Optimized Stencil Shadow Volumes

Cass Everitt & Mark J. Kilgard

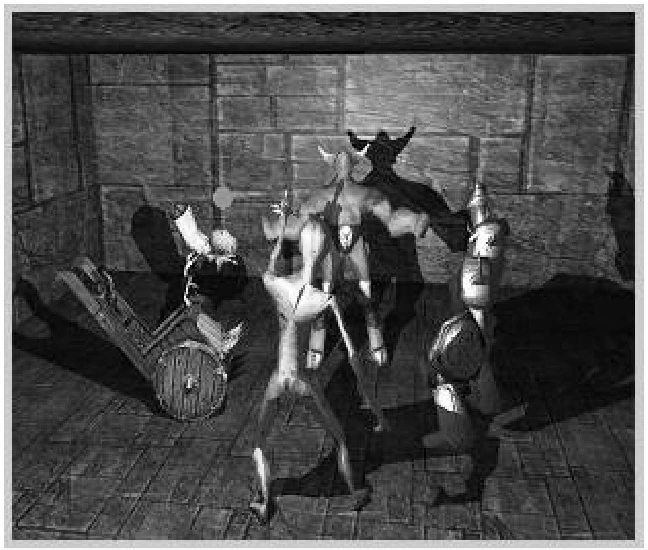
Outline: Complete Lecture

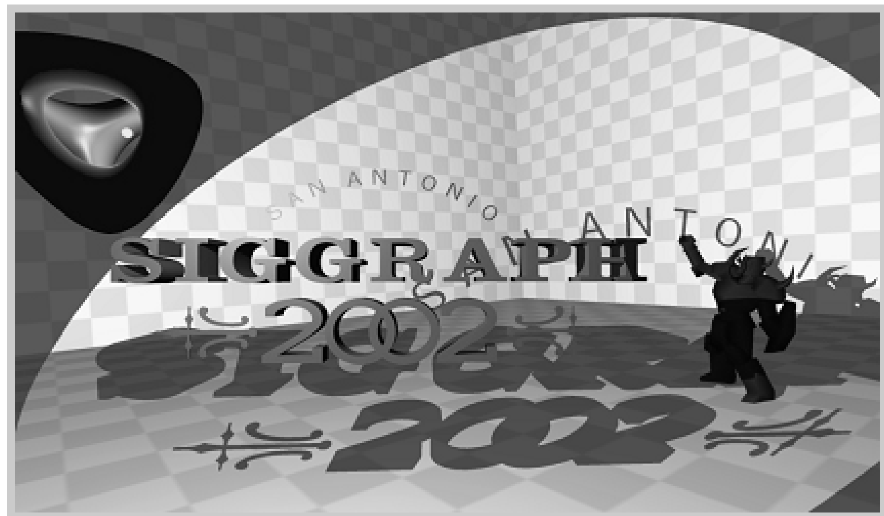
- Review of Stenciled Shadow Volumes
- Stenciled Shadow Volumes Optimizations
- More Information

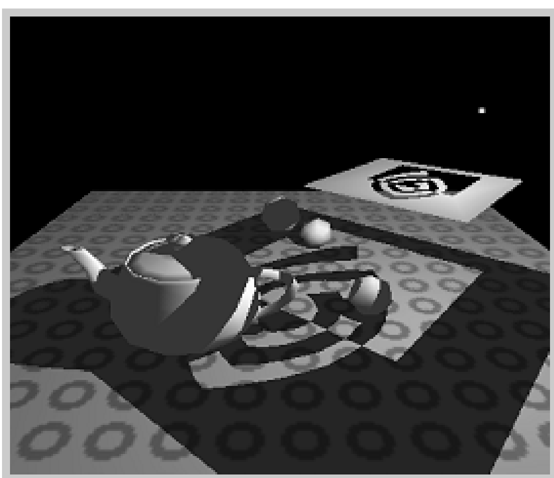
Outline: Review of Stenciled Shadow Volumes

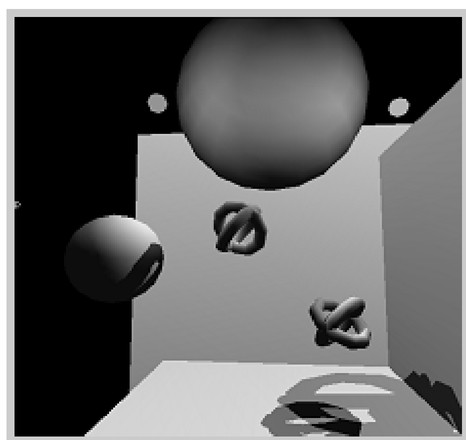
- Shadow Volume Basics
- Zpass vs. Zfail
- Infinite Rendering Techniques

Stenciled Shadow Volumes in Practice



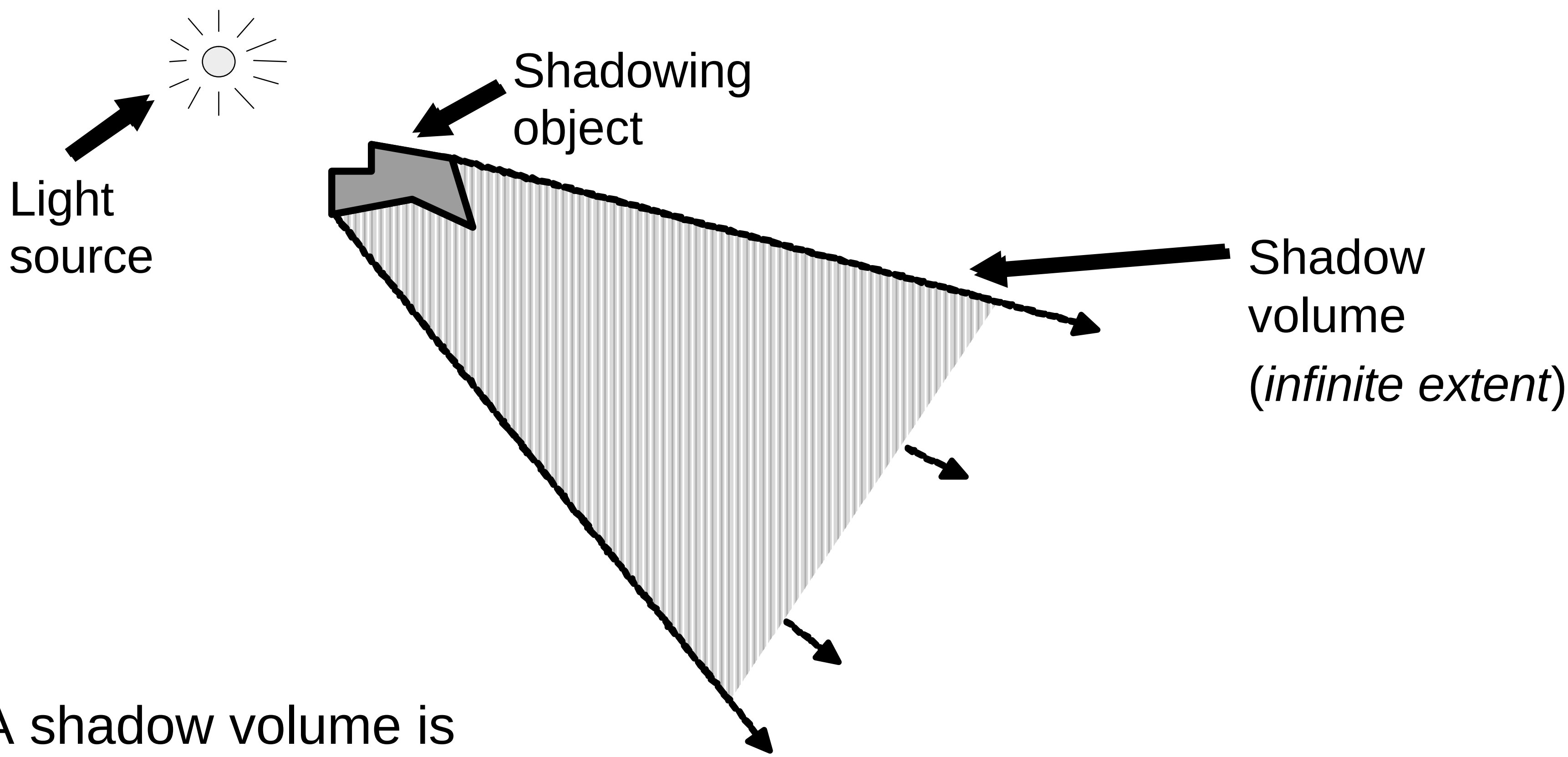






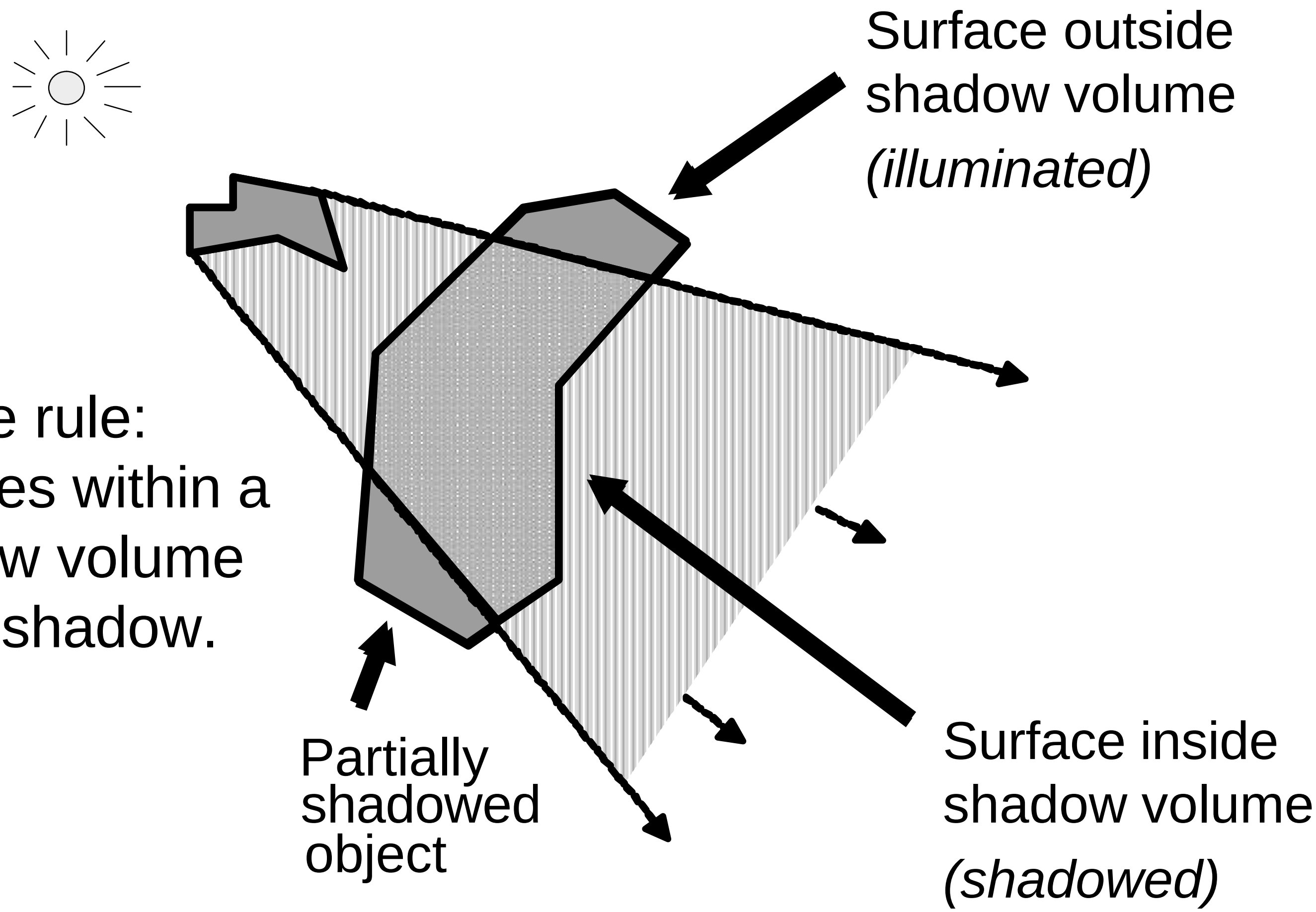
Notice the proper self-shadowing!

Shadow Volume Basics



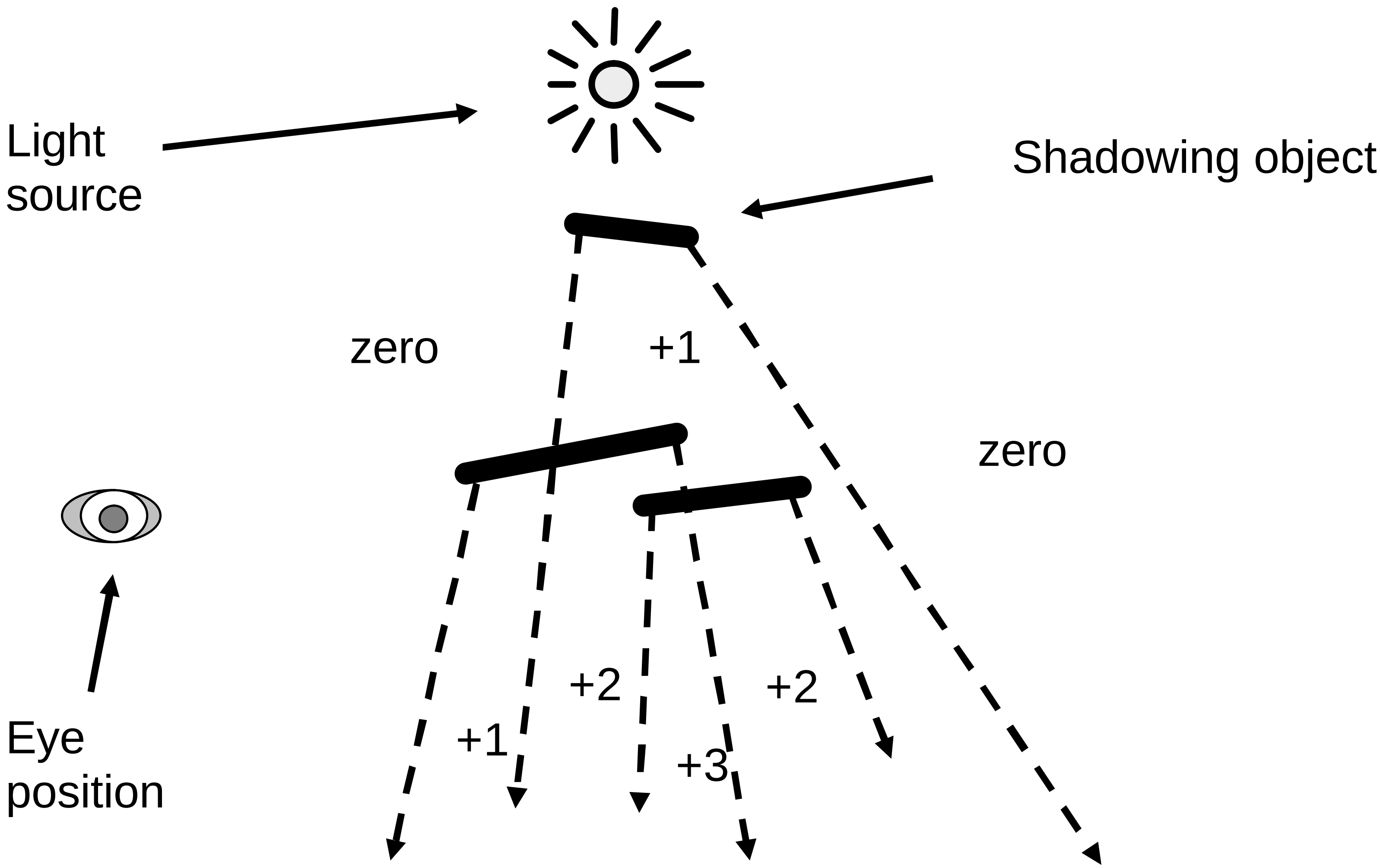
A shadow volume is
simply the half-space defined
by a light source and a shadowing object.

Shadow Volume Basics (2)

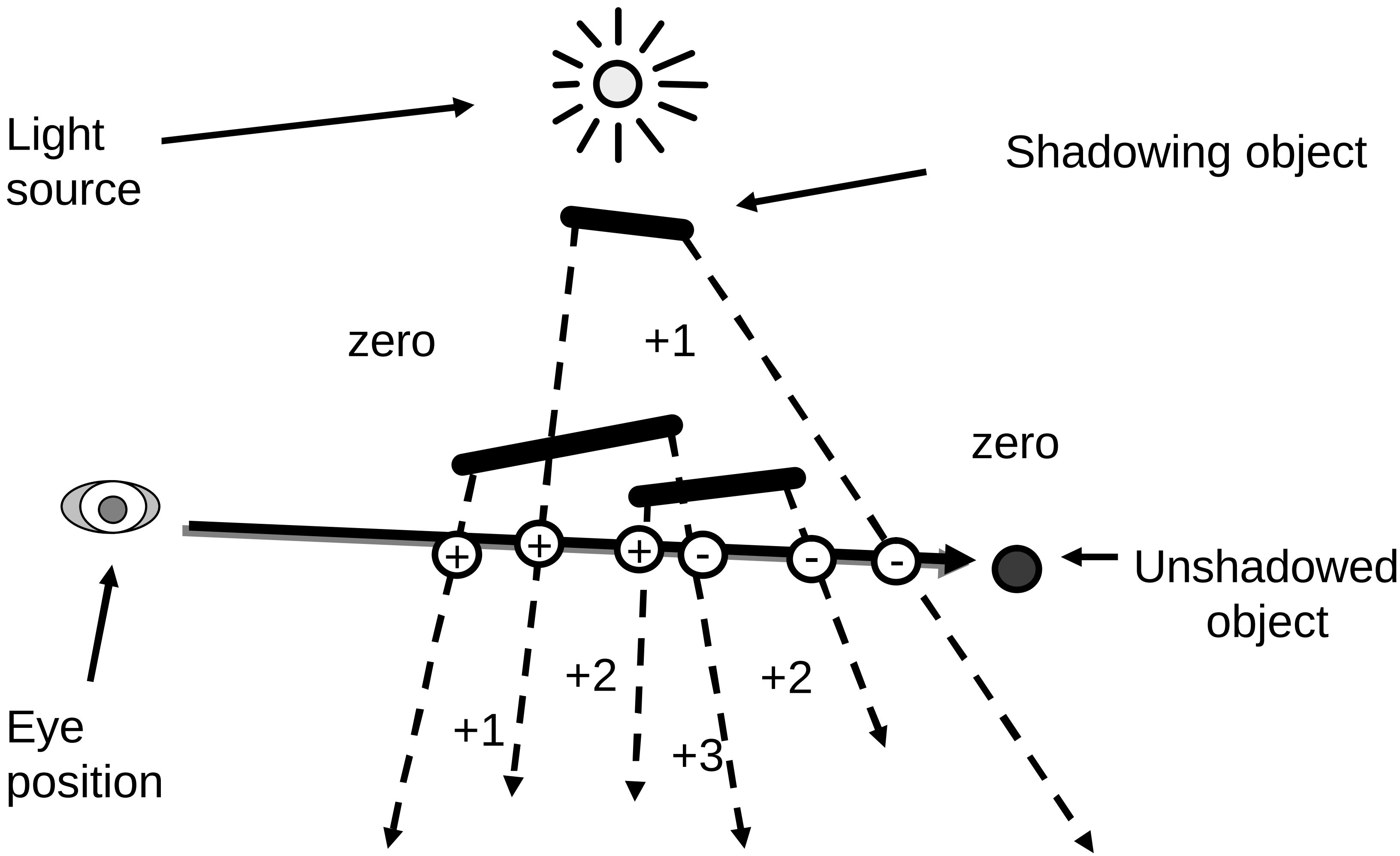


Simple rule:
samples within a
shadow volume
are in shadow.

Why Eye-to-Object Stencil Counting Approach Works

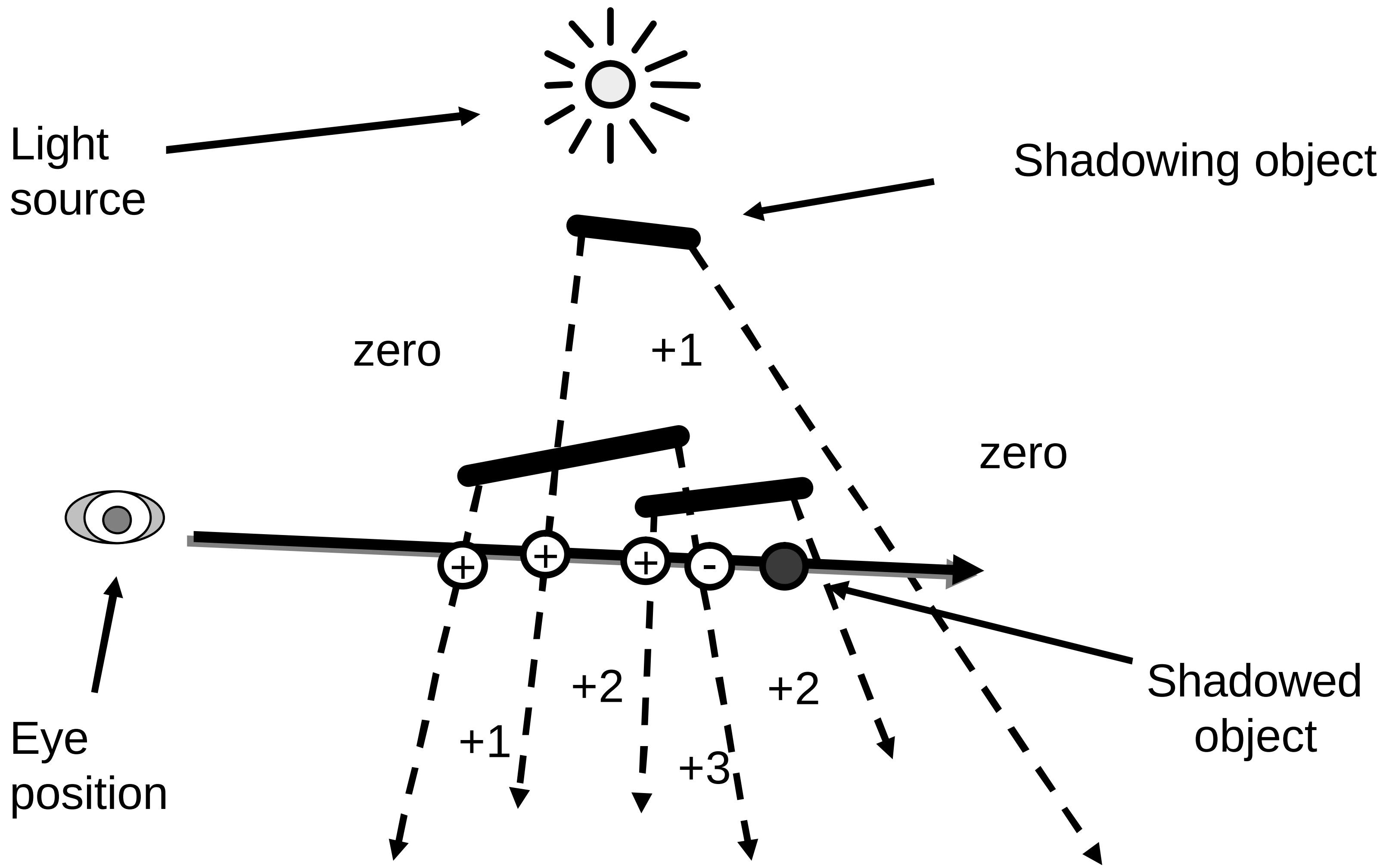


Illuminated, Behind Shadow Volumes (*Zpass*)



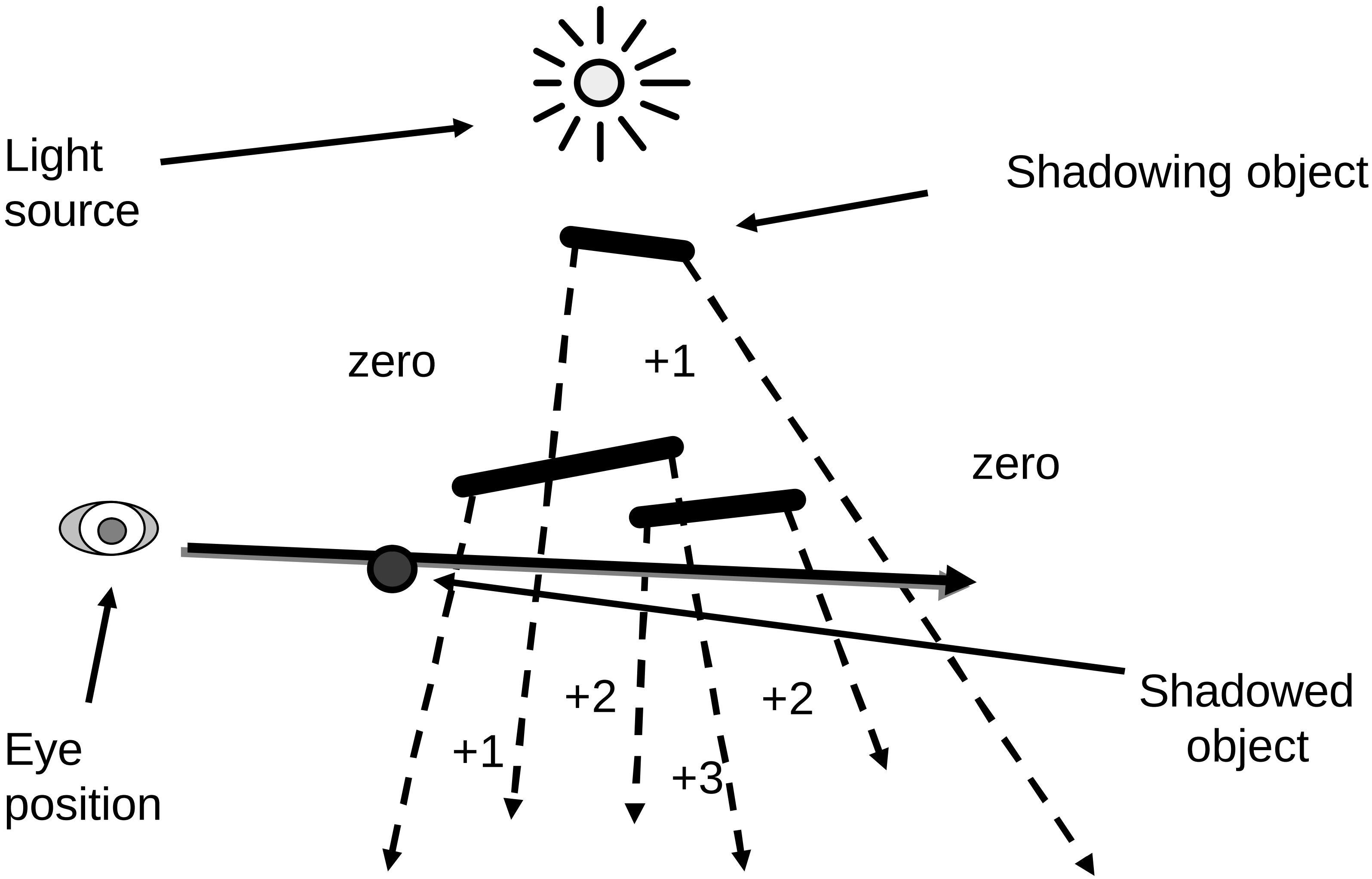
Shadow Volume Count = +1+1+1-1-1-1 = 0

Shadowed, Nested in Shadow Volumes (*Zpass*)



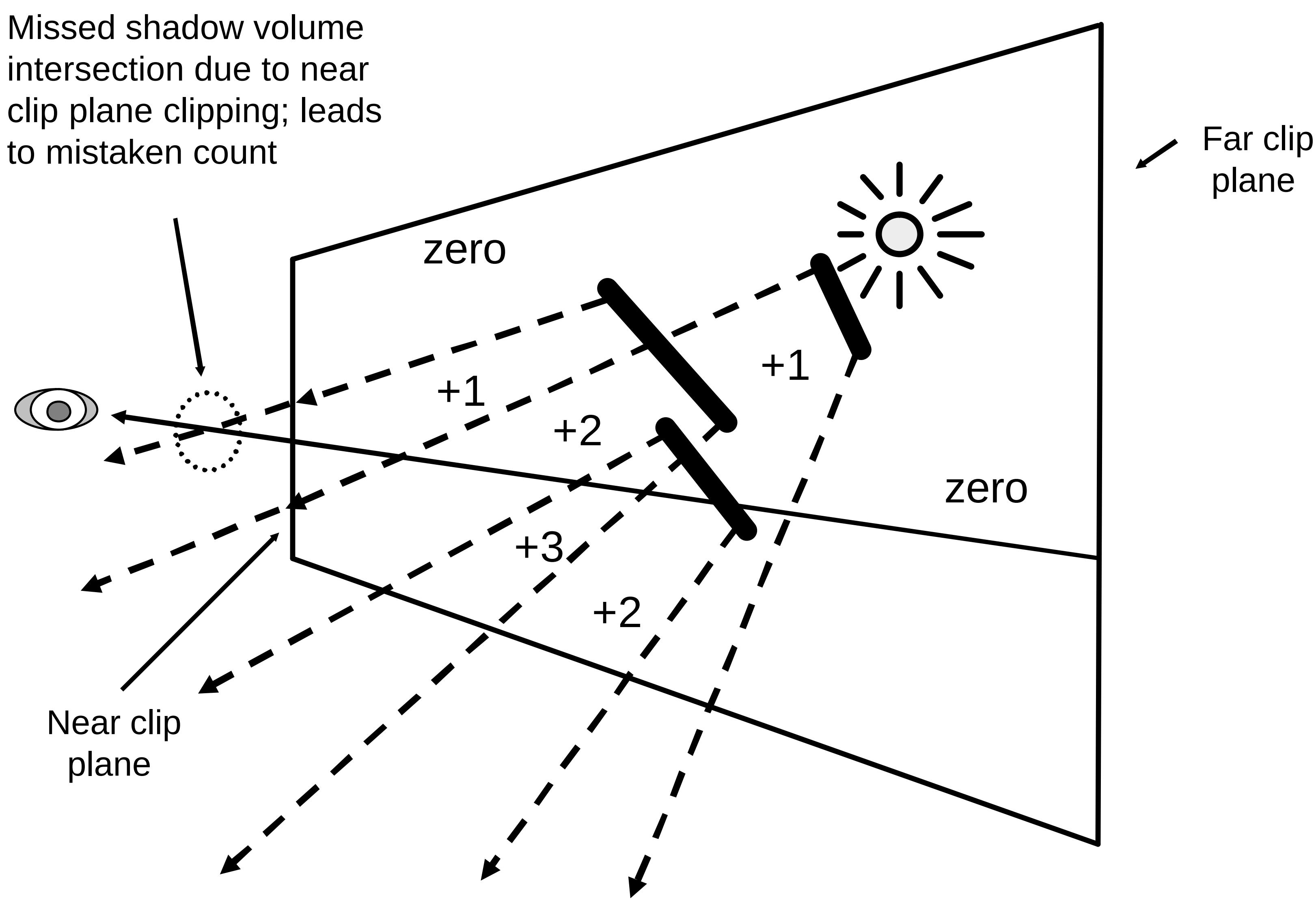
Shadow Volume Count = +1+1+1-1 = 2

Illuminated, In Front of Shadow Volumes (*Zpass*)

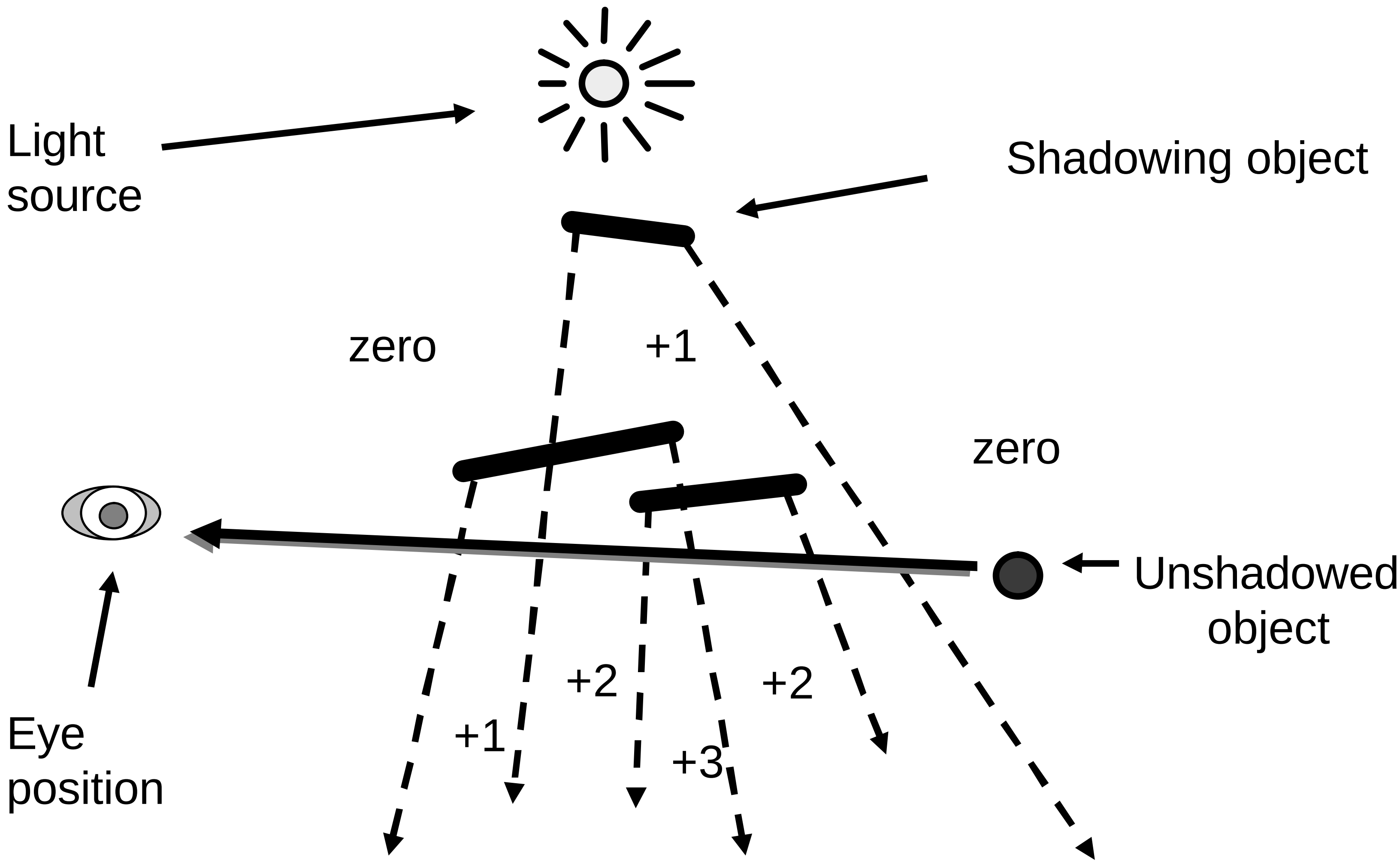


Shadow Volume Count = 0 (no depth tests pass)

Problems Created by Near Clip Plane (*Zpass*)

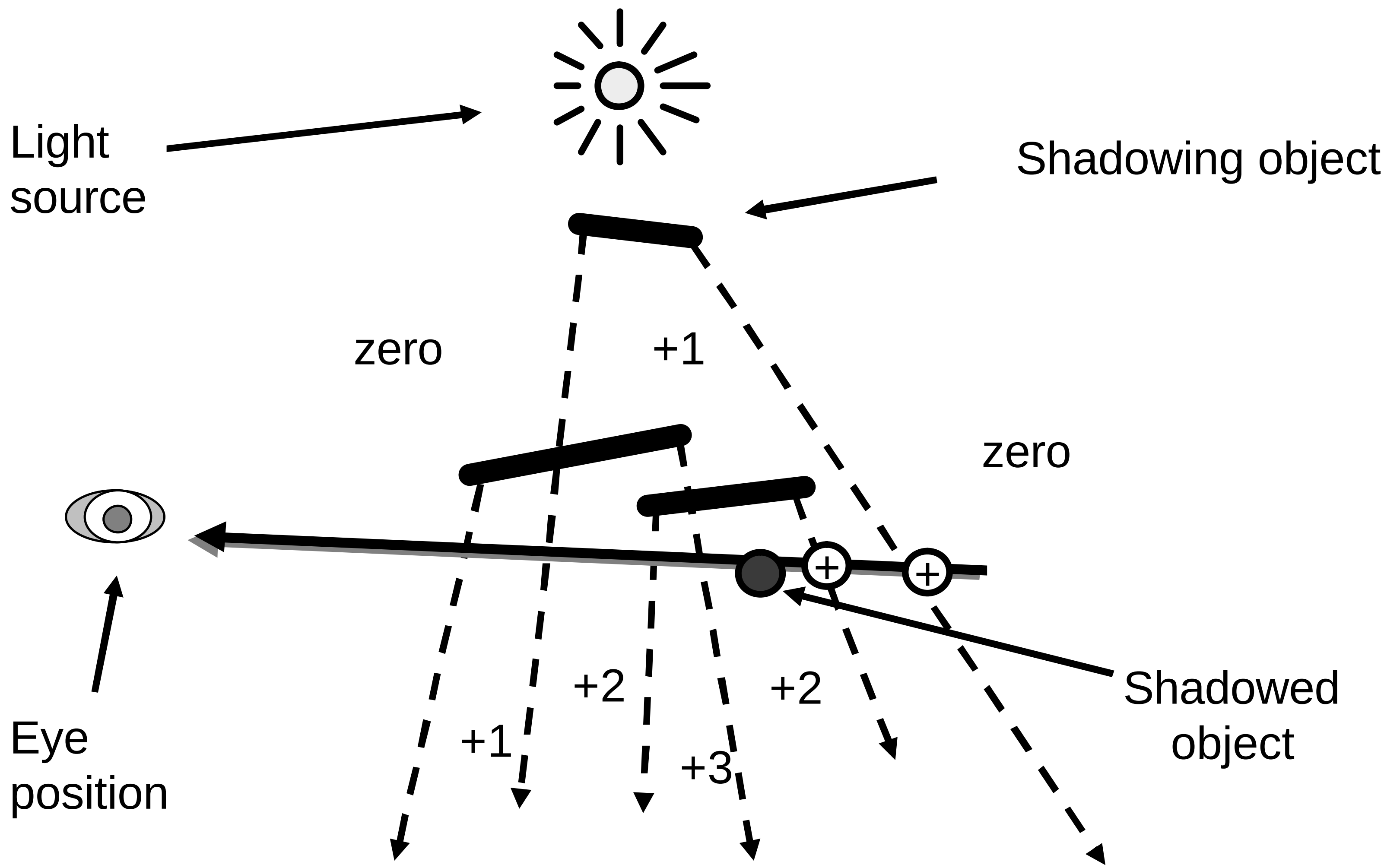


Illuminated, Behind Shadow Volumes (*Zfail*)



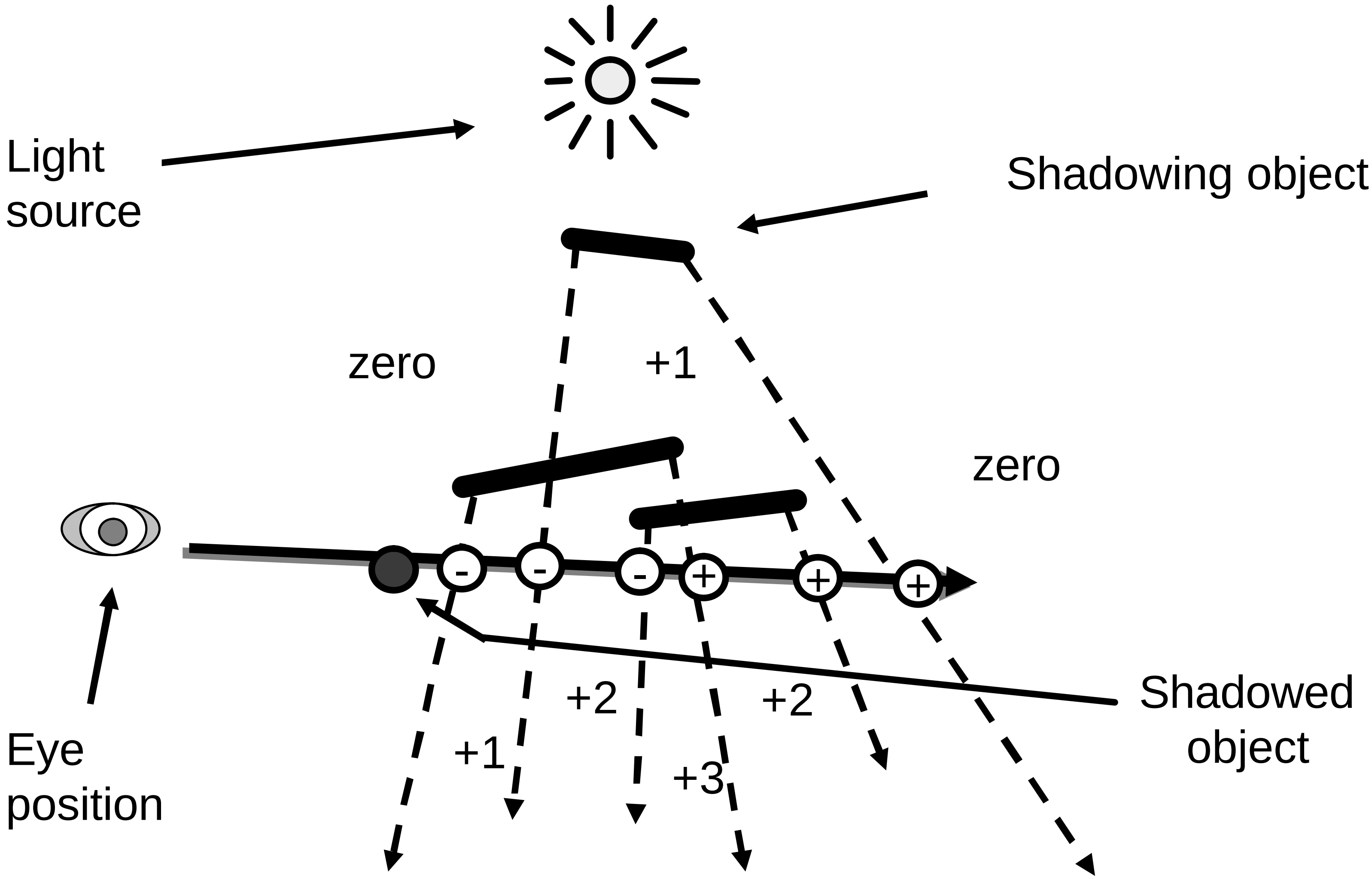
Shadow Volume Count = 0 (zero depth tests fail)

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

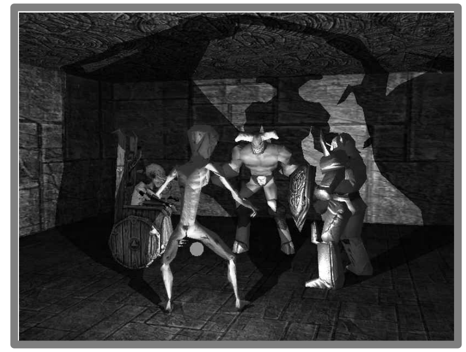
Nested Shadow Volumes Stencil Counts Beyond One

Shadowed scene

Stencil buffer contents

green = stencil value of 0
red = stencil value of 1
darker reds = stencil value > 1

Animation of Stencil Buffer Updates for a Single Light’s Shadow Volumes

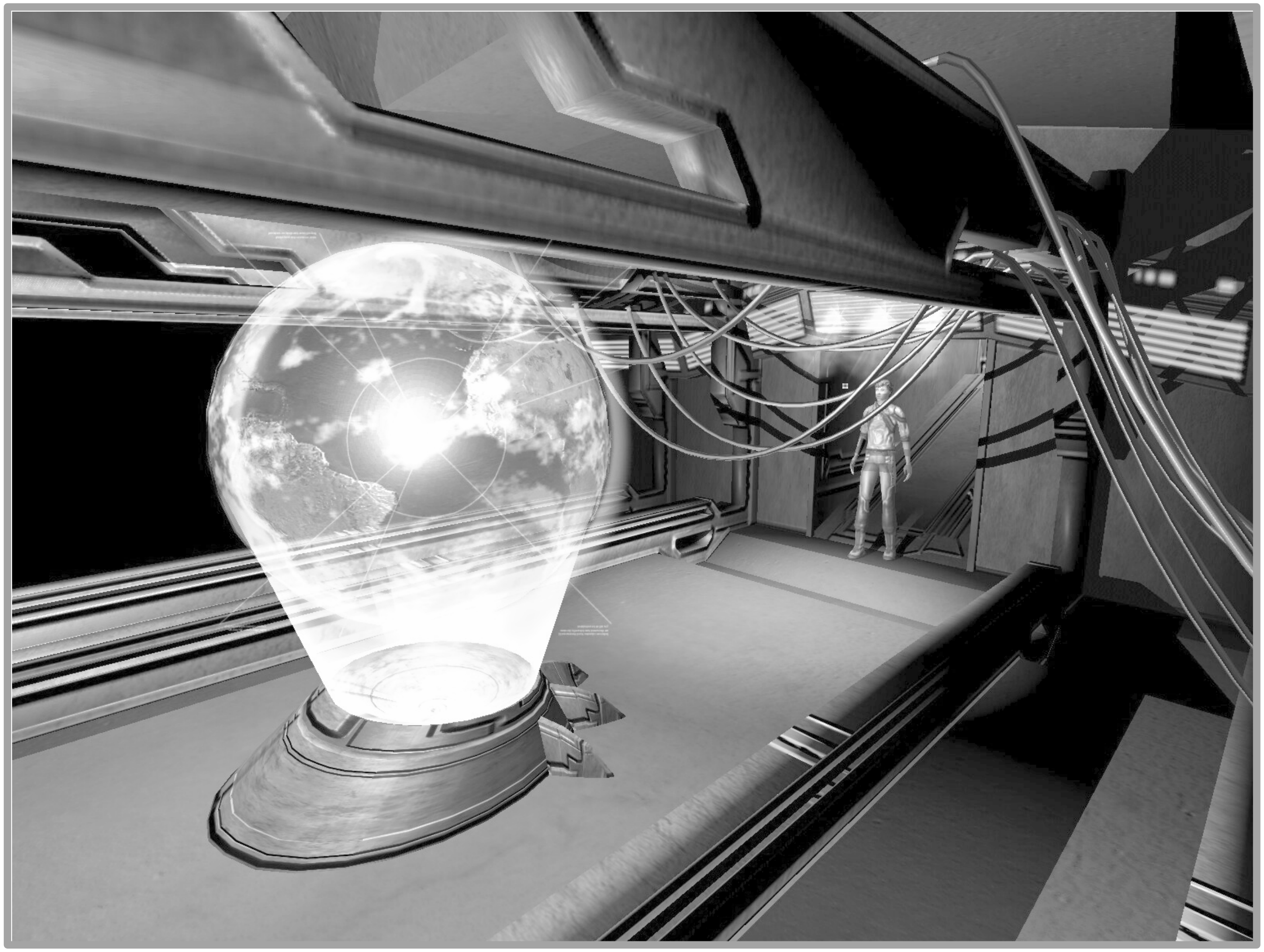


Fully shaded scene

Final stencil state

Every frame is 5 additional stencil shadow volume polygon updates. Note how various intermediate stencil values do not reflect the final state.

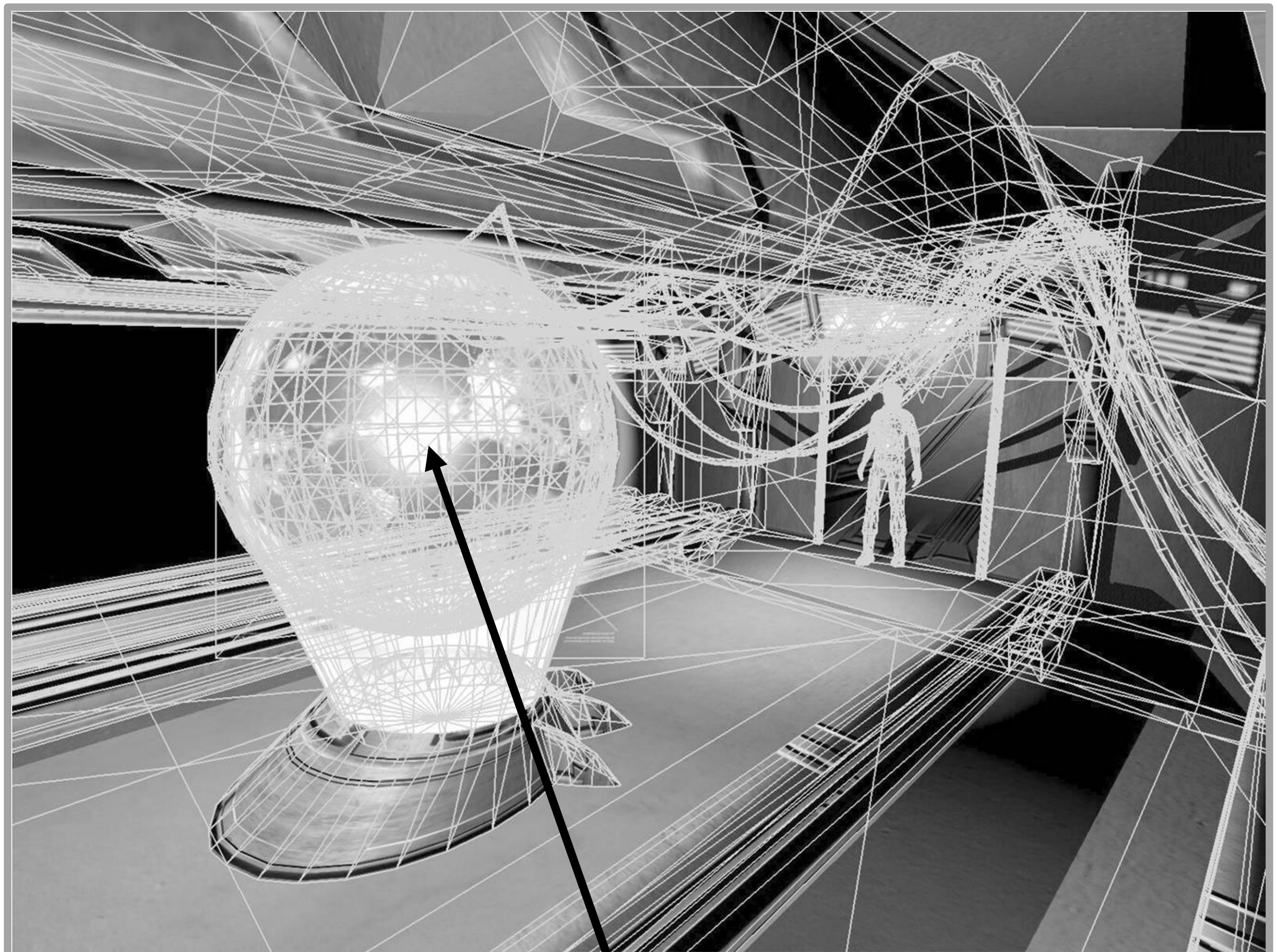
Shadows in a Real Game Scene



CONTRABAND
ENTERTAINMENT

Abducted game images courtesy Joe Riedel at Contraband Entertainment

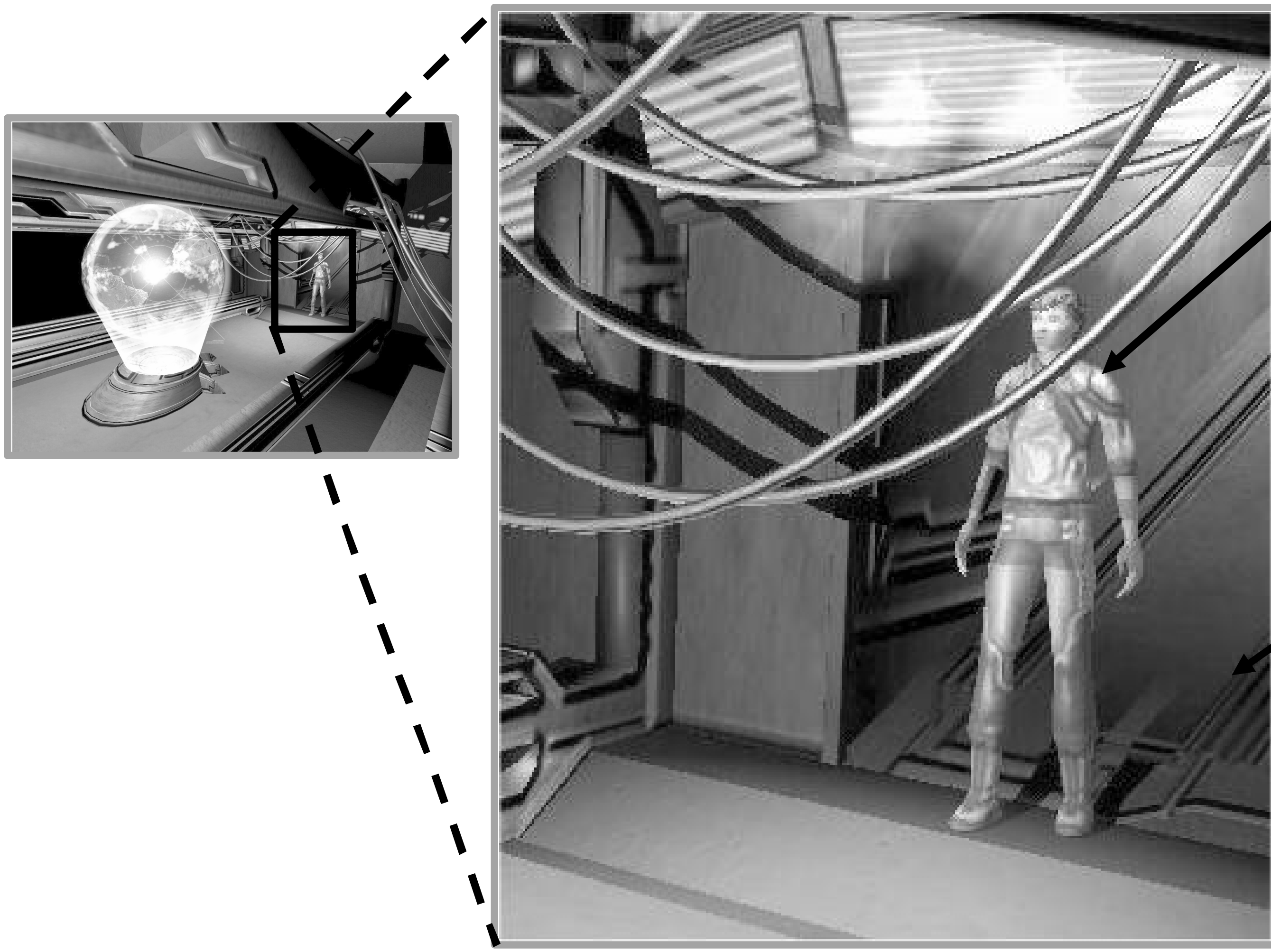
Scene's *Visible* Geometric Complexity



Primary light source location

Wireframe shows geometric complexity of visible geometry

Blow-up of Shadow Detail



Notice cable shadows on player model

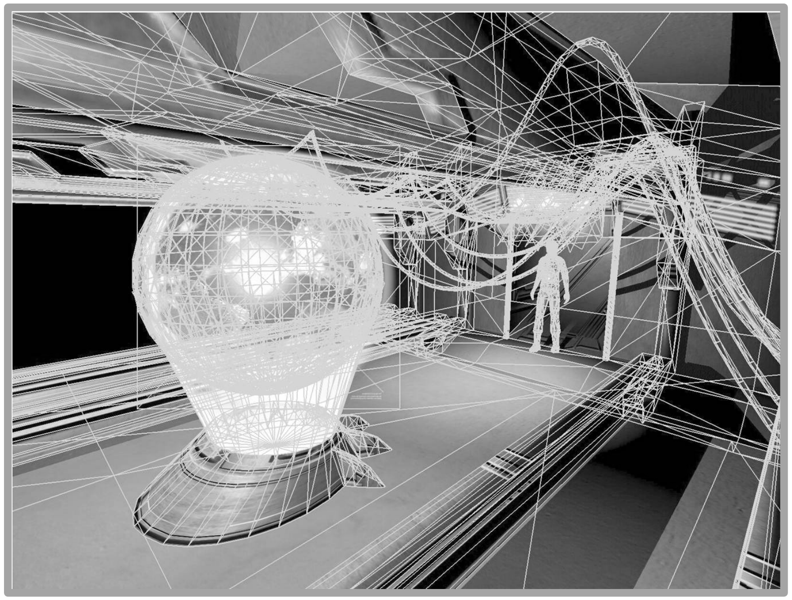
Notice player's own shadow on floor

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



Visible geometry

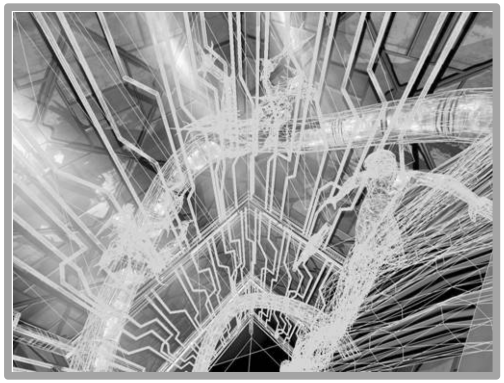
<<

Shadow volume geometry

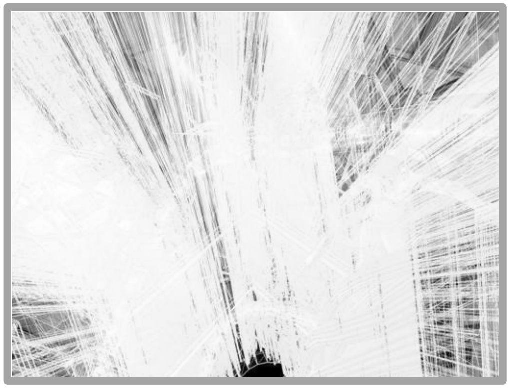
Typically, shadow volumes generate considerably more pixel updates than visible geometry

Other Example Scenes (1 of 2)

Dramatic chase scene with shadows



Visible geometry



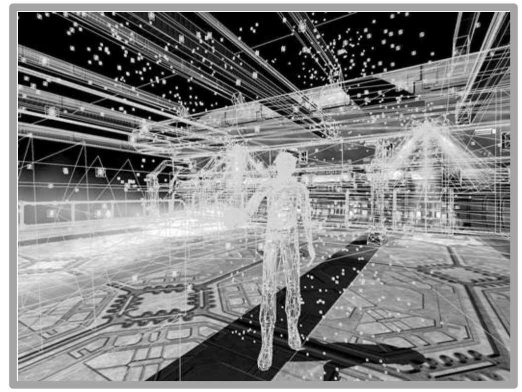
Shadow volume geometry

CONTRABAND ENTERTAINMENT

*Abducted g*ame images courtesy
Joe Riedel at Contraband Entertainment

Other Example Scenes (2 of 2)

Scene with multiple light sources



Visible geometry

Shadow volume geometry

*Abducted g*ame images courtesy
Joe Riedel at Contraband Entertainment

CONTRABAND
ENTERTAINMENT

Situations When Shadow Volumes Are Too Expensive

Chain-link fence is shadow volume nightmare!

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

*Fuel g*ame image courtesy Nathan d'Obrenan at Firetoad Software

Shadow Volumes vs. Shadow Maps

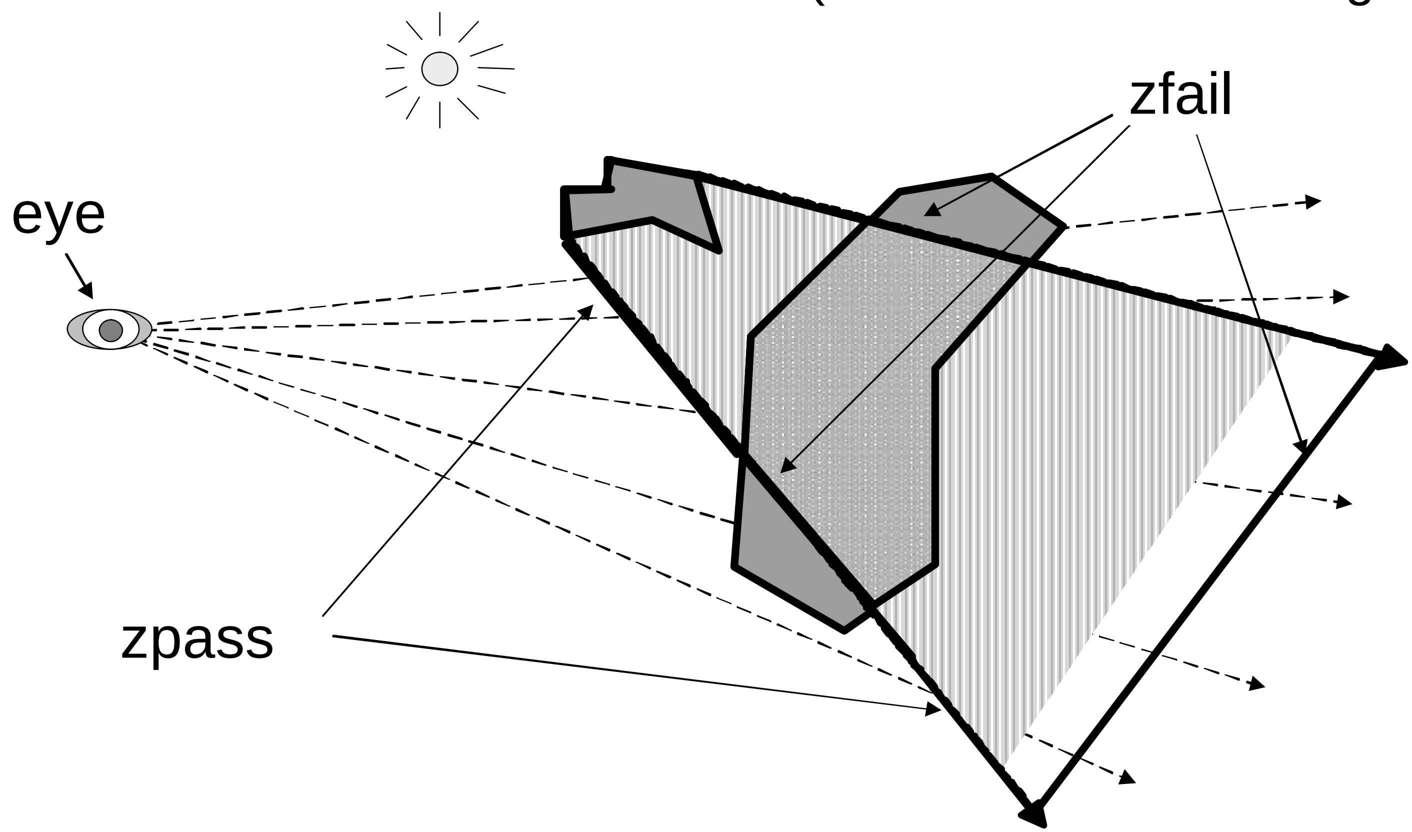
- Shadow mapping via projective texturing
  - The other prominent hardware-accelerated shadow technique
  - Standard part of OpenGL 1.4
- Shadow mapping advantages
  - Requires no explicit knowledge of object geometry
  - No 2-manifold requirements, etc.
  - View independent
- Shadow mapping disadvantages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

Outline: Stenciled Shadow Volumes Optimizations

- Fill Rate Optimizations
  - Dynamic Zpass vs. Zfail determination
  - Bounds
    - Scissoring
    - Depth bounds test
    - Efficient stencil clears
- Culling Optimizations
  - Portal-based culling
  - Occluder and cap culling
- Silhouette Determination Optimizations
  - Efficient data structures for static occluder & dynamic light
  - Pre-computing shadow volumes for static occluders & static lights
  - Simplified occluder geometry
- Shadow Volume Rendering Optimizations
  - Vertex programs for shadow volume rendering
  - Two-sided stencil testing
  - Directional lights

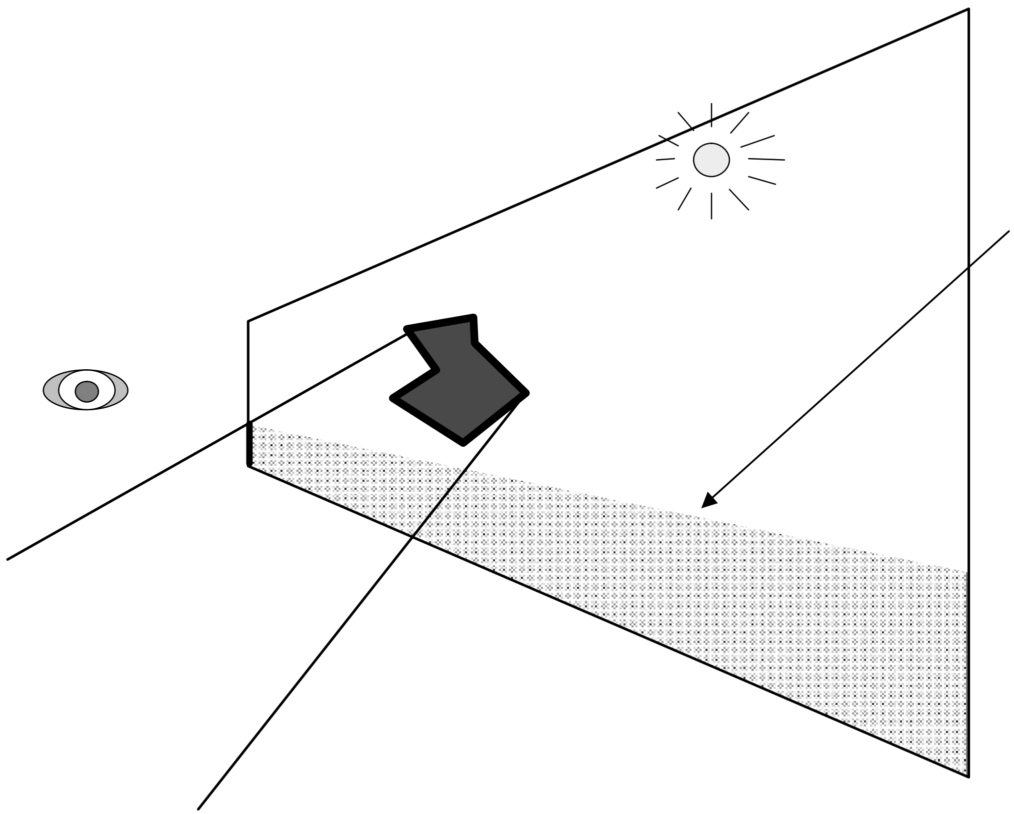
Zpass vs. Zfail (1)

- **Zpass counting requires no caps**
- **Zpass is often more efficient** (due to occlusion culling hw)



Zpass vs Zfail (2)

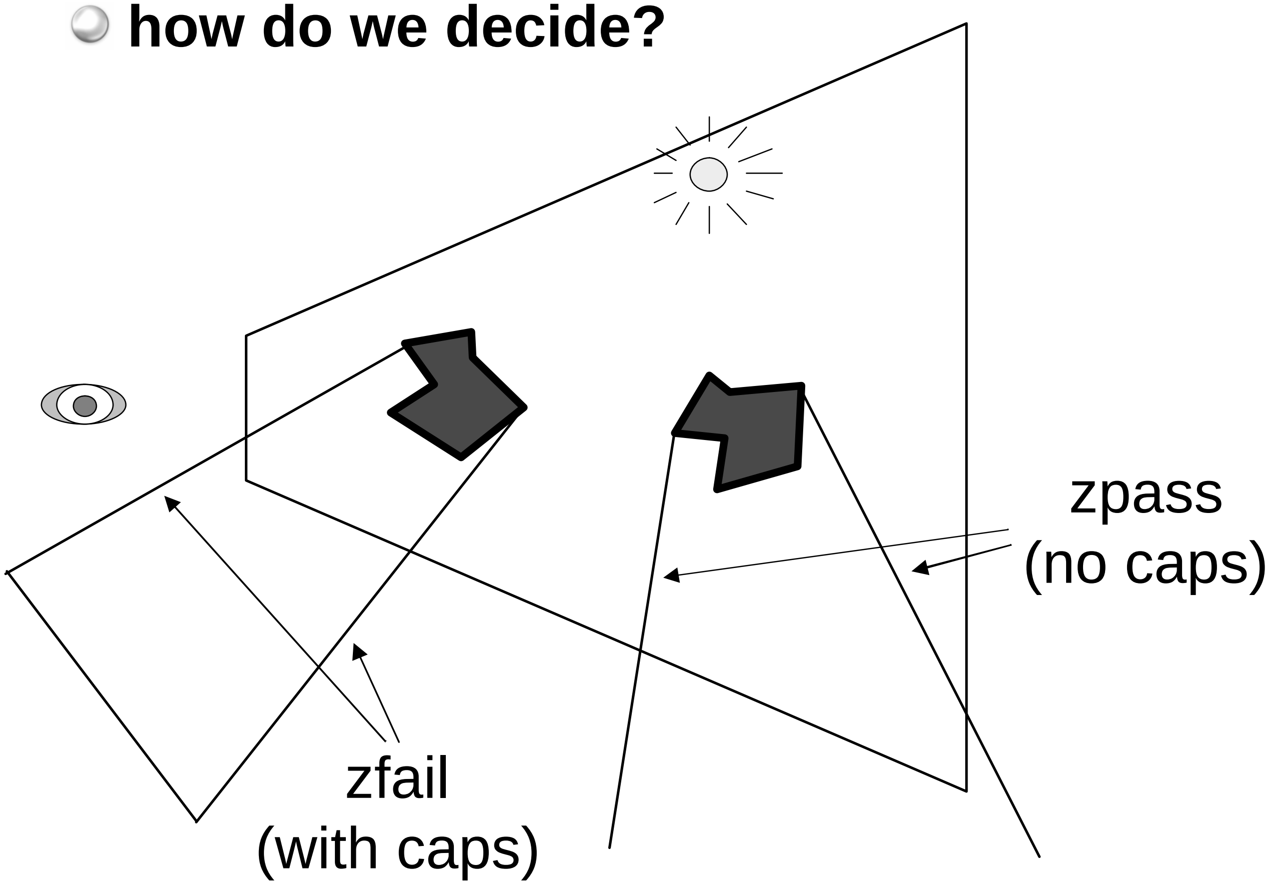
- **Zpass is hard to get right if shadow volume intersects near plane**



zpass requires complex near plane capping to avoid throwing the shadow count off for this whole region

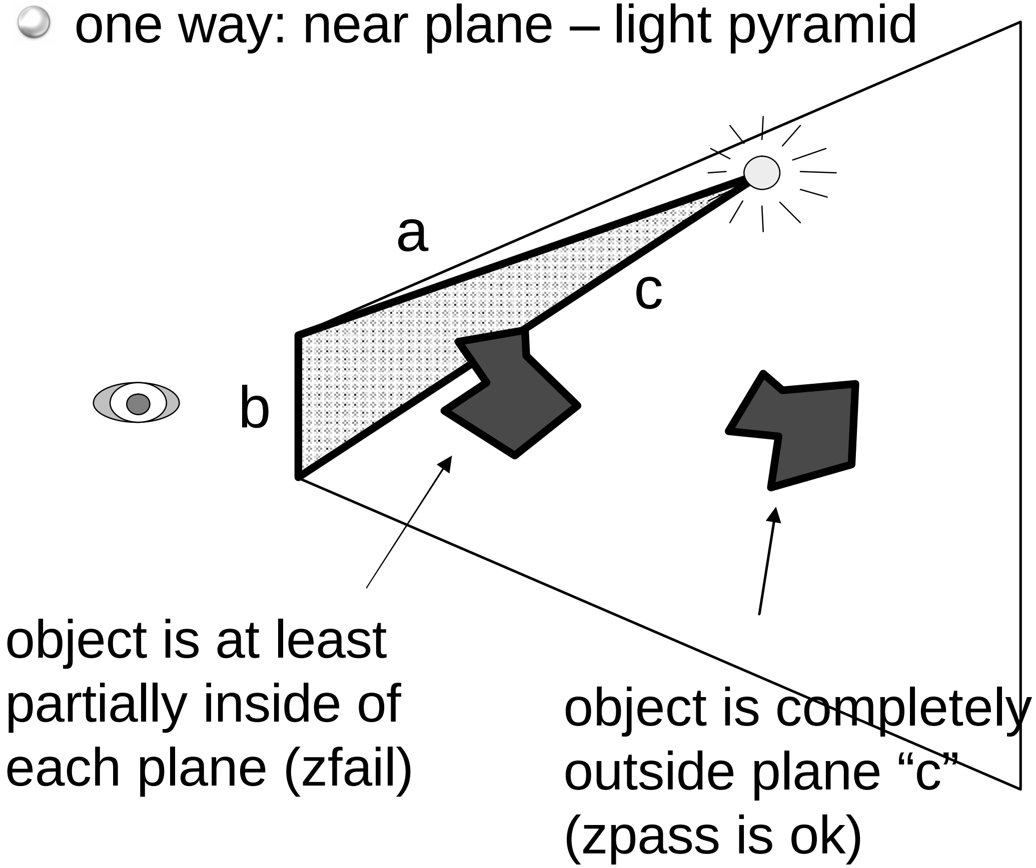
Zpass vs Zfail (3)

- We can decide zpass vs zfail on a per-object basis
  - do zfail when we have to
  - how do we decide?



Zpass vs Zfail (4)

- **How do we decide?**
  - one way: near plane – light pyramid



This diagram shows three planes in the “near plane - light pyramid”.

Any object that is completely outside one of the planes can be rendered with zpass.

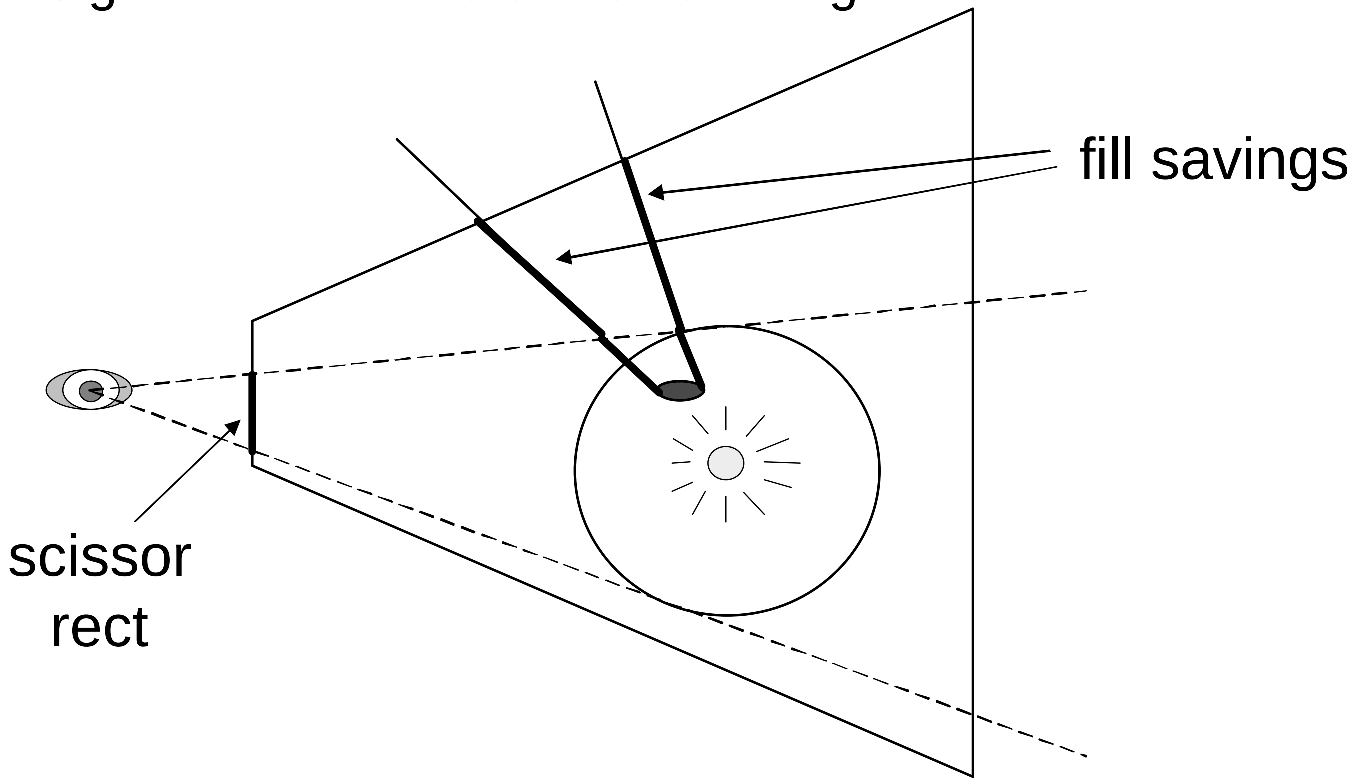
Use object bounding volume for speed.

Exploit Bounds

- Infinite shadow volumes can cover a lot of pixels
- With some knowledge of bounds, we can reduce fill
  - lights
    - attenuation
    - environment (walls/floor)
  - shadow volumes
    - depth range of shadow
    - redundant shadows

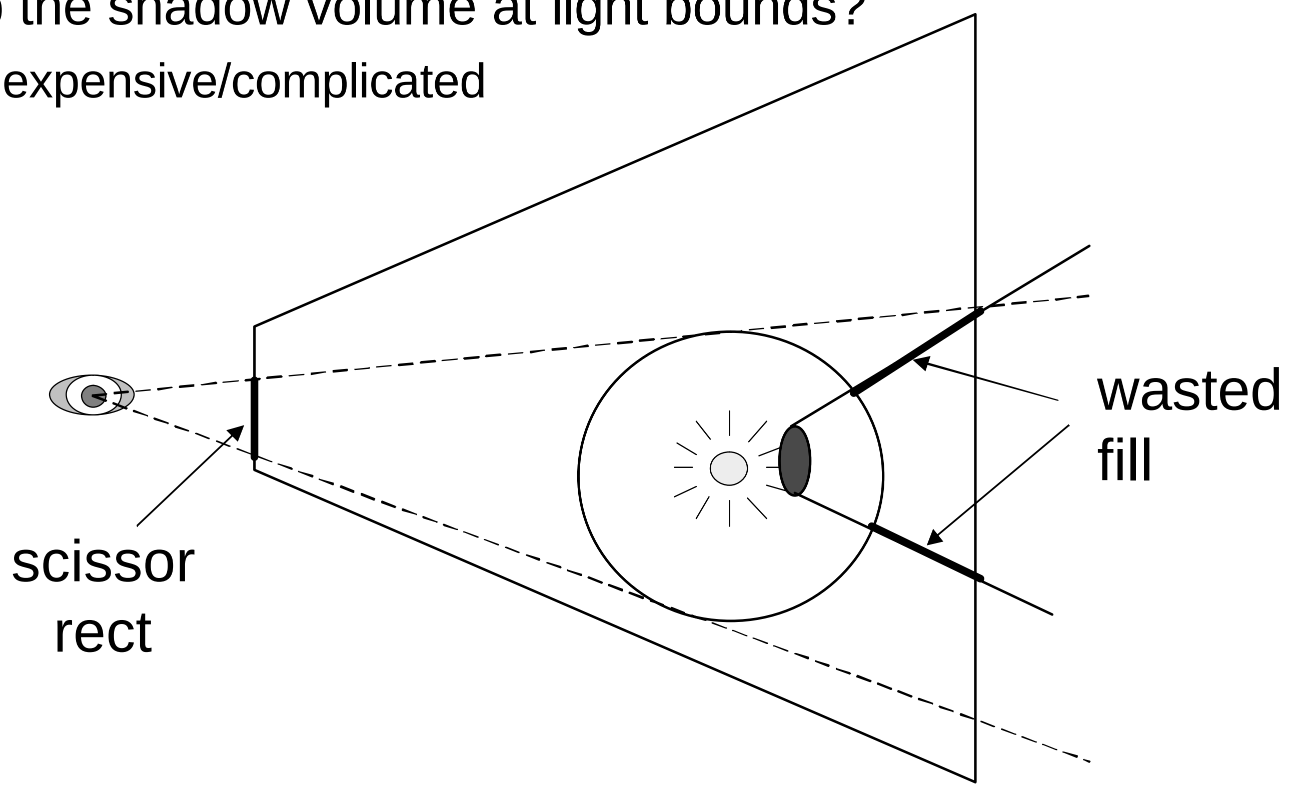
Attenuated Light Bounds

- **For attenuated lights, we can use the scissor test to constrain the shadow volume**
  - works great for occluders beside the light



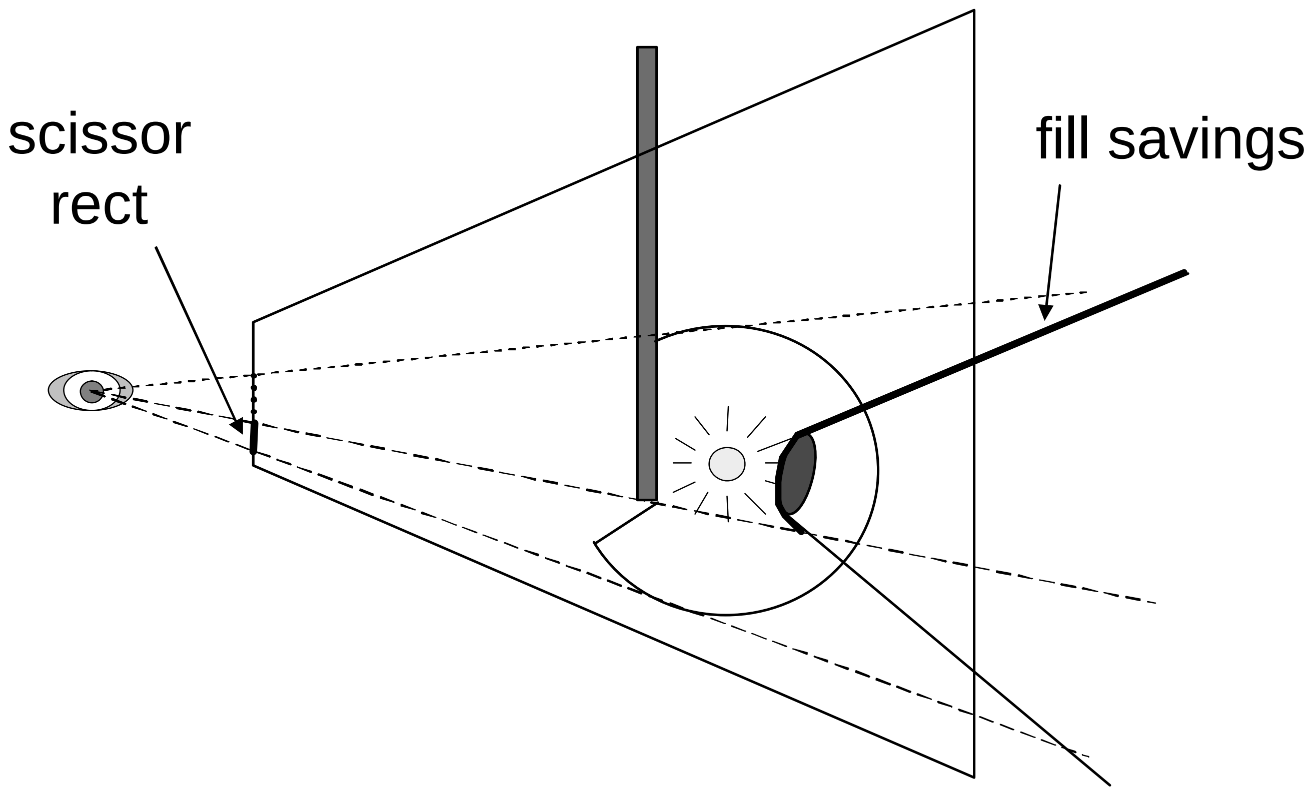
Attenuated Light Bounds (2)

- **The scissor test does not help as much with occluders in front of or behind the light**
  - Cap the shadow volume at light bounds?
    - expensive/complicated



Light Bounds due to Environment

- **Some environments offer big opportunities for bounding optimizations**

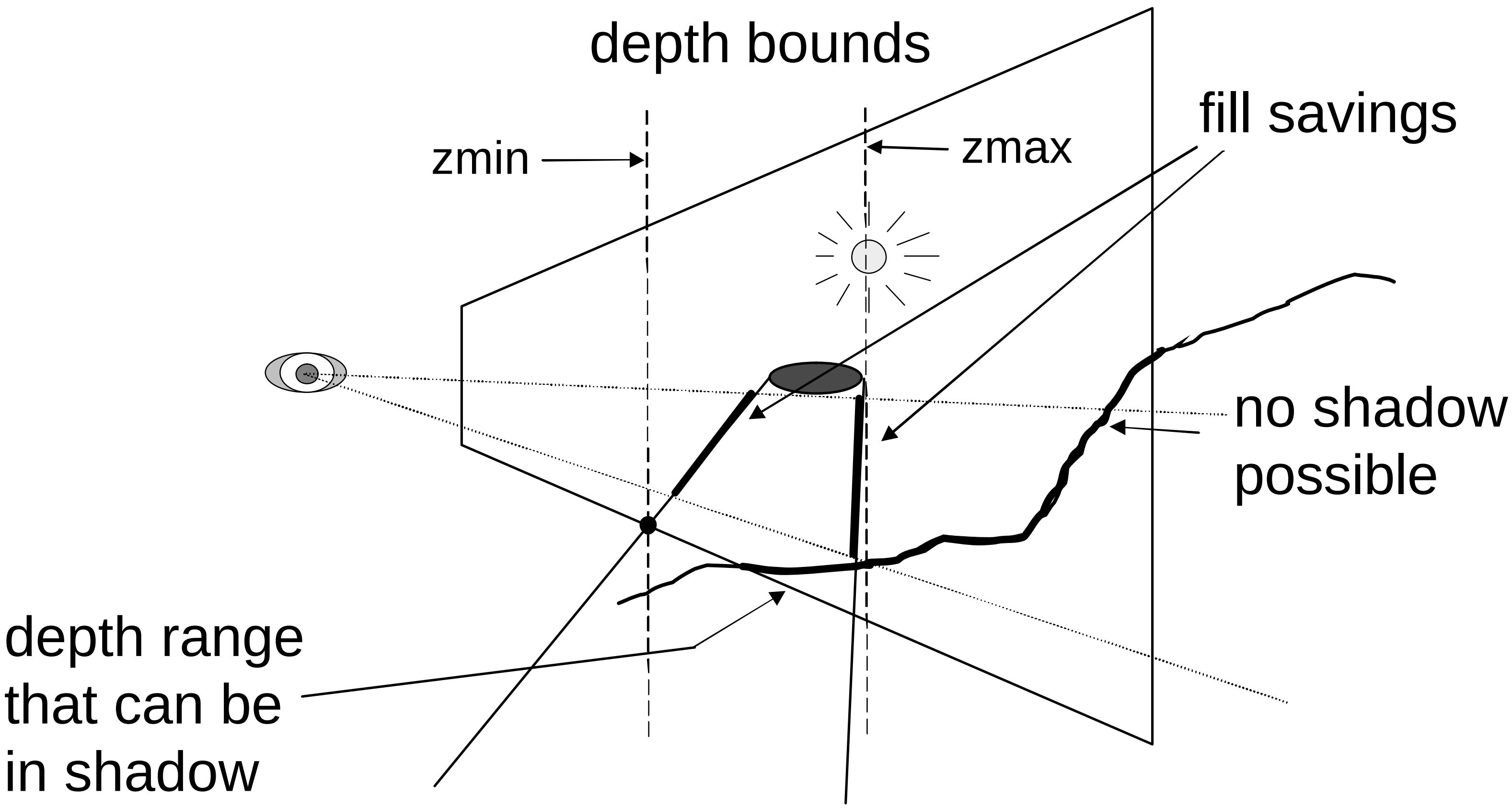


Depth Bounds Test

- Discards fragments when the depth of the pixel in the depth buffer (***not***) the depth of the fragment) is outside the specified bounds
- Exposed in upcoming NV_depth_bounds_test extension
- Simple API
  - `glDepthBoundsNV( GLclampd zmin, GLclampd zmax );`
  - `glEnable( GL_DEPTH_BOUNDS_TEST_NV );`
  - `glDisable( GL_DEPTH_BOUNDS_TEST_NV );`
- The depths given are in the same [0,1] range as arguments given to glDepthRange()

Depth Bounds Test (2)

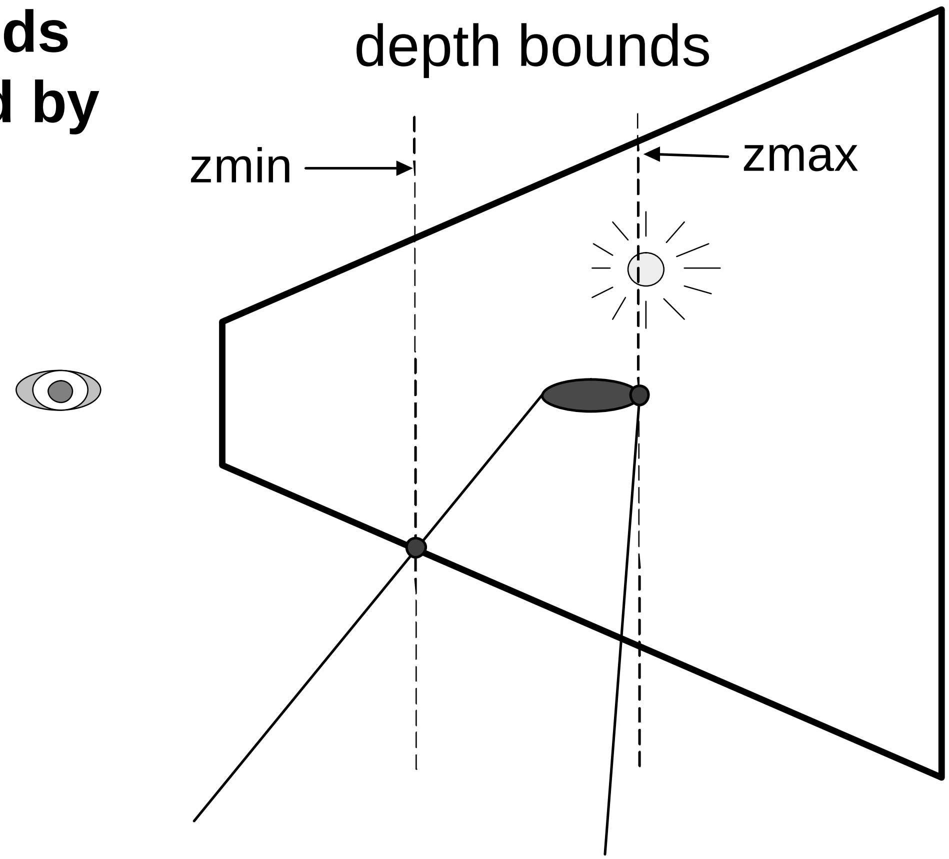
- **Some depth values cannot be in shadow, so why bother to INCR and DECR their stencil index?**



Computing Depth Bounds

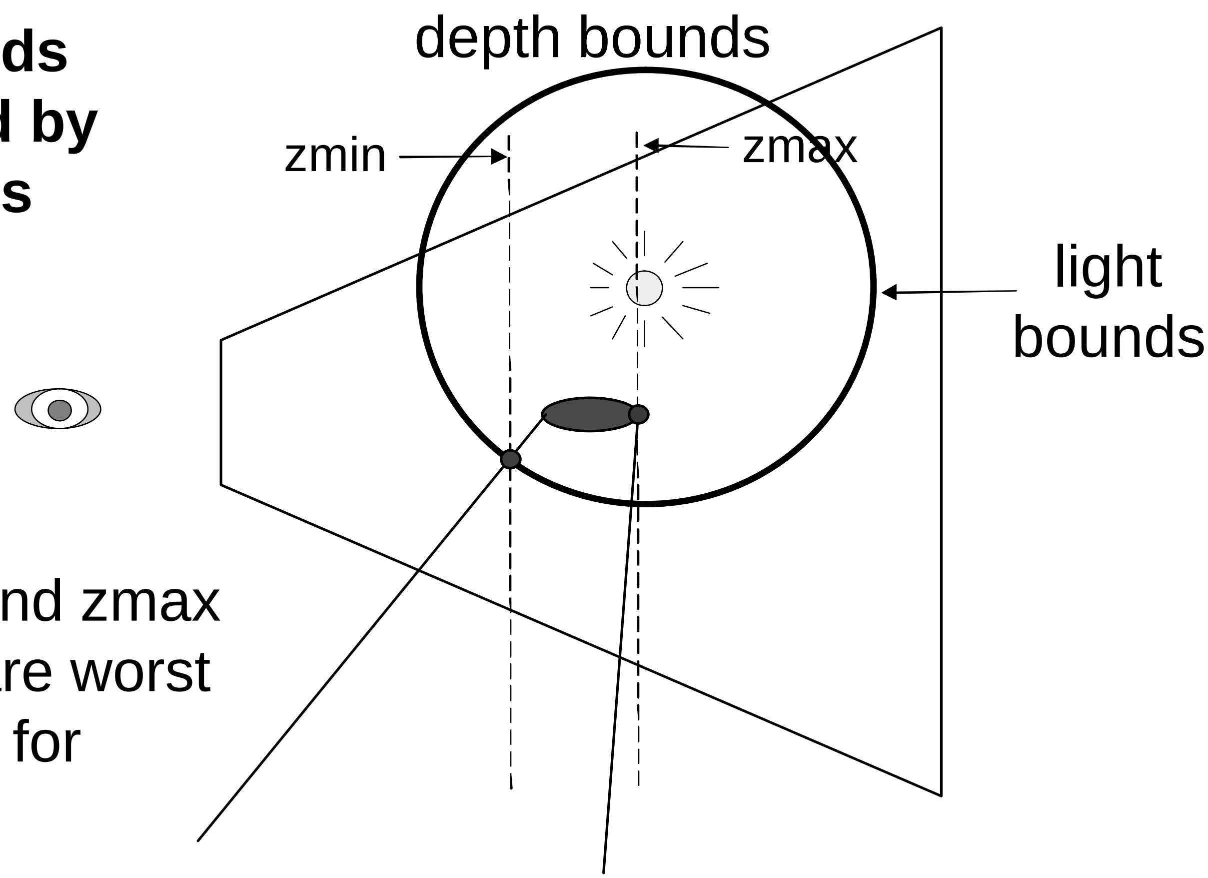
- **Shadow volumes are infinite, but the depth bounds for possible shadows are constrained**

depth bounds constrained by frustum



Computing Depth Bounds

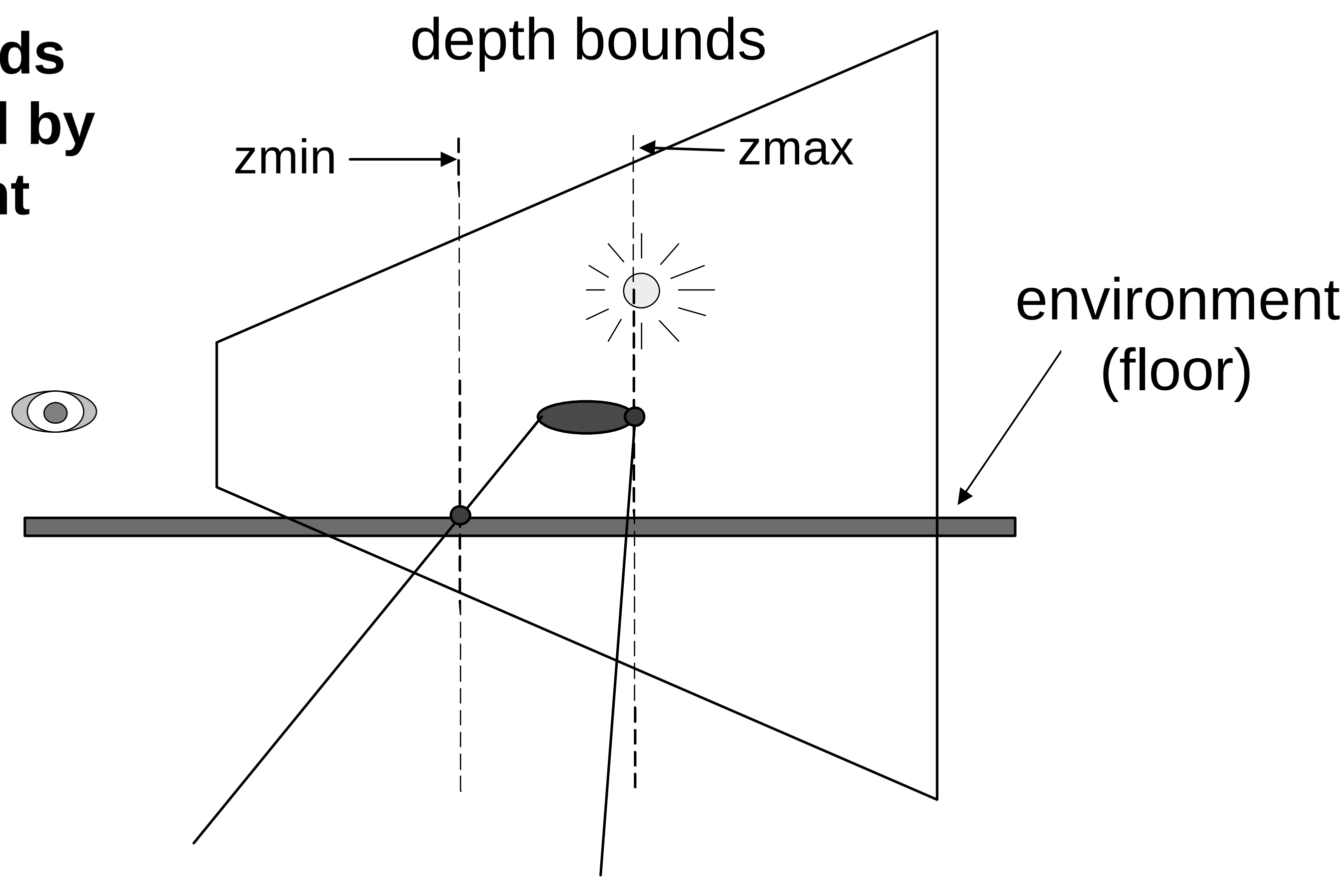
depth bounds constrained by light bounds



Note that zmin and zmax of light bounds are worst case constraints for bounded lights.

Computing Depth Bounds

depth bounds constrained by environment



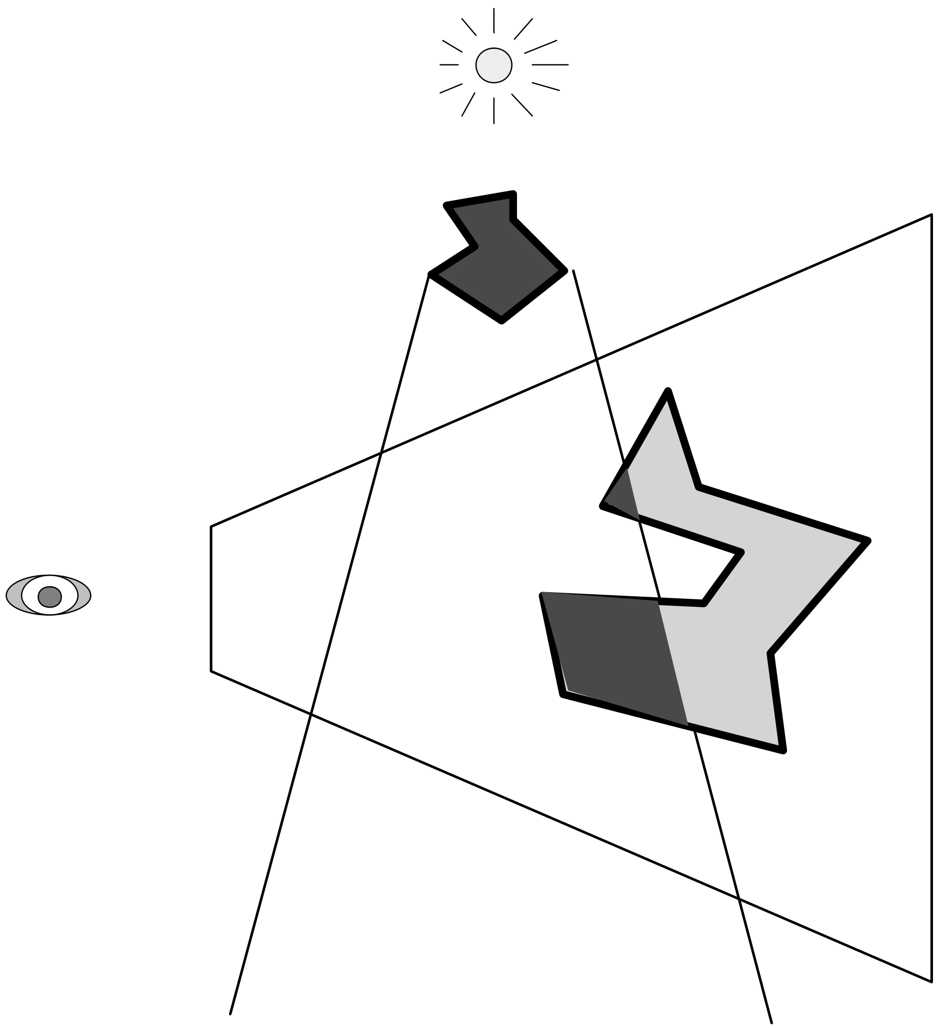
Culling Optimizations

- Shadow volume culling
  - Culling the shadow volume extrusion
  - Specialized shadow volume culling for Zfail
    - Culling the near caps
    - Culling the far caps
- Portal-based culling

Shadow Volume Culling

- Conventional view frustum culling
  - Simple idea: If object is not within view frustum, don’t draw object
  - Use conservative bounding geometry to efficiently determine when object is outside the view frustum
    - Often done hierarchically
- Shadow volume culling
  - Just because an object is not within the view frustum doesn’t mean it can’t cast a shadow that *is* visible within the view frustum!

Shadow Volume Culling Example



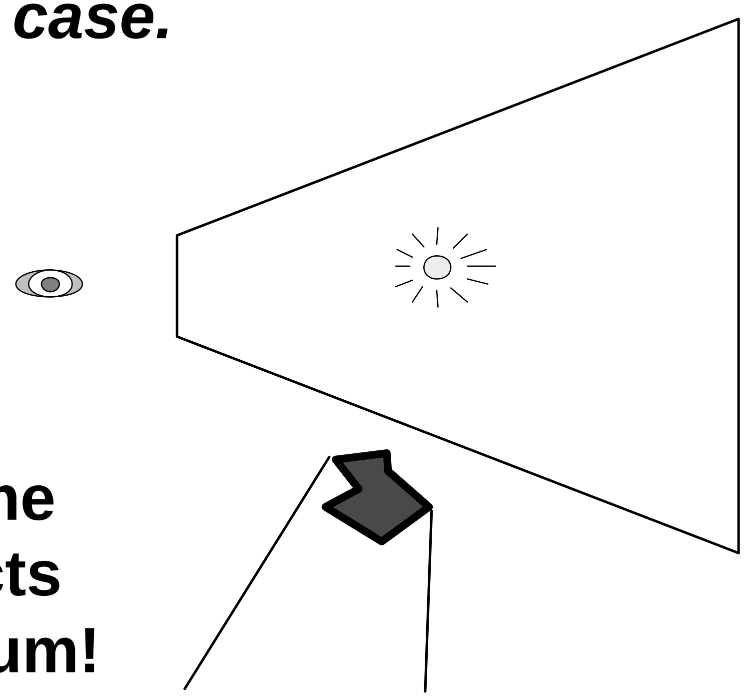
Light and occluder both outside the view frustum.

But occluder still casts shadow *into* the view frustum.

Must consider shadow volume of the occluder even though the occluder could be itself view frustum culled!

Shadow Volume Culling Easy Case

- ***If the light source is within the view frustum <u>and</u> the occluder is outside the view frustum, there's no occluder's shadow is within the view frustum. Culling (not rendering) the shadow volume is correct in this case.***
- **Example:**

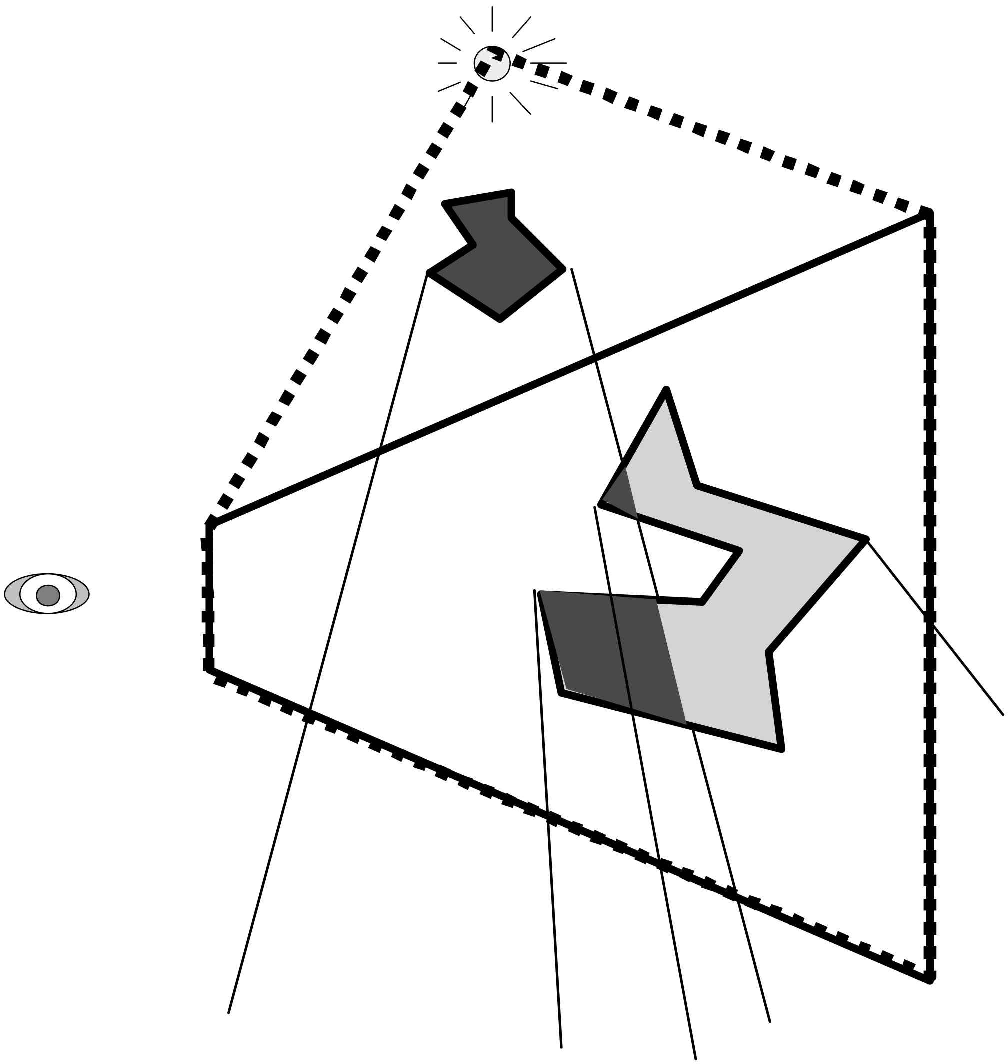


Shadow volume never intersects the view frustum!

Generalized Shadow Volume Culling Rule

- **Form the convex hull of the light position and the view frustum**
  - **For an infinite light, this could be an infinite convex hull**
  - **If your far plane is at infinity due to using a P_{inf} projection matrix, the view frustum is infinite**
    - **The convex hull including the light (even if local) is also infinite**
- **If an occluder is within this hull, you must draw the shadow volume for an occluder**

Shadow Volume Culling Example Reconsidered (1)

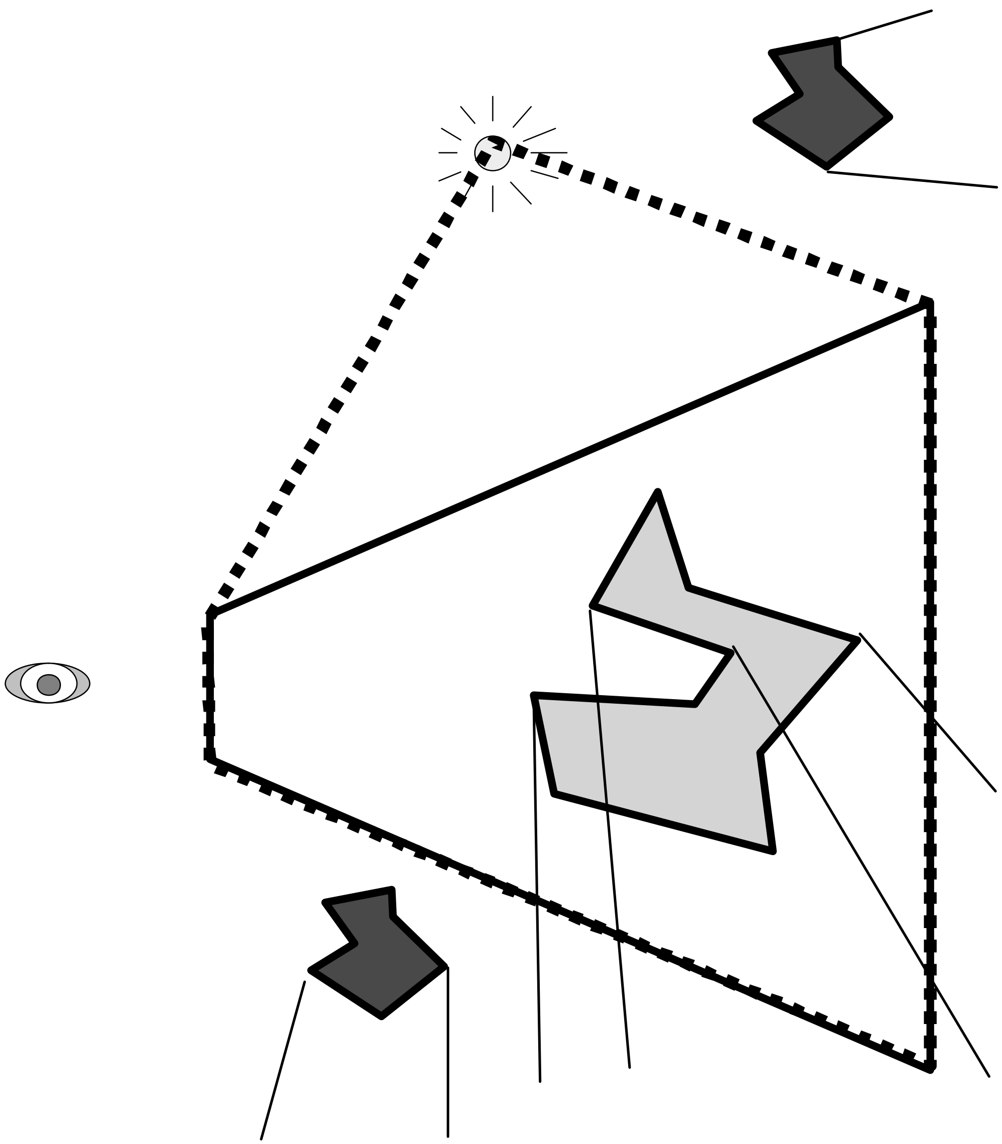


Light and occluder both outside the view frustum.

But occluders still within convex hull formed by the light and view frustum

Must consider shadow volume of the purple occluder even though the purple occluder could be itself view frustum culled!

Shadow Volume Culling Example Reconsidered (2)



Light and occluder both outside the view frustum.

But purple occluders still **also** outside convex hull formed by the light and view frustum

No need to render shadow volume of the purple occluders or the purple occluders!

Green occluder must be considered however!

Portal-based Shadow Volume Optimizations

- Portal systems are convenient for
  - quickly identifying visible things
  - quickly eliminating hidden things
- For bounded lights, we can treat the light bounds as the “visible object” we’re testing for
  - If the light bounds are visible, we need to process the light
  - If the light bounds are invisible, we can safely skip the light

Silhouette Determination Optimizations

- Pre-computing shadow volumes for static occluders & static lights
- Efficient data structures for static occluder & dynamic light
- Simplified occluder geometry

Precompute Shadow Volumes for Static Occluders and Static Lights

- **Shadows are view-independent**
- **If occluder & light are static with respect to each other, shadow volumes can be precomputed**
  - **Example: Headlights on a car have a static shadow volume w.r.t. the car itself as an occluder**
- **Precomputation of shadow volumes can include bounding shadow volumes to the light bounds**

Efficient Data Structures for Static Occluder and Dynamic Light Interactions

- Brute force algorithm for possible silhouette detection inspects every edge to see if triangles joining form a possible silhouette edge
  - Always works, but expensive for large models
  - Perhaps the best practical approach for dynamic models
- But static occluders could exploit precomputation
  - See SIGGRAPH 2000 paper “Silhouette Clipping” by Sander, Gui, Gortler, Hoppe, and Snyder
    - Check out the “Fast Silhouette Extraction” section
    - Data structure is useful for fast shadow volume possible silhouette edge determination

Simplified Occluder Geometry

- More geometrically complex models can generate lots more possible silhouette edges
- More triangles in model means
  - More time spent searching for possible silhouette edges
- More possible silhouette edges means
  - More fill rate consumed by shadow volumes
- Consider substituting simplified model for shadow volume construction
  - Simplified substitute must “fit within” the original model

Shadow Volume Rendering Optimizations

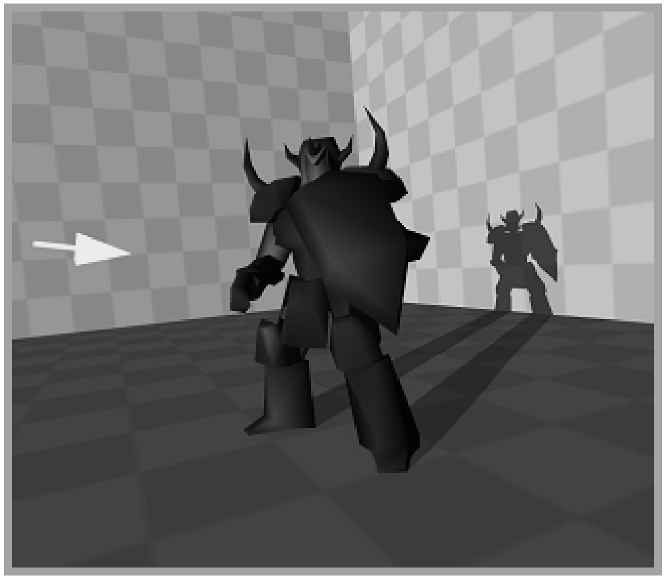
- Compute Silhouette Loops on CPU
  - Only render silhouette for zpass
  - Vertex transform sharing
- Extrude Triangles instead of quads
  - For all directional dights
  - For zpass SVs of local lights
- Wrapping stencil to avoid overflows
- Two-sided stencil testing
- Vertex programs for shadow volume rendering

Avoid Transforming Shadow Volume Vertices Redundantly

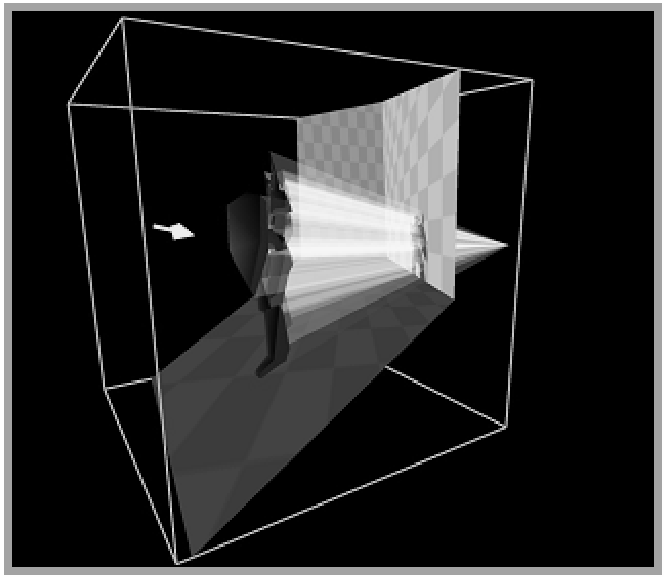
- Use GL_QUAD_STRIP rather than GL_QUADS to render extruded shadow volume quads
  - Requires determining possible silhouette loop connectivity
  - Once you find an edge, look for connected possible silhouette edge loops until you form a loop

Shadow Volume Extrusion using Triangles or Triangle Fans

- **Extrusion can be rendered with a GL_TRIANGLE_FAN**
  - **Directional light's shadow volume always projects to a single point at infinity**
  - **Same is true for local lights – except the point is the "external" version of the light's position (caveats)**



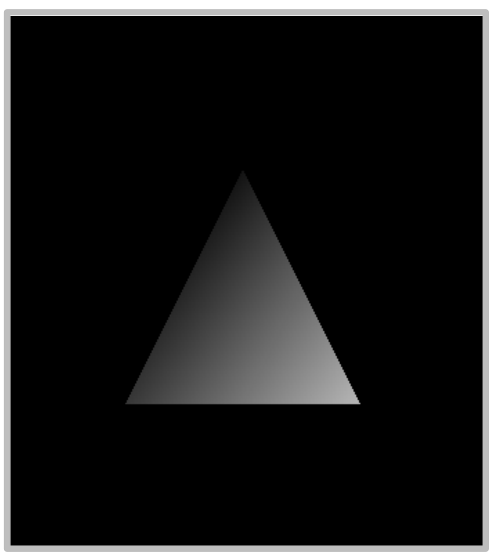
Scene with directional light.



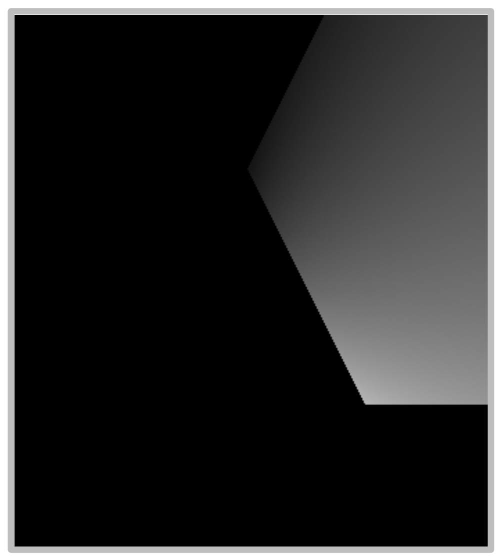
Clip-space view of shadow volume

Shadow Volume Extrusion using Triangle Fans

- **For Local Lights, triangle fans can be used to render zpass (uncapped) shadow volumes**
  - **if the light position is (Lx, Ly, Lz, 1), the center of the triangle fan is the “external” point (-Lx, -Ly, -Lz, -1)**
  - **this is simpler in terms of the geometry**
  - **can offer some fill efficiencies as well**



red vertex is **internal**



red vertex is **external**

What is an external triangle?

A triangle where one or two vertices has w<0

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*´2^N, *3*´2^N**, etc. can alias to zero**

background

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

background

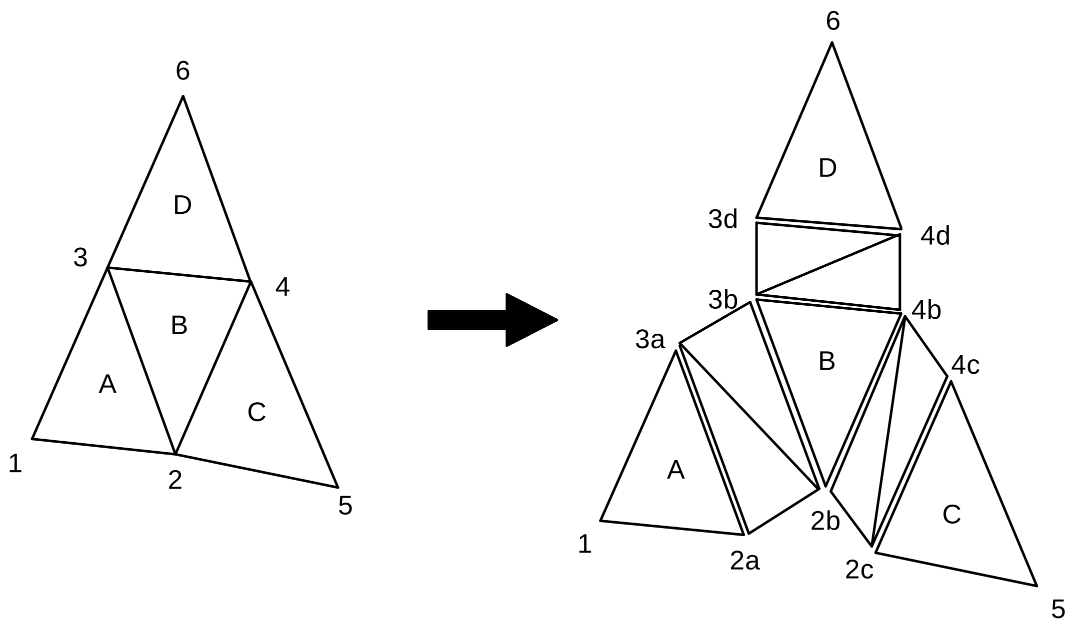
Vertex Programs for Shadow Volumes

- Three techniques to consider
  - Fully automatic shadow volume extrusion
    - Everything off-loaded to GPU
    - Quite inefficient, not recommended
  - Vertex normal-based extrusion
    - Prone to tessellation anomalies
  - Semi-automatic shadow volume extrusion
    - CPU performs possible silhouette edge detection for each light
    - GPU projects out quads from single set of vertex data based on light position parameter
    - Doom3's approach

Fully Automatic Shadow Volume Extrusion (1)

- 3 unique vertexes for each triangle of the original triangle mesh
- Insert degenerate “edge quads” at each & every edge of the original triangle mesh
  - This needs a LOT of extra vertices
    - Primary reason this technique is impractical
    - Too much transform & triangle setup overhead
- No way to do zpass cap optimizations
- No way to do triangle extrusion optimizations

Bloating the original triangle mesh



Formula for geometry:

$v_{bloat} = 3 * t_{orig}$

$t_{bloat} = t_{orig} + 2 * e_{orig}$

Bloated geometry based only on number of *triangles* and *edges* of original geometry.

Original triangle mesh
6 vertexes
4 triangles

Bloated triangle mesh
12 vertexes
10 triangles

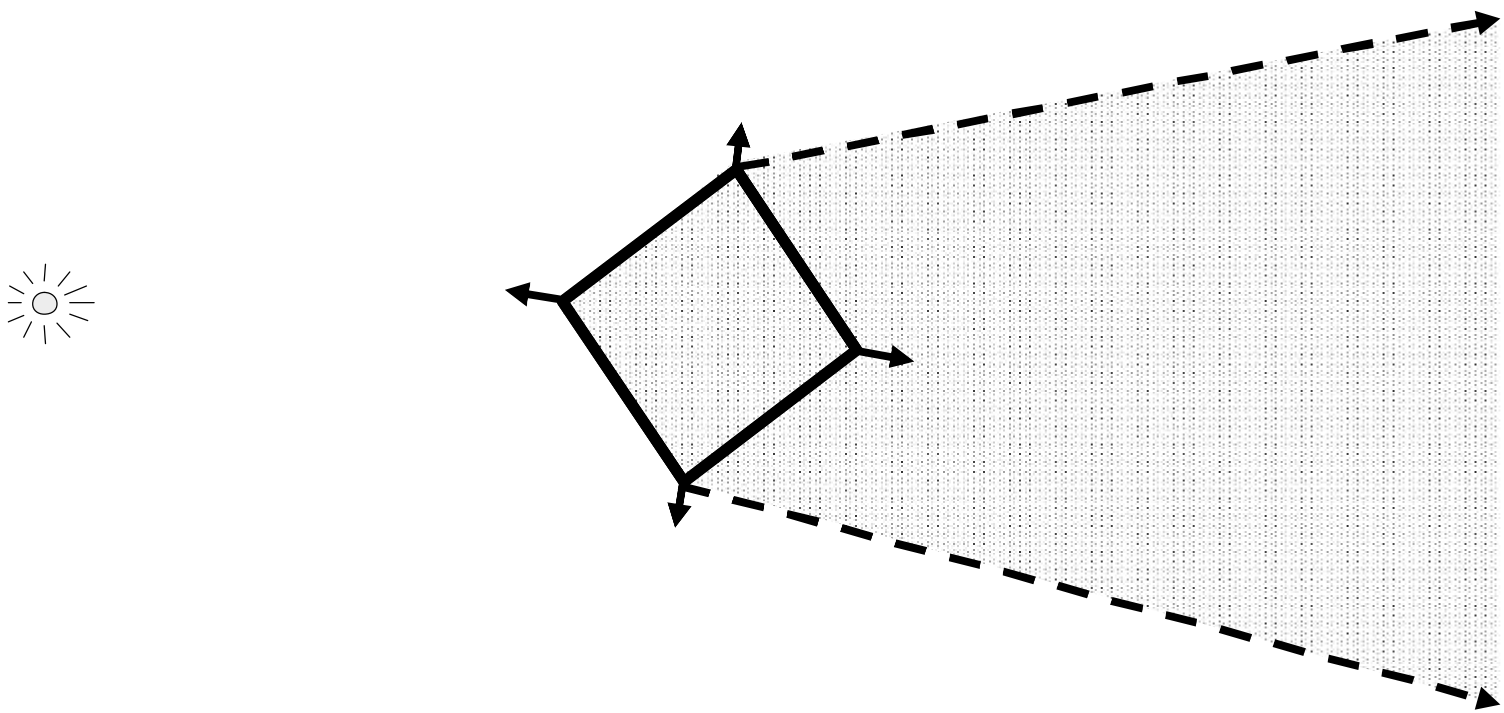
A *lot* of extra geometry!

Vertex Normal-based Extrusion (1)

- Use per-vertex normals
  - If $N \cdot L$ is greater or equal to zero, leave vertex alone
  - If $N \cdot L$ is less than zero, project vertex away from light
- Advantages
  - Simple, requires only per-vertex computations
  - Maps well to vertex program
- Disadvantages
  - *Too* simple – only per-vertex
  - Best when normals approximate facet normals well
  - Worst on faceted, jaggy, or low-polygon count geometry
  - No way to do zpass cap optimizations
  - No way to do triangle extrusion optimizations

Vertex Normal-based Extrusion (2)

- **Example breakdown case**

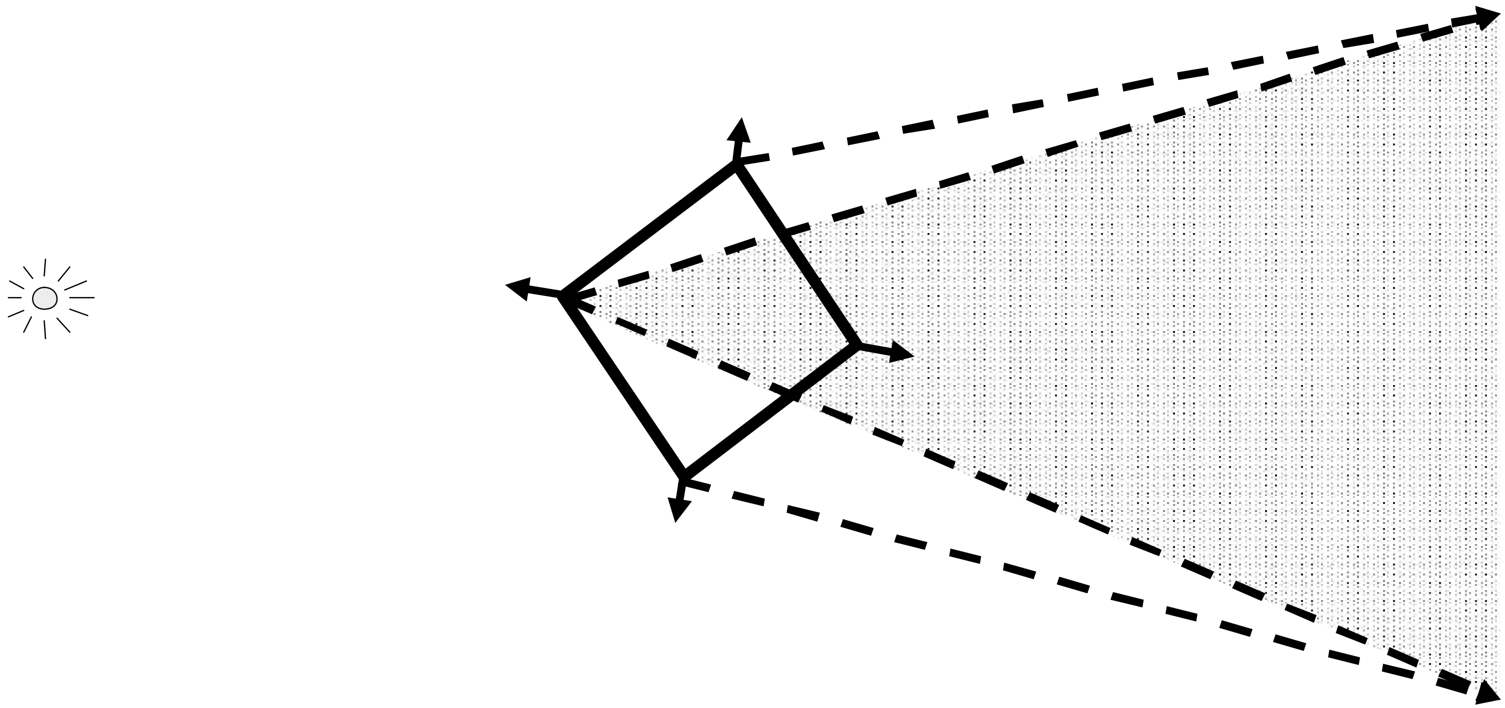


Facet-based SV Extrusion (correct)

Vertex Normal-based SV Extrusion (incorrect)

Vertex Normal-based Extrusion (3)

- **Example breakdown case**

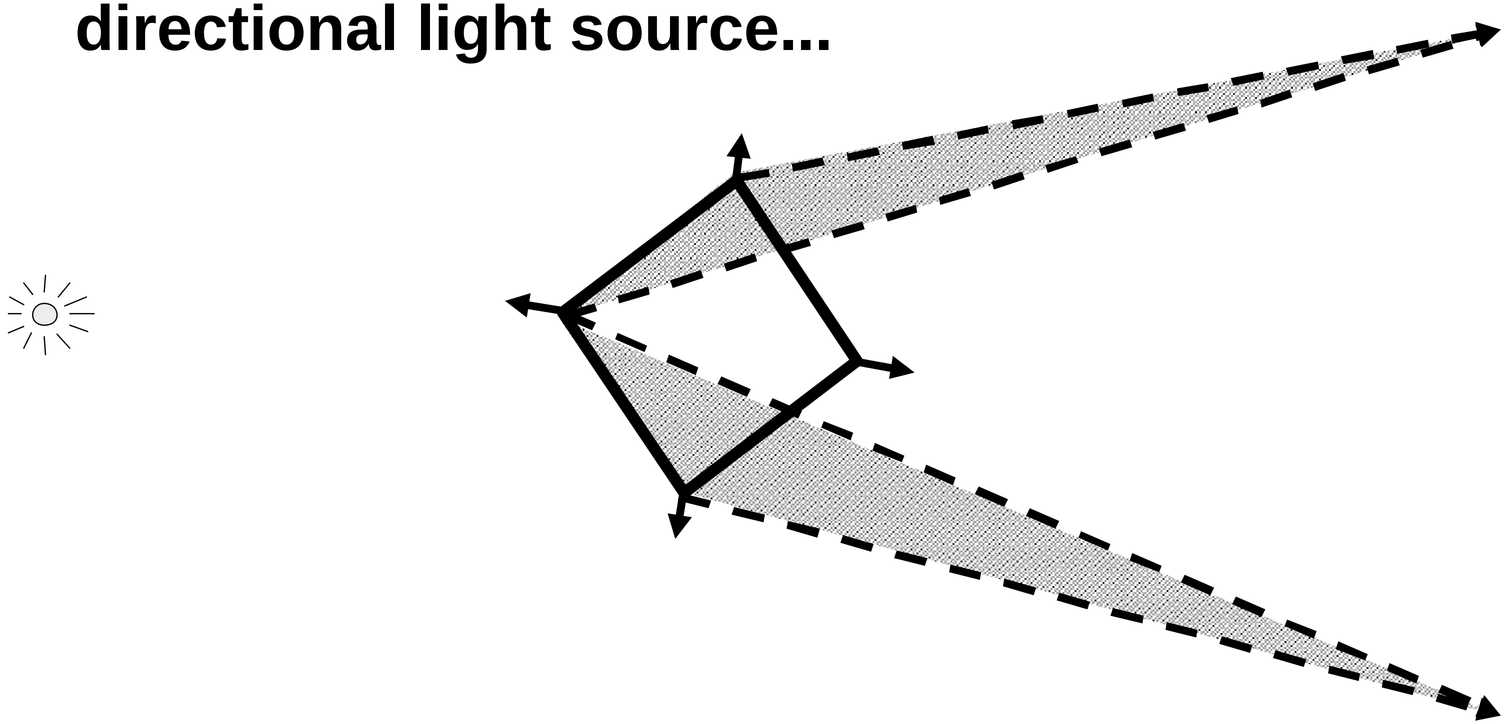


Facet-based SV Extrusion (correct)

Vertex Normal-based SV Extrusion (incorrect)

Vertex Normal-based Extrusion (4)

- **This shadow volume would collapse with a directional light source...**



Facet-based SV Extrusion (correct)

Vertex Normal-based SV Extrusion (incorrect)

Incorrectly unshadowed region (incorrect)

Vertex Normal-based Extrusion (3)

- Per-vertex normals are typically intended for lighting computations, not shadowing
  - Vertex normals convey curvature well, but not orientation
- Shadowing really should be based on facet orientations
  - That is, the normal for the triangle, not a vertex
- Hack compromise
  - Insert duplicate vertices at the same position, but with different normals

Semi-automatic Shadow Volume Extrusion (1)

- **Possible silhouette edge detection done by CPU**
  - **For each light**
- **Want to only have a single set of vertices for each model**
  - **As opposed to a unique set of possible silhouette edge of vertices <u>per</u> light <u>per</u> model**
- **Vertex program projects a single set of vertices appropriate for each possible silhouette edge for any particular light**

Semi-automatic
Shadow Volume Extrusion (2)

- Make two copies of every vertex,
  each with 4 components (x,y,z,w)
  1. v = (x,y,z,1)
  2. v’ = (x,y,z,0)
- For a possible silhouette edge with vertices p & q
  - Assume light position L
  - Render a quad using vertices p, q, p’, q’
  - Vertex program computes

```
pos = pos´pos.w +
        (pos´L.w - L´pos.w)´(1-pos.w)
```

- Straightforward to handle caps too

Semi-automatic Shadow Volume Extrusion (3)

- Vertex array memory required = 8 floats / vertex
  - Independent of number of lights
  - Vertex program required is very short
  - MUCH less vertex array memory than fully automatic approach

Shadow Volume Polygon Rendering Order (1)

- Naïve approach
  - Simply render cap & projection shadow volume polygons in "memory order" of the edges and polygons in CPU memory
  - Disadvantages
    - Potentially poor rasterization locality for stencil buffer updates
    - Typically sub-par vertex re-use
  - Advantages
    - Friendly to memory pre-fetching
    - Obvious, easy to implement approach

Shadow Volume Polygon Rendering Order (2)

- GPU Optimized Approach
  - Possible silhouette edges form loops
    - Render the projected edge shadow volume quads in "loop order"
    - Once you find a possible silhouette edge, search for another edge sharing a vertex – continue until you get back to your original edge
  - When you must render finite and infinite caps
    - Greedily search for adjacent cap polygons
    - Continue until the cap polygons bump into possible silhouette edge loops – then look for more un-rendered capping polygons

Shadow Volume Polygon Rendering Order (3)

- Why use the GPU Optimized Approach?
  - Tends to maximize vertex re-use
    - Avoids retransforming vertices multiple times due to poor locality of reference
  - Maximizes stencil update efficiency
    - Adjacent polygons make better use of memory b/w
  - Convenient for optimizations
    - zpass cap elimination
    - triangle instead of quad extrusion
  - Easy to implement
    - Once you locate a possible silhouette edge, it’s easy to follow the loop
    - Easy to greedily search for adjacent finite & infinite cap polygons

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 History (5)

- **Everitt & Kilgard [2002] Integrate Multiple Ideas into a robust solution**
  - **Dietrich, Bilodeau, 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**
- **NVIDIA Hardware Enhancements**
  - **Depth clamping [2001]: *better depth precision***
  - **Two-sided stencil testing [2002]: *performance***
  - **Depth bounds test [2003]: *performance***