VGP353 – Week 5

- Agenda:
  - Quiz #2
  - More shadow maps:
    - Quantifying shadow map aliasing
    - Reducing shadow map aliasing
      - Perspective shadow maps (PSMs)
      - Parallel-split shadow maps (PSSMs)
      - Resolution matched shadow maps (RMSMs)

Shadow Map Aliasing

- A shadow map texel represents an area $d_s \times d_s$
  - d_s is the reciprocal of the shadow map resolution
    - As shadow map resolution increases, d_s decreases
  - As r_s increases, each texel covers more of the surface
  - The projected size of a surface at distance r_s is approximately:

$$\frac{d_s r_s}{N \cdot L}$$

Shadow Map Aliasing

- An image pixel represents an area $d_i \times d_i$
  - d_i is the reciprocal of the image resolution
    - As image resolution increases, d_i decreases
  - As r_i increases, each pixel covers more of the surface
  - The projected size of a surface at distance r_i is approximately:

$$\frac{d_i r_i}{N \cdot V}$$

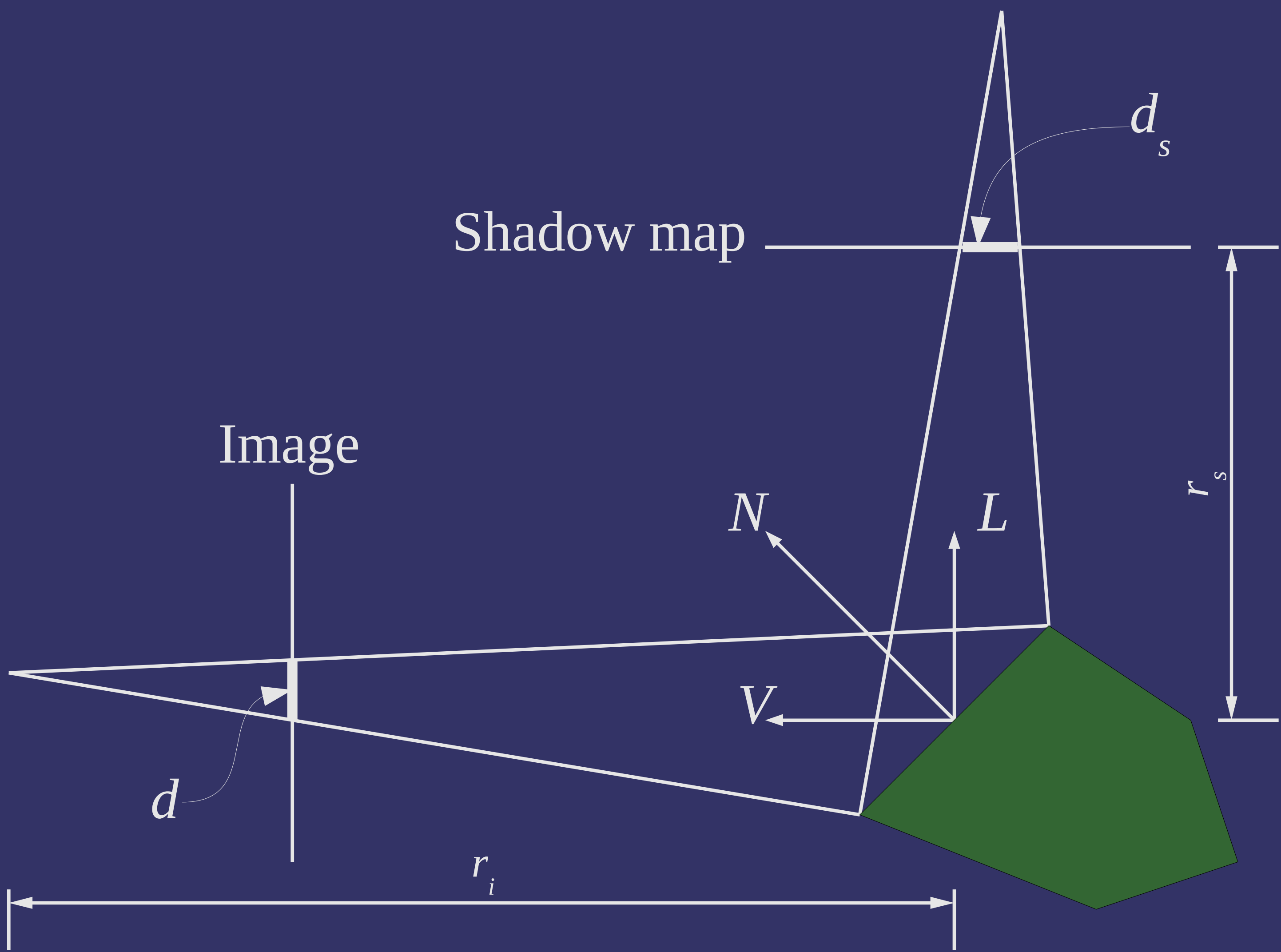
Shadow Map Aliasing

- The size of the projection of the shadow texel in the final image is:

$$d = d_s \frac{r_s}{r_i} \frac{N \cdot V}{N \cdot L}$$

  - Aliasing occurs when $d > d_i$

Shadow Map Aliasing



Shadow Map Aliasing

- The size of the projection of the shadow texel in the final image is:

$$d = d_s \frac{r_s}{r_i} \frac{N \cdot V}{N \cdot L}$$

  - Aliasing occurs when $d > d_i$
  - Matches intuition:
    - If the shadow area is small in the shadow map but large in the final image, there will be aliasing.

Shadow Map Aliasing

$$d_s \frac{r_s}{r_i} \frac{N \cdot V}{N \cdot L}$$

Large when light rays are nearly tangent to surface geometry, but surface geometry faces towards the viewer

- This is called *projection aliasing*
- Dependent on orientation of scene geometry
- Can change even when light and viewer are stationary
- Difficult to fix!

Shadow Map Aliasing

$$d_s \frac{r_s}{r_i} \frac{N \cdot V}{N \cdot L}$$

Occurs when the view is close to individual texels of the shadow map

- This is called *perspective aliasing*
- Occurs if the shadow map is too small (i.e., d_s is large)
- Can only increase shadow map size so much!
- Also occurs if $r_s \gg r_i$

Large when light rays are nearly tangent to surface geometry, but surface geometry faces towards the viewer

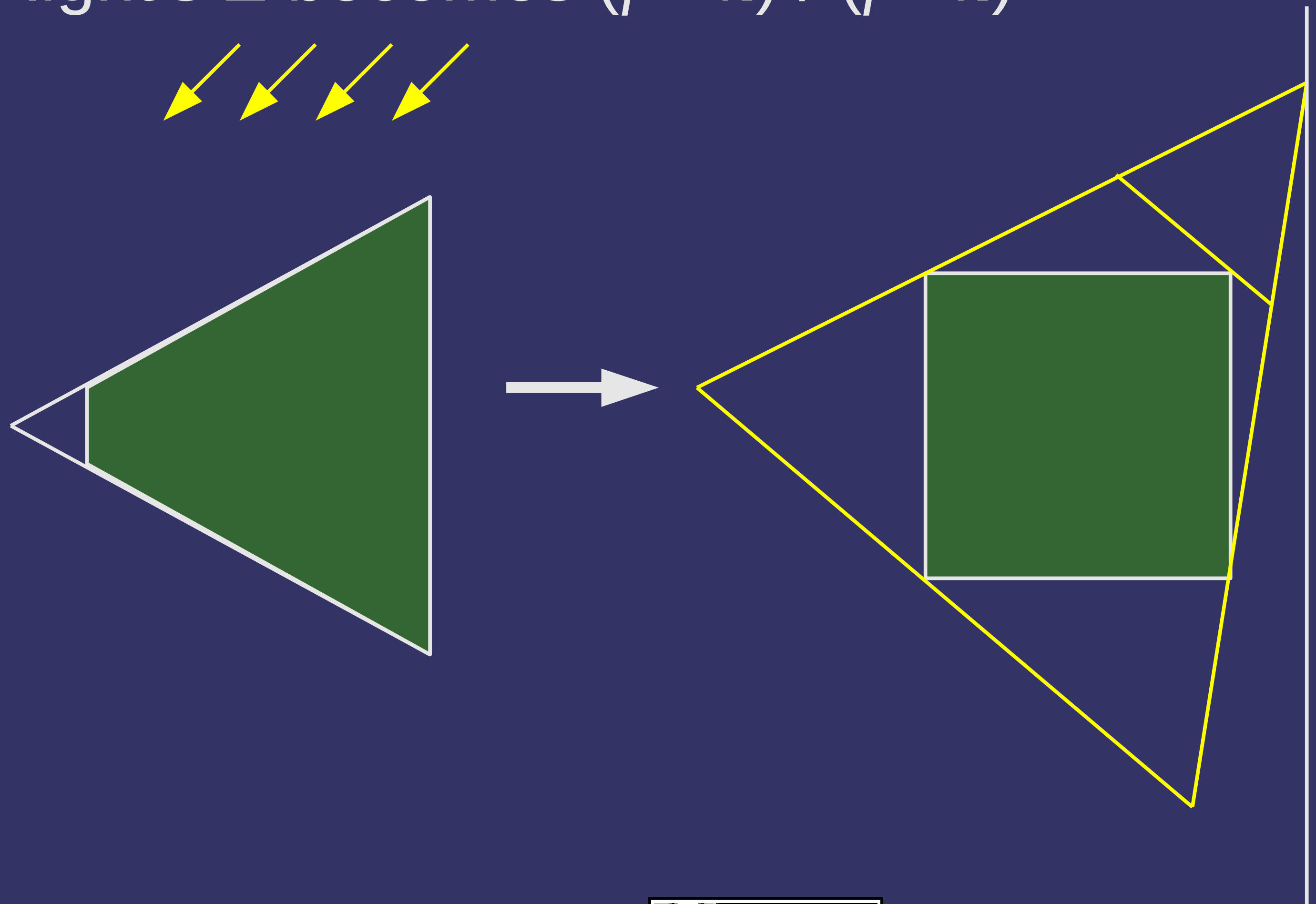
- This is called *projection aliasing*
- Dependent on orientation of scene geometry
- Can change even when light and viewer are stationary
- Difficult to fix!

Perspective Shadow Maps

- If the problem stems from the relationship between the camera frustum and light frustum, then the solution make take both frusta into account
  - Perform shadow map calculations in post-projection camera space *instead of* world space
  - The projection remaps the frustum volume to a cube, this cube is then sampled to create the shadow map
    - Applying this to the world before applying the light's view effectively changes the “shape” of the light

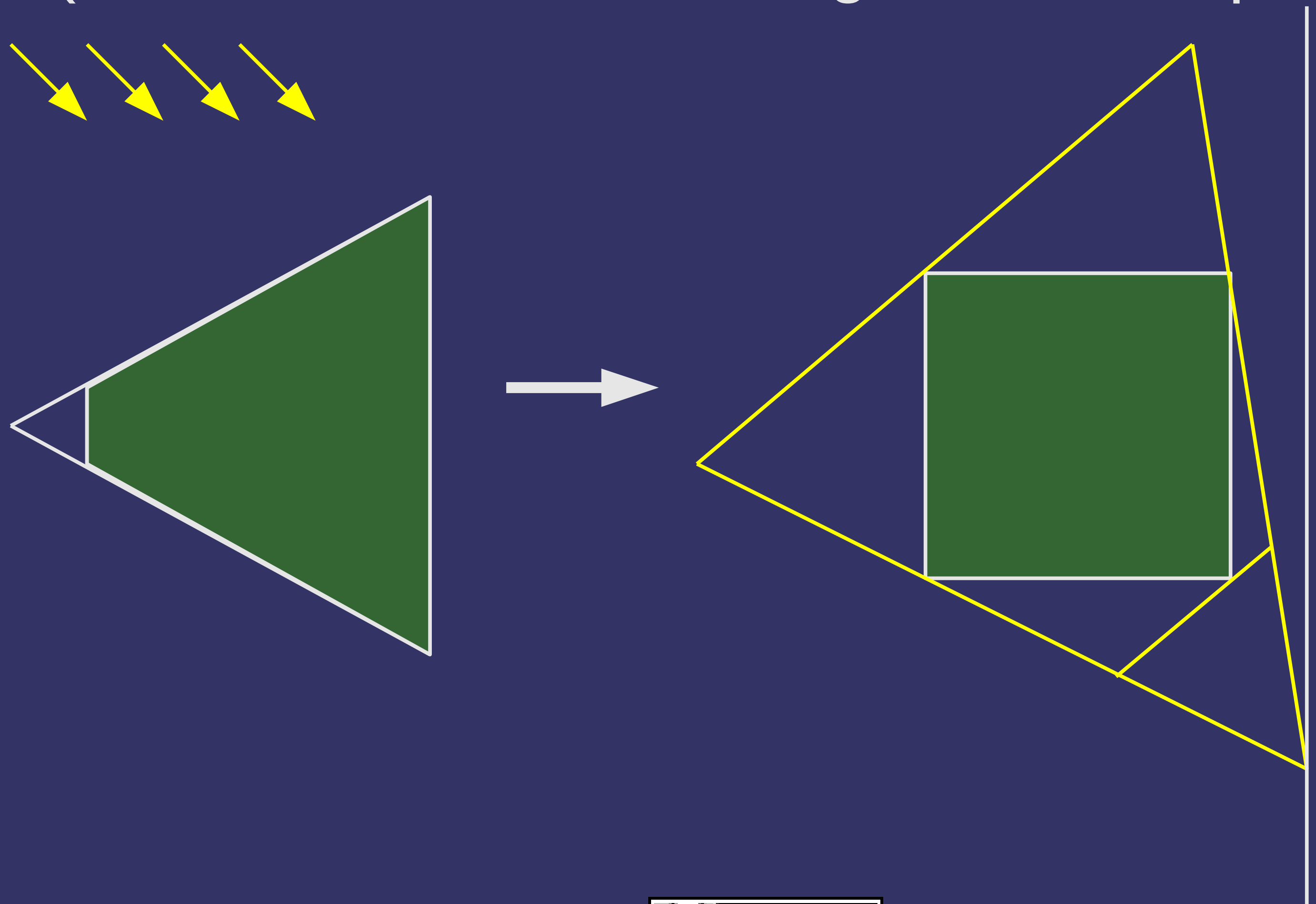
Perspective Shadow Maps

- Directional lights become point lights "on the infinity plane"
  - The light's Z becomes $(f + n) / (f - n)$



Perspective Shadow Maps

- Directional front-lights become inverted
  - Reverse the order of the usual depth and shadow tests (i.e., less-than becomes greater-or-equal)

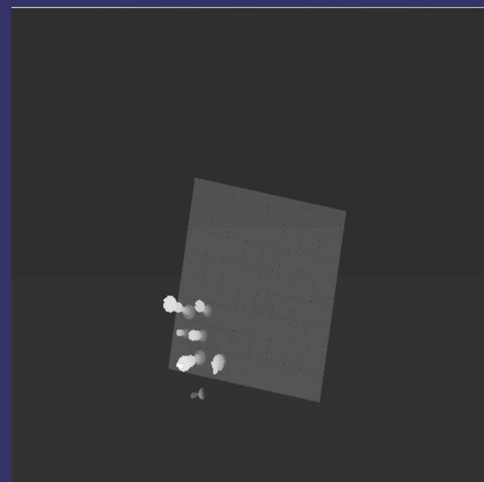


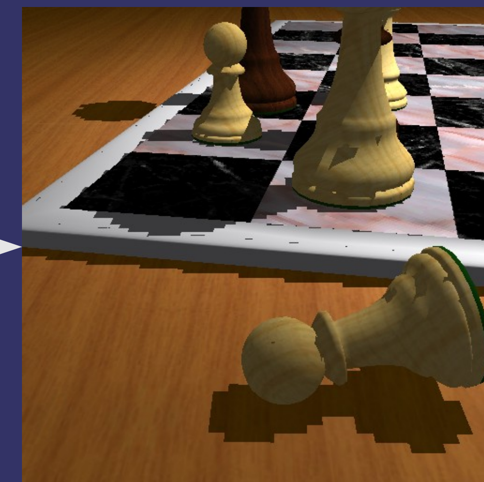
Perspective Shadow Maps

- Directional lights have other quirks
  - The more parallel the light and view direction, the lower the quality
    - A directional light pointing in the exact opposite direction of the view direction degrades back to the classic shadow map case
  - Casters behind the viewer (i.e., negative Z) are inverted and projected past the far plane
    - Several methods to handle this special case
  - Point lights have similar issues

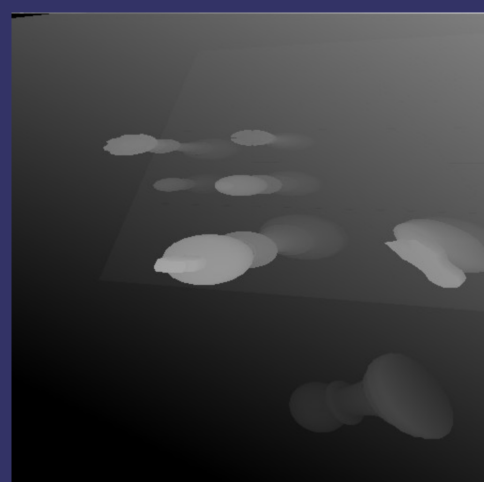
Perspective Shadow Map

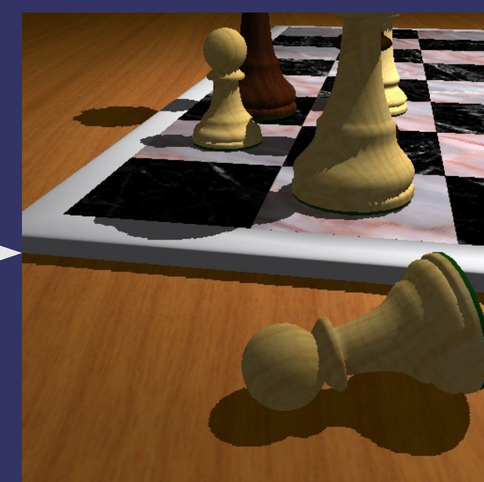
Standard
shadow map





Perspective
shadow map





Images from http://www-sop.inria.fr/reves/publications/data/2002/SD02/index.gb.html

Perspective Shadow Maps

- Advantages:
  - Improves quality for many common cases
  - Easy to implement for directional light sources
- Disadvantages:
  - PSMs are view dependent
    - Must be regenerated when the camera moves
  - Dual perspective transforms exaggerate shadow acne
  - As the viewer moves, the quality of the shadow map changes...even if the rest of the scene is static
    - For *most* games, this is the deal breaker

References

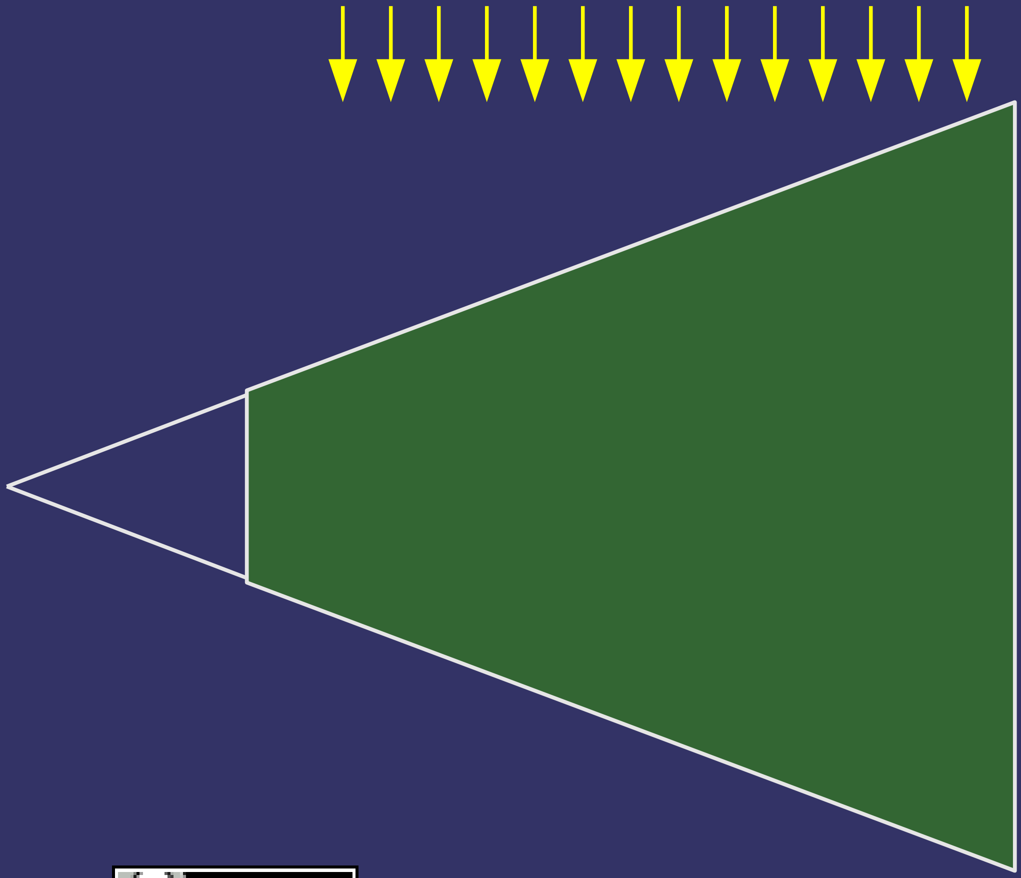
Stamminger, M. and Drettakis, G. 2002. Perspective shadow maps. In *Proceedings of the 29th Annual Conference on Computer Graphics and interactive Techniques* (San Antonio, Texas, July 23 - 26, 2002). SIGGRAPH '02. ACM, New York, NY, 557-562. http://www-sop.inria.fr/reves/publications/data/2002/SD02/index.gb.html

Perspective Shadow Maps

- Some significant problems:
  - Shadow map *quality* is view-dependent
  - Several special cases that must be handled depending on light direction / position
  - Difficulties handling shadow casters behind the camera
- Introduced some good ideas:
  - Re-parameterizing the scene based on the camera / light frusta
  - Quantitatively determining when aliasing will occur

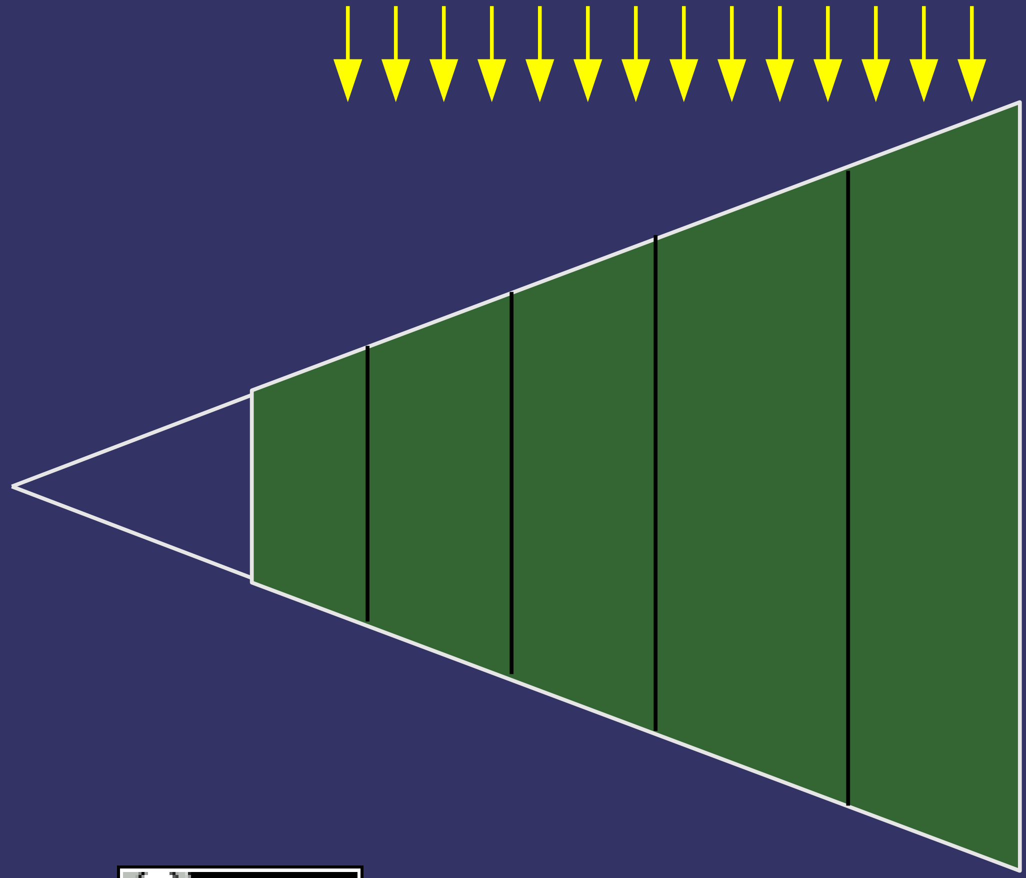
Parallel-Split Shadow Maps

- PSSMs solve most of these problems



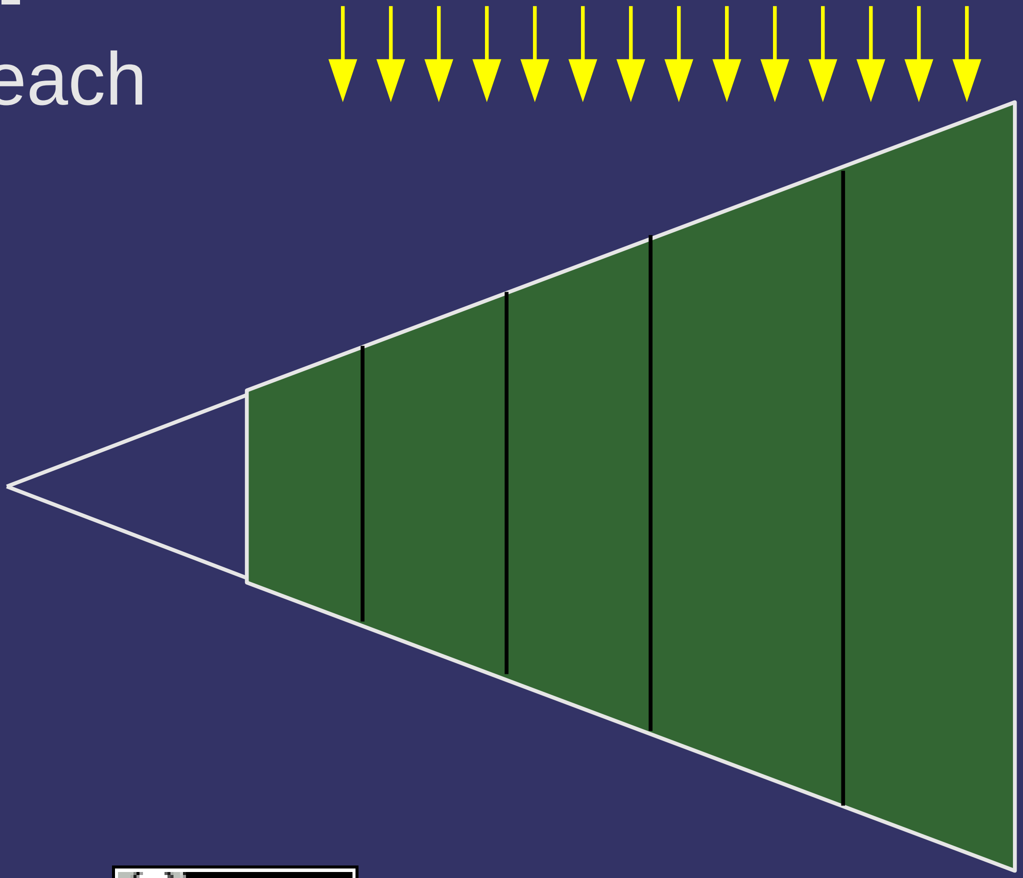
Parallel-Split Shadow Maps

- PSSMs solve most of these problems
  - Split view frustum into m parts with planes parallel to the near / far plane



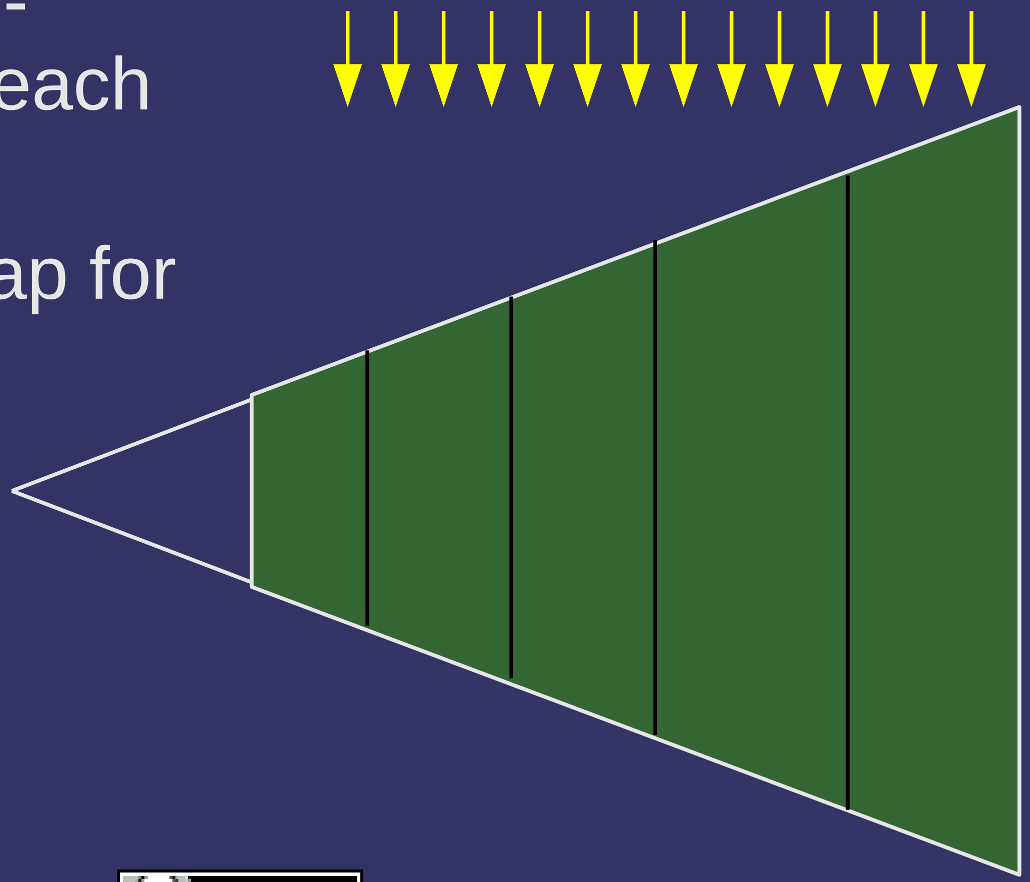
Parallel-Split Shadow Maps

- PSSMs solve most of these problems
  - Split view frustum into m parts with planes parallel to the near / far plane
  - Calculate light's view-projection matrix for each split region



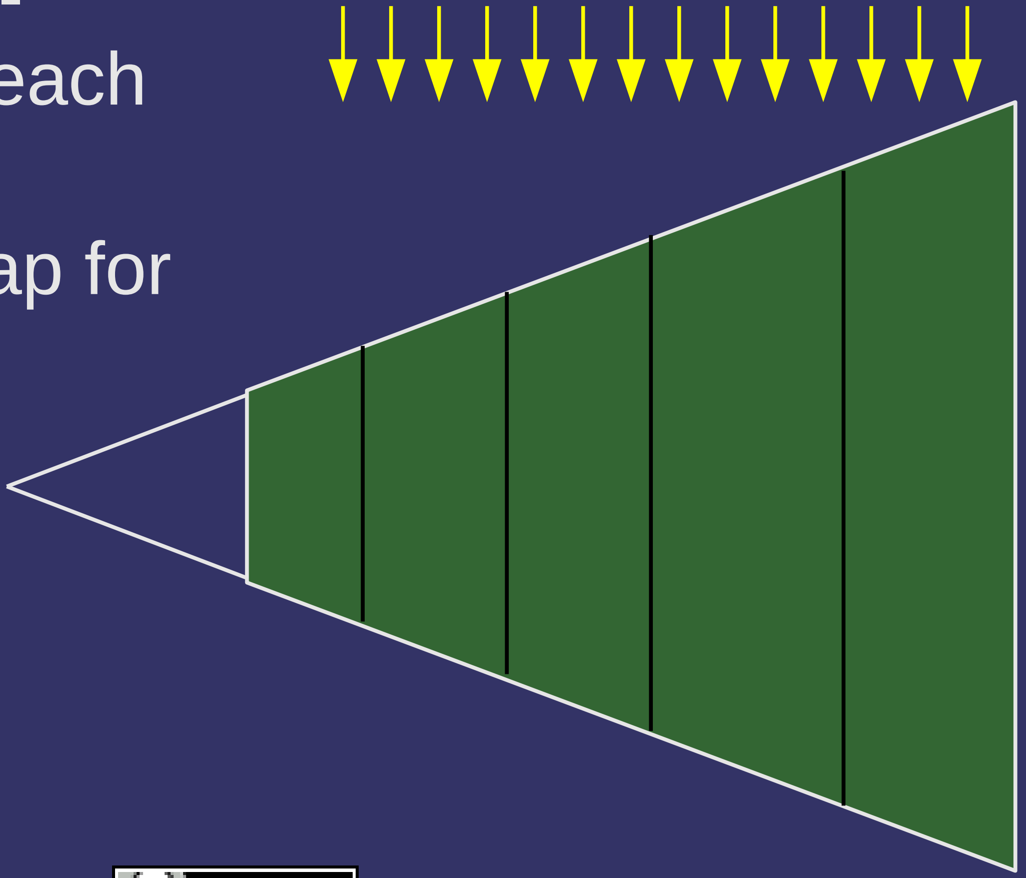
Parallel-Split Shadow Maps

- PSSMs solve most of these problems
  - Split view frustum into m parts with planes parallel to the near / far plane
  - Calculate light's view-projection matrix for each split region
  - Generate shadow map for each split region



Parallel-Split Shadow Maps

- PSSMs solve most of these problems
  - Split view frustum into m parts with planes parallel to the near / far plane
  - Calculate light's view-projection matrix for each split region
  - Generate shadow map for each split region
  - Apply shadow maps to scene



Parallel-Split Shadow Maps

- Aliasing occurs when $d > d_i$

$$d = d_s \frac{r_s}{r_i} \frac{N \cdot V}{N \cdot L}$$

  – Rename r_i as z, and call dz the change in z relative to one unit in ds

$$d = \frac{dz}{z\,ds} \frac{N \cdot V}{N \cdot L}$$

  – Ignoring perspective aliasing, this means that we want dz / zds to be constant over the entire view
    – Call this constant ρ

Parallel-Split Shadow Maps

- Optimal shadow map distribution is:

$$\frac{ds}{z\,dz} = \rho \Rightarrow s(z) = \int_0^s ds = \frac{1}{\rho}\int_n^z \frac{1}{z}\,dz = \frac{1}{\rho}\ln\left(\frac{z}{n}\right)$$

  - Since $s(f) = 1$, $\rho = \ln(f / n)$

Parallel-Split Shadow Maps

- Current hardware can't do this non-linear z transform
  - Discretely perform the mapping in steps at the split planes

$$s_i = s\left(C_i^{\log}\right) = \frac{1}{\ln(f/n)} \ln\left(\frac{C_i^{\log}}{n}\right)$$

  - Each split gets 1 / m of total texture resolution, substituting i / m for s_i

$$C_i^{\log} = n\left(\frac{f}{n}\right)^{i/m}$$

Parallel-Split Shadow Maps

- Alternately, the view frustum could be divided into equally sized pieces

$$C_i^{uni} = \frac{(f - n) \times i}{m} + n$$

Parallel-Split Shadow Maps

- Neither split strategy works very well
  - Logarithmic splitting groups split-planes too close to the near plane
  - Uniform splitting doesn't group split-planes close enough to the near plane

Parallel-Split Shadow Maps

- Neither split strategy works very well
  - Logarithmic splitting groups split-planes too close to the near plane
  - Uniform splitting doesn't group split-planes close enough to the near plane
- Instead, use a hybrid of the two

$$C_i = \lambda C_i^{\log} + (1 - \lambda) C_i^{uni}$$

  - λ is tunable parameter
  - The paper calls this the *practical split scheme*

Parallel-Split Shadow Maps

- Light transformation matrices are determined much like before
  - Calculate view-projection matrix for light relative to whole view frustum
  - Transform each split region to light's post-projection space
  - Calculate AABB for transformed split region
  - Use AABB to calculate “crop” transformation to scale and center split region to full view

Parallel-Split Shadow Maps

- To apply shadows, the shader must determine which region contains the current fragment
  - Determine the split-plane, C_s, nearest the camera but farther away than the current fragment
  - C_s determines which shadow map to apply
  - The light transforms, C_i distances, and shadow maps (samplers) must be provided to the shader as arrays of uniforms
    - m is a compile-time constant

Parallel-Split Shadow Maps

- Only directional lights have been dealt with so far

Parallel-Split Shadow Maps

- Only directional lights have been dealt with so far
  - Light transformations for each split region are calculated from the light's post-projection space

Parallel-Split Shadow Maps

- Only directional lights have been dealt with so far
  - Light transformations for each split region are calculated from the light's post-projection space
  - For point lights, transform by the light's view-projection matrix *first*
    - This effectively converts the point-light to a directional light!

References

Zhang, F., Sun, H., Nyman, O. “Parallel-Split Shadow Maps on Programmable GPUs,” in *GPU Gems 3,* ed. Hubert Nguyen, pp. 202 – 237. Boston, MA: Addison-Wesley, 2008. http://appsrv.cse.cuhk.edu.hk/~fzhang/pssm_project/

Wimmer, M., Scherzer, D., and Purgathofer, W. “Light Space Perspective Shadow Maps,” in *Proceedings of Eurographics Symposium on Rendering*, pp. 143 - 151. Norrköping, Sweden: Eurographics Association, 2004. http://www.cg.tuwien.ac.at/research/vr/lispsm/

Fixing Shadow Map Aliasing

- Recall the two sources of resampling aliasing:
  - Perspective aliasing – Comes from the relative orientation and distances of the light and camera
  - Projective aliasing – Comes from the relative orientation of surfaces, camera, and light
- PSMs and PSSMs only handle perspective aliasing
  - Projective aliasing require expensive scene analysis

Fixing Shadow Map Aliasing

- Adaptive shadow maps (ASM) resolve both using a hierarchical data structure
  - Maps are stored in an adaptive quadtree
  - Tree is built iteratively

Adaptive Shadow Map Construction

```
render low-resolution shadow map
do
    for all camera pixels:
        Calculate (s, t, z, l) shadowmap coordinate and LOD
        Lookup shadow map texel
        If texel on shadow edge & page not in ASM:
            Convert (s, t, z, l) to page request
        Transfer page requests to CPU
        Remove invalid page requests
        Generate unique page requests
        Allocate new page in quadtree
        Bin requests into superpages
        Render shadow data into superpages
        Copy shadow data from superpage to quadtree
until page requests == 0
```

Adaptive Shadow Map Construction

- Problems:
  - The edge finding algorithm is EXPENSIVE!
  - The edge finding algorithm can miss some fine shadows

Resolution Matched Shadow Maps

- Store quadtree in a two part structure:
  - Store page table in a mipmap texture
    - LOD specifies the quad tree level
    - The s and t coordinates specify position in quadtree level
    - Value stored specifies location of page in second part of structure
  - Store pages in a single, large texture

RMSM Construction

```
for all pixels in image rendered from camera do
    calculate (s, t, z, l) shadowmap coordinates and LOD
    convert (s, t, z, l) to shadow page request
eliminate redundant requests via connected-components
eliminate invalid requests (compaction)
sort page requests
compact again to generate unique page requests
transfer unique page requests to CPU
allocate new page in quadtree
bin requests into superpages
render shadow data into superpages
copy shadow data from superpage to quadtree memory
```

RMSM Construction

- Phase one:
  - At each pixel calculate (s, t, z) for shadow map lookup
  - Calculate LOD, l, as:

$$dX = \left(\frac{\partial s}{\partial x}, \frac{\partial t}{\partial x} \right)$$

$$dY = \left(\frac{\partial s}{\partial y}, \frac{\partial t}{\partial y} \right)$$

$$A = \left| dX \times dY \right|$$

$$l = \log_2 (\sqrt{A})$$

RMSM Construction

- Eliminate redundant requests:
  - Mark only requests whose below and left neighbors request different pages
- Compact list:
  - Remove all unmarked page requests
  - See [Lefohn et al. 2006]
- Sort list of requests
  - See [Govindaraju et al. 2006]
- Remove all non-unique requests
- Transfer list to CPU

RMSM Construction

- Phase two:
  - Generate quadtree structure on the CPU
  - Merge (bin) requested pages into 1024x1024 “super pages”
  - Render super pages
  - Copy subsections of super pages into final data structure

References

Govindaraju, N. K., Gray, J., Kumar, R., and Manocha, D. 2006. GPUTeraSort: High performance graphics coprocessor sorting for large database management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. 325–336.

Lefohn, A. E., Kniss, J., Strzodka, R., Sengupta, S., and Owens, J. D. 2006. Glift: Generic, efficient, random-access GPU data structures. *ACM Transactions on Graphics*, vol. 26, no. 1, pages 60–99.

Lefohn, A. E., Sengupta, S., and Owens, J. D. 2006. "Resolution Matched Shadow Maps." *ACM Transactions on Graphics*, vol. 26, no. 4, pages 20:1--20:17. ACM, 2007. http://www.idav.ucdavis.edu/publications/print_pub?pub_id=919

Next week...

- Back to shadow volumes
  - Fixing z-pass and z-fail with ZP+ and ++ZP
  - Soft shadows using shadow volumes
  - Shadow volume optimization
  - Read:

Brabec, Stefan and Annen, Thomas and Seidel, Hans-Peter, "Shadow Mapping for Hemi-spherical and Omnidirectional Light Sources." In *Advances in Modeling, Animation and Rendering* (Proceedings Computer Graphics International 2002) , pages 397-408. Springer, 2002. http://www.mpi-inf.mpg.de/~brabec/

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/

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.