What is projective texturing?

- An intuition for projective texturing
  - The slide projector analogy

Source: Wolfgang Heidrich [99]

Texture matrix

\[
\begin{bmatrix}
    s \\
    t \\
    r \\
    q
\end{bmatrix} = \begin{bmatrix}
    \frac{1}{2} & 0 & 0 & \frac{1}{2} \\
    0 & \frac{1}{2} & 0 & \frac{1}{2} \\
    0 & 0 & 1 & \frac{1}{2} \\
    0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    \text{Light Frustum (projection)} \\
    \text{View Matrix (lookat)} \\
    \text{Modeling Matrix}
\end{bmatrix} \begin{bmatrix}
    x_0 \\
    y_0 \\
    z_0 \\
    w_0
\end{bmatrix}
\]

From “The Cg Tutorial,” p. 252.
# Projective texturing vertex shader

```cpp
void C9E4v_projTexturing(float4 position : POSITION,
float3 normal : NORMAL,
out float4 oPosition : POSITION,
out float4 texCoordProj : TEXCOORD0,
out float4 diffuseLighting : TEXCOORD1,
uniform float Kd,
uniform float4x4 modelViewProj,
uniform float3 lightPosition,
uniform float4x4 textureMatrix)
{
    oPosition = mul(modelViewProj, position);
    // Compute texture coordinates for querying the projective texture
    texCoordProj = mul(textureMatrix, position);
    // Compute diffuse lighting
    float3 N = normalize(normal);
    float3 L = normalize(lightPosition - position.xyz);
    diffuseLighting = Kd * max(dot(L, N), 0);
}
```

From "The Cg Tutorial"

---

## Watch out for reverse projection!


---

## Projective texturing pixel shader

```cpp
void C9E5f_projTexturing(float4 texCoordProj : TEXCOORD0,
float4 diffuseLighting : TEXCOORD1,
out float4 color : COLOR,
uniform sampler2D projectiveMap)
{
    // Fetch color from the projective texture
    float4 projColor = tex2Dproj(projectiveMap, texCoordProj);
    color = projColor * diffuseLighting;
}
```

From "The Cg Tutorial"

---

## A dramatic shadow in 2K Games’ BioShock

The shadow mapping concept (1)

- Depth testing from the light’s point-of-view
  - Two pass algorithm
- First, render depth buffer from the light’s point-of-view
  - the result is a “depth map” or “shadow map”
  - essentially a 2D function indicating the depth of the closest pixels to the light
- This depth map is used in the second pass


The shadow mapping concept (2)

- Shadow determination with the depth map
  - Second, render scene from the eye’s point-of-view
  - For each rasterized fragment
    - determine fragment’s XYZ position relative to the light
    - this light position should be setup to match the frustum used to create the depth map
    - compare the depth value at light position XY in the depth map to fragment’s light position Z


The shadow mapping concept (3)

- The Shadow Map Comparison
  - Two values
    - \( A = Z \) value from depth map at fragment’s light XY position
    - \( B = Z \) value of fragment’s XYZ light position
  - If \( B \) is greater than \( A \), then there must be something closer to the light than the fragment
    - then the fragment is shadowed
  - If \( A \) and \( B \) are approximately equal, the fragment is lit


Shadow mapping with a picture in 2D (1)

- The \( A < B \) shadowed fragment case

Shadow mapping with a picture in 2D (2)

The A \cong B unshadowed fragment case

- depth map Z = A
- eye view image plane, a.k.a. the frame buffer


Visualizing the shadow mapping technique (1)

- A fairly complex scene with shadows


Shadow mapping with a picture in 2D (3)

Note image precision mismatch!

- The depth map could be at a different resolution from the framebuffer
- This mismatch can lead to artifacts


Visualizing the shadow mapping technique (2)

- Compare with and without shadows

• The scene from the light’s point-of-view

**FYI:** from the eye’s point-of-view again

Visualizing the shadow mapping technique (5)

• Projecting the depth map onto the eye’s view

**FYI:** depth map for light’s point-of-view again

Visualizing the shadow mapping technique (6)

• Projecting light’s planar distance onto eye’s view
Visualizing the shadow mapping technique (7)

- Comparing light distance to light depth map

Green is where the light planar distance and the light depth map are approximately equal.

Non-green is where shadows should be.


Visualizing the shadow mapping technique (8)

- Scene with shadows

Notice how specular highlights never appear in shadows.

Notice how curved surfaces cast shadows on each other.


Depth map bias issues

- Too little bias, everything begins to shadow
- Too much bias, shadow starts too far back
- Just right


Dedicated hardware shadow mapping support

- Performs the shadow test as a texture filtering operation
  - looks up texel at (s/q, t/q) in a 2D texture
  - compares lookup value to r/q
  - if texel is greater than or equal to r/q, then generate 1.0
  - if texel is less than r/q, then generate 0.0
- Modulate color with result
  - zero if fragment is shadowed or unchanged color if not

Hardware shadow map filtering example

GL_NEAREST: blocky
GL_LINEAR: antialiased edges

Low shadow map resolution used to heighten filtering artifacts


Combine with Projective Texturing for Spotlight Shadows

- Use a spotlight-style projected texture to give shadow maps a spotlight falloff