EECS 487: Interactive Computer Graphics

Lecture 16:

- Phong Illumination Model
- Shading

Phong Illumination Model

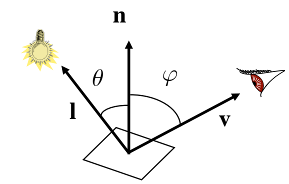
A local illumination model

- one bounce: light ⟶ surface ⟶ viewer

Lighting a single point

At the point:

$\mathbf{n}$: surface normal (orientation of surface)
$\mathbf{l}$: light vector (surface to light)
$\mathbf{v}$: viewing vector (surface to eye)
θ: light angle of incidence
φ: viewing angle



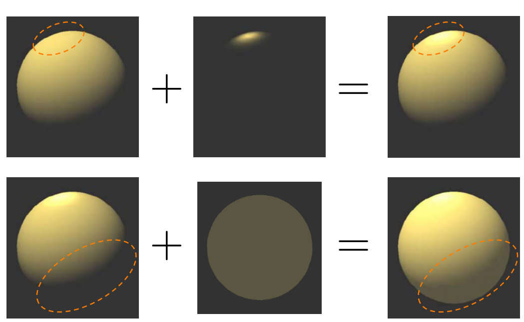
Phong Illumination Model

Approximate surface color as sum of three components:

- an ideal diffuse component
- a glossy/blurred specular component
- an ambient term

Light sources:

s_d	diffuse light intensity
s_s	specular light intensity
s_a	ambient light intensity



Surface Reflection Coefficients

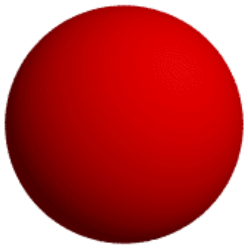
Approximate BRDFs: not physically based

Modify surface color by changing reflection coefficients based on:

- material type
- surface finish
- texture maps
- what looks good
  - artistic license
  - trial and error
  - personal library

m_a	ambient color reflection
m_d	diffuse color reflection
m_s	specular color reflection
m_{shi}	shininess (blurriness)
m_e	emissive color intensity

Diffuse Example



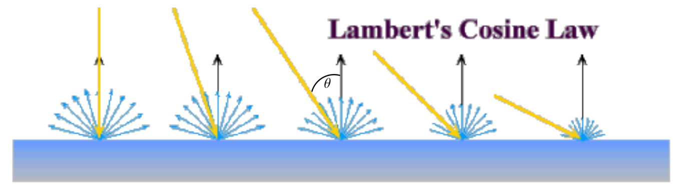
Where is the light?

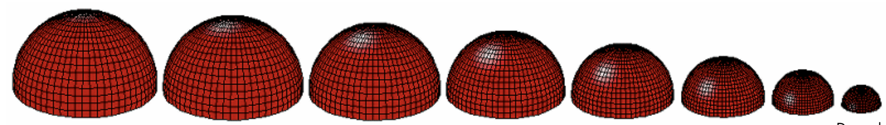
Where's the normal direction at the brightest point?

Ideal Diffuse Reflectance

Ideal diffuse surface reflects light equally in all directions, according to Lambert's cosine law:

- amount of light energy that falls on surface and gets reflected is proportional to the incidence angle, θ
- perceived brightness (reflectance) is view independent



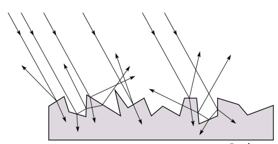


Durand

Ideal Diffuse Reflectance

At the microscopic level, an ideal diffuse surface is a very rough surface

- microfacets are oriented in any which way
- examples: chalk, clay, surface of moon



Curless

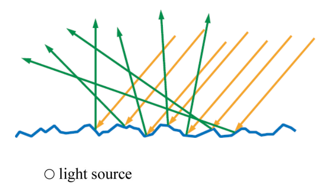
 

Durand

Ideal Diffuse Model

Amount of light energy that falls on surface is proportional to incidence angle, θ:

$\mathbf{c}_d = s_d \otimes m_d \cos\theta$

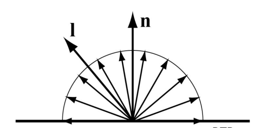


For normalized $\mathbf{n}$ and $\mathbf{l}$:

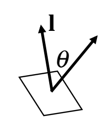
$\mathbf{c}_d = s_d \otimes m_d \max((\mathbf{n}\cdot\mathbf{l}), 0)$

- $(\mathbf{n}\cdot\mathbf{l}) < 0 \Rightarrow \cos\theta < 0 \Rightarrow \theta > 90^{\circ}$, light source is behind surface
- why must $\mathbf{n}$ and $\mathbf{l}$ be normalized?

$$\mathbf{u}\cdot\mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\|\cos\theta \quad \Rightarrow \quad \cos\theta = \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$



RTR2



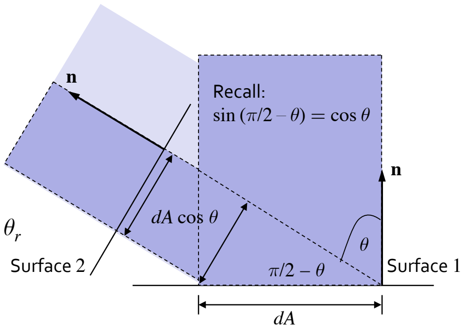
Diffuse Reflectance and Viewing Angle

Lambert's Cosine Law: amount of light energy that falls on surface and gets reflected is proportional to incidence angle, θ_i

- $\cos\theta_i =$ dot product of light vector with normal (both normalized)

Shouldn't the amount of energy reflected also be proportional to viewing angle, θ_r?

- no, whereas larger θ_i means energy arriving over area $dA\cos\theta_i$ is spread across the larger area dA, larger θ_r simply means collecting energy from the same area dA, but channelling it through the smaller area $dA\cos\theta_r$

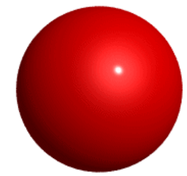
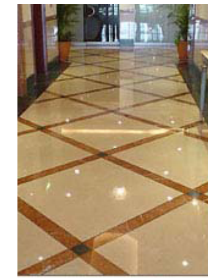


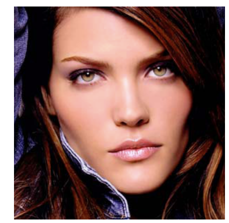
TP3, FvD94

Ideal Specular/Mirror Reflectance

Accounts for highlight seen on objects with smooth, shiny surfaces, such as:

- metal
- polished stone
- plastics
- apples
- skin




Curless,Zhang

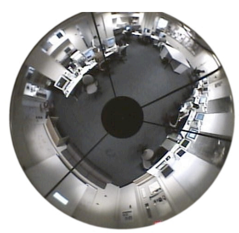
Ideal Specular/Mirror Reflectance

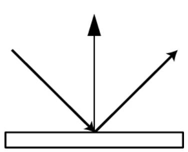
Reflection only at mirror angle

- highlight intensity depends on viewing direction

Model: all microfacets of mirror surface are oriented in the same direction as the surface itself

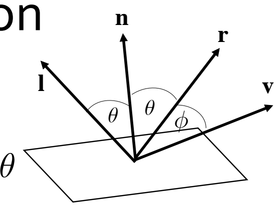
- examples: mirrors, highly polished metals





Durand

Phong Specular Reflection



Simulates a highlight

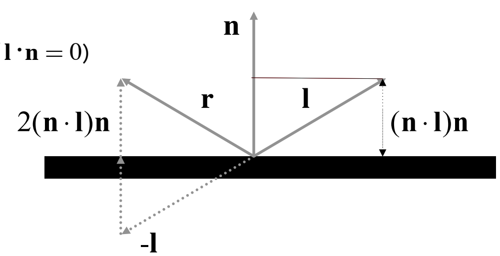
Reflection angle $=$ incidence angle $= \theta$

Most intense specular reflection when $\mathbf{v} = \mathbf{r}$

How to compute $\mathbf{r}$?

$\mathbf{r} = -\mathbf{l} + 2(\mathbf{n}\cdot\mathbf{l})\mathbf{n}$

(in OpenGL, specular term is 0 if $\mathbf{l}\cdot\mathbf{n} = 0$)



Glossy Reflectors

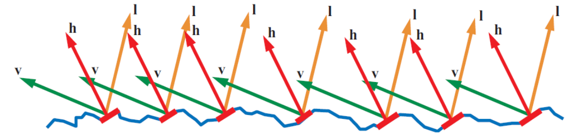
Real materials tend to deviate significantly from ideal mirror reflectors ⇒ highlight and reflections are blurry

Also known as: "rough specular", "directional diffuse", or "glossy" reflection

(there are no ideal diffuse surfaces either ...)

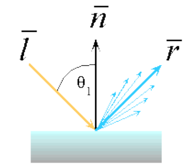
Durand

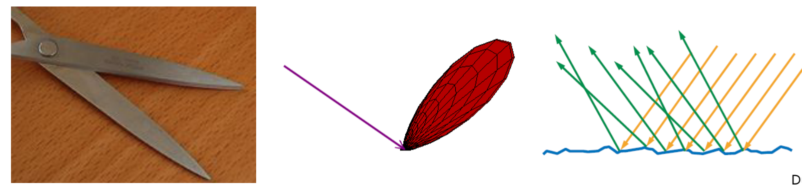
Glossy Reflectors



Simple empirical model:

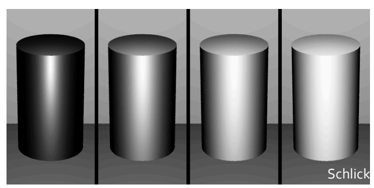
- we expect most of the reflected light to travel in the direction of the ideal reflection ray
- but due to variations in microfacet orientations, some of the light will be reflected just slightly off from the ideal reflected direction
- as we angle away from the reflected ray, we expect to see less light reflected



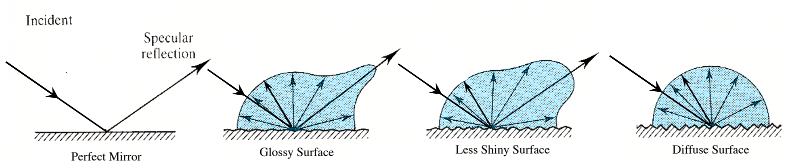


Durand

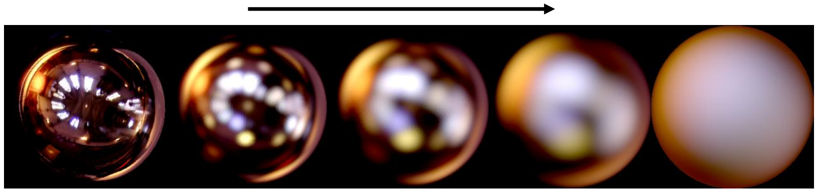
Surface Roughness



specular diffuse

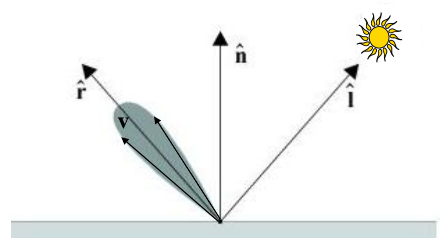


increasing roughness

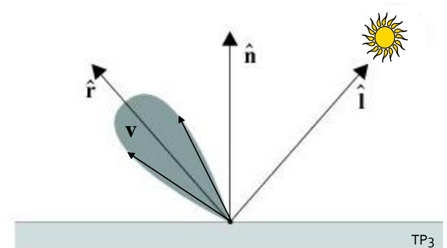


Phong "Glossy" Reflection Model

As $\mathbf{v}$ angles away from $\mathbf{r}$, specular reflection falls off, simulating "glossy" reflection:



less glossy/
more specular

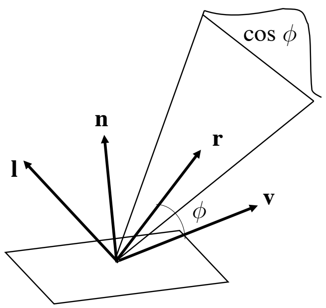


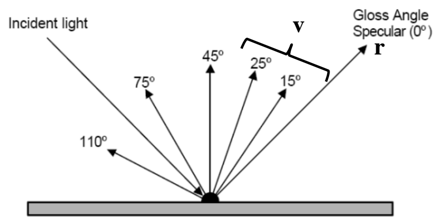
more glossy/blurry

Phong "Glossy" Reflection Model

Phong specular model:

$\mathbf{c}_s = s_s \otimes m_s \cos\phi = s_s \otimes m_s \max((\mathbf{r} \cdot \mathbf{v}), 0)$



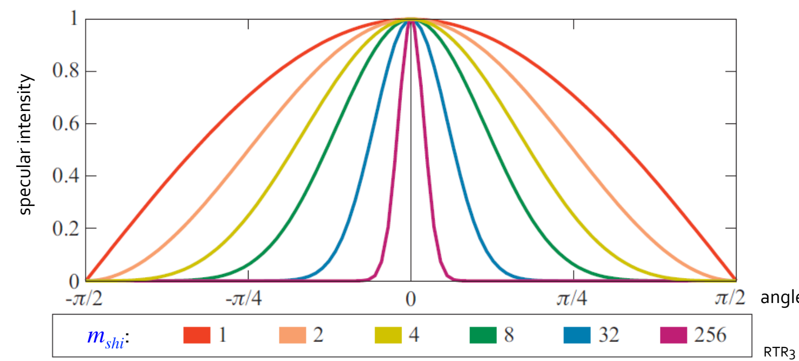


Material Glossiness

To account for the shininess of different material:

$\mathbf{c}_s = s_s \otimes m_s \max((\mathbf{r} \cdot \mathbf{v}), 0)^{m_{shi}}$

larger m_{shi} gives tighter and shinier highlight, with sudden dropoff, simulating a more mirror-like surface

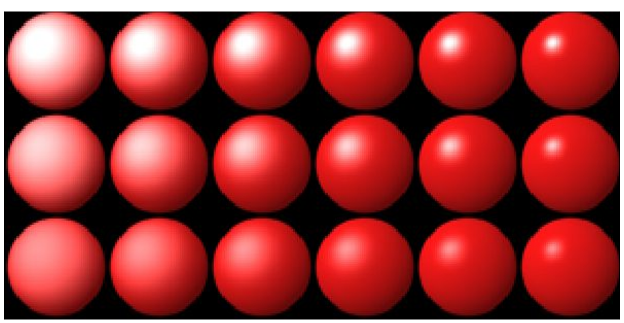


RTR3

Phong Specular Reflection

larger m_{shi}, tighter highlight →

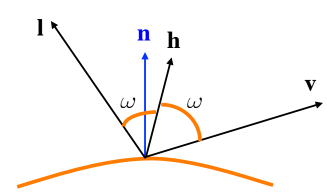
larger m_s, shinier ↑



Blinn-Phong (Blinn-Torrance) Model

Back to micro facets: 

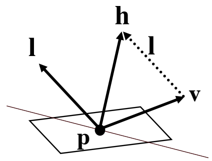
- model surface by a collection of tiny mirrors
- specular reflectance comes from mirrors oriented halfway between $\mathbf{l}$ and $\mathbf{v}$, in direction of $\mathbf{h}$



Manocha

Blinn-Phong

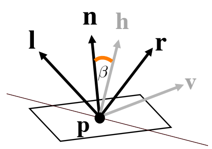
Model specular reflection with "halfway" vector ($\mathbf{h}$) instead of $\mathbf{r}$



- consider the microfacet through point $\mathbf{p}$ that reflects light perfectly to the viewer
- $\mathbf{h}$ is the normal of this plane, it is halfway between $\mathbf{l}$ and $\mathbf{v}$ (by definition), $\mathbf{h}$ normalized

$$\mathbf{h} = \frac{\mathbf{l}+\mathbf{v}}{\|\mathbf{l}+\mathbf{v}\|}$$

- angle between $\mathbf{h}$ and $\mathbf{n}$ is β; $\mathbf{v} = \mathbf{r}$ when $\mathbf{h} = \mathbf{n}$ (β is not the angle between $\mathbf{v}$ and $\mathbf{r}$)
- specular reflection modeled as:
  $\mathbf{c}_s = s_s \otimes m_s\, (\mathbf{n} \bullet \mathbf{h})^{4m_{shi}} = s_s \otimes m_s\, (\cos\,\beta)^{4m_{shi}}$



Illumination Models

A rendering process can be modeled as an integral equation representing the transport of light through the environment ⇒ the rendering equation

Local illumination: an approx. of the rendering eqn.

- assumes light is scattered only once: light from light source is reflected by a surface and modulated on its way towards the eye

Global illumination:

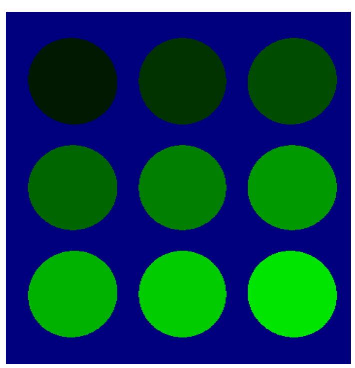
- light rays traveling from light to surface may be
  - blocked by intervening surfaces (shadows) or
  - bent or scattered by intervening material (refraction and atmospheric effects) or
- light arriving at a surface may come indirectly via another surface (reflection and color bleed)

Phong Ambient Term

"Approximates" the contribution of all indirect illumination

Surface uniformly lit

- areas with no direct illumination are not completely dark
- independent of:
  - light direction
  - surface normal
  - viewing angle

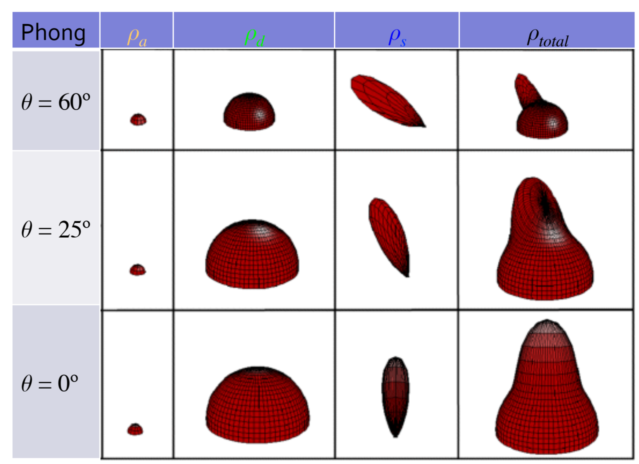


Spheres rendered with ambient reflection only

Putting It All Together

Phong Illumination Model

$$\mathbf{c}_t = m_a s_a + (m_d(\mathbf{n}\cdot\mathbf{l}) + m_s(\mathbf{r}\cdot\mathbf{v})^{m_{shi}})f(d)s$$



Durand

OpenGL Light Sources

`glLightfv(lightname,param,value)`

- parameters
  - `GL_AMBIENT`
  - `GL_DIFFUSE`
  - `GL_SPECULAR`
  - `GL_POSITION`

  - `GL_SPOT_DIRECTION`
  - `GL_SPOT_CUTOFF`
  - `GL_SPOT_EXPONENT`

  - `GL_CONSTANT_ATTENUATION`
  - `GL_LINEAR_ATTENUATION`
  - `GL_QUADRATIC_ATTENUATION`

Turning on the lights:

- `glEnable(GL_LIGHTING)`
- `glEnable(GL_LIGHT0)`

OpenGL Lighting and Reflectance

```
/*  Initialize material property, light source,
    lighting model, and depth buffer. */
void init(void)
{
   GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat mat_shininess[] = { 50.0 };
   GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_SMOOTH);

   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

   glLightfv(GL_LIGHT0, GL_POSITION, light_position);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);
}
```

OpenGL Material Properties

`glMaterial(face,param,value)`

- face
  - `GL_FRONT`
  - `GL_BACK`
  - `GL_FRONT_AND_BACK`
- parameters
  - `GL_AMBIENT`
  - `GL_DIFFUSE`
  - `GL_AMBIENT_AND_DIFFUSE`
  - `GL_SPECULAR`
  - `GL_SHININESS`
  - `GL_EMISSION`

See Redbook Chapter 5 for techniques to minimize performance costs associated with changing material properties

Choosing the Parameters

Experiment with different parameter settings

A few suggestions:

- $m_a + m_d + m_s < 1$
- use a small m_a (~ 0.1)
- try $m_{shi} \in [0, 100]$

TP3

material	m_d	m_s	m_{shi}
metal	small, color of metal	large, color of metal	large
plastic	medium, color of object	medium, color of light (or white)	medium
planet	varying	0	0

Highlight Color

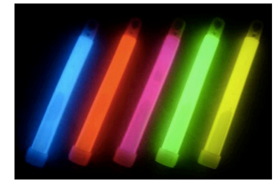
For non-metals, e.g., plastics, highlight color is color of light (plastics has a transparent/white coating)

For metals, e.g., brass, highlight depends on surface color

plastic metal clay

Apodaca&Gritz

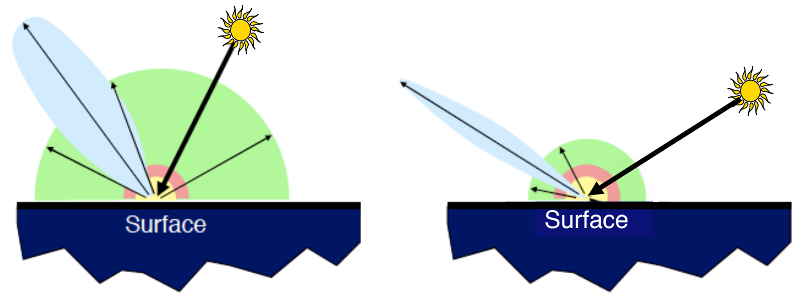
Emissive Term



Material's emissive parameter m_e

- assumed not lighting other objects

$$\mathbf{c}_t = m_e + m_a s_a + (m_d(\mathbf{n}\cdot\mathbf{l}) + m_s(\mathbf{v}\cdot\mathbf{r})^{m_{shi}})f(d)s$$

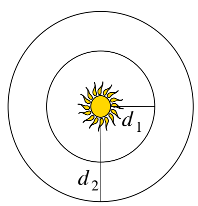


Funkhouser

Attenuation Model

Whereas the ambient term simulates indirect lighting, attenuation model and fog simulate scattering effect

Intensity attenuation: light falls off the further away one gets from the source



- distinguishes overlapping surfaces having the same reflection parameters
- radiant energy disperses as $1/d^2$
  - d is the distance from the light source to surface
  - reason for the inverse square falloff?
- attenuation function $f(d) = 1/(a_0 + a_1 d + a_2 d^2)$
  - user defined constants a_0, a_1, a_2
  - since we're not modeling atmospheric scattering, $1/d^2$ often appears too harsh
  - instead, we use $f(d) = 1/(a_0 + a_1 d)$ or no attenuation

Lighting with attenuation: $\mathbf{c}' = f(d)\mathbf{c}$

Global Ambient and Multiple Lights

Global ambient light source: $\mathbf{c}_g = s_g \otimes m_a$

- s_g set with:
  `glLightModel*(GL_LIGHT_MODEL_AMBIENT,ambient_light)`

Multiple lights:

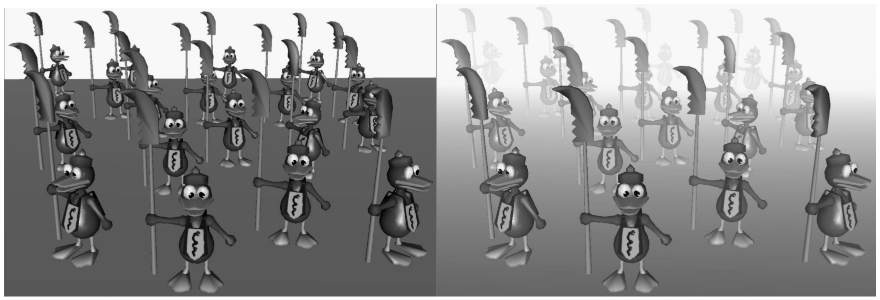
$$\mathbf{c}_t = \mathbf{c}_g + m_e + \sum_{k=1}^{n} \mathbf{c}_a^{(k)} + f(d^{(k)})(\mathbf{c}_d^{(k)} + \mathbf{c}_s^{(k)}),$$

k: light number, not exponentiation

Fog

Simple atmospheric effect

- a little better realism
- help to convey distances
- 2 forms of depth cueing (ignoring scattering)
  - light-to-object distance conveyed by light attenuation
  - object-to-viewer atmospheric scattering simulated by fog



Akenine-Möller02

Fog

color of fog: $\mathbf{c}_f$
color of surface: $\mathbf{c}_s$
fog effect:

$$\mathbf{c}_p = f\,\mathbf{c}_s + (1-f)\,\mathbf{c}_f, f \in [0,1]$$

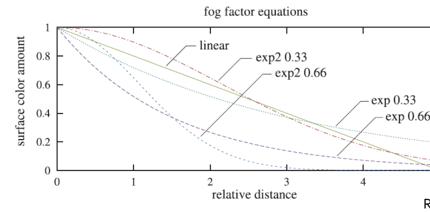
f_p: fog factor at point p

$z_{end} - z_{start}$: the foggy range

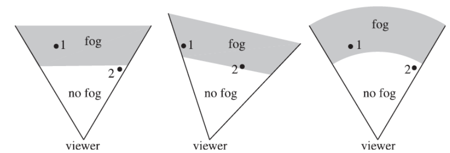
d_f : fog density

How to compute f?

- linear: $f_p = \dfrac{z_{end} - z_p}{z_{end} - z_{start}}$
- exponential: $f_p = e^{-d_f z_p}$
- exponential-squared: $f_p = e^{-(d_f z_p)^2}$



RTR3



can also be a function of height off the ground

radial- or range-based, not simply z-based

Fog in OpenGL

```
glEnable(GL_FOG);
glHint(GL_FOG_HINT,GL_[NICEST|FASTEST]);
                  // per fragment or per vertex
glFog*(GL_FOG_MODE, GL_[LINEAR|EXP|EXP2]);
glFog*(GL_FOG_[START|END|DENSITY], param);
glFog*(GL_FOG_COLOR, param);
```

To specify a fog depth for a given vertex:

```
glFog*(GL_FOG_COORD_SRC, GL_FOG_COORD);
glFogCoord*(z);
glVertex(…);
```

Lighting and Shading

Lighting: computing interaction between light and surfaces of different materials, and interaction with the geometry of objects to determine the luminous intensity reflected from a specified 3D point

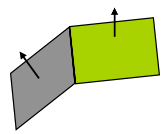
Shading: performing the lighting computations on polygonal objects and coloring the pixels

flat shading | Gouraud shading | Phong shading

RTR3

Flat Shading

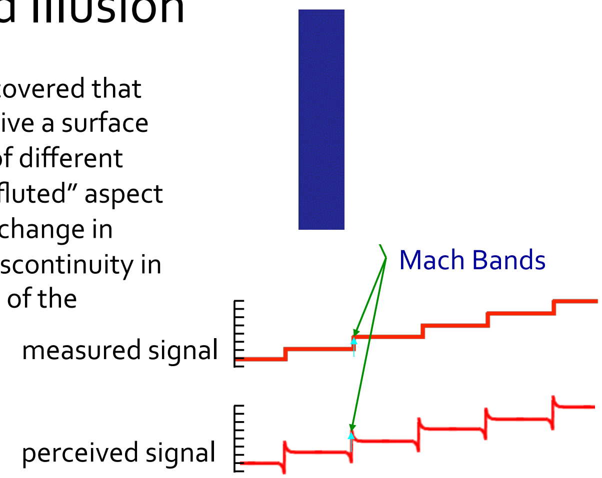
Entire polygon displayed with the same intensity
- calculate intensity from the reflection model
- use the surface normal (for triangles)
- compute an average normal (for >3 vertex polygons)



Is a valid shading model when:
- object is truly planar
  (not an approximation of a curved surface)
- all k light sources are far enough away that
  $\mathbf{n} \cdot \mathbf{l}^{(k)}$ is constant over the polygon surface
- viewing position is far enough away that
  $\mathbf{n} \cdot \mathbf{h}$ is constant over the polygon surface

Problem with Flat Shading: Mach Band Illusion

In 1865, Mach discovered that human eyes perceive a surface with flat shading of different hues as having a "fluted" aspect due to the abrupt change in shades of color (discontinuity in the first derivative of the shading function)

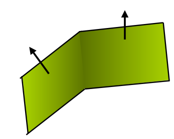


Lateral Inhibition

Mach Band Illusion is due to lateral inhibition of the human visual system:
- neighboring receptors are connected
- when one fires, it inhibits its neighbors from firing

Effect:
- eye sensitive to difference from surrounding area
  - good for edge and motion detection

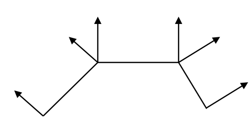
Gouraud Shading



Match intensity across polygon edges

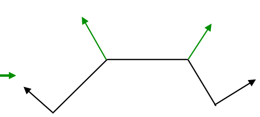
Linear interpolation of color across polygon surface

Gouraud shading algorithm:
- determine average normal at each vertex (averaged over the normals of all polygons that share the vertex)
- compute color at each vertex using the average normal
- linearly interpolate color across a single polygon surface
  (`GL_SMOOTH` does only the last step! `GL_FLAT` colors polygon with the color of the last vertex)



$$\mathbf{n}_{avg} = \frac{\sum_{i=1}^{n} \mathbf{n}_i}{\left\| \sum_{i=1}^{n} \mathbf{n}_i \right\|}$$

$n =$ all polygons that share the vertex



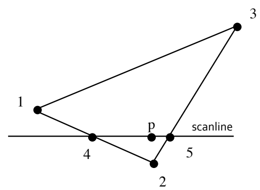
Gouraud Shading Example

1. Compute average normal at each vertex $(1, 2, 3)$
2. Compute colors $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$
3. Compute colors $\mathbf{c}_4$, $\mathbf{c}_5$

$$\mathbf{c}_4 = \frac{y_4 - y_2}{y_1 - y_2}\mathbf{c}_1 + \frac{y_1 - y_4}{y_1 - y_2}\mathbf{c}_2$$

4. Compute color $\mathbf{c}_p$

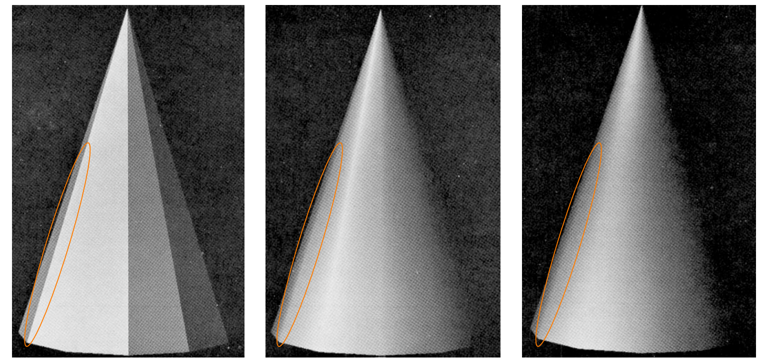
$$\mathbf{c}_p = \frac{x_5 - x_p}{x_5 - x_4}\mathbf{c}_4 + \frac{x_p - x_4}{x_5 - x_4}\mathbf{c}_5$$



Or use barycentric coordinates!
Steps 2-4 were cheap enough to be implemented in hardware even for fixed pipeline GPU

Mach Band not Eliminated

when there's an abrupt change in the orientation of two polygons and there's discontinuity in the 1^{st} derivative of the shading function



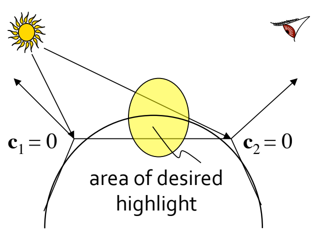
effect can be ameliorated with increased polygon count or Phong shading …

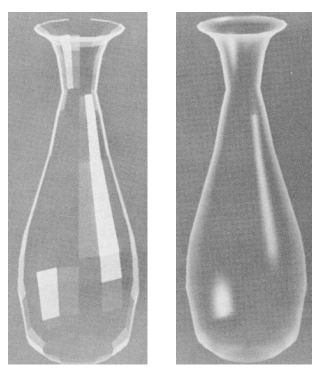
Limitation: Wrong Highlights

Highlights depend on polygonal shape

Does not capture highlight in the middle of polygon

- $\mathbf{c}_1 = 0$ because $(\mathbf{n} \cdot \mathbf{h}) < 0$
- $\mathbf{c}_2 = 0$ because $(\mathbf{n} \cdot \mathbf{l}) < 0$
- any interpolation of $\mathbf{c}_1$ and $\mathbf{c}_2$ will be 0





Shows up in animation as flashing highlights between frames as orientation of polygons change

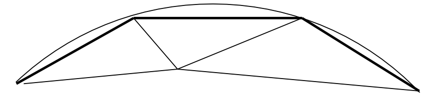
Limitation: Wrong Highlights

Why does Gouraud give inaccurate simulation of highlights?

- color at vertex may not be (highlight) color inside the polygon

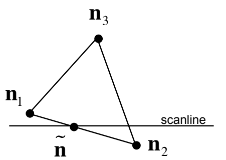
What would give the most accurate simulation of highlights?

- compute normal of actual surface (not polygon-approximated surface) at each pixel and color accordingly ⇒ very expensive!



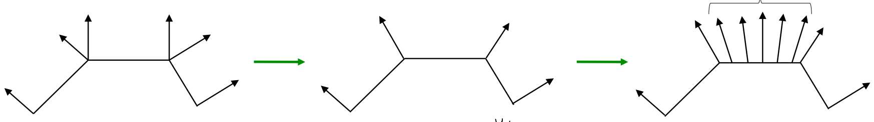
Phong shading: interpolate normal of vertices across polygon surface and color each pixel according to interpolated normals

Phong Shading

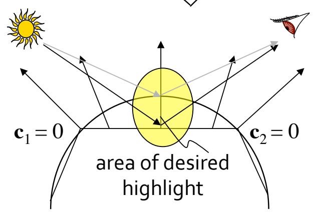


$$\tilde{\mathbf{n}} = \frac{y - y_2}{y_1 - y_2}\mathbf{n}_1 + \frac{y_1 - y}{y_1 - y_2}\mathbf{n}_2$$

Algorithm:
- determine average normal at each vertex
- linearly interpolate per-pixel normal across surface
- compute color for each pixel using the value of the approximated per-pixel normal



Phong illumination model ≠
Phong specular reflection ≠
Phong shading



OpenGL Lighting and Reflectance

```
/*  Initialize material property, light source,
    lighting model, and depth buffer. */
void init(void)
{
   GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat mat_shininess[] = { 50.0 };
   GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_SMOOTH); // or GL_FLAT
   // interpolate color of vertices, but does not
   // average normals at the vertices

   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

   glLightfv(GL_LIGHT0, GL_POSITION, light_position);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);
}
```

Where in the Pipeline?