Lighting

- So far, we assumed that each surface has a solid colour (or a blend). The colour is completely dependent on the surface.
- In reality, colour depends on both the properties of the surface and the light source.
- A flat surface does not look uniform throughout generally.
- We now look at how to model lighting effects.
- Realistic modelling of lighting requires knowledge in physics: many approximations and simplifications in computer graphics

Global Lighting

- In “real world”, there can be many light sources.
- Some objects can emit light.
- Other objects can absorb, scatter, and/or reflect light.
- What we see is how much light is scattered and reflected to the viewer from a particular point.
- When light bounces off a surface, it can then interact with something else.
- Need to trace light recursively.
- Computationally intensive.

Local Lighting

- To simplify calculations, we compute local lighting.
- For each position and for each light source, compute amount of light arriving at the viewer.
- Consider only light directly from light sources.

Surfaces

There are three main types of surfaces (can be a mixture of these):

- Specular: most of the light is reflected or scattered in a narrow range of angles. Appears shiny.
- Diffuse: scattered in all directions. Appears matte or flat.
- Translucent: some light can pass through the surface. Can also bend light (refraction).

Light Sources

- Light is not simply an intensity. It is an intensity function depending on wavelength (colour).
- We model a light source with a three component intensity

$$I = (I_r, I_g, I_b)$$

  representing the intensity of the RGB components of the light.
- There may also be a direction.
- Four basic types: ambient, point, spotlights, distant light.

Ambient Light

- There is often "general" lighting (e.g. sun, overhead lights, etc.) that provide somewhat uniform lighting in the whole scene.
- Modelling each such source can be computationally intensive.
- Instead, we define a general uniform lighting as the ambient light and apply this uniformly everywhere:

$$I_a = (I_{ar}, I_{ag}, I_{ab})$$

- Each point in the scene receives the same ambient light.

Point Light Sources

- An ideal point source emits light equally in all direction.
- It is located at a point p_0:

$$I(p_0) = (I_r(p_0), I_g(p_0), I_b(p_0))$$

- For any given point p, the amount of light received from p_0 depends on the distance (squared):

$$i(p, p_0) = \frac{1}{\|p - p_0\|^2} I(p_0)$$

- It is simple but may not be realistic.
- Often bright or dark, but not smooth. Larger light sources can smooth out shadows, for example.

Spotlights

- Light is projected in a cone. The width of the cone is determined by an angle.
- Usually light is more concentrated at the center of the cone, and decreases moving away from center.
- If $\vec{u}$ is unit vector for the direction of the spotlight, and $\vec{v}$ is the unit vector from spotlight to object, then $\theta = \cos^{-1}(\vec{u} \cdot \vec{v})$ can be used to determine intensity received.

Distant Light Sources

- We often need to compute a vector from light source to a point.
- If a light source is very far away, this vector can be considered constant for all points.
- Direction is simply represented as a vector (same for all points).

Phong Lighting Model

- Four vectors are needed to compute the colour of each point p:
  - $\vec{n}$: normal vector to the surface at point p
  - $\vec{v}$: vector from point p to the viewer (center of projection for perspective viewing)
  - $\vec{l}$: vector from point p to the light source (assuming point light source)
  - $\vec{r}$: direction of a perfectly reflected light from $\vec{l}$ (computed from $\vec{n}$ and $\vec{l}$)
- All vectors are normalized to have length 1.
- Each light source has three components: ambient, diffuse, and specular, each with RGB components.

- For each light source, this can be specified as a matrix:

$$L = \begin{bmatrix} L_{ra} & L_{ga} & L_{ba} \\ L_{rd} & L_{gd} & L_{bd} \\ L_{rs} & L_{gs} & L_{bs} \end{bmatrix}$$

- We can use matrices or three different vectors (ambient, diffuse, specular) in implementation.
- We can also use 4-component RGBA colours.

Phong Lighting Model

- Each point also has a reflective value (in $[0,1]$) for each type of light and colour to determine the proportion of light that is reflected:

$$R = \begin{bmatrix} R_{ra} & R_{ga} & R_{ba} \\ R_{rd} & R_{gd} & R_{bd} \\ R_{rs} & R_{gs} & R_{bs} \end{bmatrix}$$

- We can also use 4-component RGBA colours.
- The light seen for each component is simply the corresponding light source multiplied by the reflection value.
- The amount of light seen is the combination of the three components. For example, for the red component:

$$I_r = R_{ra} \cdot L_{ra} + R_{rd} \cdot L_{rd} + R_{rs} \cdot L_{rs} + I_{ar}$$

I_{ar} is an optional global ambient term

- If there are multiple light sources, need to add the contribution from each light source.
- To simplify presentation, we will remove the r, g, b subscripts

Ambient Reflection

- Each point in the surface has the same reflection coefficient $k_a \in [0, 1]$.
- $I_a = k_a \cdot L_a$.
- L_a can be any individual light sources, or global ambient term.
- The RGB components of k_a can describe the colour of the surface (assuming white light)

Diffuse Reflection

- Ideally, diffuse reflector scatters light equally in all directions.
- Also called Lambertian surfaces.
- How much light is scattered depends on the angle between the surface and light source: brightest if it is perpendicular, darkest if it is parallel:

  $$R_d = k_d \cdot \cos\theta = k_d(\vec{l} \cdot \vec{n})$$

  where θ is the angle between $\vec{l}$ and $\vec{n}$, and k_d is some constant (property of surface)
- The intensity from diffuse term is:

  $$I_d = k_d(\vec{l} \cdot \vec{n}) \cdot L_d$$

- If we wish to incorporate the distance to the light: divide by $a + bd + cd^2$ for some constants a, b, c and distance d.

- If the light source is below the surface, this can be negative (set to 0 instead).

Specular Reflection

- Allows for "shininess"
- Amount of light reflected depends on how close we are to the reflected angle.
- Drops off quickly as we move further.

$$I_s = k_s \cdot L_s \cos^{\alpha} \phi$$

- k_s is property of the material
- α controls how quickly it drops off (shininess). A value of 100–500 correspond to most metallic surfaces.
- ϕ: angle between $\vec{r}$ and $\vec{v}$, so

$$\cos\phi = \vec{r} \cdot \vec{v}$$

- Note: if the dot product is negative, $I_s = 0$ because the light is on the other side of the surface

Blinn-Phong Model

- $$I_s = k_s \cdot L_s(\vec{r} \cdot \vec{v})^\alpha$$
- ϕ: angle between $\vec{r}$ and $\vec{v}$.
- $\vec{r}$ has to be computed for each point
- One approximation: use the "halfway vector":

$$\vec{H} = \frac{\vec{v} + \vec{l}}{\|\vec{v} + \vec{l}\|}$$

- Replace $\vec{r} \cdot \vec{v}$ by $\vec{n} \cdot \vec{H}$ and increase α (heuristically $4\times$):

$$I_s = k_s \cdot L_s(\vec{n} \cdot \vec{H})^{4\alpha}$$

- If $\vec{n} \cdot \vec{H} < 0$, set $I_s = 0$.
- Why can this be faster? If we assume camera and light are "far away",

then $\vec{v}$ and $\vec{l}$ can be assumed to be constant on entire surface.

Normal Vectors Computations

- Sometimes we can just specify the normals by hand.
- If we have a triangle specified by p_1, p_2, p_3 in counterclockwise order, then
$$\vec{n}' = (p_2 - p_1) \times (p_3 - p_1)$$
and
$$\vec{n} = \frac{\vec{n}'}{\|\vec{n}'\|}$$
- How does $\vec{n}$ gets transformed by model-view matrix? If M is the model-view matrix (only the top-left 3×3 portion), then transform $\vec{n}$ by $M \cdot \vec{n}$ does not work (e.g. scaling)!
- Need to transform $\vec{n}$ by the Normal matrix:
$$N = (M^{-1})^T$$
(see textbook for the math behind this)

Gourad vs Phong Shading

- Gourand Shading: determine the colours at each vertex. The fragments inside the primitive are interpolated from the vertices.
- Can be done in vertex shader.
- Phong Shading: determine the normal vectors at each vertex. The fragments interpolate the normal vectors and compute colours.
- Can be done in fragment shader.
- If a surface is flat and all vertices have same normal vector, these two approaches are similar (not exactly the same)
- Otherwise, Phong shading produces more realistic results.

OpenGL Implementation

- Represent all properties of light and surfaces (I_a, I_d, I_s, k_a, k_d, k_s as a vector of RGB components (can also add alpha).
- Specify light position as a 4-diemsnional vector.
- Many libraries will allow componentwise vector multiplication:

$$I_a * k_a = (I_{ar}k_{ar}, I_{ag}k_{ag}, I_{ab}k_{ab})$$

- In our matrix/vector library:
  - `normalize` scales a vector to unit length
  - `dot` computes dot product
  - `cross` computes cross product
  - `Normal` computes the normal matrix
- Pass them as uniform variables to vertex shader.

OpenGL Implementation

In vertex shader:

```
position = view * model * vPosition;
gl_position = projection * position;
N = Normal * aNormal;
      // aNormal passed in as normal in model space
```

Make the normal vector and position part of the output variable to the fragment shader (interpolated).

OpenGL Implementation

In fragment shader:

```
L = normalize(lightPosition - position)
V = normalize(-position);
        // viewer is at origin
H = normalize(V + L);
```

Light and surface properties can be passed to fragment shader as uniform variables from application.

Compute ambient, diffuse and specular terms separately and add them.

Ray Tracing

- Ray tracing is a technique to perform global shading computation.
- For each pixel, shoot a ray towards the centre of projection and see what is the first object it hits. Then bounce it towards a light source (stength adjusted by angles)
- If the object is reflective, recursively trace the reflected ray (up to some limit)
- More realistic scenes, can handle shadows and reflections.
- Computationally more intensive.