Review: Last Week

- We have learnt that the order of how we draw objects in 3D can have an influence on how the final image looks
  - Depth-sort
  - Z-buffer
  - Transparency
  - Orientation of triangle (order of vertices) influences lighting, culling
Review: The Z-Buffer

- Several fragments created by rasterizing primitives can map to the same target pixel
- Compare and update value in the z-buffer (resolve depth-ordering problem per fragment)
Review: Alpha Blending

- Most common alpha-compositing modes are order dependent
- Back-to-front rendering employed for transparent geometry
Adding Light to the Scene

• 3D objects look flat if all of their sides are colored the same.
• In reality, we see shades of colors on 3D objects, even if they consist of a single homogeneous material.
• The reason for this is lighting.

• In graphics, algorithms that compute shades of color based on lighting are often referred to as object shading techniques.
Local and Global Lighting

- **Local** lighting techniques compute shades based on
  - Local surface geometry (normal, etc.)
  - Material properties (reflectance, color, etc…)
  - Light properties (position, color, etc…)
  - Optional: Viewing direction or position

- **Global** lighting techniques additionally take other objects/surfaces of the scene into consideration
Local and Global Lighting

Local lighting only

Approximation of global lighting (here: ambient occlusion)

Better:
- Raytracing
- Radiosity
Local Lighting

• We focus on local lighting
  • Implemented in fixed function pipeline
  • Shaded color can be computed for individual vertices or pixels in isolation (fragment shader!)

• Materials interact with light
  • Specular surfaces (mirror, shiny metals, …)
  • Diffuse surfaces (painted walls, …)
  • Translucent surfaces (water, glass, …)
Types of Lights

- Lights have a **color** (RGB intensities)

**Ambient lights** are designed to scatter light across a scene uniformly. Approximated by global ambient intensities (RGB) – avoids completely dark surfaces.

- Lights have geometric properties:
  - **Point light** (light emitted from a position)
  - **Spotlight** (emitted intensity depends on angle)
  - **Directional light** (point light at infinity – direction instead of position)
The Phong Lighting Model

- How do we compute shading from light and local material properties?
- Phong lighting model uses 4 vectors
- Efficient approximation of lighting

\[ r = \text{direction of ideal reflection} \]
The Phong Lighting Model

- Phong lighting model uses **ambient**, **diffuse**, and **specular** interactions
- Light source has RGB components for each

\[
(L_{ra}, L_{ga}, L_{ba})
\]
\[
(L_{rd}, L_{gd}, L_{bd})
\]
\[
(L_{rs}, L_{gs}, L_{bs})
\]
The Phong Lighting Model

- Surfaces reflect these contributions differently based on material properties, direction of normal, etc.

\[
(R_{ra}, R_{ga}, R_{ba})
\]

\[
(R_{rd}, R_{gd}, R_{bd})
\]

\[
(R_{rs}, R_{gs}, R_{bs})
\]

- What we see from a surface is a combination of light intensities and reflection terms (red intensity, one light):

\[
I_r = R_{ra} \cdot L_{ra} + R_{rd} \cdot L_{rd} + R_{rs} \cdot L_{rs}
\]

\[
= I_{ra} + I_{rd} + I_{rs}
\]
Ambient Reflection

- Ambient light intensity is the same everywhere on a surface.
- Surface defines a an ambient reflection coefficient $k_a$

\[
R_a = k_a \quad 0 \leq k_a \leq 1
\]

\[
I_a = k_a \cdot L_a
\]
Diffuse Reflection

- Diffuse reflection is considered to be the same in all directions. **But:** “amount of light” per area matters.

$$R_d \propto \cos \alpha$$  
**Lambert’s Law**
Diffuse Reflection

- Cosines of angles can be computed with the dot product

$$\cos \alpha = l \cdot n$$

$l, n$ are unit length light and normal vectors

$$I_d = k_d \cdot (l \cdot n) L_d$$

$k_d$ reflection coefficient

$$I_d = k_d \cdot \max((l \cdot n), 0) L_d$$

$$I_d = \frac{k_d}{d(p, p_0)} \cdot \max((l \cdot n), 0) L_d$$

distance attenuation
Specular Reflection

- Specular reflection produces highlights on shiny objects
- Highlights depend on current viewing direction
Specular Reflection

- During specular reflection, light is concentrated around the direction of perfect reflection $r$

$$I_s = k_s \max((r \cdot v)^\alpha, 0)L_s$$

$\alpha$  shininess coefficient
Specular Reflection

- Shininess coefficient controls how concentrated specular highlight appears
The Phong Model

- Intensity is sum of individual reflected intensities

\[ I = \frac{1}{a + bd + cd^2} \left( k_d L_d \max(l \cdot n, 0) + k_s L_s \max((r \cdot v)^\alpha, 0) \right) + k_a L_a \]

- distance attenuation
- specular reflection
- diffuse reflection
- ambient reflection
The Blinn-Phong Model

- Phong-Model requires recalculation of $r \cdot v$
- The **Blinn-Phong** model replaces this calculation by

$$n \cdot h \quad \text{Where} \quad h = \frac{l + v}{|l + v|}$$

is the vector half way in between $l$ and $v$

- If $v$ lies in the same plane as $l,n,r$ then the angle between $n$ and $h$ is simply half of the angle between $r$ and $v$
The Blinn-Phong Model

- The **Blinn-Phong** model is standard in OpenGL’s fixed function pipeline

- Programmable fragment shaders allow us to perform more accurate shading computations in real time
Shading of Polygonal Geometry

- Object normals are typically precomputed for efficiency

\[
\begin{pmatrix}
  n_x \\
  n_y \\
  n_z \\
  0
\end{pmatrix}
\]

- Normals are unit length vectors
- Normals are orthogonal to the local surface tangent
- Cross product allows direct computation of triangle normals
Shading of Polygonal Geometry

- For polygonal geometry we can distinguish three types of shading:
  - Flat shading
  - Gouraud Shading (smooth shading)
  - Phong Shading

- Flat shading and smooth shading compute shades per-vertex and interpolate the result across the triangle to obtain per-fragment colors
- Shaders allow us to perform per-fragment shading
Flat Shading

• For a fixed light $n, l, r$ are constant per triangle
• Assuming that the viewer is far away from the scene, $v$ is constant over the triangle
• Shading needs to be computed only once per triangle
Gouraud/Smooth Shading

- For smooth shading, normals cannot stay constant over a triangle
- Triangles can have three different normals (one at each vertex)
- We compute normals for vertices as:

\[ n_v = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|} \]
Gouraud/Smooth Shading

- Shading colors are computed per-vertex.
- Resulting colors are interpolated across triangles

looks better, but still not optimal for low resolution meshes
Phong Shading

- With shaders, we can compute **per-fragment shading**
- Like all vertex attributes, normals are interpolated across triangles
- Now we can use an individual normal for every fragment

per-fragment shading with specular highlight