THE RENDERING PIPELINE

Vertices and attributes

Vertex Shader
- Modelview transform
- Per-vertex attributes

Vertex Post-Processing
- Viewport transform
- Clipping

Rasterization
- Scan conversion
- Interpolation

Fragment Shader
- Texturing/...
- Lighting/shading

Per-Sample Operations
- Depth test
- Blending

Framebuffer
LIGHTING/SHADING

- Goal
  - Model the interaction of light with surfaces to render realistic images
  - Generate per (pixel/vertex) color
FACTORS

• Light sources
  • Location, type & color
• Surface materials
  • How surfaces reflect light
• Transport of light
  • How light moves in a scene
• Viewer position
FACTORS

• Light sources
  • Location, type & color

• Surface materials
  • How surfaces reflect light

• Transport of light
  • How light moves in a scene

• Viewer position

• How can we do this in the pipeline?
**ILLUMINATION MODELS/ALGORITHMS**

Local illumination - Fast
Ignore real physics, approximate the look
Interaction of each object with light
• Compute on surface (light to viewer)

Global illumination - Slow
Physically based
Interactions between objects
THE BIG PICTURE (BASIC)

• Light: energy in a range of wavelengths
  • White light – all wavelengths
  • Colored (e.g. red) – subset of wavelengths
• Surface “color” – reflected wavelength
  • White – reflects all lengths
  • Black – absorbs everything
  • Colored (e.g. red) absorbs all but the reflected color
• Multiple light sources add (energy sums)
MATERIALS

• Surface reflectance:
  • Illuminate surface point with a ray of light from different directions
  • How much light is reflected in each direction?
BASIC TYPES

diffuse

glossy

mirror
Most surfaces exhibit complex reflectances
  • Vary with incident and reflected directions.
  • Model with combination – known as BRDF
    • BRDF: Bidirectional Reflectance Distribution Function
BRDF MEASUREMENTS/PLOTS

2D slice
Intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.
Computing Diffuse Reflection

Depends on angle of incidence: angle between surface normal and incoming light

\[ I_{\text{diffuse}} = k_d \ I_{\text{light}} \cos \theta \]

In practice use vector arithmetic

\[ I_{\text{diffuse}} = k_d \ I_{\text{light}} (n \cdot l) \]

Scalar (B/W intensity) or 3-tuple (color)

- \( k_d \): diffuse coefficient, surface color
- \( I_{\text{light}} \): incoming light intensity
- \( I_{\text{diffuse}} \): outgoing light intensity (for diffuse reflection)

NB: Always normalize vectors used in lighting

- \( n \), \( l \) should be unit vectors
DIFFUSE LIGHTING EXAMPLES

- Lambertian sphere from several lighting angles:

  ![Lambertian Sphere](image)

- need only consider angles from 0° to 90°
PHYSICS OF SPECULAR REFLECTION

- Geometry of specular (perfect mirror) reflection
  - Snell’s law

\[ n \sin \alpha = -l + 2(n \cdot l) n \]
PHYSICS OF SPECULAR REFLECTION

• Geometry of specular (perfect mirror) reflection
  • Snell’s law
  • In GLSL: use `reflect(-l,n)`

\[
\begin{align*}
  n &= \frac{r \mp l}{2n} \\
  \alpha &= \frac{r}{l}
\end{align*}
\]
EMPIRICAL APPROXIMATION

• Snell’s law = perfect mirror-like surfaces
  • But ..
    • few surfaces exhibit perfect specularity
    • Gaze and reflection directions never EXACTLY coincide
  • Expect most reflected light to travel in direction predicted by Snell’s Law
• But some light may be reflected in a direction slightly off the ideal reflected ray
• As angle from ideal reflected ray increases, we expect less light to be reflected
EMPIRICAL APPROXIMATION

• Angular falloff

• How to model this falloff?
Most common lighting model in computer graphics (Phong Bui-Tuong, 1975)

\[
I_{\text{specular}} = k_s I_{\text{light}} (\cos \phi)^{n_s}
\]

\[
I_{\text{specular}} = k_s I_{\text{light}} (\mathbf{v} \cdot \mathbf{r})^{n_s}
\]

\(\phi\): angle between \(\mathbf{r}\) and view direction \(\mathbf{v}\)

\(n_s\): purely empirical constant, varies rate of falloff

\(k_s\): specular coefficient, highlight color

no physical basis, “plastic” look
PHONG EXAMPLES

varying light position

varying $n_s$
ALTERNATIVE MODEL

Blinn-Phong model (Jim Blinn, 1977)
• Variation with better physical interpretation
  • $h$: halfway vector; $r$: roughness

$$I_{\text{specular}} = k_s \cdot (h \cdot n)^{1/r} \cdot I_{\text{light}}; \text{ with } h = (l + v)/2$$
MATERIALS (LAST BIT)

- Light is **linear**
  - If multiple rays illuminate the surface point the result is just the sum of the individual reflections for each ray

\[
\sum_{p} I_p (k_d (n \cdot l_p) + k_s (r_p \cdot v)^n)
\]
MATERIALS DEMO

• (switch to Maya)
**AMBIENT LIGHT**

- Non-directional light – environment light
- Object illuminated with same light everywhere
  - Looks like silhouette
- Illumination equation  \[ I = I_a k_a \]
  - \( I_a \) – ambient light intensity
  - \( k_a \) – fraction of this light reflected from surface

![Tea Pot Image]
ILLUMINATION EQUATION (PHONG)

• If we take the previous formula and add ambient component:

\[ I_a k_a + \sum_p I_p (k_d (n \cdot l_p) + k_s (r_p \cdot v)^n) \]
LIGHT SOURCE TYPES

• Point Light
  • light originates at a point

• Directional Light (point light at infinity)
  • light rays are parallel
  • Rays hit a planar surface at identical angles

• Spot Light
  • point light with limited angles
LIGHT SOURCE TYPES

• Point Light
  • light originates at a point
  • defined by location only

• Directional Light (point light at infinity)
  • light rays are parallel
  • Rays hit a planar surface at identical angles
  • defined by direction only

• Spot Light
  • point light with limited angles
  • defined by location, direction, and angle range
WHICH LIGHTS/MATERIALS ARE USED HERE?
LIGHT SOURCE FALLOFF

- Quadratic falloff (point- and spot lights)
  - Brightness of objects depends on power per unit area that hits the object
  - The power per unit area for a point or spot light decreases quadratically with distance
ILLUMINATION EQUATION WITH ATTENUATION

- For multiple light sources:

\[
I = I_a k_a + \sum_p \frac{I_p}{A(d_p)} (k_d (n \cdot l_p) + k_s (r_p \cdot v)^n)
\]

- \(d_p\) - distance between surface and light source + distance between surface and viewer, A – attenuation function
Light has color

Interacts with object color \((r, g, b)\)

\[
I = I_a k_a
\]

\[
I_a = (I_{ar}, I_{ag}, I_{ab})
\]

\[
k_a = (k_{ar}, k_{ag}, k_{ab})
\]

\[
I = (I_r, I_g, I_b) = (I_{ar}k_{ar}, I_{ag}k_{ag}, I_{ab}k_{ab})
\]

Blue light on white surface?

Blue light on red surface?
LIGHT AND MATERIAL SPECIFICATION

• Light source: amount of RGB light emitted
  • value = intensity per channel
    e.g., (1.0,0.5,0.5)
  • every light source emits ambient, diffuse, and specular light

• Materials: amount of RGB light reflected
  • value represents percentage reflected
    e.g., (0.0,1.0,0.5)

• Interaction: multiply components
  • Red light (1,0,0) x green surface (0,1,0) = black (0,0,0)
WHEN TO APPLY LIGHTING MODEL?

- per polygon “flat shading”
- per vertex “Gouraud shading”
- per pixel “per pixel lighting” “Phong shading”
NOTES ON SHADING

• To do all the calculations, we need to choose a coordinate system
• Typically View Coordinate System
• We need to have
  • Vertex Coordinates
  • Normals
  • Light Positions/Directions
COLORED WIREFRAMES
AMBIENT LIGHTING
PER-POLYGON SHADING
PER VERTEX SHADING
PER PIXEL SHADING
CURVED SURFACES WITH PER-PIXEL SHADING
COMPLEX LIGHTING AND SHADING
TEXTURE MAPPING
DISPLACEMENT MAPPING
REFLECTION MAPPING
GLOBAL ILLUMINATION
SUBSURFACE SCATTERING
TRANSFORMING NORMALS
COMPUTING NORMALS

- polygon:

\[ N = \frac{(P_2 - P_1) \times (P_3 - P_1)}{\| (P_2 - P_1) \times (P_3 - P_1) \|} \]

- assume vertices ordered CCW when viewed from visible side of polygon
TRANSFORMING NORMALS

Line + Normal

Transform both by same matrix

Transformed line and correct normal
TRANSFORMING NORMALS

- When transforming triangle(s) can we use the same transformation to transform the normal & avoid re-computation?

- What is a normal?
  - Vector
    - Orthogonal (perpendicular) to plane/surface

- Do standard transformations preserve orthogonality?
  - Or angles in general?
FIRST THINGS FIRST

• Dot product notation: $a \cdot b$
• Matrix notation: $a^T b$
  • Both $a$ and $b$ are columns
Let’s take a plane $Ax + By + Cz + D = 0$

And two points on the plane: $P_1, P_2$

$$ (A, B, C, *) \cdot (P_1 - P_2) = 0 $$

$$ \mathbf{n} \cdot (P_1 - P_2) = 0 $$
Let’s take a plane $Ax + By + Cz + D = 0$

And two points on the plane: $P_1, P_2$

$$(A, B, C, *) \cdot (P_1 - P_2) = 0$$

$$n \cdot (P_1 - P_2) = 0$$

or, exactly the same:

$$n^T M^{-1} M (P_1 - P_2) = 0$$
Let's take a plane $Ax + By + Cz + D = 0$

And two points on the plane: $P_1, P_2$

$$(A,B,C,*) \cdot (P_1 - P_2) = 0$$

$n \cdot (P_1 - P_2) = 0$

or, exactly the same:

$$n^T M^{-1} M (P_1 - P_2) = 0$$

After transformation $M$:

$$(n')^T (MP_1 - MP_2) = 0$$
PLANES AND NORMALS

Let’s take a plane $Ax + By + Cz + D = 0$
And two points on the plane: $P_1, P_2$

$$\begin{align*}
(A, B, C, *) \cdot (P_1 - P_2) &= 0 \\
n \cdot (P_1 - P_2) &= 0
\end{align*}$$

or, exactly the same:

$$n^T M^{-1} M (P_1 - P_2) = 0$$

After transformation $M$:

$$(n')^T (MP_1 - MP_2) = 0$$

So,

$$n^T M^{-1} = (n')^T$$

$$n' = (M^{-1})^T n$$
TRANSFORMING NORMALS

\[ n' = (M^{-1})^T n \]

Normals are transformed by
Transpose of Inverse
IN THREE.JS

• In vertex shader:

```javascript
pointInVCS = modelViewMatrix * vec4(position, 1.0);

normalInVCS = normalMatrix * normal;
```

transpose of inverse of modelViewMatrix
PROBLEMS ON ILLUMINATION

(whiteboard)
THE RENDERING PIPELINE

Vertices and attributes

Vertex Shader
- Modelview transform
- Per-vertex attributes

Vertex Post-Processing
- Viewport transform
- Clipping

Rasterization
- Scan conversion
- Interpolation

Fragment Shader
- Texturing/...
- Lighting/shading

Per-Sample Operations
- Depth test
- Blending

Framebuffer
SHADOWS
SHADOWS

Need at least 2 shader passes:

1. Draw everything as it’s viewed from the LIGHT SOURCE
SHADOWS

Need at least 2 shader passes:

1. Draw everything as it’s viewed from the LIGHT SOURCE
   Depth per pixel (‘depth map’)
SHADOWS (IDEA)

Need at least 2 shader passes:

1. Draw everything as it’s viewed from the LIGHT SOURCE Depth per pixel (‘depth map’).
2. Now draw everything from CAMERA

When computing color per pixel:

- Find corresponding depth map pixel: 
  D - distance from light source

- Is distance from me to the camera > D?
  - Yes: I am occluded! I’m in SHADOW.
  - No: I’m LIT!
SHADOWS (IDEA)

Need at least 2 shader passes:

1. Draw everything as it’s viewed from the LIGHT SOURCE Depth per pixel (‘depth map’).
2. Now draw everything from CAMERA.

When computing color per pixel:

- Find corresponding depth map pixel:
  D - distance from light source

- Is distance from me to the camera > D?
  - Yes: I am occluded! I’m in SHADOW.
  - No: I’m LIT!
CODING A3 & THEORY A3

• Out tonight
• Coding A3 *(due Nov, 6th)*:
  • Lighting & Shading
  • A bit of texturing
• Theory A3 *(due Oct, 30th, in class)*:
  • Clipping
  • Rasterization
  • Lighting & Shading
SOME HINTS ON THEORY A3
SHAPES - CURVES/SURFACES

- Mathematical representations:
  - Explicit functions
  - Parametric functions
  - Implicit functions
SHAPES: EXPLICIT FUNCTIONS

• Curves: $y := \sin(x)$
  • $y$ is a function of $x$:
  • Only works in 2D

• Surfaces: $z := \sin(x) + \cos(y)$
  • $z$ is a function of $x$ and $y$:
  • Cannot define arbitrary shapes in 3D
SHAPES: PARAMETRIC FUNCTIONS

- Curves:
  - 2D: $x$ and $y$ are functions of a parameter value $t$
  - 3D: $x$, $y$, and $z$ are functions of a parameter value $t$

$$C(t) := \begin{pmatrix} \cos(t) \\ \sin(t) \\ t \end{pmatrix}$$
Surfaces:
• Surface $S$ is defined as a function of parameter values $s$, $t$
• Names of parameters can be different to match intuition:

$$S(\phi, \theta) := \begin{pmatrix} \cos(\phi) \cos(\theta) \\ \sin(\phi) \cos(\theta) \\ \sin(\theta) \end{pmatrix}$$
SHAPES: IMPLICIT

• Surface (3D) or Curve (2D) defined by zero set (roots) of function
  • E.g:

\[ S(x, y, z) : x^2 + y^2 + z^2 - 1 = 0 \]
HOW TO INTERSECT?

• Two lines in 2D?
• A line and a plane?
• A line and a sphere?
• (Whiteboard)