Stencil Shadows

Nathan Cournia

nathan@cournia.com

Clemson University

Outline

- Overview
- Depth-Pass
- Depth-Fail (Carmack's Reverse)
- Capping
- Problems
- Uber Support

Overview

- Stencil shadow volumes enable the visualization of shadows
- Shadows help to delineate form
- Shadows are independent of eye position

Shadow Volumes

- Shadows can be thought of as volumes
- Frank Crow first presented the idea of using shadow volumes for shadow casting in 1977

Determing a Shadow Volume

- Extrude the silhouette of an occluder
- Silhouette/Extrusion is in respect to the light source
- Extrude to infinity

Determining the Silhouette

- Many methods
- No fast methods are known
- Mesh connectivity information helps
- A simple solution:

```
foreach edge in mesh {
  if( sign( dot(edge.left_tri.normal, light_pos) ) !=
      sign( dot(edge.right_tri.normal, light_pos) )) {
    silhouette.add_edge( edge );
  } //end of triangle/light facing check
} //end of edge loop
```

Stencil Shadows

- Tim Heidmann first used stencil buffer to render shadows
- Used stencil buffer to mark entering/leaving shadow volume

Basic Stencil Shadow Volume Algorithm

- Basic stencil shadow algorithm is as follows:

```
enable_depth_writes( );
render_ambient_light( );
foreach light in scene {
  enable_stencil_test( );
  disable_depth_writes( );
  render_shadow_volumes_to_stencil_buffer( );
  enable_depth_writes( );
  render_diffuse_specular_to_non_stenciled_regions( );
}
```

Rendering Shadow Volumes

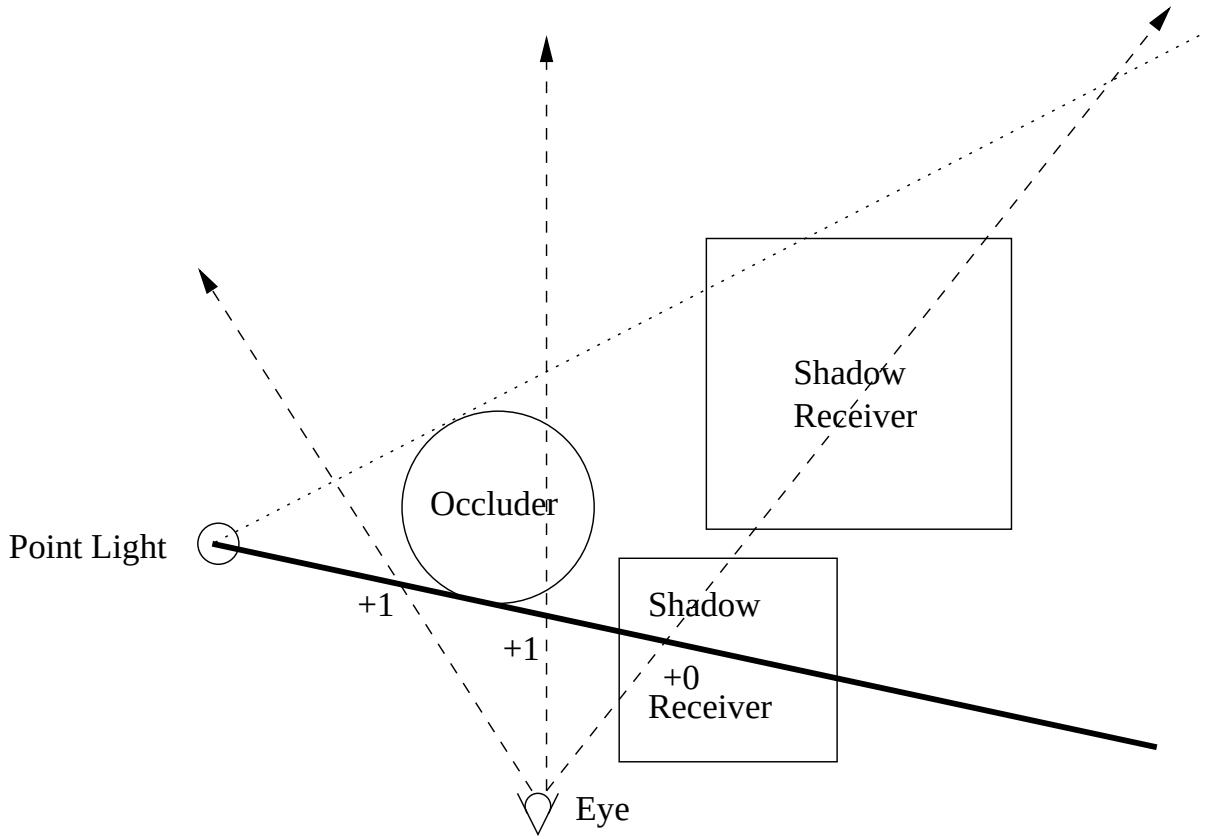
- Two techniques:
  - Depth-Pass
  - Depth-Fail (Carmack's Reverse)
- Techniques differ in 2 areas:
  - Front/Back face rendering order
  - How to update the stencil buffer when depth test passes/fails
- Each technique also suffers from a unique set of problems

Depth-Pass

- Basic depth-pass technique works as follows:
  - Render **front** faces. If depth test passes **increment** stencil value
  - Render **back** faces. If depth test passes **decrement** stencil value
  - If stencil value is non-zero, then the pixel is in shadow

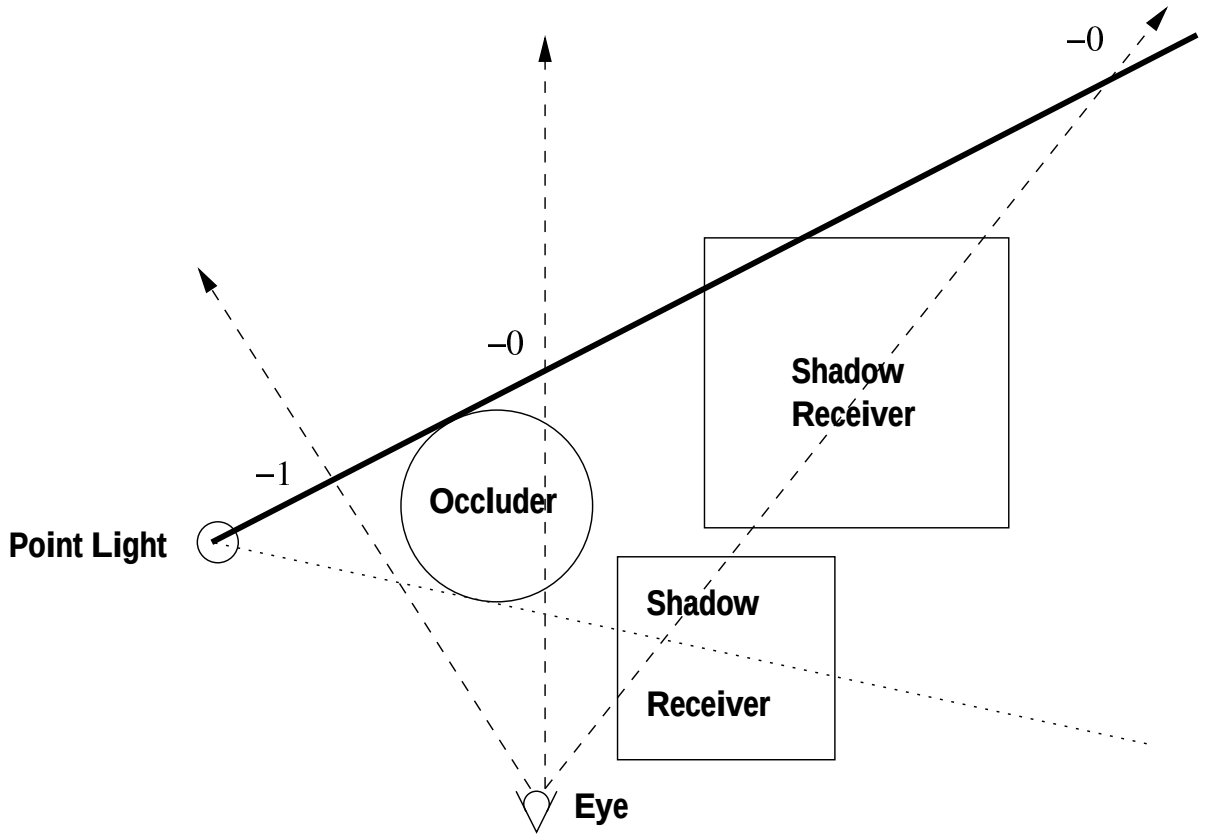
Depth-Pass Example

- Render **front** faces. If depth test passes **increment** stencil value



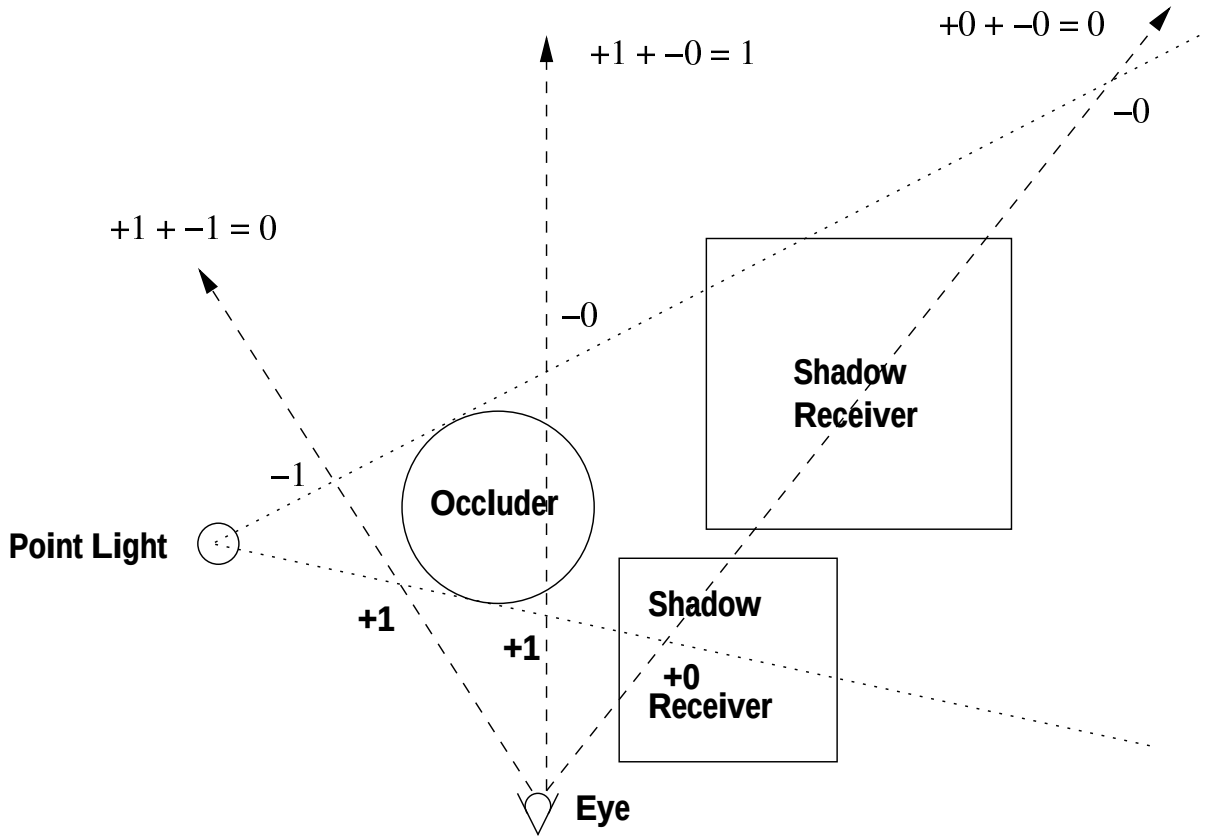
Depth-Pass Example Cont.

- Render **back** faces. If depth test passes **decrement** stencil value



Depth-Pass Example Cont.

- If stencil value is non-zero, then the pixel is in shadow



Depth-Pass Problems

- Occluders may be clipped by near clip plane (far clip plane is not a problem)
- Incorrect stencil values if eye enters shadow volume
- Solving the near clip plane problem is not trivial
- Solving the eye in shadow volume problem requires a slight change in the depth-pass algorithm...

Depth-Fail

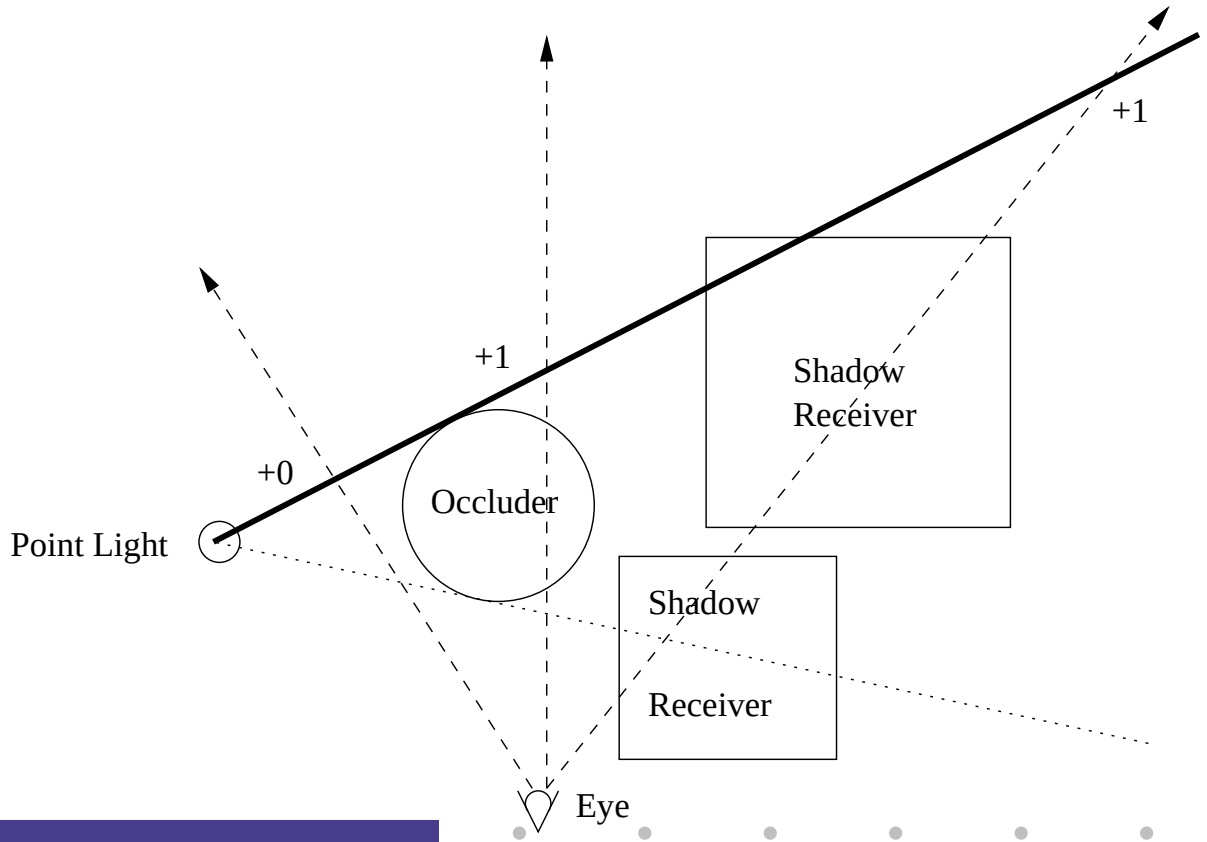
- Discovered independently by both John Carmack and the team of Bill Bilodeau and Mike Songy
- Often referred to as "Carmack's Reverse"
- Produces correct stencil values when eye enters shadow volume

Depth-Fail Cont.

- Basic depth-fail technique works as follows:
  - Render **back** faces. If depth test fails **increment** stencil value
  - Render **front** faces. If depth test fails **decrement** stencil value
  - If stencil value is non-zero, then the pixel is in shadow

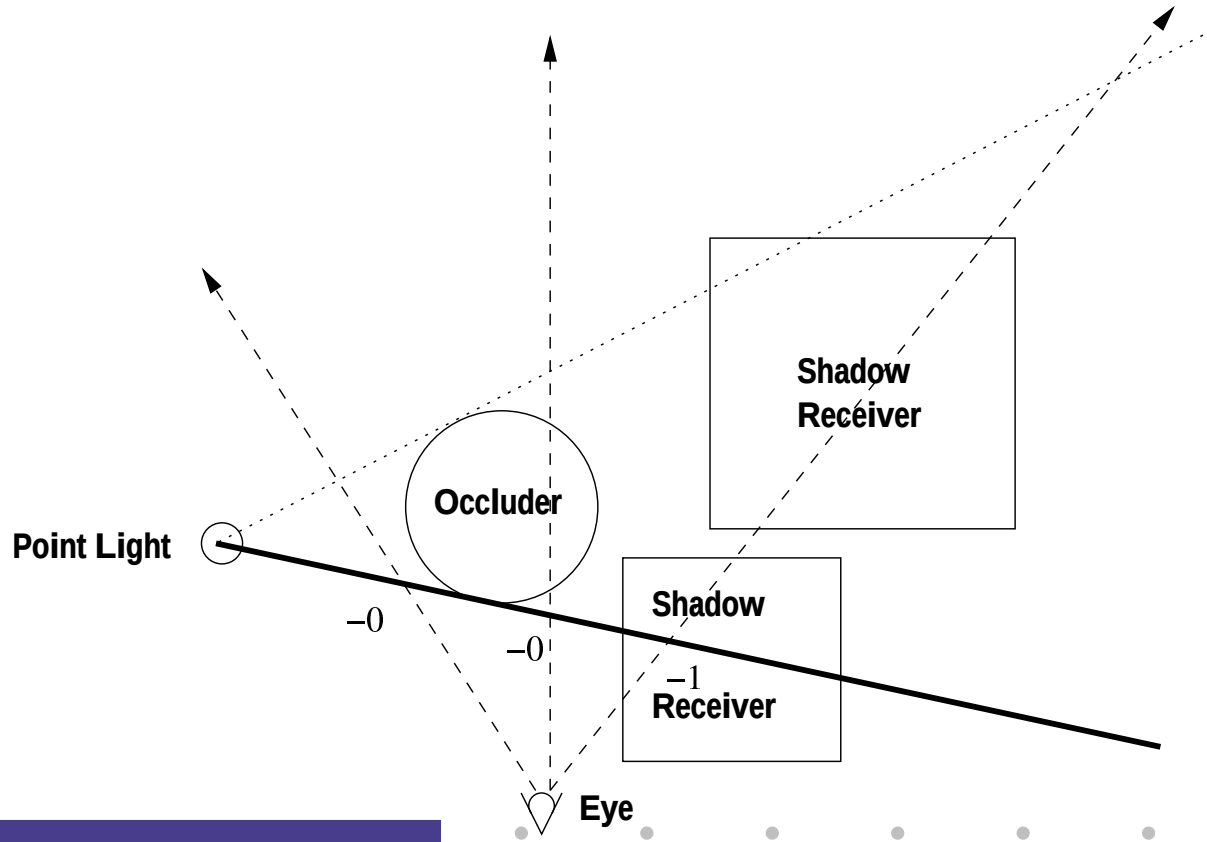
Depth-Fail Example

- Render **back** faces. If depth test fails **increment** stencil value



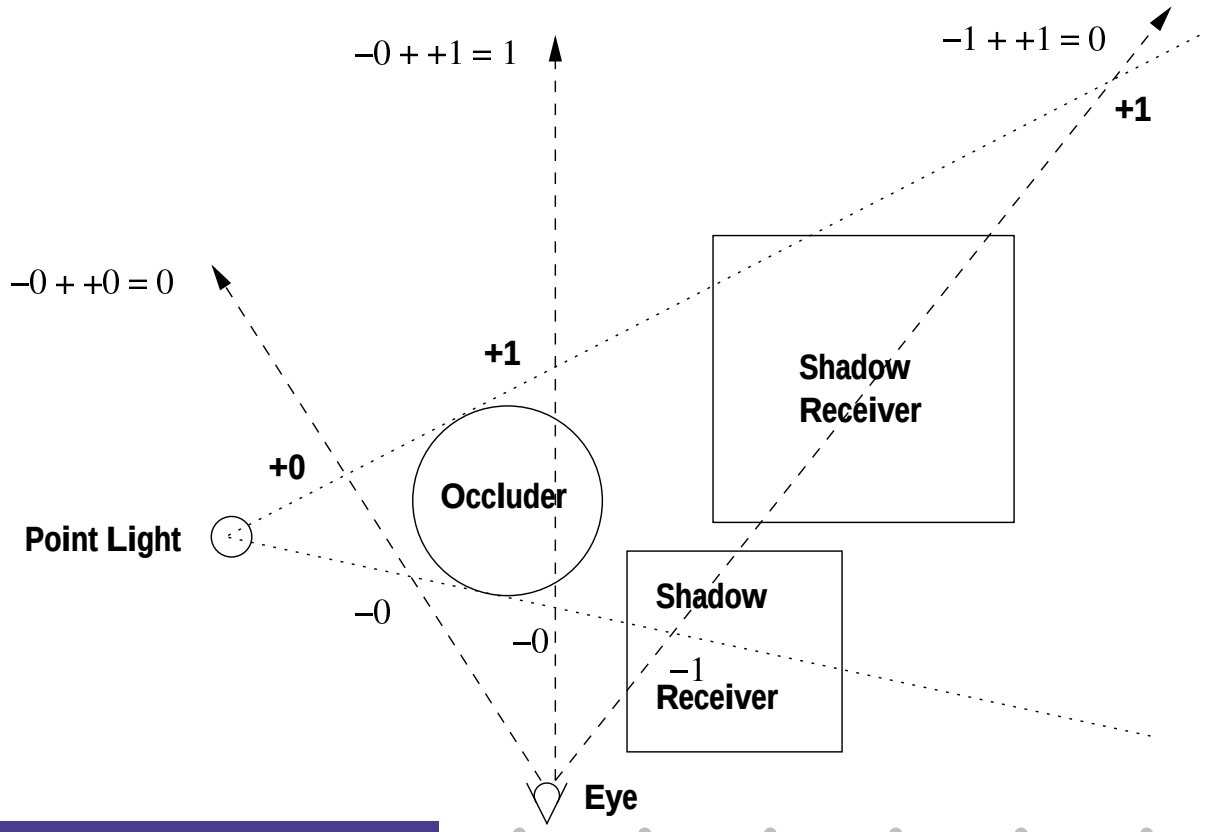
Depth-Fail Example Cont.

- Render **front** faces. If depth test fails **decrement** stencil value



Depth-Fail Example Cont.

- If stencil value is non-zero, then the pixel is in shadow



Depth-Fail Problem: Capping

- Need to cap shadow volume to produce correct stencil value results
- Caps need to be outward facing

Generating Shadow Volume Caps

- Front Cap: Simply use the occluders front facing polygons
- Back cap
  - Extrude occluders front faces to infinity
  - Create a triangle strip at inifinity

Another Depth-Fail Problem: Clipping

- Far clip plane may clip shadow volume
- This causes erroneous results
- Solution: Place the far clip plane at inifinity

Placing Far Clip Plane at Infinity

- Suggested by Mark Kilgard
- Change perspective matrix to:

$$\begin{bmatrix} \frac{2*Near}{Right-Left} & 0 & \frac{Right+Left}{Right-Left} & 0 \\ 0 & \frac{2*Near}{Top-Bottom} & \frac{Top+Bottom}{Top-Bottom} & 0 \\ 0 & 0 & -1 & -2*Near \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Uber Shadows

- `mesh3` contains triangle/edge connectivity information
- `mesh3::render_shadow_volume` renders the mesh's shadow volume in respect to a given light
- `demos/meshviewer/q2` demonstrates depth-pass technique

Questions?