Shadows

Prof. George Wolberg
Dept. of Computer Science
City College of New York
Objectives

• Introduce Shadow Algorithms
• Expand to projective textures
Flashlight in the Eye Graphics

• When do we not see shadows in a real scene?
  - When the only light source is a point source at the eye or center of projection
  - Shadows are behind objects and not visible

• Shadows are a global rendering issue
  - Is a surface visible from a source
  - May be obscured by other objects
Projective Shadows

• Oldest methods
  - Used in flight simulators to provide visual clues

• Projection of a polygon is a polygon called a shadow polygon

• Given a point light source and a polygon, the vertices of the shadow polygon are the projections of the original polygon’s vertices from a point source onto a surface
Shadow Polygon

\((x_i, y_i, z_i)\)
Computing Shadow Vertex

1. Source at \((x_l, y_l, z_l)\)
2. Vertex at \((x, y, z)\)
3. Consider simple case of shadow projected onto ground at \((x_p, 0, z_p)\)
4. Translate source to origin with \(T(-x_l, -y_l, -z_l)\)
5. Perspective projection
   \[
   M = \begin{bmatrix}
   1 & 0 & 0 & 0 \\
   0 & 1 & 0 & 0 \\
   0 & 0 & 1 & 0 \\
   0 & \frac{1}{-y_l} & 0 & 0
   \end{bmatrix}
   \]
6. Translate back
Shadow Process

1. Put two identical triangles and their colors on GPU (black for shadow triangle)
2. Compute two model-view matrices as uniforms
3. Send model-view matrix for original triangle
4. Render original triangle
5. Send second model-view matrix
6. Render shadow triangle
   - Note shadow triangle undergoes two transformations
   - Note hidden surface removal takes care of depth issues
Generalized Shadows

- Approach was OK for shadows on a single flat surface
- Note with geometry shader we can have the shader create the second triangle
- Cannot handle shadows on general objects
- There exists a variety of other methods based on same basic idea
- We’ll pursue methods based on projective textures
Image Based Lighting

• We can project a texture onto the surface in which case we are treating the texture as a “slide projector”
• This technique is the basis of projective textures and image based lighting
• Supported in OpenGL and GLSL through four-dimensional texture coordinates
4D Textures Coordinates

- Texture coordinates \((s, t, r, q)\) are affected by a perspective division so the actual coordinates used are \((s/q, t/q, r/q)\) or \((s/q, t/q)\) for a two dimensional texture.

- GLSL has a variant of the function texture \(\text{textureProj}\) which will use the two- or three-dimensional texture coordinate obtained by a perspective division of a 4D texture coordinate a texture value from a sampler.

\[
\text{color} = \text{textureProj}(\text{my_sampler, tex\_coord})
\]
Shadow Maps

• If we render a scene from a light source, the depth buffer will contain the distances from the light source to each fragment.

• We can store these depths in a texture called a depth map or shadow map

• Note that although we don’t care about the image in the shadow map, if we render with some light, anything lit is not in shadow.

• Form a shadow map for each source
Final Rendering

• During the final rendering we compare the distance from the fragment to the light source with the distance in the shadow map

• If the depth in the shadow map is less than the distance from the fragment to the source, the fragment is in shadow (from this light source)

• Otherwise we use rendered color
Application’s Side

- Start with vertex in object coordinates
- Want to convert representation to texture coordinates
- Form LookAt matrix from light source to origin in object coordinates (MVL)
- From projection matrix for light source (PL)
- From a matrix to convert from [-1, 1] clip coordinates to [0, 1] texture coordinates
- Concatenate to form object to texture coordinate matrix (OTC)
Vertex Shader

uniform mat4 modelview;
uniform mat4 projection;
uniform normalmatrix; // for diffuse lighting
uniform mat4 otc; // object to texture coordinate
uniform vec4 diffuseproduct; // diffuse light*diffuse reflectivity

in vec4 vPosition;
in vec4 normal;

out vec4 color;
out vec4 shadowCoord;

void main()
{
    // compute diffuse color as usual
    // using normal, normal matrix, diffuse product
    color = ...;

    gl_Position = projection*modelview*vPosition;
    shadowCoord = OTC*vPosition;
}
**textureProj function**

- Application provides the shadow map as a texture object.
- The GLSL function `textureProj` compares the third value of the texture coordinate with the third value of the texture image.
- For nearest filtering of the texture object, `textureProj` returns 0.0 if the shadow map value is less than the third coordinate and 1.0 otherwise.
- For other filtering options, `textureProj` returns values between 0.0 and 1.0.
uniform sampler2DShadow ShadowMap;

in vec4 shadowCoord;
in vec4 Color;

main()
{
    // assume nearest sampling in ShadowMap
    float shadeFactor = textureProj(ShadowMap, shadowCoord);
    gl_FragColor = vec4(shadeFactor*Color.rgb, Color.a)
}