BRDF Evolution

- BRDFs have evolved historically
- 1970's: Empirical models
  - Phong's illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990's
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000's
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000's
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)

Physically-Based Shading Models

- Phong model produces pretty pictures
- Cons: empirical (fudged?) ($cos^{\alpha}\phi$), plastic look
- Shaders can implement better lighting/shading models
- Big trend towards Physically-based lighting models
- Physically-based?
  - Based on physics of how light interacts with actual surface
  - Apply Optics/Physics theories
- Classic: Cook-Torrance shading model (TOGS 1982)

Cook-Torrance Shading Model

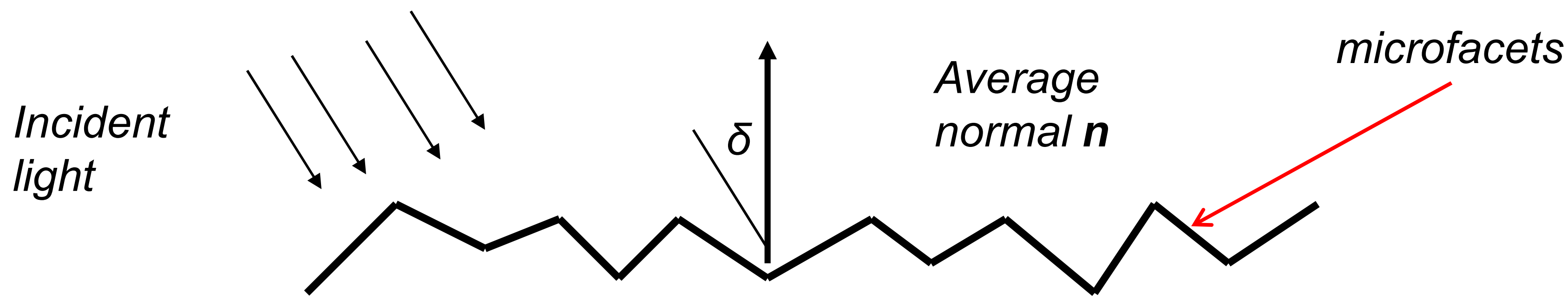
- Same ambient and diffuse terms as Phong
- New, better specular component than ($cos^{\alpha}\phi$),

$$\cos^{\alpha}\phi \rightarrow \frac{F(\phi,\eta)DG}{(\mathbf{n}\cdot\mathbf{v})}$$

- Where
  - D - Distribution term
  - G – Geometric term
  - F – Fresnel term

Distribution Term, D

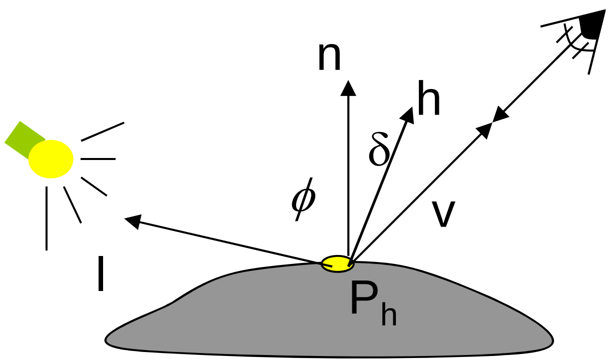
- **Idea:** surfaces consist of small V-shaped **microfacets (grooves)**

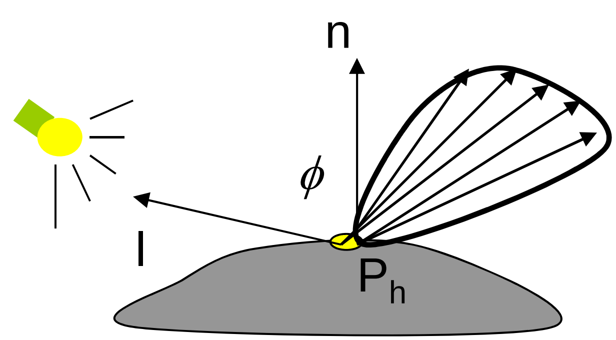


- Many grooves at each surface point
- Grooves facing a direction contribute
- D(δ) term: what fraction of grooves facing each angle δ
- E.g. half of grooves at hit point face 30 degrees, etc

Cook-Torrance Shading Model

- Define angle δ as deviation of **h** from surface normal
- Only microfacets with pointing along halfway vector, **h = s + v**, contributes





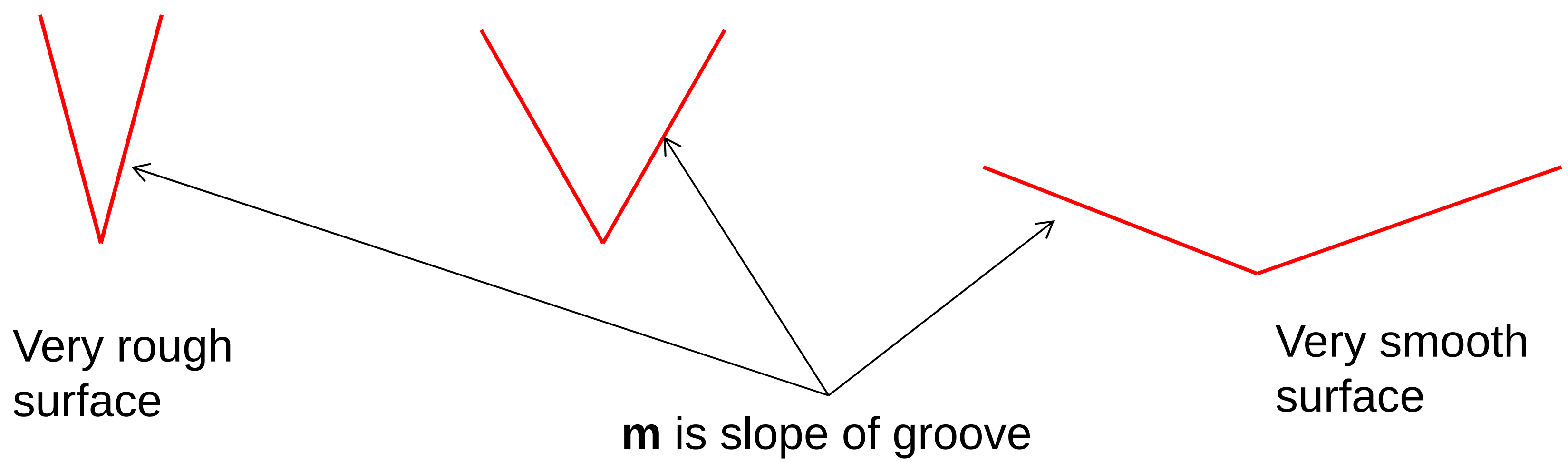
- Can use old Phong cosine ($cos^n \phi$), as D
- Use Beckmann distribution instead

$$D(\delta) = \frac{1}{4\mathbf{m}^2 \cos^4(\delta)} e^{-\left(\frac{\tan(\delta)}{\mathbf{m}}\right)^2}$$

- **m** expresses roughness of surface. How?

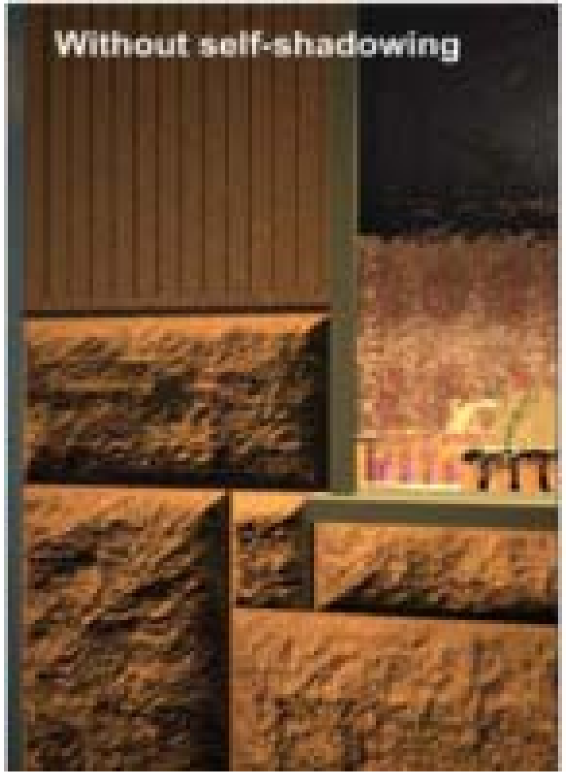
Cook-Torrance Shading Model

- **m** is Root-mean-square (RMS) of **slope** of V-groove
- m = 0.2 for nearly smooth
- m = 0.6 for very rough



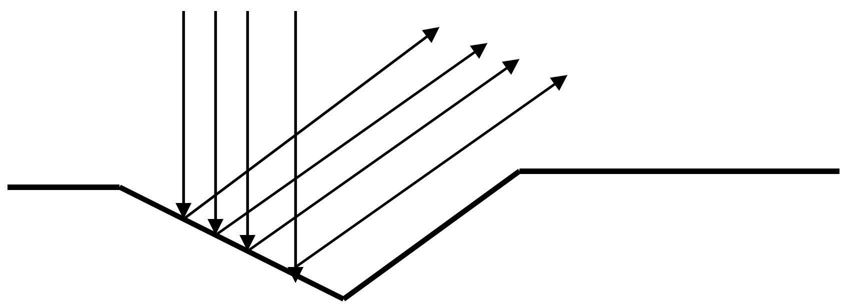
Self-Shadowing (G Term)

- Some grooves on extremely rough surface may block other grooves



Geometric Term, G

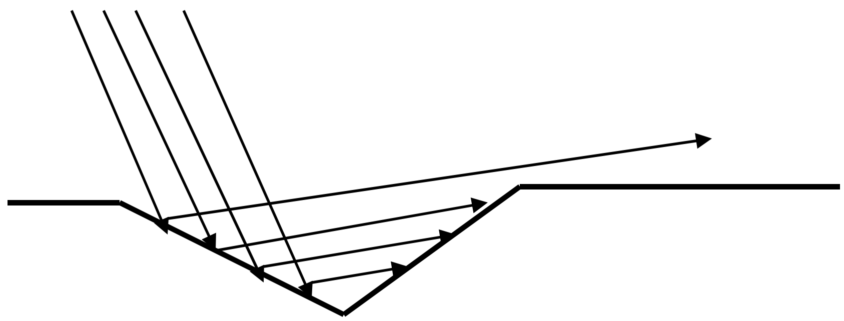
- Surface may be so rough that interior of grooves is blocked from light by edges
- Self blocking known as **shadowing** or **masking**
- Geometric term G accounts for this
- Break G into 3 cases:
- G, case a: No self-shadowing (light in-out unobstructed)



- Mathematically, G = 1

Geometric Term, G

- G_m, case b: No blocking on entry, blocking of exitting light **(masking)**



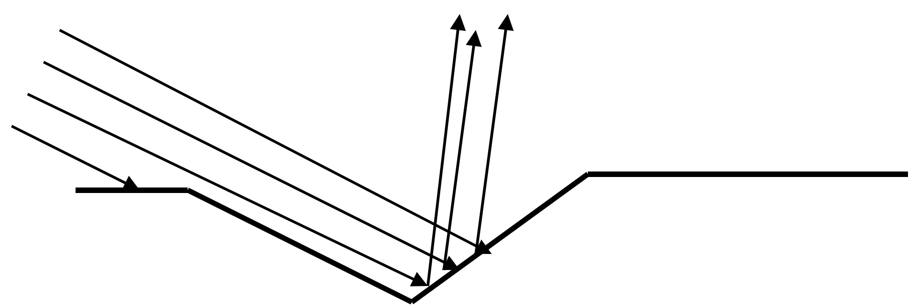
- Mathematically,

$$G_m = \frac{2(\mathbf{n} \cdot \mathbf{h})(\mathbf{n} \cdot \mathbf{h})}{\mathbf{h} \cdot \mathbf{s}}$$

Geometric Term, G

- G_s, case c: blocking of incident light, no blocking of exitting light (**shadowing)**
- Mathematically,

$$G_s = \frac{2(\mathbf{n}\cdot\mathbf{h})(\mathbf{n}\cdot\mathbf{h})}{\mathbf{h}\cdot\mathbf{s}}$$



- G term is minimum of 3 cases, hence

$$G = (1, G_m, G_s)$$

Fresnel Term, F

- So, again recall that specular term

$$spec = \frac{F(\phi,\eta)DG}{(\mathbf{n}\cdot\mathbf{v})}$$

- Microfacets not perfect mirrors
- F term, $F(\phi,\eta)$ gives fraction of incident light reflected

$$F = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left\{1+\left(\frac{c(g+c)-1}{c(g-c)-1}\right)^2\right\}$$

F is function of material and incident angle

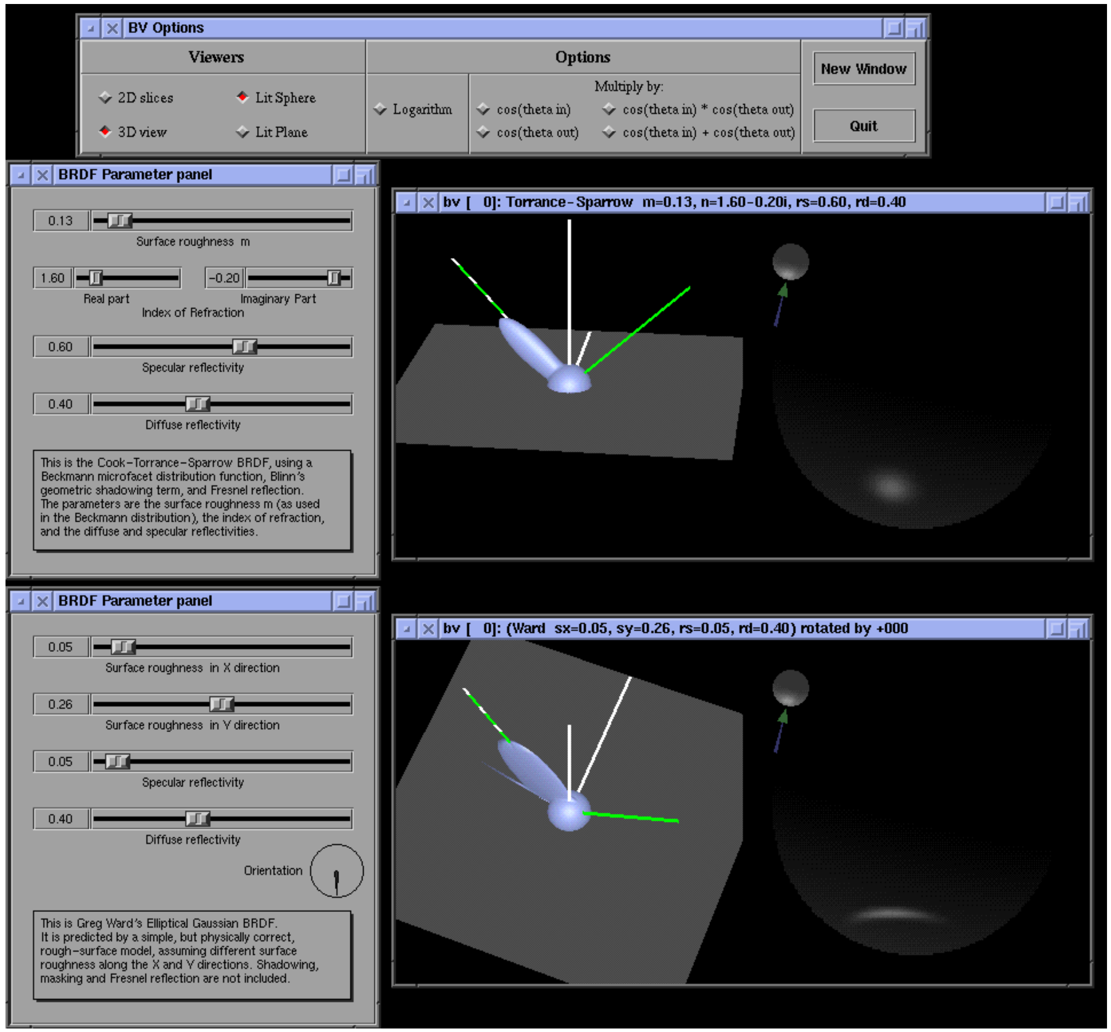
- where c = cos(ϕ) = **n.s** and $g^2 = \eta^2 + c^2 + 1$
- ϕ is incident angle, η is refractive index of material

Other Physically-Based BRDF Models

- Oren-Nayar – Diffuse term changed not specular
- Aishikhminn-Shirley – Grooves not v-shaped. Other Shapes
- Microfacet generator (Design your own microfacet)

BV BRDF Viewer

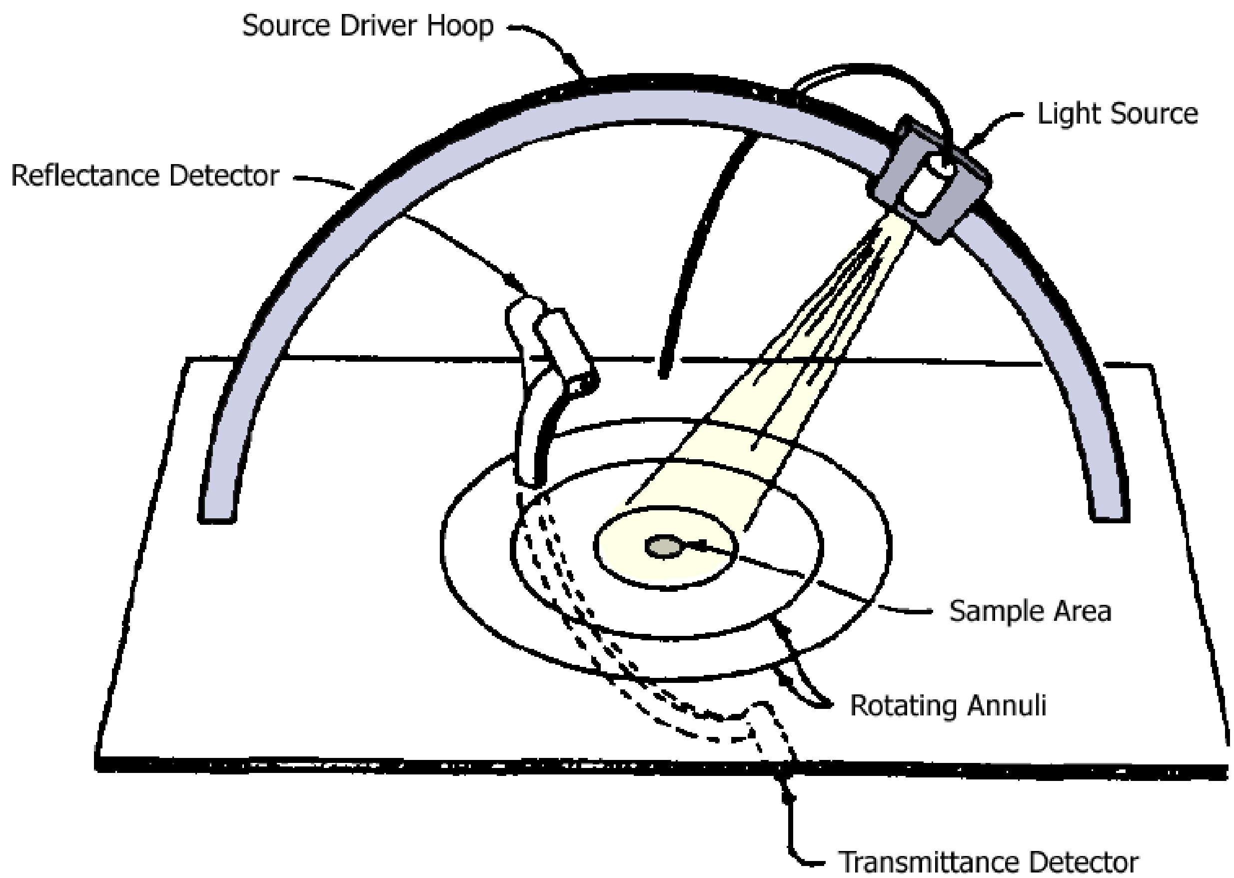
BRDF viewer (View distribution of light bounce)



BRDF Evolution

- BRDFs have evolved historically
- 1970's: Empirical models
  - Phong's illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990's
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000's
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000's
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)

Measuring BRDFs



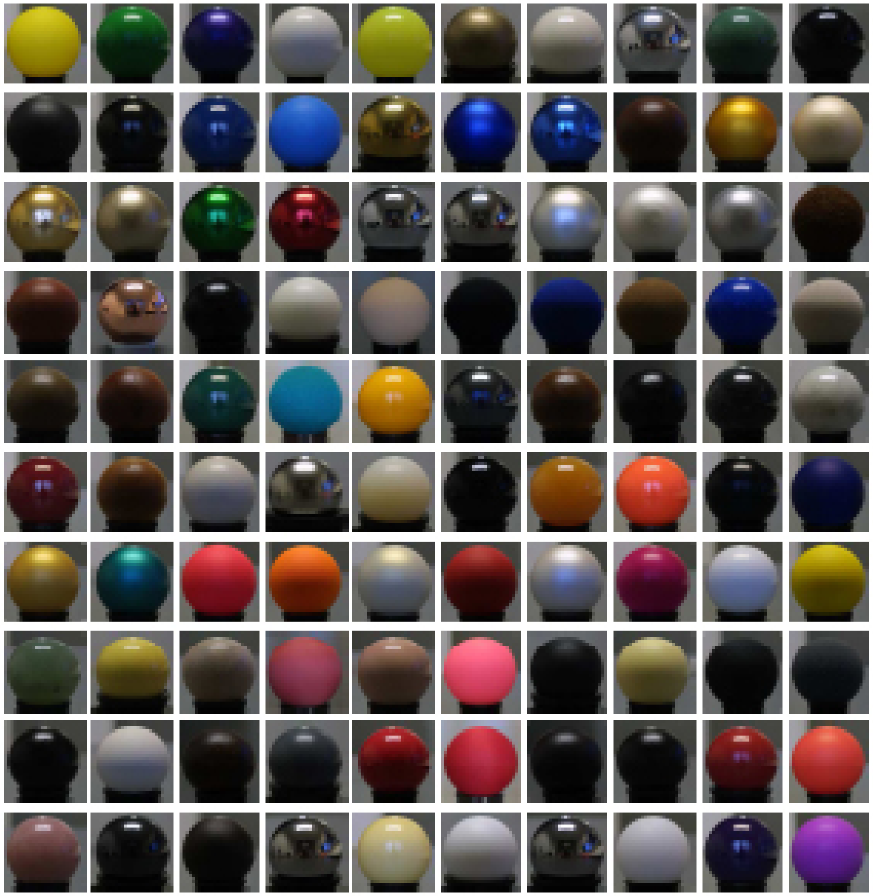
Murray-Coleman and Smith Gonioreflectometer. (Copied and Modified from [Ward92]).

Measured BRDF Samples

- Mitsubishi Electric Research Lab (MERL)

http://www.merl.com/brdf/

- Wojciech Matusik
- MIT PhD Thesis
- 100 Samples



BRDF Evolution

- BRDFs have evolved historically
- 1970's: Empirical models
  - Phong's illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990's
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000's
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000's
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)

Time-varying BRDF

- BRDF: How different materials reflect light
- Time varying?: how reflectance changes over time
- 

Computer Graphics (CS 4731)
Lecture 19: Shadows and Fog

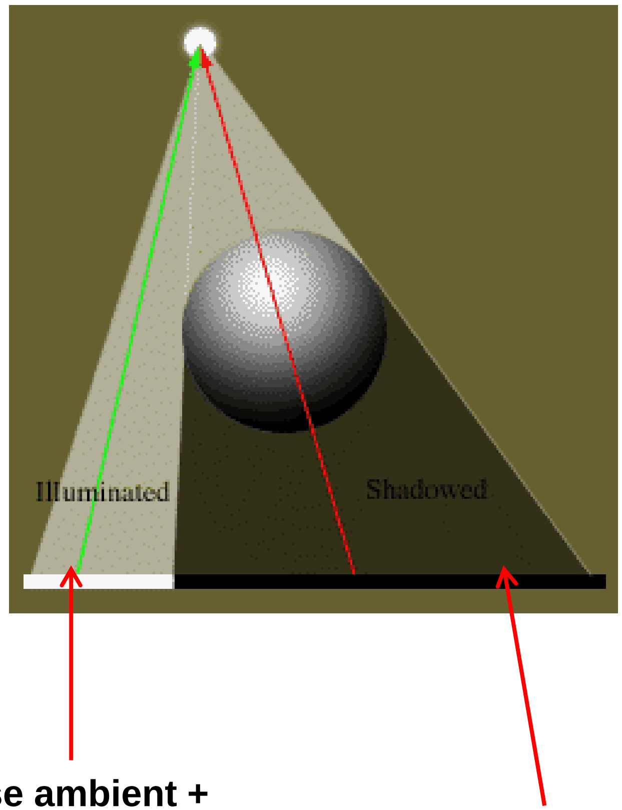
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

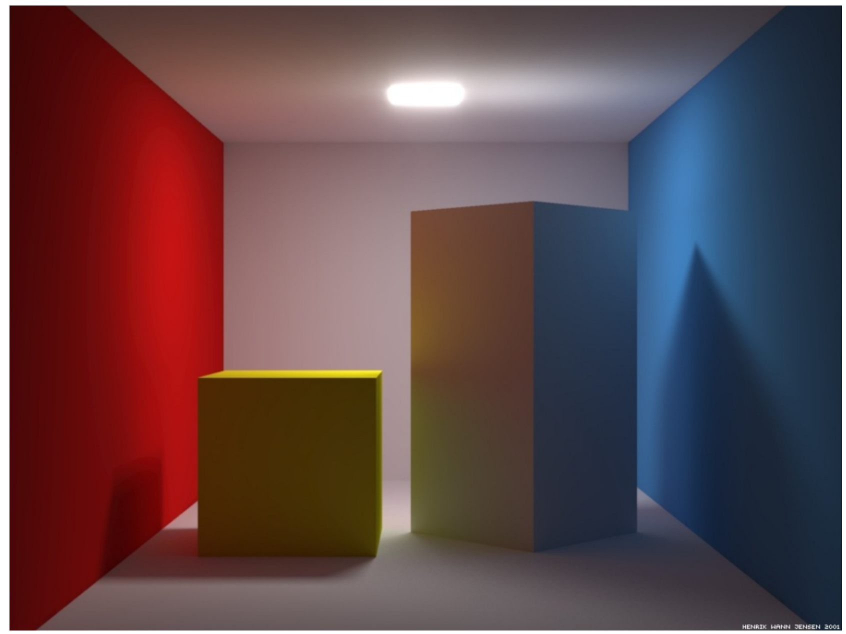
Introduction to Shadows

- Shadows give information on relative positions of objects



Use ambient + diffuse + specular components

Use just ambient component

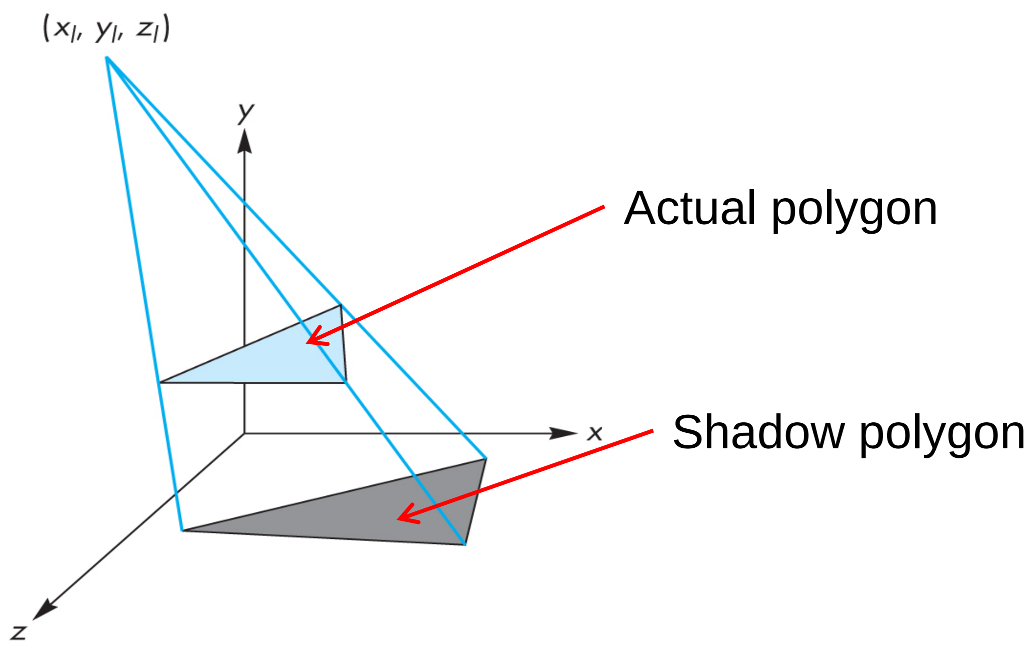


Introduction to Shadows

- Two popular shadow rendering methods:
  1. Shadows as texture (projection)
  2. Shadow buffer
- Third method used in ray-tracing (covered in grad class)

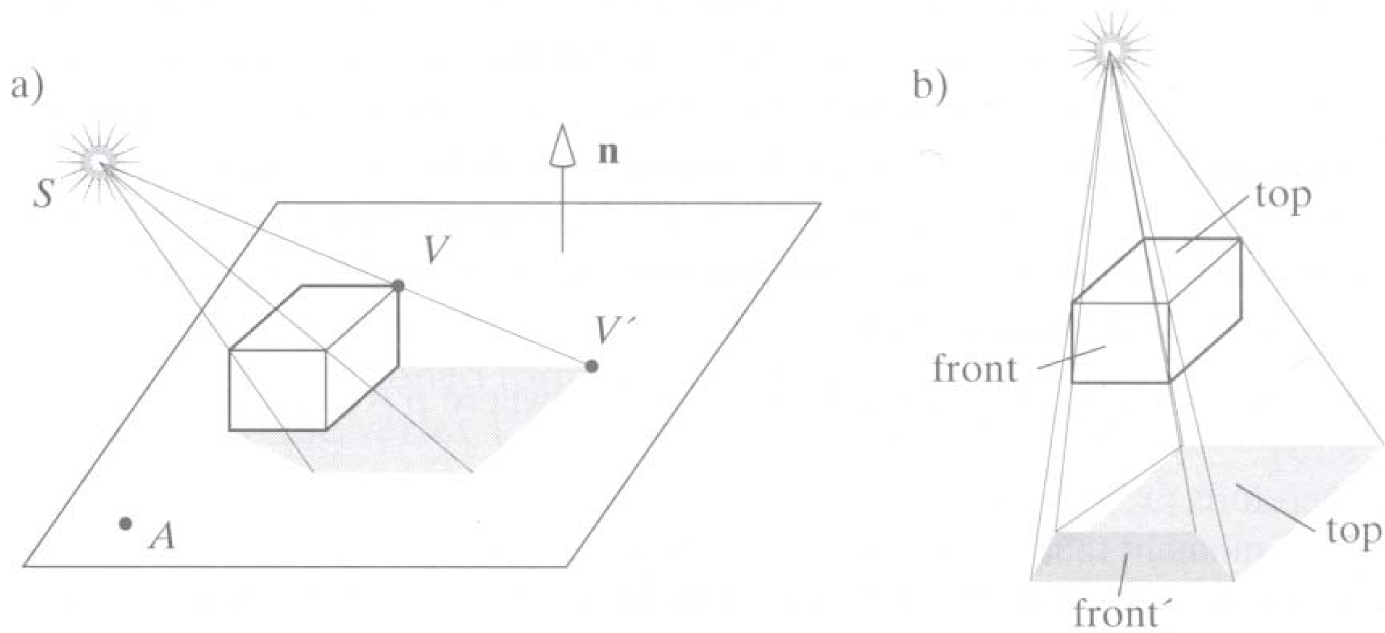
Projective Shadows

- Oldest method: Used in early flight simulators
- Projection of polygon is polygon called **shadow polygon**



Projective Shadows

- Works for flat surfaces illuminated by point light
- For each face, project vertices **V** to find **V’** of shadow polygon
- Object shadow = union of projections of faces

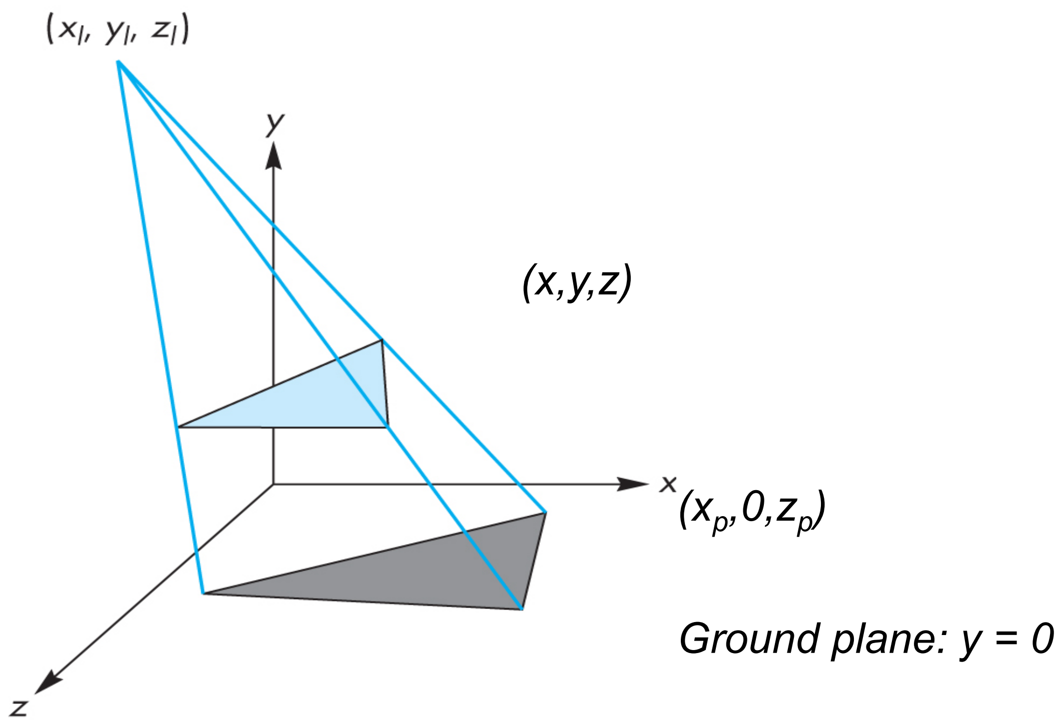


Projective Shadow Algorithm

- Project light-object edges onto plane
- Algorithm:
  - First, draw ground plane using specular+diffuse+ambient components
  - Then, draw shadow projections (face by face) using only ambient component

Projective Shadows for Polygon

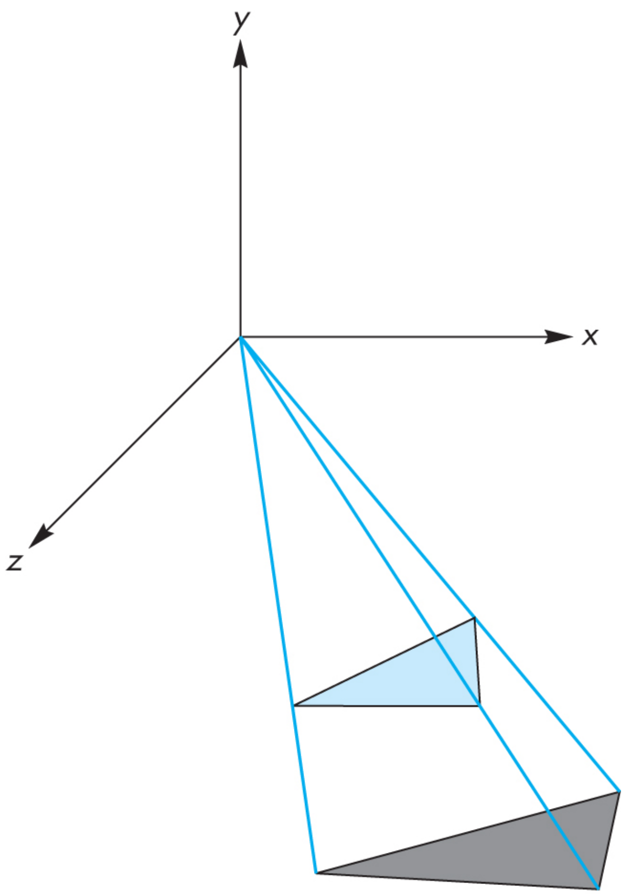
1. If light is at (x_l, y_l, z_l)
2. Vertex at (x, y, z)
3. Would like to calculate shadow polygon vertex V projected onto ground at $(x_p, 0, z_p)$



Projective Shadows for Polygon

- If we move original polygon so that light source is at origin
- Matrix *M* projects a vertex V to give its projection V' in shadow polygon

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{-y_l} & 0 & 0 \end{bmatrix}$$



Building Shadow Projection Matrix

1. Translate source to origin with $T(-x_l, -y_l, -z_l)$
2. Perspective projection
3. Translate back by $T(x_l, y_l, z_l)$

$$M = \begin{bmatrix} 1 & 0 & 0 & x_l \\ 0 & 1 & 0 & y_l \\ 0 & 0 & 1 & z_l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{-y_l} & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_l \\ 0 & 1 & 0 & -y_l \\ 0 & 0 & 1 & -z_l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Final matrix that projects
Vertex V onto V' in shadow polygon

Code snippets?

- Set up projection matrix in OpenGL application

```
float light[3];  // location of light
mat4 m;    // shadow projection matrix initially identity

M[3][1] = -1.0/light[1];
```

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{-y_l} & 0 & 0 \end{bmatrix}$$

Projective Shadow Code

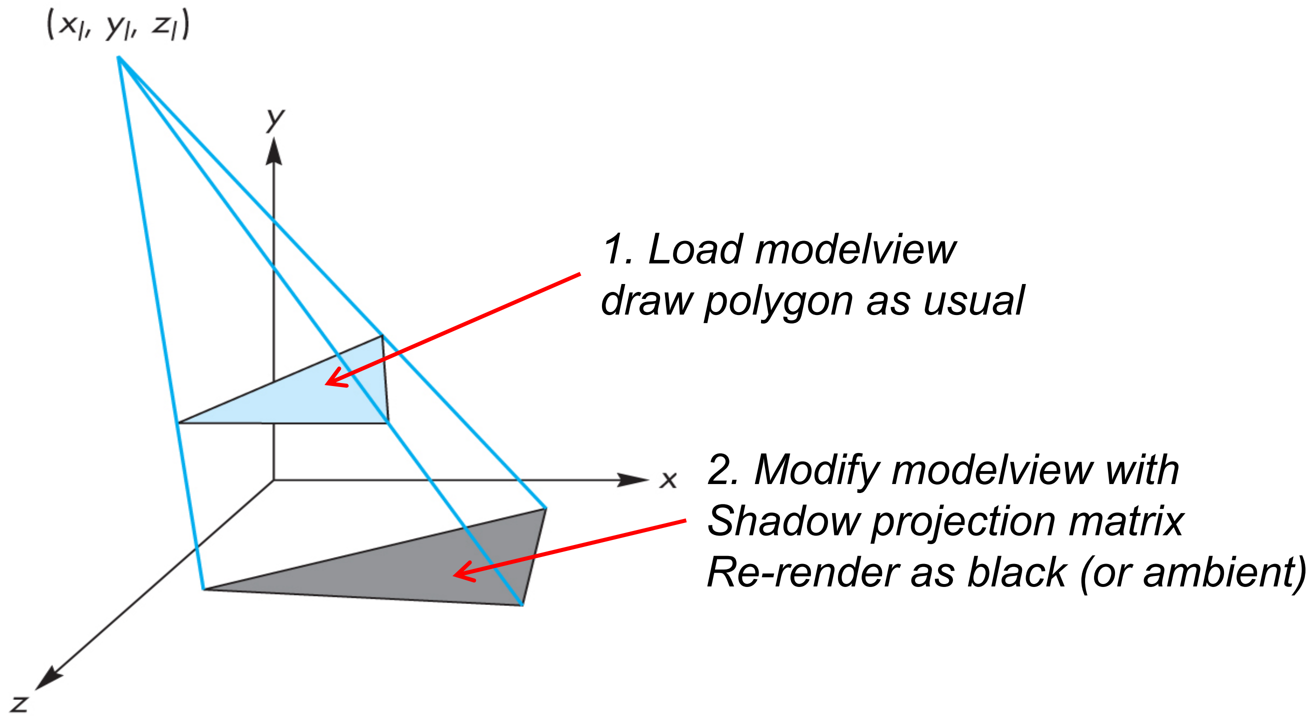
- Set up object (e.g a square) to be drawn

```
point4 square[4] = {vec4(-0.5, 0.5, -0.5, 1.0}
                   {vec4(-0.5, 0.5, -0.5, 1.0}
                   {vec4(-0.5, 0.5, -0.5, 1.0}
                   {vec4(-0.5, 0.5, -0.5, 1.0}
```

- Copy square to VBO
- Pass modelview, projection matrices to vertex shader

What next?

- Next, we load model_view as usual then draw original polygon
- Then load shadow projection matrix, change color to black, re-render polygon



Shadow projection Display( ) Function

```
void display( )
{
    mat4 mm;
    // clear the window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // render red square (original square) using modelview
    // matrix as usual (previously set up)
    glUniform4fv(color_loc, 1, red);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

Shadow projection Display( ) Function

```
// modify modelview matrix to project square
// and send modified model_view matrix to shader
mm = model_view
     * Translate(light[0], light[1], light[2]
     *m
     * Translate(-light[0], -light[1], -light[2]);
glUniformMatrix4fv(matrix_loc, 1, GL_TRUE, mm);

//and re-render square as
// black square (or using only ambient component)
 glUniform4fv(color_loc, 1, black);
 glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
 glutSwapBuffers( );
}
```

$$M = \begin{bmatrix} 1 & 0 & 0 & x_l \\ 0 & 1 & 0 & y_l \\ 0 & 0 & 1 & z_l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{-y_l} & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_l \\ 0 & 1 & 0 & -y_l \\ 0 & 0 & 1 & -z_l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

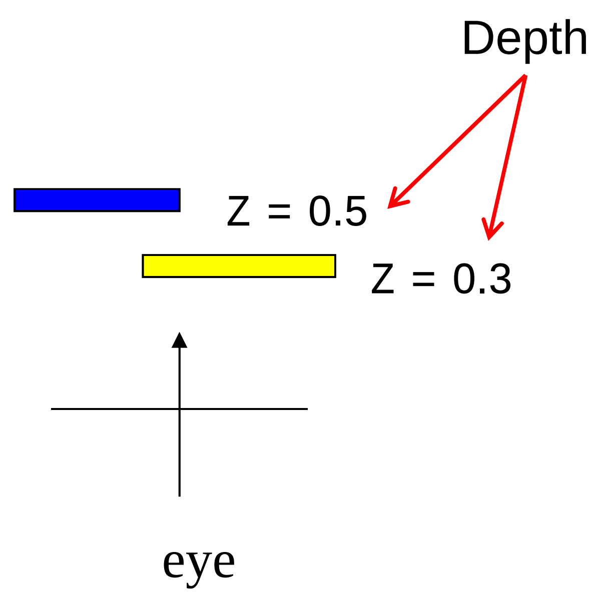
Shadow Buffer Approach

- Uses second **depth buffer** called shadow buffer
- Pros: not limited to plane surfaces
- Cons: needs lots of memory
- Depth buffer?

OpenGL Depth Buffer (Z Buffer)

- **Depth:** While drawing objects, depth buffer stores distance of each polygon from viewer
- **Why?** If multiple polygons overlap a pixel, only closest one polygon is drawn

1.0	1.0	1.0	1.0
1.0	0.3	0.3	1.0
0.5	0.3	0.3	1.0
0.5	0.5	1.0	1.0



Setting up OpenGL Depth Buffer

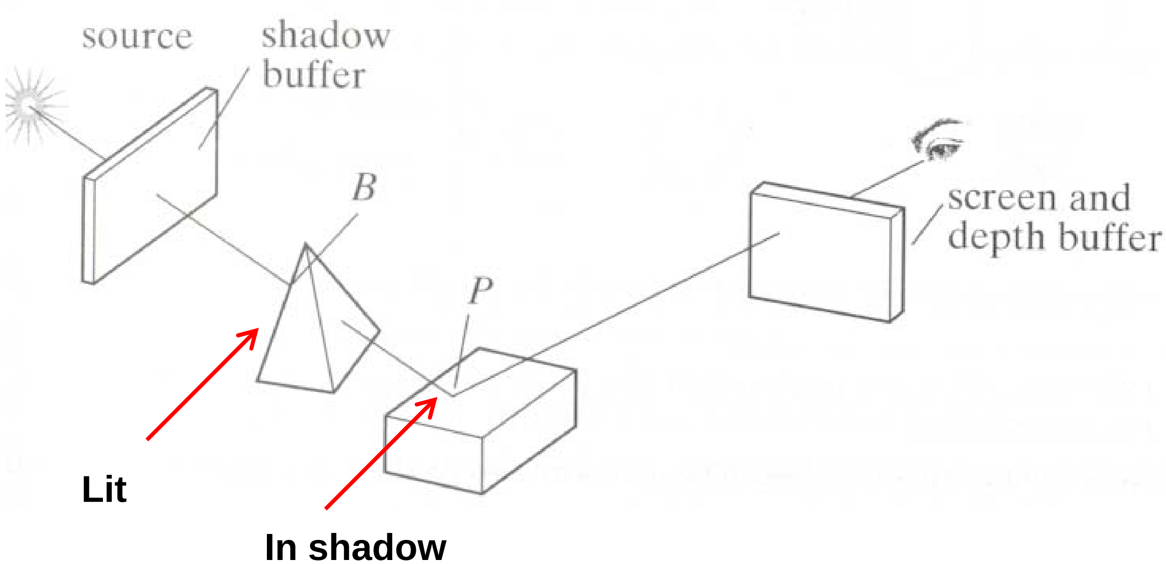
- **Note:** You did this in order to draw solid cube, meshes

1. `glutInitDisplayMode(GLUT_DEPTH | GLUT_RGB)` instructs openGL to create depth buffer
2. `glEnable(GL_DEPTH_TEST)` enables depth testing
3. `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`

   Initializes depth buffer every time we draw a new picture

Shadow Buffer Theory

- Along each path from light
  - Only closest object is lit
  - Other objects on that path in shadow
- Shadow buffer stores closest object on each path

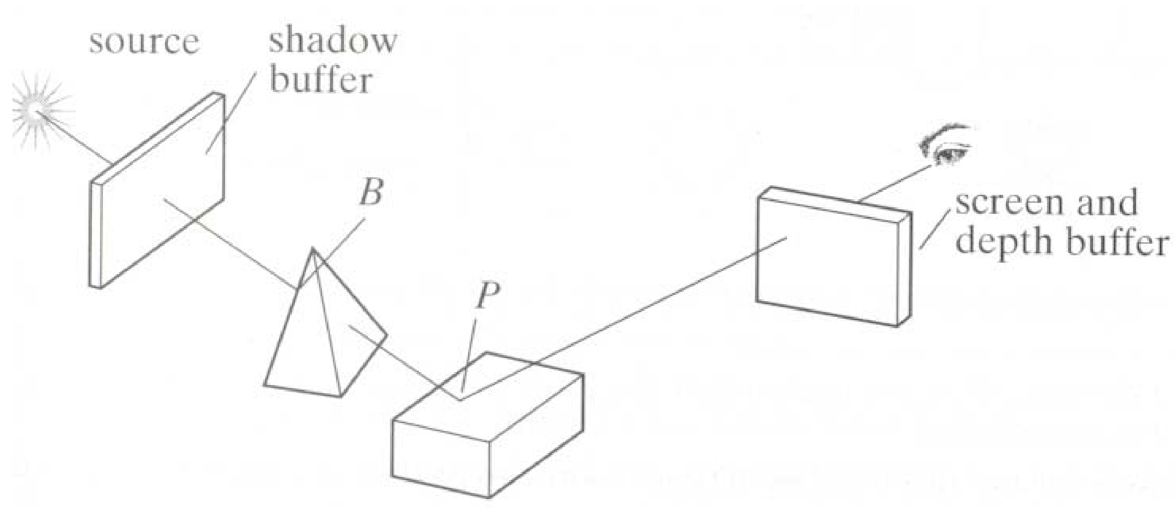


Shadow Buffer Approach

- Rendering in two stages:
  - Loading shadow buffer
  - Render the scene

Loading Shadow Buffer

- Initialize each element to 1.0
- Position a camera at light source
- Rasterize each face in scene updating closest object
- Shadow buffer tracks smallest depth on each path

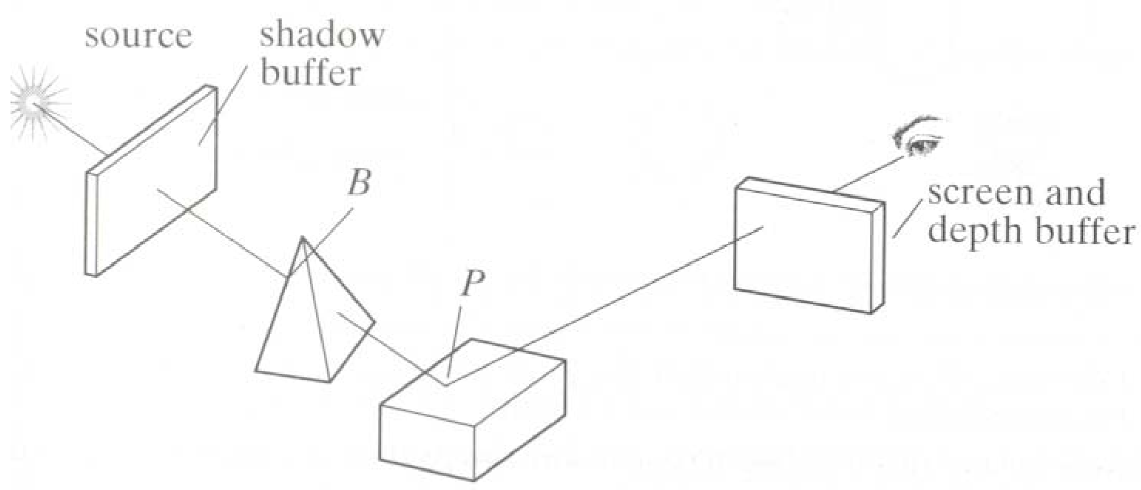


Shadow Buffer (Rendering Scene)

- Render scene using camera as usual
- While rendering a pixel find:
  - pseudo-depth D from light source to P
  - Index location [i][j] in shadow buffer, to be tested
  - Value d[i][j] stored in shadow buffer
- If d[i][j] < D (other object on this path closer to light)
  - point P is in shadow
  - set lighting using only ambient
- Otherwise, not in shadow

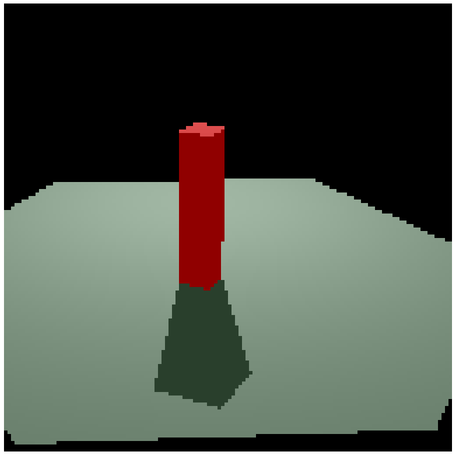
Loading Shadow Buffer

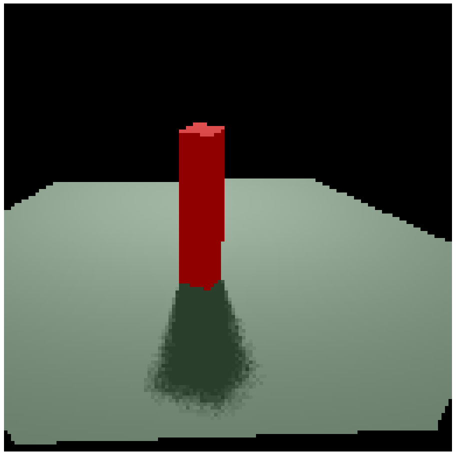
- Shadow buffer calculation is independent of eye position
- In animations, shadow buffer loaded once
- If eye moves, no need for recalculation
- If objects move, recalculation required



Other Issues

- Point light sources => simple hard shadows, unrealistic
- Extended light sources => more realistic
- Shadow has two parts:
  - Umbra (Inner part) => no light
  - Penumbra (outer part) => some light





References

- Interactive Computer Graphics (6th edition), Angel and Shreiner
- Computer Graphics using OpenGL (3rd edition), Hill and Kelley
- Real Time Rendering by Akenine-Moller, Haines and Hoffman