# *VGP353 – Week 4*

- Agenda:
  - Course road-map
  - Introduce shadows
    - Importance of shadows
    - Planar projected shadows
    - Soft shadows
  - 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
- Programming assignments
  - Five weekly or bi-weekly programming assignments worth 10 points each
  - One term project of at least 3 weeks worth 50 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*

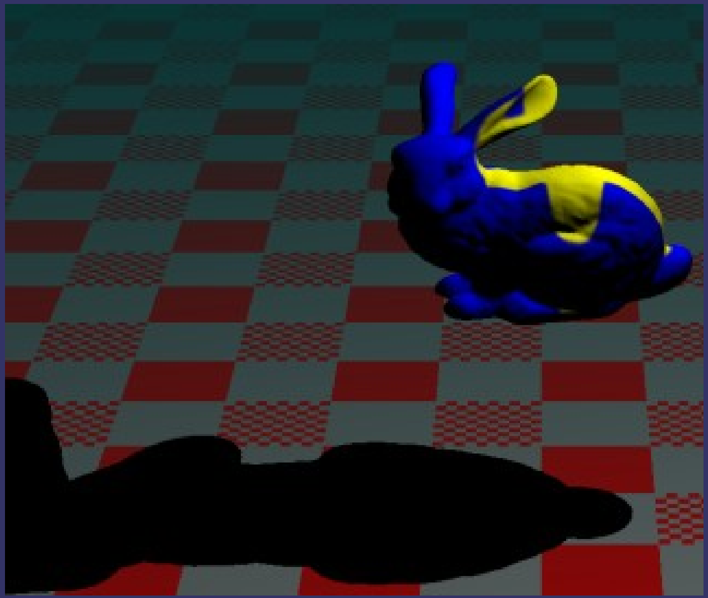
- Why are shadows important to 3D rendering?

# 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

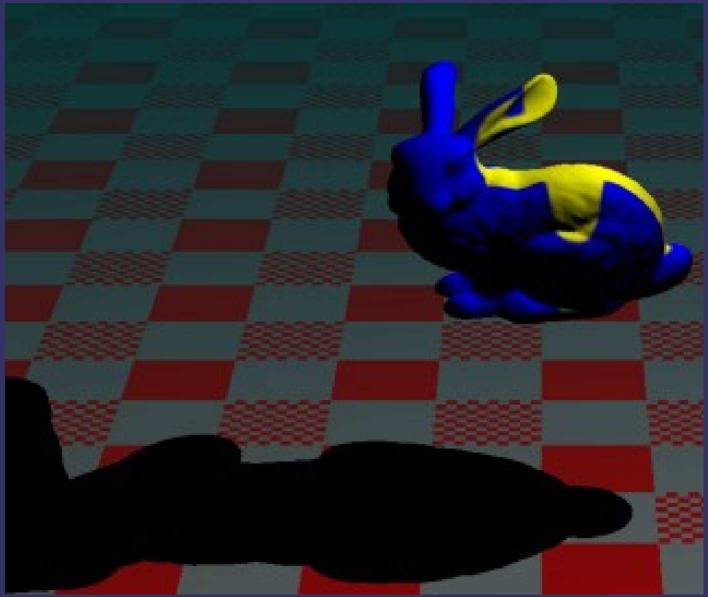
# *Planar Projected Shadows*

- 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, $P$, create a matrix that projects an arbitrary point onto that plane:

$$M_p = \begin{bmatrix} n \cdot P + d - P_x n_x & -P_x n_y & -P_x n_z & -P_x d \\ -P_y n_x & n \cdot P + d - P_y n_y & -P_y n_z & -P_y d \\ -P_z n_x & -P_z n_y & n \cdot P + d - P_z n_z & -P_z d \\ -n_x & -n_y & -n_z & n \cdot P \end{bmatrix}$$

  - 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 $P$ 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 $P$ 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 $P$ 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
- 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*

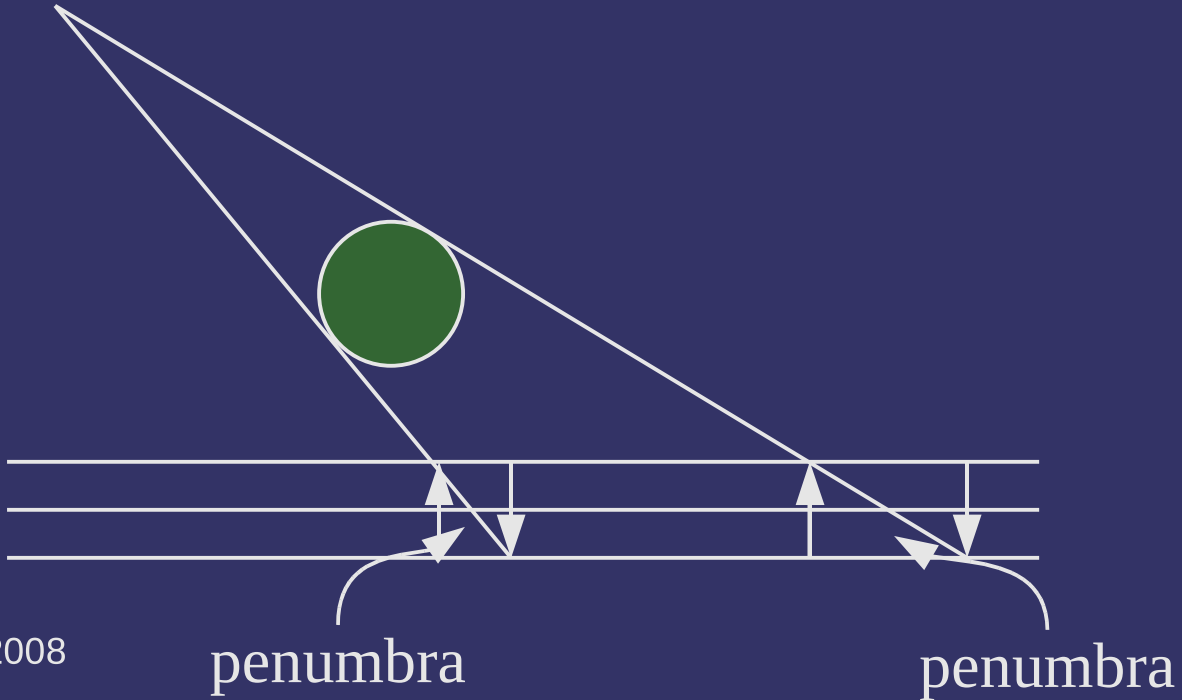
- Simulate an area light with many point lights on the area light's surface
  - If *lots* of sample points are used, this method produces *very* good results
  - If *lots* of sample points are used, this method produces *very* slow results

# *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



# 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/

# *Next week...*

- Shadow textures
- Projective texturing
  - We talked about this in VGP351, so this *should* just be a refreher!

# *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.