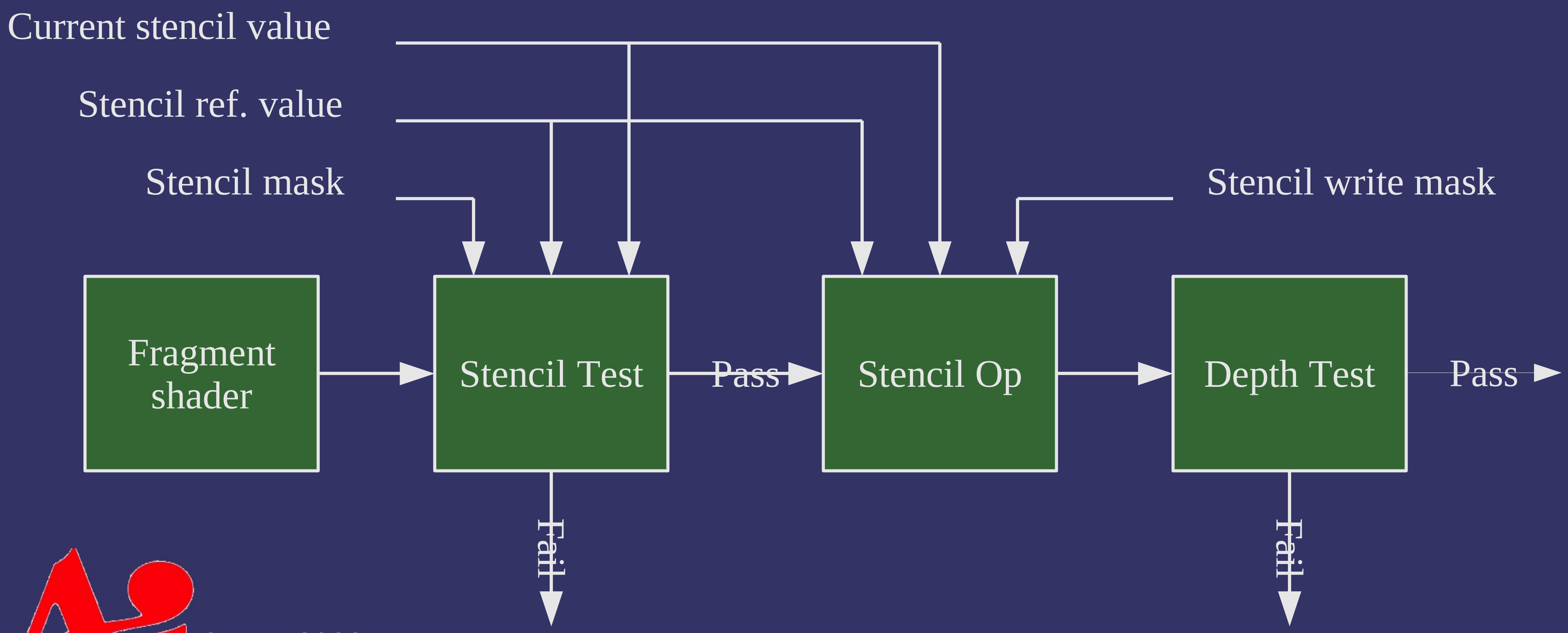
VGP353 – Week 6

- Agenda:
  - Stencil-buffer refresher
  - Theory of shadow volumes

6-May-2008

Stencil Buffer

- Extra per-pixel buffer containing integer values
  - Stencil test and stencil operation occur *after* per-fragment operations and *before* depth testing



Stencil Buffer

- Stencil function is one GL's usual comparators
  - `GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, `GL_ALWAYS`
  - Performs bit-wise operations of (`stencil` & `mask`) `func` (`ref` & `mask`)

Stencil Buffer

```
glStencilFuncSeparate(
  GLenum face,
  GLenum func,
  GLint ref,
  GLuint mask);
```

Stencil Buffer

```
glStencilFuncSeparate(
    GLenum face,
    GLenum func,
    GLint ref,
    GLuint mask);
```

Polygon facing selector: different operations for front and back facing polygons

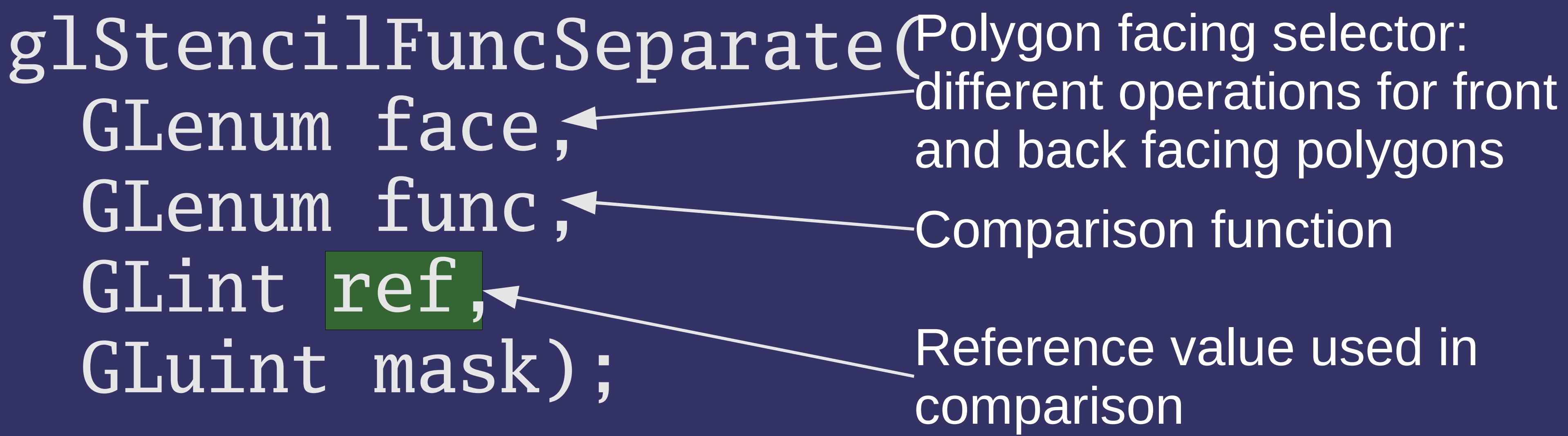
Stencil Buffer

```
glStencilFuncSeparate(
  GLenum face,
  GLenum func,
  GLint ref,
  GLuint mask);
```

Polygon facing selector: different operations for front and back facing polygons

Comparison function

Stencil Buffer



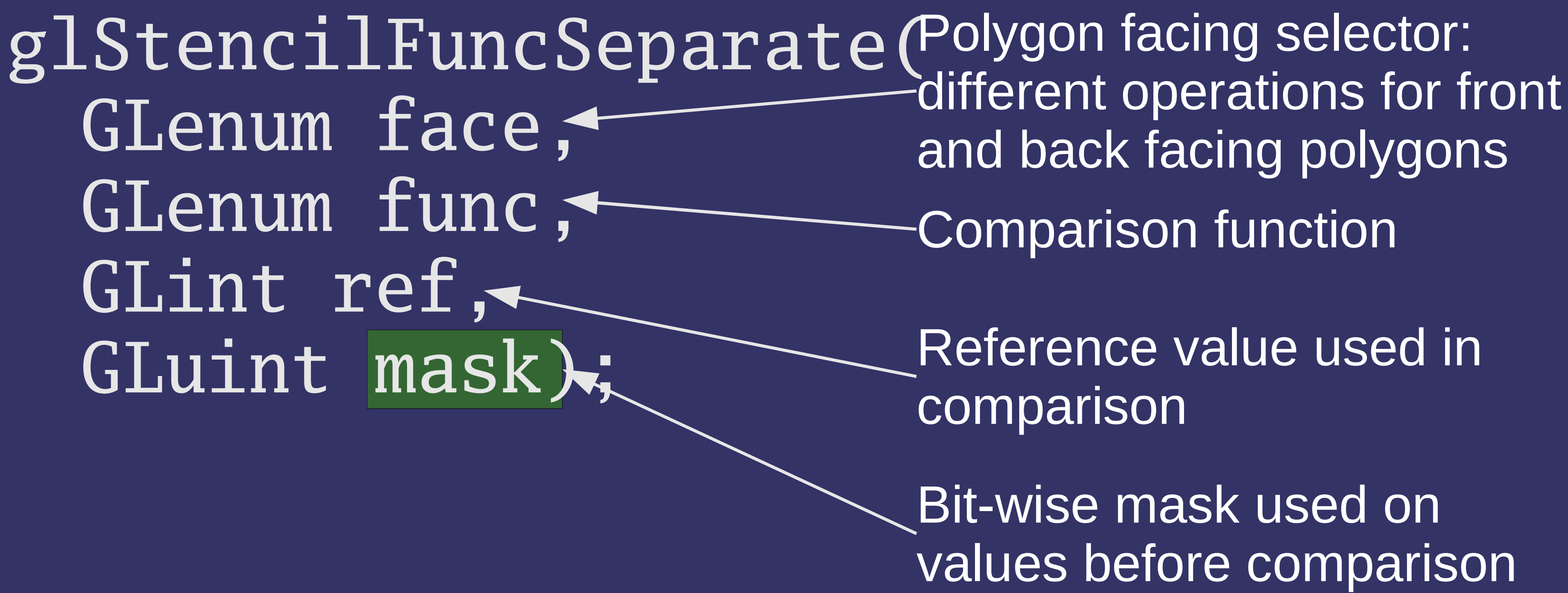
```
glStencilFuncSeparate(
   GLenum face,
   GLenum func,
   GLint ref,
   GLuint mask);
```

Polygon facing selector: different operations for front and back facing polygons

Comparison function

Reference value used in comparison

Stencil Buffer



```
glStencilFuncSeparate(
   GLenum face,
   GLenum func,
   GLint ref,
   GLuint mask);
```

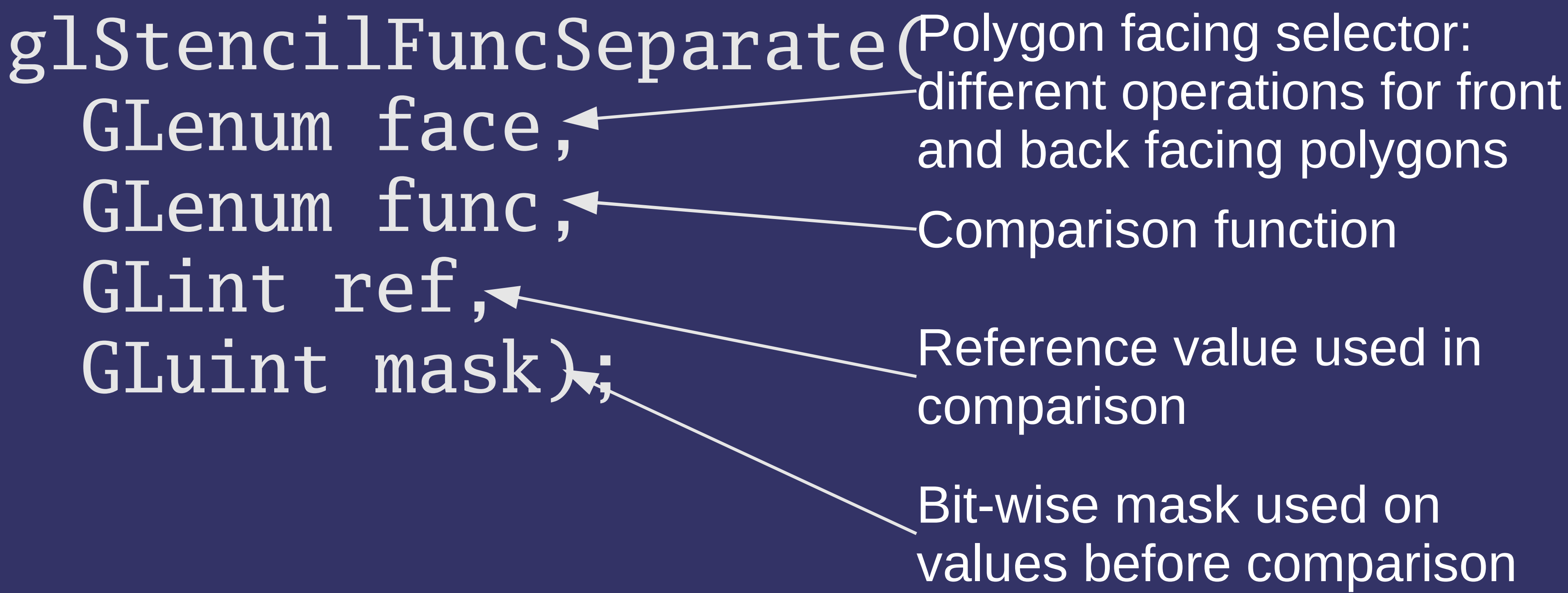
Polygon facing selector: different operations for front and back facing polygons

Comparison function

Reference value used in comparison

Bit-wise mask used on values before comparison

Stencil Buffer



```
glStencilFuncSeparate(
    GLenum face,
    GLenum func,
    GLint ref,
    GLuint mask);
```

Polygon facing selector: different operations for front and back facing polygons

Comparison function

Reference value used in comparison

Bit-wise mask used on values before comparison

- Passing `GL_FRONT_AND_BACK` for `face` acts like GL 1.x `glStencilFunc` function
  - Radeon r300 (e.g., Radeon 9800) needs front and back `ref` and `mask` to be the same

Stencil Buffer

- Stencil operation modifies value in stencil buffer
  - Stencil buffer may be modified even if stencil and depth tests fail!
  - Operation is one of GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR, GL_INVERT, GL_INCR_WRAP, and GL_DECR_WRAP
    - GL_INCR and GL_DECR saturate to maximum value or zero
    - GL_REPLACE stores reference value

Stencil Buffer

```
glStencilOpSeparate(
  GLenum face,
  GLenum sfail,
  GLenum dfail,
  GLenum dpass);
```

Stencil Buffer

```
glStencilOpSeparate(
  GLenum face,
  GLenum sfail,
  GLenum dfail,
  GLenum dpass);
```

Polygon facing selector: different operations for front and back facing polygons

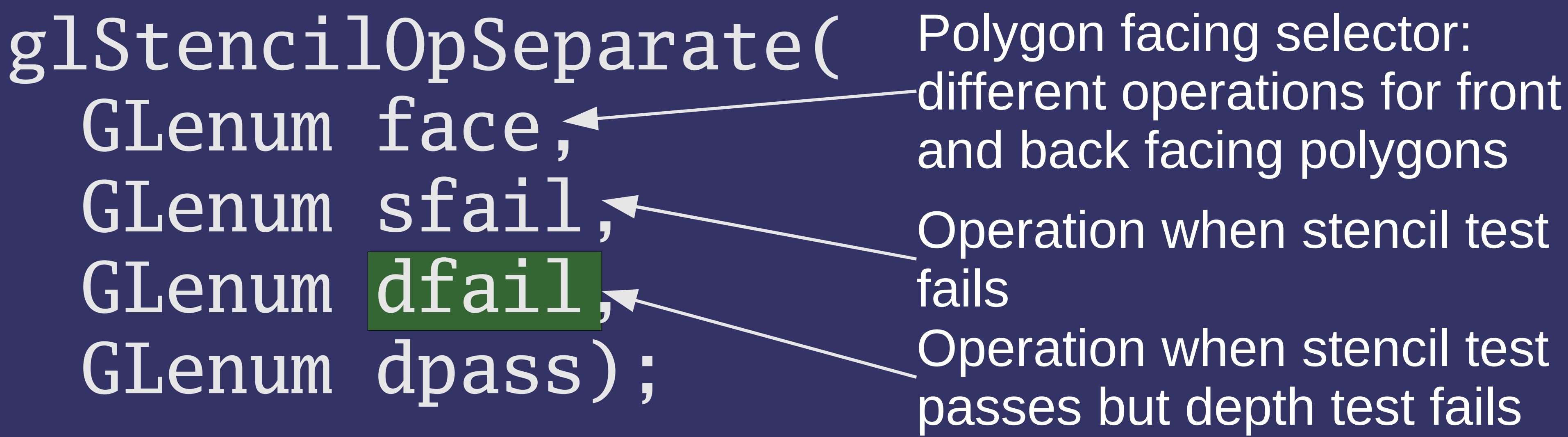
Stencil Buffer

```
glStencilOpSeparate(
   GLenum face,
   GLenum sfail,
   GLenum dfail,
   GLenum dpass);
```

Polygon facing selector: different operations for front and back facing polygons

Operation when stencil test fails

Stencil Buffer



```
glStencilOpSeparate(
  GLenum face,
  GLenum sfail,
  GLenum dfail,
  GLenum dpass);
```

Polygon facing selector: different operations for front and back facing polygons

Operation when stencil test fails

Operation when stencil test passes but depth test fails

Stencil Buffer

```
glStencilOpSeparate(
  GLenum face,
  GLenum sfail,
  GLenum dfail,
  GLenum dpass);
```

Polygon facing selector: different operations for front and back facing polygons

Operation when stencil test fails

Operation when stencil test passes but depth test fails

Operation when stencil and depth tests pass

Stencil Buffer

```
glStencilOpSeparate(
  GLenum face,
  GLenum sfail,
  GLenum dfail,
  GLenum dpass);
```

face: Polygon facing selector: different operations for front and back facing polygons

sfail: Operation when stencil test fails

dfail: Operation when stencil test passes but depth test fails

dpass: Operation when stencil and depth tests pass

- Passing `GL_FRONT_AND_BACK` for `face` acts like GL 1.x `glStencilOp` function

Stencil Buffer

- Stencil buffer can also be cleared
  - `glClearStencil` sets the cleared value
  - Pass `GL_STENCIL_BUFFER_BIT` to `glClear`
  - If depth *and* stencil are used, always clear both together
- Writing of particular bits can be controlled with `glStencilMaskSeparate`
  - Passing `GL_FRONT_AND_BACK` for face parameter acts like GL 1.x `glStencilMask` function
  - Radeon r300 (e.g., Radeon 9800) needs front and back mask to be the same

Stencil Buffer – Example

```
glClearStencil(0);
glClear(GL_STENCIL_BUFFER_BIT);
glEnable(GL_STENCIL_TEST);

/* Write 1 to stencil where polygon is drawn.
 */
glStencilFunc(GL_ALWAYS, 1, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
draw_some_polygon();

/* Draw scene only where stencil buffer is 1.
 */
glStencilFunc(GL_EQUAL, 1, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
draw_scene();
```

Stencil Buffer – Window System

- Stencil buffer is often stored interleaved with depth buffer
  - 8-bit stencil with 24-bit depth is most common
  - Other combinations such as 1-bit stencil with 15-bit depth do exist (very, *very* rare these days)
- Must request a stencil buffer with your window
  - With SDL, this means setting the stencil size attribute to the minimum number of stencil bits required

```
SDL_GL_SetAttribute(SDL_GL_STENCIL_SIZE, 4);
```

Stencil Buffer – FBOs

- Stencil buffers can also be used with framebuffer objects
  - Create with `glRenderbufferStorageEXT` and an internal type of `GL_STENCIL_INDEX_EXT`
    - Sized types are also available
    - There are *no* stencil textures
  - Attach to `GL_STENCIL_ATTACHMENT_EXT`

Stencil Buffer – FBOs

- If depth *and* stencil are required, use the `GL_EXT_packed_depth_stencil` extension
  - Create renderbuffer *or* texture with internal type of `GL_DEPTH_STENCIL_EXT`
    - One sized type of `GL_DEPTH24_STENCIL8_EXT` also available
    - `type` parameter must be `GL_UNSIGNED_INT_24_8_EXT`
    - Treated as a depth texture for texturing
  - Bind same object to both the depth and stencil attachments

Stencil Buffer – FBO Example

```
glGenFramebuffersEXT(1, &fb);
glGenTextures(2, tex_names);

// Setup color texture (mipmap)
glBindTexture(GL_TEXTURE_2D, tex_names[0]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB8, 512, 512, 0, GL_RGBA, GL_INT, NULL);
glGenerateMipmapEXT(GL_TEXTURE_2D);

// Setup depth_stencil texture (not mipmap)
glBindTexture(GL_TEXTURE_2D, tex_names[1]);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH24_STENCIL8_EXT, 512, 512, 0,
             GL_DEPTH_STENCIL_EXT, GL_UNSIGNED_INT_24_8_EXT, NULL);

glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
                          GL_TEXTURE_2D, tex_names[0], 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT,
                          GL_TEXTURE_2D, tex_names[1], 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_STENCIL_ATTACHMENT_EXT,
                          GL_TEXTURE_2D, tex_names[1], 0);
```

Stencil Buffer – FBO Example

```
glGenFramebuffersEXT(1, &fb);
glGenTextures(2, tex_names);

// Setup color texture (mipmap)
glBindTexture(GL_TEXTURE_2D, tex_names[0]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB8, 512, 512, 0, GL_RGBA, GL_INT, NULL);
glGenerateMipmapEXT(GL_TEXTURE_2D);

// Setup depth_stencil texture (not mipmap)
glBindTexture(GL_TEXTURE_2D, tex_names[1]);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH24_STENCIL8_EXT, 512, 512, 0,
             GL_DEPTH_STENCIL_EXT, GL_UNSIGNED_INT_24_8_EXT, NULL);

glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
                          GL_TEXTURE_2D, tex_names[0], 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT,
                          GL_TEXTURE_2D, tex_names[1], 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_STENCIL_ATTACHMENT_EXT,
                          GL_TEXTURE_2D, tex_names[1], 0);
```

Same object attached both places

Break

Shadow Volumes

- Proposed by Frank Crow in 1977
  - Add new geometry to the scene that describes the volume occluded from the light source
  - Objects within the volume are in shadow, objects not within the volume are not
  - Sometimes called *Crow shadows* or *Crow shadow volumes*

Shadow Volumes

- Proposed by Frank Crow in 1977
  - Add new geometry to the scene that describes the volume occluded from the light source
  - Objects within the volume are in shadow, objects not within the volume are not
  - Sometimes called *Crow shadows* or *Crow shadow volumes*
- In 1991, Tim Heidmann showed how the stencil buffer can be used to apply these volumes to a scene
  - This adaptation often called *stencil volume shadows*

Shadow Volumes

- Basic algorithm:
  1. Render scene using only ambient light
  2. For each light in the scene:
     a. Using the depth information from the initial pass, construct a stencil with “holes” where there the light is not occluded.
        - Stencil will be 0 where the light is visible

     b. Render scene again with normal lighting. Use the stencil mask to only draw where the light is not occluded.
        - Configure stencil test to draw only where stencil = 0
  - Two common methods to create this stencil: z-pass and z-fail

Shadow Volumes

- Problems?

Shadow Volumes

- Problems?
  - ***Very*** fill-rate intensive
  - Calculating shadow volumes can be complex and time consuming
  - Difficult to extend to soft-shadows

Shadow Volumes

- Problems?
  - ***Very*** fill-rate intensive
  - Calculating shadow volumes can be complex and time consuming
  - Difficult to extend to soft-shadows
- Advantages?

Shadow Volumes

- Problems?
  - ***Very*** fill-rate intensive
  - Calculating shadow volumes can be complex and time consuming
  - Difficult to extend to soft-shadows
- Advantages?
  - Since everything is done in geometry-space instead of image-space, **no aliasing artifacts!!!**
  - No shadow acne either!

Shadow Volumes – Z-Pass

1. Disable depth and color writes
2. Configure stencil operation:
   - GL_INCR_WRAP on depth pass front-faces
   - GL_DECR_WRAP on depth pass back-faces
   - GL_KEEP for all other cases
3. Draw shadow volumes

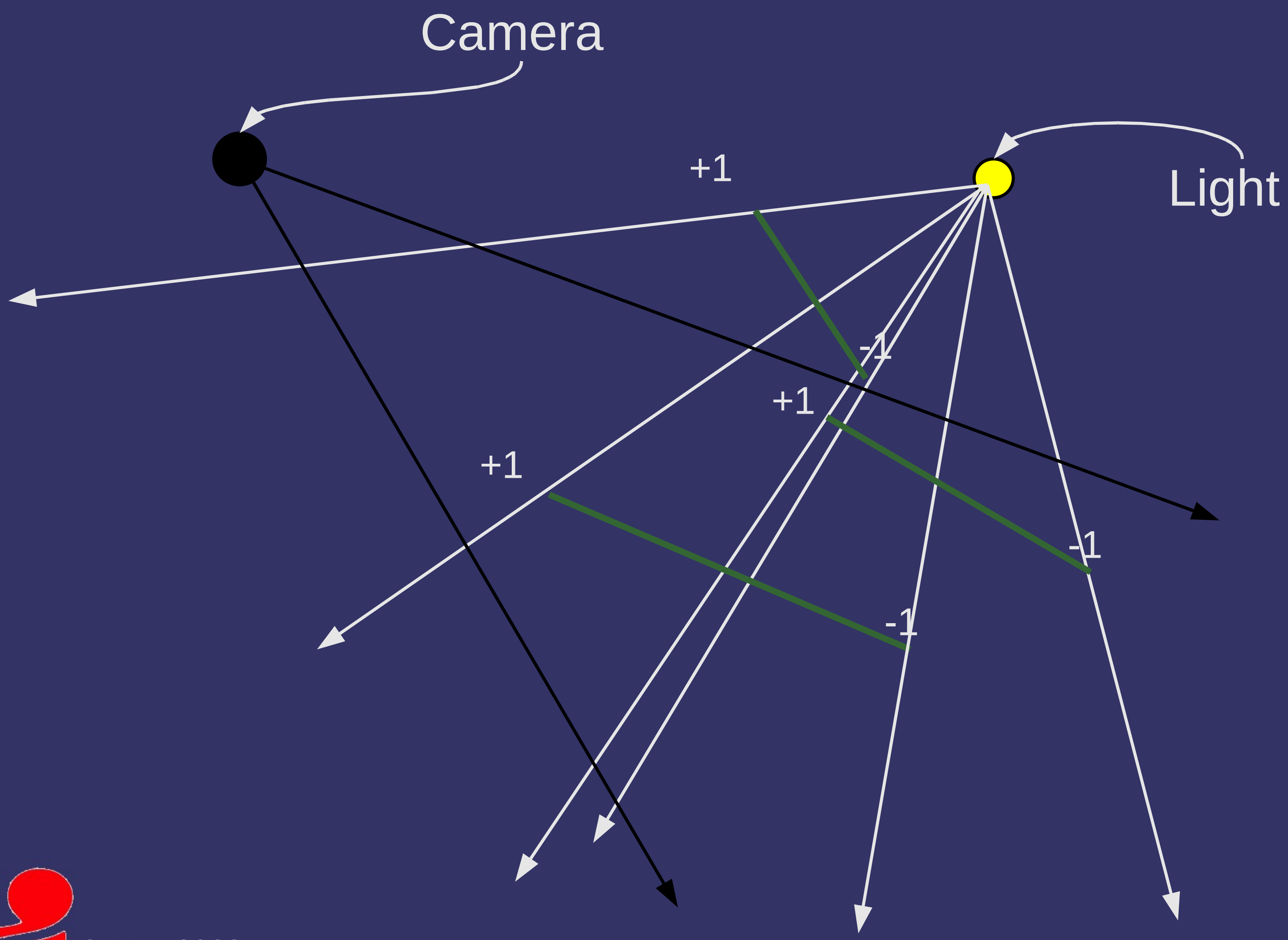
⇨ Why use GL_INCR_WRAP and GL_DECR_WRAP instead of GL_INCR and GL_DECR?

Shadow Volumes – Z-Pass

1. Disable depth and color writes
2. Configure stencil operation:
   - `GL_INCR_WRAP` on depth pass front-faces
   - `GL_DECR_WRAP` on depth pass back-faces
   - `GL_KEEP` for all other cases
3. Draw shadow volumes

- Why use `GL_INCR_WRAP` and `GL_DECR_WRAP` instead of `GL_INCR` and `GL_DECR`?
  - Otherwise, if there are more than 2^n increments before a decrement, the count will be wrong

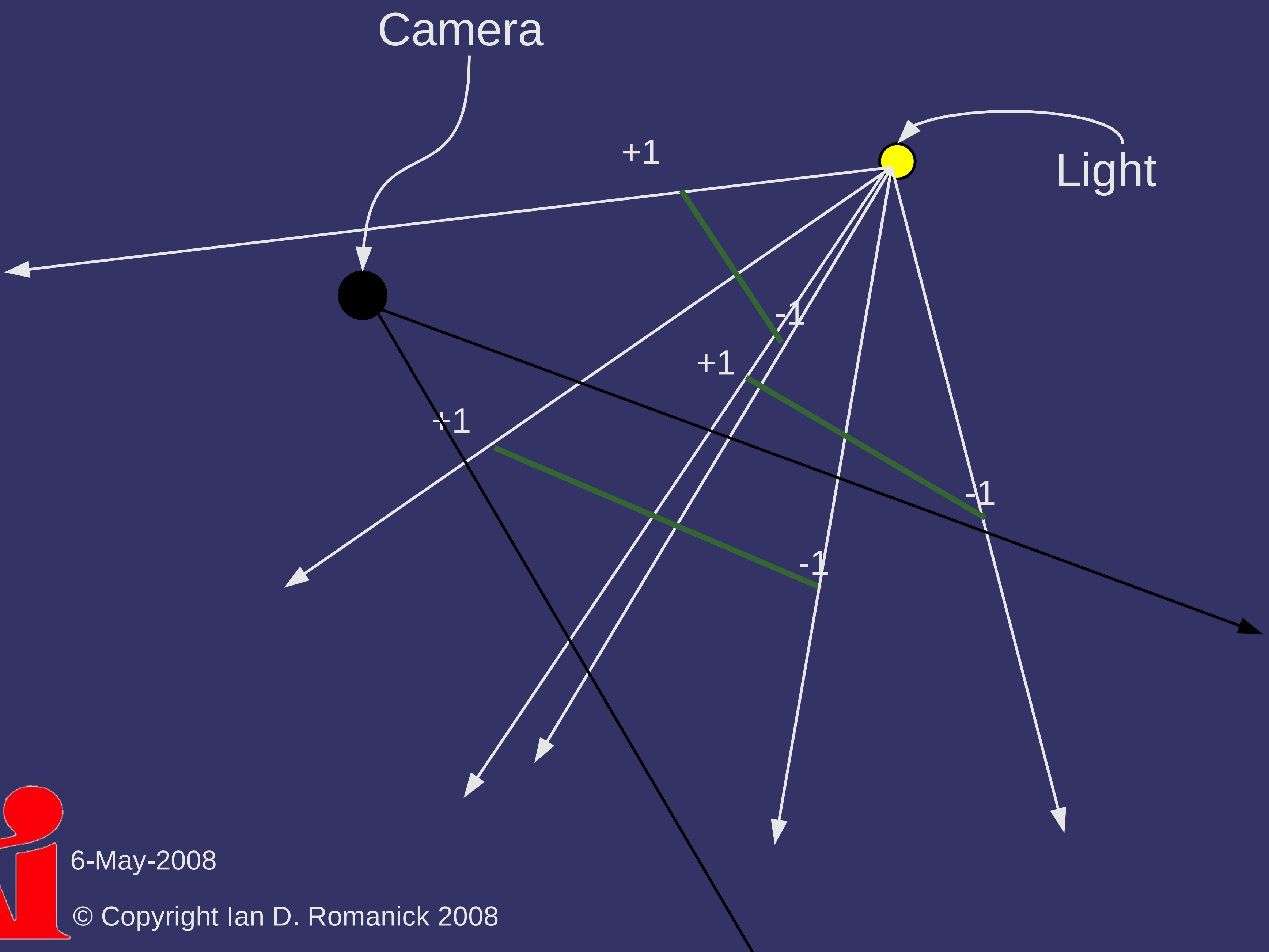
Shadow Volumes – Z-Pass



Shadow Volumes – Z-Pass

- Big problem with z-pass: What if the camera is *inside* a shadow volume?

Shadow Volumes – Z-Pass



6-May-2008

Shadow Volumes – Z-Pass

- Big problem with z-pass: What if the camera is *inside* a shadow volume?
  - The count is too low!

Shadow Volumes – Z-Pass

- Big problem with z-pass: What if the camera is *inside* a shadow volume?
  - The count is too low!
- Possible solutions:
  - Clear stencil buffer to +1 for each volume the camera is inside
    - Expensive to compute
  - Add a “cap” at the near plane for each volume the camera is inside
    - Expensive to compute
  - Use z-fail

Shadow Volumes – Z-Fail

1. Disable depth and color writes
2. Configure stencil operation:
   - `GL_INCR_WRAP` on depth fail back-faces
   - `GL_DECR_WRAP` on depth fail front-faces
   - `GL_KEEP` for all other cases
3. Draw shadow volumes

➪ Method first *publicly* described by John Carmack while working on Doom 3

- Often called *Camack's reverse*

Shadow Volumes – Z-Fail

1. Disable depth and color writes
2. Configure stencil operation:
   - `GL_INCR_WRAP` on depth fail back-faces
   - `GL_DECR_WRAP` on depth fail front-faces
   - `GL_KEEP` for all other cases
3. Draw shadow volumes

Note that the depth test and the polygon facing are reversed compared to z-pass

Shadow Volumes – Z-Fail

- Big problems with z-fail:
  - Since more geometry fails the depth test than passes, this method can use orders of magnitude *more* fill rate
  - US Patent #6,384,822

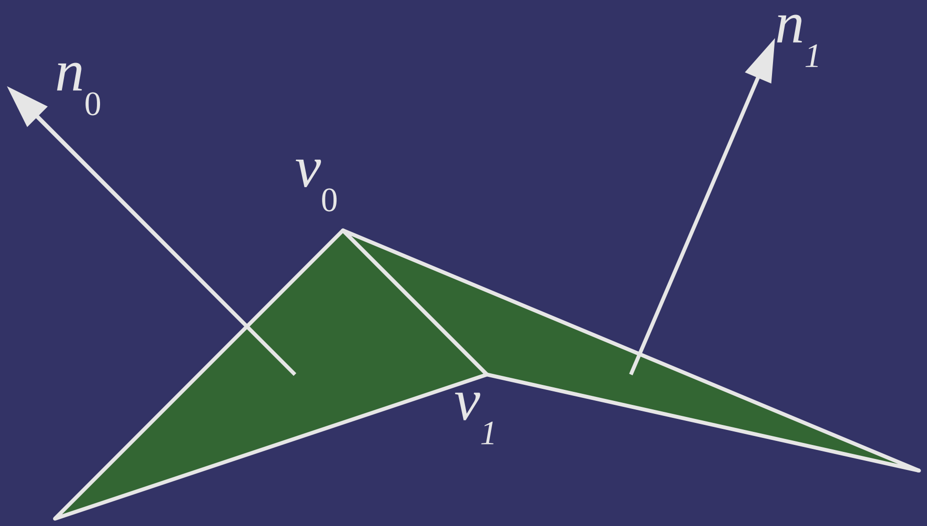
Shadow Volumes

- Shadow volume geometry is made of 3 types of polygons:
  - Front faces of the object (w.r.t. the light)
  - Quads from each silhouette edge (w.r.t. the light) projected to “infinity”
  - Back faces of the object (w.r.t. the light) projected to “infinity”

Shadow Volumes

- Front and back caps are trivial. What about the sides?
  - Add a degenerate quad at each edge of the model
  - Quad stores normals of one polygon with one vertex pair and normals of the other polygon with the other vertex pair
  - In vertex shader, test vertex normal against light. If normal points away from light, project to infinity
    - For silhouette edges one pair will be projected away and the other pair will not

Shadow Volumes



Vertex data for shadow volume quad:

v0	n0
v1	n0
v1	n1
v0	n1

Shadow Volumes

- Advantages?
  - Shadow volume geometry is independent of light position and object orientation
  - Very little work done on the CPU per-frame
  - Static shadow volume data does not need to be re-uploaded to GPU every frame
- Disadvantages?
  - For static lights and geometry a *lot* of redundant work is done every frame
  - True shadow volumes only exist on the GPU, so we can't determine whether the camera is inside a shadow volume

References

http://en.wikipedia.org/wiki/Shadow_volume

Next week...

- More shadow volumes
  - Creating the evil “fins” *Muahahaha!*
  - Quiz #3
    - Week 5 material (PSSMs)
    - Week 6 material (shadow volume theory)

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.