# Real-Time Rendering
# *Ad-hoc Shadows*

CSE 781

Prof. Roger Crawfis

# Ad-Hoc and Custom Shadows

- Fake proxy geometry.
- Projection of model to a plane.
- Projection of a texture to a plane.

# Fake Proxy Shadow

- Shadows are simple hand-drawn polygons or textures.

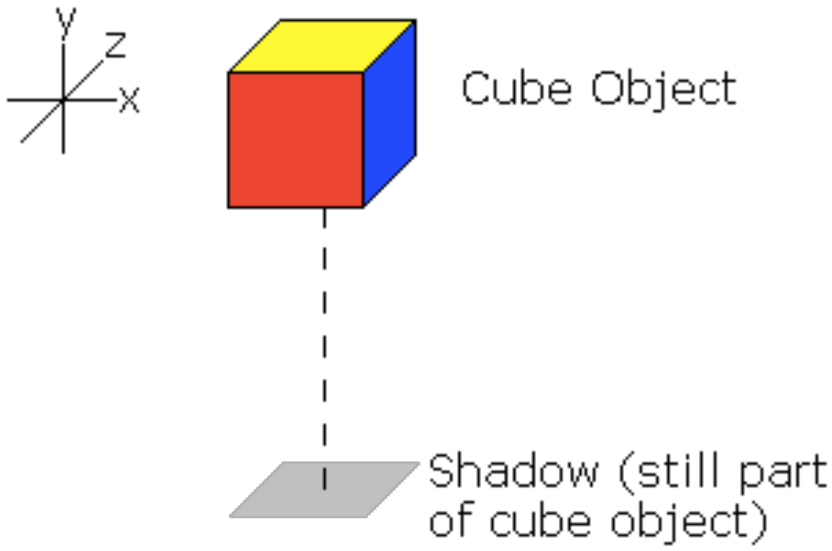

Images from TombRaider. ©Eidos Interactive.

# Fake Proxy Shadow

- Neither static lighting or dynamic lighting – it is faked.
- Do not care whether it is a static or dynamic occluder.
- Typically a single object (occluder) to a single, and simple, object (receiver).
- Hard and soft (fake) shadows are easily supported.
- For certain cases works great!

# Fake Proxy Shadow

- Approximation of shadow position and shape based on object's typical use.
- Typically assumes overhead lighting.
- Typically assumes a single flat ground plane as a receiver.
- E.g., draw the bottom of the bounding box.



# Fake Proxy Shadow

- Consider this model of a desk with a fake shadow using an ellipse:

- Know where the shadow is going to be.
- Will change some depending on the light placement in the room, but good enough!
- The ellipse is part of the model.

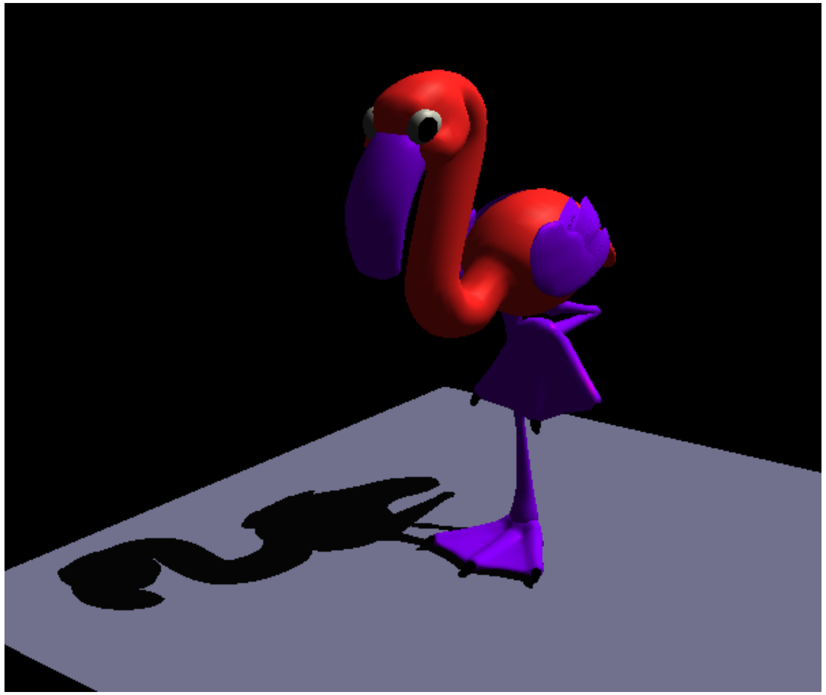
# Fake Proxy Shadow

# Fake Proxy Shadow

- Quite complex model.
- Know it will sit on a flat floor.
- Will fail if we place another object behind or underneath it.

# Projected Occluder

- Shadows for large planar receivers
  - Ground plane
  - Walls
- Use mathematics to flatten (splat) the object to the plane.



# Projected Occluder

- Works for:
  - Static or dynamic occluders.
  - Only planar receivers.
    - A wall and a floor can be shadowed separately.
  - Static or dynamic light sources.
  - Mainly hard shadows.
  - Usually a single light source.

# Projected Occluder

- Projection of a vertex, *v*, to a plane with normal, *n*, and coefficient *d*.

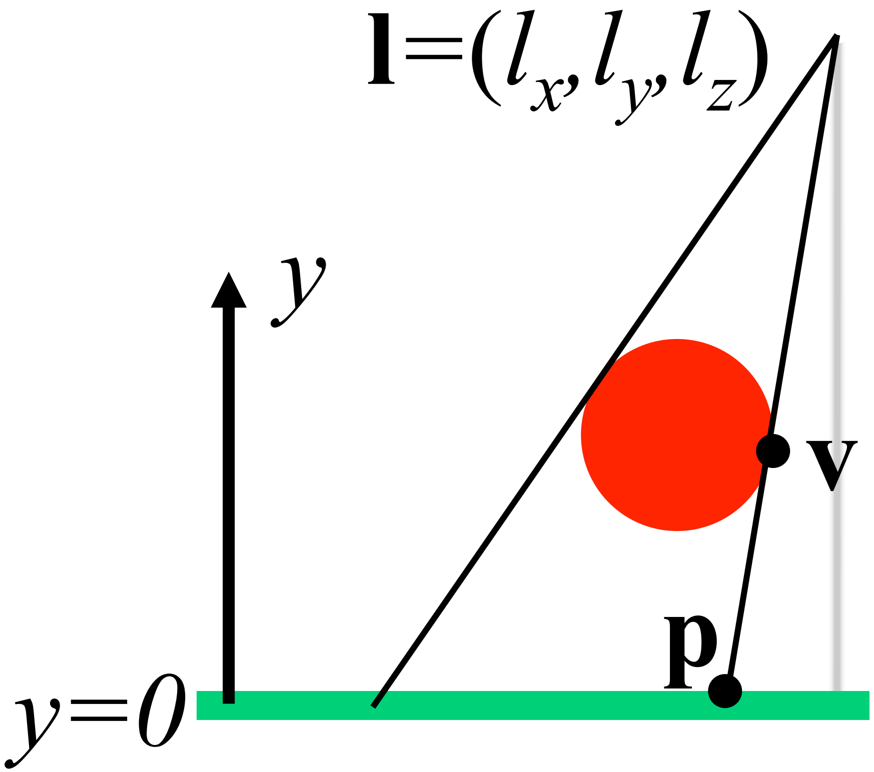
$$\vec{n} \bullet \vec{x} + d = 0$$

$$\vec{v}' = \vec{l} - \frac{d + \vec{n} \bullet \vec{l}}{\vec{n} \bullet (\vec{v} - \vec{l})} (\vec{v} - \vec{l})$$

- Could be done in shader, but also leads to a 4x4 matrix.

# Projected Occluder

- Example: *xz* plane at *y=0*




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

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

# Projected Occluder

- Transformation as a 4 by 4 matrix

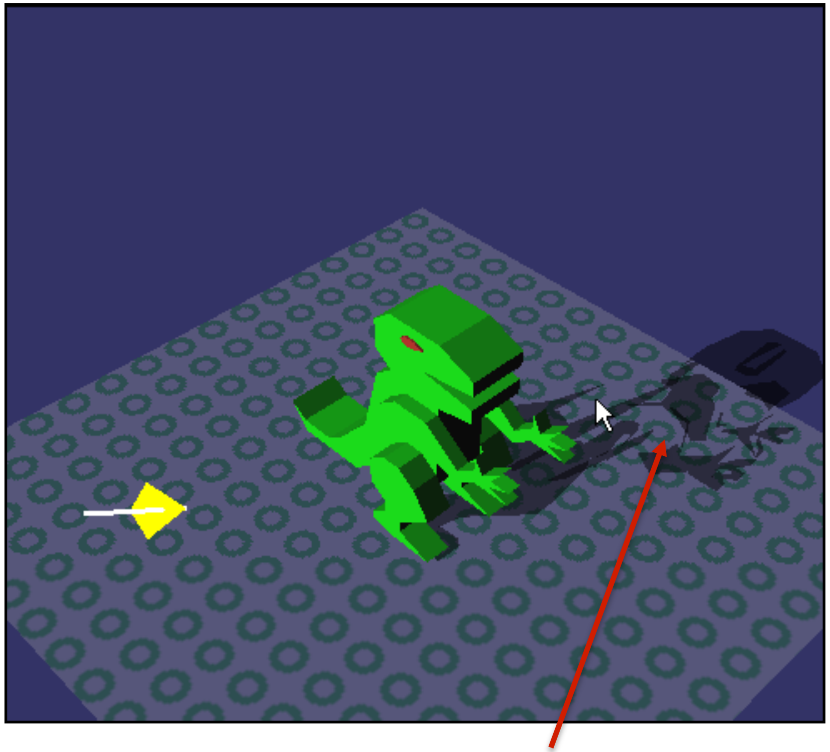
$$\vec{p} = \begin{pmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_z & l_y & 0 \\ 0 & -1 & 0 & l_y \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix}$$

# Projected Occluder

- Basic algorithm
  - Render scene (full lighting)
  - For each receiver plane
    - Compute projection matrix *M*
    - Multiply with actual transformation (modelview)
      - Note, even though this is a projection.
      - Need to flatten it in world space.
    - Render selected (occluder) geometry
      - Darken/Black

# Projected Occluder Problems

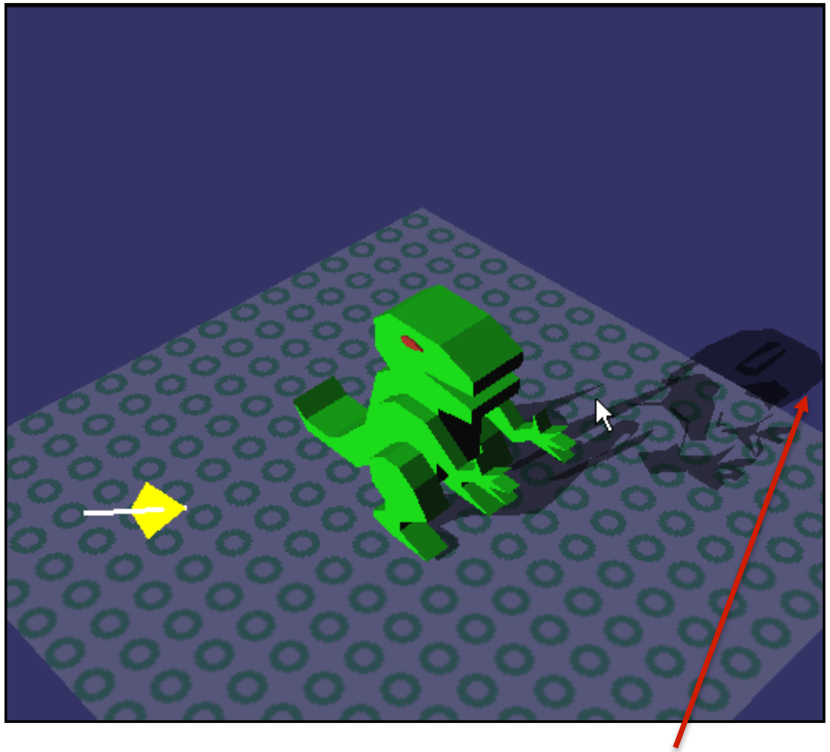
- Z-Fighting
  - Use bias when rendering shadow polygons
  - Use stencil buffer (no depth test)



Z fighting

# Projected Occluder Problems

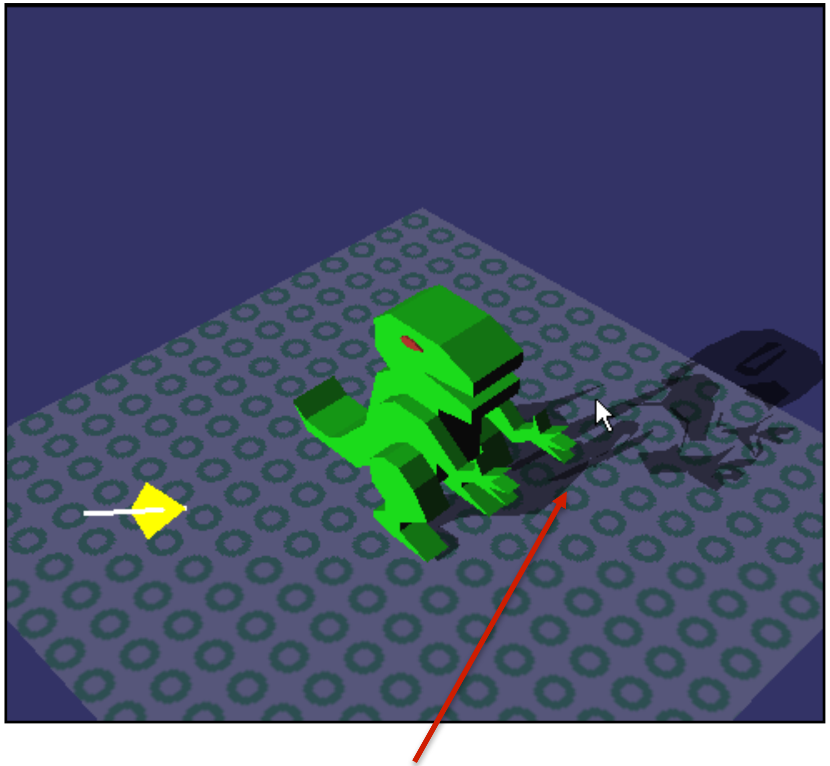
- Bounded receiver polygon
  - Use stencil buffer (restrict drawing to receiver area)



extends off ground region

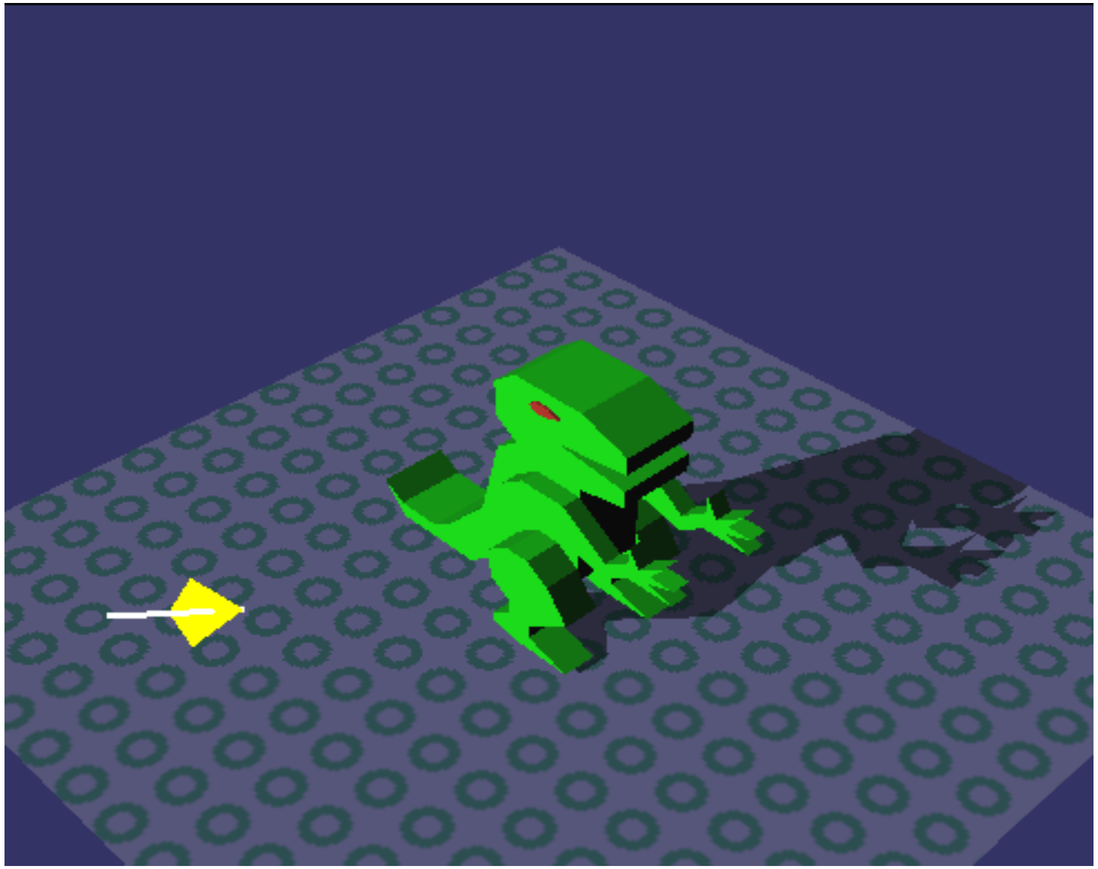
# Projected Occluder Problems

- Shadow polygon overlap
  - Use stencil count (only the first pixel gets through)



double blending

# Projected Occluder - Fixed

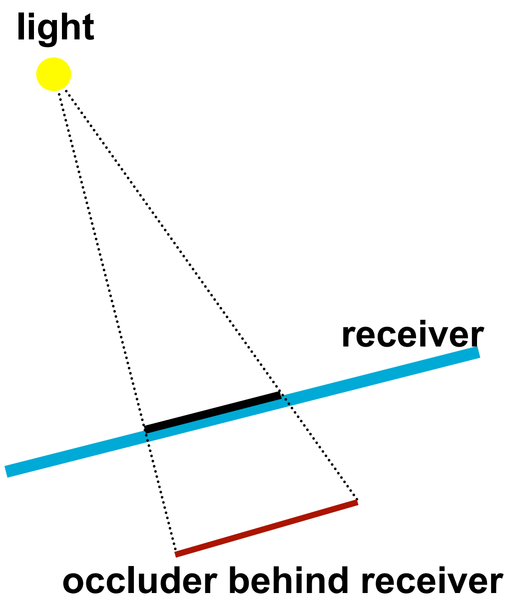


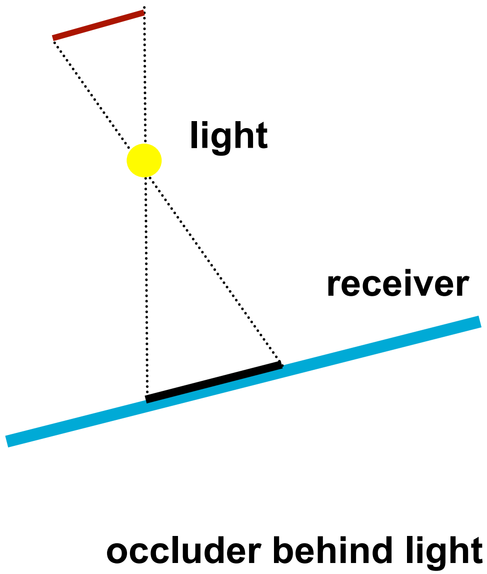
# Projected Occluder Algorithm

- Stencil buffer algorithm (1bit stencil)
  1. Render scene without receiver polygon
  2. Clear stencil buffer
  3. Render receiver plane
     - Set the stencil buffer for all visible pixels
  4. Render occluder polygons
     - No depth testing
     - Check if stencil buffer is set
     - Use the stencil operation ‘clear’
     - Blend in the polygons (darken)

# Projected Occluder Problems

- Wrong Shadows & Anti-Shadows
  - Objects behind light source
  - Objects behind receiver



# Projected Occluder

- Summary
  - Only practical for very few, large receivers
  - Easy to implement
  - Use stencil buffer (z fighting, overlap, receiver)
  - Requires occluder geometry to be redrawn for each light source.
    - Can use a simplified model (proxy occluder geometry).

# Projected Shadow Texture

- Sky layers
- Cast shadows

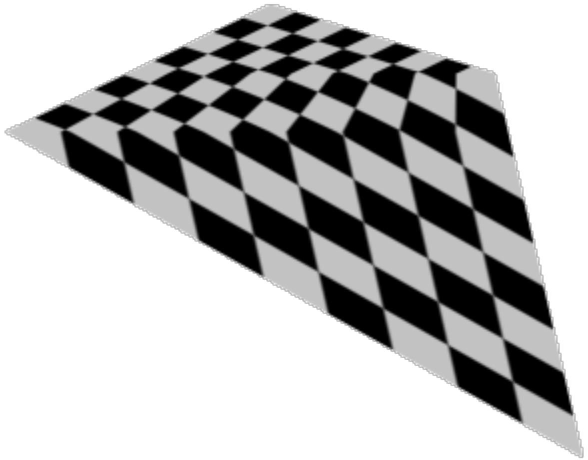
# Projective Textures

- Textures can be projected like a slide projector.
- Before we talk about this projective textures let's look at texture interpolation.


*Source: Wolfgang Heidrich [99]*

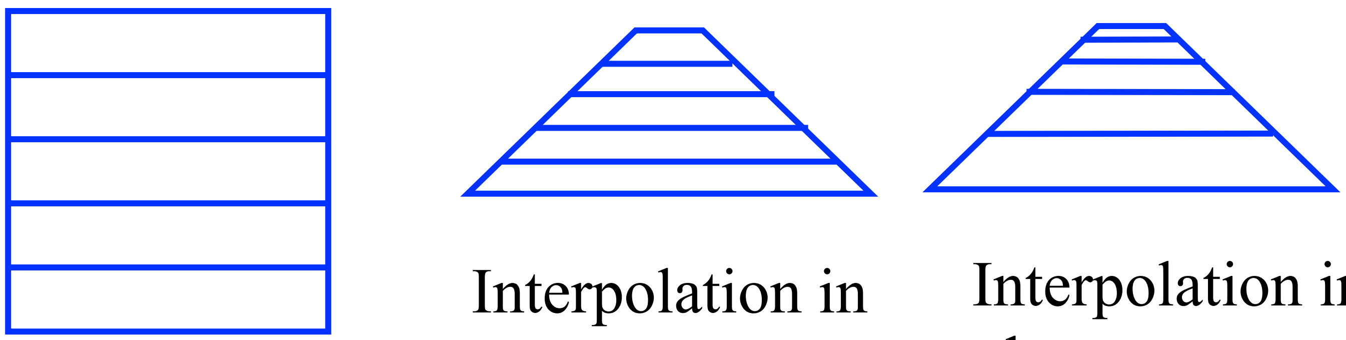
# Perspective-Correct Texturing

- While we think of 2D texture mapping using only the (s, t) coordinates, doing this will lead to errors.
- The texture will *swim*.
- A fix for this is needed for regular 2D texture mapping.



# Perspective-Correct Texturing

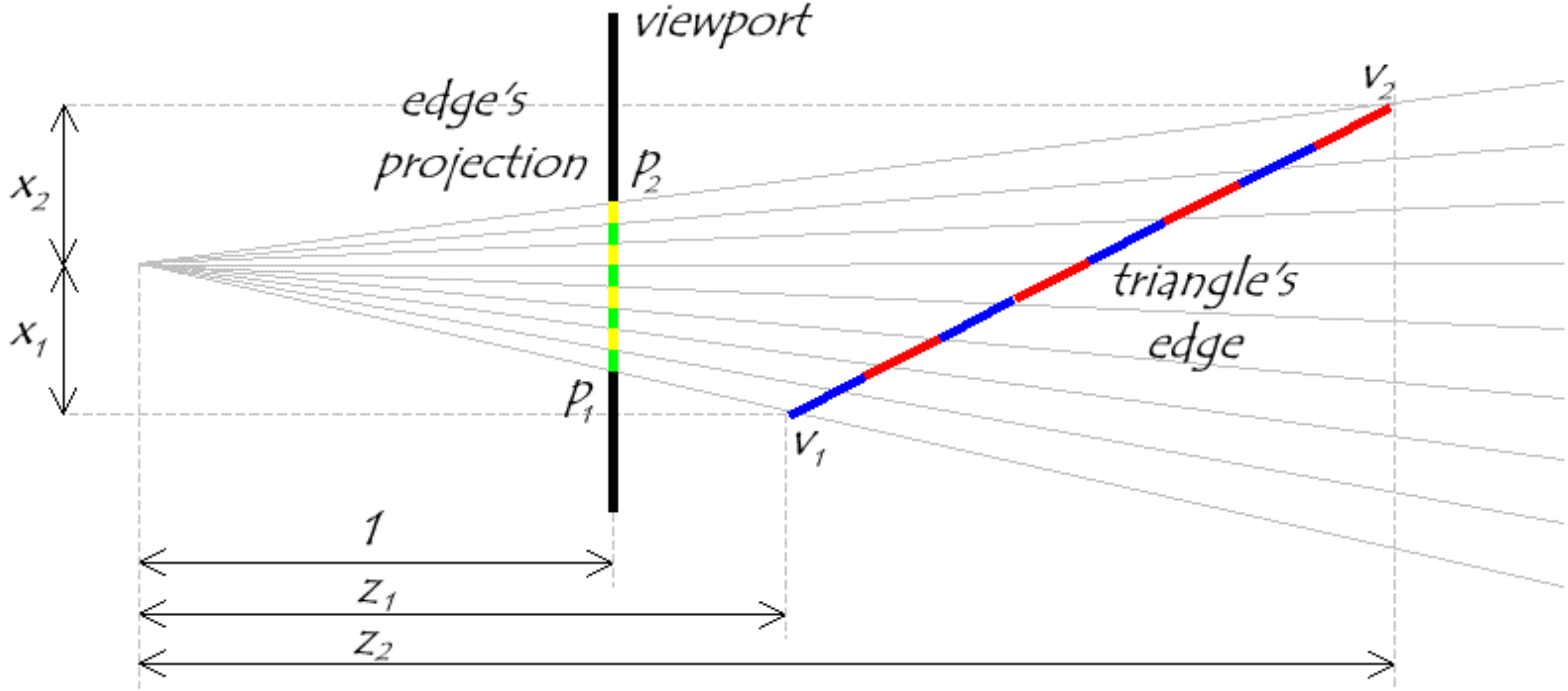
- Interpolation in screen space is not the same as interpolation in 3-space
  - Problem is perspective
  - Need to interpolate in the plane of the triangle.



Interpolation in screen space

Interpolation in plane

# Visualizing the Problem



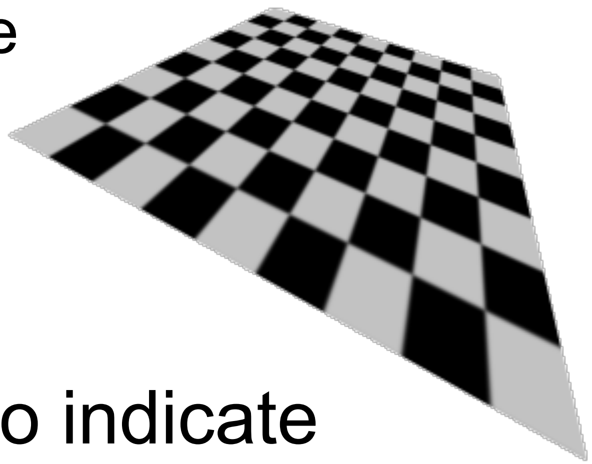

*Notice that uniform steps on the image plane do not correspond to uniform steps along the edge.*

# Perspective-Correct Texturing

- 2D perspective-correct texture mapping
  - (s, t) should be interpolated linearly in eye-space.
  - Compute per-vertex s/w, t/w, and 1/w
  - Linearly interpolate these three parameters over the polygon.
  - Per-fragment compute:

    s' = (s/w) / (1/w)

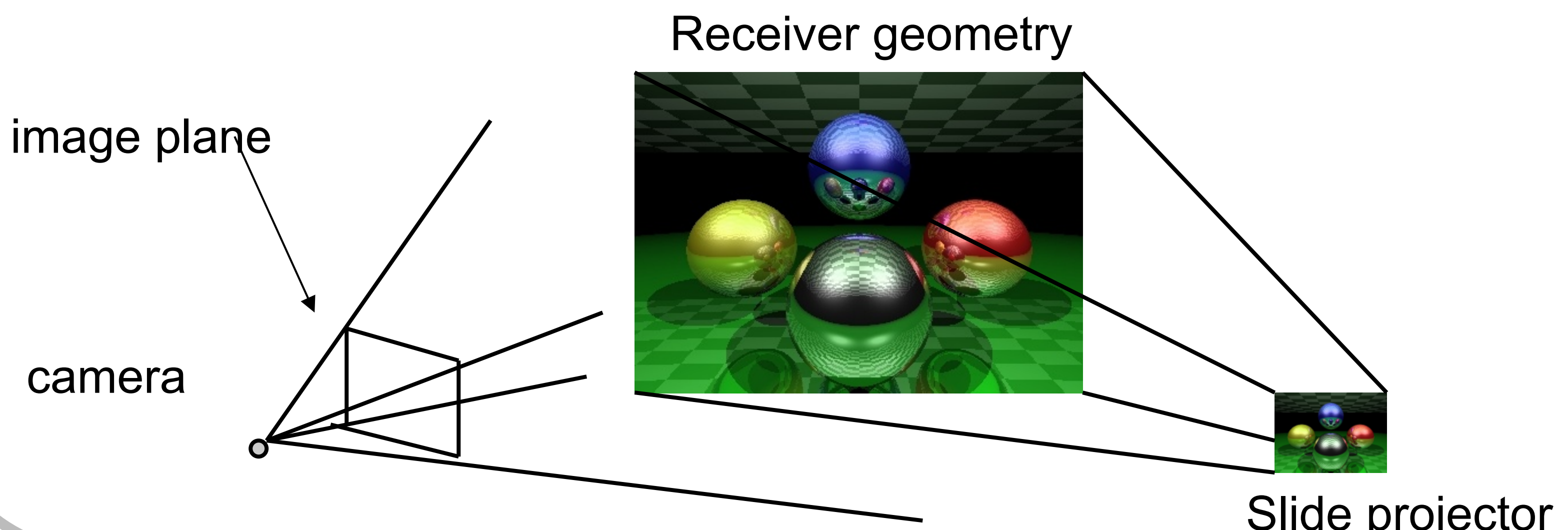
    t' = (t/w) / (1/w)



  - There is an OpenGL hint to indicate perspective texture interpolation.
    - This is on by default with modern hardware.

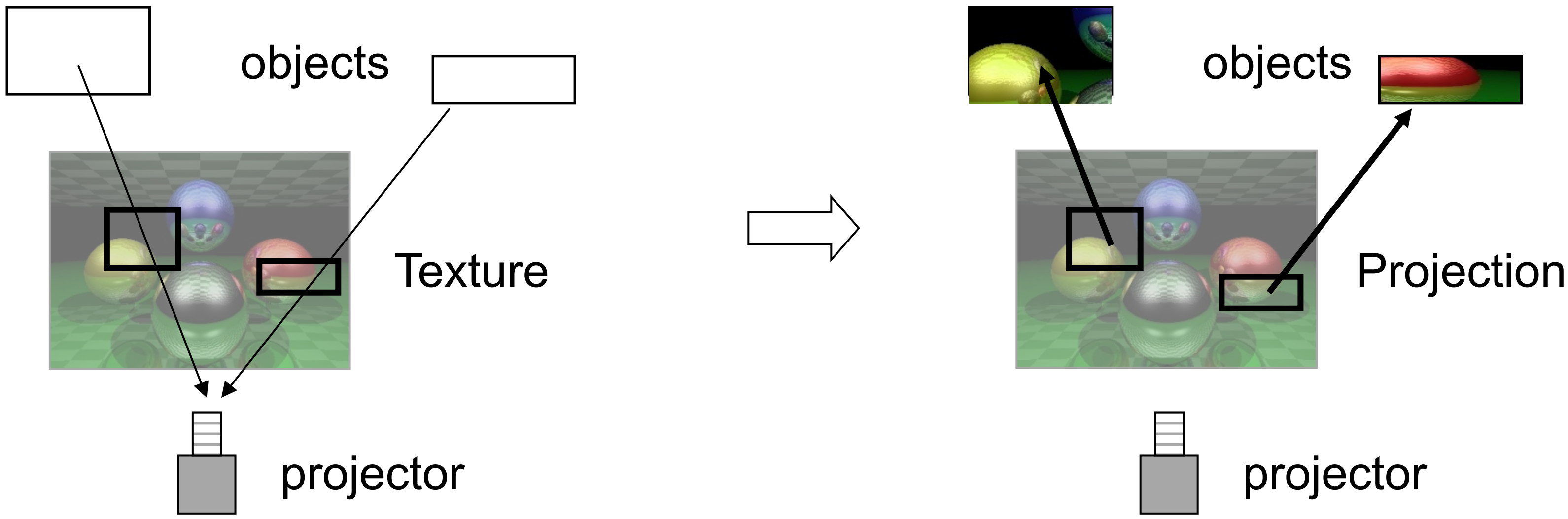
# Projective Textures

- Similar to projecting objects to the screen.
- Now project the scene to the light source.
- Use this projection from the receivers as their texture coordinates (a texture parameterization).



# Projective Textures

- Texture Coordinates – Project the objects to the “image plane” of the projector and use the projector’s NDC to calculate the texture coordinates



# Projective Textures

- The receiver's need to know about the projected texture, the ***light*** does not automatically apply to objects and is not an OpenGL state.
- OpenGL allows 4D texture coordinates, which can handle the projection.

# Projective Texturing

- Tricking hardware into doing projective textures
  - By interpolating q/w (perspective correction), hardware computes per-fragment
    - (s/w) / (q/w) = s/q
    - (t/w) / (q/w) = t/q
  - Net result: Projective texturing
    - OpenGL (glTexGen) or a vertex shader, specifies the texture parameterization. Typically want this in world space, but like headlights can be done in eye space.

# Projective Texture Shadows

Light's point-of-view

Shadow projective texture (modulation image or light-map)

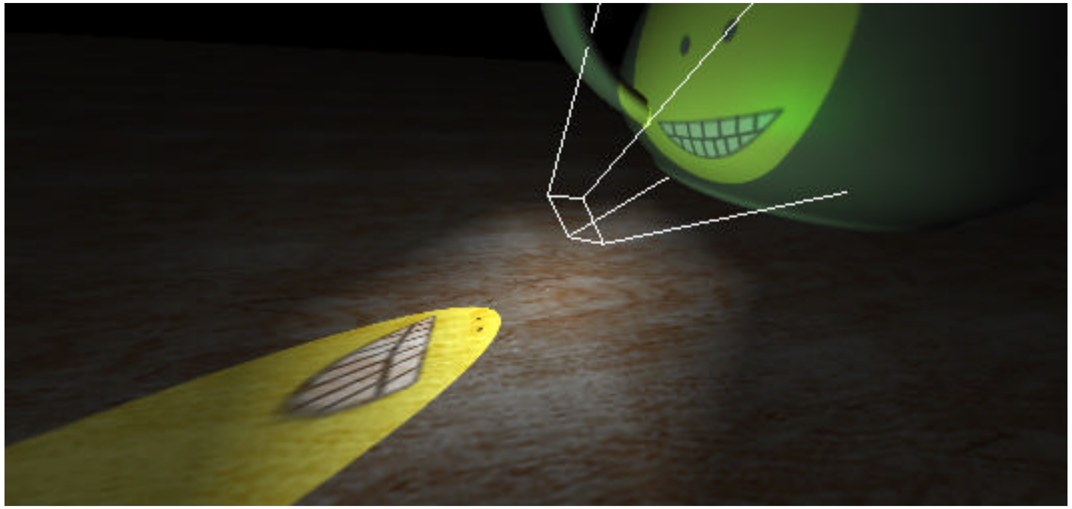
Eye's point-of-view, projective texture applied to ground-plane

# Projective Texture Shadows

- Two-pass approach
- For each light source:
  - Create a light camera that encloses shadowed area (bounding box of the occluder).
  - Render shadow casting objects into light's view.
    - Use a simple shader (set fragment color to black).
  - Create projective texture from light's view
- Render Scene using the projective textures.
  - Render fully-lit shadow receiving objects.
    - Modulate light contribution with the projective-texture for that light.
  - Render fully-lit shadow casting objects

# Projected Texture Problems

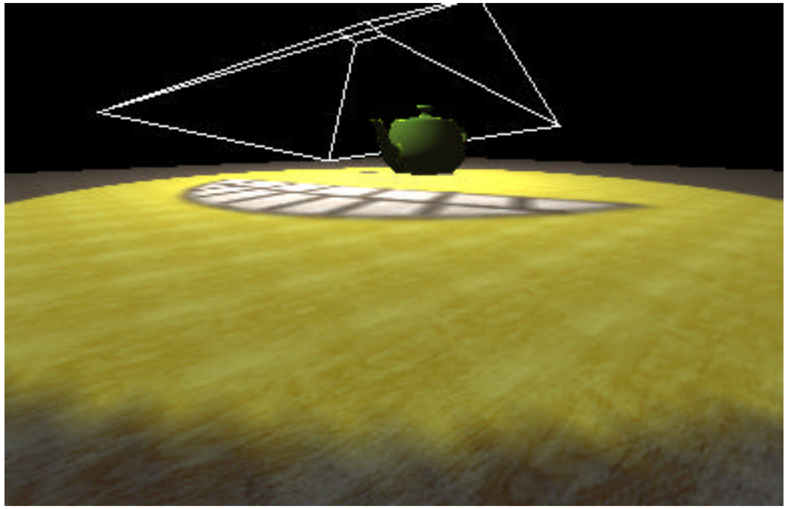
- Similar problems to the projected occluders:
  - Receiver is behind the projector.
  - Occluder is behind receiver.



*Projected Texture Mapping, Cass Everitt, nVidia*

# Projected Texture Problems

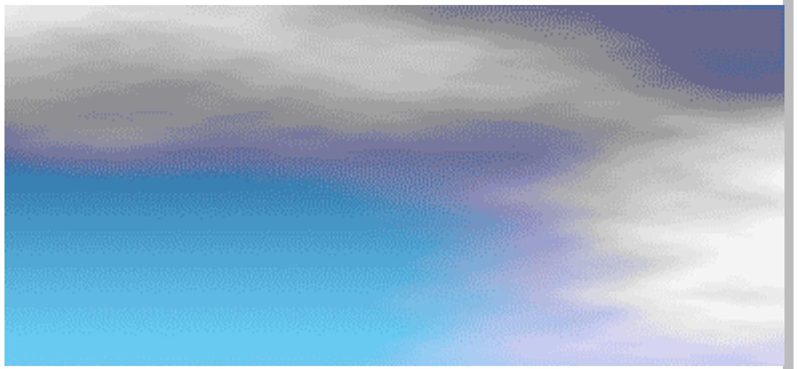
- Precision issues:
  - Occluder very close to light (wide frustum).
  - Projector frustum faces the viewing frustum (sampling rate needed varies greatly).



*Projected Texture Mapping, Cass Everitt, nVidia*

# Projective Texture Shadows

- Texture can easily be projected onto multiple receivers.
- Receivers do not need to be planar.
- Static scenes only or you need to regenerate textures.
  - A sky layer can however move its shadow image with the clouds.



- No self shadowing.
- No area light sources (you can blur the texture though for a fake effect).

# Ad-Hoc Shadow Summary

- A common theme of these methods is that the occluders and/or receivers were predetermined.
- For Fake shadows, the occluder was part of the model. Any receiver rendered before it would be darkened.
- For the projection-based techniques, either the occluder had a priori knowledge of the receiver (projected occluders) or the receiver had a priori knowledge of the occluder(s) (projected shadow textures).
- The occluder must also be different than the receiver (no self-shadowing).