More Rendering

Thomas A. Funkhouser

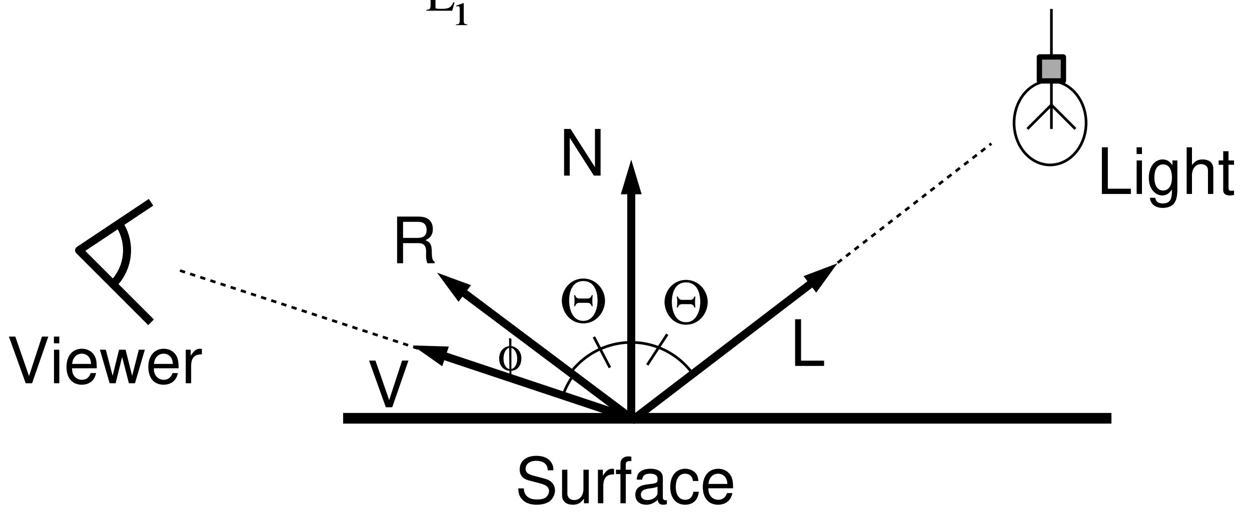
Overview

- Lighting Review
- Transparency
- Shadows
- Texture Mapping

Lighting Review

- **Light sources**
  - – Point
  - – Directional
  - – Spot
  - – Polygon
- **Surface illumination**

$$I = I_E + K_A I_{AL} + \sum_{L_i} (K_D (N \bullet L_i) + K_S (V \bullet R_i)^n) I_{L_i}$$

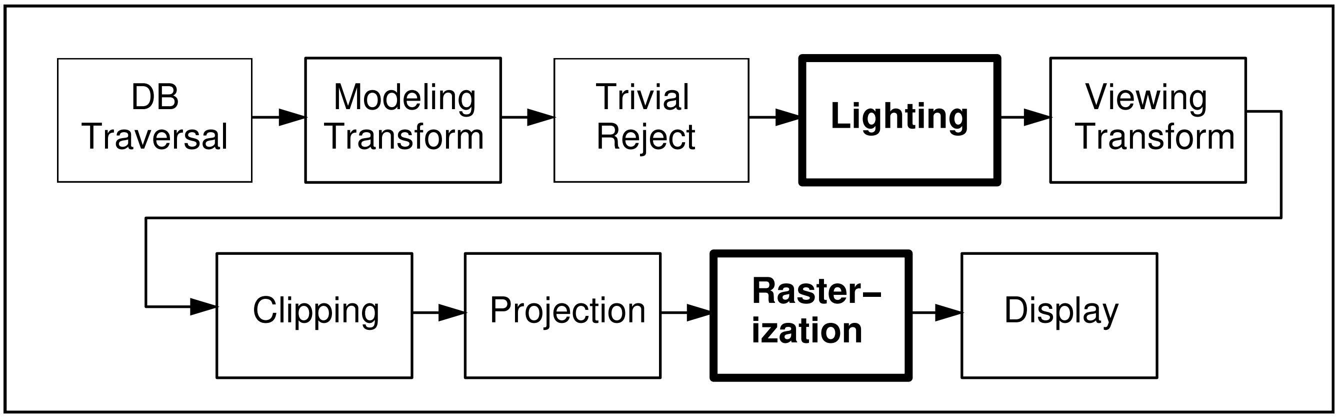


- **Polygon shading**
  - – Flat
  - – Gouraud
  - – Phong

Transparency

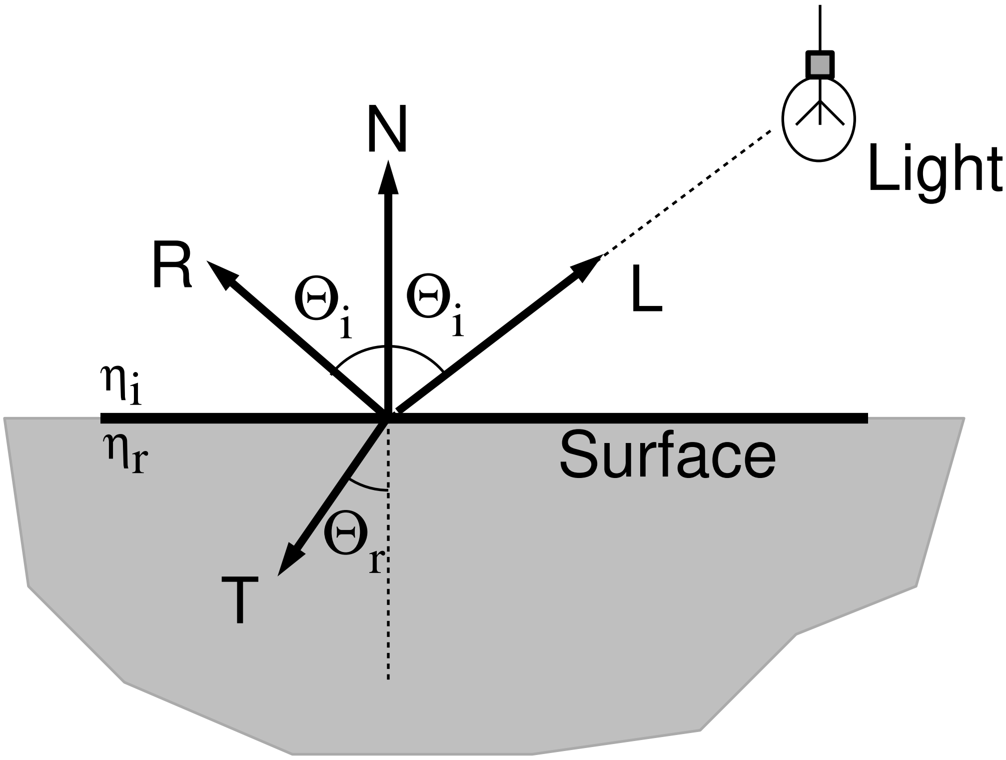
- **Goal**
  – Simulate translucent materials (e.g., glass)

Greg Ward



Transparency

- **Transmission into objects:**

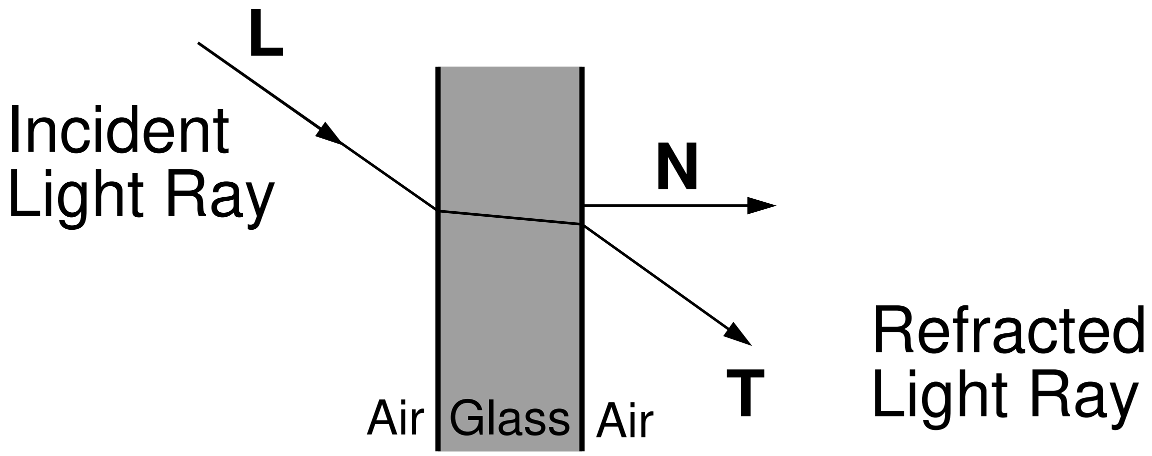


Snell's Law:

$$\eta_r \sin \Theta_r = \eta_i \sin \Theta_i$$

Transparency

- ♦ **Transmission through objects:**

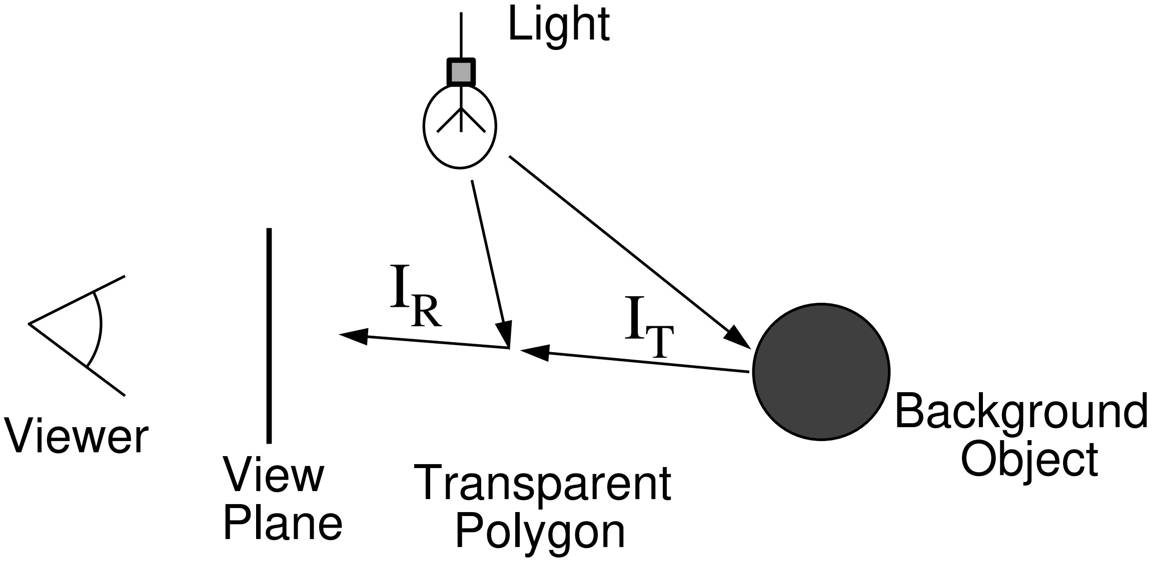


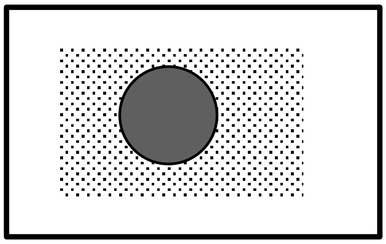
Note: incident and refracted light rays are parallel

$$\mathbf{T} = (\eta_i/\eta_r \cos \Theta_i, -\cos \Theta_r)\mathbf{N} - \eta_i/\eta_r\, \mathbf{L}$$

Transparency

- **Rendering transparent polygons**
  - Generally ignore refraction



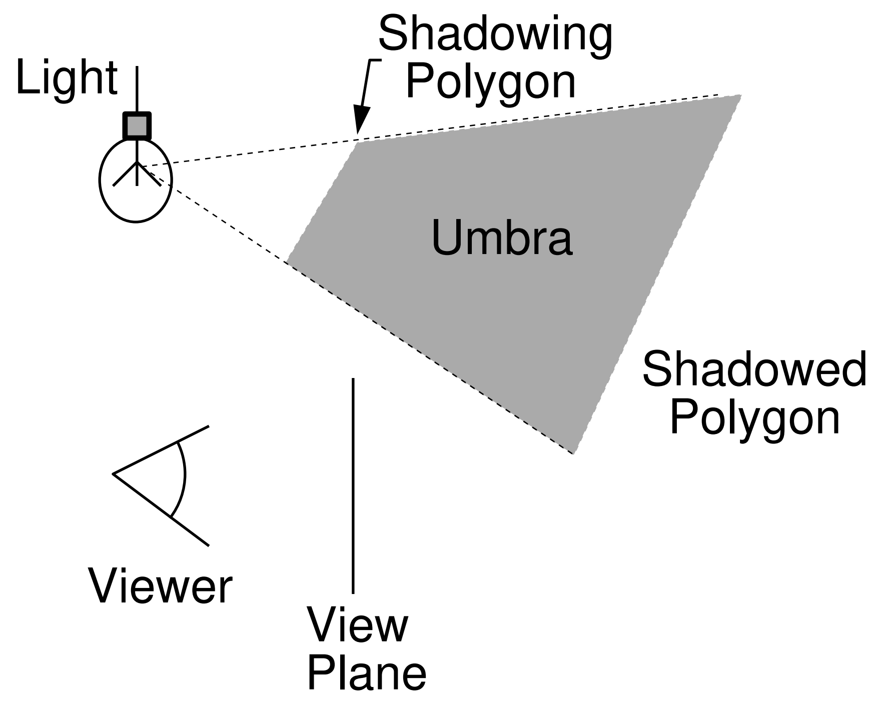


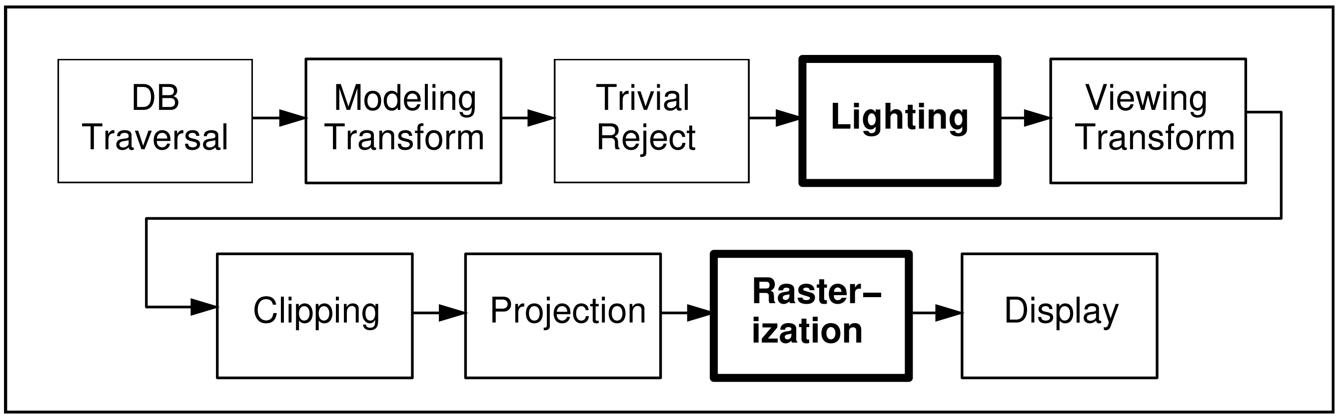
$$I = (1 - K_t)\, I_R + K_t\, I_T$$

- Kt is surface transparency coefficient
- Must render polygons in order sorted from back to front with depth sort, scan–line, A–Buffer, or BSP algorithm

Shadows

- **Goal**
  - Simulate shadows cast by one object onto another





Shadows

- **Direct illumination with shadows**
  - For each illumination computation, compute Si based on direct shadows:

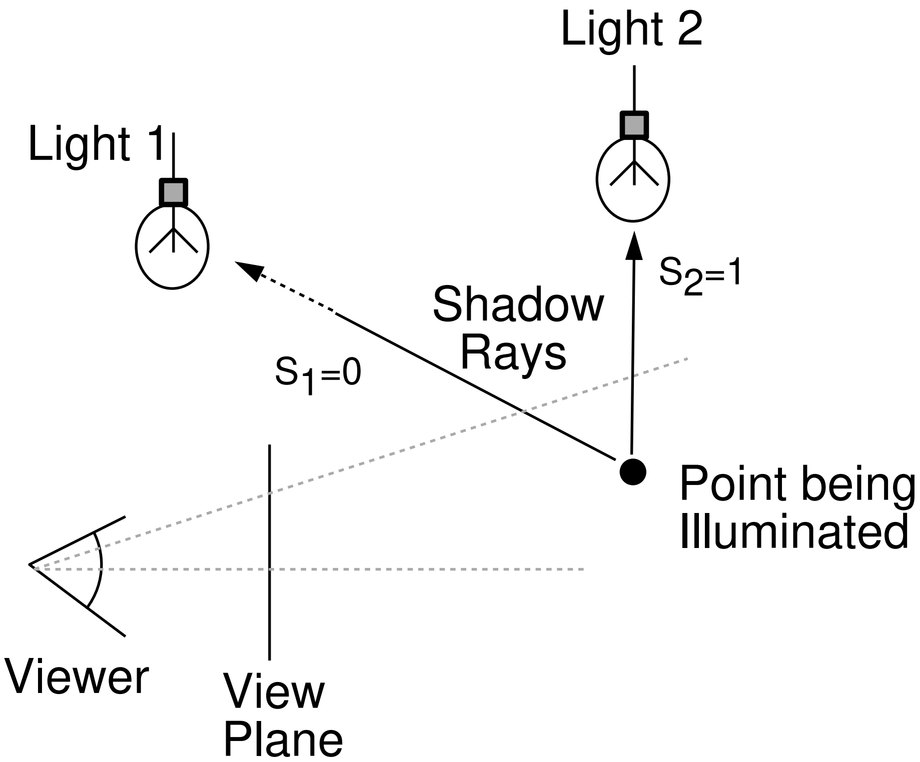
$$I = I_E + K_A I_{AL} + \sum_{L_i} \left(K_D (N \bullet L_i) + K_S (V \bullet R_i)^n \right) \mathbf{S_i} I_{L_i}$$

$$S_i = \begin{cases} 0, & \text{if light } i \text{ is blocked} \\ 1, & \text{if light } i \text{ is not blocked} \end{cases}$$

- **Shadow algorithms**
  - Ray casting
  - Light buffer algorithm
  - *Scan line algorithm*
  - *Shadow volumes*
  - *etc.*

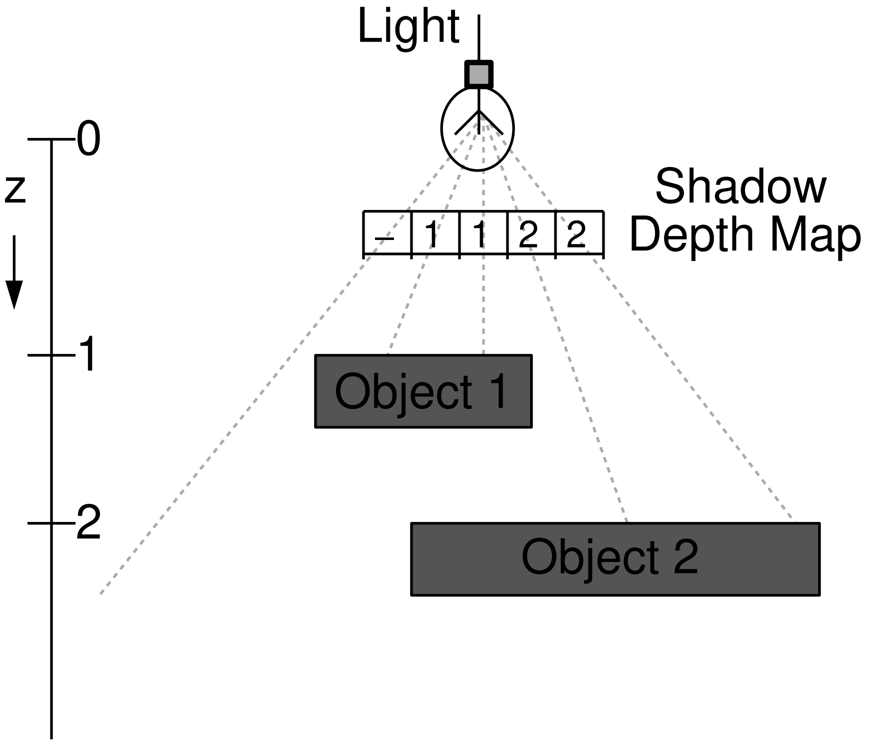
Shadow Algorithms

- **Ray Casting**
  - Cast ray to each light source
  - Check for intersections with other polygons to see if the light source is shadowed ($S_i = 0$) or not ($S_i = 1$)



Shadow Algorithms

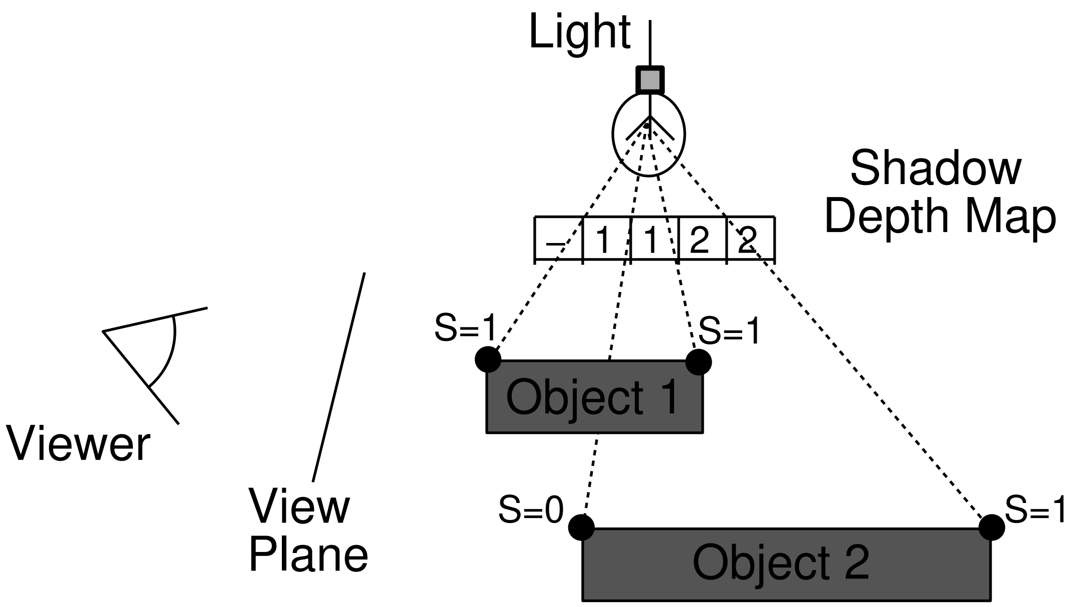
- **Shadow buffers**
  - At beginning, pre–render scene from viewpoint of light source and store ‘‘shadow depth map’’
    - SB(x,y) = minimum Z coord of any polygon projecting to (x,y) from light’s viewpoint



Shadow Algorithms

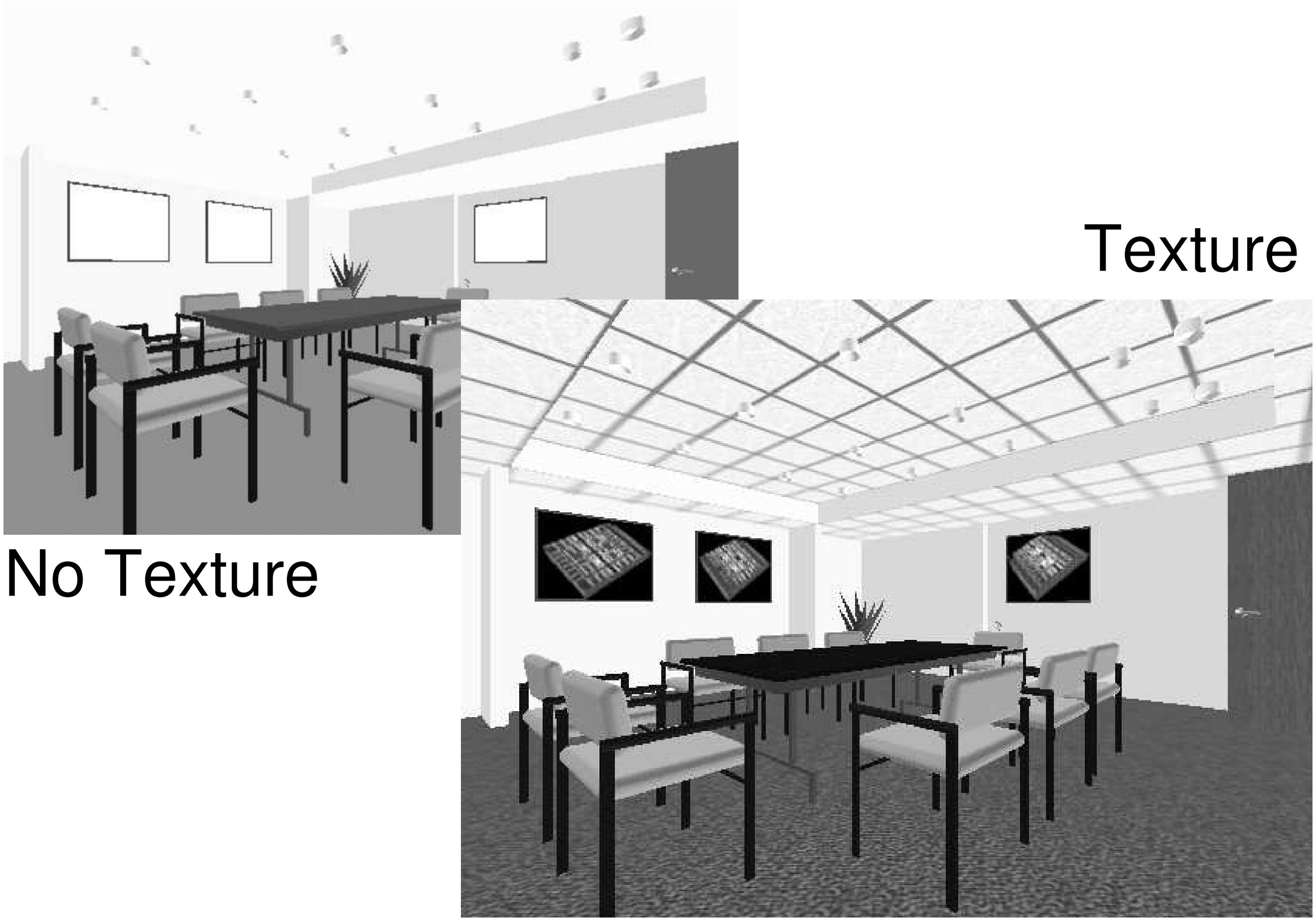
- **Shadow buffers (cont)**
  - When computing illumination for a point, transform (x,y,z) into (x',y',z') in the light's coordinate system.

$$S_i = \begin{cases} 1, \text{ if } & z' <= SB(x',y') \\ 0, \text{ if } & z' > SB(x',y') \end{cases}$$



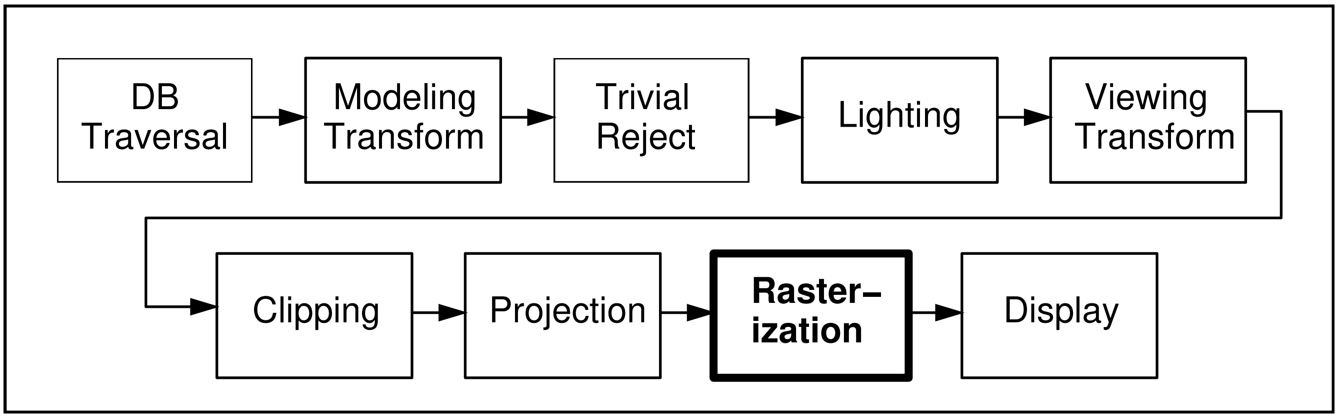
Texture Mapping

- **Goal:**
  - Apply surface detail to scene



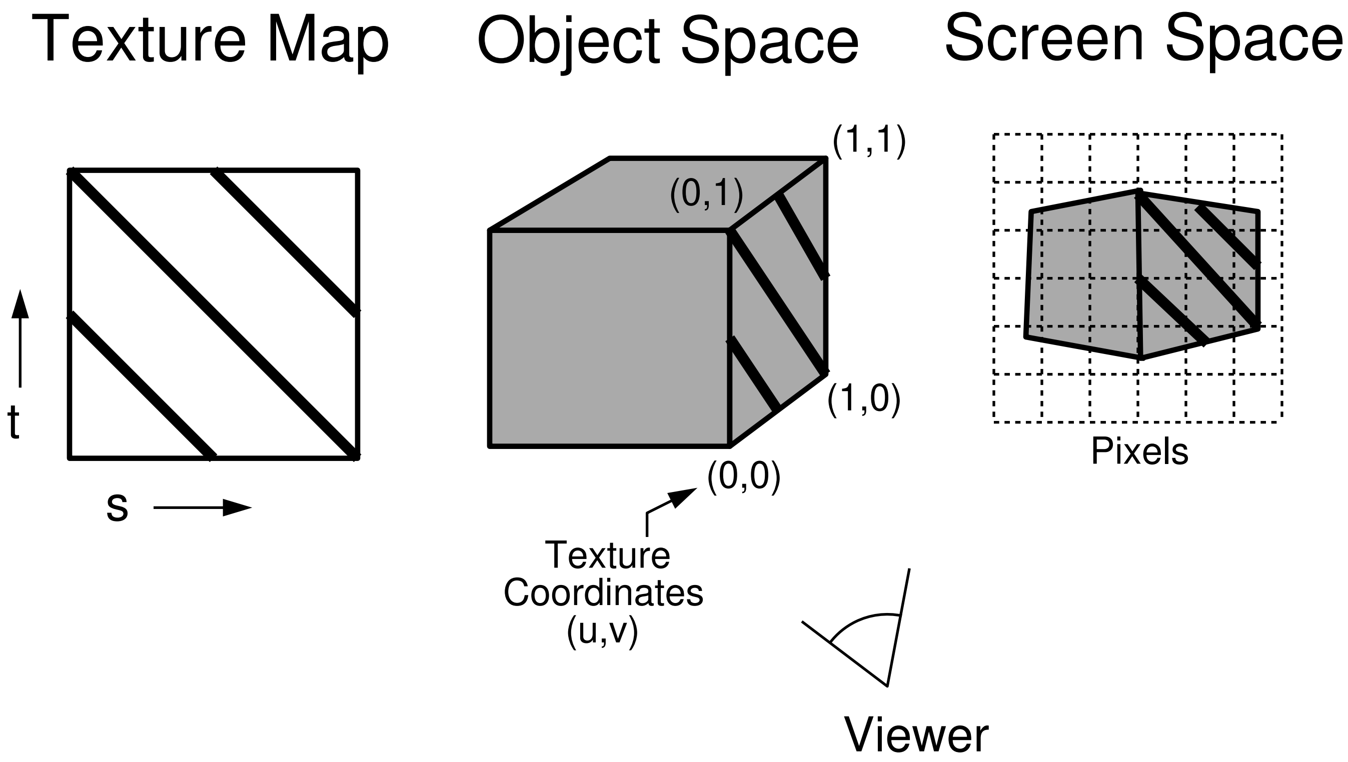
Texture

No Texture



Texture Mapping

- **Technique:**
  - Define texture map T(s,t)
  - Specify mapping from vertex texture coordinates (u,v) to texture map
  - When rendering polygon, light pixels based on values fetched from texture map



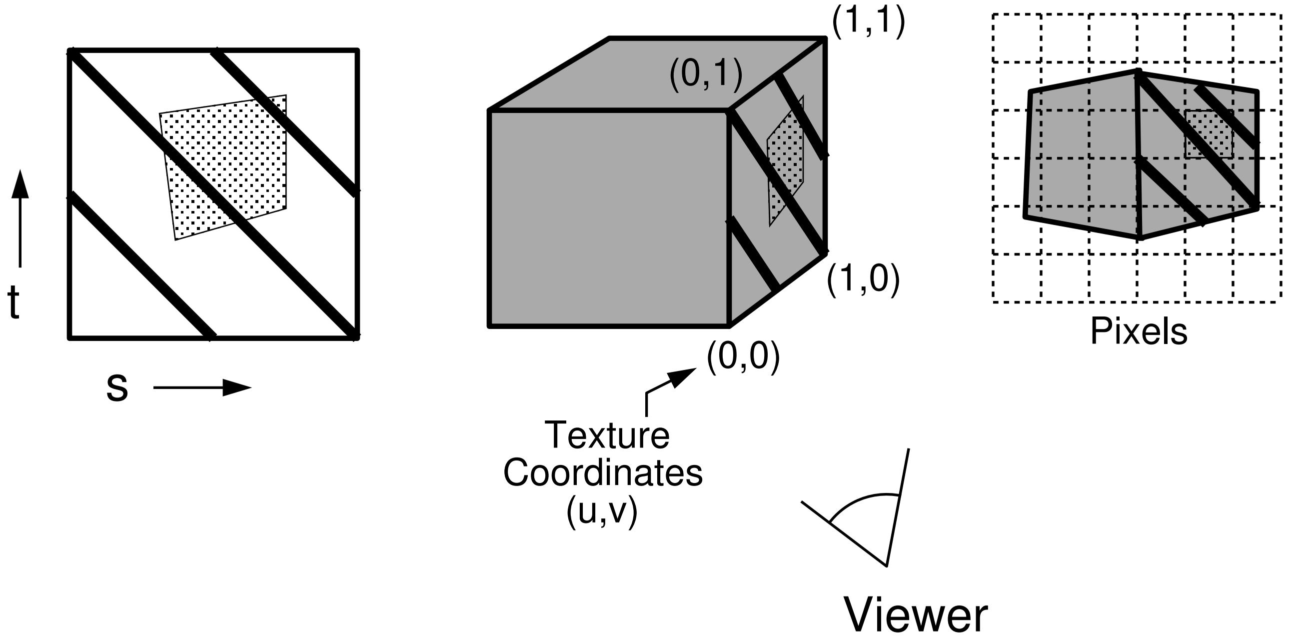
Texture Mapping

- **Coordinate Systems:**
  - Screen space (pixels)

    ↓ Inverse projection
  - Object space (world coordinates)

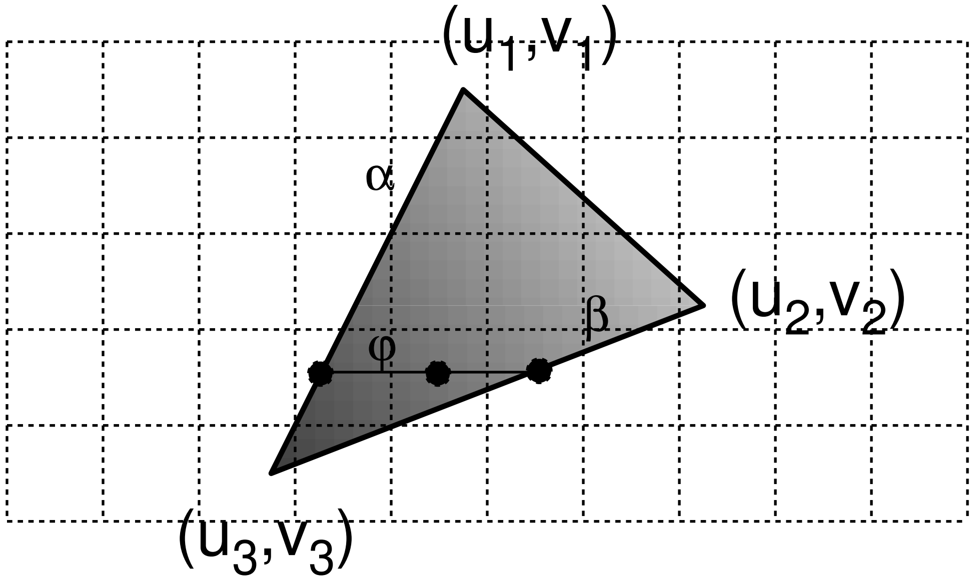
    ↓ Texture coordinate mapping
  - Texture map (texels)

Texture Map ← Object Space ← Screen Space



Texture Mapping

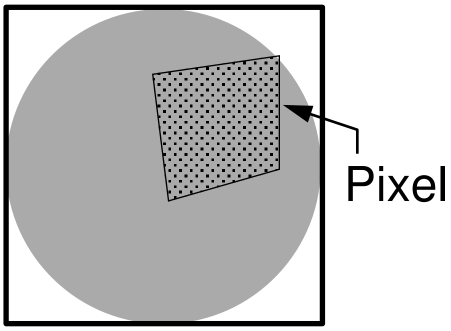
- **Scan conversion**
  - Interpolate texture coordinates down and across scan lines
  - Distortion due to bilinear interpolation approximation to perspective projection can be a problem
    - Cut polygons into smaller ones
    - Perspective divide at each pixel



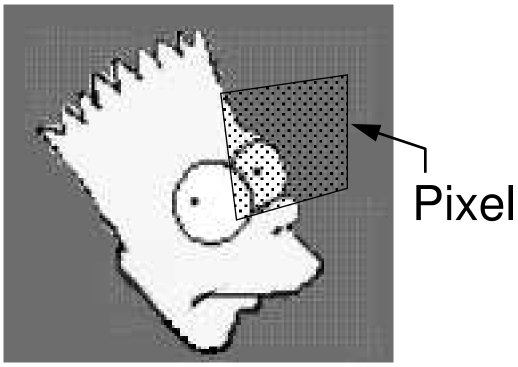
Texture Mapping

- **Procedural texture maps**

$$T(s,t) = \begin{cases} 0.5, \text{ if } (s-0.5)^2 + (t-0.5)^2 < 0.25 \\ 1, \text{ otherwise} \end{cases}$$



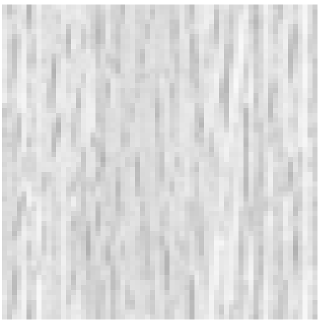
- **Image texture maps**

$I(x,y) =$ 

  – Filter image to compute texture value

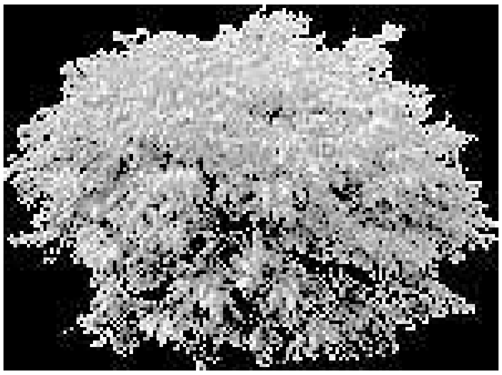
Texture Mapping

- **Modulation textures:**
  - Map texture values to intensity scale factor



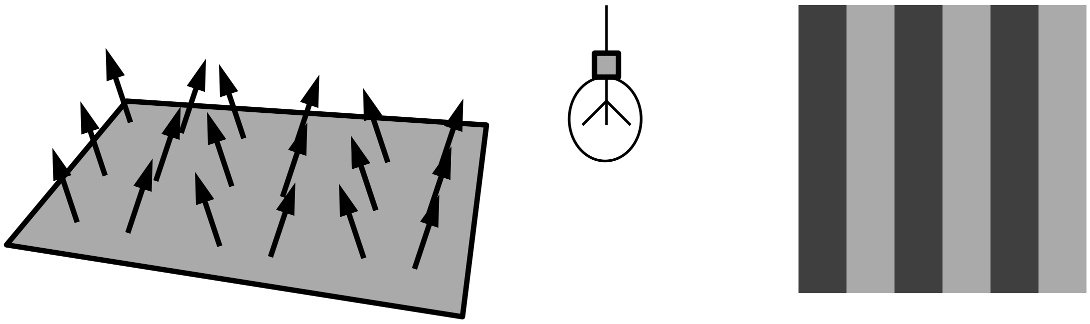
$$I = \mathbf{T(s,t)}\,(I_E + K_A I_A + \sum_{L_i} (K_D(N \bullet L_i) + K_S(V \bullet R_i)^n) S_i\, I_{L_i})$$

- **Illumination mapping:**
  - Map texture values to any lighting coeficient
  - e.g., $K_t = T(s,t)$ for shrub

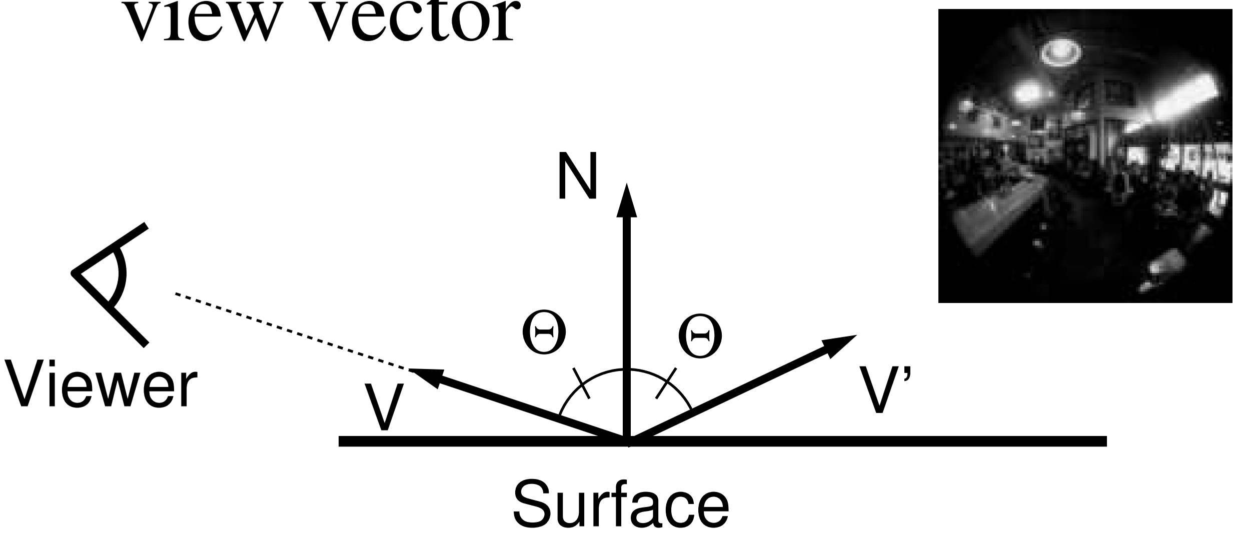


Texture Mapping

- **Bump mapping:**
  – Map texture values to perterbations of normal direction



- **Environment mapping:**
  – Compute (u,v) texture coordinates as polar coordinates of reflected view vector



Direct Illumination Summary

- Emission
- Diffuse reflection
- Specular reflection
- Ambient light
- Transparency
- Shadows
- Texture mapping

$$I = T(s,t)\,(I_E + K_A I_A + \sum_{L_i} (K_D(N \bullet L_i) + K_S(V \bullet R_i)^n) S_i\, I_{L_i})$$