Shadow Volumes

Shadow Volume History (1)
- Invented by Frank Crow ['77]
  - Software rendering scan-line approach
- Brotman and Badler ['84]
  - Software-based depth-buffered approach
  - Used lots of point lights to simulate soft shadows
- Pixel-Planes [Fuchs, et.al. '85] hardware
  - First hardware approach
  - Point within a volume, rather than ray intersection
- Bergeron ['96] generalizations
  - Explains how to handle open models
  - And non-planar polygons

Shadow Volume History (2)
- Fournier & Fussell ['88] theory
  - Provides theory for shadow volume counting approach within a frame buffer
- Akeley & Foran invent the stencil buffer
  - IRIS GL functionality, later made part of OpenGL 1.0
  - Patent filed in '92
- Heidmann [IRIS Universe article, '91]
  - IRIS GL stencil buffer-based approach
- Deifenbach’s thesis ['96]
  - Used stenciled volumes in multi-pass framework

Shadow Volume History (3)
- Dietrich slides [March '99] at GDC
  - Proposes zfail based stenciled shadow volumes
- Kilgard whitepaper [March '99] at GDC
  - Invert approach for planar cut-outs
- Bilodeau slides [May '99] at Creative seminar
  - Proposes way around near plane clipping problems
  - Reverses depth test function to reverse stencil volume ray intersection sense
- Carmack [unpublished, early 2000]
  - First detailed discussion of the equivalence of zpass and zfail stenciled shadow volume methods

Shadow Volume History (4)
- Kilgard [2001] at GDC and CEDEC Japan
  - Proposes zpass capping scheme
    - Project back-facing (w.r.t. light) geometry to the near clip plane for capping
    - Establishes near plane ledge for crack-free near plane capping
    - Applies homogeneous coordinates (w=0) for rendering infinite shadow volume geometry
  - Requires much CPU effort for capping
  - Not totally robust because CPU and GPU computations will not match exactly, resulting in cracks

Shadow Volume Basics

A shadow volume is simply the half-space defined by a light source and a shadowing object.
Shadow Volume Basics (2)

- Simple rule: samples within a shadow volume are in shadow.
- Partially shadowed object
- Surface inside shadow volume (shadowed)
- Surface outside shadow volume (illuminated)

Simple rule:
samples within a shadow volume are in shadow.

- Surface inside shadow volume (shadowed)
- Surface outside shadow volume (illuminated)

Shadow Quality: Shadow Maps

- Draw polygons along boundary of region in shadow (occluders)
- Along ray from eye to first visible surface:
  - Count up for in event
  - Count down for out events
  - If result zero when surface hit, is lit
- Can be implemented with stencil buffer
- Near/far plane clip causes problems

Shadow Quality: Stencil Shadow Volumes

- Draw polygons along boundary of region in shadow (occluders)
- Along ray from eye to first visible surface:
  - Count up for in event
  - Count down for out events
  - If result zero when surface hit, is lit
- Can be implemented with stencil buffer
- Near/far plane clip causes problems

Shadow Volumes

- Draw polygons along boundary of region in shadow (occluders)
- Along ray from eye to first visible surface:
  - Count up for in event
  - Count down for out events
  - If result zero when surface hit, is lit
- Can be implemented with stencil buffer
- Near/far plane clip causes problems

Shadow Volume Advantages

- Omni-directional approach
  - Not just spotlight frustums as with shadow maps
- Automatic self-shadowing
  - Everything can shadow everything, including self
  - Without shadow acne artifacts as with shadow maps
- Window-space shadow determination
  - Shadows accurate to a pixel
  - Or sub-pixel if multisampling is available
- Required stencil buffer broadly supported today
  - OpenGL support since version 1.0 (1991)
  - Direct3D support since DX8 (1998)

Point Inside 2D Polygon
Point Inside 2D Polygon

• Infinite “polygon”
• Union of polygons
• Line segment

Optimizing shadow volumes

Use silhouette edges only

Shadow volumes [Crow77]

• Shadow volumes define closed volumes of space that are in shadow
Step 1: Render scene

- Front face: +1
- Back face: ±0 (Depth test)

Σ = ±0

Step 2: Render shadow volume faces

- Front face: +1
- Back face: ±0 (Depth test)

Σ = +1

Σ = ±0

Step 3: Apply shadow mask to scene
Shadow Volumes w/ Stencils (Zpass)

- Details of the basic algorithm:
  - Compute shadow volumes
  - View-independent
  - Clear stencil buffer
  - Render the scene without (diffuse) specular lighting (ambient only)
  - Sets the Depth Buffer and color buffer
  - "Render" front faces of shadow volumes
    - Turn off color, depth updates (but leave depth test on)
    - Visible polygons increment pixel stencil buffer count
    - Increment when depth test passes
  - "Render" back faces of shadow volumes
    - Turn off color, depth updates (but leave depth test on)
    - Visible polygons decrement pixel stencil buffer count
    - Decrement when depth test passes
  - Render scene with only diffuse/spec lighting
    - Only update pixels where stencil = 0
    - Others are in shadow (ambient only)!!
Shadow Volumes (Zfail)

- Details of the basic algorithm:
  - Compute shadow volumes
  - View-independent
  - Clear stencil buffer
  - Render the scene without diffuse/spec lighting
    - Sets the Depth Buffer and Color Buffer
    - "Render" back faces of shadow volumes
      - Visible polygons increment pixel stencil buffer count
      - Increment when depth test fails
    - "Render" front faces of shadow volumes
      - Turn off color, depth updates (but leave depth test on)
      - Visible polygons decrement pixel stencil buffer count
      - Decrement when depth test fails
    - Render scene with only diffuse/spec lighting
      - Only update pixels where stencil = 0
      - Others are in shadow (ambient only)!

Zfail versus Zpass Comparison

- When stencil increment/decrements occur
  - Zpass: on depth test pass
  - Zfail: on depth test fail
- Increment on
  - Zpass: front faces
  - Zfail: back faces
- Decrement on
  - Zpass: back faces
  - Zfail: front faces

Illuminated, Behind Shadow Volumes (Zfail)

Light source

Shadowing object

Eye position

Shadow Volume Count = 0 (zero depth tests fail)

Zfail versus Zpass Comparison

- Both cases order passes based stencil operation
  - First, render increment pass
  - Second, render decrement pass
- Why?
  - Because standard stencil operations saturate
  - Wrapping stencil operations can avoid this
- Which clip plane creates a problem
  - Zpass: near clip plane
  - Zfail: far clip plane
- Either way is foiled by view frustum clipping
  - Which clip plane (front or back) changes

Shadowed, Nested in Shadow Volumes (Zfail)

Light source

Shadowing object

Eye position

Shadow Volume Count = +1+1 = 2

Illuminated, In Front of Shadow Volumes (Zfail)

Light source

Shadowing object

Eye position

Shadow Volume Count = -1-1+1+1+1+1 = 0
Nested Shadow Volumes
Stencil Counts Beyond One

Shadowed scene
Stencil buffer contents

Amount of pixel processing

Adapted from [Chan and Durand 2004]

Shadows in a Real Game Scene

Scene’s Visible
Geometric Complexity

Wireframe shows geometric complexity of visible geometry

Primary light source location

Blow-up of Shadow Detail

Scene’s Shadow Volume
Geometric Complexity

Wireframe shows geometric complexity of shadow volume geometry

Shadow volume geometry projects away from the light source
Visible Geometry vs. Shadow Volume Geometry

Typically, shadow volumes generate considerably more pixel updates than visible geometry.

Other Example Scenes (1 of 2)

Dramatic chase scene with shadows

Other Example Scenes (2 of 2)

Scene with multiple light sources

Shadow Volumes Too Expensive

Fuel game image courtesy Nathan d’Obrenan at Firetoad Software

Shadow Volume Advantages

- Omni-directional approach
  - Not just spotlight frustums as with shadow maps
- Automatic self-shadowing
  - Everything can shadow everything, including self
  - Without shadow acne artifacts as with shadow maps
- Window-space shadow determination
  - Shadows accurate to a pixel
  - Or sub-pixel if multisampling is available
- Required stencil buffer broadly supported today
  - OpenGL support since version 1.0 (1991)
  - Direct3D support since DX6 (1996)

Shadow Volume Disadvantages

- Ideal light sources only
  - Limited to local point and directional lights
  - No area light sources for soft shadows
- Requires polygonal models with connectivity
  - Models must be closed (2-manifold)
  - Models must be free of non-planar polygons
- Silhouette computations are required
  - Can burden GPU
  - Particularly for dynamic scenes
- Inherently multi-pass algorithm
- Consumes lots of GPU fill rate
Shadows: Volumes vs. Maps

- Shadow mapping via projective texturing
  - The other prominent hardware-accelerated shadow technique
  - Standard part of OpenGL 1.4
- Shadow mapping advantages
  - Requires no explicit knowledge of object geometry
  - No 2-manifold requirements, etc.
  - View independent
- Shadow mapping disadvantages
  - Sampling artifacts
  - Not omni-directional

Issues with Shadow Volumes

Stencil Shadow Pros

- Very accurate and robust
- Nearly artifact-free
  - Faceting near the silhouette edges is the only problem
- Work for point lights and directional lights equally well
- Low memory usage

Stencil Shadow Cons

- Too accurate — hard edges
  - Need a way to soften
- Very fill-intensive
  - Scissor and depth bounds test help
- Significant CPU work required
  - Silhouette determination
  - Building shadow volumes

Stencil Shadow Volume Optimizations (1)

- Use `GL_QUAD_STRIP` rather than `GL_QUADS` to render extruded shadow volume quads
  - Requires determining possible silhouette loop connectivity
- Mix Zfail and Zpass techniques
  - Pick a single formulation for each shadow volume
  - Zpass is more efficient since shadow volume does not need to be closed
  - Mixing has no effect on net shadow depth count
  - Zfail can be used in the hard cases

Stencil Shadow Volume Optimizations (2)

- Pre-compute or re-use cache shadow volume geometry when geometric relationship between a light and occluder does not change between frames
  - Example: Headlights on a car have a static shadow volume w.r.t. the car itself as an occluder
- Advanced shadow volume culling approaches
  - Uses portals, Binary Space Partitioning trees, occlusion detection, and view frustum culling techniques to limit shadow volumes
  - Careful to make sure such optimizations are truly correct
Stenciled Shadow Volume Optimizations (3)

- Take advantage of ad-hoc knowledge of scenes whenever possible
  - Example: A totally closed room means you do not have to cast shadow volumes for the wall, floor, ceiling
- Limit shadows to important entities
  - Example: Generate shadow volumes for monsters and characters, but not static objects
  - Characters can still cast shadows on static objects
- Mix shadow techniques where possible
  - Use planar projected shadows or light-maps where appropriate

Stenciled Shadow Volume Optimizations (4)

- Shadow volume’s extrusion for directional lights can be rendered with a `GL_TRIANGLE_FAN`
  - Directional light’s shadow volume always projects to a single point at infinity

Hardware Enhancements: Wrapping Stencil Operations

- Conventional OpenGL 1.0 stencil operations
  - `GL_INCR` increments and clamps to $2^N - 1$
  - `GL_DECR` decrements and clamps to zero
- DirectX 6 introduced “wrapping” stencil operations
  - Exposed by OpenGL’s stencil wrap
    - `GL_INCR_WRAP` increments modulo $2^N$
    - `GL_DECR_WRAP` decrements modulo $2^N$
  - Avoids saturation throwing off the shadow volume depth count
    - Still possible, though very rare, that $2^N$, $2 \cdot 2^N$, $3 \cdot 2^N$, etc. can alias to zero

Hardware Enhancements: Two-sided Stencil Testing (1)

- Past stenciled shadow volumes required rendering shadow volume geometry twice
  - First, rasterizing front-facing geometry
  - Second, rasterizing back-facing geometry
- Two-sided stencil testing requires only one pass
  - Two sets of stencil state: front- and back-facing
  - Boolean enable for two-sided stencil testing
  - When enabled, back-facing stencil state is used for stencil testing back-facing polygons
  - Otherwise, front-facing stencil state is used
  - Rasterizes just as many fragments, but more efficient for CPU & GPU

Hardware Enhancements: Two-sided Stencil Testing (2)

- `glStencilMaskSeparate` and `glStencilOpSeparate(face, fail, zfail, zpass)`
- `glStencilFuncSeparate(face, func, ref, mask)`
  - Control of front- and back-facing stencil state update

Performance

- Have to render lots of huge polygons
  - Front face increment
  - Back face decrement
  - Possible capping pass
- Burns fill rate like crazy
- Turn off depth and color write, though
- Gives accurate shadows
  - IF implemented correctly
  - When fails, REALLY fails
- Need access to geometry if want to use silhouette optimization
Slide Credits

• Cass Everitt & Mark Kilgard, NVidia – GDC 2003 presentation
• Timo Aila, Helsinki U. Technology
• Jeff Russell
• David Luebke, University of Virginia
• Michael McCool, University of Waterloo
• Eric Lengyel, Naughty Dog Games

These are extra slides

• Hacks to further improve shadow volumes

Scissor Optimizations

• Most important fill-rate optimization for stencil shadows
• Even more important for penumbral wedge shadows
• Hardware does not generate fragments outside the scissor rectangle — very fast

Scissor Optimizations

• Scissor rectangle can be applied on a per-light basis or even a per-geometry basis
• Requires that lights have a finite volume of influence
  – What type of light is this?

Light Scissor

[Diagram of Light Scissor]
Light Scissor
• Project light volume onto the image plane
• Intersect extents with the viewport to get light’s scissor rectangle
• Mathematical details at:
  – http://www.gamasutra.com/features/20021011/lengyel_01.htm

No Light Scissor
Shadow volumes extend to edges of viewport

Light Scissor
Shadow volume fill reduced significantly

Depth Bounds Test
• Like a z scissor, but...
• Operates on values already in the depth buffer, not the depth of the incoming fragment
• Saves writes to the stencil buffer when shadow-receiving geometry is out of range

Depth Bounds Test
• Shadow volumes extend closer to and further from camera than necessary
• Shadow volume fill outside depth bounds is removed
No Depth Bounds Test
A lot of extra shadow volume fill where we know it can’t have any effect

Depth Bounds Test
Parts that can’t possibly intersect the environment removed

Depth Bounds Test
• Let $P$ be the projection matrix and let $[d_{\text{min}},\ d_{\text{max}}]$ be the depth range
• Viewport depth $d$ corresponding to camera space $z$ is given by

$$d = \frac{d_{\text{max}} - d_{\text{min}}}{2}(P_{33}z + P_{34}) + \frac{d_{\text{max}} + d_{\text{min}}}{2}$$

Geometry Scissor
• We can do much better than a single scissor rectangle per light
• Calculate a scissor rectangle for each geometry casting a shadow

Geometry Scissor
• Define a bounding box for the light
  – Doesn’t need to contain the entire sphere of influence, just all geometry that can receive shadows
  – For indoor scenes, the bounding box is usually determined by the locations of walls
Geometry Scissor

• For each geometry, define a simple bounding polyhedron for its shadow volume
  – Construct a pyramid with its apex at the light's position and its base far enough away to be outside the light's sphere of influence
  – Want pyramid to be as tight as possible around geometry

Geometry Scissor

• Clip shadow volume’s bounding polyhedron to light's bounding box
• Project vertices of resulting polyhedron onto image plane
• This produces the boundary of a much smaller scissor rectangle
• Also gives us a much smaller depth bounds range
No Geometry Scissor
- Light scissor rectangle and depth bounds test are no help at all in this case

Geometry Scissor
- Shadow volume fill drastically reduced

Scissor and Depth Bounds
- Performance increase for ordinary stencil shadows not spectacular
- Real-world scenes get about 5-8% faster using per-geometry scissor and depth bounds test
- Hardware is doing very little work per fragment, so reducing number of fragments is not a huge win

Scissor and Depth Bounds
- For penumbral wedge rendering, it’s a different story
- We will do much more work per fragment, so eliminating a lot of fragments really helps
- Real-world scenes can get 40-45% faster using per-geometry scissor and depth bounds test