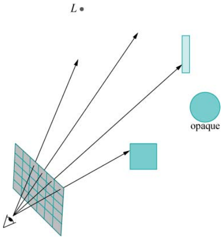
Light

Shading

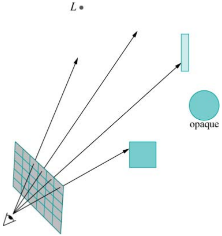
Shading

Shading = find color values at pixels of screen (when rendering a virtual 3D scene).



Shading

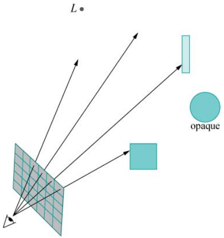
Shading = find color values at pixels of screen (when rendering a virtual 3D scene).



Same as finding color value for the closest triangle on the ray of the pixel (assuming this is an opaque object, and air is clear).

Shading

Shading = find color values at pixels of screen (when rendering a virtual 3D scene).



Same as finding color value for the closest triangle on the ray of the pixel (assuming this is an opaque object, and air is clear).

Core objective: Find color values for intersection of a ray with a triangle.

Shading

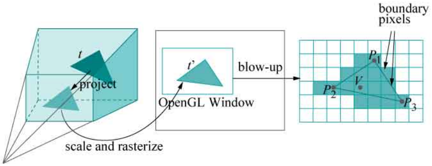
Core objective: Find color values for point at intersection of a ray with a triangle.

Shading

Core objective: Find color values for point at intersection of a ray with a triangle.

Recall:

- Rendering is triangle-driven (`foreach triangle: render`).
- Triangles are simply (triples of) vertices until rasterization phase, where pixels of the triangle are found from pixels of the vertices.



So the actual rays are determined in the rasterization phase.

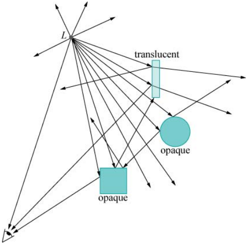
Modeling Light

Core objective: Find color values for intersection of a ray with a triangle.

Modeling Light

Core objective: Find color values for intersection of a ray with a triangle.

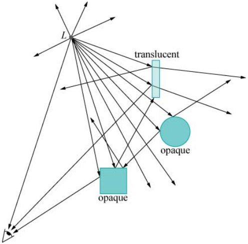
Model physical light (photons)



Modeling Light

Core objective: Find color values for intersection of a ray with a triangle.

Model physical light (photons)



Photons are

- Emitted from ligth sources.
- Reflected, absorbed, re-emitted, transmitted when hitting objects.

Modeling Light

Highly complex physical proces. Zillions of photons.

Can only be modeled to a certain degree mathematically (ongoing research expands on the available models).

Modeling Light

Highly complex physical proces. Zillions of photons.

Can only be modeled to a certain degree mathematically (ongoing research expands on the available models).

(Figure by Jason Jacobs)

Modeling Light

Realtine rendering additionally has severe time constraints. Framerate $\sim 30/\text{sec}$, screen size $\sim 10^6$ pixels $\Rightarrow$ few GPU cycles available for calculation per ray.

Modeling Light

Realtine rendering additionally has severe time constraints. Framerate $\sim 30/\text{sec}$, screen size $\sim 10^6$ pixels $\Rightarrow$ few GPU cycles available for calculation per ray.

Hence, realtime rendering uses very rough models.

Modeling Light

Realtine rendering additionally has severe time constraints. Framerate $\sim$ 30/sec, screen size $\sim 10^6$ pixels $\Rightarrow$ few GPU cycles available for calculation per ray.

Hence, realtime rendering uses very rough models.

Today: the classic model (Phong's lighting model, 1975) built into OpenGL. Very heuristic (skimpy physical backing).

Modeling Light

Realtine rendering additionally has severe time constraints. Framerate $\sim$ 30/sec, screen size $\sim 10^6$ pixels $\Rightarrow$ few GPU cycles available for calculation per ray.

Hence, realtime rendering uses very rough models.

Today: the classic model (Phong's lighting model, 1975) built into OpenGL. Very heuristic (skimpy physical backing).

More advanced models: use programmable GPU (shaders = programs for light calculations, see Ch. 20).

Modeling Light

Realtine rendering additionally has severe time constraints. Framerate $\sim$ 30/sec, screen size $\sim 10^6$ pixels $\Rightarrow$ few GPU cycles available for calculation per ray.

Hence, realtime rendering uses very rough models.

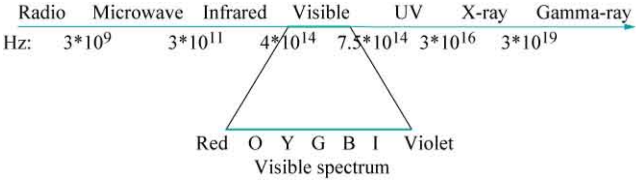
Today: the classic model (Phong's lighting model, 1975) built into OpenGL. Very heuristic (skimpy physical backing).

More advanced models: use programmable GPU (shaders $=$ programs for light calculations, see Ch. 20).

Actually, on modern programmable GPUs, the classic pipeline is just a default shader program. (Used to be hardwired into GPUs).

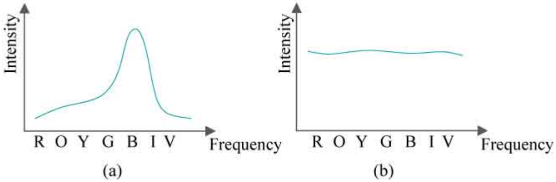
Color

Photons/light waves have frequencies:



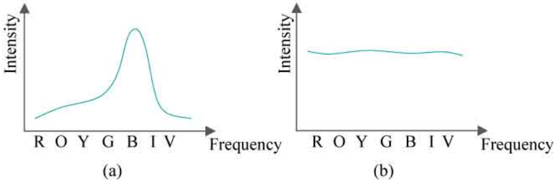
Color

The lights following a ray has in real life a spectrum:



Color

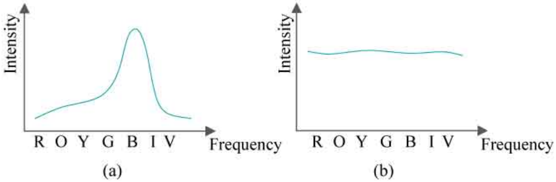
The lights following a ray has in real life a spectrum:



The eye sees colors by ligth-sensitive cells called cones. Three types of cone cells, with different sensitivity to various wavelengths. Peaks of sensitiviy in red, green, blue parts of spectrum, respectively.

Color

The lights following a ray has in real life a spectrum:

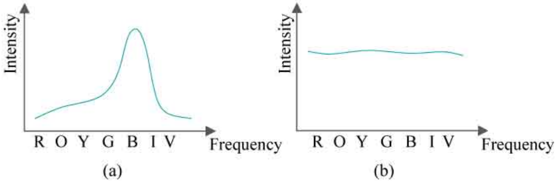


The eye sees colors by ligth-sensitive cells called cones. Three types of cone cells, with different sensitivity to various wavelengths. Peaks of sensitiviy in red, green, blue parts of spectrum, respectively.

So input spectrum in ray ⇒ three-tuple output from each cone-triple to brain. Different spectra can give same output to brain.

Color

The lights following a ray has in real life a spectrum:



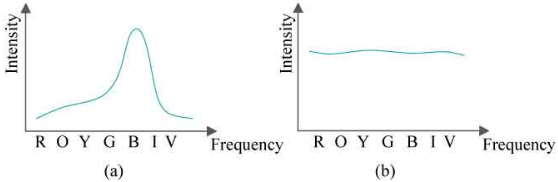
The eye sees colors by ligth-sensitive cells called cones. Three types of cone cells, with different sensitivity to various wavelengths. Peaks of sensitiviy in red, green, blue parts of spectrum, respectively.

So input spectrum in ray $\Rightarrow$ three-tuple output from each cone-triple to brain. Different spectra can give same output to brain.

On computer displays: use mix of (monocromatic) red, green, blue to stimulate cones and control the eye/brain's color perception.

Color

The lights following a ray has in real life a spectrum:



The eye sees colors by ligth-sensitive cells called cones. Three types of cone cells, with different sensitivity to various wavelengths. Peaks of sensitiviy in red, green, blue parts of spectrum, respectively.

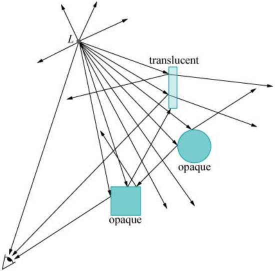
So input spectrum in ray ⇒ three-tuple output from each cone-triple to brain. Different spectra can give same output to brain.

On computer displays: use mix of (monocromatic) red, green, blue to stimulate cones and control the eye/brain's color perception.

Hence, displays (and hence OpenGL) work with (R,G,B)-tuples as color values. [Or four-tuples, if alpha/transparency information is included.]

Lightning Models

- Define virtual lights.
- Define light/surface interactions.



Virtual Lights in OpenGL

- Directional: light direction same for all points in scene (light emits infinitely far from scene—think sun).

Virtual Lights in OpenGL

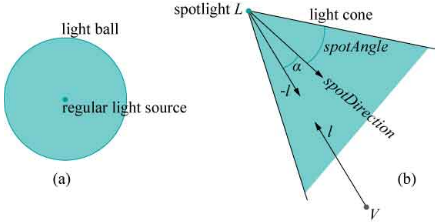
- Directional: light direction same for all points in scene (light emits infinitely far from scene—think sun).
- Positional: light emits from a 3D point in the scene. Light direction varies for different points in the scene.

Virtual Lights in OpenGL

- Directional: light direction same for all points in scene (light emits infinitely far from scene—think sun).
- Positional: light emits from a 3D point in the scene. Light direction varies for different points in the scene.
- Spot: like positional, but with cone restricting light emission. Attenuation factor towards side of cone: $(\cos\alpha)^h$

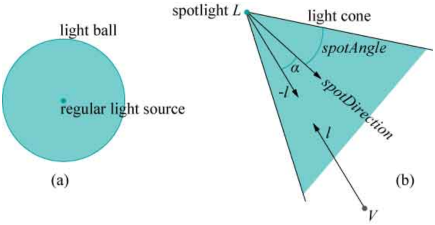
Virtual Lights in OpenGL

- Directional: light direction same for all points in scene (light emits infinitely far from scene—think sun).
- Positional: light emits from a 3D point in the scene. Light direction varies for different points in the scene.
- Spot: like positional, but with cone restricting light emission. Attenuation factor towards side of cone: $(\cos\alpha)^h$



Virtual Lights in OpenGL

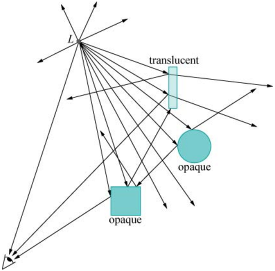
- Directional: light direction same for all points in scene (light emits infinitely far from scene—think sun).
- Positional: light emits from a 3D point in the scene. Light direction varies for different points in the scene.
- Spot: like positional, but with cone restricting light emission. Attenuation factor towards side of cone: $(\cos\alpha)^h$



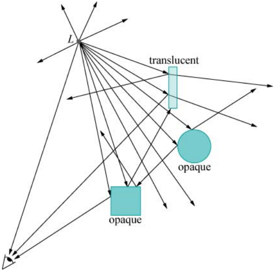
Attenuation factor for positional/spot lights (d = distance from surface point to light position):

$$\frac{1}{a + b \cdot d + c \cdot d^2}$$

Phongs Lightning Model



Phongs Lightning Model



- Models only opaque objects.
- Models only one level of light/surface interactions (except ambient term, see below).
- Light/surface interaction is modeled by two simple submodels, diffuse and specular term.
- Models indirect light effects very crudely (ambient term).
- Light actually generated at surface can be added (emissive term).
- Occlusion is not modeled (all objects see all lights).

Generic Material and Light Interaction

Surfaces (materials) and light has color.

Generic Material and Light Interaction

Surfaces (materials) and light has color.

Light: intensity value in $[0, 1]$ for each of the three RGB-channels. One triple for each light.

Generic Material and Light Interaction

Surfaces (materials) and light has color.

Light: intensity value in $[0, 1]$ for each of the three RGB-channels. One triple for each light.

Material: scaling factor in $[0, 1]$ for each of the three RGB-channels. One triple for each vertex in each primitive.

Generic Material and Light Interaction

Surfaces (materials) and light has color.

Light: intensity value in $[0, 1]$ for each of the three RGB-channels. One triple for each light.

Material: scaling factor in $[0, 1]$ for each of the three RGB-channels. One triple for each vertex in each primitive.

Basic interaction:

$$\text{(light intensity value)} \times \text{(material attenuation factor)}.$$

Generic Material and Light Interaction

Surfaces (materials) and light has color.

Light: intensity value in $[0, 1]$ for each of the three RGB-channels. One triple for each light.

Material: scaling factor in $[0, 1]$ for each of the three RGB-channels. One triple for each vertex in each primitive.

Basic interaction:

$$\text{(light intensity value)} \times \text{(material attenuation factor)}.$$

(Note: multiplication performed separately on each of the three RGB-channels).

Generic Material and Light Interaction

Surfaces (materials) and light has color.

Light: intensity value in $[0, 1]$ for each of the three RGB-channels. One triple for each light.

Material: scaling factor in $[0, 1]$ for each of the three RGB-channels. One triple for each vertex in each primitive.

Basic interaction:

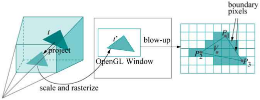
$$\text{(light intensity value)} \times \text{(material attenuation factor)}.$$

(Note: multiplication performed separately on each of the three RGB-channels).

(Note: actually one light intensity triple (lights) and one attenuation factor (material) for each of the terms ambient, diffuse, specular (see later). But this flexibility often not used/needed.)

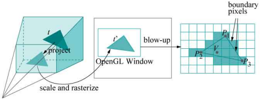
Shading models

So we have information in each vertex. How spread color calculation over entire triangle pixels?



Shading models

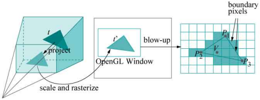
So we have information in each vertex. How spread color calculation over entire triangle pixels?



- Flat shading: Color calculated for one point is used for entire triangle.

Shading models

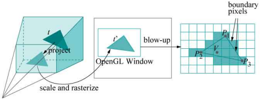
So we have information in each vertex. How spread color calculation over entire triangle pixels?



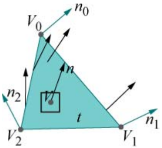
- Flat shading: Color calculated for one point is used for entire triangle.
- Smooth shading (aka. Gouraud shading): Colors calculated for three vertices are interpolated across the entire triangle (individually for each RGB-channel).

Shading models

So we have information in each vertex. How spread color calculation over entire triangle pixels?

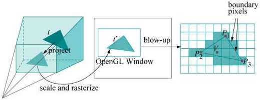


- **Flat shading**: Color calculated for one point is used for entire triangle.
- **Smooth shading** (aka. Gouraud shading): Colors calculated for three vertices are interpolated across the entire triangle (individually for each RGB-channel).
- **Phong shading**: Color calculation done for all points of pixels.

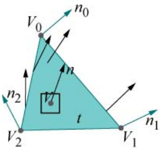


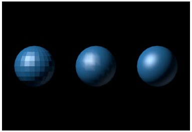
Shading models

So we have information in each vertex. How spread color calculation over entire triangle pixels?



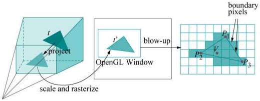
- Flat shading: Color calculated for one point is used for entire triangle.
- Smooth shading (aka. Gouraud shading): Colors calculated for three vertices are interpolated across the entire triangle (individually for each RGB-channel).
- Phong shading: Color calculation done for all points of pixels.



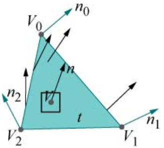


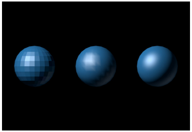
Shading models

So we have information in each vertex. How spread color calculation over entire triangle pixels?



- Flat shading: Color calculated for one point is used for entire triangle.
- Smooth shading (aka. Gouraud shading): Colors calculated for three vertices are interpolated across the entire triangle (individually for each RGB-channel).
- Phong shading: Color calculation done for all points of pixels.





Calculation time increases down the list. Phong shading needs programmable shaders (not part of OpenGL).

Diffuse Term in Phong

L'Amberts law [1760] for perfectly scattered light (100% matte surfaces).

Diffuse Term in Phong

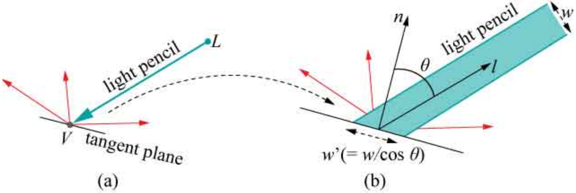
L'Amberts law [1760] for perfectly scattered light (100% matte surfaces).

- Light influx per area on surface depends on angle θ between light vector (light direction) and surface normal at point. Dependency/attenuation is factor of $\cos\theta$.
- Light is scattered equally from point in all directions ($\Rightarrow$ eye ray vector does not matter).

Diffuse Term in Phong

L'Amberts law [1760] for perfectly scattered light (100% matte surfaces).

- Light influx per area on surface depends on angle θ between light vector (light direction) and surface normal at point. Dependency/attenuation is factor of $\cos\theta$.
- Light is scattered equally from point in all directions ($\Rightarrow$ eye ray vector does not matter).



Specular Term in Phong

Models highlights/shininess using heuristic formula.

Specular Term in Phong

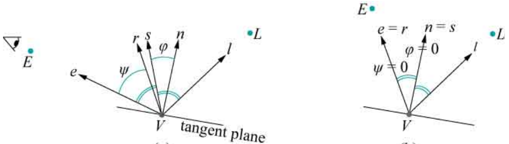
Models highlights/shininess using heuristic formula.

Depends on light vector (light direction), eye ray vector, and surface normal at point.

Specular Term in Phong

Models highlights/shininess using heuristic formula.

Depends on light vector (light direction), eye ray vector, and surface normal at point.

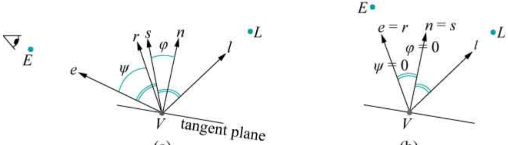


(a) (b)

Specular Term in Phong

Models highlights/shininess using heuristic formula.

Depends on light vector (light direction), eye ray vector, and surface normal at point.



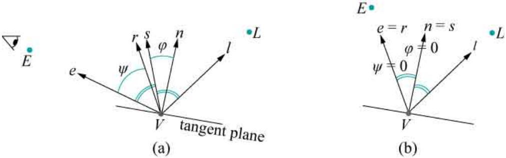
(a) (b)

Let ϕ be angle between halfway vector $s = (l + e)$ and the normal vector n.

Specular Term in Phong

Models highlights/shininess using heuristic formula.

Depends on light vector (light direction), eye ray vector, and surface normal at point.



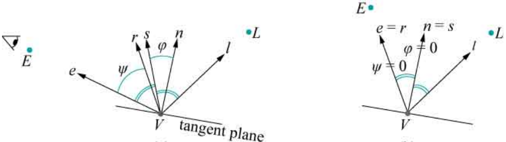
Let ϕ be angle between halfway vector $s = (l + e)$ and the normal vector n.

Attenuation factor: $(\cos\phi)^f$

Specular Term in Phong

Models highlights/shininess using heuristic formula.

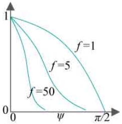
Depends on light vector (light direction), eye ray vector, and surface normal at point.

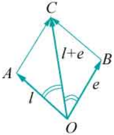


(a) (b)

Let ϕ be angle between halfway vector $s = (l + e)$ and the normal vector n.

Attenuation factor: $(\cos\phi)^f$





Ambient Term(s) in Phong

Models indirect light (crudely).

Ambient Term(s) in Phong

Models indirect light (crudely).

“Everywhere is some light”.

Ambient Term(s) in Phong

Models indirect light (crudely).

“Everywhere is some light”.

Light calculation does not depend points normal vector, direction of eye ray, direction of light (except for spot attenuation).

Ambient Term(s) in Phong

Models indirect light (crudely).

“Everywhere is some light”.

Light calculation does not depend points normal vector, direction of eye ray, direction of light (except for spot attenuation).

More precisely: there is one global ambient term, plus one ambient term for each light. The latter allows for individually colored light, and distance and spot attenuation. Besides distance and spot attenuation, the calculation is just the basic multiplication of material and light.

Emissive Term in Phong

This is another material value signifying if light is actually created at the point.

This is just an RGB-triple added to the result of the rest of the calculations for the point.

Emissive Term in Phong

This is another material value signifying if light is actually created at the point.

This is just an RGB-triple added to the result of the rest of the calculations for the point.

Note: the “emitted light” is only used for the color value of the pixel (of the ray hitting the point). It is not taken into consideration when calculating color values at other points.

Emissive Term in Phong

This is another material value signifying if light is actually created at the point.

This is just an RGB-triple added to the result of the rest of the calculations for the point.

Note: the "emitted light" is only used for the color value of the pixel (of the ray hitting the point). It is not taken into consideration when calculating color values at other points.

Thus, having a bulb both visible and giving light in a scene will involve:

- Creating polygons for the bulb and setting their emissive properties to non-zero (probably close to (1,1,1)).
- Creating a virtual light at the center of the bulb.

Lighting Equation

Add basic interactions between material and light as follows:

- One term for global ambient light.
- For each light defined: add terms for per-light ambient term, diffuse term, and specular term (with distance and spot attenuation where appropriate).

This is done once for each RGB-channel. A resulting value above 1.0 is just truncated to 1.0.

Lighting Equation in Math

$$\begin{aligned}
\text{channel value} &= \text{emissive\_term} \\
&+ \text{amb\_material} \times \text{amb\_global} \\
&+ \textstyle\sum_{\text{all lights}} \text{dist\_attenuation} \times \text{spot\_attenuation} \times \\
&\quad (\text{amb\_material} \times \text{amb\_light} + \\
&\quad \max\{\vec{L} \cdot \vec{n}, 0\} \times \text{diff\_material} \times \text{diff\_light} + \\
&\quad (\max\{\vec{s} \cdot \vec{n}, 0\})^f \times \text{spec\_material} \times \text{spec\_light})
\end{aligned}$$