Shadow Volumes

Why Shadow Volumes?

• Dynamic shadows improve your game
  – Dramatic effects
  – Better sense of 3D

• Most game shadows today are very limited
  – Planar projected shadows [Blinn ’88]
  – Limited to floor planes, perhaps walls

Not good enough for today’s games!

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

• Biodeau 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• Carmack [unpublished, early 2000]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.

Shadow Quality: “Blobs”

Surface outside shadow volume (illuminated)

Surface inside shadow volume (shadowed)

Shadow Quality: Shadow Maps

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

Shadow Quality: Stencil Shadow Volumes

Surface outside shadow volume (illuminated)

Surface inside shadow volume (shadowed)

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 1.2 (1991)
  - Direct3D support since DX6 (1998)

Point Inside 2D Polygon

- Infinite "polygon"
- Union of polygons
- Line segment
Optimizing shadow volumes

- Use silhouette edges only

Shadow volumes [Crow 77]

- Shadow volumes define closed volumes of space that are in shadow

Step 1: Render scene → Z-values

Step 2: Render shadow volume faces

Σ = ±0

Front face: +1

Back face: ±0 (Depth test)

Σ = +1
Shadow Volumes [Crow 77]
The shadow mask can also be computed in the alpha-channel which performs even faster than the original algorithm.

Then the shadow mask can be copied efficiently into an alpha texture map and applied afterwards.

**Advantages:**
- Scene is rendered only once for quick/dirty shadows.
- Computation of shadow mask with lower resolution than screen buffer → shadow mask is rasterized much faster.

### Shadow Volumes without Stencils
- Efficient computation of dynamic shadows possible without stencil buffer.
- Shadow mask is computed either in screen buffer or in alpha-channel (PS2).
- **Idea:** Utilize *2, 2 operations instead of +1, -1.
- Different modes of application: Black, ambient, or quick/dirty shadows (scene rendered only once in the latter case).
- By copying the shadow mask into a alpha-texture the shadow mask can be computed at lower resolutions than the screen buffer ⇒ overcome rasterization bottleneck.

### Shadow Volumes w/ Stencils (Zpass)
- Details of the basic algorithm:
  - Compute shadow volumes
    - View-independent
  - Clear stencil buffer
  - Render the scene without diffuse/spec 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)
Illuminated, Behind Shadow Volumes (Zpass)

[Diagram showing light source, shadowing object, and eye position.]

Shadow Volume Count = +1+1+1-1-1-1 = 0

Problems Created by Near Clip Plane (Zpass)

[Diagram showing missed shadow volume intersection due to near clip plane clipping, leading to mistaken count.]

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
      - Turn off color, depth updates (but leave depth test on)
      - 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
      - Decrease when depth test passes
    - 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 increments/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)

Shadow Volume Count = 0 (zero depth tests fail)

Shadowed, Nested in Shadow Volumes (Zfail)

Shadow Volume Count = +1+1 = 2

Illuminated, In Front of Shadow Volumes (Zfail)

Shadow Volume Count = -1-1+1+1+1+1 = 0

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

Nested Shadow Volumes Stencil Counts Beyond One

green = stencil value of 0
red = stencil value of 1
darker reds = stencil value > 1
Scene’s Visible Geometric Complexity

- Wireframe shows geometric complexity of visible geometry
- Primary light source location

Blow-up of Shadow Detail

- Notice cable shadows on player model
- Notice player’s own shadow on floor

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
Visible geometry

Shadow Volume

Scene with multiple light sources
Visible geometry

Other Example Scenes (2 of 2)

Visible geometry

Dramatic chase scene with shadows

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 (1998)

Shadow Volume Too Expensive

Chain-link fence's shadow appears on truck & ground with shadow maps

Fuel game image courtesy Nathan d’Oberon at Firetoad Software

Chain-link fence is shadow volume nightmare!

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 CPU
  – Particularly for dynamic scenes
• Inherently multi-pass algorithm
• Consumes lots of GPU fill rate

Stopped here on Tues
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

Hardware Support

- GL_EXT_stencil_two_side
- GL_ATI_separate_stencil_func
  - Both allow different stencil operations to be executed for front and back facing polygons
- GL_EXT_depth_bounds_test
  - Helps reduce frame buffer writes
- Double-speed rendering

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

• 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

Light Scissor

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
Depth Bounds Test

No Depth Bounds Test
Shadow volumes extend closer to and further from camera than necessary.

Depth Bounds Test
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
• Depths bounds specified in viewport coordinates
• To get these from camera space, we need to apply projection matrix and viewport transformation
• Apply to points (0,0,z,1)

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

\[
d = \frac{d_{\text{max}} - d_{\text{min}}}{2} \left( \frac{P_{33}z + P_{34}}{P_{43}z + P_{44}} \right) + \frac{d_{\text{max}} + d_{\text{min}}}{2}
\]

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

Insight!
- If we could avoid either near plane or far plane view frustum clipping, shadow volume rendering could be robust
- Avoiding near plane clipping
  - Not really possible
  - Objects can always be behind you
  - Moreover, depth precision in a perspective view goes to hell when the near plane is too near the eye
- Avoiding far plane clipping
  - Perspective make it possible to render at infinity
  - Depth precision is terrible at infinity, but we just care about avoiding clipping

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
Avoiding Far Plane Clipping

- Usual practice for perspective GL projection matrix
  - Use `glFrustum` (or `gluPerspective`)
  - Requires two values for near & far clip planes
    - Near plane’s distance from the eye
    - Far plane’s distance from the eye
  - Assumes a finite far plane distance
- Alternative projection matrix
  - Still requires near plane’s distance from the eye
  - But assume far plane is at infinity
- What is the limit of the projection matrix when the far plane distance goes to infinity?

\[
\begin{bmatrix}
2 \times \text{Near} & 0 & \text{Right + Left} & 0 \\
0 & 2 \times \text{Near} & \text{Right + Left} & 0 \\
0 & 0 & \text{Top + Bottom} & 0 \\
0 & 0 & 0 & -1 \\
\end{bmatrix}
\]

- Only third row depends on Far and Near

Limit of `glFrustum` Projection Matrix as Far Plane is Moved to \(\infty\)

\[
\lim_{\text{Far} \to \infty} P = P_{\text{inf}} =
\begin{bmatrix}
2 \times \text{Near} & 0 & \text{Right + Left} & 0 \\
0 & 2 \times \text{Near} & \text{Right + Left} & 0 \\
0 & 0 & \text{Top + Bottom} & 0 \\
0 & 0 & 0 & -1 \\
\end{bmatrix}
\]

- First, second, and fourth rows are the same as in \(P\)
- But third row no longer depends on \(\text{Far}\)
- Effectively, \(\text{Far}\) equals \(\infty\)

Verifying \(P_{\text{inf}}\) Will Not Clip Infinitely Far Away Vertices (1)

- What is the most distant possible vertex in front of the eye?
  - Ok to use homogeneous coordinates
  - OpenGL convention looks down the negative Z axis
  - So most distant vertex is \((0,0,-D,0)\) where \(D > 0\)
- Transform \((0,0,-D,0)\) to window space
  - Is such a vertex clipped by \(P_{\text{inf}}\)?
  - No, it is not clipped, as explained on the next slide

Verifying \(P_{\text{inf}}\) Will Not Clip Infinitely Far Away Vertices (2)

- Transform eye-space \((0,0,-D,0)\) to clip-space

\[
\begin{bmatrix}
1 & x & y & z \\
0 & 0 & 0 & -D \\
\end{bmatrix}
\begin{bmatrix}
2 \times \text{Near} & 0 & \text{Right + Left} & 0 \\
0 & 2 \times \text{Near} & \text{Right + Left} & 0 \\
0 & 0 & \text{Top + Bottom} & 0 \\
0 & 0 & 0 & -1 \\
\end{bmatrix}
\begin{bmatrix}
x' \\
y' \\
-w \\
\end{bmatrix}
\]

- Then, assuming `glDepthRange(0,1)`, transform clip-space position to window-space position

\[
z_w = 0.5 \times \frac{z'}{w'} + 0.5 \times \frac{-D}{D} + 0.5 = 1
\]

- So \(z\) in front of eye transforms to the maximum window-space \(Z\) value, but is still within the valid depth range (i.e., not clipped)

Is \(P_{\text{inf}}\) Bad for Depth Buffer Precision?

- Naïve question
  - Wouldn’t moving the far clip plane to infinity waste depth buffer precision? Seems plausible, but
- Answer: Not really
  - Minimal depth buffer precision is wasted in practice
  - This is due to projective nature of perspective
- Say Near is 1.0 and Far is 100.0 (typical situation)
  - \(P\) would transform eye-space infinity to only 1.01 in window space
  - Only a 1% compression of the depth range is required to render infinity without clipping
  - Moving near closer would hurt precision
**P_{\inf} Depth Precision Scale Factor**

- **Using** $P_{\inf}$ with Near instead of $P$ with Near and Far **compresses** (scales) the depth precision by $\frac{(\text{Far} - \text{Near})}{\text{Far}}$
  - The compression of depth precision is uniform, but the depth precision itself is already non-uniform on eye-space interval [Near, Far] due to perspective
  - So the discrete loss of precision is more towards the far clip plane
- Normally, Far >> Near so the scale factor is usually less than but still nearly 1.0
- So the compression effect is minor

**Robust Stenciled Shadow Volumes w/o Near (or Far) Plane Capping**

- **Use** Zfail Stenciling Approach
  - Must render geometry to close shadow volume extrusion on the model and at infinity (explained later)
- **Use** the $P_{\inf}$ Projection Matrix
  - No worries about far plane clipping
  - Losses some depth buffer precision (but not much)
- **Draw** the infinite vertices of the shadow volume using homogeneous coordinates (w=0)

**Rendering Closed, but Infinite, Shadow Volumes**

- To be robust, the shadow volume geometry must be closed, even at infinity
- Three sets of polygons close the shadow volume
  1. Possible silhouette edges extruded to infinity away from the light
  2. All of the occluder’s back-facing (w.r.t. the light) triangles projected away from the light to infinity
  3. All of the occluder’s front-facing (w.r.t. the light) triangles
- We assume the object vertices and light position are homogeneous coordinates, i.e. $(x,y,z,w)$
  - Where $w\geq 0$

**1st Set of Shadow Volume Polygons**

- Assuming $A$ and $B$ are vertices of an occluder model’s possible silhouette edge
  - And $L$ is the light position
- For all $A$ and $B$ on silhouette edges of the occluder model, render the quad
  
  $\{A, B, L, B\}$
  $\{A, L, A, L\}$
  $\{B, L, B, L\}$
  $\{-L, A, -L, A\}$
  $\{-L, B, -L, B\}$
  $\{-L, L, -L, L\}$

- What is a possible silhouette edge?
  - One polygon sharing an edge faces toward $L$
  - Other faces away from $L$

**2nd and 3rd Set of Shadow Volume Polygons**

- **2nd set of polygons**
  - Assuming $A$, $B$, and $C$ are each vertices of occluder model’s back-facing triangles w.r.t. light position $L$
  
  $\{A, L, A, L\}$
  $\{B, L, B, L\}$
  $\{C, L, C, L\}$

- These vertices are effectively directions (w=0)

- **3rd set of polygons**
  - Assuming $A$, $B$, and $C$ are each vertices of occluder model’s front-facing triangles w.r.t. light position $L$
  
  $\{A, A, A, A\}$
  $\{B, B, B, B\}$
  $\{C, C, C, C\}$

**Stenciled 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
Stenciled 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 car 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 EXT_stencil_wrap extension
  - GL_INCR_WRAP_EXT increments modulo $2^N$
  - GL_DECR_WRAP_EXT decrements modulo $2^N$
- Avoids saturation throwing off the shadow volume depth count
  - Still possible, though very rare, that $2^N$, $2 	imes 2^N$, $3 	imes 2^N$, etc. can alias to zero

Hardware Enhancements: Depth Clamp (1)

- What is depth clamping?
  - Boolean hardware enable/disable
  - When enabled, disables the near & far clip planes
  - Interpolate the window-space depth value
  - Clamps the interpolated depth value to the depth range, i.e. $[\min(n,f), \max(n,f)]$
    - Assuming glDepthRange(n,f);
  - Geometry “behind” the far clip plane is still rendered
  - Depth value clamped to farthest Z
  - Similar for near clip plane, as long as $w > 0$, except clamped to closest Z

Hardware Enhancements: Depth Clamp (2)

- Advantage for stenciled shadow volumes
  - With depth clamp, P (rather than Pinf) can be used with our robust stenciled shadow volume technique
  - Marginal loss of depth precision re-gained
  - Orthographic projections can work with our technique with depth clamping
- NV_depth_clamp OpenGL extension
  - Easy to use
    - glEnable(GL_DEPTH_CLAMP_NV);
    - glDisable(GL_DEPTH_CLAMP_NV);
  - GeForce3 & GeForce4 Ti support (soon)
Hardware Enhancements: Two-sided Stencil Testing (1)

- Current 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)

NV_stencil_two_side OpenGL extension
- Enable applies if GL_STENCIL_TEST also enabled
  ```
glEnable(GL_STENCIL_TEST_TWO_SIDE_NV);
glDisable(GL_STENCIL_TEST_TWO_SIDE_NV);
```
- Control of front- and back-facing stencil state update
  ```
glActiveStencilFaceNV(GL_FRONT);
glActiveStencilFaceNV(GL_BACK);
```
- Existing stencil routines (glStencilOp, glStencilMask, glStencilFunc) update the active stencil face state
- glClear and non-polygon primitives always use the front-facing stencil state
- Expect on future GPUs

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

Shadow Volumes

- Basic idea:
  - Create polygonal objects to represent shadowed volumes
  - Make clever use of stencil buffer so that these objects affect what lighting is done

Stencil Buffer

- The stencil buffer has been around since OpenGL 1.0
  - Basic idea: provide a per-pixel flag to indicate whether pixels are drawn or not
  - But...
  - Let that flag be an integer (usually 8 bits)
    - Usually shared with depth buffer
  - And let drawing operations increment or decrement the stencil buffer based on different events
    - Always, depth-pass, depth-fail, etc.

Shadow Volumes

- Refinements (see book, next slides)
  - NV30, XBox supports signed stencil addition
    - Two-sided lighting determines whether polygon adds or subtracts 1 from stencil buffer
    - One-pass algorithm! But a little slower in practice?
    - What if you’re inside a shadow volume?
    - Invert meaning of stencil test
  - What if near clip intersects shadow plane?
    - Carmack, others: use z-fail test
    - Clever extensions in NV2X help this idea out
  - Creating shadow volumes: vertex program!
    - ATI: clever degenerate-edge trick again
Shadow Volumes

• Advantages:
  – Robust
  – Self-shadowing
  – GPU

• Disadvantages:
  – Can be geometry limited
    • Stencil polys
    • Multi-pass scene geometry
  – Can be fill limited
  – Stencil test – per pixel expense
  – Hard shadows

• McCool

Near Plane Clip Issues

• Near plane clip discards part of shadow volume
• Can see inside, messes up count
• Can draw “caps”
  – Use projected shadows on near plane
  – Not exact, get little pixel dropouts
• Better: do another pass, see where can see inside of shadow volume
• Only do extra pass when volume intersects visible near plane
• Or, use z clamping when rendering…
• Reversal of z test can help

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