Intro to Lighting
2-10-15
Outline

- Projection Normalization
- Introduction to Lighting (and Shading)

Read: Angel
- Chapter 5., sections 5.4 - 5.7 Parallel Projections
- Chapter 6, sections 6.1 - 6.3 Lighting & Shading

Lab2: use ASDW to move a 2D shape around
From model to viewing

Consider the vertex shaders for each of these examples:

- Coordinate systems (from Chastine)
- ColorCube example (from Matsuda)
- perspective example (from Angel)
in vec4 vPosition;  // The vertex in the local coordinate system
uniform mat4 mM;  // The matrix for the pose of the model
uniform mat4 mV;  // The matrix for the pose of the camera
uniform mat4 mP;  // The projection matrix (perspective)

void main () {
  gl_Position = mP*mV*mM*vPosition;
}

Original (local) position
New position in NDC
ColoredCube Vertex Shader

attribute vec4 a_Position;
attribute vec4 a_Color;
uniform mat4 u_MvpMatrix;
varing vec4 v_Color;

void main() {
    gl_Position = u_MvpMatrix * a_Position;
    v_Color = a_Color;
}
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void main() {
  gl_Position = projectionMatrix*modelViewMatrix*vPosition;
  fColor = vColor;
}
WebGL Orthogonal Viewing

\( \text{ortho(left, right, bottom, top, near, far)} \)

\( \text{near and far measured from camera} \)
WebGL Perspective

frustum(left, right, bottom, top, near, far)
With frustum it is often difficult to get the desired view.

\texttt{perspective(fovy, aspect, near, far)} often provides a better interface.
Computing Matrices

- Compute in JS file, send to vertex shader with `gl.uniformMatrix4fv`
- Dynamic: update in `render()` or shader
Projection Normalization

- Rather than derive a different projection matrix for each type of projection, we can convert all projections to orthogonal projections with the default view volume.
- This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping.
Pipeline View

- modelview transformation
- projection transformation
- perspective division

4D → 3D

- clipping
- projection

3D → 2D

against default cube
We stay in four-dimensional homogeneous coordinates through both the modelview and projection transformations

Both these transformations are nonsingular
Default to identity matrices (orthogonal view)

Normalization lets us clip against simple cube regardless of type of projection

Delay final projection until end
Important for hidden-surface removal to retain depth information as long as possible
Orthogonal Normalization

\( \text{ortho}(\text{left, right, bottom, top, near, far}) \)

normalization \(\Rightarrow\) find transformation to convert specified clipping volume to default
Orthogonal Matrix

Two steps

Move center to origin
T(-(left+right)/2, -(bottom+top)/2,(near+far)/2))

Scale to have sides of length 2
S(2/(left-right),2/(top-bottom),2/(near-far))

\[
P = ST =
\begin{bmatrix}
\frac{2}{right - left} & 0 & 0 & -\frac{right - left}{right - left} \\
0 & \frac{2}{top - bottom} & 0 & \frac{top - bottom}{top - bottom} \\
0 & 0 & \frac{2}{near - far} & \frac{far + near}{far - near} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Final Projection

- Set $z = 0$
- Equivalent to the homogeneous coordinate transformation

$$M_{orth} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}$$

- Hence, general orthogonal projection in 4D is

$$P = M_{orth}ST$$
The OpenGL/WebGL projection functions cannot produce general parallel projections

- oblique parallel projections are useful
- an oblique projection is characterized by the angle the projectors make with the projection plane

However if we look at the example of the cube it appears that the cube has been sheared

Oblique Projection = Shear + Orthogonal Projection
General Shear

side view

top view
Shear Matrix

*xy* shear (*z* values unchanged)

\[
H(\theta, \phi) = \begin{bmatrix}
1 & 0 & -\cot \theta & 0 \\
0 & 1 & -\cot \phi & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Shearing matrix

Projection matrix \[ P = M_{\text{orth}} H(\theta, \phi) \]

General case: \[ P = M_{\text{orth}} \text{STH}(\theta, \phi) \]
Simple Perspective

Consider a simple perspective with the COP at the origin, the near clipping plane at \( z = -1 \), and a 90 degree field of view determined by the planes

\[
x = \pm z, \quad y = \pm z
\]
Simple projection matrix in homogeneous coordinates

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\]

Note that this matrix is independent of the far clipping plane.
after perspective division, the point \((x, y, z, 1)\) goes to

\[ x'' = \frac{x}{z} \]
\[ y'' = \frac{y}{z} \]
\[ Z'' = -(\alpha + \frac{\beta}{z}) \]

which projects orthogonally to the desired point regardless of \(\alpha\) and \(\beta\)
Picking $\alpha$ and $\beta$

If we pick

$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

$$\beta = \frac{2 \text{near} \times \text{far}}{\text{near} - \text{far}}$$

the near plane is mapped to $z = -1$
the far plane is mapped to $z = 1$
and the sides are mapped to $x = \pm 1$, $y = \pm 1$

Hence the new clipping volume is the default clipping volume
Normalization Transformation

original clipping volume

original object

new clipping volume

distorted object projects correctly
Although our selection of the form of the perspective matrices may appear somewhat arbitrary, it was chosen so that if $z_1 > z_2$ in the original clipping volume then the for the transformed points $z_1' > z_2'$. Thus hidden surface removal works if we first apply the normalization transformation. However, the formula $z'' = -(\alpha + \beta/z)$ implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small.
 WebGL Perspective

- `gl.frustum` allows for an unsymmetric viewing frustum (although `gl.perspective` does not)
The normalization in frustum requires an initial shear to form a right viewing pyramid, followed by a scaling to get the normalized perspective volume. Finally, the perspective matrix results in needing only a final orthogonal transformation.

\[ P = \text{NSH} \]

our previously defined perspective matrix

shear and scale
Why do we do it this way?

- Normalization allows for a single pipeline for both perspective and orthogonal viewing.
- We stay in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading.
- We simplify clipping.
Perspective Matrices

frustum

\[
P = \begin{bmatrix}
\frac{2 \cdot \text{near}}{\text{right} - \text{left}} & 0 & \frac{\text{right} - \text{left}}{\text{top} - \text{bottom}} & 0 \\
0 & \frac{2 \cdot \text{near}}{\text{top} + \text{bottom}} & 0 & 0 \\
0 & 0 & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} & -2 \cdot \frac{\text{far} \cdot \text{near}}{\text{far} - \text{near}} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

perspective

\[
P = \begin{bmatrix}
\frac{\text{near}}{\text{right}} & 0 & 0 & 0 \\
0 & \frac{\text{near}}{\text{top}} & 0 & 0 \\
0 & 0 & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} & -2 \cdot \frac{\text{far} \cdot \text{near}}{\text{far} - \text{near}} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]
What is Light?

A very complex process

- Find a dark area – how is it being lit?
- Light bounces (mirrors, shiny objects)
- Light refracts through other media (water, heat)
- Light comes from everywhere (Global Illumination)
- Light bounces off of lakes in weird ways (Fresnel effect)

THUS

- We’re forced to make approximations
- Tradeoff between time and realism
- “If it looks good, it is good”
  – Michael Abrash


http://darrentakenaga.com/3d.html
Lighting Principles

- Lighting simulates how objects reflect light
  - material composition of object
  - light’s color and position
  - global lighting parameters
- Lighting functions deprecated in 3.1
- Can implement in
  - Application (per vertex)
  - Vertex or fragment shaders
Shading

- Why does the image of a real sphere look like

- Light-material interactions cause each point to have a different color or shade

- Need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation
Scattering

- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A
  and so on
The infinite scattering and absorption of light can be described by the rendering equation.

- Cannot be solved in general
- Ray tracing is a special case for perfectly reflecting surfaces
- Rendering equation is global and includes:
  - Shadows
  - Multiple scattering from object to object
Global Effects

- Shadow
- Multiple reflection
- Translucent surface
Correct shading requires a global calculation involving all objects and light sources.

Incompatible with pipeline model which shades each polygon independently (local rendering).

However, in computer graphics, especially real time graphics, we are happy if things "look right.”

Exist many techniques for approximating global effects.
Light-Material Interaction

- Light that strikes an object is partially absorbed and partially scattered (reflected).
- The amount reflected determines the color and brightness of the object.
  A surface appears red under white light because the red component of the light is reflected and the rest is absorbed.
- The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface.
General light sources are difficult to work with because we must integrate light coming from all points on the source.
Simple Light Sources

- Point source
  Model with position and color
  Distant source = infinite distance away (parallel)

- Spotlight
  Restrict light from ideal point source

- Ambient light
  Same amount of light everywhere in scene
  Can model contribution of many sources and reflecting surfaces
Surface Types

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflect the light.
- A very rough surface scatters light in all directions.

smooth surface

rough surface
Phong Model

- A simple model that can be computed rapidly
- Has three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source
  - To viewer
  - Normal
  - Perfect reflector

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015
Ideal Reflector

- Normal is determined by local orientation
- Angle of incidence = angle of reflection
- The three vectors must be coplanar

\[ r = 2 (l \cdot n) n - l \]
Lambertian Surface

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light
  
  reflected light $\sim \cos \theta_i$

  $\cos \theta_i = \mathbf{l} \cdot \mathbf{n}$ if vectors normalized

  There are also three coefficients, $k_r, k_b, k_g$ that show how much of each color component is reflected
Specular Surfaces

- Most surfaces are neither ideal diffusers nor perfectly specular (ideal reflectors).
- Smooth surfaces show specular highlights due to incoming light being reflected in directions concentrated close to the direction of a perfect reflect.
Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased.

$$I_r \sim k_s I \cos^\alpha \phi$$

- reflected intensity
- shininess coef
- incoming intensity
- absorption coef
The Shininess Coefficient

- Values of $\alpha$ between 100 and 200 correspond to metals
- Values between 5 and 10 give surface that look like plastic
Modified Phong Model

- Computes a color or shade for each vertex using a lighting model (the modified Phong model) that takes into account:
  - Diffuse reflections
  - Specular reflections
  - Ambient light
  - Emission

- Vertex shades are interpolated across polygons by the rasterizer
The Modified Phong Model

- The model is a balance between simple computation and physical realism

- The model uses
  - Light positions and intensities
  - Surface orientation (normals)
  - Material properties (reflectivity)
  - Viewer location

- Computed for each source and each color component
Surface Normals

- Normals define how a surface reflects light
  - Application usually provides normals as a vertex attribute
  - Current normal is used to compute vertex’s color
  - Use *unit* normals for proper lighting
    - scaling affects a normal’s length
Material Properties

- Define the surface properties of a primitive
  - you can have separate materials for front and back

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Diffuse</strong></td>
<td><em>Base object color</em></td>
</tr>
<tr>
<td><strong>Specular</strong></td>
<td><em>Highlight color</em></td>
</tr>
<tr>
<td><strong>Ambient</strong></td>
<td><em>Low-light color</em></td>
</tr>
<tr>
<td><strong>Emission</strong></td>
<td><em>Glow color</em></td>
</tr>
<tr>
<td><strong>Shininess</strong></td>
<td><em>Surface smoothness</em></td>
</tr>
</tbody>
</table>
A BASIC LIGHTING CONCEPT

• How can we determine how much light should be cast onto a triangle from a directional light?

Directional light
- position doesn’t matter
- triangle is almost fully lit
A BASIC LIGHTING CONCEPT

• How can we determine how much light should be cast onto a triangle from a directional light?
A BASIC LIGHTING CONCEPT

- How can we determine how much light should be cast onto a triangle from a directional light?
A BASIC LIGHTING CONCEPT

• How can we determine how much light should be cast onto a triangle from a directional light?
A BASIC LIGHTING CONCEPT

• How can we determine how much light should be cast onto a triangle from a directional light?
A BASIC LIGHTING CONCEPT

• How can we determine how much light should be cast onto a triangle from a directional light?

Lesson learned: Lighting depends on angles between vectors!
A BASIC LIGHTING CONCEPT

• How can we determine how much light should be cast onto a triangle from a directional light?

Assuming $N$ and $L$ are normalized, and $N \cdot L$ isn’t negative

$$\text{intensity} = \arccos(N \cdot L)$$
BASIC LIGHTING

- Four independent components:
  - **Diffuse** – the way light “falls off” of an object
  - Specular – the “shininess” of the object
  - Ambient – a minimum amount of light used to simulate “global illumination”
  - Emit – a “glowing” effect

Only diffuse
BASIC LIGHTING

- Four independent components:
  - Diffuse – the way light “falls off” of an object
  - **Specular** – the “shininess” of the object
  - Ambient – a minimum amount of light used to simulate “global illumination”
  - Emit – a “glowing” effect
BASIC LIGHTING

- Four independent components:
  - Diffuse – the way light “falls off” of an object
  - Specular – the “shininess” of the object
  - Ambient – a minimum amount of light used to simulate “global illumination”
  - Emit – a “glowing” effect
BASIC LIGHTING

- Four independent components:
  - Diffuse – the way light “falls off” of an object
  - Specular – the “shininess” of the object
  - Ambient – a minimum amount of light used to simulate “global illumination”
  - Emit – a “glowing” effect

Note: emit does not produce light!
INTERACTION BETWEEN MATERIAL AND LIGHTS

- Final color of an object is comprised of many things:
  - The base object color (called a “material”)
  - The light color
  - Example: a purple light on a white surface
  - Any textures we apply (later)
- Materials and lights have four *individual* components
  - Diffuse color \( (c_d \text{ and } l_d) \)
  - Specular color \( (c_s \text{ and } l_s) \)
  - Ambient color \( (c_a \text{ and } l_a) \)
  - Emit color \( (c_e \text{ and } l_e) \)
- \( c_d \times l_d = [c_{d.r} \times l_{d.r}, c_{d.g} \times l_{d.g}, c_{d.b} \times l_{d.b}] \) // R, G, B
GENERAL LIGHTING

- Primary vectors
  - $l$ – the incoming light vector
  - $n$ – the normal of the plane/vertex
  - $r$ – the reflection vector
  - $v$ – the viewpoint (camera)
LAMBERTIAN REFLECTANCE
(DIFFUSE COMPONENT)

- Light falling on an object is the same regardless of the observer’s viewpoint
- Good for rough surfaces without specular highlights
- $\text{final\_color_{diffuse}} = n \cdot l \cdot c_d \cdot l_d$ where $n$ and $l$ are normalized
LAMBERTIAN REFLECTANCE
(DIFFUSE COMPONENT)

- Light falling on an object is the same regardless of the observer's viewpoint
- Good for rough surfaces without specular highlights
- $final_{\text{color}}_{\text{diffuse}} = n \cdot l \cdot c_d \cdot l_d$ where $n$ and $l$ are normalized

Note: $final_{\text{color}}_{\text{diffuse}}$ has R, G, B
LAMBERTIAN REFLECTANCE
(DIFFUSE COMPONENT)

Technically, it should be:

\[ final\_color_{diffuse} = \max(n \cdot l, 0) \cdot c_d \cdot l_d \]
BLINN-PHONG REFLECTION
(SPECULAR COMPONENT)

- Describes the specular highlight and is dependent on viewpoint $v$
- Also describes a “half-vector” $h$ that is halfway between $v$ and $l$
BLINN-PHONG REFLECTION
(SPECULAR COMPONENT)

\[ h = v + l \] - which is really Blinn's contribution to the original Phong model

Note: vectors should be normalized
BLINN-PHONG REFLECTION
(SPECULAR COMPONENT)

Our final specular equation is:

$$final\_color_s = (n \cdot h)^s \cdot c_s \cdot l_s$$
DETERMINING $s$

$$\text{final_color}_s = (n \cdot h)^s \cdot c_s \cdot l_s$$

- Realize that $n \cdot h$ will always be $< 1.0$, so raising it to a power will make it smaller
- $s$ is the "shininess" factor
  - It relates to the size of the specular highlight

$s = \sim 1$

$s = \sim 30$

$s = \sim 255$
AMBIENT AND EMIT COMPONENTS

- Ambient:
  - Used to simulate light bouncing around the environment (global illumination)
  - Real world is far too complex for real time, so just add a little light!
- Emit:
  - Used to make the object “glow"
  - Does not emit light!!!
- Both:
  - Independent of viewpoint
  - Super easy to calculate

\[
final\_color_{ambient} = l_a + c_a
\]
\[
final\_color_{emit} = l_e + c_e
\]
FINAL COLOR

- To determine the final color (excluding textures) we sum up all components:

\[
\text{final\_color}_{\text{diffuse}} + \text{final\_color}_{\text{specular}} + \text{final\_color}_{\text{ambient}} + \text{final\_color}_{\text{emit}} = \text{final\_color}
\]

WHAT ABOUT MULTIPLE LIGHTS?

- Calculate final colors and sum them all together
- Assuming results are in $f[i]$ and there are $count$ number of lights

$$final\_color = \sum_{i=1}^{count} (f[i]_d, f[i]_s, f[i]_a, f[i]_e)$$