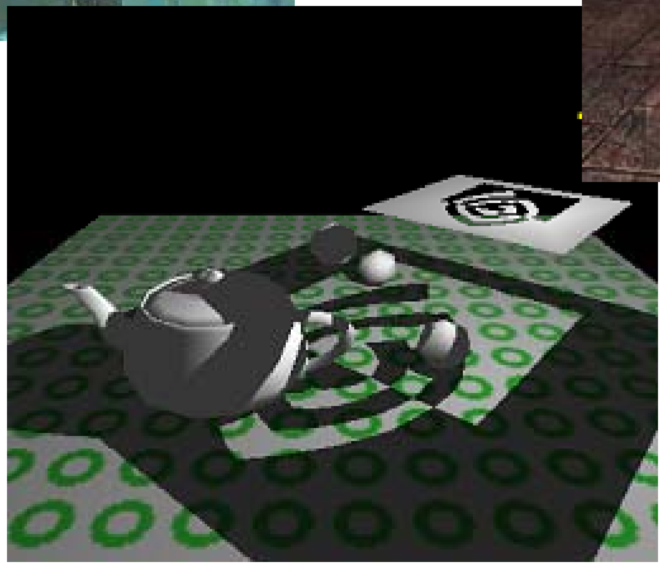
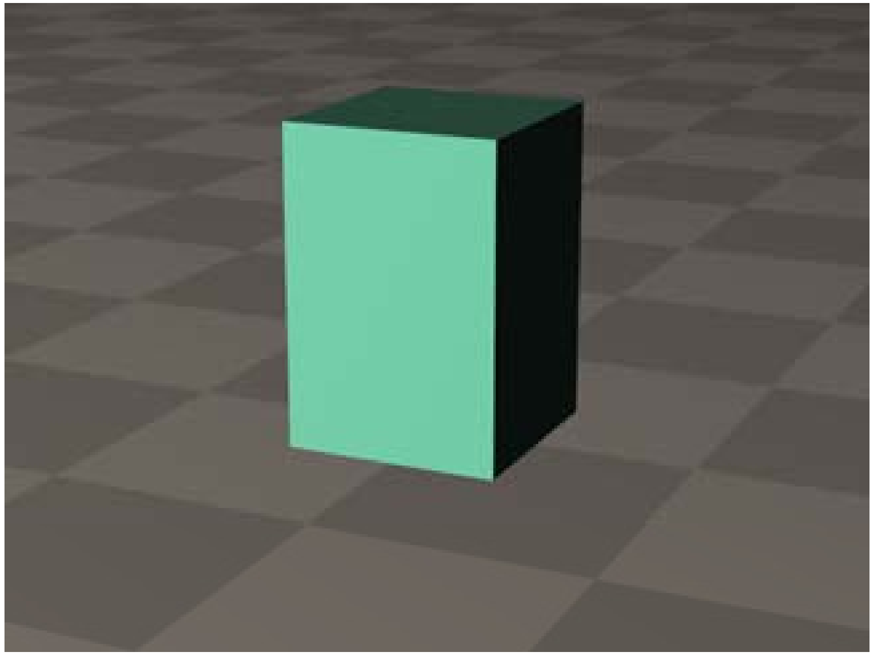
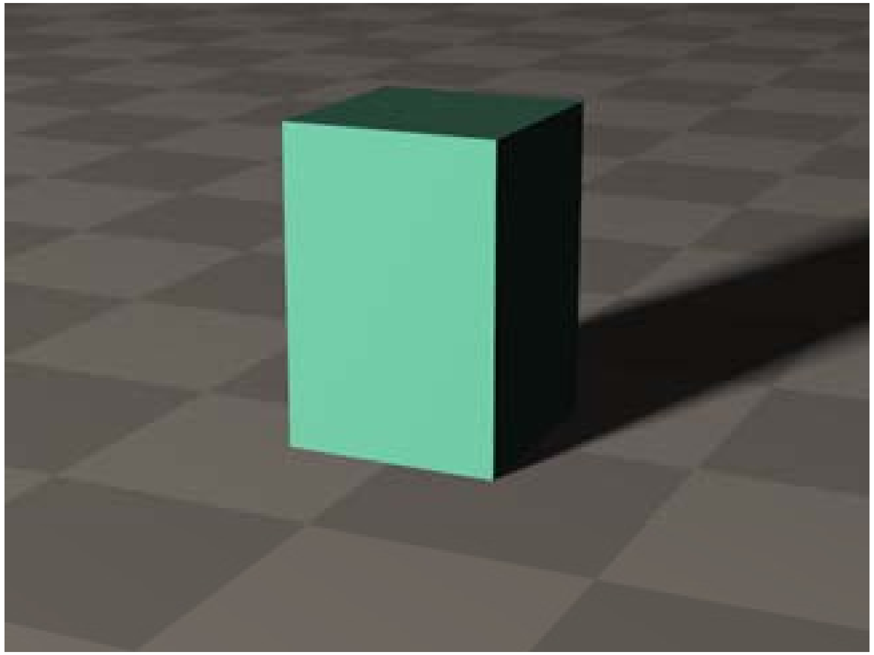
# Shadows



# What for?

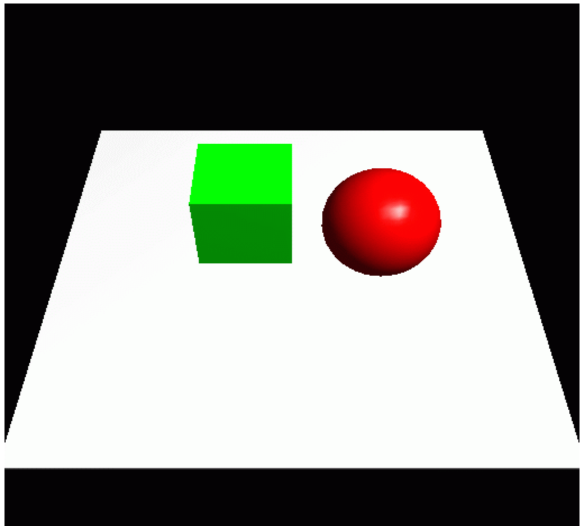
- Shadows tell us about the relative locations and motions of objects

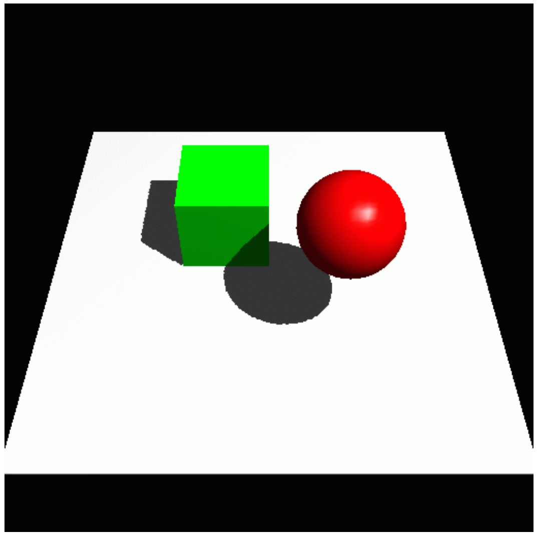




# What for?

- Shadows tell us about the relative locations and motions of objects
- And about light positions





# What for?

- Objects look like they are “floating”

→ Shadows can fix that!

# Motivation

- Shadows contribute significantly to realism of rendered images
  - Anchors objects in scene
- **Global** effect → expensive!
- Light source behaves very similar to camera
  - Is a point visible from the light source?

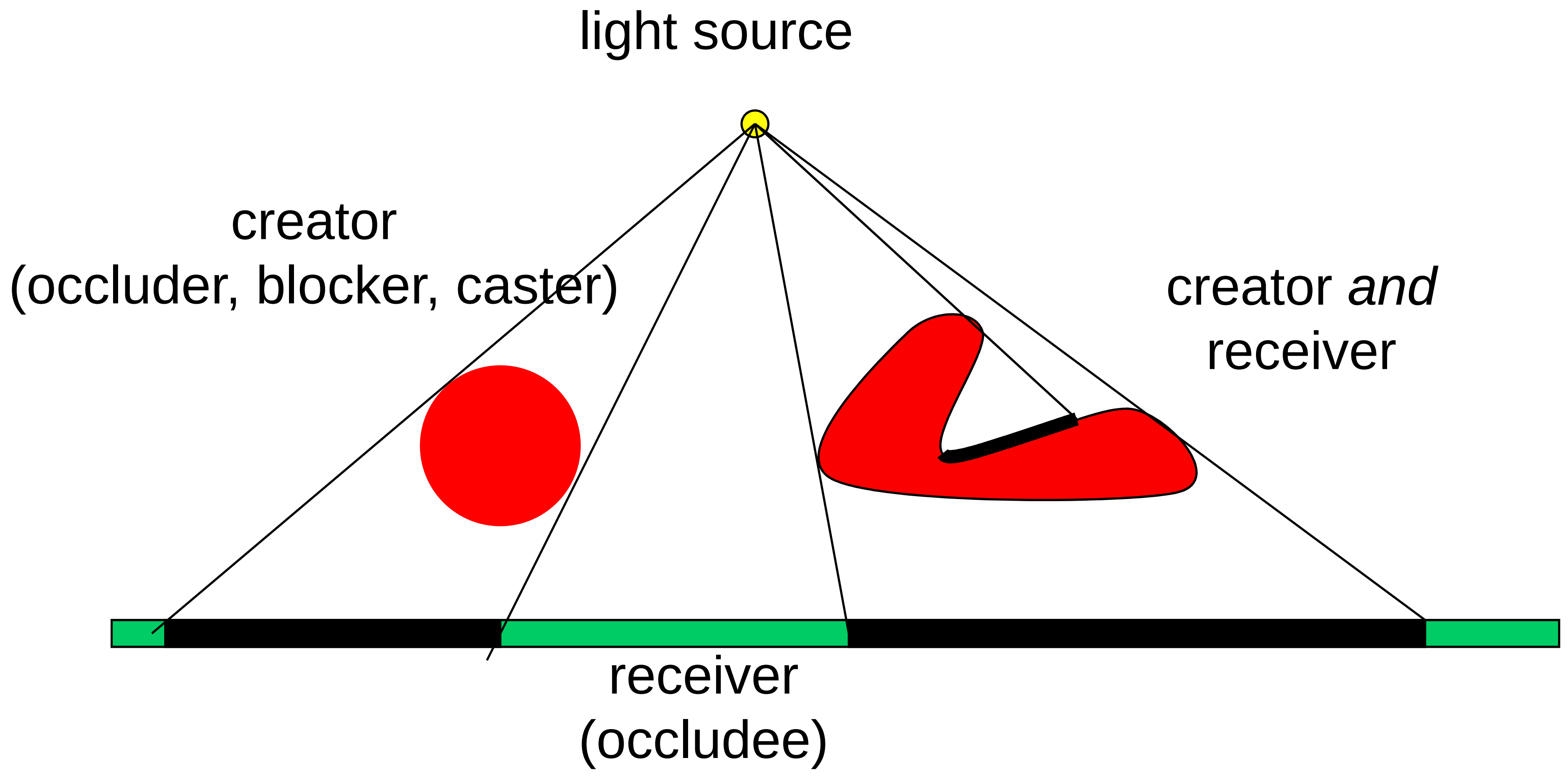
    → shadows are “hidden” regions
  - Shadow is a projection of caster on receiver

    → projection methods
- Best done completely in hardware through shaders

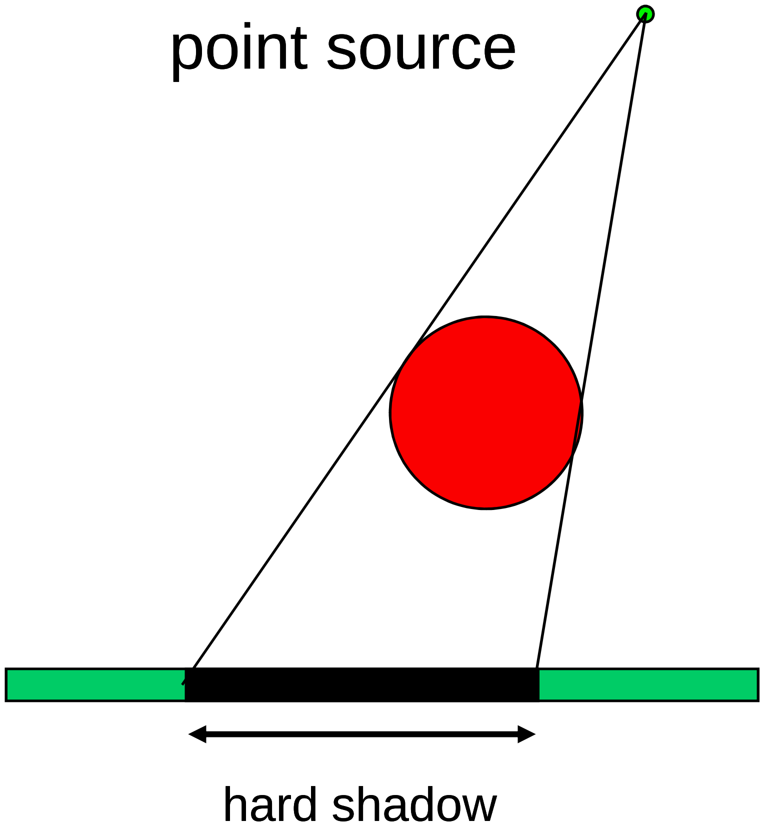
# Shadow Algorithms

- Static shadow algorithms (lights + objects)
  - Radiosity, ray tracing → lightmaps
- Approximate shadows
- Projected shadows (Blinn 88)
- Shadow volumes (Crow 77)
  - Object-space algorithm
- Shadow maps (Williams 78)
  - Projective image-space algorithm
- Soft shadow extensions for all above algorithms
  - Still hot research topic (500+ shadow publications)

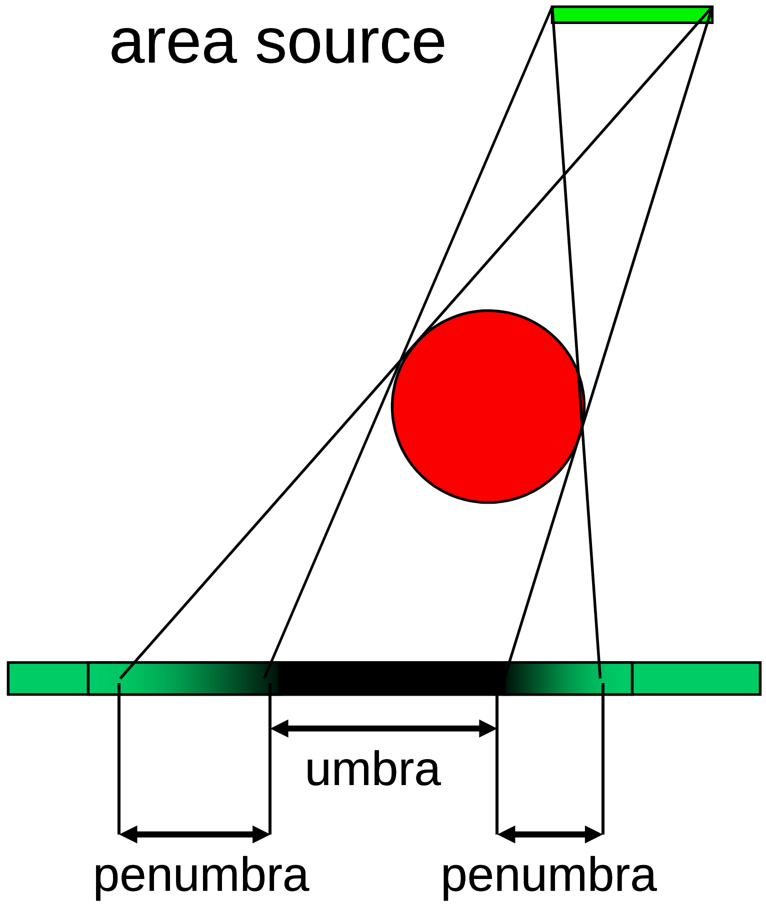
# Shadow Terms



# Hard vs. Soft Shadows




+fast
-only good for localized lights (sun, projectors)
+fake soft shadow through filtering




+ very realistic
- very expensive
+ becomes more and more usable

# Static Shadows

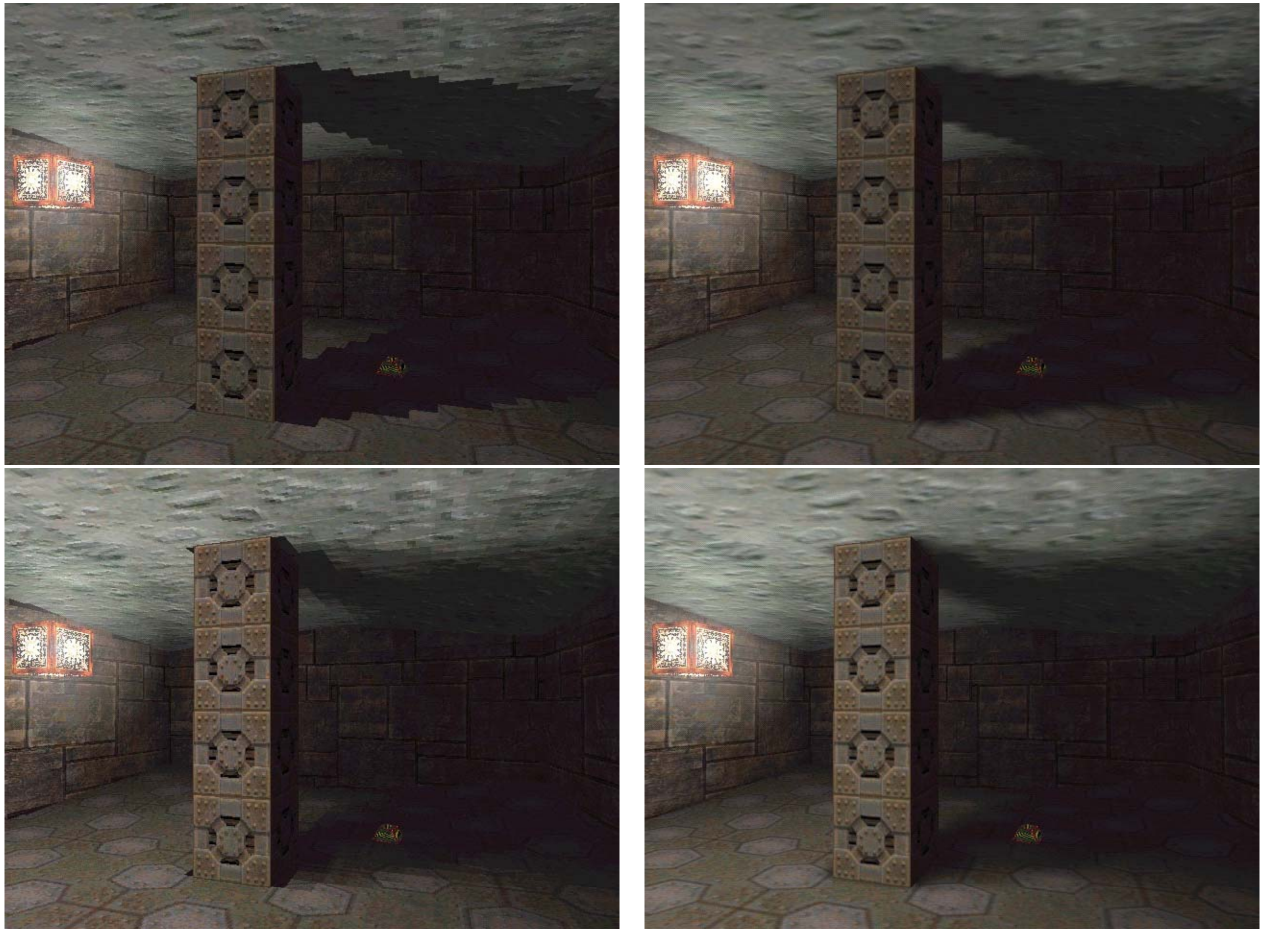
- Glue to surface whatever we want
- Idea: incorporate shadows into light maps
  - For each texel, cast ray to each light source
- Bake soft shadows in light maps
  - Not by texture filtering alone, but:
  - Sample area light sources

# Static Soft Shadow Example

no filtering | filtering

1 sample

n samples



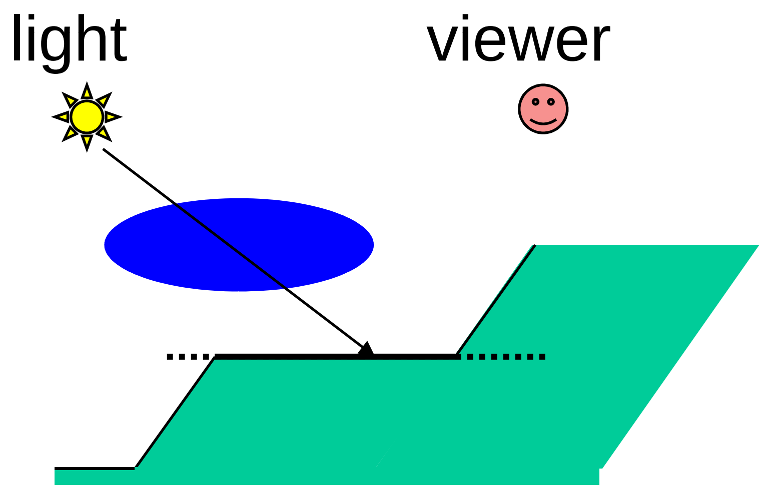
# Approximate Shadows

- Hand-drawn approximate geometry
  - Perceptual studies suggest: shape not so important
  - Minimal cost

# Approximate Shadows

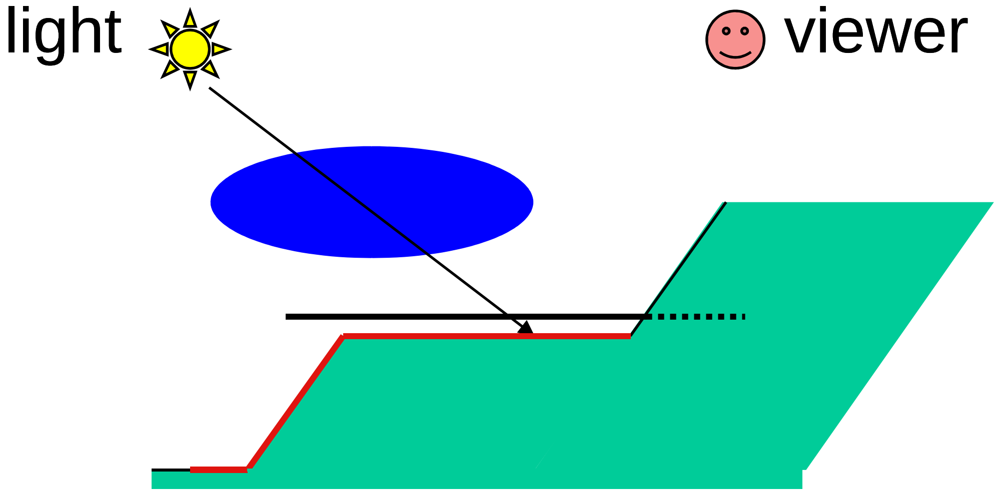
- Dark polygon (maybe with texture)
  - Cast ray from light source through object center
  - Blend polygon into frame buffer at location of hit
  - May apply additional rotation/scale/translation
    - Incorporate distance and receiver orientation
- Problem with z-quantization:

rrors!

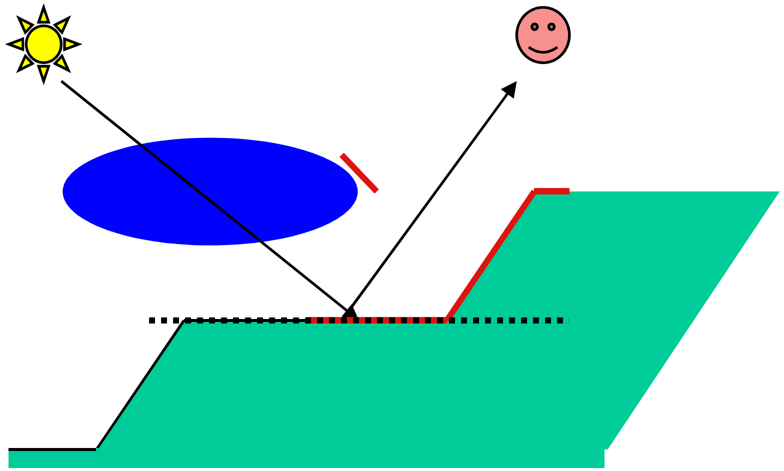


# Approximate Shadows

on

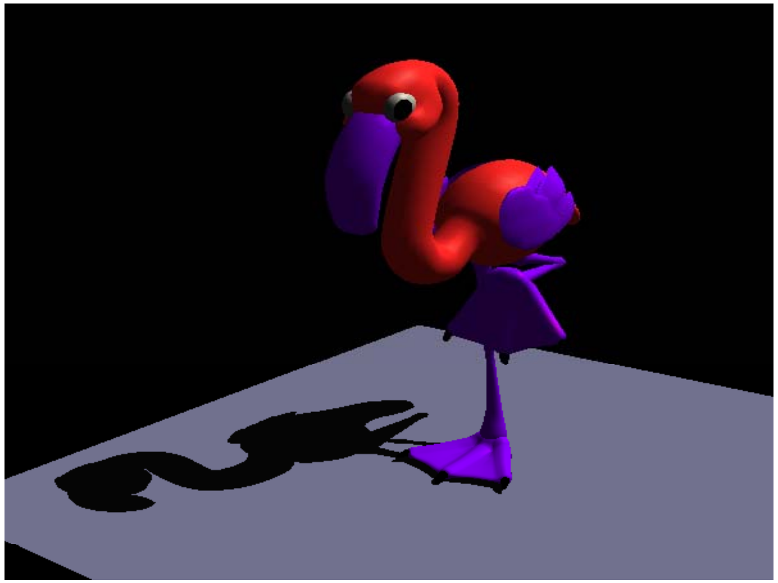



ay



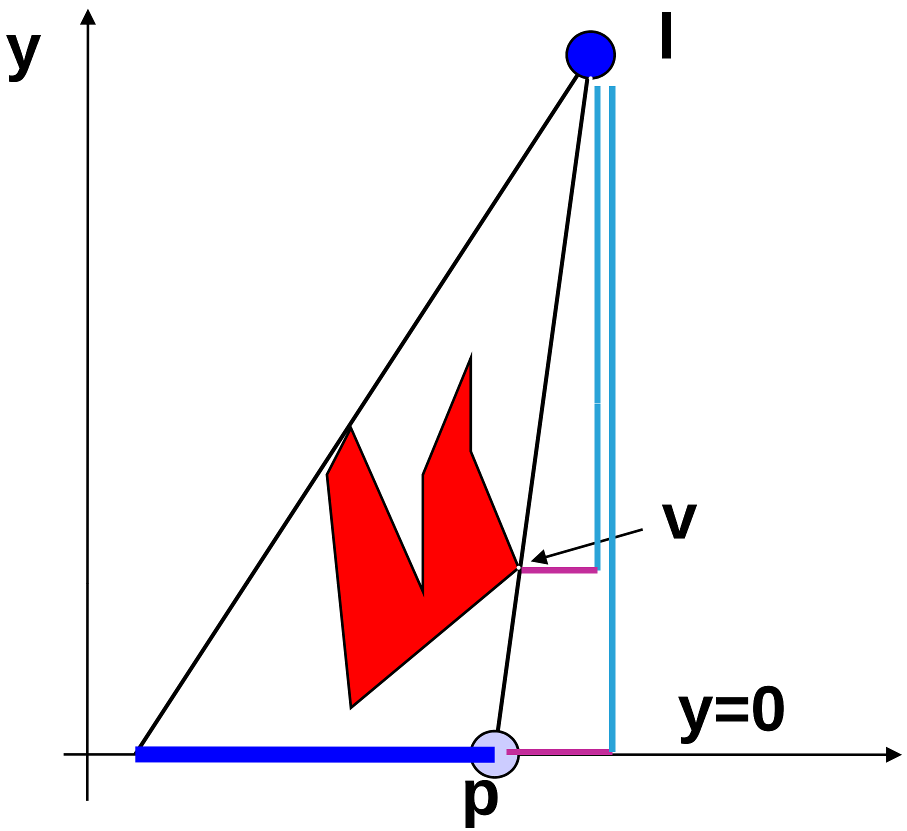
# Projection Shadows (Blinn 88)

- Shadows for selected large ***planar*** receivers
  - Ground plane
  - Walls
- Projective geometry: flatten 3D model onto plane
  - and “darken” using framebuffer blend



# Projection for Ground Plane

- Use similar-triangle tricks




$$\frac{p_x - l_x}{v_x - l_x} = \frac{l_y}{l_y - v_y}$$

$$p_x = \frac{l_y v_x - l_x v_y}{l_y - v_y}$$

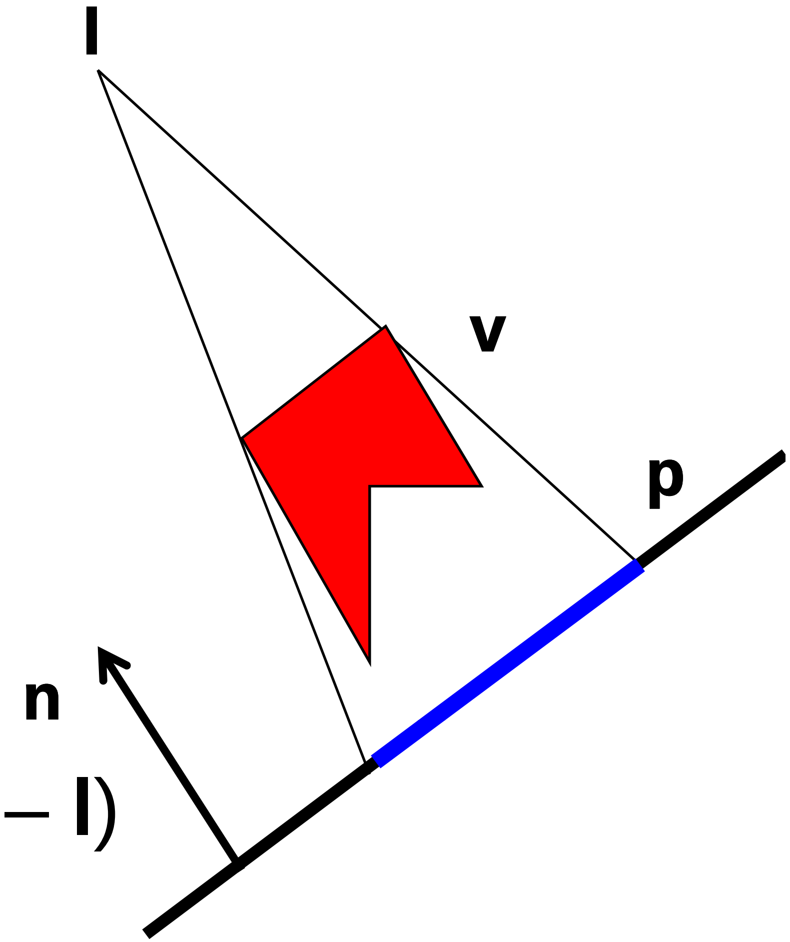
$$p_z = \frac{l_y v_z - l_z v_y}{l_y - v_y}$$

$$p_y = 0$$

# Projection Matrix

- Projective 4x4 matrix:

$$M = \begin{pmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_z & l_y & 0 \\ 0 & -1 & 0 & l_y \end{pmatrix}$$



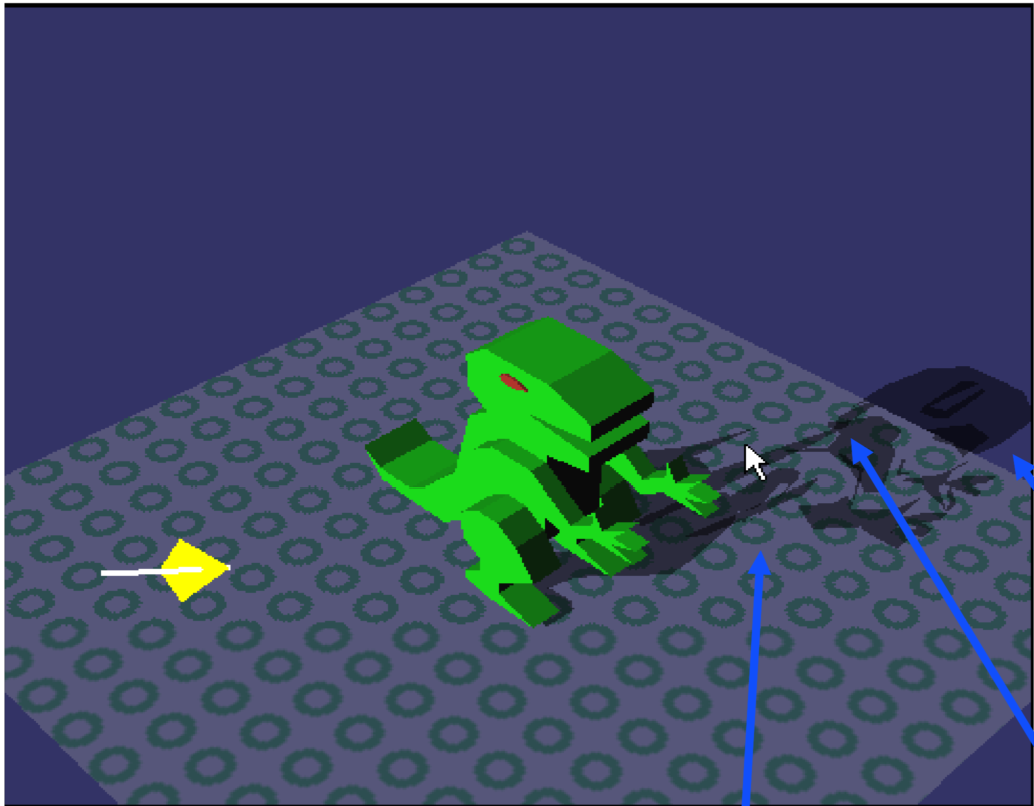

- Arbitrary plane:
  - Intersect line **p** = **l** – α (**v** – **l**)
  - with plane **n x** + d = 0
  - Express result as a 4x4 matrix
- Append this matrix to view transform

# Projection Shadow Algorithm

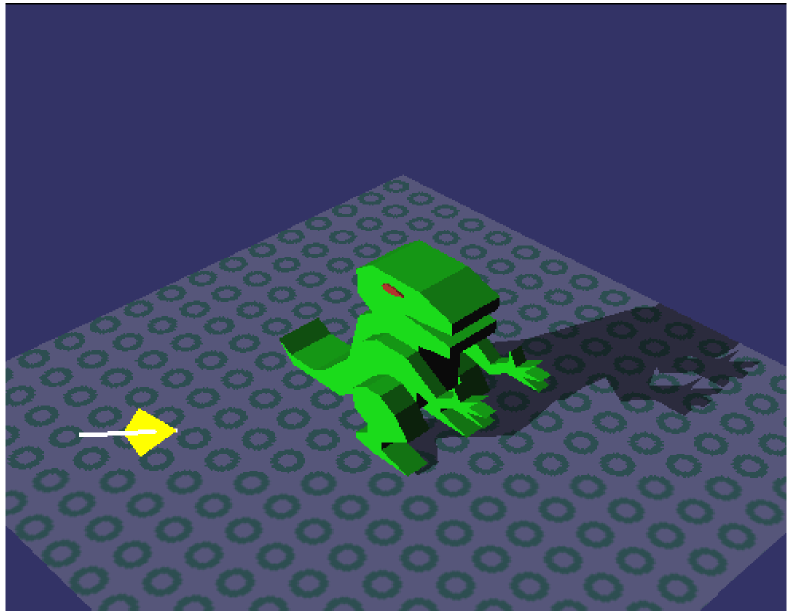
- Render scene (full lighting)
- For each receiver polygon
  - Compute projection matrix M
  - Append to view matrix
  - Render selected shadow caster
    - With framebuffer blending enabled

# Projection Shadow Artifacts

Bad



Good



extends off
ground region

Z fighting

double blending

# Stencil Buffer Projection Shadows

- Stencil can solve all of these problems
  - Separate 8-bit frame buffer for numeric ops
- Stencil buffer algorithm (requires 1 bit):
  - Clear stencil to 0
  - Draw ground polygon last and with
    - `glStencilOp(GL_KEEP, GL_KEEP, GL_ONE);`

      fail zfail pass
  - Draw shadow caster with no depth test but
    - `glStencilFunc(GL_EQUAL, 1, 0xFF);`
      `glStencilOp(GL_KEEP, GL_KEEP, GL_ZERO);`
- Every plane pixel is touched at most once

# Stencil Buffer Planar Reflections

- Draw object twice, second time with:
  - **`glScalef(1, -1, 1)`**
- Reflects through floor


Bad


Good, stencil used to limit reflection.

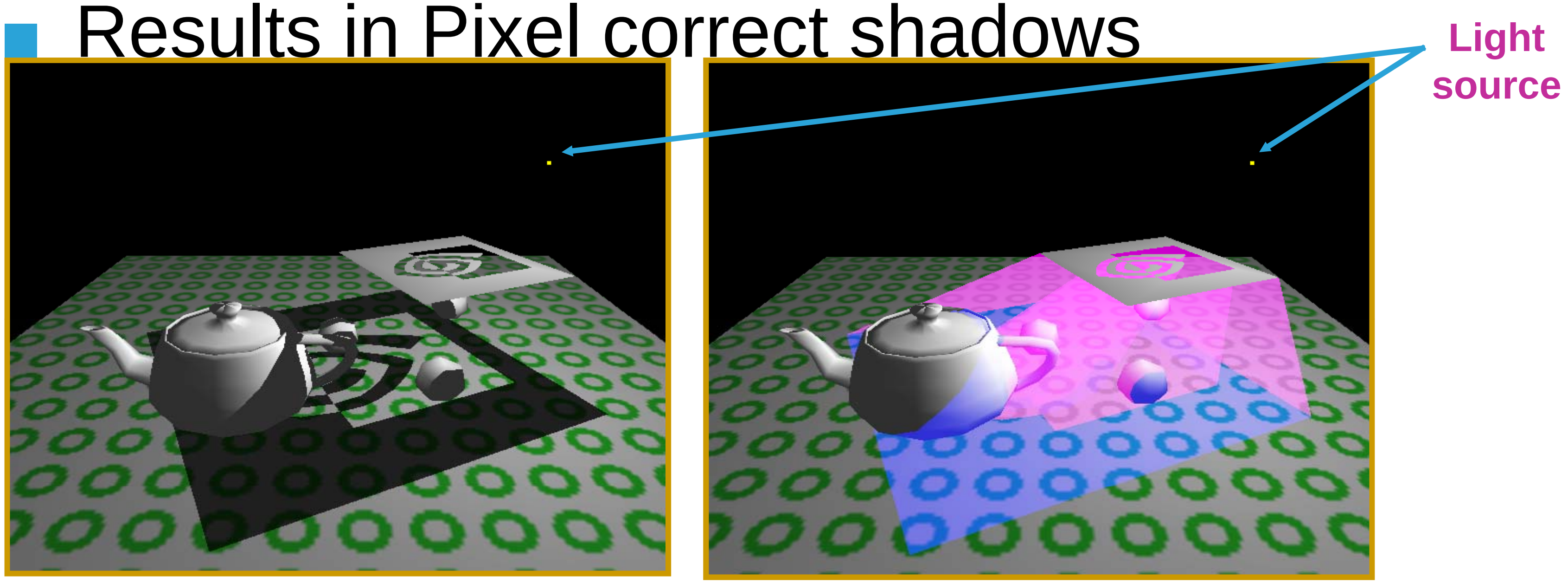
# Projection Shadow Summary

- Easy to implement
  - GLQuake first game to implement it
- Only practical for very few, large receivers
- No self shadowing

- Possible remaining artifacts: wrong shadows
  - Objects behind light source
  - Objects behind receiver

# Shadow Volumes (Crow 1977)

- Occluders and light source cast out a 3D shadow volume
  - Shadow through new geometry
  - Results in Pixel correct shadows




Shadowed scene

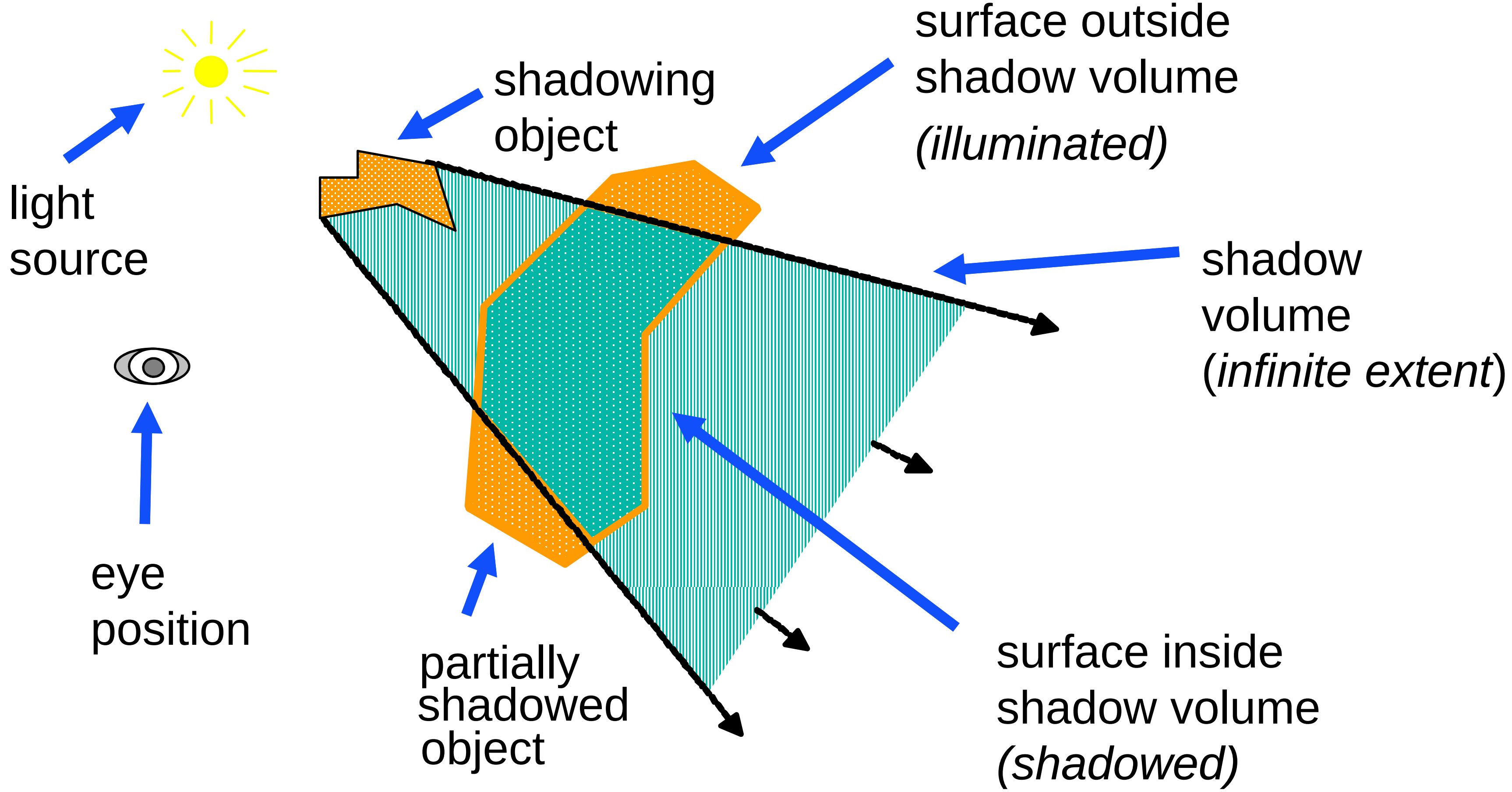
Visualization of shadow volume

# Shadow Volumes (Crow 1977)

- Heavily used in Doom3

# 2D Cutaway of Shadow Volume

- Occluder polygons extruded to semi-infinite volumes

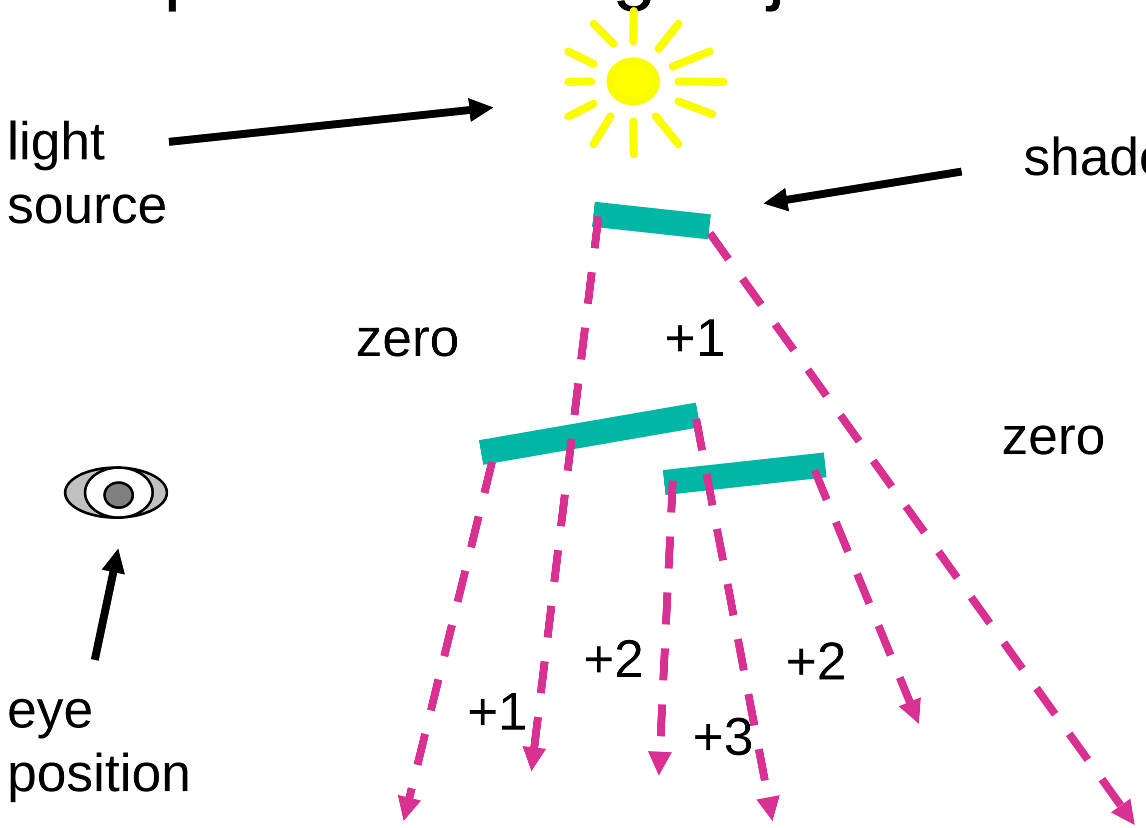


# Shadow Volume Algorithm

- 3D point-in-polyhedron indide-outside test
- Principle similar to 2D point-in-polygon test
  - Choose a point known to be outside the volume
  - Count intersection on ray from test point to known point with polyhedron faces
    - Front face +1
    - Back face -1
  - Like non-zero winding rule!
- Known point will distinguish algorithms:
  - Infinity: “Z-fail” algorithm
  - Eye-point: “Z-pass” algorithm

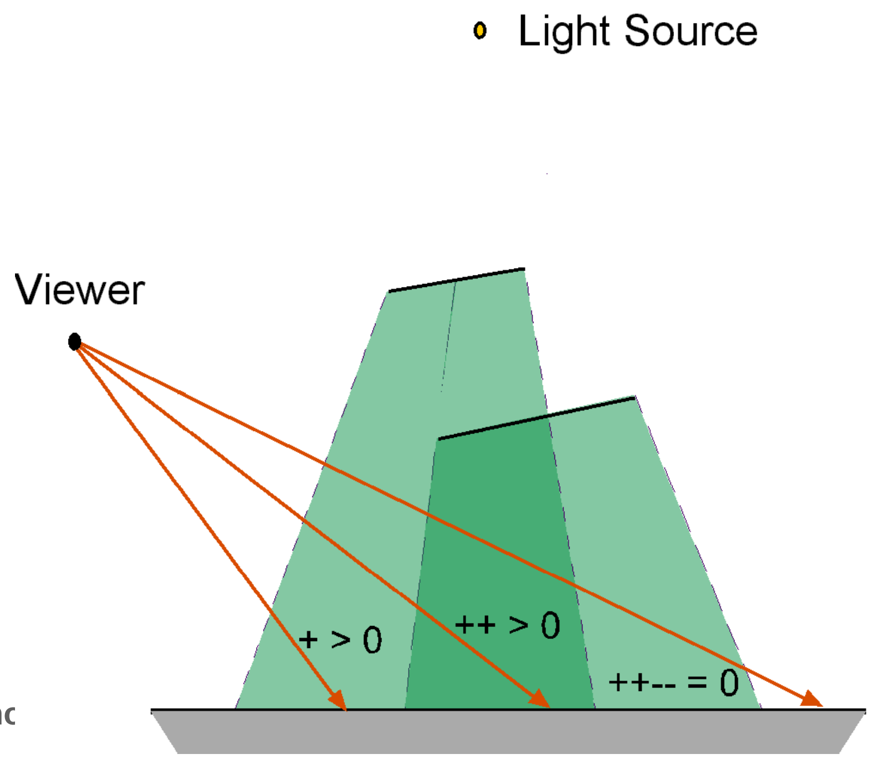
# Enter/Leave Approach

- Increment on enter, decrement on leave
- Simultaneously test all visible pixels
  → Stop when hitting object nearest to viewer

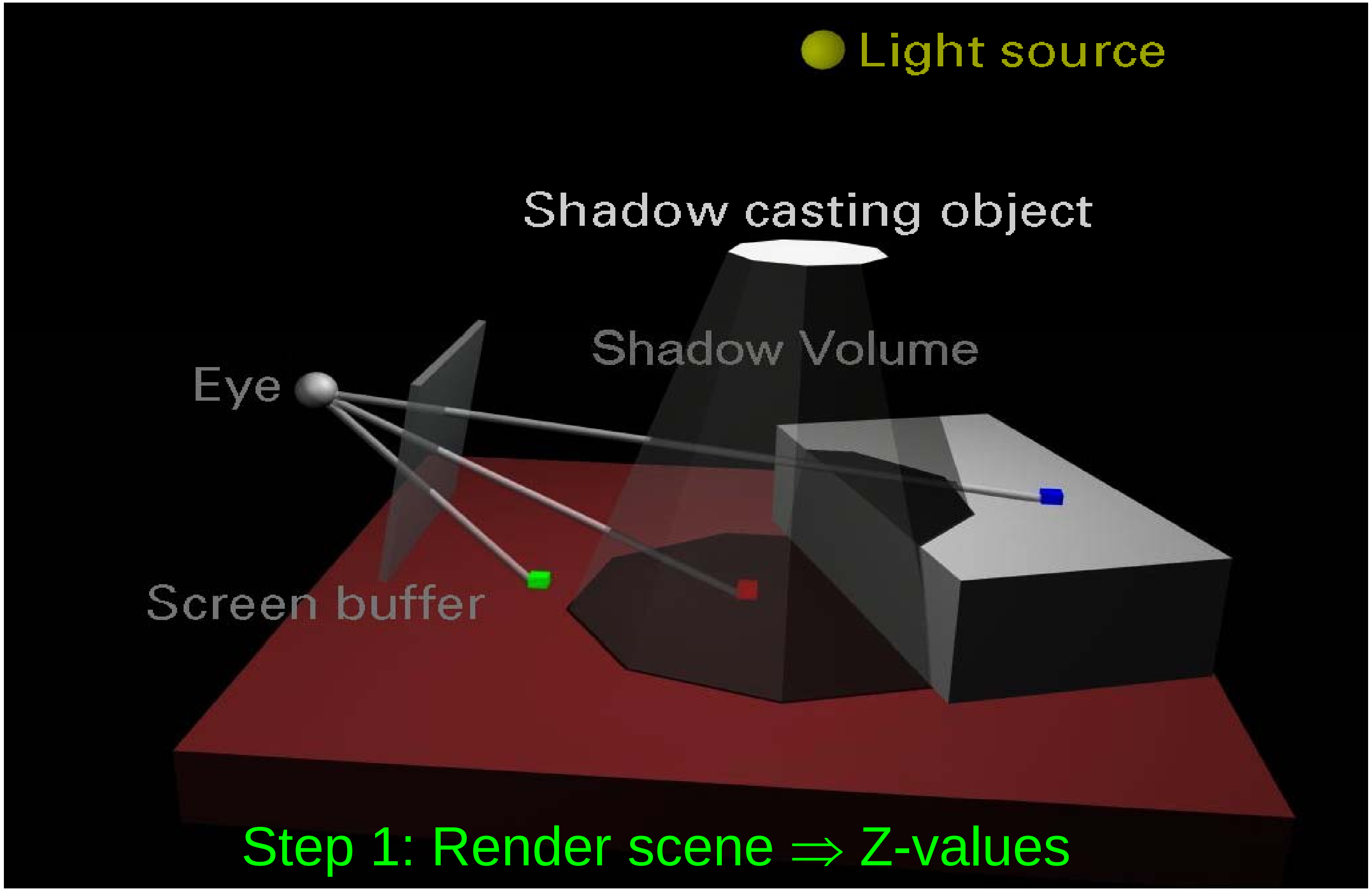


# Shadow Volume Algorithm

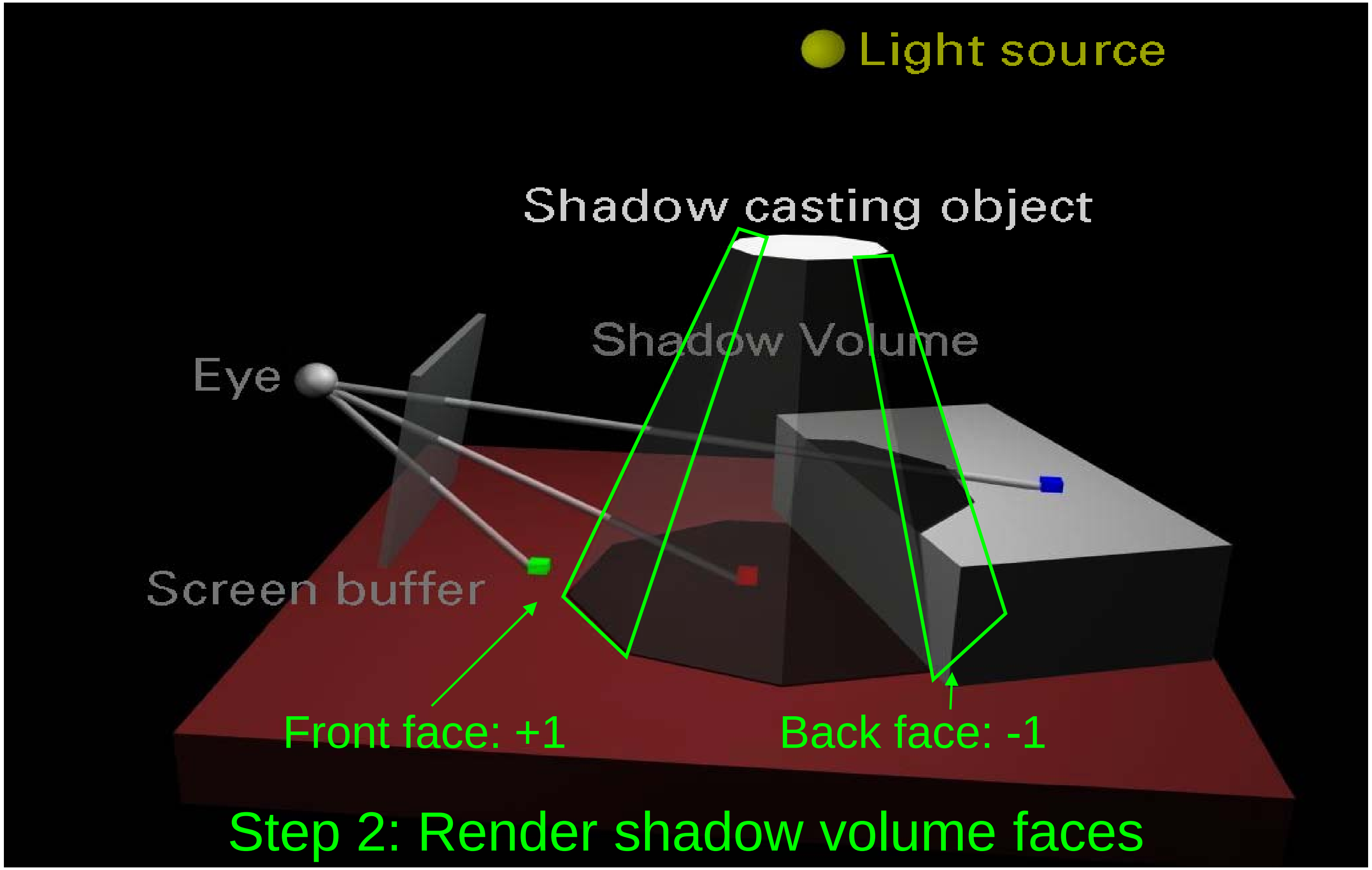
- Shadow volumes in object precision
  - Calculated by CPU/Vertex Shaders
- Shadow test in image precision
  - Using stencil buffer as counter!



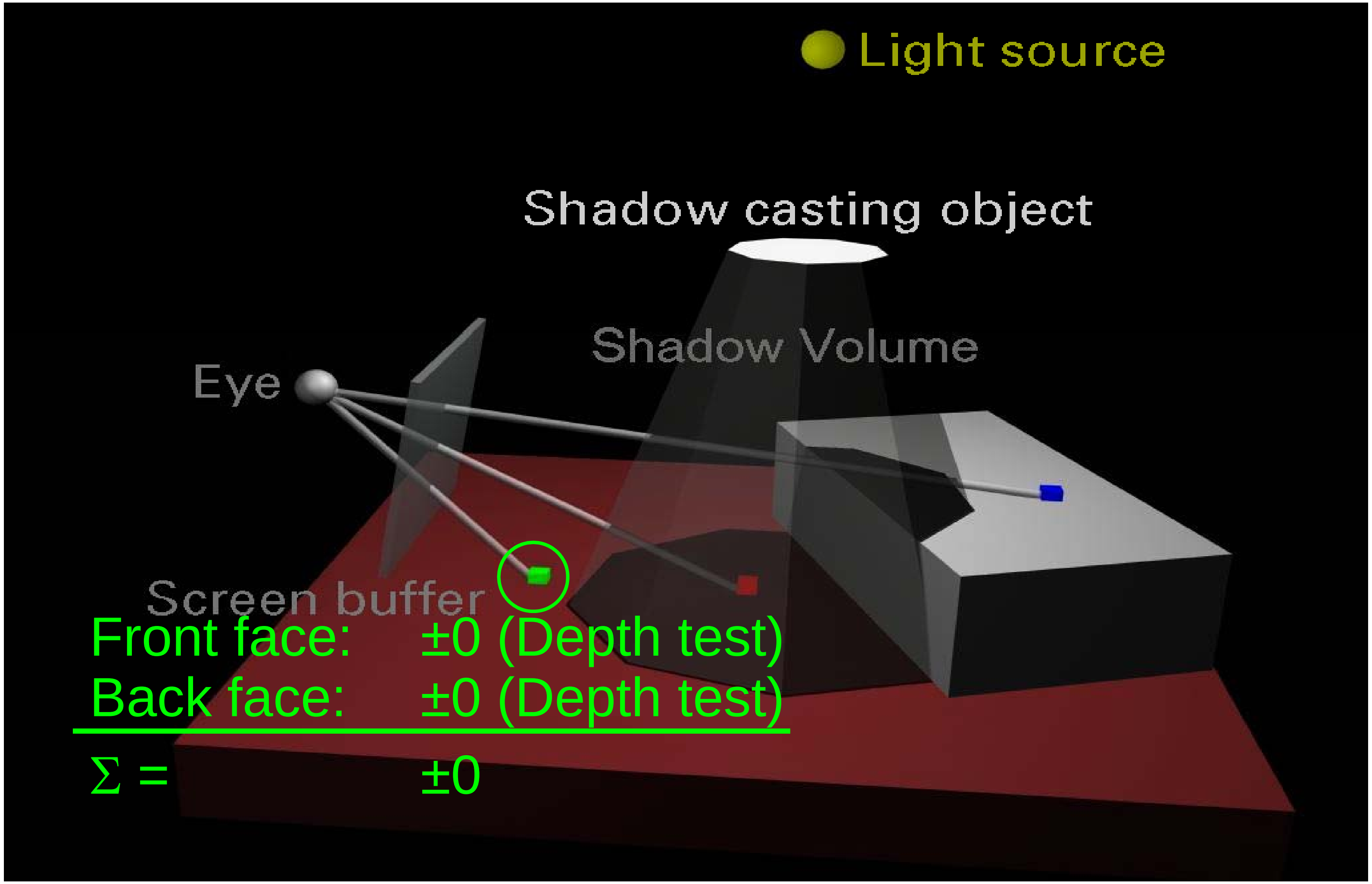
# Shadow Volume Algorithm



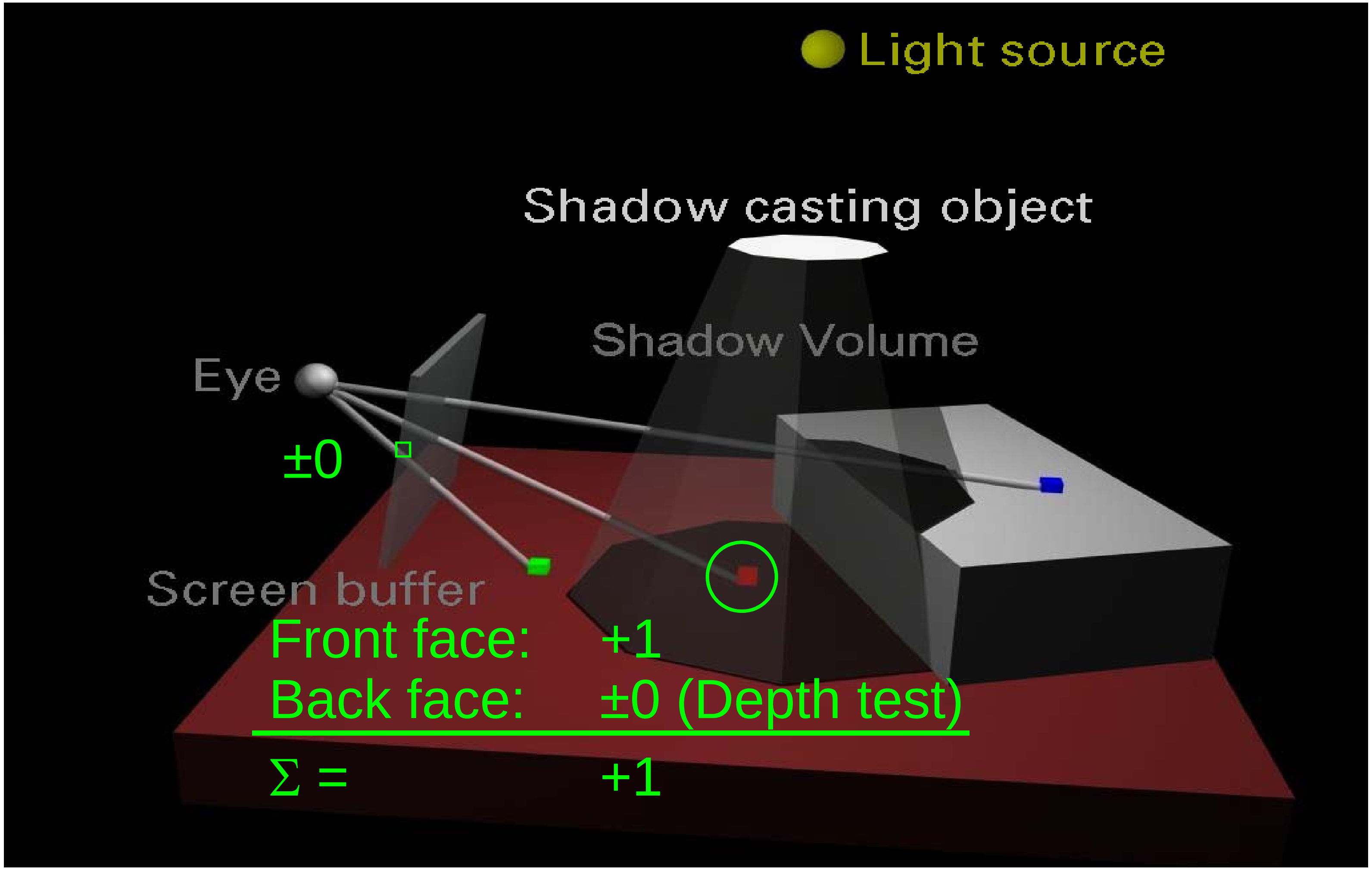
# Shadow Volume Algorithm



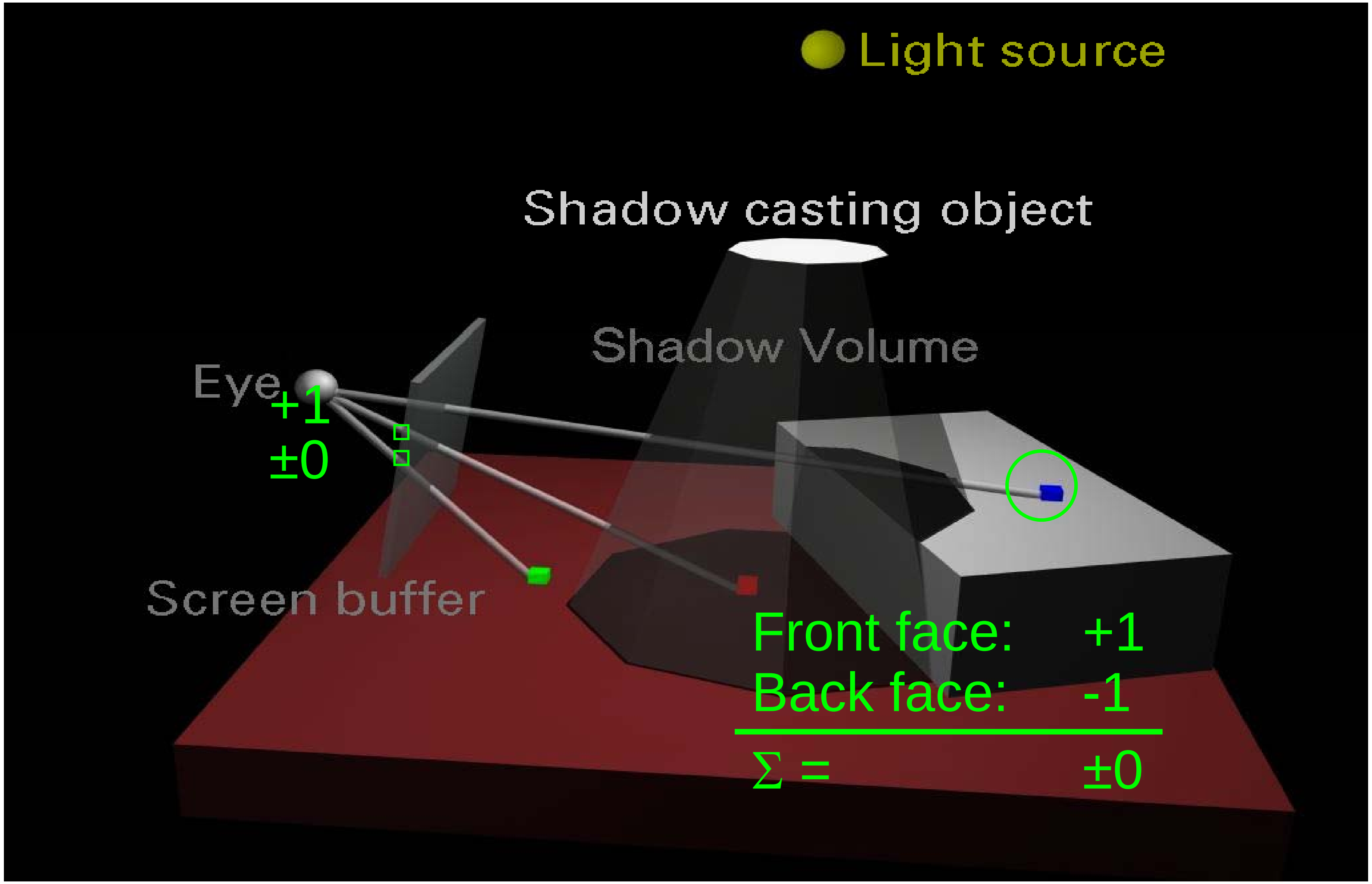
# Shadow Volume Algorithm



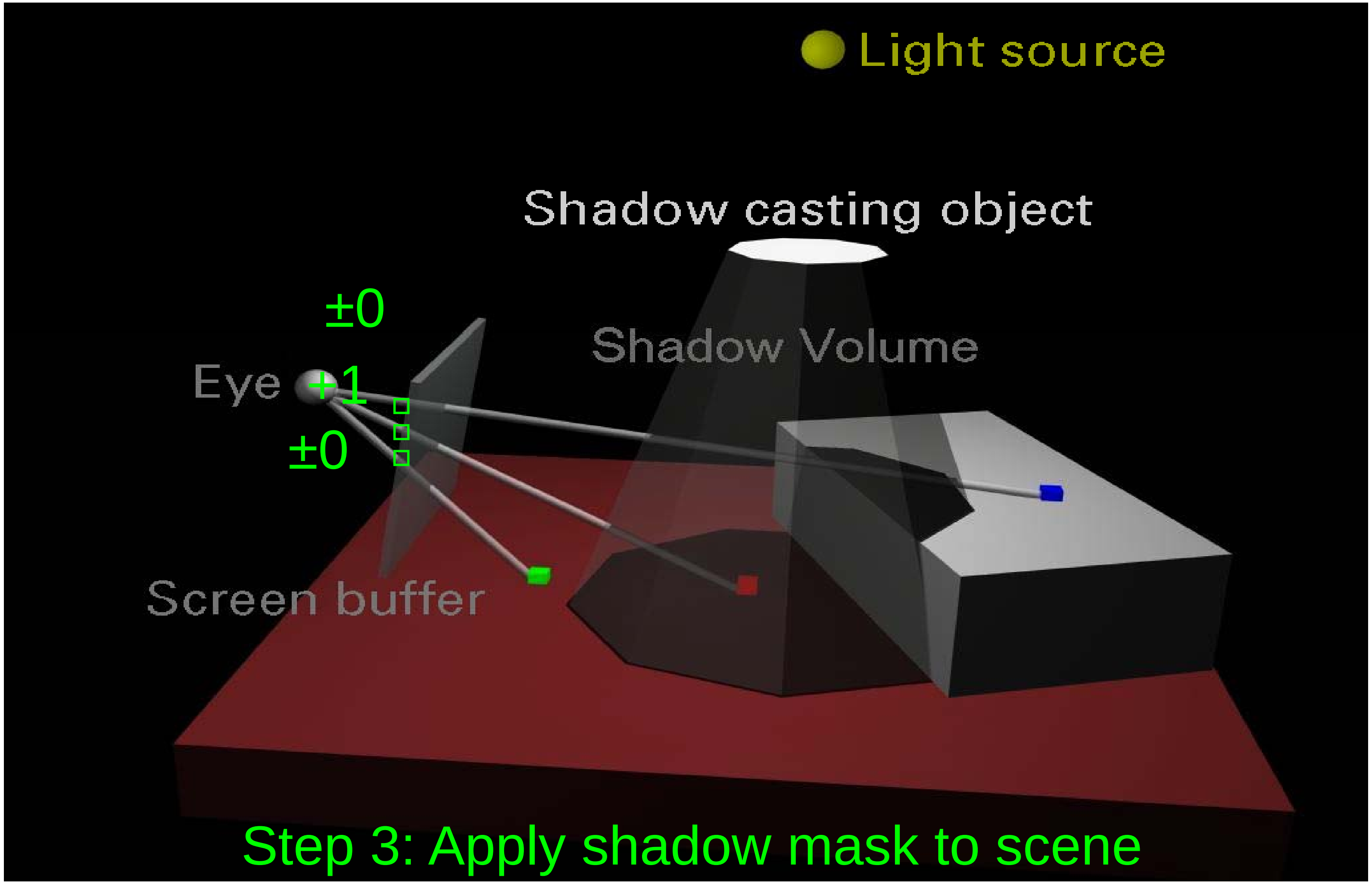
# Shadow Volume Algorithm



# Shadow Volume Algorithm



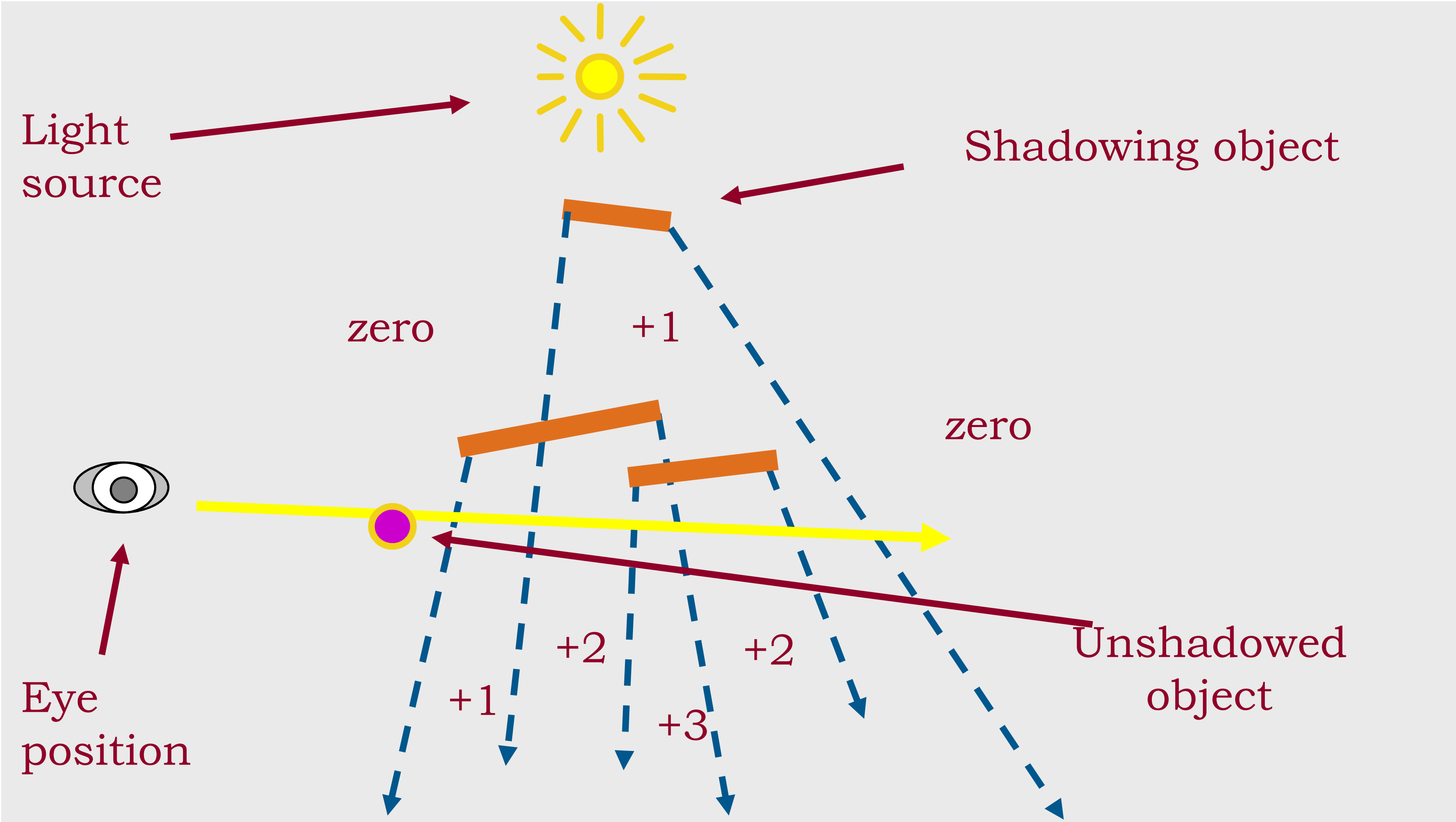
# Shadow Volume Algorithm



# Shadow Volume Algorithm (Zpass)

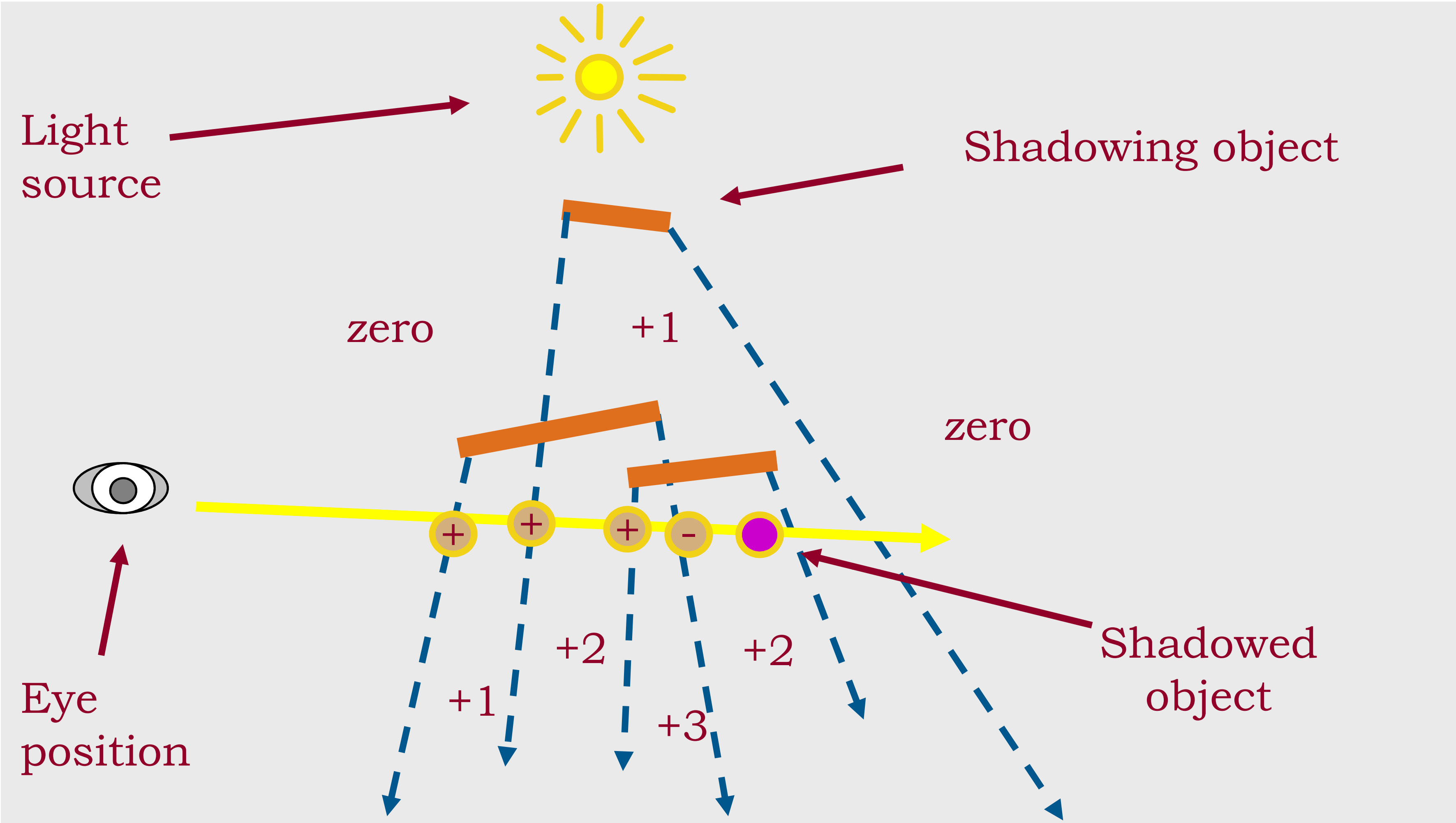
- Render scene to establish z-buffer
  - Can also do ambient illumination
- For each light
  - Clear stencil
  - Draw shadow volume twice using culling
    - Render front faces and increment stencil
    - Render back faces and decrement stencil
  - Illuminate all pixels not in shadow volume
    - Render testing stencil = 0
    - Use additive blend

# Zpass Technique (Before Shadow)



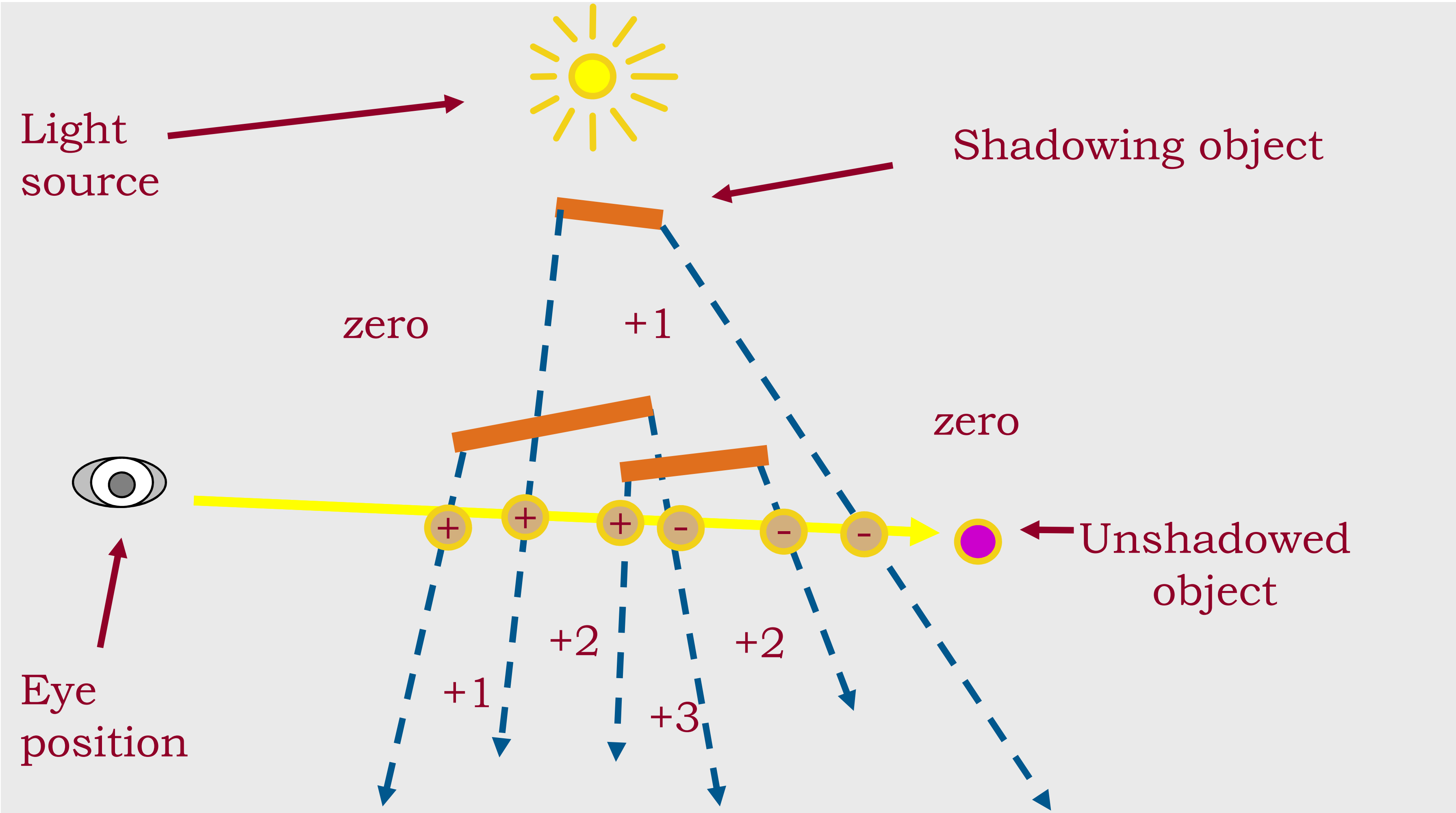

**Shadow Volume Count = 0 (no depth tests passes)**

# Zpass Technique (In Shadow)



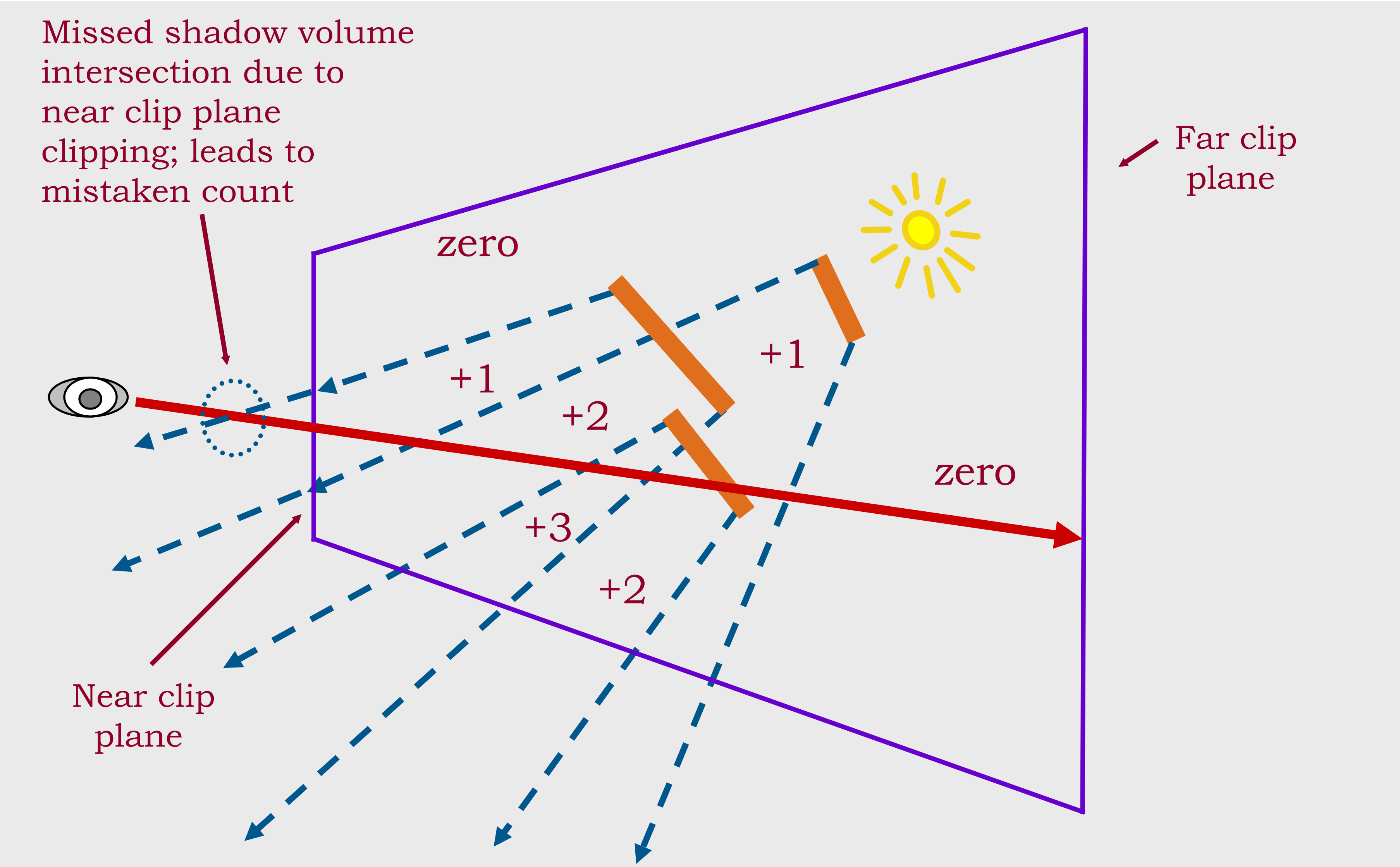

**Shadow Volume Count = +1+1+1-1 = 2**

# Zpass Technique (Behind Shadow)




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

# Zpass Near Plane Problem

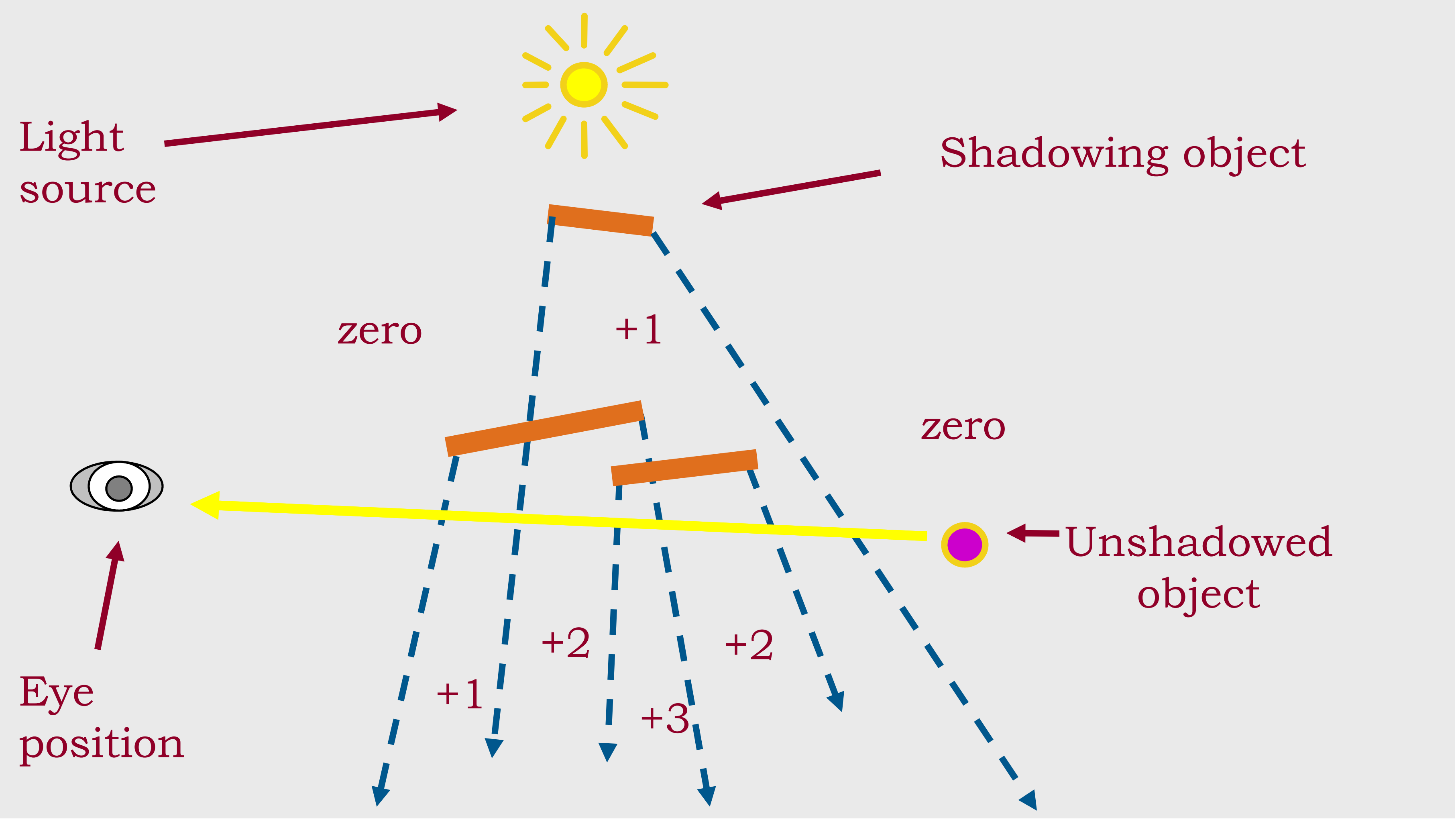


# Alternative: Zfail Technique

- Zpass near plane problem difficult to solve
  - Have to “cap” shadow volume at near plane
  - Expensive and not robust, many special cases
- Try reversing test order → Zfail technique (also known as Carmack’s reverse)
  - Start from infinity and stop at nearest intersection

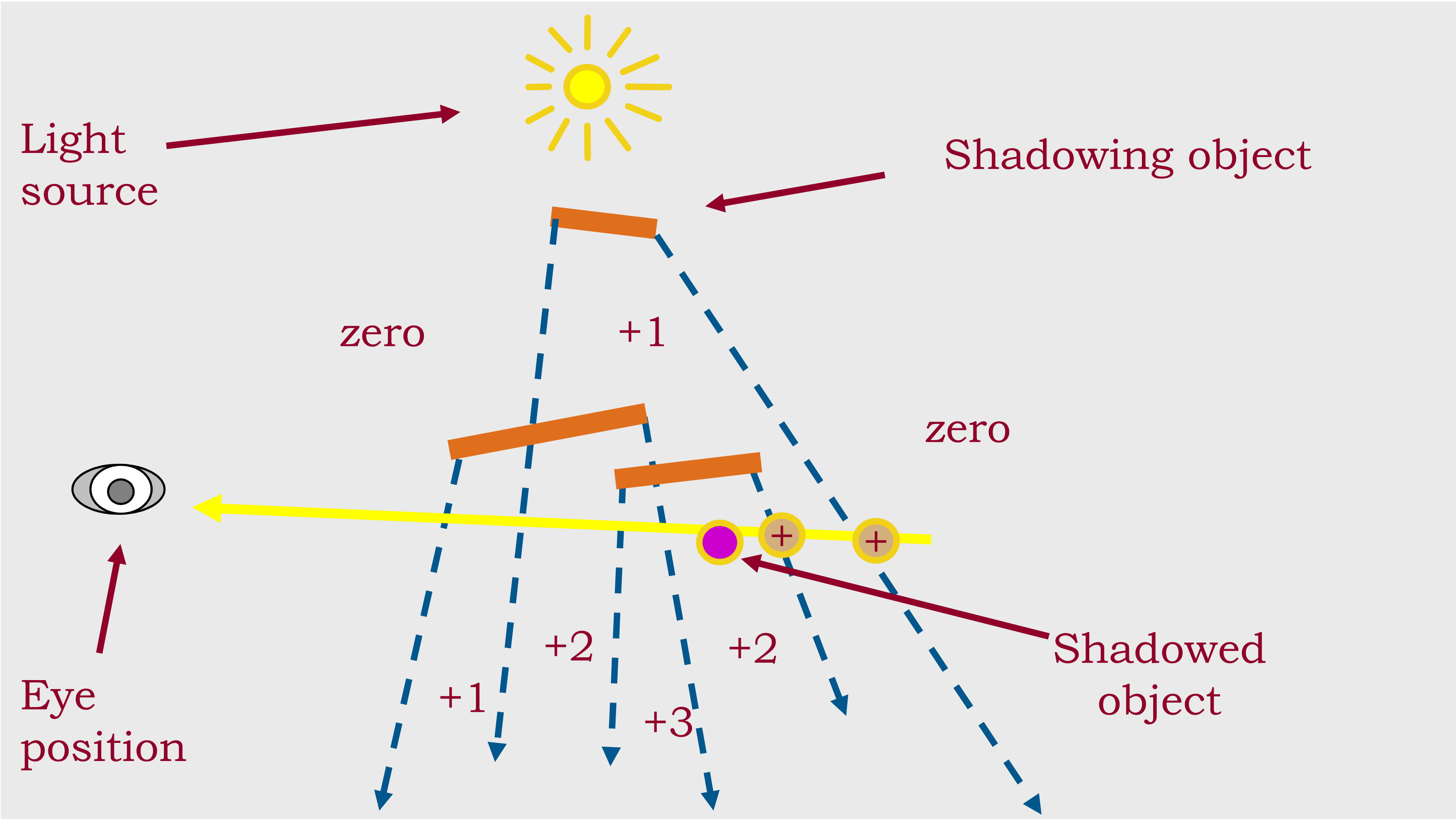
    → Render shadow volume fragments only when depth test fails
  - Render back faces first and increment
  - Then front faces and decrement
  - Need to cap shadow volume at infinity or light extent

# Zfail, Behind Shadow



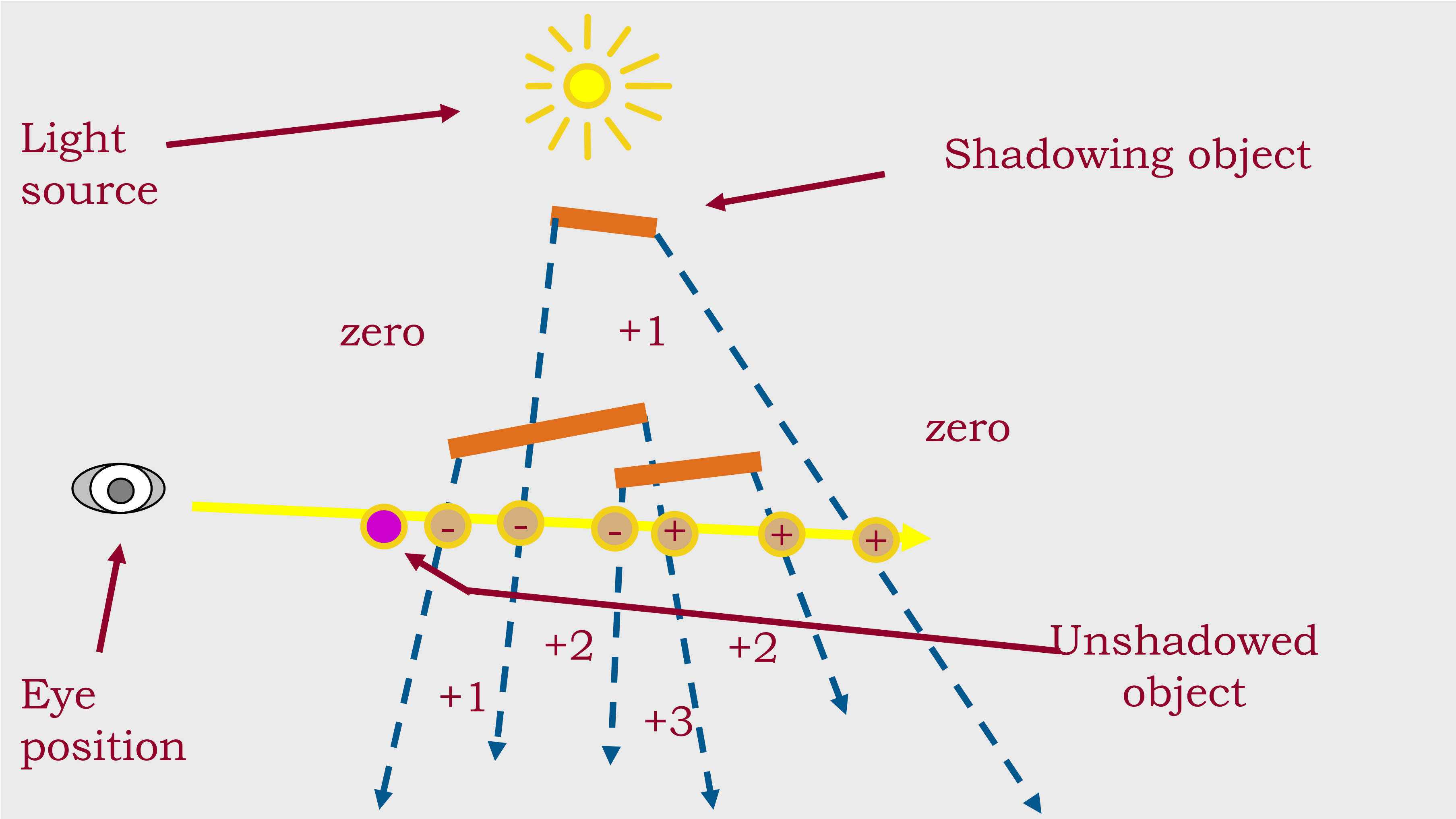

**Shadow Volume Count = 0 (zero depth tests fail)**

# Zfail, in Shadow




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

# Zfail, before Shadow




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

# Shadow Volumes

- Shadow volume = closed polyhedron
- Actually 3 sets of polygons!
  1. Object polygons facing the light (“light cap”)
  2. Object polygons facing away from the light and projected to infinity (with w = 0) (“dark cap”)
  3. Actual shadow volume polygons (extruded object edges) (“sides”)
     → but which edges?

# Zpass vs. Zfail

- Equivalent, but reversed
- Zpass
  - Faster (light cap and dark cap not needed)
    - Light cap inside object → always fails z-test
    - Dark cap infinitely far away → either fails or falls on background
  - Problem at near clip plane (no robust solution)
- Zfail
  - Slower (need to render dark and light caps!)
  - Problem at far clip plane when light extends farther than far clip plane
    - Robust solution with infinite shadow volumes!

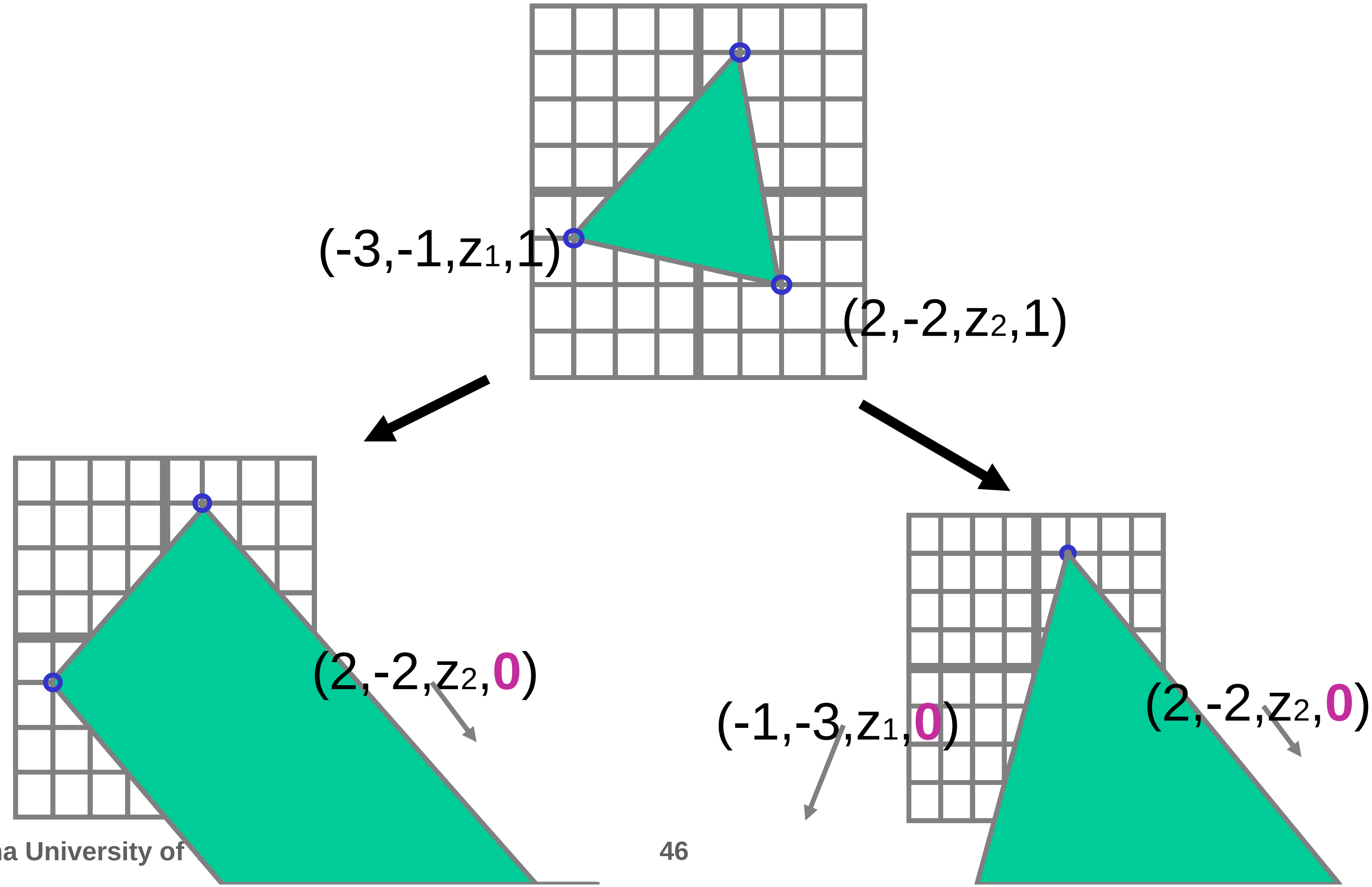
# Zpass vs. Zfail

- Idea: Combine techniques!
  - Test whether viewport in shadow → Zfail
  - Otherwise → Zpass
- Idea: avoid far plane clipping in Zfail!
  - Send far plane to infinity in projection matrix
    - Easy, but loses some depth buffer precision
  - Draw infinite vertices using homogeneous coordinates: project to infinity → w = 0

→ robust solution!

# W=0 Rasterization

- At infinity, vertices become vectors



# Computing Actual SV Polygons

- Trivial but bad: one volume per triangle
  - 3 shadow volume polygons per triangle
- Better: find exact silhouette
  - Expensive on CPU
- Even better: possible silhouette edges
  - Edge shared by a back-facing and front-facing polygon (with respect to light source!), extended to infinity
  - Actual extrusion can be done by vertex shader

# Possible Silhouette Edges

# Shadow Volumes Summary

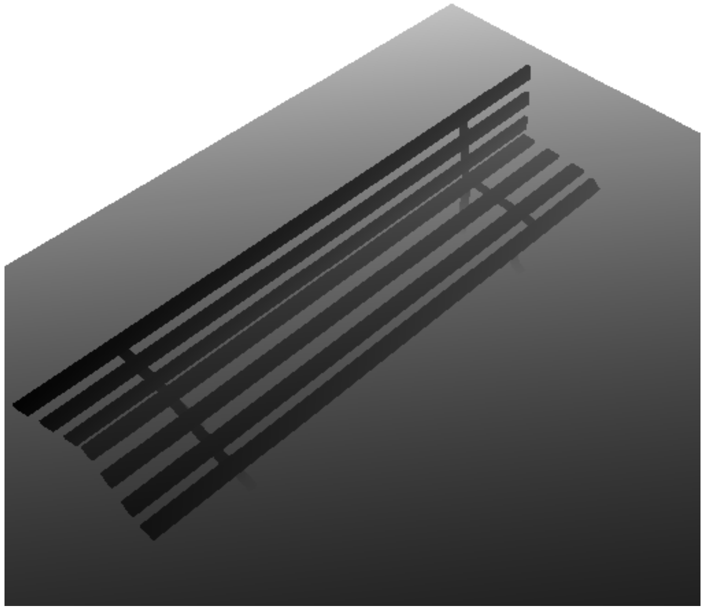
- Advantages
  - Arbitrary receivers
  - Fully dynamic
  - Omnidirectional lights (unlike shadow maps!)
  - Exact shadow boundaries (pixel-accurate)
  - Automatic self shadowing
  - Broad hardware support (stencil)
- Disadvantages
  - Fill-rate intensive
  - Difficult to get right (Zfail vs. Zpass)
  - Silhouette computation required
  - Doesn’t work for arbitrary casters (smoke, fog…)

# Shadow Volume Issues

- Stencil buffering fast and present in all cards
- With 8 bits of stencil, maximum shadow depth is 255
  - `EXT_stencil_wrap` overcomes this
- Two-sided stencil tests can test front- and back triangles simultaneously
  - Saves one pass – available on NV30+
- NV_depth_clamp (hardware capping)
  - Regain depth precision with normal projection
- Requires watertight models with connectivity, and watertight rasterization

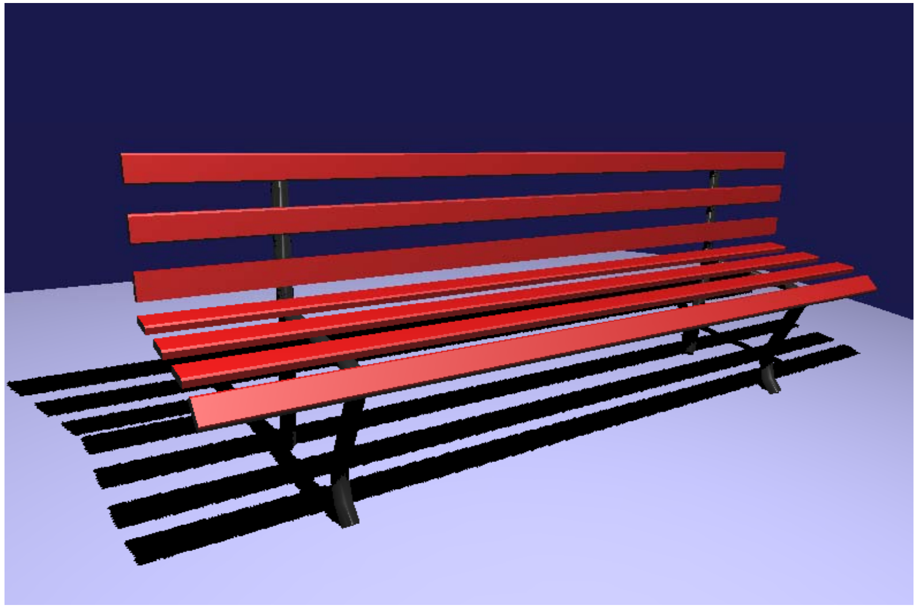
# Shadow Maps

- Casting curved shadows on curved surfaces
  - Image-space algorithm, 2 passes

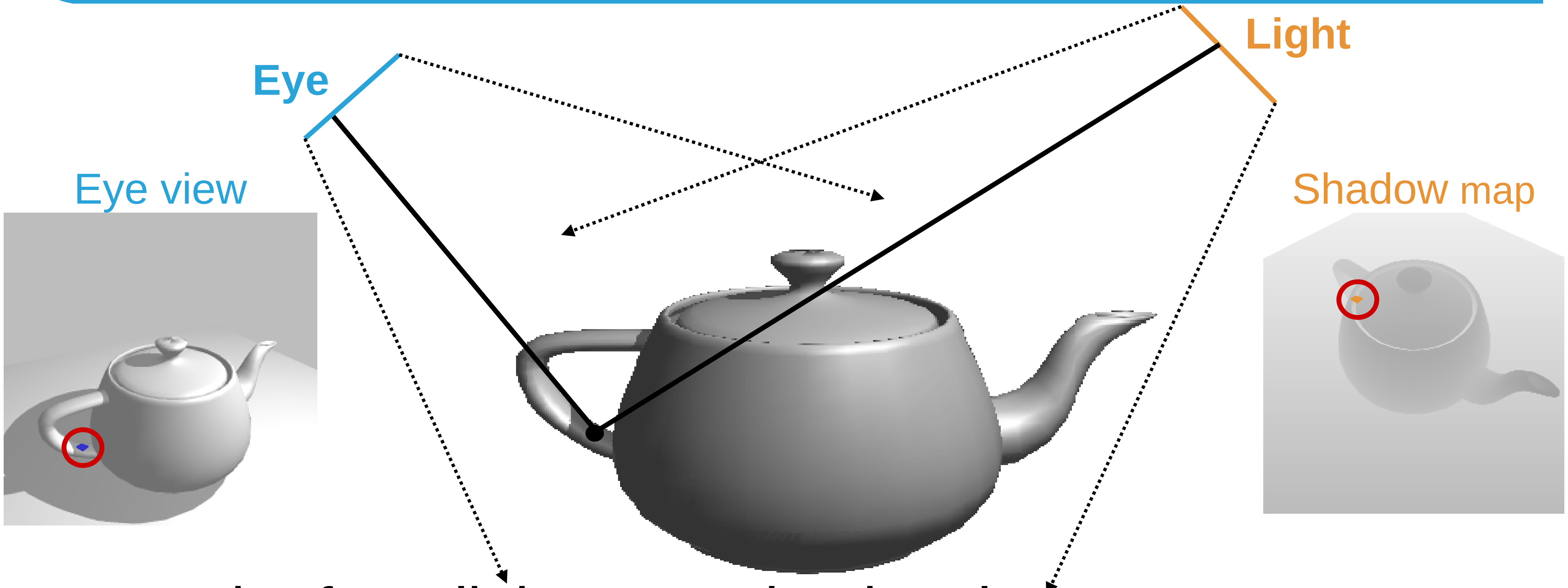


Shadow map

Final scene



# Shadow Map Algorithm




- Render from light; save depth values
- Render from eye
  - Transform all fragments to light space
  - Compare $z_{eye}$ and $z_{light}$ (both in light space!!!)
  - $z_{eye} > z_{Licht}$ $\Rightarrow$ fragment in shadow

# Shadow Maps in Hardware

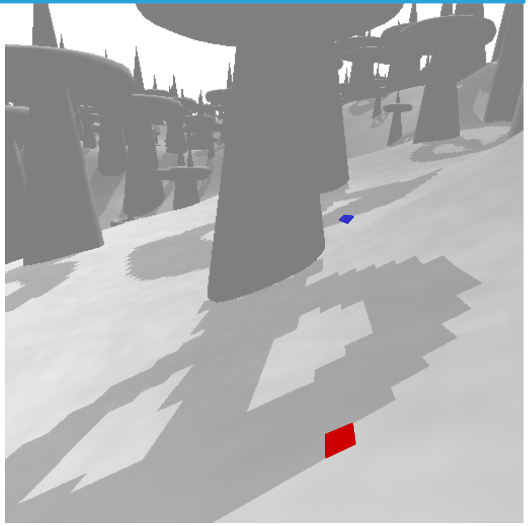
- Render scene to z-buffer (from light source)
  - Copy depth buffer to texture
  - Render to depth texture + pbuffer
- Project shadow map into scene (remember projective texturing!)
- Hardware shadow test (ARB_shadow)
  - Use homogeneous texture coordinates
  - Compare r/q with texel at (s/q, t/q)
  - Output 1 for lit and 0 for shadow
  - Blend fragment color with shadow test result

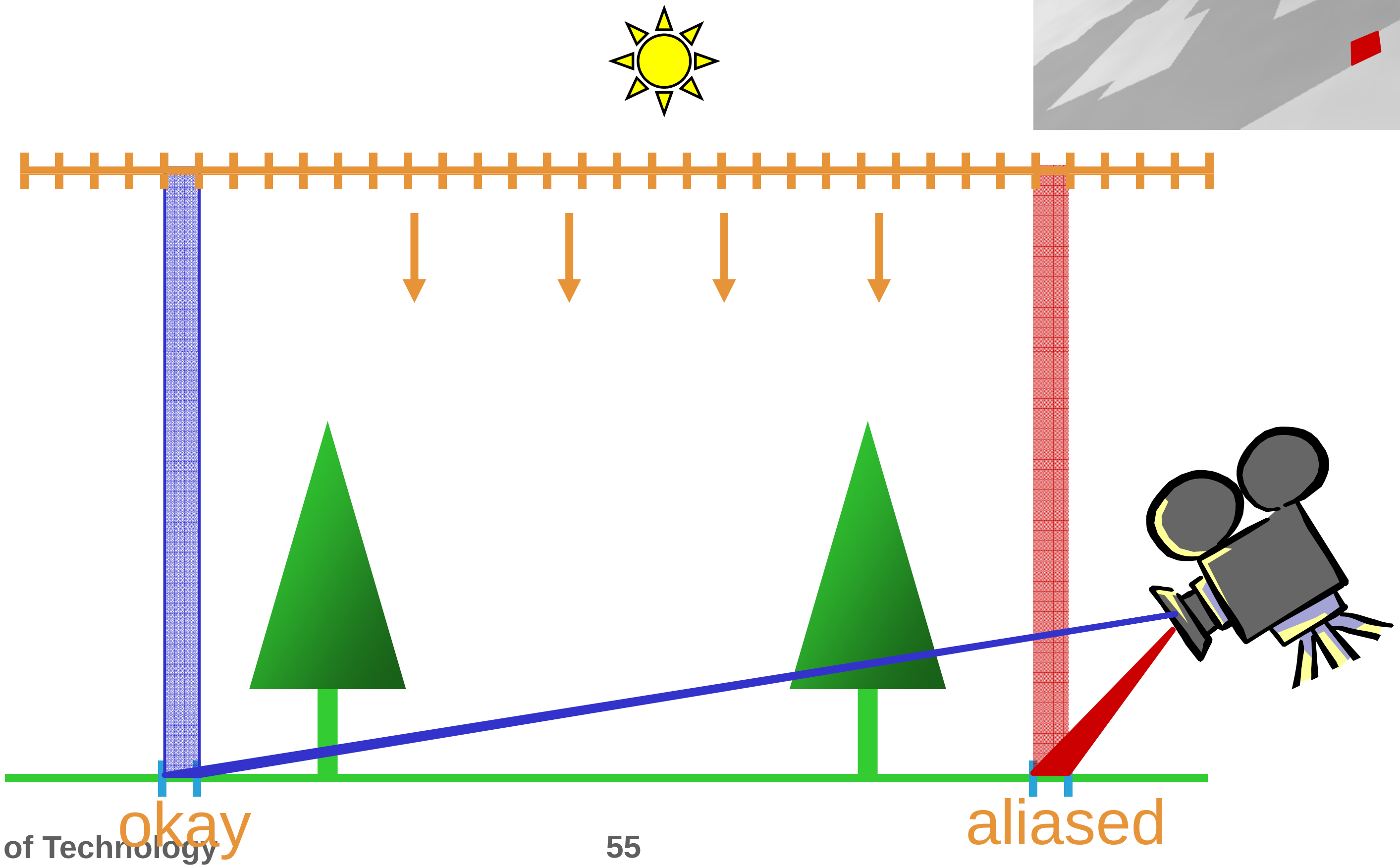
# Shadow Maps in Hardware

- Shadow extension available since GeForce3
  - Requires high precision texture format (`ARB_depth_texture`)
- On modern hardware:
  - Render lightspace depth into texture
  - In vertex shader:
    - Calculate texture coordinates as in projective texturing
  - In fragment shader:
    - Depth compare

# Problem: Perspective Aliasing

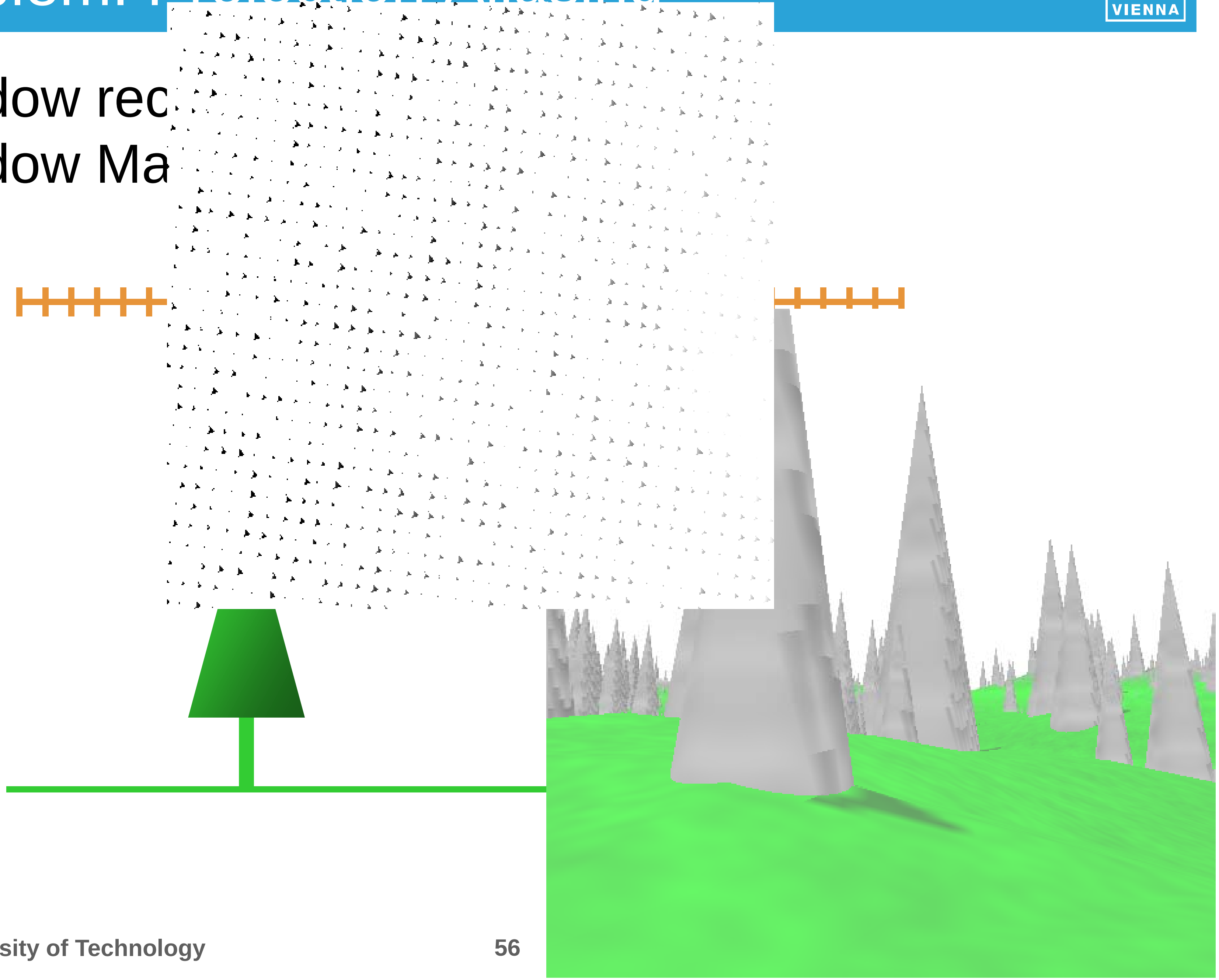
- Sufficient resolution far from eye
- Insufficient resolution near eye



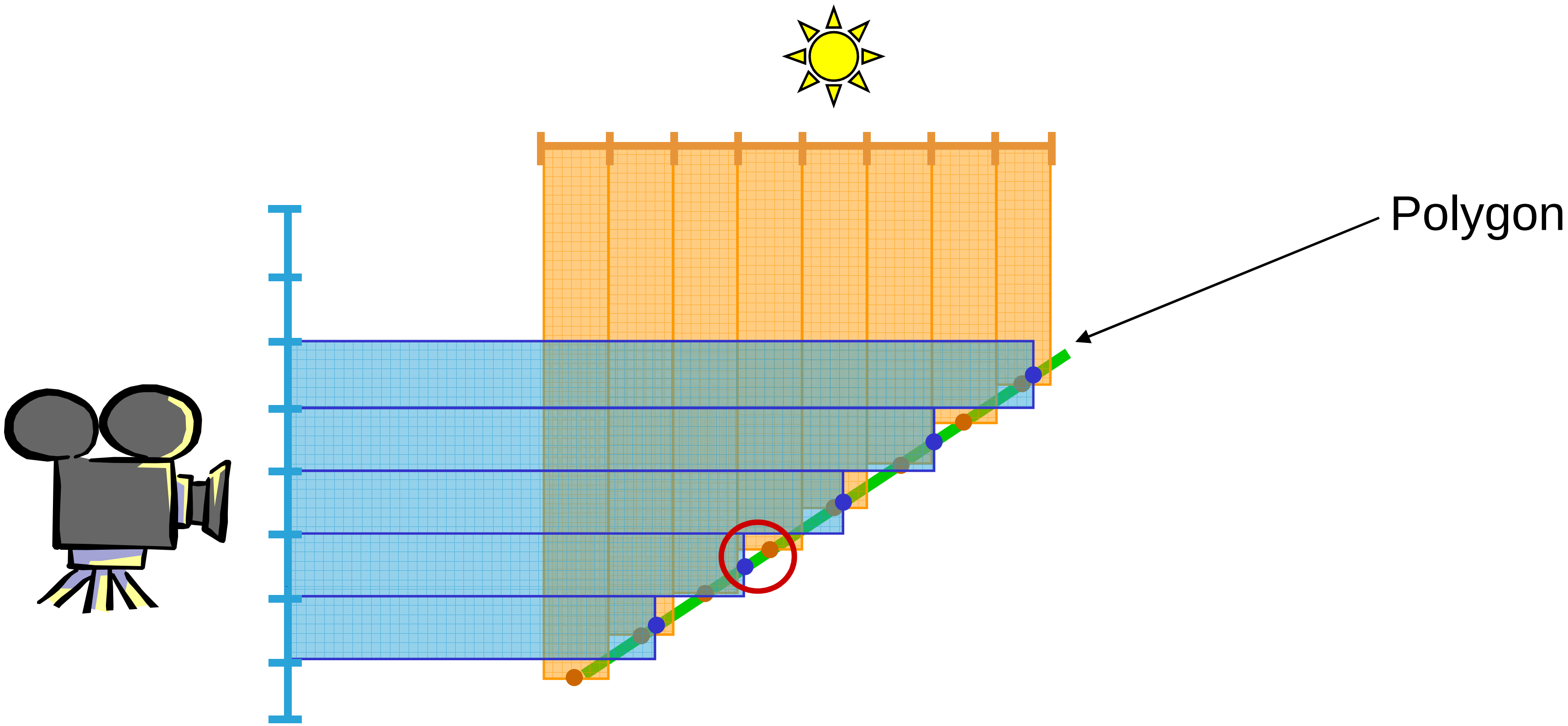


# Problem: Projection Aliasing

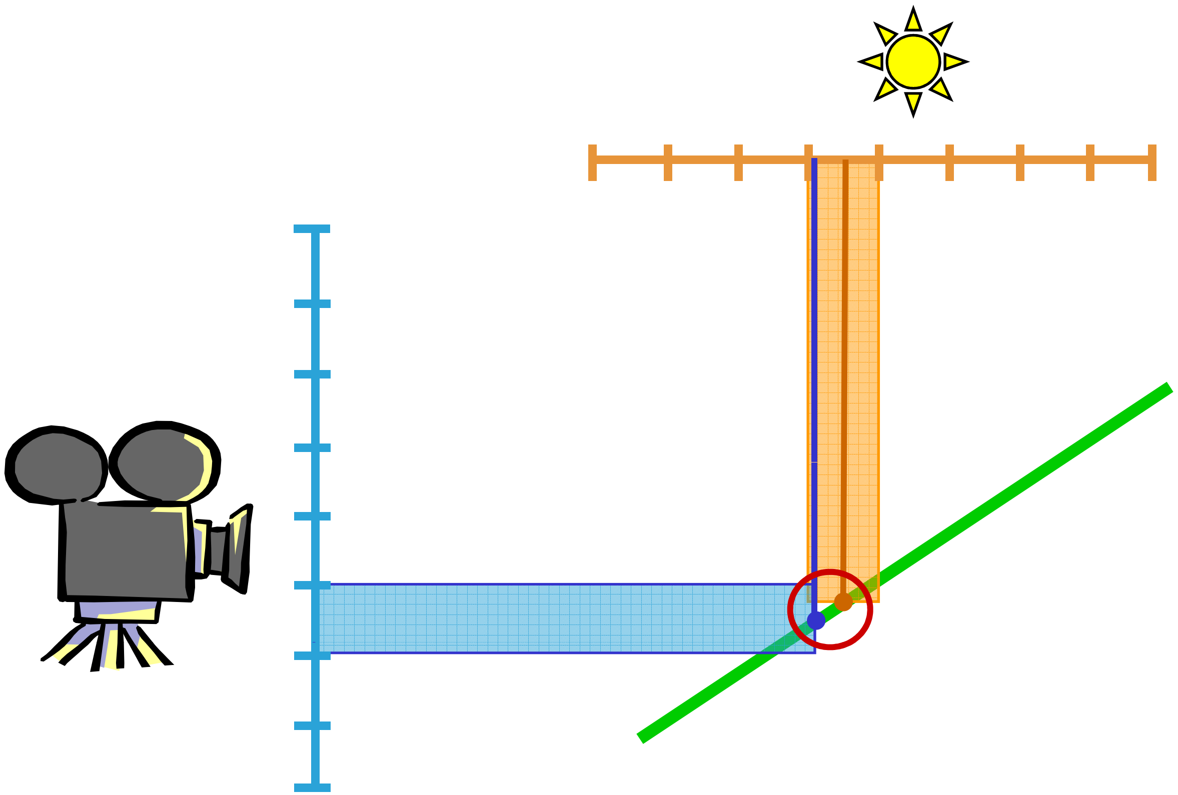
- Shadow rec
  Shadow Ma

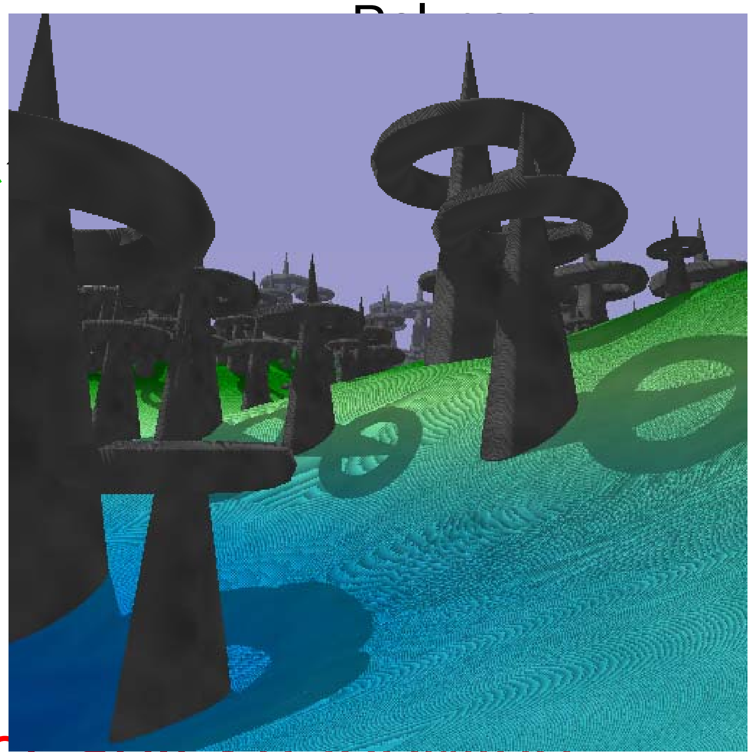


# Problem: Incorrect Self-Shadowing



# Problem: Incorrect Self-Shadowing

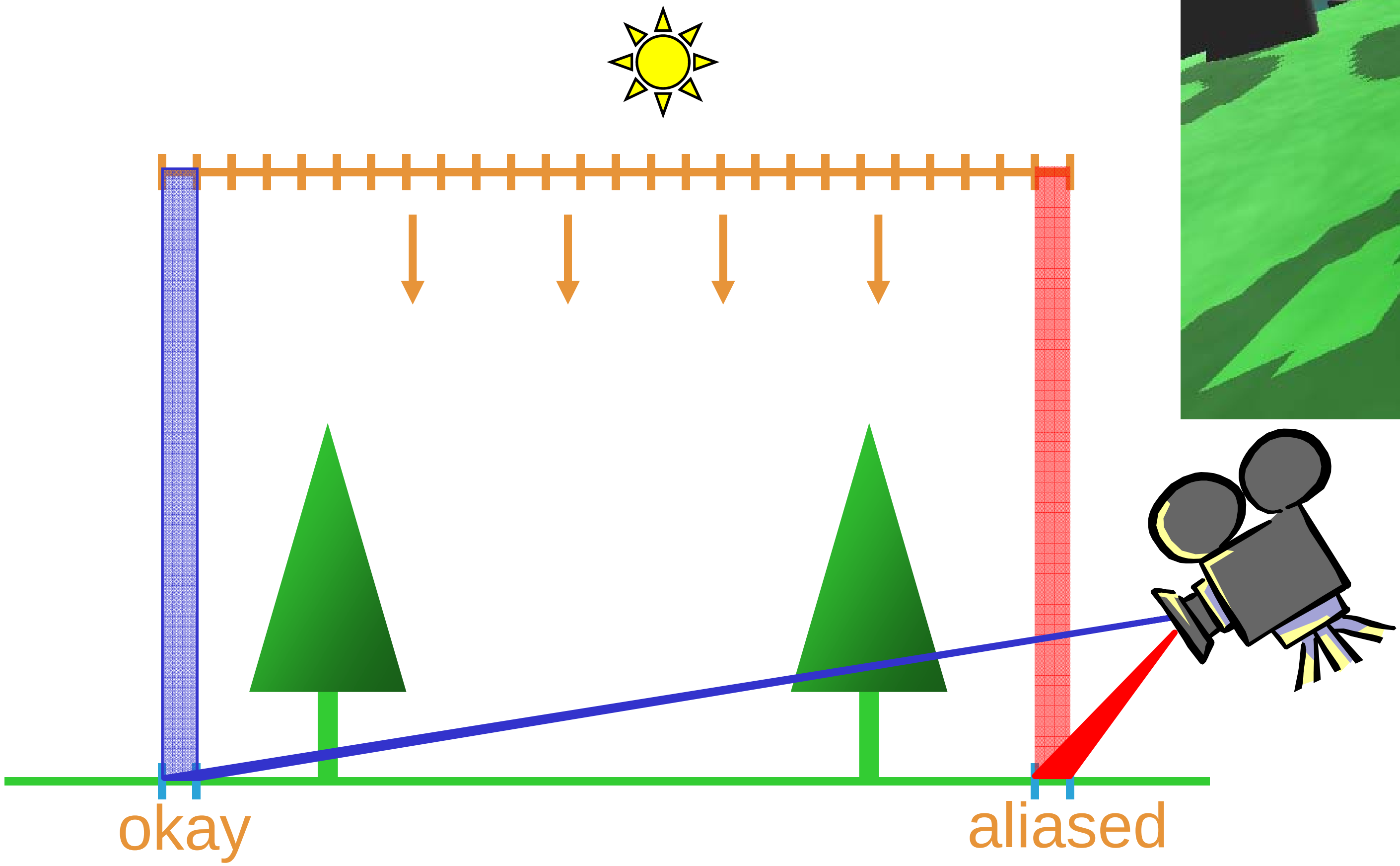


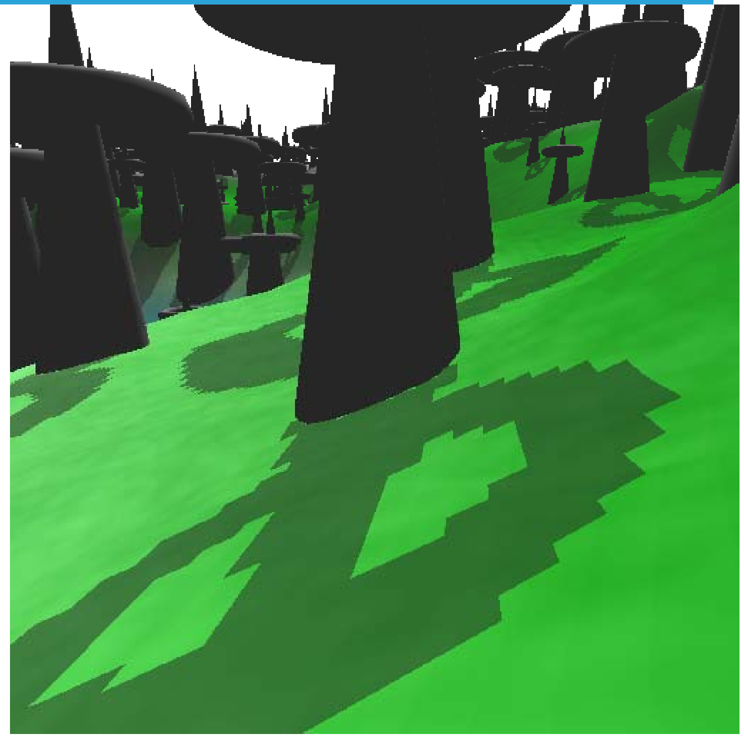


$z_{eye} > z_{light}$ ➡ Incorrect Self-shadowing

# Solution for Perspective Aliasing

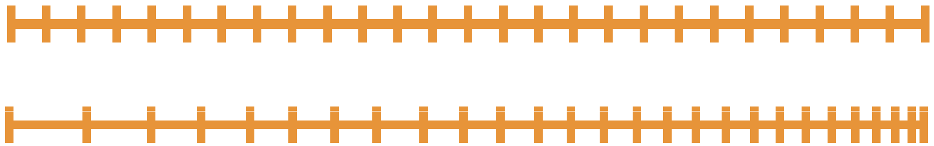
- Insufficient resolution near eye



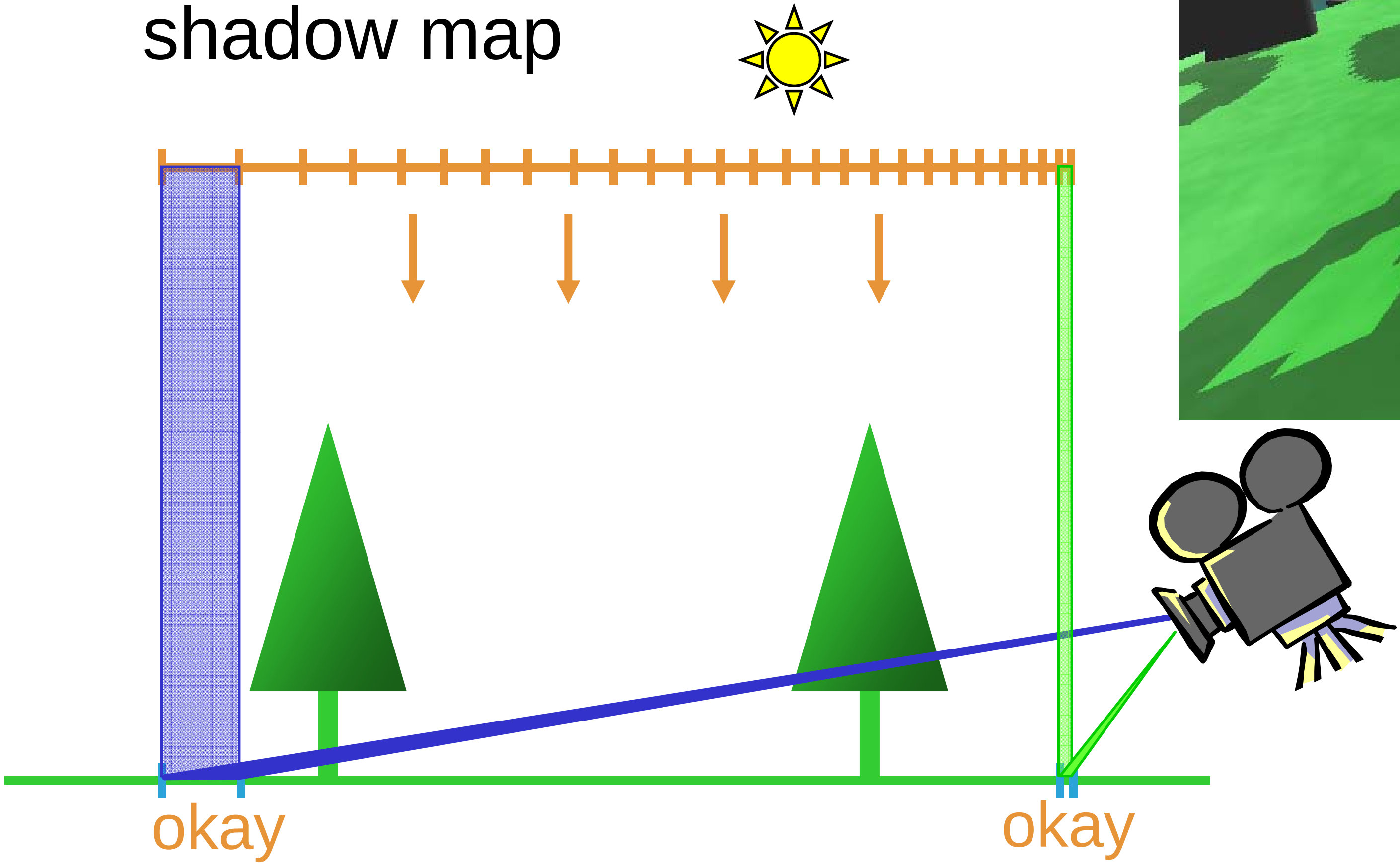
# Solution for Perspective Aliasing

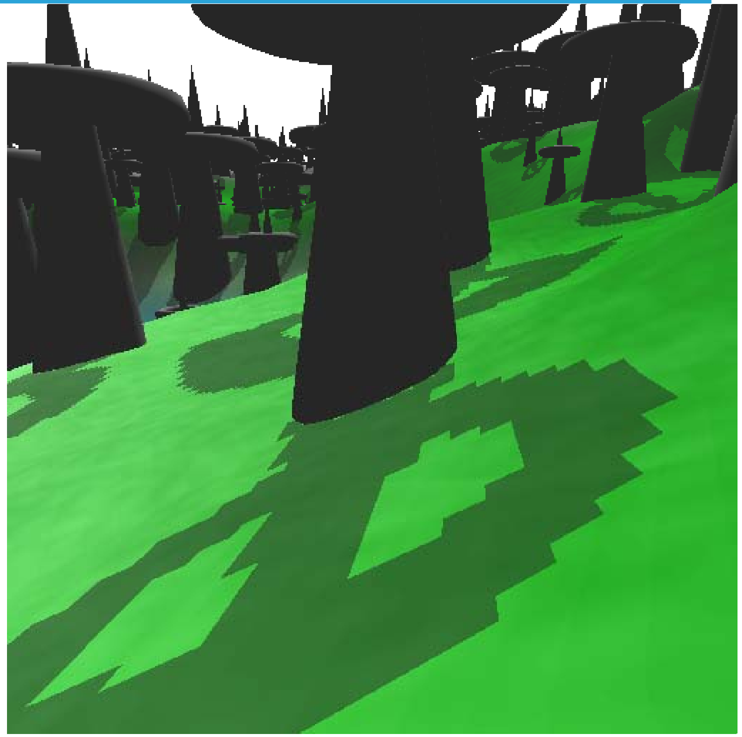
- Insufficient resolution near eye
- **Redistribute** values in shadow map



# Solution for Perspective Aliasing

- **Sufficient** resolution near eye
- **Redistribute** values in shadow map

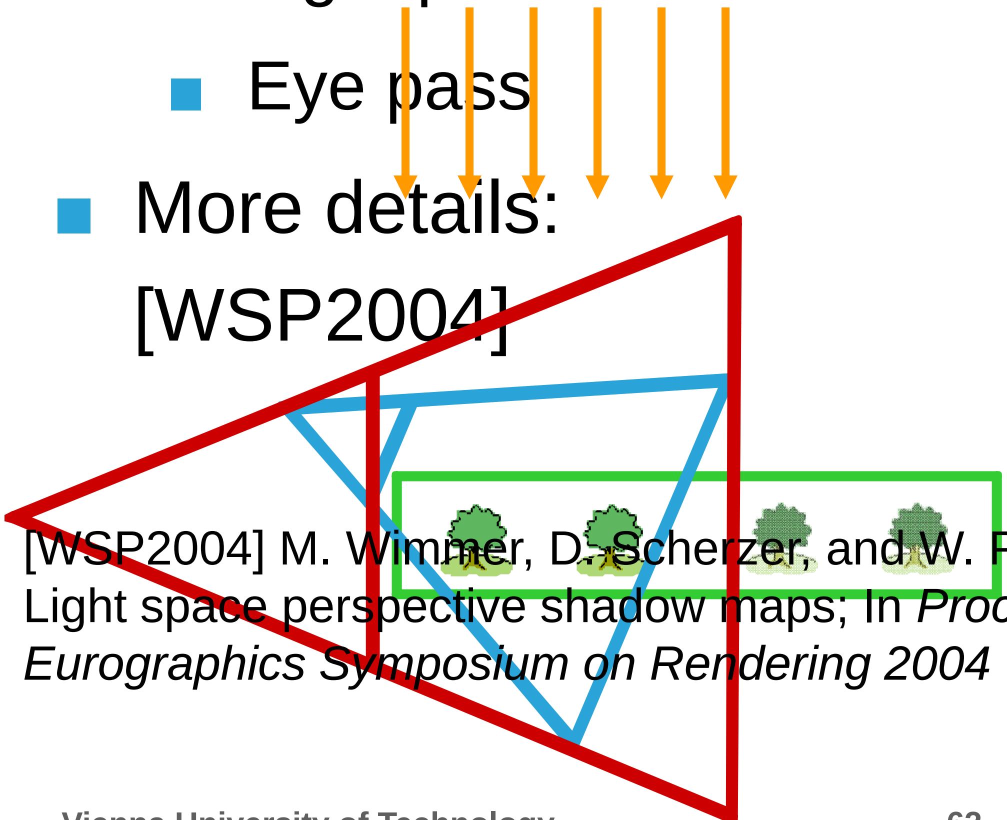


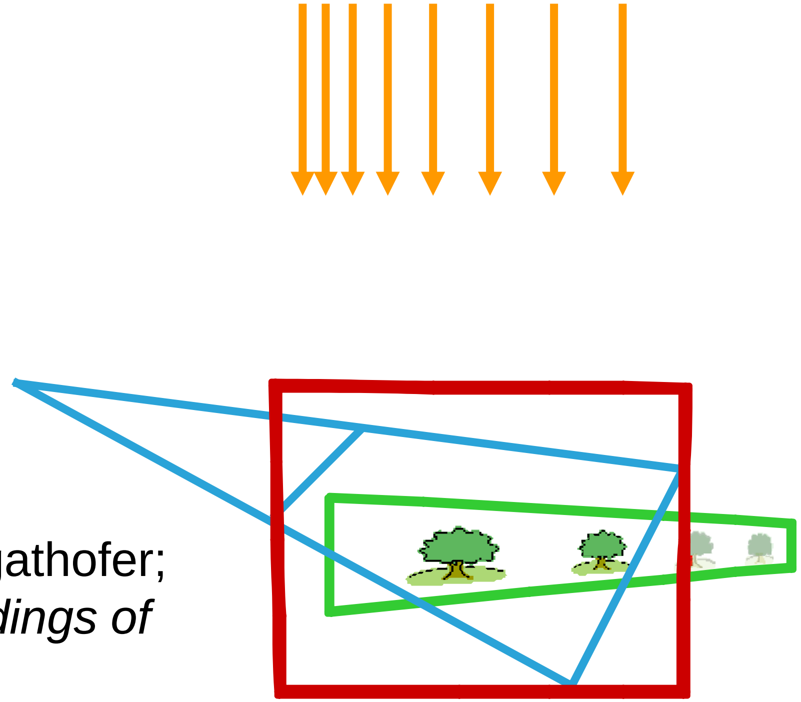



# Solution for Perspective Aliasing

- How to **redistribute**?
- Use **perspective transform**
- Additional perspective matrix, used in both:
  - Light pass
  - Eye pass
- More details:

  [WSP2004]

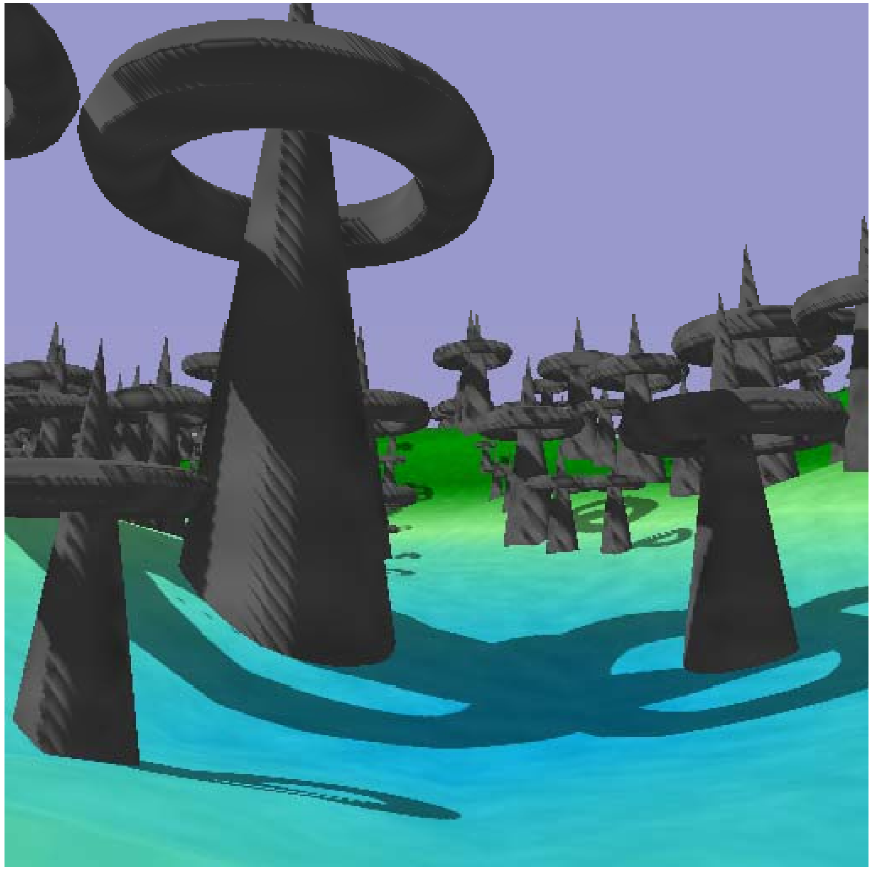


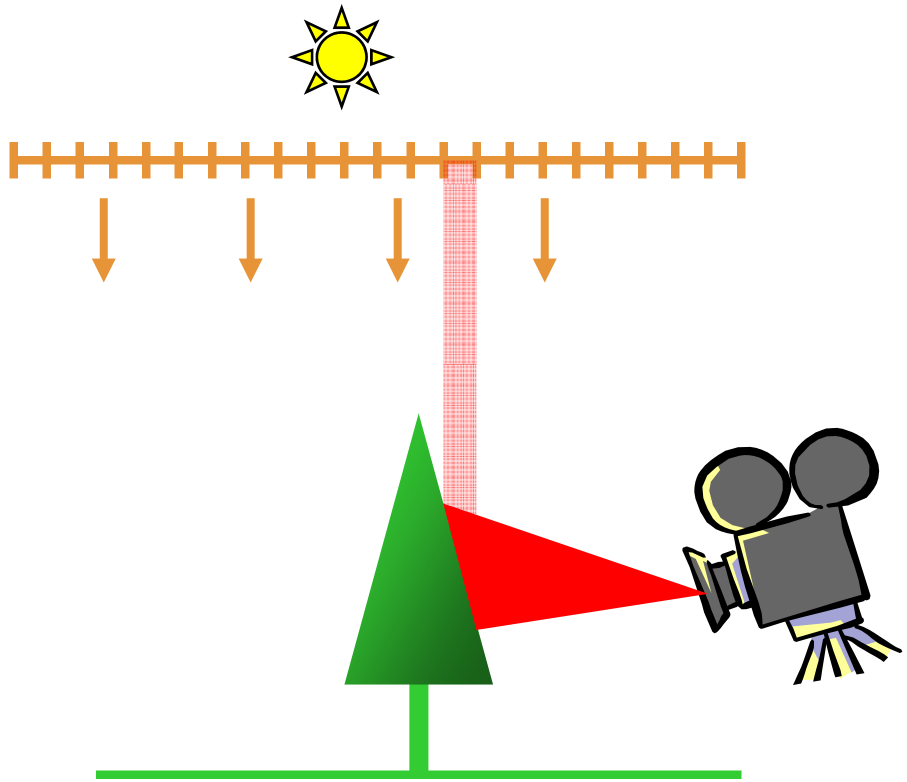


[WSP2004] M. Wimmer, D. Scherzer, and W. Purgathofer; Light space perspective shadow maps; In *Proceedings of Eurographics Symposium on Rendering 2004*

# Solution for Projection Aliasing

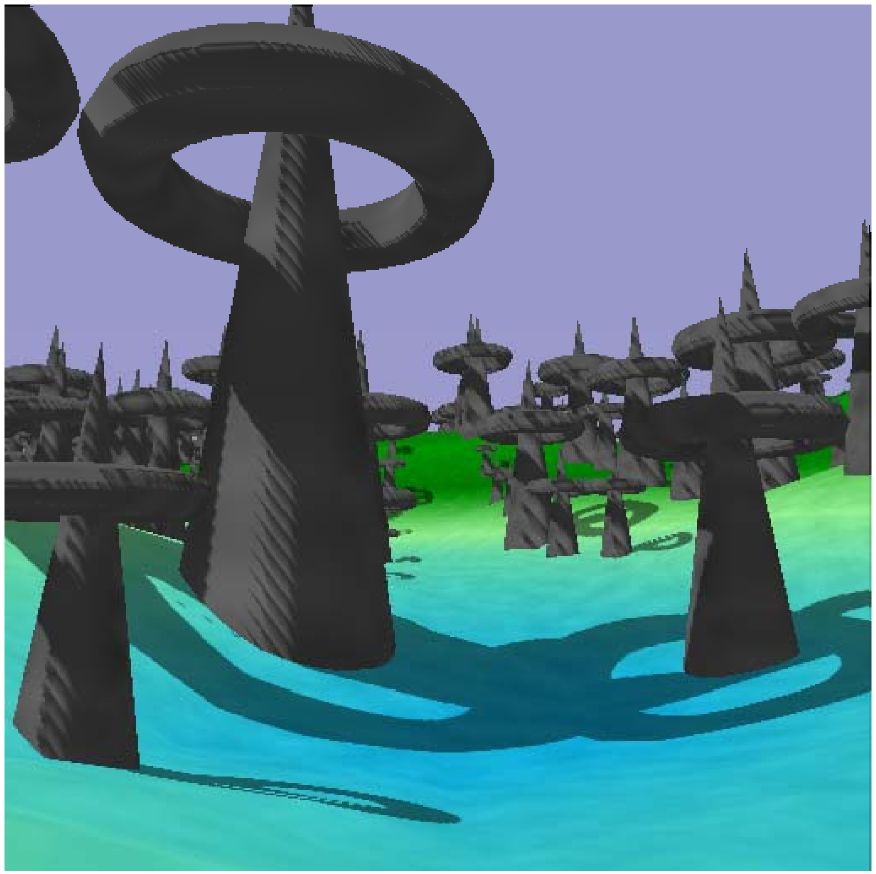
- Shadow receiver ~ **orthogonal** to Shadow Map plane
- Redistribution does not work
- **But...**

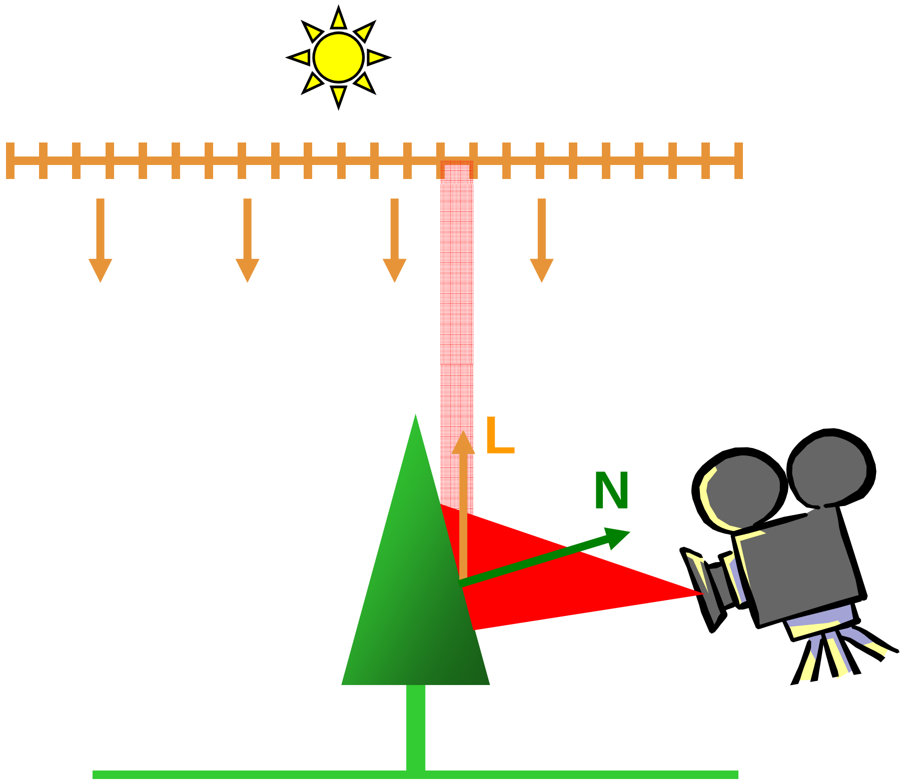




# Solution for Projection Aliasing

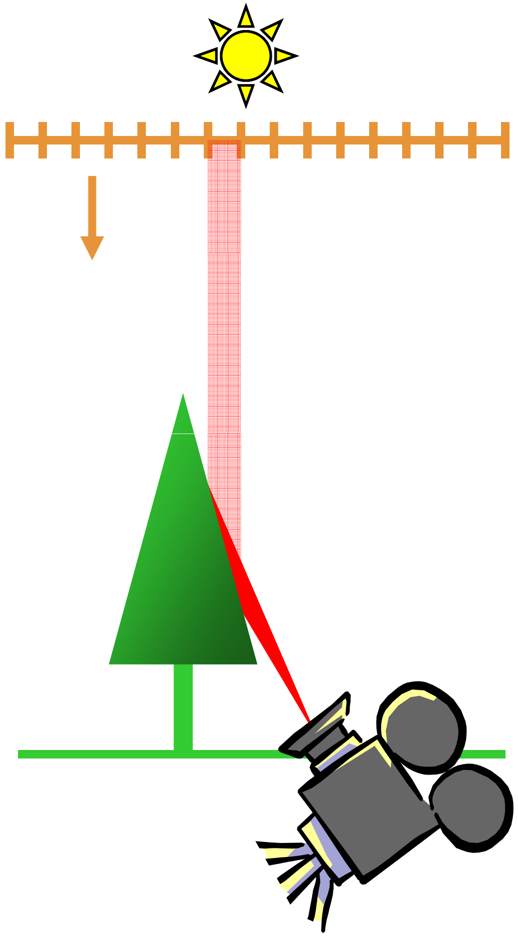
- Diffuse lighting: **I** = $I_L$ *max*( *dot*( **L**, **N** ), 0 )
- Almost orthogonal receivers have small **I**
- Dark ➡ artifacts not very visible!





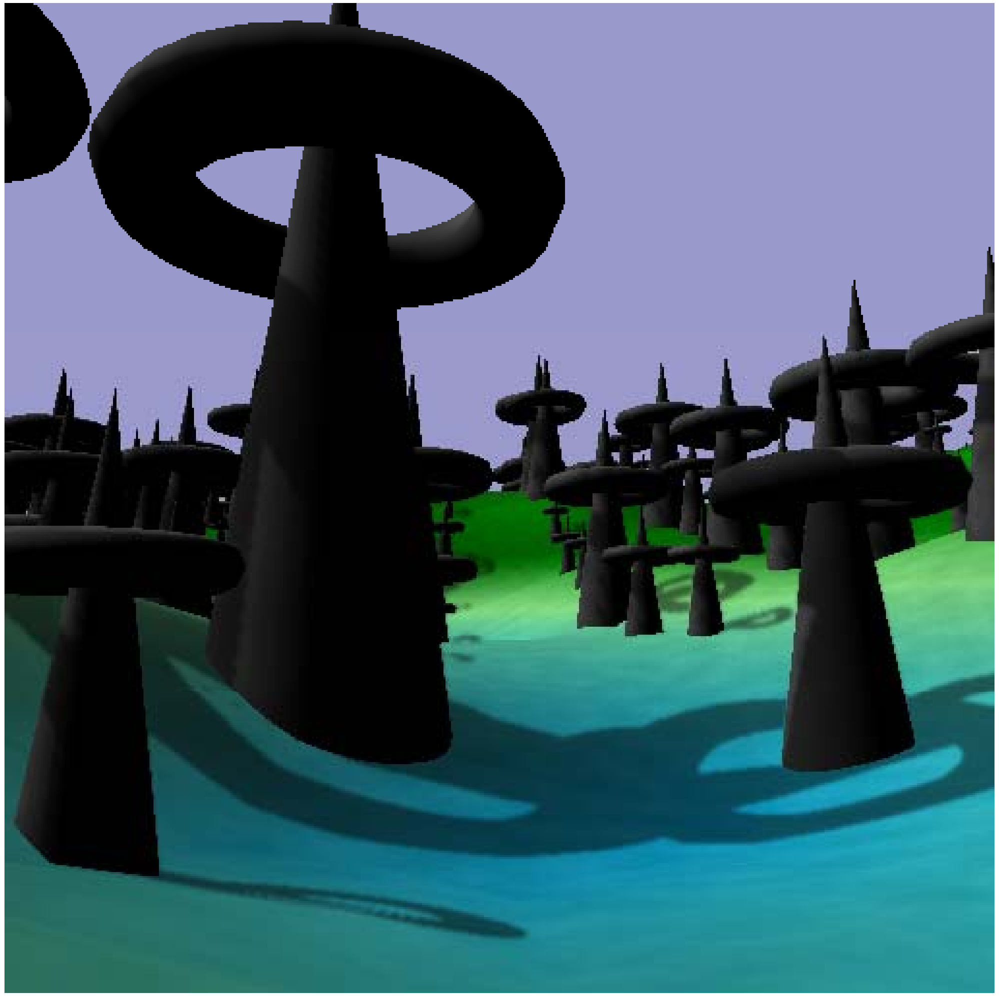
# Solution for Projection Aliasing

- Recommendations
  - Small **ambient** term
  - **Diffuse term** hides artifacts
  - **Specular term** not problematic
    - Light and view direction almost identical
    - Shadow Map resolution sufficient

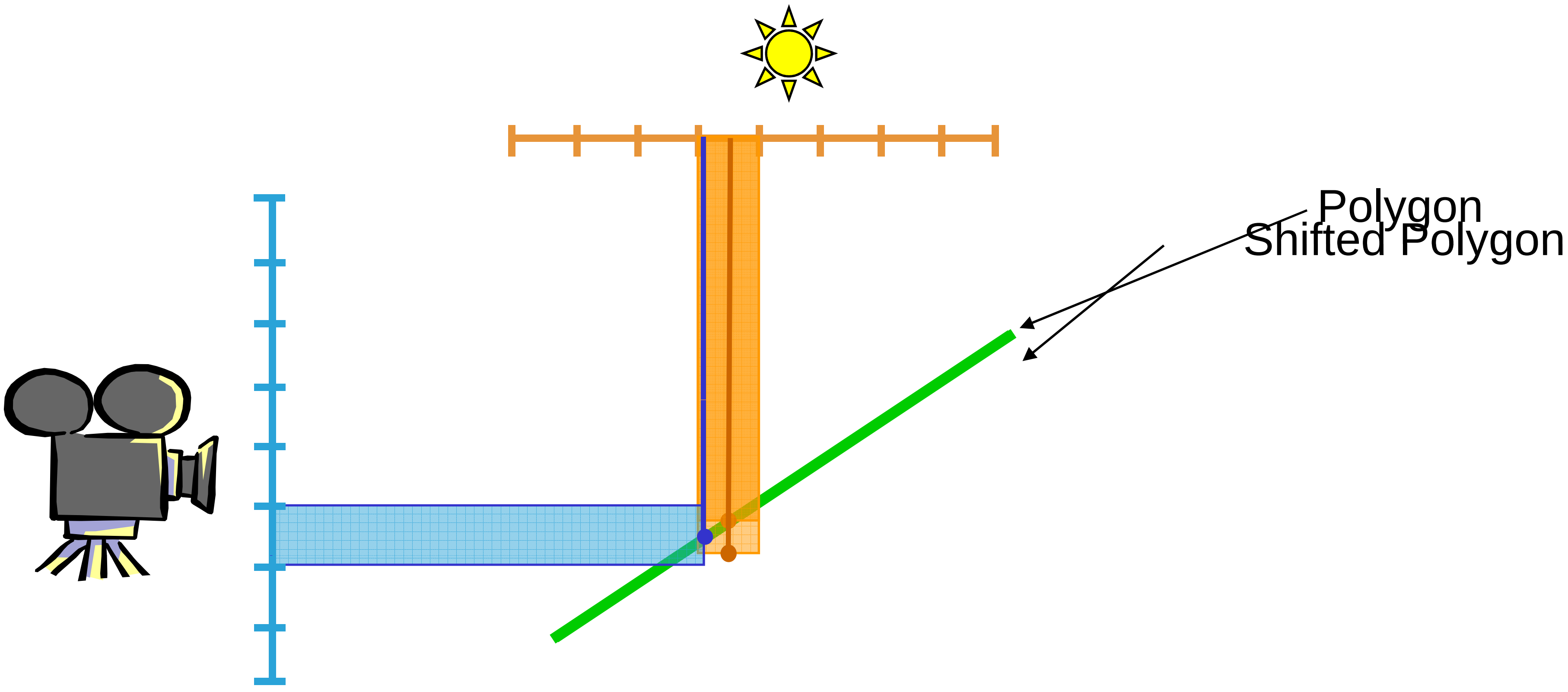


# Solution for Projection Aliasing

- **Blur** shadows
  - Hides artifacts
  - Soft shadow borders
- Render shadow result values to separate texture and blur



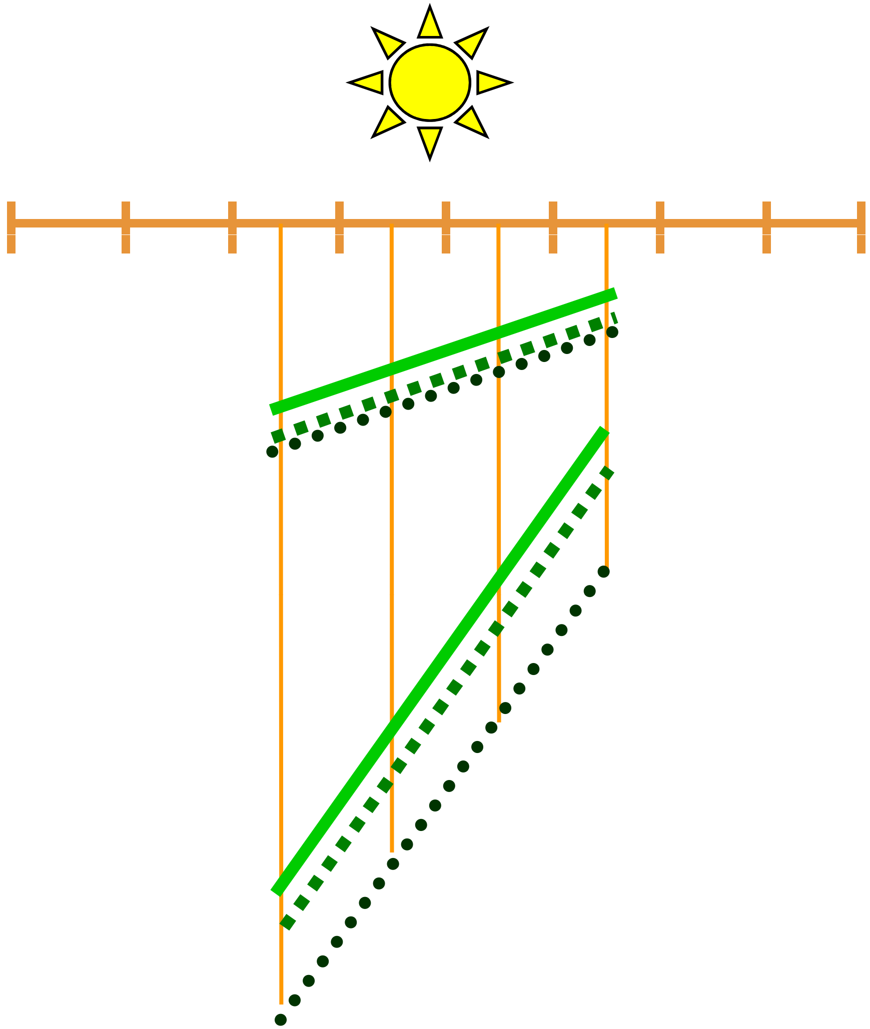
# Solution for Incorrect Self-Shadowing

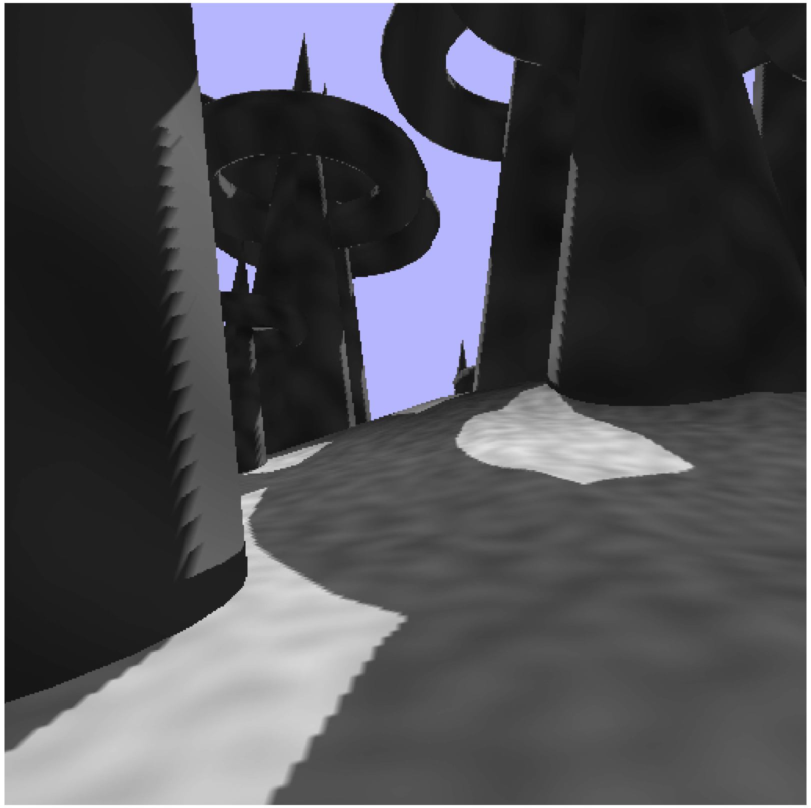



$z_{Aug} > z_{Licht}$ ➡ Incorrect Self-shadowing

$z_{Aug} < z_{Licht}$ ➡ No Self-shadowing

- How to choose bias?





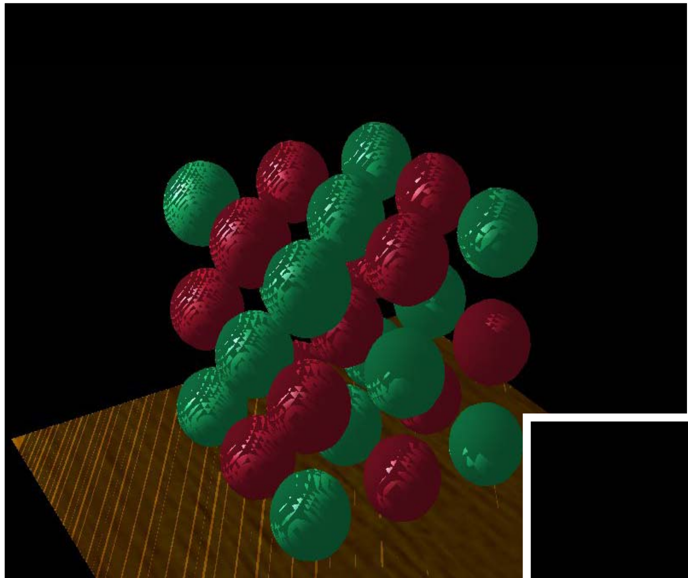
No Bias

Constant Bias

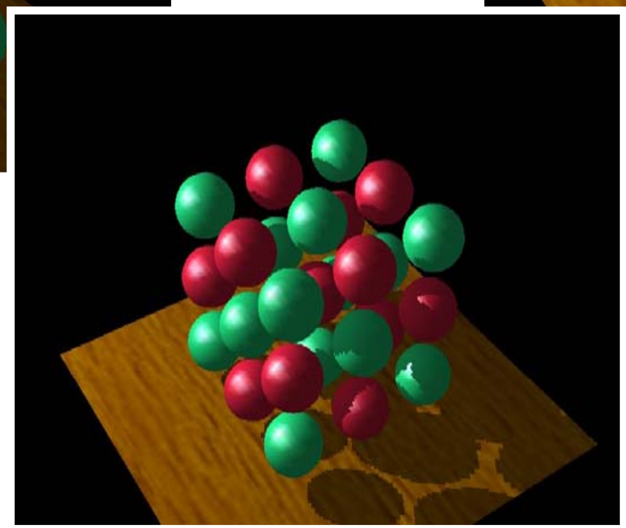
Slope-Scale Bias

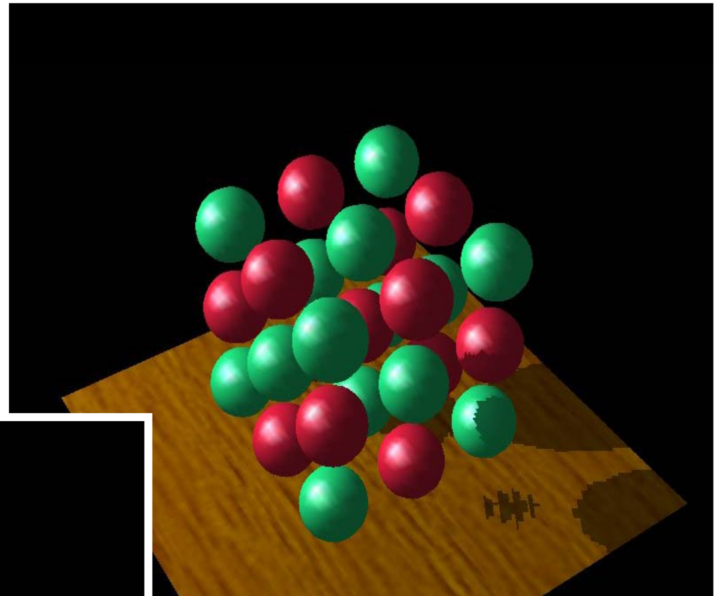
# Depth Bias

- `glPolygonOffset(1.1,4.0)` works well
  - Works in window coordinates



*Too little bias, everything begins to shadow*

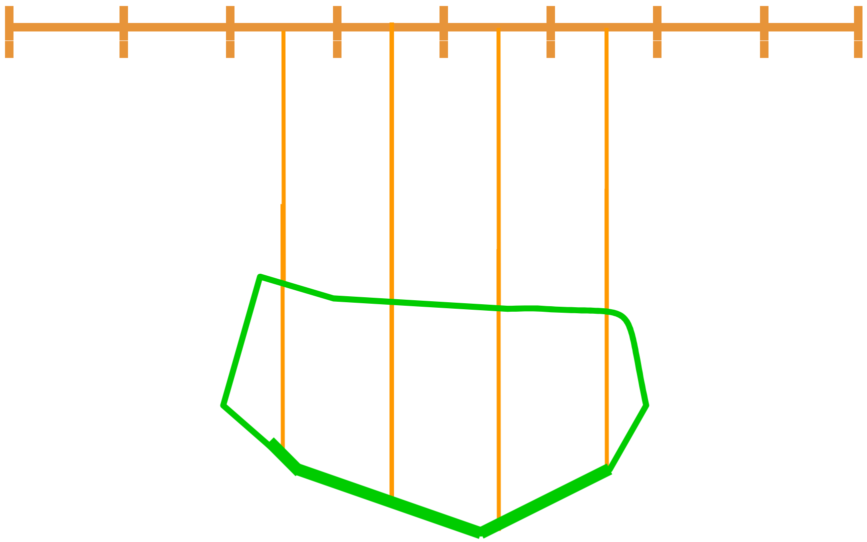


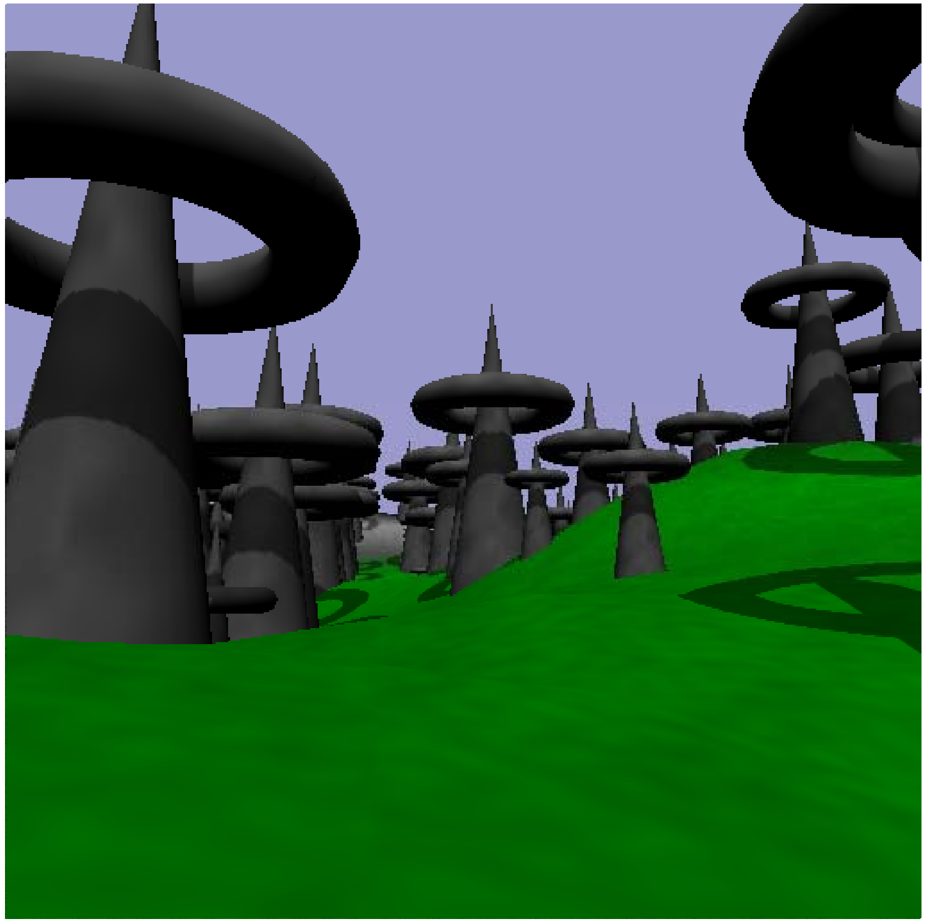


*Too much bias, shadow starts too far back*

# Solution for Incorrect Self-Shadowing

- Other possibility:





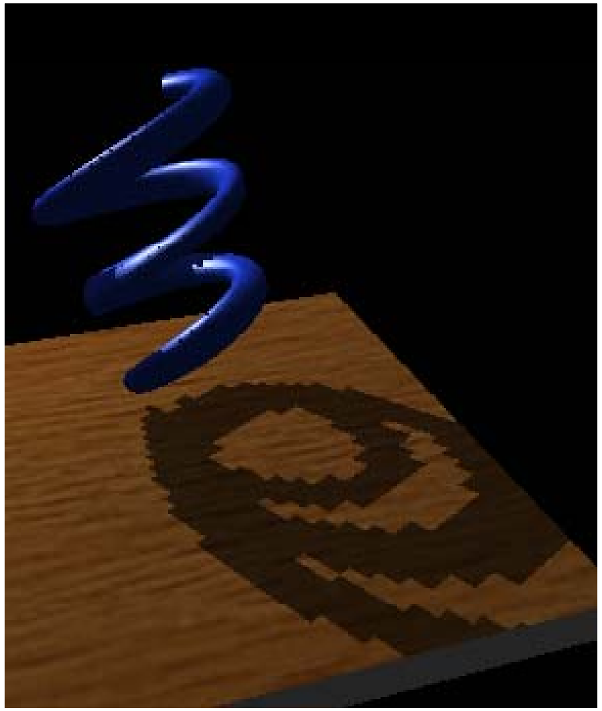
- Previous: render front faces into Shadow Map
- Now: render back faces into Shadow Map: **Back-Side Rendering**

# Problem: Aliasing Artifacts

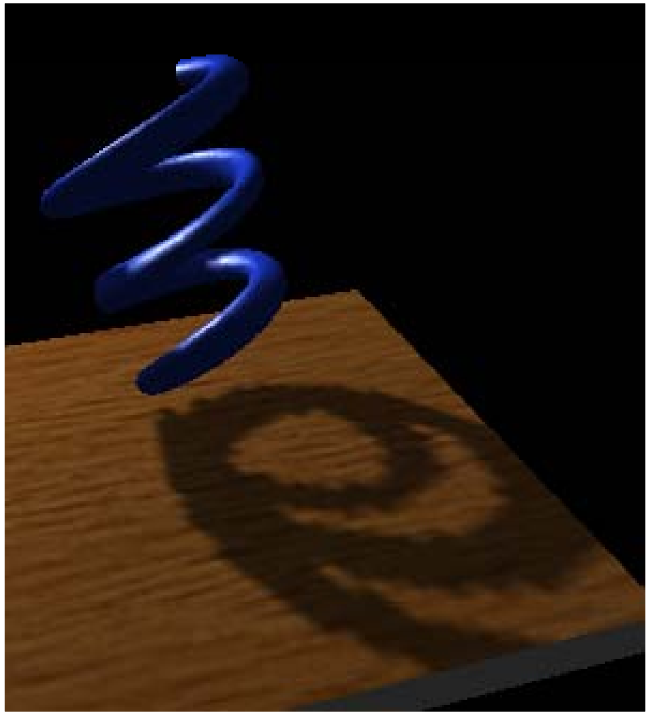
- Resolution mismatch image/shadow map!
  - Use perspective shadow maps
- Use “percentage closer” filtering
  - Normal color filtering cannot be used
  - Filter lookup result, not depth map values!
  - 2x2 PCF in hardware for NVIDIA
  - Better: Poisson-disk distributed samples (e.g., 6 averaged samples)

# Shadow Map Filtering

GL_NEAREST



GL_LINEAR



# Shadow Map Summary

- Advantages
  - Fast – only one additional pass
  - Independent of scene complexity
    (no additional shadow polygons!)
  - Self shadowing (but beware bias)
  - Can sometimes reuse depth map
- Disadvantages
  - Problematic for omnidirectional lights
  - Biasing tweak (light leaks, surface acne)
  - Jagged edges (aliasing)

# OGRE shadow demo

# Conclusions

- Shadows are very important but still difficult
- Many variations based on shadow volumes/shadow maps to do shadowing:
  - Variance shadow mapping (VSM)
  - Perspective shadow mapping (PSM)
  - Hierarchical shadow volume
  - Subdivided shadow maps
  - …