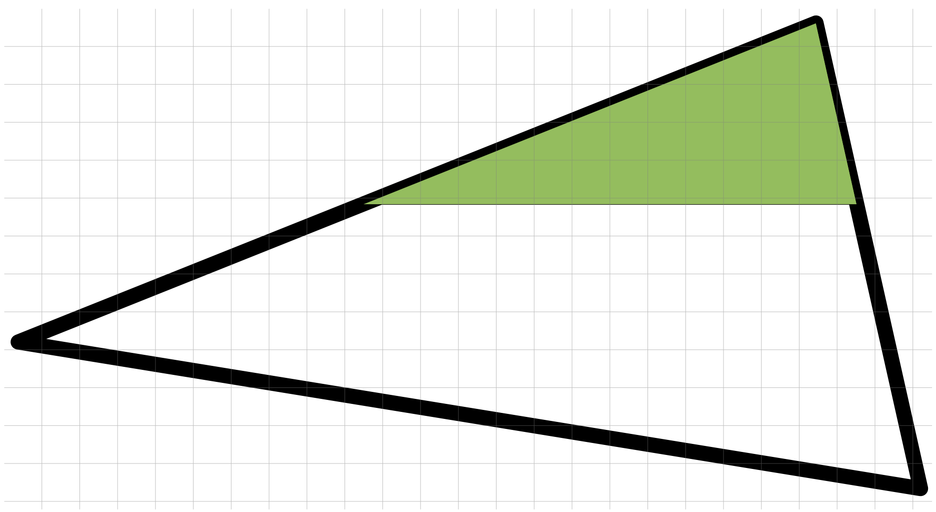
CS475/CS675
Computer Graphics

Shading

Shading

- Assigning colour to pixels or fragments.
- Modelling Illumination
- We shall see how it is done in a rasterization model.



Shading

- Illumination Model : The Phong Model
  - For a single light source total illumination at any point is given by:

$$I = k_a I_a + k_d I_d + k_s I_s$$

where

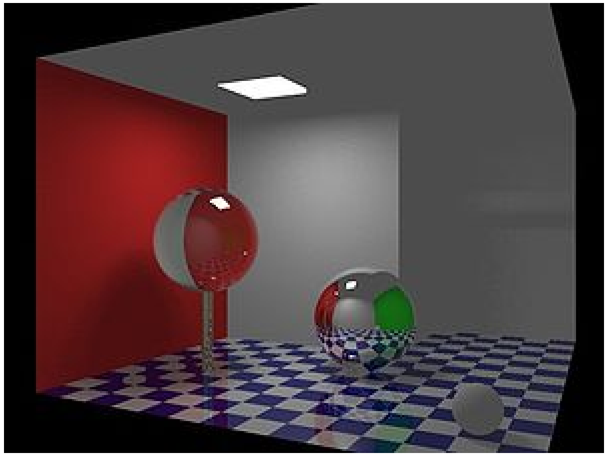
$k_a I_a$ is the contribution due to ambient reflection

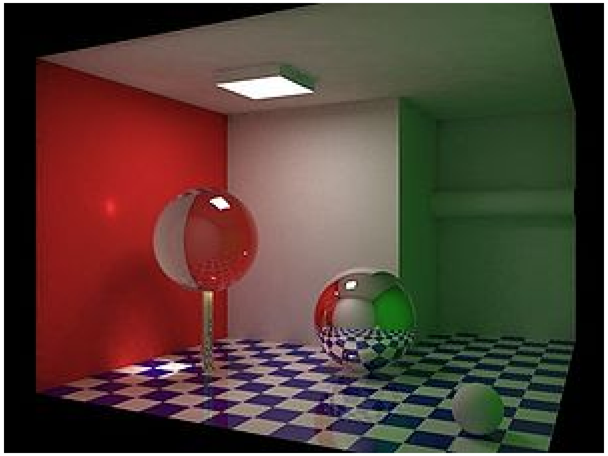
$k_d I_d$ is the contribution due to diffuse reflection

$k_s I_s$ is the contribution due to specular reflection

Shading

- Components of the Phong Model
- Ambient Illumination: I_a
  - Represents the reflection of all indirect illumination.
  - Has the same value everywhere.
  - Is an approximation to computing *Global Illumination*.





From http://en.wikipedia.org/wiki/Global_illumination (14/08/2009)

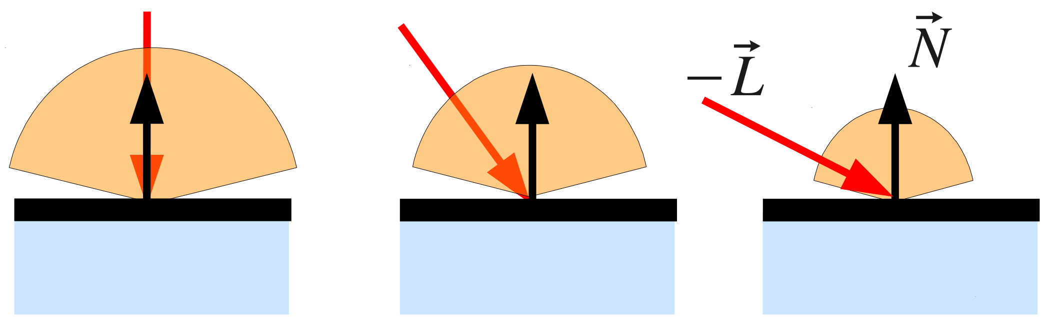
Shading

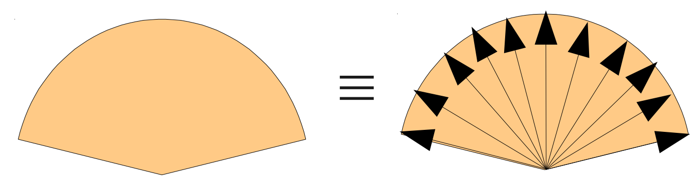
- Components of the Phong Model
- Diffuse Illumination: $I_d = I_L \cos\theta_L$
  - Assumes Ideal Diffuse Surface – that reflects light equally in all direction.
  - Surface is very rough at microscopic level. For e.g., Chalk and Clay.

Shading

- Components of the Phong Model
- Diffuse Illumination: $I_d = I_L \cos\theta_L$
  - Reflects light according to *Lambert's Cosine Law*

$$I_d = I_L \cos\theta_L$$
$$= I_L(\vec{L}.\vec{N})$$





$\vec{L}$: vector to the light source

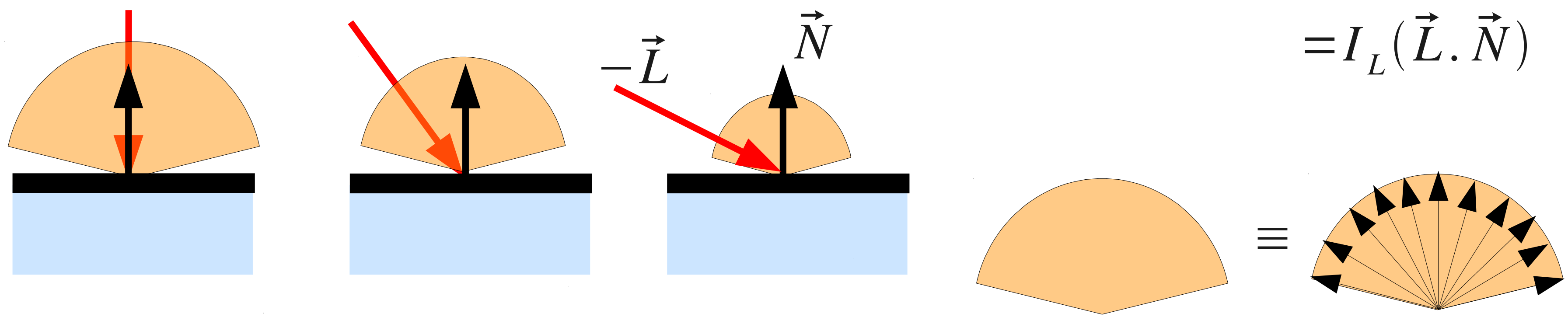
I_L : intensity of the light source

$\vec{N}$: surface normal

Shading

- Components of the Phong Model
- Diffuse Illumination: $I_d = I_L \cos\theta_L$
  - Reflects light according to *Lambert's Cosine Law*

$$I_d = I_L \cos\theta_L = I_L(\vec{L}.\vec{N})$$

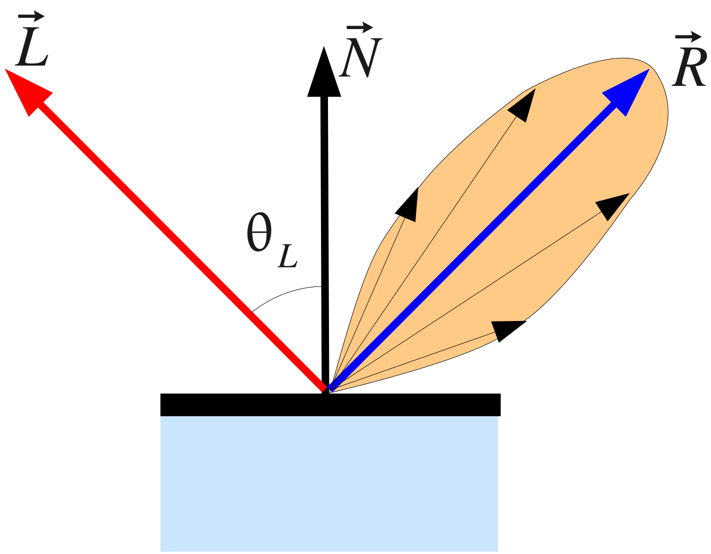


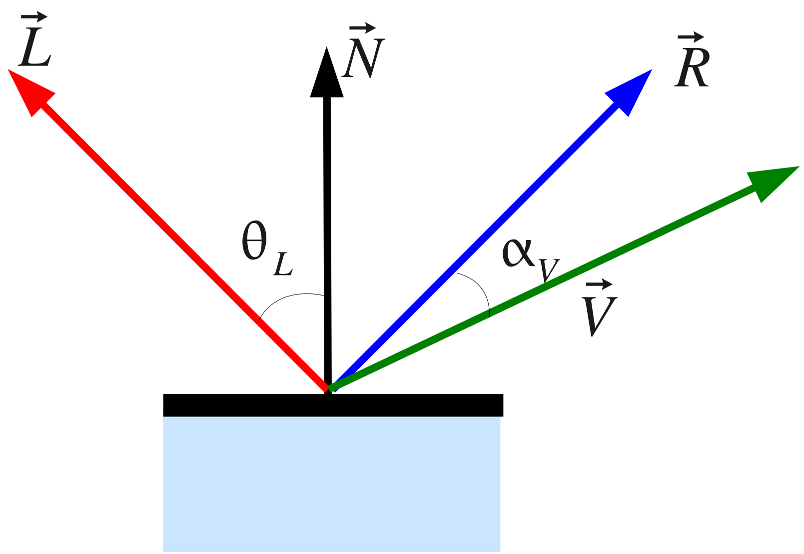
If $\vec{L}$ and $\vec{N}$ are in opposite directions then the dot product is negative. Use $max(\vec{L}.\vec{N}, 0)$ to get the correct value.

If r is distance to the light source and I_t is its true intensity then a distance based attenuation can be modelled by an inverse square falloff, i.e., $I_L = I_t / r^2$

Shading

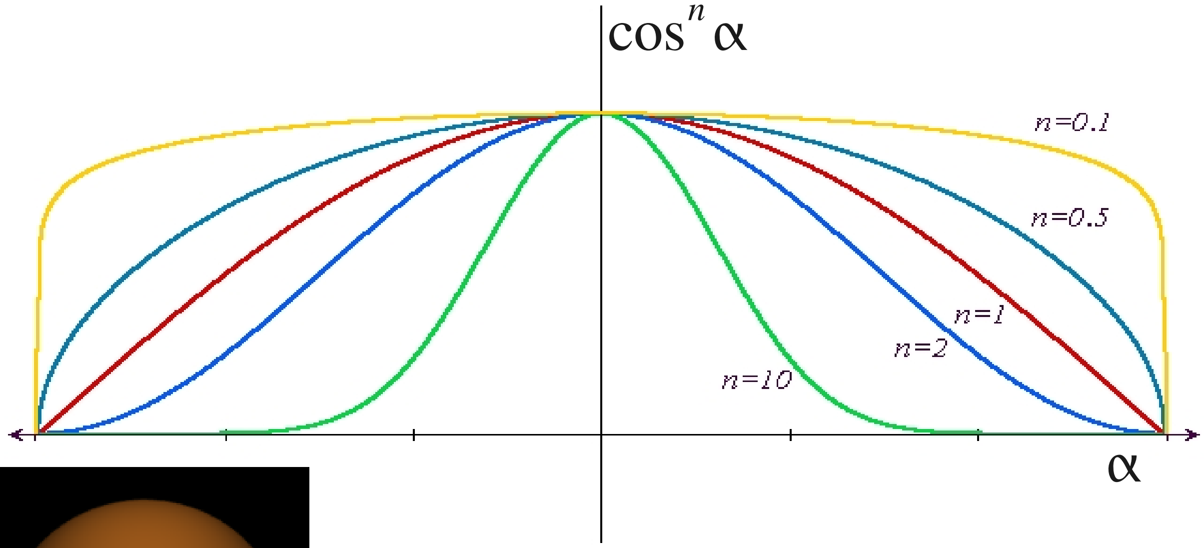
- Components of the Phong Model
- Specular Illumination: $I_s = I_L \cos^n \alpha_v = I_L (\vec{R} . \vec{V})^n$
  - Ideal specular surface reflects only along one direction.
  - Reflected intensity is view dependent – Mostly it is along the reflected ray but as we move away some of the reflection is slightly offset from the reflected ray due to microscopic surface irregularites.

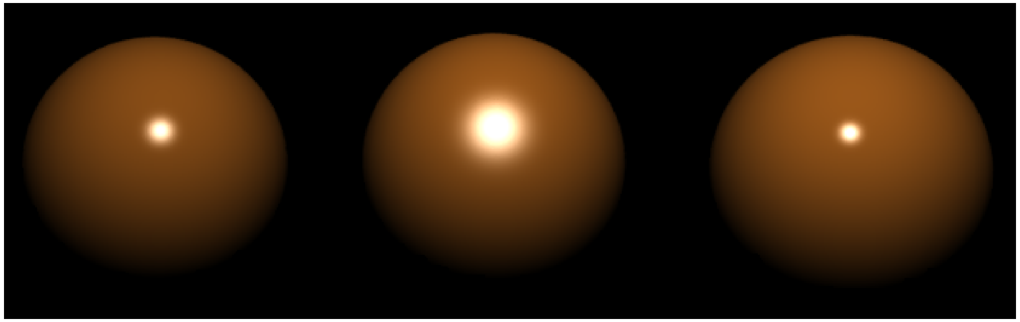




Shading

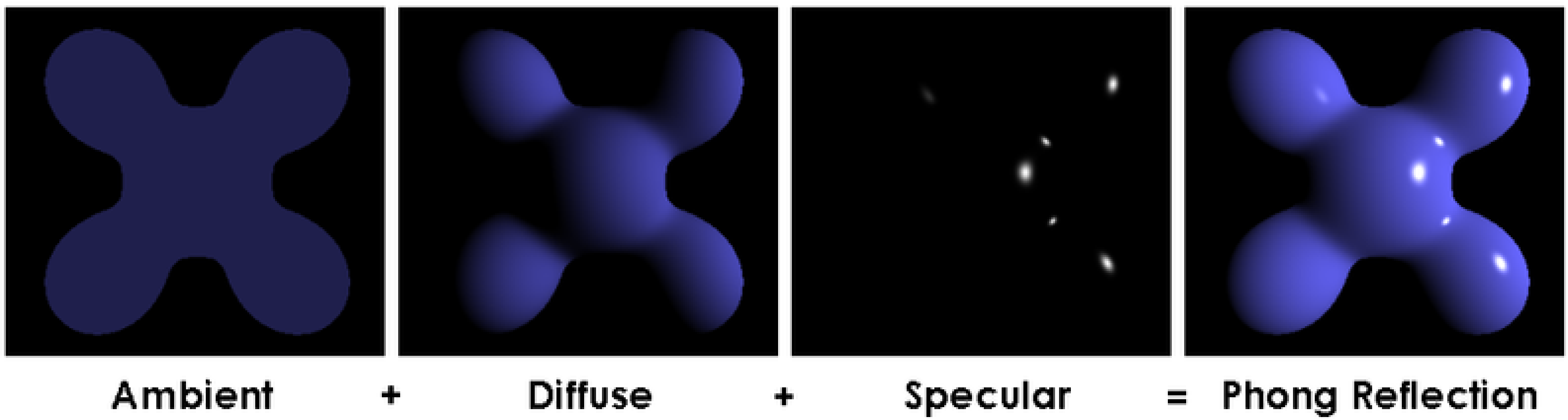
- Components of the Phong Model
- Specular Illumination: $I_s = I_L \cos^n \alpha_v = I_L (\vec{R} . \vec{V})^n$
  - n is called the coefficient of shininess and $I_L = I_t / r^2$





Shading

- The Phong Illumination Model



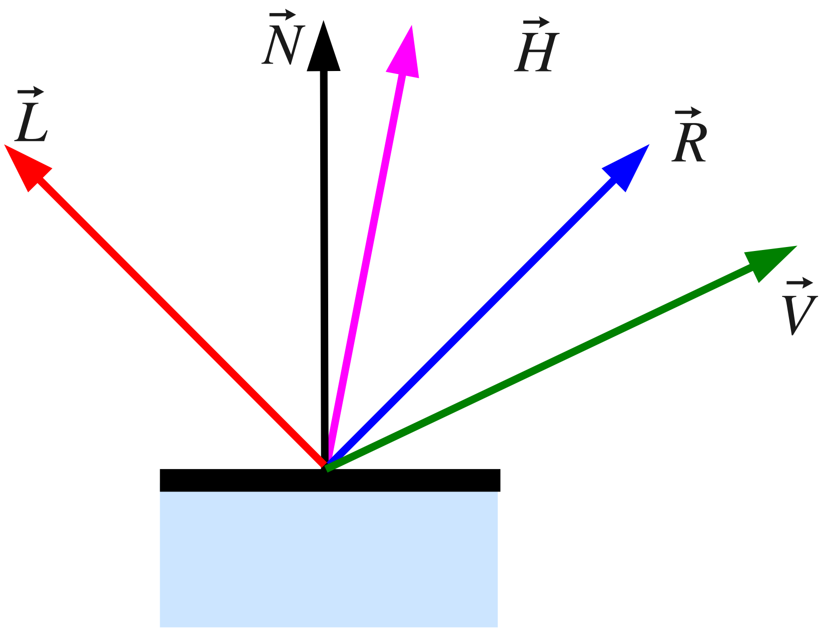
http://en.wikipedia.org/wiki/Phong_shading

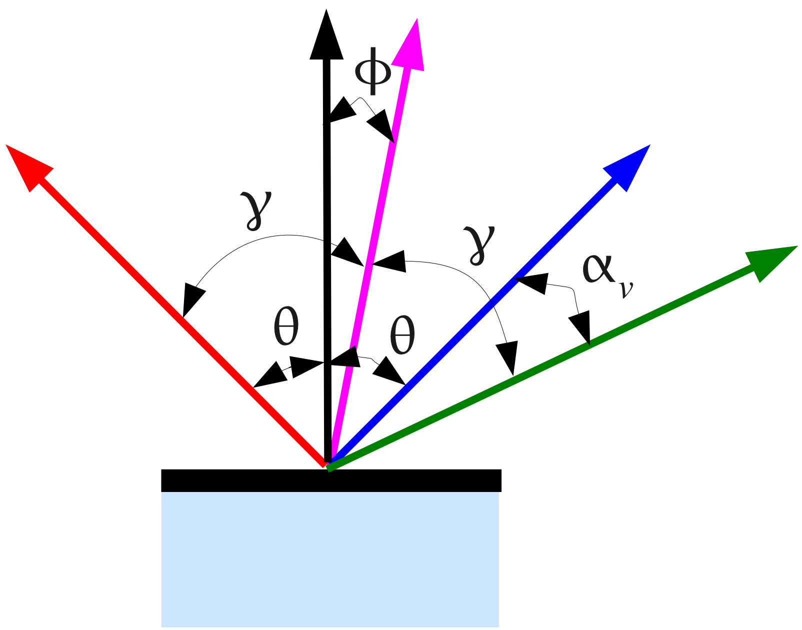
$$I = k_a I_a + k_d I_d + k_s I_s$$

  - k_a, k_d, k_s are material constants defining the amount of light that is reflected as ambient, diffuse and specular. They may be defined in as three values with R, G, B components.

Shading

- The Blinn-Phong Illumination Model





$$\vec{H} = \frac{\vec{L}+\vec{V}}{\|\vec{L}+\vec{V}\|}$$

$$I_s = I_L \cos^{n'} \phi = I_L (\vec{H} . \vec{N})^n$$

$$\theta + \alpha_v = \phi + \gamma$$

$$\theta + \phi = \gamma$$

$$\Rightarrow \alpha_v - \phi = \phi \quad \text{or} \quad \alpha_v = 2\phi$$

Shading

- Local Illumination Model

$$I_{local} = k_a I_a + \sum_{1 \le i \le m} (k_d I_{di} + k_s I_{si})$$

Global Illumination Model

$$I_{global} = I_{local} + k_r I_{reflected} + k_t I_{transmitted}$$

Shading

- Surface Material Properties
- Colour – For each object there can be a
  - Diffuse colour, Specular colour, Reflected colour and Transmitted colour
  - Remember differently coloured light is at different wavelength so:

$$I_\lambda = k_a C_{d\lambda} I_a + \sum_{1 \le i \le m} (k_d C_{d\lambda} I_{di} + k_s C_{s\lambda} I_{si}) + k_r C_{r\lambda} I_r + k_t C_{t\lambda} I_t$$

- Accounting for shadows:

$$I_\lambda = k_a C_{d\lambda} I_a + \sum_{1 \le i \le m} S_i (k_d C_{d\lambda} I_{di} + k_s C_{s\lambda} I_{si}) + k_r C_{r\lambda} I_r + k_t C_{t\lambda} I_t$$

Shading

- OpenGL uses the ***local*** Phong Illumination Model.

$$I = k_a I_a + \sum_{1 \le i \le m} (k_d I_{di} + k_s I_{si})$$

- Where and how is colour of objects computed?

Shading

- Enabling lighting and individual lights
  - glEnable(GL_LIGHTING);
  - glEnable(GL_LIGHT0);
- Every GL implementation has at least 8 lights.
- Property for the lights is defined using:
  - glLightf{v}(GLenum *light*, GLenum *pname*, GLfloat {*}*param*)
  - *light* is the light enum like GL_LIGHT1
  - *pname can be GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION, GL_SPOT_CUTOFF, GL_SPOT_DIRECTION, GL_SPOT_EXPONENT, GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, and GL_QUADRATIC_ATTENUATION*

Deprecated OGL2.x content. See the shading tutorial instead.

Shading

- For example:

```
GLfloat light_ambient(0.0, 0.0, 0.0, 1.0);
GLfloat light_diffuse(1.0, 1.0, 1.0, 1.0);
GLfloat light_specular(0.0, 1.0, 0.0, 1.0);
GLfloat light_position(3.0, 4.0, 0.0, 1.0);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glEnable(GL_LIGHT0);
```

Deprecated OGL2.x content. See the shading tutorial instead.

Shading

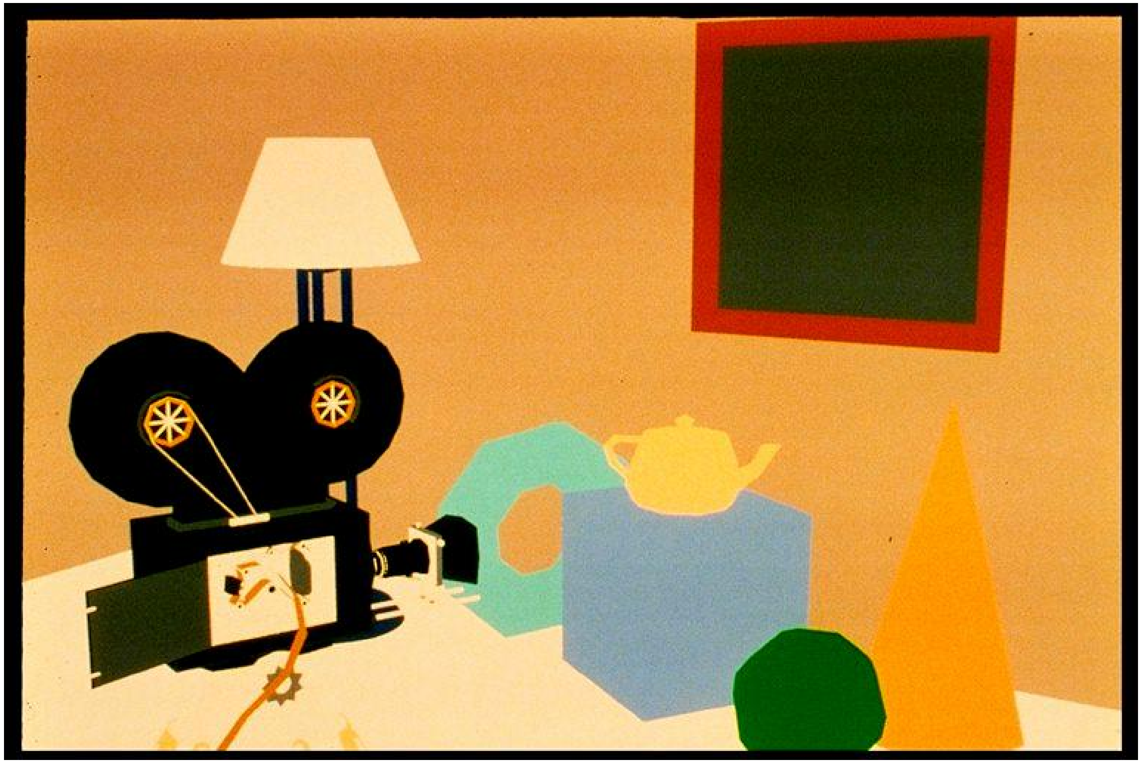
- Material properties can be specified using
  - glMaterialf{v}(GLenum face, GLenum pname, const GLfloat{*} params);
  - *face* can be GL_FRONT, GL_BACK or GL_FRONT_AND_BACK
  - *pname can be GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS, GL_AMBIENT_AND_DIFFUSE*
  - Then colour is computed at:

$$I_\lambda = k_{a\lambda} I_a + \sum_{1 \le i \le m} \left(k_{d\lambda} I_{di} + k_{s\lambda} I_{si} \right)$$

Deprecated OGL2.x content. See the shading tutorial instead.

Shading

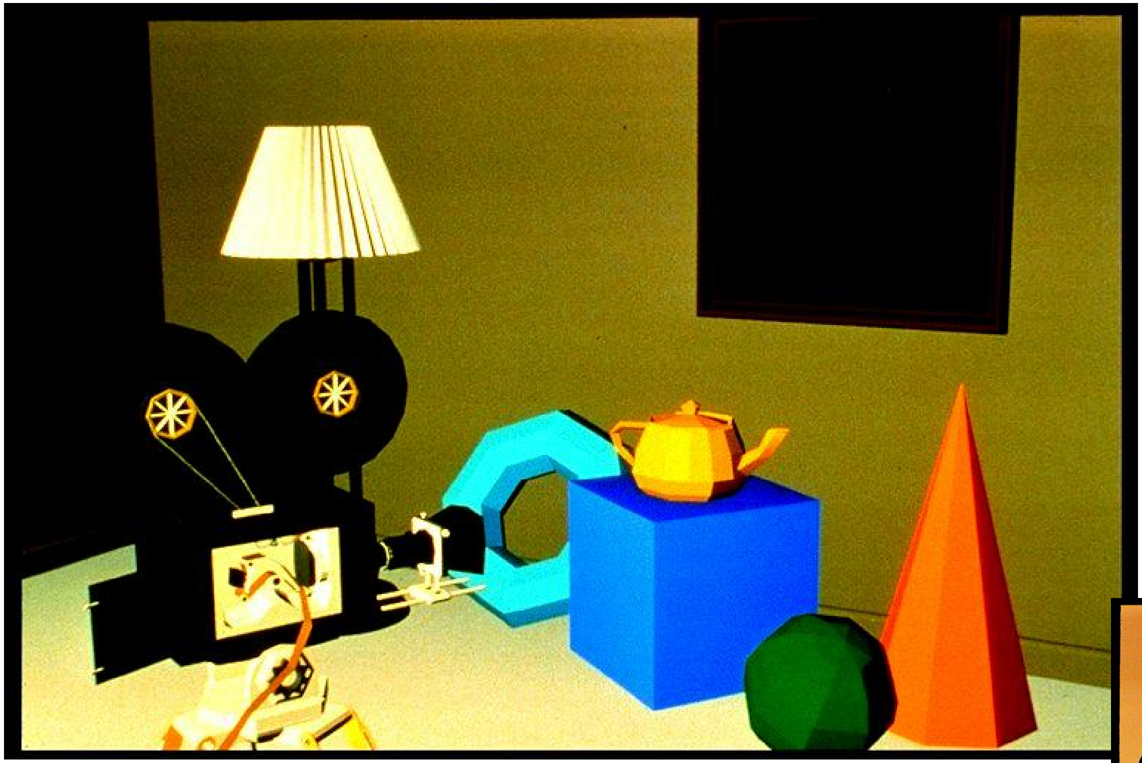
- Constant Shading – no interpolation of intensity, one intensity for whole object. No depth cues.

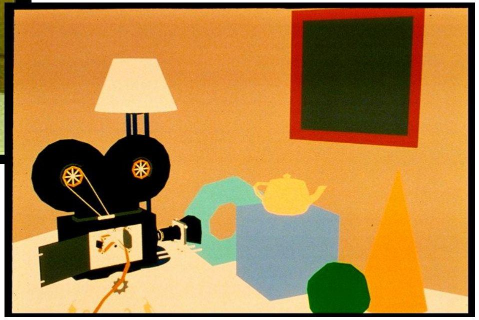


Pixar Shutterbug images from:
http://www.siggraph.org/education/materials/HyperGraph/scanline/shade_models/constant.htm

Shading

- Faceted Shading – One intensity per polygon computed from the surface normal and light vector. (GL_FLAT)

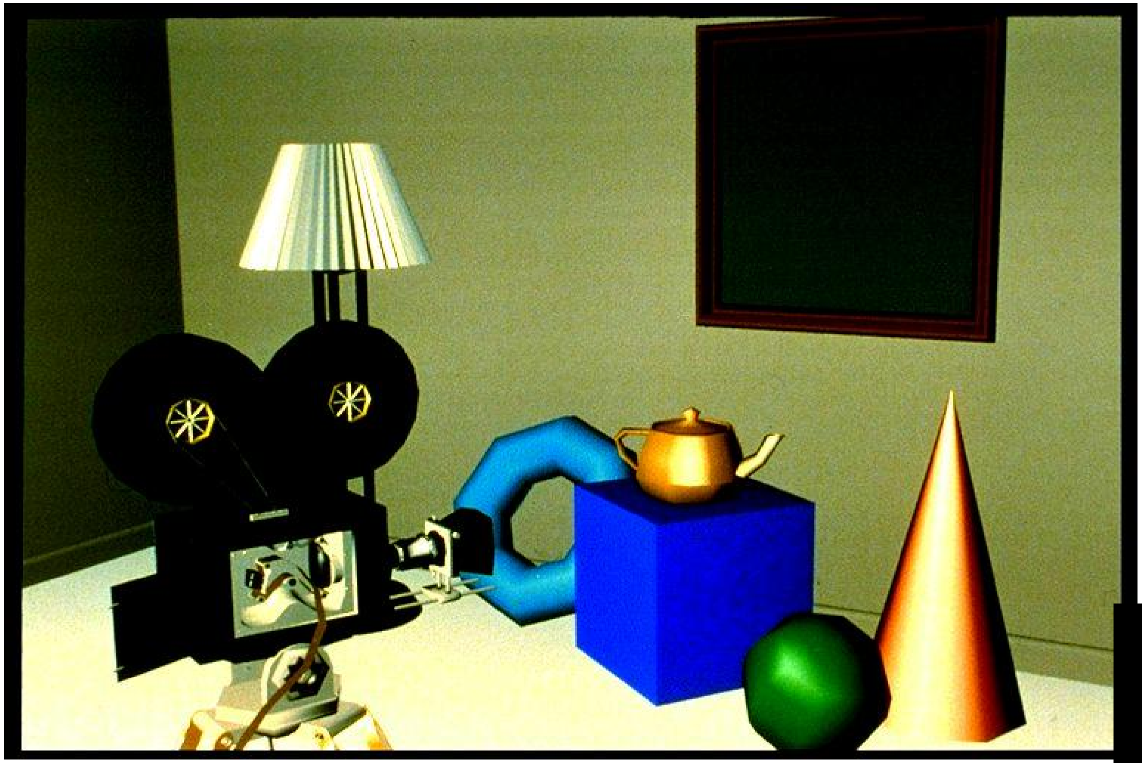


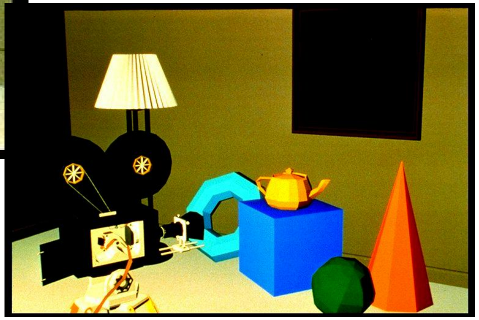


Pixar Shutterbug images from:
http://www.siggraph.org/education/materials/HyperGraph/scanline/shade_models/constant.htm

Shading

- Gouraud Shading – Linear interpolation of intensity across triangles to eliminate edge discontinuity. (GL_SMOOTH)

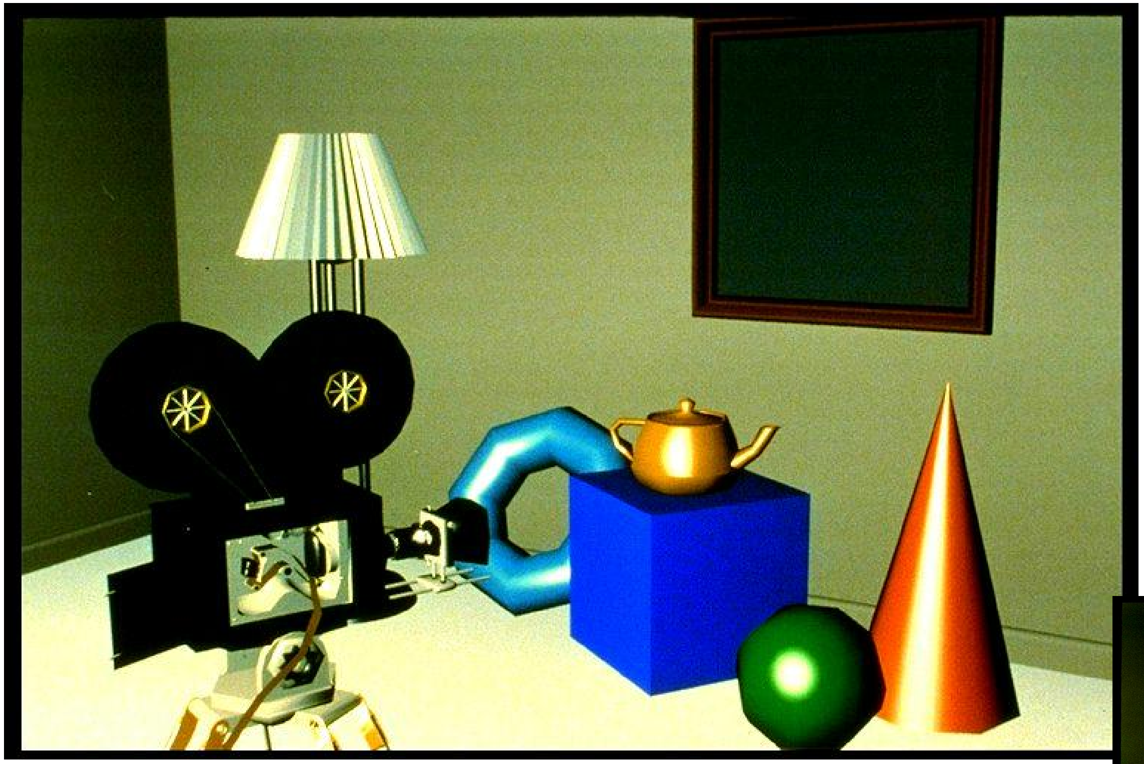


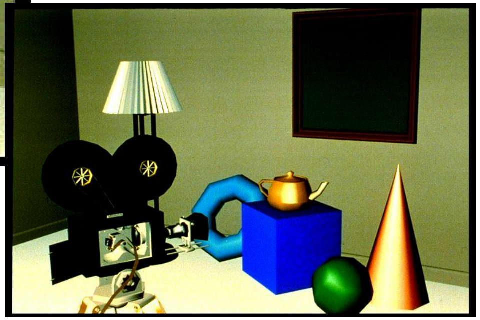


Pixar Shutterbug images from:
http://www.siggraph.org/education/materials/HyperGraph/scanline/shade_models/constant.htm

Shading

- Phong Shading – Interpolation of surface normals. Still local illumination – No GI.

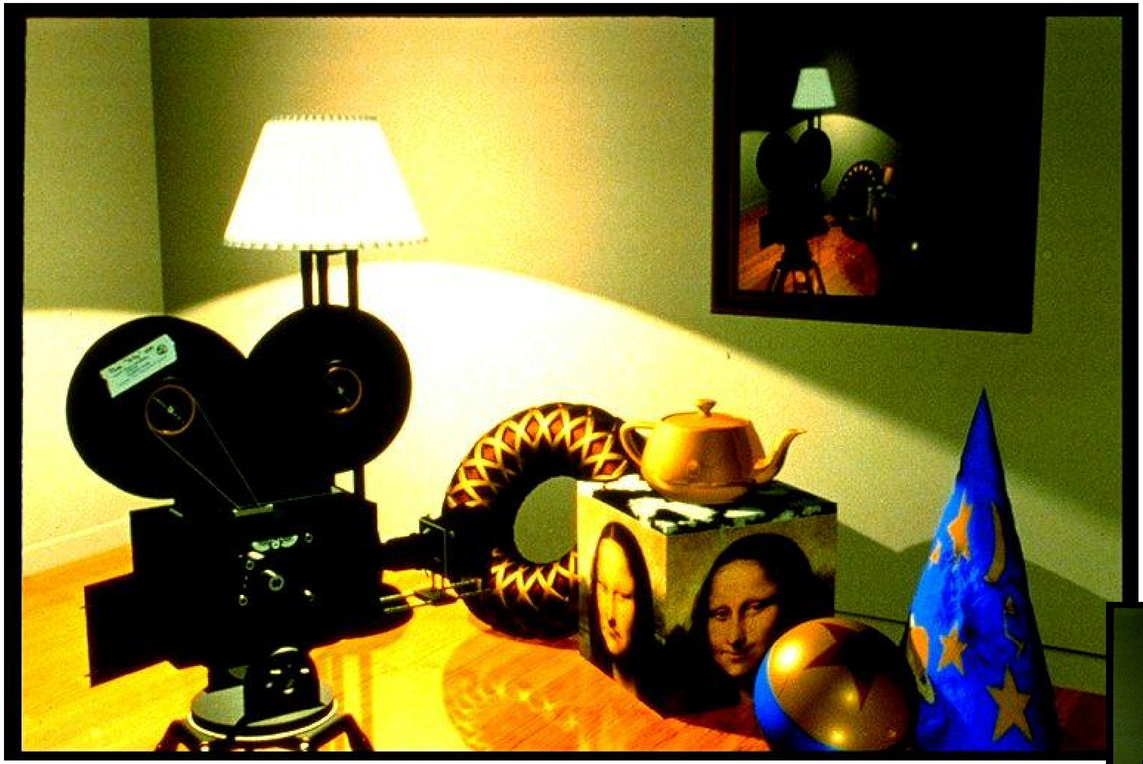


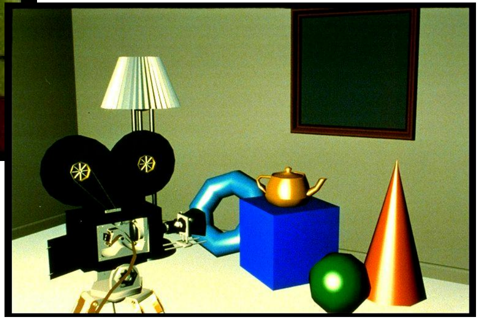


Pixar Shutterbug images from:
http://www.siggraph.org/education/materials/HyperGraph/scanline/shade_models/constant.htm

Shading

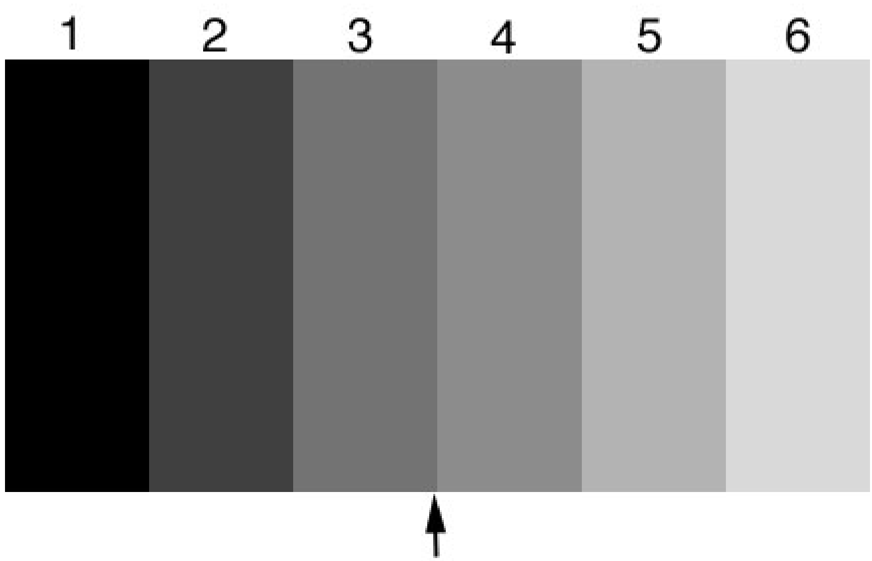
- Shadows, texture mapping, reflection mapping – simulating GI.

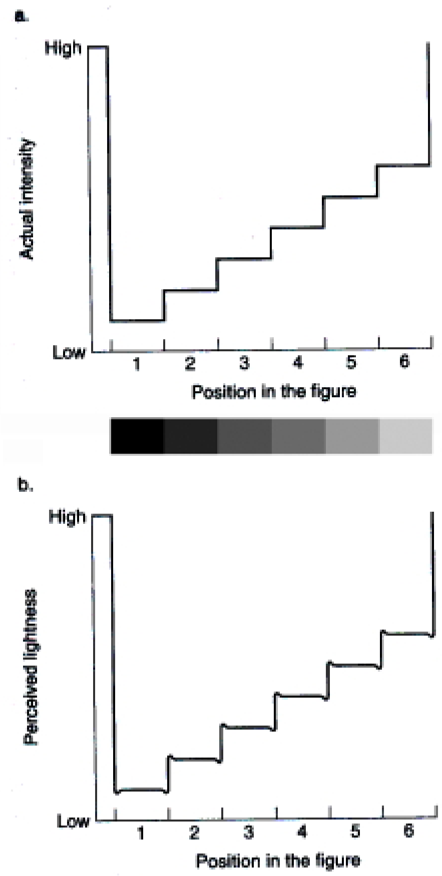




Shading

- Faceted Shading
  - Fast
  - Surface does not look smooth if a piece wise linear approximation to a flat surface is being done
  - *Mach Band Effect* accentuate the facets.

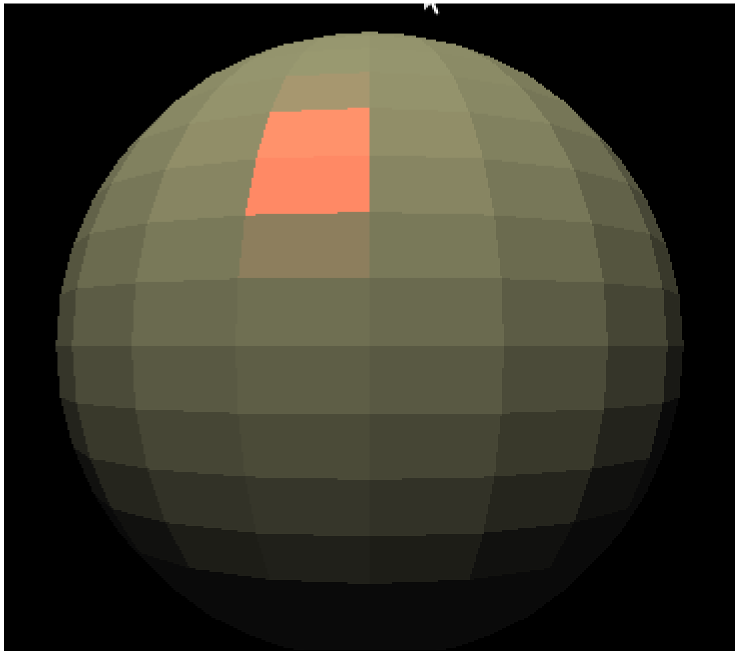


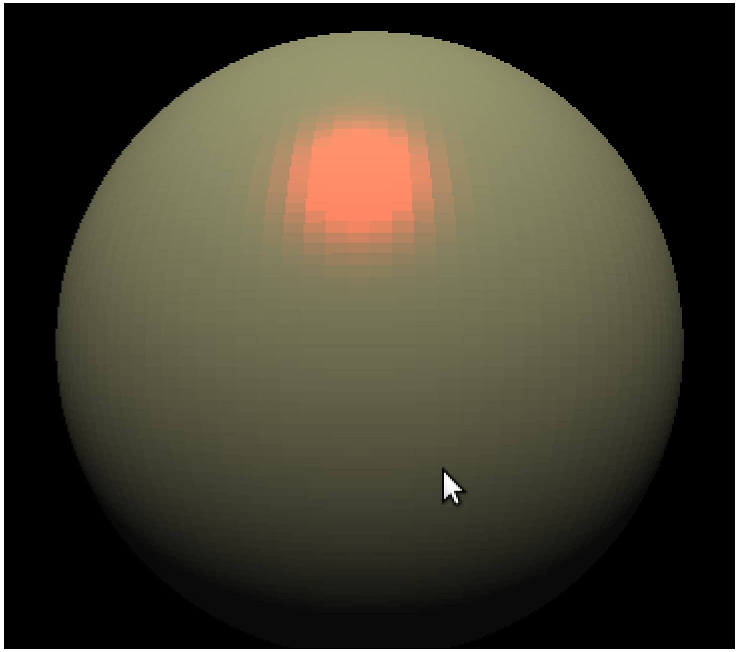


http://www.skidmore.edu/~hfoley/Perc4.htm

Shading

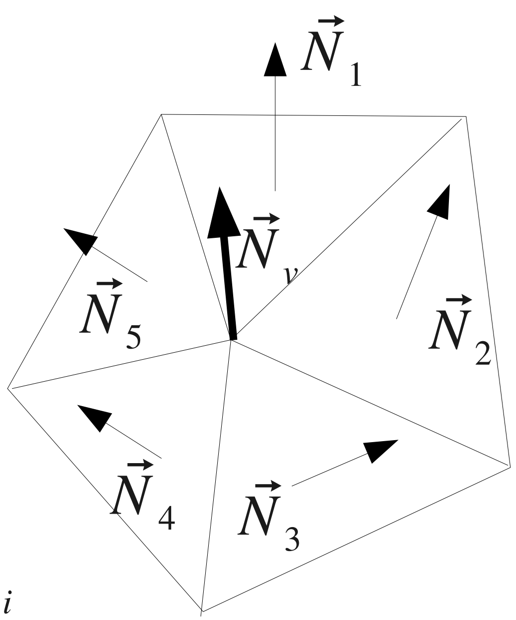
- Faceted Shading
  - glShadeModel(GL_FLAT);





Shading

- Gouraud Shading
  - Linearly interpolate intensity along scan lines: eliminates intensity discontinuities at polygon edges; still have gradient discontinuities, mach banding is largely ameliorated, not eliminated.
  - must differentiate desired creases from tesselation artifacts (edges of cube vs. edges on tesselated sphere).



- Calculate approximate vertex normals as an average of normals of polygons meeting at that vertex.

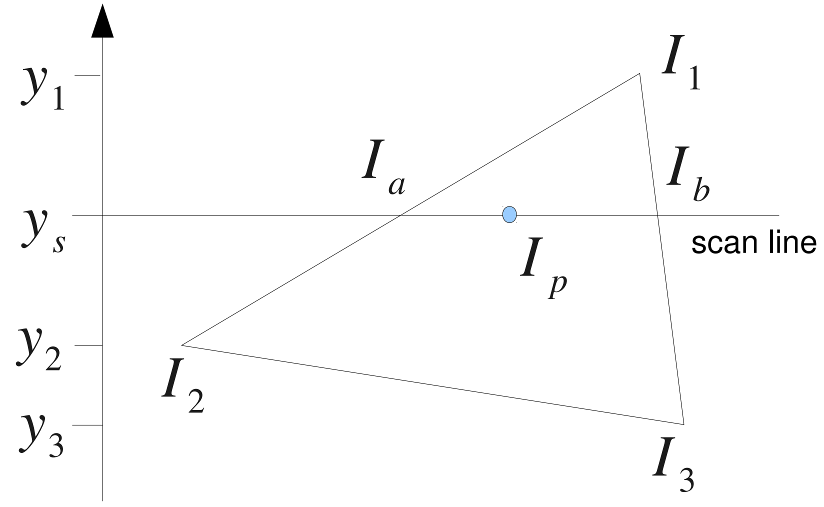
- Neighboring polygons sharing vertices and edges approximate smoothly curved surfaces and will not have greatly differing surface normals hence this approximation is reasonable.

- Calculate intensity at vertices.

$$\vec{N}_v = \frac{\sum_{i=1}^{n} \vec{N}_i}{\|\sum_{i=1}^{n} \vec{N}_i\|}$$

Shading

- Gouraud Shading
  - Linearly interpolate intensity along scan lines: eliminates intensity discontinuities at polygon edges; still have gradient discontinuities, mach banding is largely ameliorated, not eliminated.
  - must differentiate desired creases from tesselation artifacts (edges of cube vs. edges on tesselated sphere).



- Interpolate intensity along polygon edges.
- Interpolate along scan lines

$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}$$

Shading

Gouraud Shading: Vertex Shader

```
#version 430

in vec3 VertexPosition;
in vec3 VertexNormal;
in vec2 VertexTex;
out Data
{
        vec3 FrontColor;
        vec3 BackColor;
        vec2 TexCoord;
} data;

struct LightInfo
{
        vec3 Position;          //Light Position in eye-coords
        vec3 La;                //Ambient light intensity
        vec3 Ld;                //Diffuse light intensity
        vec3 Ls;                //Specular light intensity
};

struct MaterialInfo
{
        vec3 Ka;                //Ambient reflectivity
        vec3 Kd;                //Diffuse reflectivity
        vec3 Ks;                //Specular reflectivity
        float Shininess;        //Specular shininess factor
};
```

Shading

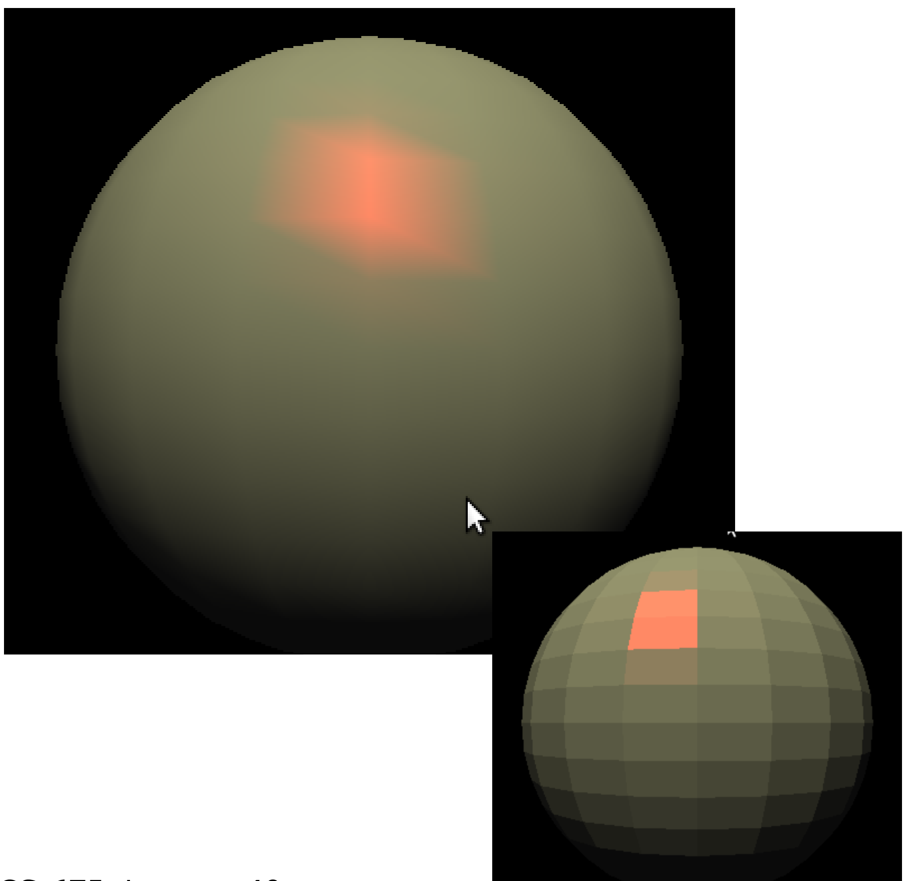
```
uniform LightInfo Light[LIGHTCOUNT];
uniform MaterialInfo Material;

uniform mat4 ModelViewMatrix;
uniform mat3 NormalMatrix;
uniform mat4 MVP;


void getEyeSpace( out vec3 norm, out vec3 position )
{
        norm = normalize( NormalMatrix * VertexNormal );
        position = vec3( ModelViewMatrix * vec4( VertexPosition, 1 ) );
}

vec3 light( int lightIndex, vec3 position, vec3 norm )
{
        vec3 s = normalize( vec3( Light[lightIndex].Position - position ) );
        vec3 v = normalize( -position.xyz );
        vec3 r = reflect( -s, norm );

        vec3 ambient = Light[lightIndex].La * Material.Ka;
        float sDotN = max( dot( s, norm ), 0.0 );
        vec3 diffuse = Light[lightIndex].Ld * Material.Kd * sDotN;
```

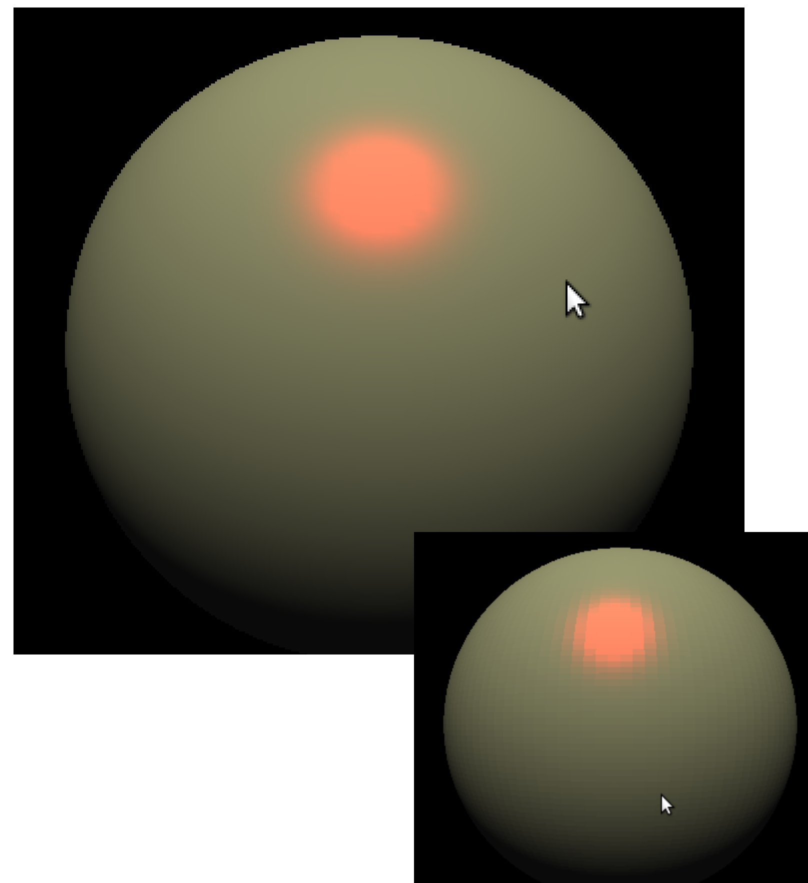
Shading

```
        vec3 spec = vec3( 0.0 );
        if ( sDotN > 0.0 )
                spec = Light[lightIndex].Ls * Material.Ks *
                         pow( max( dot(r,v) , 0.0 ), Material.Shininess );

        return ambient + diffuse + spec;
}

void main()
{
        vec3 eyeNorm;
        vec3 eyePosition;
        getEyeSpace( eyeNorm, eyePosition );

        data.FrontColor = vec3(0);
        data.BackColor = vec3(0);

        for( int i=0; i<LIGHTCOUNT; ++i )
        {
                data.FrontColor += light( i, eyePosition, eyeNorm );
                data.BackColor +=  light( i, eyePosition, -eyeNorm );
        }

        data.TexCoord = VertexTex;
        gl_Position = MVP * vec4( VertexPosition, 1 );
}
```

Shading

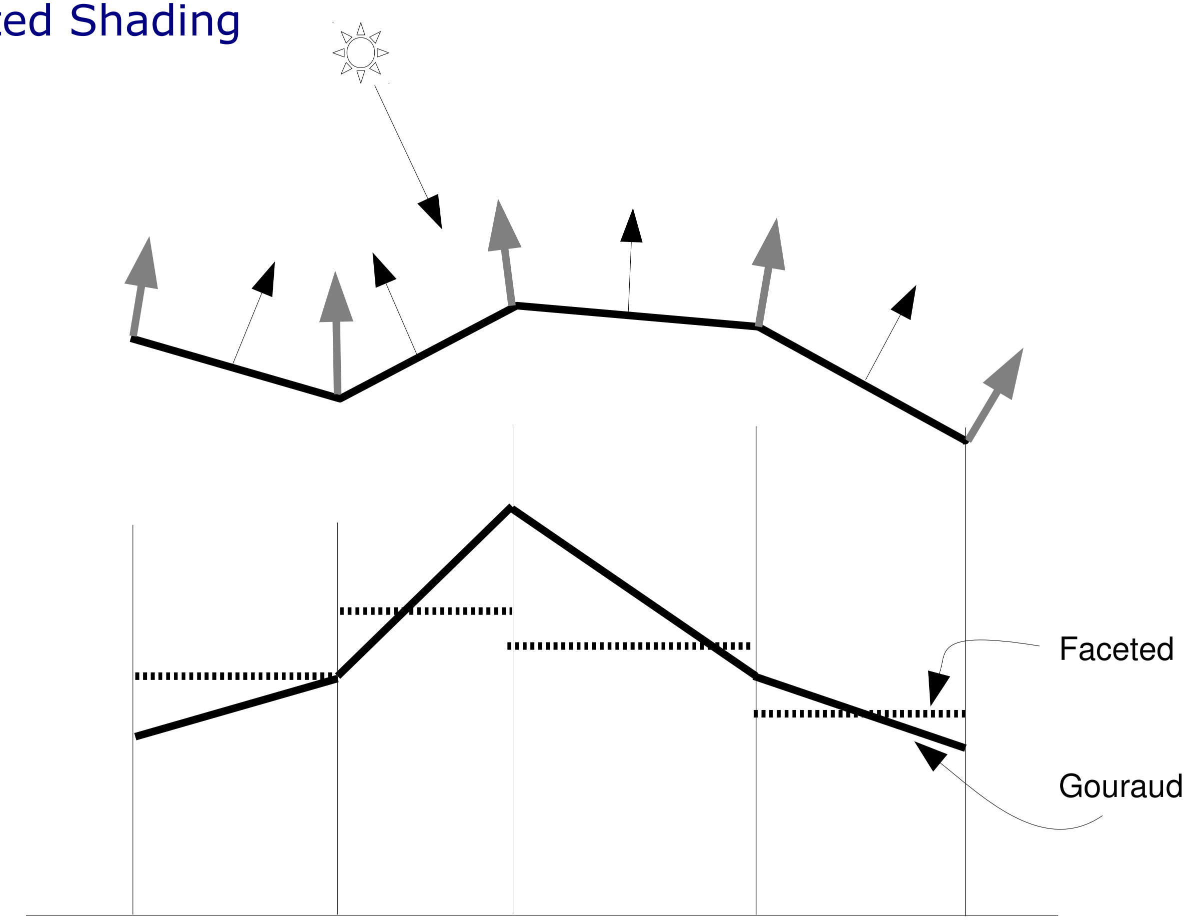
- Faceted Shading
  - glShadeModel(GL_SMOOTH);





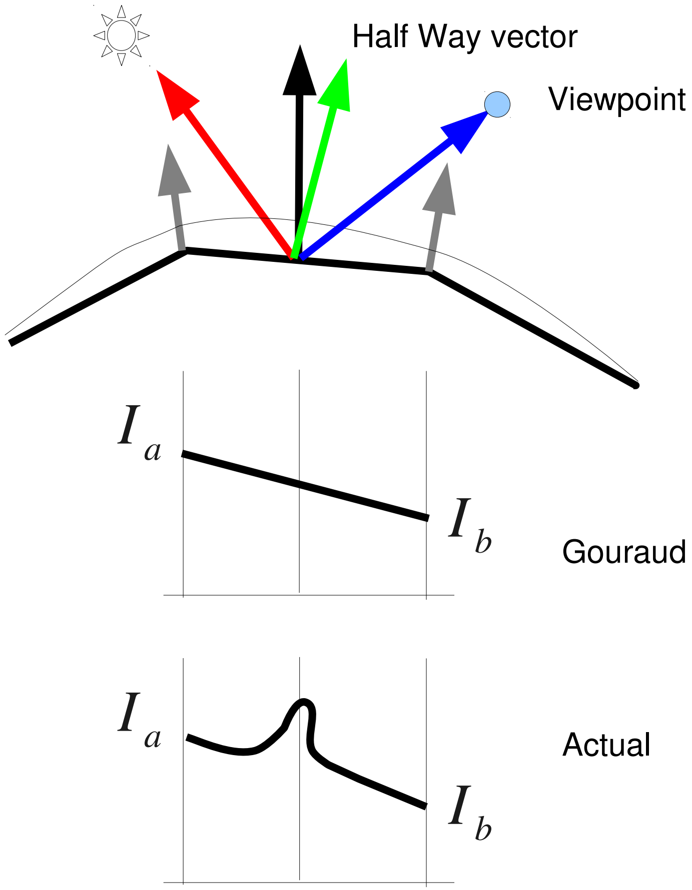
Shading

- Gouraud Shading
  - Integrates well with scanline rasterization. On an edge $\Delta I / \Delta y$ is constant.
  - vs. Faceted Shading



Shading

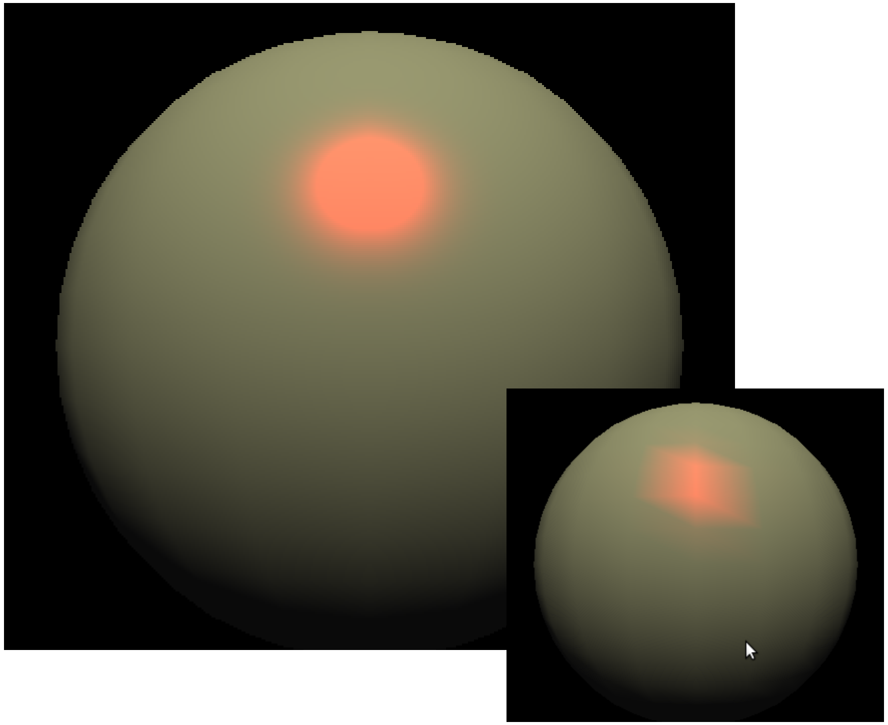
- Gouraud Shading
  - Can miss specular highlights because it interpolates vertex colors instead of calculating the intensity at every surface point.



- Interpolate normals instead – comes closer to actual surface normal.
- Called *Phong Shading* (Note: NOT Phong Illumination Model)

Shading

- Phong Shading
  - Interpolate normals along scan lines.
  - Normalize after interpolating (expensive!).
  - Not available in plain OpenGL – done as per pixel lighting on hardware.
  - Still no Global Illumination – most of the effects of Ray Tracing still missing.



Shading