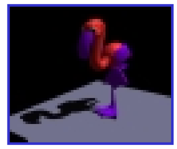
## Practical & Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering

Cass Everitt & Mark J. Kilgard

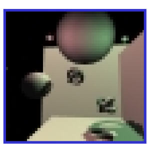
## Why Shadow Volumes?

- 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



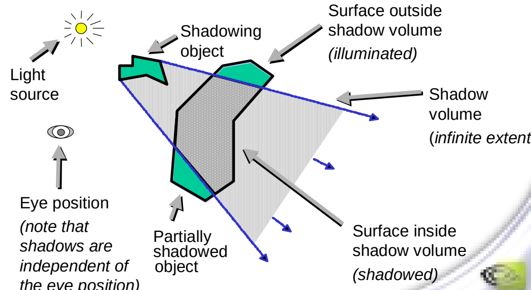
Not good enough for today's games!

## Stenciled Shadow Volumes in Practice



Notice the proper self-shadowing!

## 2D Cutaway of a Shadow Volume




## 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 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

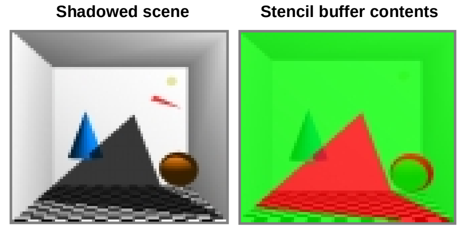
## Visualizing Shadow Volumes in 3D

- Occluders and light source cast out a shadow volume
  - Objects within the volume should be shadowed


Scene with shadows from an NVIDIA logo casting a shadow volume

Visualization of the shadow volume

## Visualizing the Stencil Buffer Counts




Stencil counts beyond 1 are possible for multiple or complex occluders.

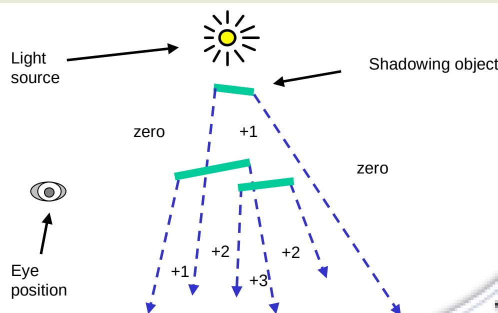
*red = stencil value of 1*
*green = stencil value of 0*

GLUT *shadowvol* example credit: Tom McReynolds

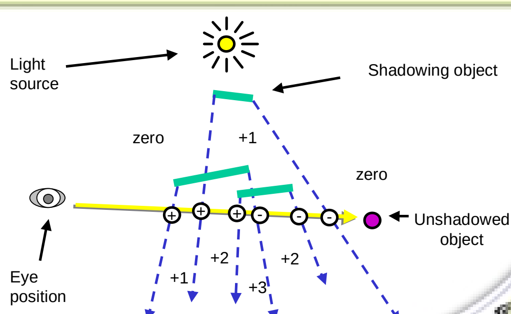
## Counting Shadow Volume Enter/Leaves With a Stencil Buffer (*Zpass* approach)

- Render scene to initialize depth buffer
  - Depth values indicate the closest visible fragments
- Use a stencil enter/leave counting approach
  - Draw shadow volume twice using face culling
    - 1st pass: render ***front*** faces and ***increment*** when depth test passes
    - 2nd pass: render ***back*** faces and ***decrement*** when depth test passes
  - Don't update depth or color
- Afterward, pixel's stencil is non-zero if pixel in shadow, and zero if illuminated

## Why Eye-to-Object Stencil Enter/Leave Counting Approach Works

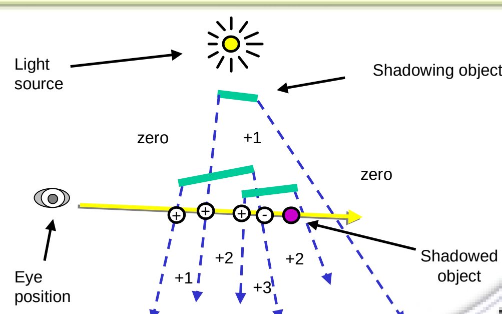



## Illuminated, Behind Shadow Volumes



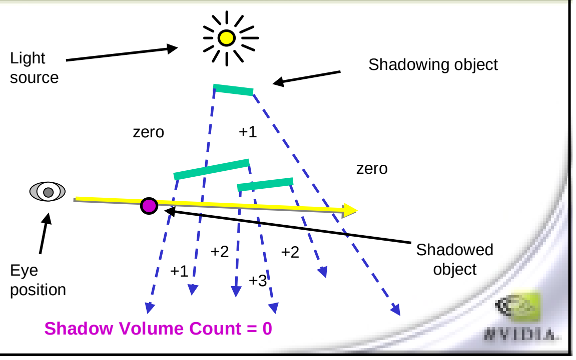

**Shadow Volume Count = +1+1+1-1-1-1 = 0**

## Shadowed, Nested in Shadow Volumes

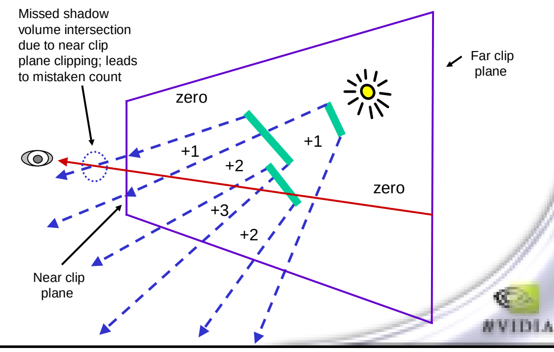



**Shadow Volume Count = +1+1+1-1 = 2**

## Illuminated, In Front of Shadow Volumes




## Problems Created by Near Plane Clipping (*Zpass approach*)




## Alternative Approach: *Zfail*

- Render scene to initialize depth buffer
  - Depth values indicate the closest visible fragments
- Use a stencil enter/leave counting approach
  - Draw shadow volume twice using face culling
    - 1st pass: render <u>*back*</u> faces and <u>*increment*</u> when depth test <u>fails</u>
    - 2nd pass: render <u>*front*</u> faces and <u>*decrement*</u> when depth test <u>fails</u>
  - Don't update depth or color
- Afterward, pixel's stencil is non-zero if pixel in shadow, and zero if illuminated

## *Zfail* versus *Zpass* Comparison (1)

- 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:* front faces
  - *Zfail:* back faces

## *Zfail* versus *Zpass* Comparison (2)

- 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

## 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

## 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?

## Standard *glFrustum* Projection Matrix

$$\mathbf{P} = \begin{bmatrix} \frac{2\times Near}{Right-Left} & 0 & \frac{Right+Left}{Right-Left} & 0 \\ 0 & \frac{2\times Near}{Top-Bottom} & \frac{Top+Bottom}{Top-Bottom} & 0 \\ 0 & 0 & -\frac{Far+Near}{Far-Near} & -\frac{2\times Far\times Near}{Far-Near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- Only third row depends on *Far* and *Near*

## Limit of *glFrustum* Projection Matrix as Far Plane is Moved to Infinity

$$\lim_{Far\to\infty} \mathbf{P} = \mathbf{P}_{\mathbf{inf}} = \begin{bmatrix} \frac{2\times Near}{Right-Left} & 0 & \frac{Right+Left}{Right-Left} & 0 \\ 0 & \frac{2\times Near}{Top-Bottom} & \frac{Top+Bottom}{Top-Bottom} & 0 \\ 0 & 0 & -1 & -2\times Near \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- First, second, and fourth rows are the same as in P
- But third row *no longer* depends on *Far*
  - Effectively, *Far* equals ∞

## Verifying $P_{inf}$ 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 $P_{inf}$?
  - No, it is not clipped, as explained on the next slide

## Verifying $P_{inf}$ Will Not Clip Infinitely Far Away Vertices (2)

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

$$\begin{bmatrix} x_c \\ y_c \\ -D \\ -D \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} \frac{2\times Near}{Right-Left} & 0 & \frac{Right+Left}{Right-Left} & 0 \\ 0 & \frac{2\times Near}{Top-Bottom} & \frac{Top+Bottom}{Top-Bottom} & 0 \\ 0 & 0 & -1 & -2\times Near \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -D \\ 0 \end{bmatrix}$$

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

$$z_w = 0.5\times\frac{z_c}{w_c} + 0.5 = 0.5\times\frac{-D}{-D} + 0.5 = 1$$

- So ∞ *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 $P_{inf}$ 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 Pinf with *Near* instead of P with *Near* and *Far* compresses (scales) the depth precision by

$$\frac{(Far - Near)}{Far}$$

- 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 Without 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≥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

$$\langle B_x, B_y, B_z, B_w \rangle$$
$$\langle A_x, A_y, A_z, A_w \rangle$$
$$\langle A_x L_w - L_x A_w, A_y L_w - L_y A_w, A_z L_w - L_z A_w, 0 \rangle$$
$$\langle B_x L_w - L_x B_w, B_y L_w - L_y B_w, B_z L_w - L_z B_w, 0 \rangle$$

*Homogenous vector differences*

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

## Examples of Possible Silhouette Edges for Quake2 Models

An object viewed from the same basic direction that the light is shining on the object has an identifiable light-view silhouette

An object's light-view silhouette appears quite jumbled when viewed form a point-of-view that does not correspond well with the light's point-of-view

## 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

$$\langle A_x L_w - L_x A_w, A_y L_w - L_y A_w, A_z L_w - L_z A_w, 0 \rangle$$
$$\langle B_x L_w - L_x B_w, B_y L_w - L_y B_w, B_z L_w - L_z B_w, 0 \rangle$$
$$\langle C_x L_w - L_x C_w, C_y L_w - L_y C_w, C_z L_w - L_z C_w, 0 \rangle$$

*Homogenous vector differences*

  - 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

$$\langle A_x, A_y, A_z, A_w \rangle$$
$$\langle B_x, B_y, B_z, B_w \rangle$$
$$\langle C_x, C_y, C_z, C_w \rangle$$

## Complete Robust Stenciled Shadow Volume Rendering Technique

- See our paper "Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering"
  - In the accompanying course notes
  - And on-line at developer.nvidia.com
- Paper has pseudo-code for rendering procedure
  - OpenGL state settings & rendering commands
  - Supports multiple per-vertex lights
  - Assumes application computes object-space determination of occluder model's polygons orientation w.r.t. each light

## Requirements for Our Stenciled Shadow Volume Technique (1)

1. Models must be composed of triangles only (avoiding non-planar polygons)
2. Models must be closed (2-manifold) and have a consistent winding order
   - Bergeron ['86] approach could be used to handle "open" models if necessary
3. Homogeneous object coordinates are permitted, assuming w≥0
   - If not, (x, y, z, -1) = (-x, -y, -z, 1)
4. Ideal light sources only
   - Directional or positional, assuming w≥0

## Requirements for Our Stenciled Shadow Volume Technique (2)

5. Connectivity information for occluding models must be available
   - So silhouette edges w.r.t. light positions can be determined at shadow volume construction time
6. Projection matrix must be perspective
   - Not orthographic
   - NV_depth_clamp extension provides orthographic support (more later)
7. Render must guarantee "watertight" rasterization
   - No double hitting pixels at shared polygon edges
   - No missed pixels at shared polygon edges

## Requirements for Our Stenciled Shadow Volume Technique (3)

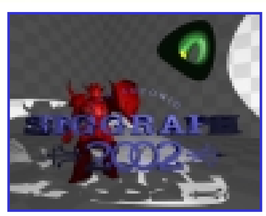
8. Enough stencil bits
   - *N* stencil bits where $2^N$ is greater than the maximum shadow depth count ever encountered
   - Scene dependent
   - 8-bits is usually quite adequate & what all recent stencil hardware provides
   - Wrapping stencil increment/decrement operations (i.e. OpenGL's EXT_stencil_wrap) permit deeper shadow counts, modulo aliasing with zero
   - Realize that shadow depths > 255 imply too much fill rate for interactive applications

## Requirements for Our Stenciled Shadow Volume Technique (4)

9. Rendering features provided by OpenGL 1.0 or DirectX 6 (or subsequent versions)
   - Transformation & clipping of homogenous positions
   - Front- and back-face culling
   - Masking color and depth buffer writes
   - Depth buffering (i.e. conventional Z-buffering)
   - Stencil-testing support

*In practice, these are quite reasonable requirements for nearly any polygonal-based 3D game or application*

## Our Approach in Practice (1)




**Scene with shadows.**
Yellow light is embedded in the green three-holed object. $P_{inf}$ is used for all the following scenes.


**Same scene visualizing the shadow volumes.**

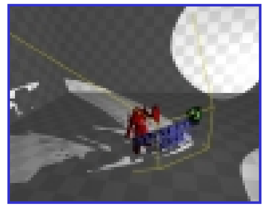
## Our Approach in Practice (2)

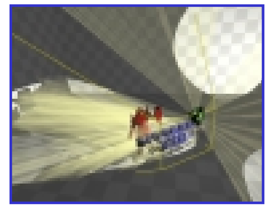
**Details worth noting . . .**

**Fine details:** Shadows of the A, N, and T letters on the knight's armor and shield.

**Hard case:** The shadow volume from the front-facing hole would definitely intersect the near clip plane.

## Our Approach in Practice (3)

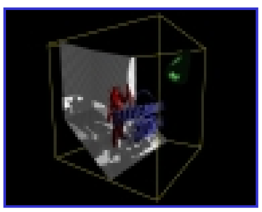


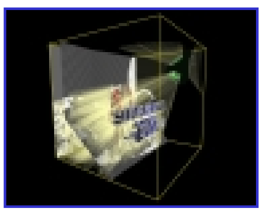


**Alternate view of same scene with shadows.** Yellow lines indicate previous view's view frustum boundary. Recall shadows are view-independent.

**Shadow volumes from the alternate view.**

## Our Approach in Practice (4)





**Clip-space view.** Original view's scene seen from clip space. The back plane is "at infinity" with very little effective depth precision near infinity.

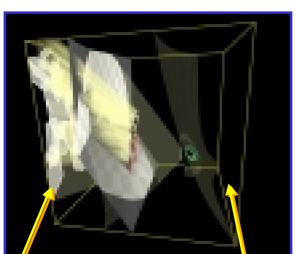
**Clip-space view of shadow volumes.** Back-facing triangles w.r.t. light are seen projected onto far plane at infinity.

## Another Example (1)

**Original eye's view.** Again, yellow light is embedded in the green three-holed object. $P_{inf}$ is used for all the following scenes.

**Eye-space view of previous eye's view.** Clipped to the previous eye's $P_{inf}$ view frustum. Shows knight's projection to infinity.

## Another Example (2)



**Clip-space view of previous eye's view.** Shows shadow volume closed at infinity and other shadow volume's intersection with the near clip plane.

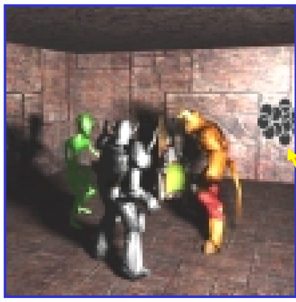
Original eye's **far** clip plane

Original eye's **near** clip plane

## Stenciled Shadow Volumes with Multiple Lights

**Three colored lights.** Diffuse/specular bump mapped animated characters with shadows. 34 fps on GeForce4 Ti 4600; 80+ fps for one light.

## Stenciled Shadow Volumes for Simulating Soft Shadows



**Cluster of 12 dim lights approximating an area light source.** Generates a soft shadow effect; careful about banding. 8 fps on GeForce4 Ti 4600.

The cluster of point lights.

## 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)

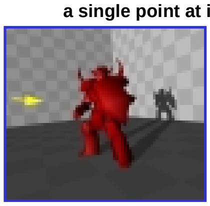
- **Pre-compute or re-use cache shadow volume geometry when geometric relationship between a light and occluder does not change between frames**
  - **Example: Headlights on a car have a static shadow volume w.r.t. the car itself as an occluder**
- **Advanced shadow volume culling approaches**
  - **Uses 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)

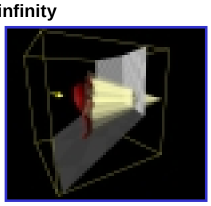
- **Take advantage of ad-hoc knowledge of scenes whenever possible**
  - **Example: A totally closed room means you do not have to cast shadow volumes for the wall, floor, ceiling**
- **Limit shadows to important entities**
  - **Example: Generate shadow volumes for monsters and characters, but not static 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**



Scene with directional light.



Clip-space view of shadow volume

## 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 $2^N$, $2\times2^N$, $3\times2^N$, 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 <u>front</u>-facing geometry
  - Second, rasterizing <u>back</u>-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

## Usage of NV_stencil_two_side & EXT_stencil_wrap

**OLD SCHOOL**

```
glDepthMask(0);
glColorMask(0,0,0,0);
glEnable(GL_CULL_FACE);
glEnable(GL_STENCIL_TEST);
glStencilMask(~0);
glStencilFunc(GL_ALWAYS, 0, ~0);
 // Increment for back faces
glCullFace(GL_BACK);
glStencilOp(GL_KEEP,   // stencil test fail
            GL_INCR,   // depth test fail
            GL_INCR);  // depth test pass
renderShadowVolumePolygons();
// Decrement for front faces
glCullFace(GL_FRONT);
glStencilOp(GL_KEEP,   // stencil test fail
            GL_DECR,   // depth test fail
            GL_KEEP);  // depth test pass
renderShadowVolumePolygons();
```

**NEW SCHOOL**

```
glDepthMask(0);
glColorMask(0,0,0,0);
glDisable(GL_CULL_FACE);
glEnable(GL_STENCIL_TEST);
glEnable(GL_STENCIL_TEST_TWO_SIDE_NV);
glActiveStencilFaceNV(GL_BACK);
glStencilOp(GL_KEEP,          // stencil test fail
            GL_INCR_WRAP_EXT, // depth test fail
            GL_KEEP);         // depth test pass
glStencilMask(~0);
glStencilFunc(GL_ALWAYS, 0, ~0);
glActiveStencilFaceNV(GL_FRONT);
glStencilOp(GL_KEEP,    // stencil test fail
            GL_DECR_WRAP_EXT,  // depth test fail
            GL_KEEP);  // depth test pass
glStencilMask(~0);
glStencilFunc(GL_ALWAYS, 0, ~0);
renderShadowVolumePolygons();
```

New approach calls renderShadowVolumePolygons() just once.

## 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

## Our Contribution

- We Integrate Multiple Ideas into a Robust Solution
  - Dietrich and Carmack's *zfail* approach
  - Kilgard's homogeneous coordinates (w=0) for rendering infinite shadow volume geometry
  - Somewhat-obscure [Blinn '93] *infinite far plane* projection matrix formulation
  - DirectX 6's wrapping stencil increment & decrement
    - OpenGL's EXT_stencil_wrap extension
- Hardware Enhancements
  - Depth clamping: *better depth precision*
  - Two-sided stencil testing: *performance*

## Conclusions

- Presented robust stenciled shadow volume technique
  - Reasonable & well-defined requirements
  - Almost impossible to buy PC in 2002 without stencil buffer support
  - Still, shadow volumes require high fill rates
- Technical paper has more detail
- Carmack's new Doom engine is already using this technique today
  - Why aren't you?