Illumination models

• It won’t look 3-D without lighting
• Part of geometry processing
  – Can also be part of rasterization

• Illumination types
  – Ambient
  – Diffuse
  – Specular
  – Emissive

Local vs. global illumination

• Local illumination
  – Direct illumination: Light shines on all objects without blocking or reflection
  – Used in most games

• Global illumination
  – Indirect illumination: Light bounces from one object to other objects
  – Adds more realism (non real-time rendering)
  – Computationally much more expensive
  – Ray tracing, radiosity

Common light sources

- Directional Light (Infinitely far away)
- Point Light (Emit in all directions)
- Spot Light (Emit within a cone)
Illumination: ambient lighting

- Not created by any light source
- A constant lighting from all directions
- Contributed by scattered light in a surrounding

\[ C_{ambient} = M_{ambient} \otimes L_{ambient} \]

Illumination: diffuse lighting

- Light sources are given
- Assume light bounces in all directions

Reflected light intensity calculation

- Reflectivity = the entry angle
- Use Lambert’s cosine Law

\[ C_{diff} = \max( -\bar{L} \cdot \hat{n}, 0 ) \cdot ( M_{diff} \otimes L_{diff} ) \]

Ambient + diffuse lighting

\[ C_{diff} = M_{ambient} \otimes L_{ambient} + \max( -\bar{L} \cdot \hat{n}, 0 ) \cdot ( M_{diff} \otimes L_{diff} ) \]
Illumination: specular lighting

- Create shining surface (surface perfectly reflects)
- Viewpoint dependent

Jim Blinn’s specular model

\[ C_{\text{spec}} = (\max(n \cdot \vec{H}, 0))^s \cdot M_{\text{spec}} \odot L_{\text{spec}} \]

- A (usually) more computationally efficient approximation of the Phong specular model that uses the reflective vector \( \vec{R} \)
- "S" controls the bright region around surface

Specular brightness effect

\[ C_{\text{spec}} = (\max(n \cdot \vec{H}, 0))^s \cdot M_{\text{spec}} \odot L_{\text{spec}} \]

where \( \vec{n} \cdot \vec{H} = |\vec{H}| \cos \theta \)

Role of brightness parameter S

\[ C_{\text{spec}} = (\max(n \cdot \vec{H}, 0))^s \cdot M_{\text{spec}} \odot L_{\text{spec}} \]
### Specular lighting effect

- A larger $S$ shows more concentration of the reflection

### Illumination: emissive lighting

- Color is emitted by the material only

\[
C_{\text{col}} = C_e + M_e \otimes L_e + \max(-\vec{L} \cdot \vec{n}, 0) \cdot (M_d + L_d) + (\max(\vec{n} \cdot \vec{H}, 0))^n \cdot M_s \otimes L_s
\]

### Common light sources (revisited)

- **Directional Light** (Infinitely far away)
  - Emit in all directions
- **Point Light**
  - Emit in all directions
- **Spot Light**
  - Emit within a cone

### Light source properties

- **Position**
- **Range**
  - Specifying the visibility
- **Attenuation**
  - The farther the light source, the dimmer the color

\[
\text{Att} = a_0 + a_1 \cdot d + a_2 \cdot d^2
\]

\[
C_{\text{col}} = C_f + M_s \otimes L_s + \frac{\max(-\vec{L} \cdot \vec{n}, 0) \cdot (M_d + L_d) + (\max(\vec{n} \cdot \vec{H}, 0))^n \cdot M_s \otimes L_s}{\text{Att}}
\]
Spotlight effect

\[ \text{spot} = (\max(\cos \alpha, 0))^f \]
\[ \text{spot} = (\max(\vec{L} \cdot \vec{d}, 0))^f \]

where \( f \) is the falloff factor

\[ C_{\text{whatever}} = \text{spot} \cdot C_{\text{whatever}} \]

- Similar in form to specular lighting (but different!)
- Falloff factor determines the fading effect of a spotlight
- \( "f" \) exponentially decreases the \( \cos(\alpha) \) value

Rasterization: shading a triangle

- Converting geometry to a raster image (i.e., pixels)
- Paint each pixel’s color (by calculating light intensity) on your display
- Gouraud shading; intensity interpolation of vertices

Gouraud shading

- Scan conversion algorithm

Comparison of shading methods

- Gouraud shading supported by (even old) 3-D graphics hardware
- Phong shading
  - Requires generating per-pixel normals to compute light intensity for each pixel, not efficient for games
  - Can be done on modern GPUs using Cg or HLSL

Source: Michal Necasek
Double buffering

- Display refreshes at 60 ~ 75 Hz
- Rendering could be “faster” than the refresh period
- Too fast leads to:
  - Frames not shown
- Too slow leads to:
  - New and old frame mixed
  - Flickering
- Solution:
  - Double or multiple buffering

The Z-buffer

- Also called depth buffer
- Draws the pixel which is nearest to the viewer
- Number of the entries corresponding to the screen resolution (e.g. 1024x768 should have a 768k-entry Z-buffer)
- Granularity matters
  - 8-bit never used
  - 16-bit z value could generate artifacts

Fog effects

- Provide depth cue
  - Simulate weather condition
  - Avoid popping effect
- Color blending

\[
\text{color} = (1 - f) \cdot \text{Color}_{\text{wall}} + f \cdot \text{Color}_{\text{fog}}
\]

\[
f = \frac{\text{dist(eye, vertex)} - \text{fog start}}{\text{fog stop} - \text{fog start}}
\]

- Calculate distance
- Calculate intensity of vertex color based on distance
  - Color blending
  - Linear density, exponential density
- Blending color schemes
  - Per-vertex (then interpolate pixels), less expensive
  - Per-fragment basis (NVIDIA hardware), better quality

Aliasing

- Jagged line (or staircase)
- Can be improved by increasing resolution (i.e. more pixels)
Anti-aliasing by multisampling
(Example: Supersampling)

- GPU samples multiple locations for a pixel
- Several different methods
  - e.g., grid (as shown), random, GeForce’s quincunx
- Downside
  - Blurry image
  - Increased memory (e.g., z-buffer) storage for subpixel information

Actual Screen Pixels

3x3 Virtual Pixels (Bartlett window)

(255, 255, 255)
(255, 159, 159)
(255, 0, 0)

Visualizing anti-aliasing example

Anti-aliasing example

No MSAA
With MSAA

Ideal