VGP353 – Week 1

- Agenda:
  - Course road-map
  - Introduce shadows
    - Importance of shadows
    - Planar projected shadows
    - Soft shadows
    - Shadow textures
  - Projective texturing review
  - First programming assignment

What should you already know?

- All of the prerequisites of VGP351 & VGP352:
  - C++ and object-oriented programming
  - Basic graphics terminology and concepts
  - Some knowledge of linear algebra and vector math
  - Using OpenGL extensions
  - OpenGL Shading Language

What will you learn?

- Algorithms and supporting data-structures for implementing shadows
  - Planar projected shadows
  - Shadow textures
  - Shadow maps
  - Shadow volumes

Grading

- Tests and quizzes
  - Bi-weekly quizzes worth 5 points each
  - Final exam worth 50 points
- Four programming assignments
  - Two will be smallish and worth 10 ponts
  - Two will be largeish and worth 20 points
- One in-class presentation worth 10 points

Grading – Programming Assignments

- Does the program produce the correct output?
- Are the required algorithms / data-structures used?
- Is the code readable and clear?
  - This includes both C++ code *and* shader code!

Grading – In-class Presentation

- Select one paper assigned during the term
- Present a summary of the paper to the class
  - What is the problem being solved?
  - How does the paper solve the problem?
    - What is the overall algorithm?
    - What simplifying assumptions are made?
    - What class of hardware does it target?
  - What is novel about the presented solution?
    - What is the paper's contribution?
  - What questions are left unanswered?
    - What areas remain for further research?

Shadow Terms

- Receiver – object that is shadowed
- Caster – object that blocks light from the receiver
  - May also be called *occluder* because it occludes the light from the receiver
- Umbra – Region on receiver that is completely shadowed
- Penumbra – Transition region between umbra and non-shadowed area

Shadows

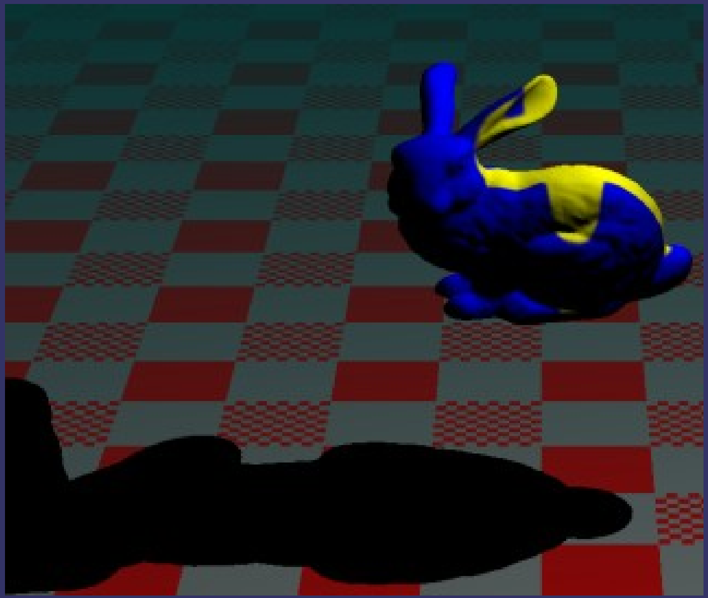
- Why are shadows important to 3D rendering?

Shadows

- Why are shadows important to 3D rendering?
  - Provide additional information about shadow casters
    - Relative position of casters
    - Relative position of casters and receivers
  - Provide additional information about shadow receivers
    - Show additional surface detail

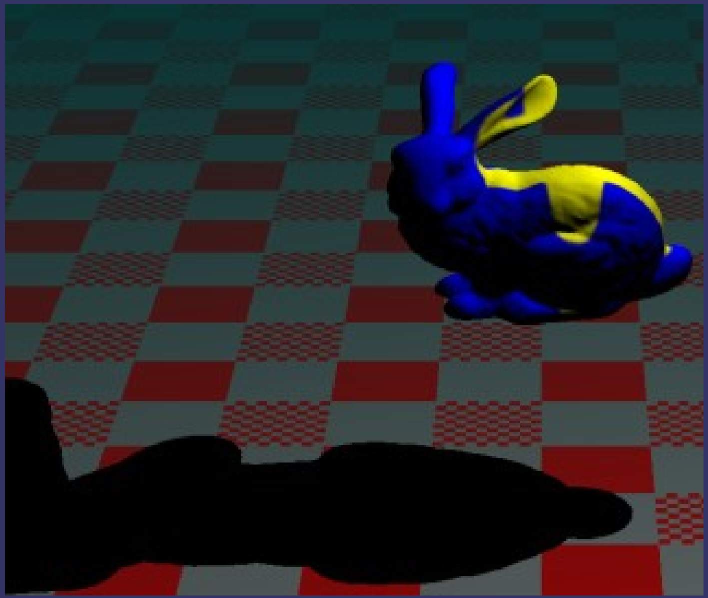
Planar Projected Shadows

- Simplest shadow algorithm: project object geometry directly onto a flat plane



Planar Projected Shadows

- Simplest shadow algorithm: project object geometry directly onto a flat plane
  - As the description implies, this is accomplished using a projection matrix



Planar Projected Shadows

- Given a point on a plane, p, and the normal of that plane, n, the plane equation is:

$$d = -(n \cdot p)$$

$$n \cdot p_i + d = 0$$

  - Every p_i where this equation is 0, is “on” the plane

Planar Projected Shadows

- Given a plane, defined by n and d, and a projection point, L, create a matrix that projects an arbitrary point onto that plane:

$$M_p = \begin{bmatrix} n \cdot L + d - L_x n_x & -L_x n_y & -L_x n_z & -L_x d \\ -L_y n_x & n \cdot L + d - L_y n_y & -L_y n_z & -L_y d \\ -L_z n_x & -L_z n_y & n \cdot L + d - L_z n_z & -L_z d \\ -n_x & -n_y & -n_z & n \cdot L \end{bmatrix}$$

  – This matrix is similar to the matrix used to project onto the view plane from the eye point

Planar Projected Shadows

- If n and d define the ground plane and L is the position of the light, M_p will project world-space geometry onto the ground plane

Planar Projected Shadows

- If n and d define the ground plane and L is the position of the light, M_p will project world-space geometry onto the ground plane
- Question: Where do we insert M_p in the transformation matrix?

Planar Projected Shadows

- If n and d define the ground plane and L is the position of the light, M_p will project world-space geometry onto the ground plane
- Question: Where do we insert M_p in the transformation matrix?
  - Answer: After the object-to-world space transformations, but before the world-to-eye space transformation

$$M = M_{eye} M_p M_{world}$$

Planar Projected Shadows

- Can be drawn several different ways

Planar Projected Shadows

- Can be drawn several different ways
  - Disable depth buffer writes

```
glDepthMask(GL_FALSE);
```

  - Draw shadow to alpha component

```
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_TRUE);
```

  - Re-enable depth buffer writes

```
glDepthMask(GL_TRUE);
```

  - Draw object normally
  - Draw ground plane and modulate with destination alpha

```
glEnable(GL_BLEND);
glBlendFunc(GL_ONE_MINUS_DST_ALPHA, GL_ONE);
```

Hard Shadows vs. Soft Shadows

- Hard shadows are better than nothing, but still not very realistic
  - Perfectly hard shadows are only cast by infinitesimal light sources...the super bright LED in a dark room
  - Or if the light is *very* far away from the shadow caster relative to the size of the light source
  - If the light has any area, it will cast soft shadows

Hard Shadows vs. Soft Shadows

- Hard shadows are better than nothing, but still not very realistic
  - Perfectly hard shadows are only cast by infinitesimal light sources...the super bright LED in a dark room
  - Or if the light is *very* far away from the shadow caster relative to the size of the light source
  - If the light has any area, it will cast soft shadows

Really talking about the solid angle of the light from the PoV of the occluder

Hard Shadows vs. Soft Shadows

- Hard shadows are better than nothing, but still not very realistic
  - Perfectly hard shadows are only cast by infinitesimal light sources...the super bright LED in a dark room
  - Or if the light is *very* far away from the shadow caster relative to the size of the light source
  - If the light has any area, it will cast soft shadows
- Can this technique be extended to create soft shadows?

Heckbert and Herf's Method

- Simulate an area light with many point lights on the area light's surface
  - If *lots* of sample points are used, this method produces *very* good results

Heckbert and Herf's Method

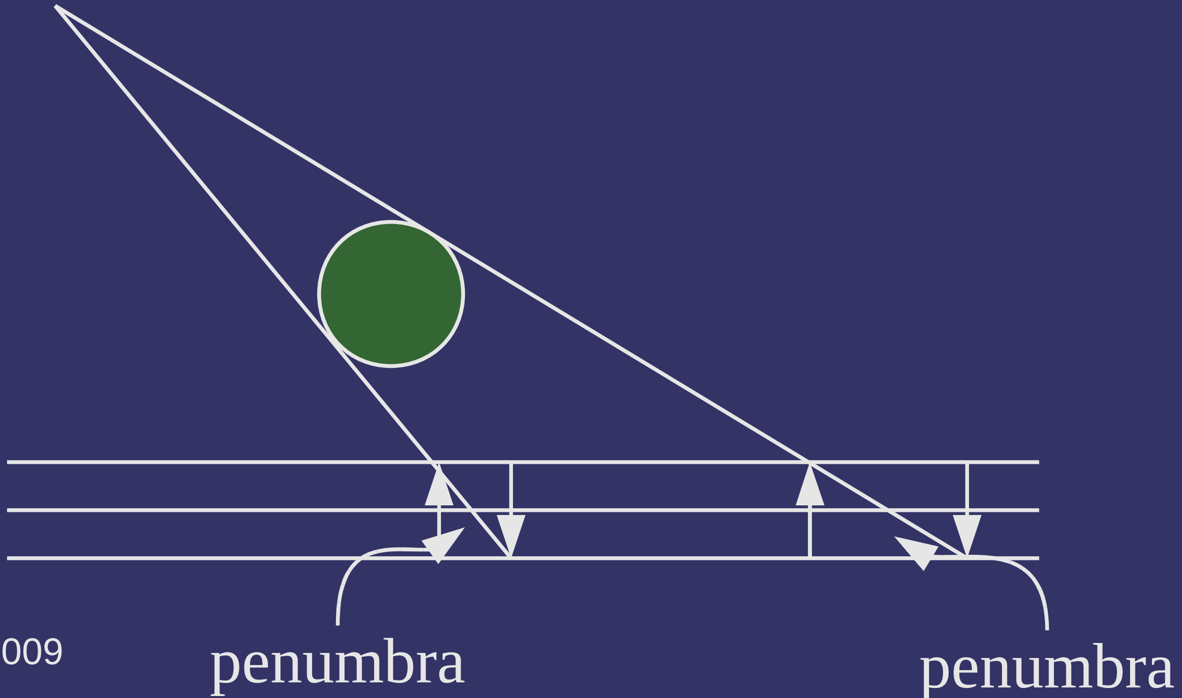
- Simulate an area light with many point lights on the area light's surface
  - If *lots* of sample points are used, this method produces *very* good results
  - If *lots* of sample points are used, this method produces *very* slow results

Heckbert and Herf's Method

- Simulate an area light with many point lights on the area light's surface
  - If *lots* of sample points are used, this method produces *very* good results
  - If *lots* of sample points are used, this method produces *very* slow results
  - Some optimizations are possible:
    - Scale number of samples with size of light
    - Scale number of samples with distance between light and shadow caster

Gooch's Method

- By moving the receiving plane towards and away from the light, the penumbra can be simulated
  - Accomplished by biasing d in the plane equation
  - After the projecting onto the offset plane, move the projected (flattened) object back
  - The simulated penumbra is always too big



14-July-2009

References

Gooch, B., Sloan, P. J., Gooch, A., Shirley, P., and Riesenfeld, R. 1999. Interactive technical illustration. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics* (Atlanta, Georgia, United States, April 26 - 29, 1999). I3D '99. ACM, New York, NY, 31-38. http://www.cs.utah.edu/~bgooch/ITI/

Paul Heckbert and Michael Herf, *Simulating Soft Shadows with Graphics Hardware.* CMU-CS-97-104, CS Dept, Carnegie Mellon U., Jan. 1997. http://www.stereopsis.com/shadow/

Planar Projected Shadows

- As discussed previously, planar projected shadows have a number of faults

Planar Projected Shadows

- As discussed previously, planar projected shadows have a number of faults
  - No self-shadowing
  - Can only cast shadows on the ground plane

Planar Projected Shadows

- As discussed previously, planar projected shadows have a number of faults
  - No self-shadowing
  - Can only cast shadows on the ground plane
  - Can only cast shadows on a ***flat*** ground plane

Planar Projected Shadows

- As discussed previously, planar projected shadows have a number of faults
  - No self-shadowing
  - Can only cast shadows on the ground plane
  - Can only cast shadows on a ***flat*** ground plane
- Shadow textures fix *most* of these problems

Shadow Textures

- Algorithm outline:
  - Render shadow caster to a texture from the point of view of the light
    - Texture background is the color of the light
    - Object is rendered in black
  - Using *projective texturing* cast the shadow texture onto each shadow receiver
  - Use the sampled texture color as the light color

Shadow Textures

- Advantages?

Original image from *Battlefield 1942* © Copyright Digital Illusions CE 2002.

Shadow Textures

- Advantages?
  - Can cast shadows on non-flat surfaces
  - Can cast shadows on multiple objects

Original image from *Battlefield 1942* © Copyright Digital Illusions CE 2002.

Shadow Textures

- Advantages?
  - Can cast shadows on non-flat surfaces
  - Can cast shadows on multiple objects
- Disadvantages?

Original image from *Battlefield 1942* © Copyright Digital Illusions CE 2002.

Shadow Textures

- Advantages?
  - Can cast shadows on non-flat surfaces
  - Can cast shadows on multiple objects
- Disadvantages?
  - No self-shadowing
    - Shadow *maps* will solve this problem...next week
  - Requires render-to-texture pass for *each* shadow caster for each light
  - Shadow receiver must sample multiple shadow textures

Shadow Texture Creation

- Setup model-view-projection matrix to render from the light looking at the object
  - The light position becomes the eye-point
  - Set the FoV to just enclose the object
    - The object's bounding box is helpful here
- Render object as shadow
  - Clear the color buffer to the light's color
  - Render the object as solid black
    - Can “fake” soft shadows by using distance from light (eye) to determine color: closer to the light is darker, farther is lighter

Determining Receiver / Caster

- For each shadow texture, determine which objects are potential receivers
  - If the object is *completely* on the opposite side of the near plane from the light, it is a candidate

Projective Texturing

- Does what it says: projects a texture onto an object
- This is a *perspective* projection, so what is needed to make it “work”?

Projective Texturing

- Does what it says: projects a texture onto an object
- This is a *perspective* projection, so what is needed to make it “work”?
  - Divide by Z...just like perspective viewing projections
    - Uses the q texture coordinate

Projective Texturing

- Algorithm outline:
  - Use object-space vertex positions as initial texture coordinates
  - Transform object-space texture coordinate to projector-space
  - Apply perspective transformation
    - Same MVP matrix as is used to render to the texture
  - Scale and bias coordinates from [-1, 1] to [0, 1]
    - Unless one of the mirroring wrap modes is being used

Projective Texturing

- Uses different sampling functions in GLSL:
  - `texture[123]DProj` vs `texture[123]D`
  - Use these functions instead of doing the perspective divide by hand
  - Cubic textures are not supported. Why?

Projective Texturing

- Uses different sampling functions in GLSL:
  - `texture[123]DProj` vs `texture[123]D`
  - Use these functions instead of doing the perspective divide by hand
  - Cubic textures are not supported. Why?
    - The q component is already used as part of the texture lookup!

Projective Texturing

- What happens if the point is *behind* the projection point?

  Hint: What happens if an object is behind the eye?

Projective Texturing

- What happens if the point is *behind* the projection point?

  Hint: What happens if an object is behind the eye?

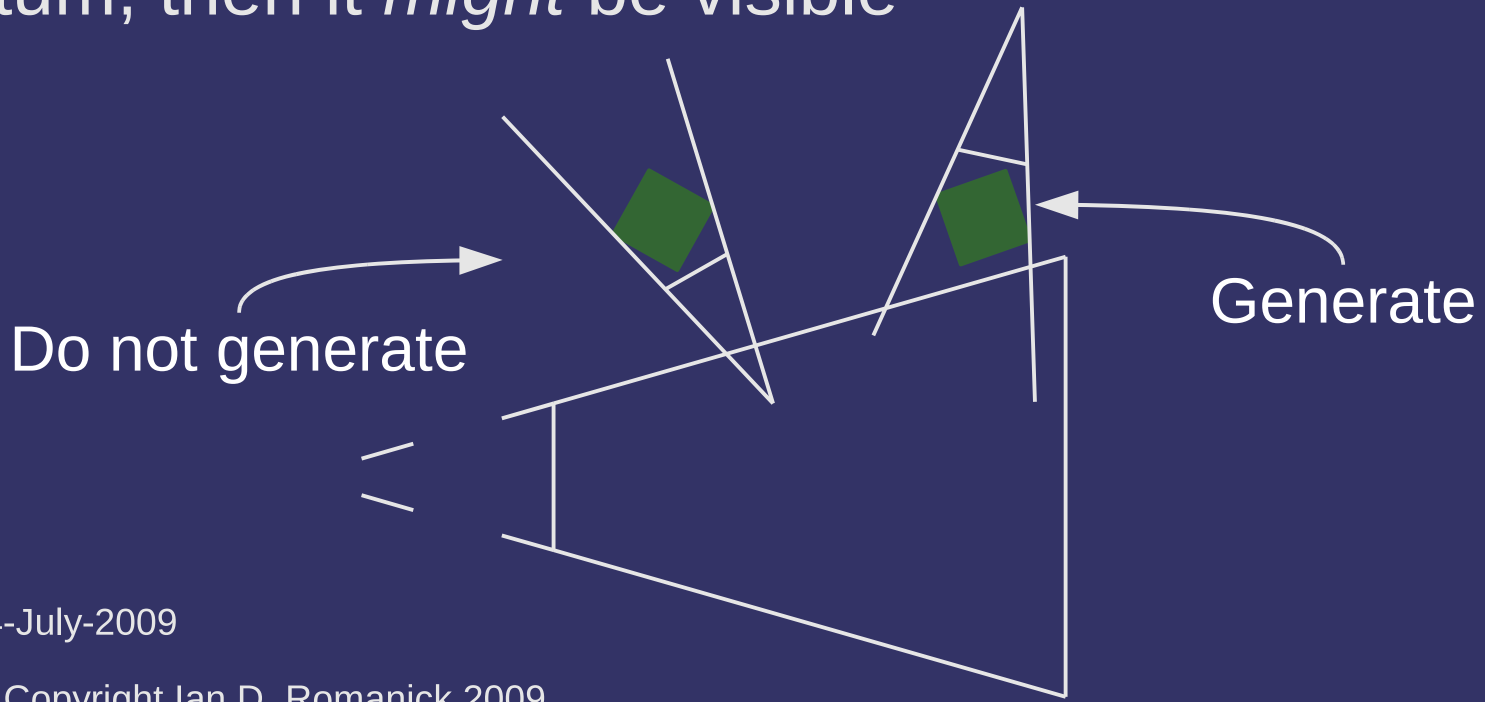
  - It gets a *negative* Z (or q) value
  - The projection then “flips” the position
    - Because it divides by a negative number

Optimizations

- Performance problems with shadow textures:
  - Lots of textures need to be generated *per frame*
  - Shadow receivers need to read lots of textures
- General speed-up techniques:
  - Regenerate a texture only if light or caster moved
  - Generate textures for shadows that might intersect view volume
  - Apply texture only to objects that might be shadowed
  - Composite multiple shadow textures together

Optimizations

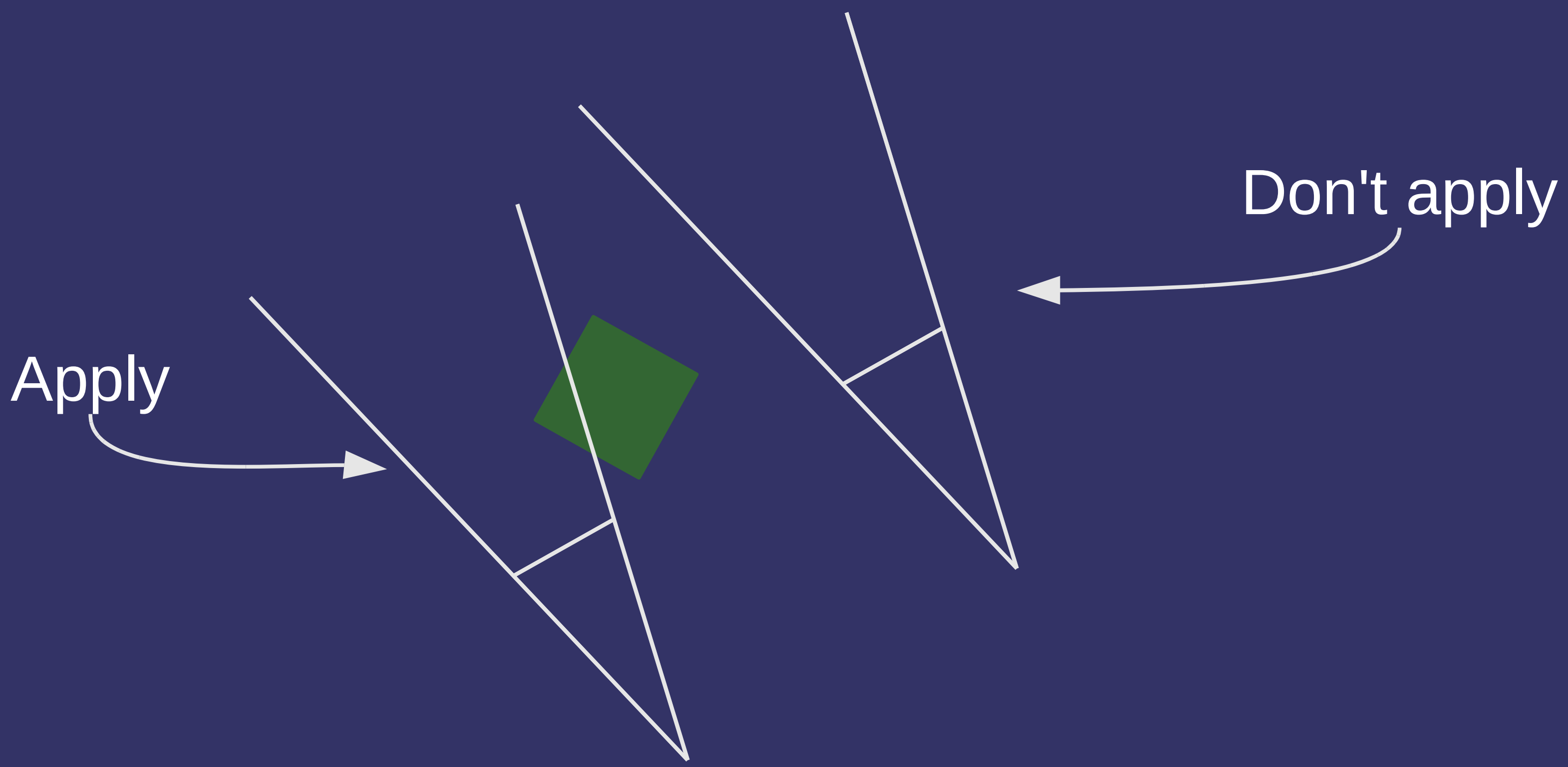
- Generate textures for shadows that might intersect view volume
  - Each shadow texture has an associated frustum
    - “View” frustum used to render the shadow texture
  - If the shadow's frustum intersects the view (eye) frustum, then it *might* be visible



14-July-2009

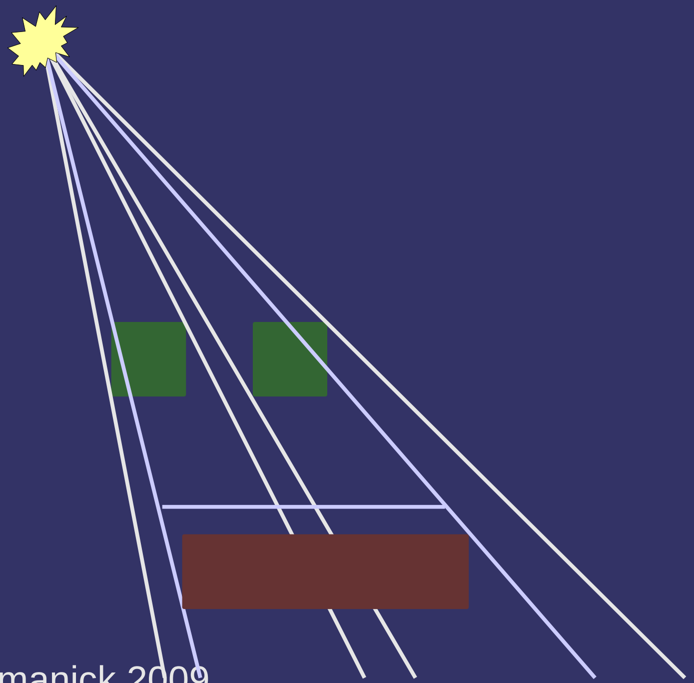
Optimizations

- Apply texture only to objects that might be shadowed
  - Any object that does not intersect the shadow's frustum is not a receiver



Optimizations

- Composite multiple shadow textures together
  - Many casters can affect all members of a group of receivers
  - Create a new shadow texture by compositing all potential casters shadow textures together
  - Project each shadow texture onto the near-plane



References

Bloom, Charles. *Projective Shadow Mapping* [article on-line]. June 30, 2000, accessed April 4, 2008; available from http://www.cbloom.com/3d/techdocs/shadowmap.txt; Internet.

Bloom, Charles, and Teschner, Phil. *Advanced Techniques in Shadow Mapping* [article on-line]. June 3, 2001, accessed April 4, 2008; available from http://www.cbloom.com/3d/techdocs/shadowmap_advanced.txt; Internet.

Next week...

- Shadow maps, part 1
  - Read:

Eric Haines, "Soft Planar Shadows Using Plateaus." journal of graphics tools , vol. 6 , no. 1 , pages 19-27. 2001. http://www.acm.org/tog/editors/erich/plateaus.pdf

Everitt, Cass; Rege, Ashu; and Cebnoyan, Cem, *Hardware Shadow Mapping*. NVIDIA. Decemeber 2001. http://developer.nvidia.com/object/hwshadowmap_paper.html

  - Start assignment #1... due *next week*

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.