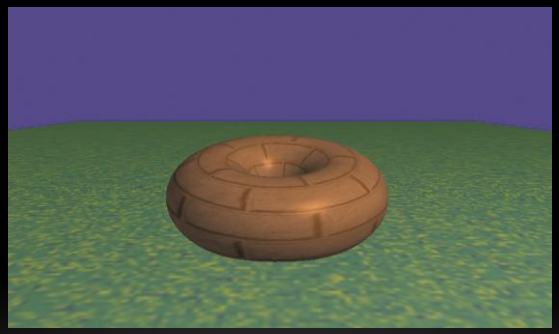
SHADOWS

OUTLINE

- Importance of Shadows
- Projective Shadows
- Shadow Volumes
- Shadow Mapping
- Example
- Shadow Mapping Artifacts

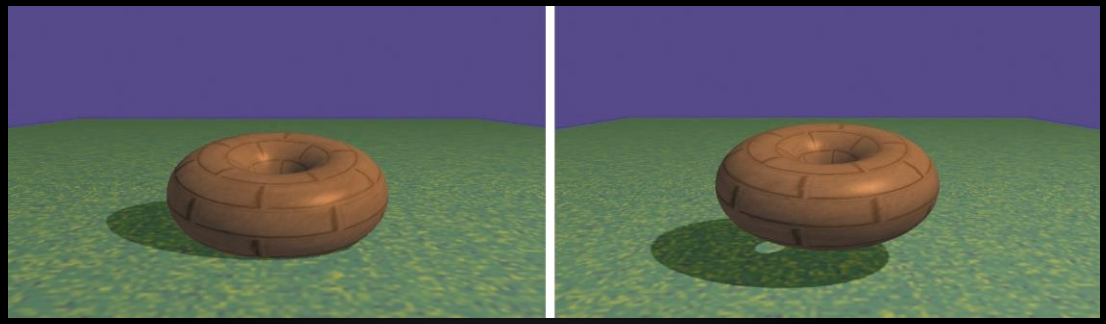
IMPORTANCE OF SHADOWS

- When there is more than one object in the scene (which is usually true)
- Shadows give clues to relative depth, location of objects



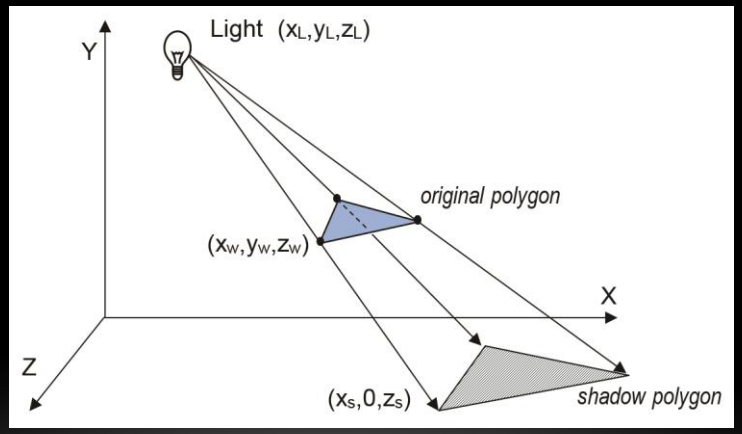
IMPORTANCE OF SHADOWS

- Compared to the previous image, now we have context for where the torus is located with respect to the surface below it
- Shadow color is defined by a darker color blended with the original color of the surface



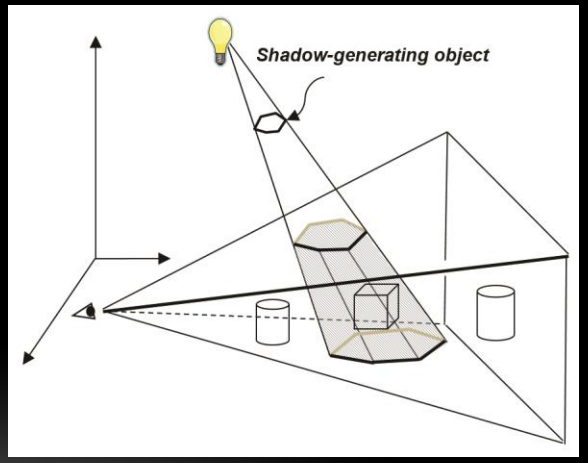
PROJECTIVE SHADOWS

- We already know how to do projection
  - This is what happens when we render a 3D object on a 2D plane
- Can use this same approach to project a shadow of an object
  - Only problem – it only works on a flat plane surface



SHADOW VOLUMES

- An approach to overcoming projective shadow problem
- Identify the spatial volume of an object's shadow
  - Find that volume's intersection with the view volume
  - Determine which scene objects are also in that shadow volume
- Very accurate
- Fewer artifacts than other methods
- But very computationally expensive



SHADOW MAPPING

- Another approach
  - Assume that anything that is not in the path of the light is in shadow
  - We can do this
    - Use a hidden surface removal algorithm and use the depth information in the z buffer to determine what is in shadow
      - Temporarily “move” the camera to the same position as the light source
      - Apply the z-buffer hidden surface removal algorithm
        - Don’t draw on this pass!
      - Use this depth information to find the shadows
      - Render second pass to the screen from the original point of view of the camera
        - If a pixel is in shadow, only draw its ambient component

SHADOW MAPPING

- Approach is called either shadow buffering or shadow texturing
  - In shadow buffering, use an additional depth buffer to contain shadow/depth info
  - In shadow texturing, copy the depth buffer to a texture buffer and apply it to objects

SHADOW MAPPING - PASS 1

- “Drawing” objects from the light position
  - First pass uses only vertex shader, not fragment shader (since we are not actually drawing)
    - Disable color output
    - Build a look-at matrix to “move” the camera to the light position
    - For each object in the scene, create the shadow MVP (model-view-perspective) matrix and call glDrawArrays (which are not being rendered, only processed)

SHADOW MAPPING - PASS 1.5

- Either:
  - Generate an empty shadow texture and use glCopyTexImage2D() to copy the active depth buffer into the shadow texture
- Or:
  - Build a custom framebuffer in pass 1 and attach shadow texture to it using glFrameBufferTexture().

SHADOW MAPPING - PASS 2

- We are using two MVP matrices
  - One for our actual scene objects
  - One for the shadows
- Our (default) scene coordinates are from -1 to 1, and default texture coordinates are from 0 to 1
  - Use the below transformation matrix on the shadowMVP to convert it to texture coords.

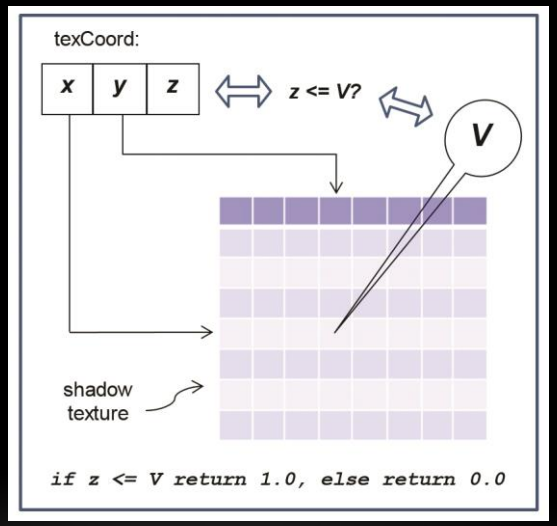
$$B = \begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

SHADOW MAPPING - PASS 2 (CONTINUED)

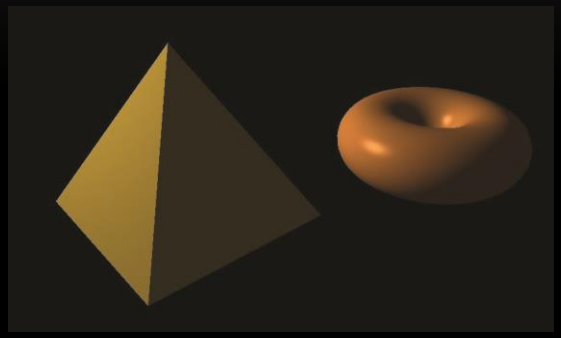
- Enable shadow texture
- Enable color output
- Enable GLSL rendering programs (both vertex and fragment shaders)
- Build object MVP matrix
- Build shadow MVP (using transform from previous slide)
- Sand matrices to shader uniform variables
- Enable all buffers (as usual)
- call glDrawArrays

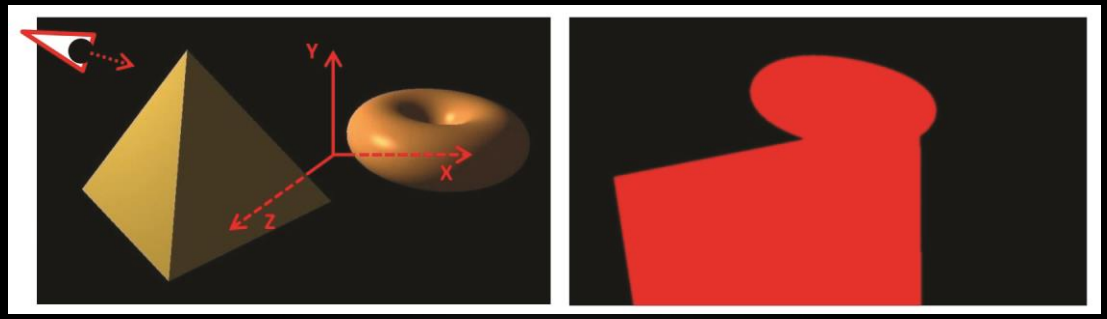
SHADOW MAPPING – PASS 2

- OpenGL provides a special sampler variable called a sampler2DShadow that can be attached to a shadow texture
- textureProj() method is used to determine if object depth is less than or equal to shadow depth in texture, and if so, return true (1) or if not, return false (0)



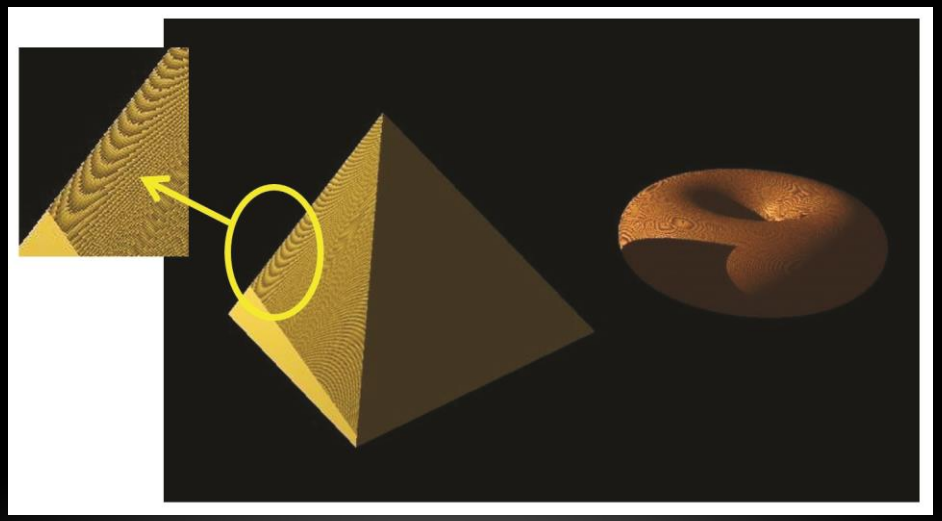
EXAMPLE





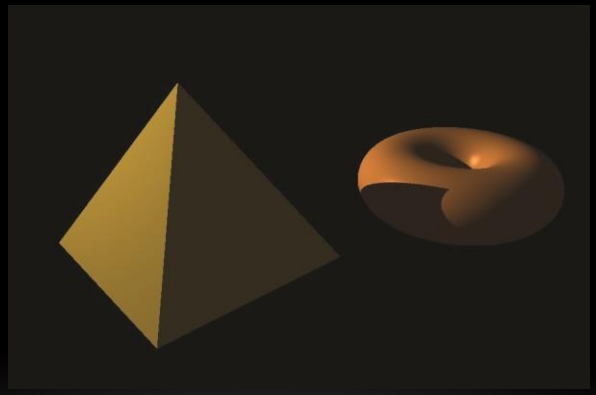
WHAT COULD POSSIBLY GO WRONG?

- “Shadow acne” or “erroneous self-shadowing”
- Shadow acne caused by:
  - Rounding errors during depth testing
  - Precision differences between texture map and depth computation



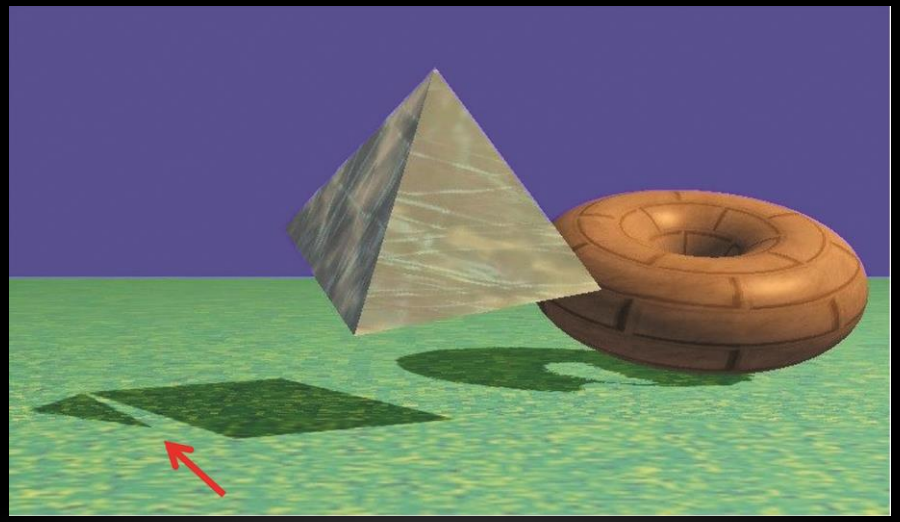
THE FIX

- Move every pixel “slightly” closer to light in pass one
- Move them back to correct position for pass two
- BUT the fix can cause other artifacts to occur



PETER PANNING

- Holes appear in the shadows
  - Adjust the offset in glPolygonOffset()
    - Too little you still get acne
    - Too much and you get peter panning

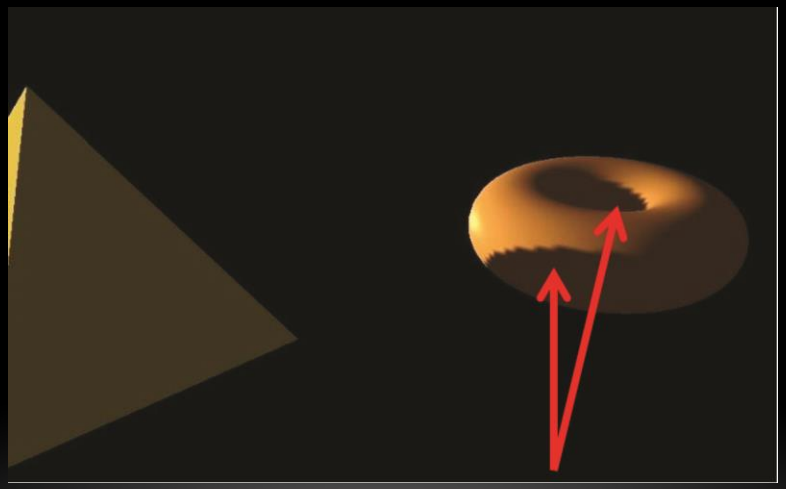


NOTHING ELSE COULD GO WRONG, COULD IT?

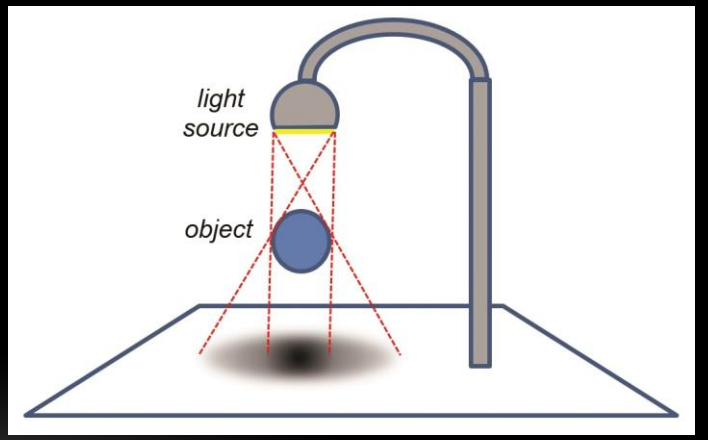
- Of course it could…
  - Can end up with repeated shadows
    - We are using two view volumes in pass 1 and pass 2
    - If a texture coordinate is used outside its range, the texture can repeat
      - (Remember GL_REPEAT?)
    - Can use GL_CLAMP_TO_EDGE to get rid of this
    - But can end up with a “shadow bar” that extends to the end of the scene

NOTHING ELSE COULD GO WRONG, COULD IT?

- Of course it could…
  - Can end up with jagged shadow edges
    - If shadow being cast is significantly larger than what the shadow buffer can accurately represent
  - Not so simple to eliminate



SOFT SHADOWS



SUMMARY

- Importance of Shadows
- Projective Shadows
- Shadow Volumes
- Shadow Mapping
- Example
- Shadow Mapping Artifacts