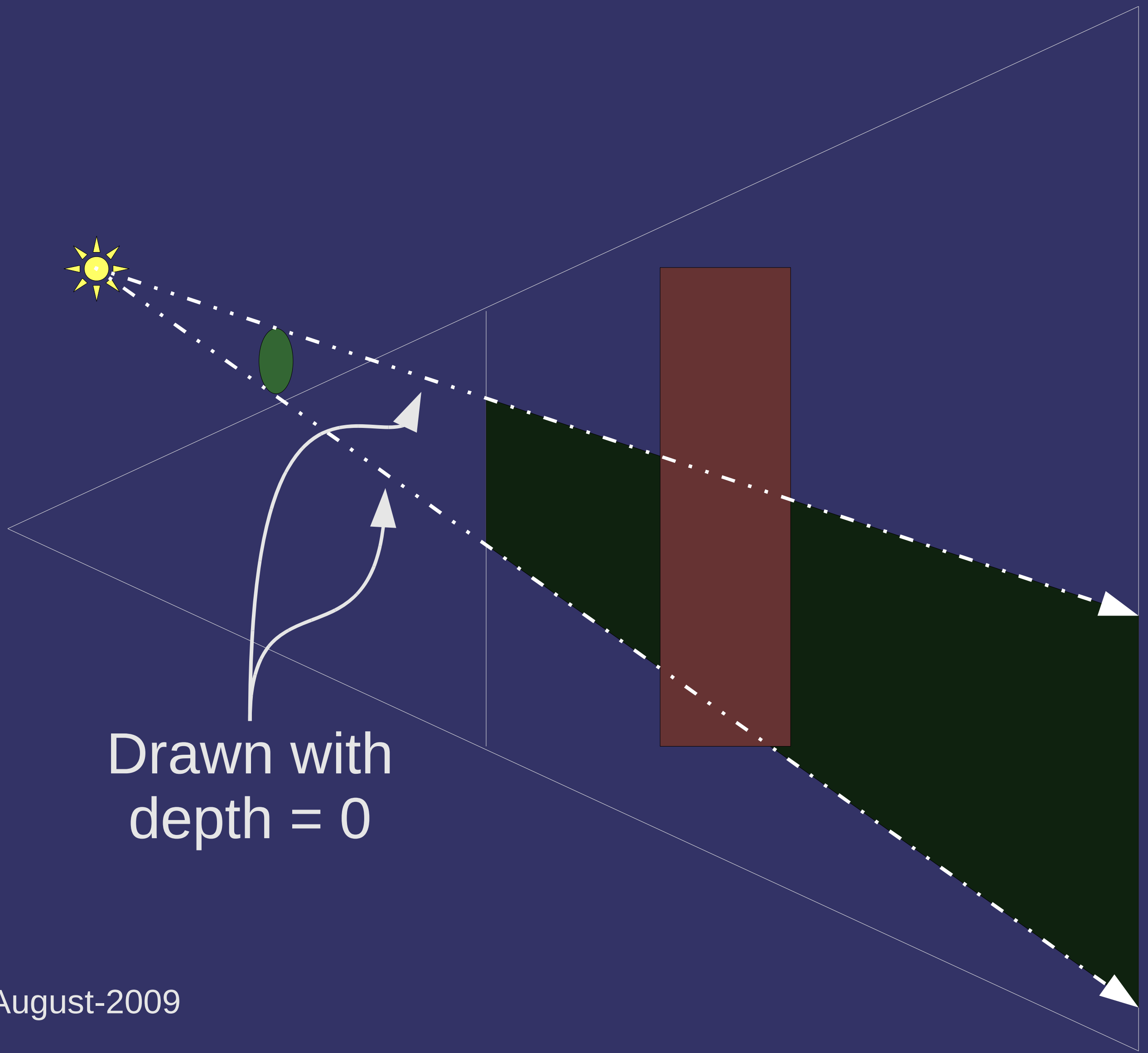
Hardware Optimizations

- Several hardware features exist to help accelerate shadow volume rendering
  - Depth clamping
  - Scissor testing
  - Depth bounds testing

Depth Clamp

- Fragments with interpolated depth values less than 0.0 or greater than 1.0 get a depth value clamped to [0, 1]
  - These are the fragments that would be clipped by the near- or far-plane
  - Eliminates need for front- and back-caps on shadow volumes
    - See ZP+ paper for more details
- Part of OpenGL 3.2
  - Also `GL_ARB_depth_clamp` and `GL_NV_depth_clamp`

Depth Clamp



Scissor Testing

- Spot lights only affect some areas of the screen
  - We end up drawing shadow volumes even where there is no light to be shadowed!
- Use scissor test to eliminate drawing of useless shadow volumes
  - Calculate x/y region of the *window* where a light can be seen
  - Set scissor rectangle to just this region
  - Fragments outside the region will be clipped

Depth Bounds Testing

- Extra per-fragment test before stencil test
  - Discards fragment if the *existing* depth value is outside a predefined range
  - Acts like a scissor test for depth
- Part of OpenGL 3.2
  - Also `GL_EXT_depth_bounds_test`

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);
```

Optimizing Shadow Volumes

- We've reduced the fill-rate a *lot* but we still...
  - Render a lot of volumes that don't produce visible shadows
  - Render a lot of volumes that do produce visible shadows in areas where they don't produce shadows
  - Render a lot of volumes from casters that are themselves completely in shadow

Optimizing Shadow Volumes

- Improve fill-rate usage two ways:
  - Cull volumes from casters that cast shadows not visible to the eye
  - Clamp shadow volume to the regions containing possible receivers

Shadow Volume Culling

- Compute two sets of objects:
  - Potential shadow receivers (PSR): Objects that may be visible to the camera
  - Potential shadow casters (PSC): Objects that may be visible to the light
- Use occlusion queries:
  - Render the scene once from the view of the light
  - Disable depth writes
  - Render object bounding boxes with occlusion queries
    - BBs with non-occluded pixels represent potentially visible objects

Shadow Volume Clamping

- Two steps:
  - Calculate continuously occupied intervals in object space
  - Reduce to discrete intervals in image space

Continuous Clamping

- From the point-of-view of the light:
  - Project each AABB into the lights near plane
    - The projections are squares
  - Determine which projections overlap
    - For each caster-receiver overlap, determine the depth interval occupied
    - The paper describes the occupied interval calculation in more detail
  - This is performed entirely on the CPU

Discrete Clamping

- From the point-of-view of the light:
  - Slice the view frustum into segments using “similarly oriented” planes
    - Planes roughly parallel to the light's near plane that pass through the camera are a good choice
  - Render slices back-to-front
    - The borders of the slice are clip planes
    - Project the caster onto the far plane
    - Render objects using occlusion query
    - If no pixels pass, the slice is empty
  - This is performed on the GPU

References

Lloyd, B., Wendt, J., Govindaraju, N., and Manocha, D. 2004. CC Shadow Volumes. In *ACM SIGGRAPH 2004 Sketches* (Los Angeles, California, August 8 - 12, 2004). R. Barzel, Ed. SIGGRAPH '04. ACM, New York, NY, 146. http://gamma.cs.unc.edu/ccsv/

Next week...

- Quiz #3
- Ambient occlusion
- Read:

Iones, A., Krupkin, A., Sbert, M., and Zhukov, S. 2003. Fast, Realistic Lighting for Video Games. *IEEE Computer Graphics and Applications*. 23, 3 (May. 2003), 54–64. http://ima.udg.edu/iiia/GGG/UsersDocs/mateu/obscurances.pdf

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.