Previously...

- We define our geometry (points, lines, triangles)
- We apply transformations (matrices)

\[ \begin{pmatrix} \cos(45^\circ) & -\sin(45^\circ) \\ \sin(45^\circ) & \cos(45^\circ) \end{pmatrix} \begin{array}{c} \hline \\ \hline \end{array} = \begin{array}{c} \hline \\ \hline \end{array} \]

When is this true?
Now...

This isn't quite true...

What exactly is here?
Material properties

• We want GPU to take into account a color property when rendering some geometry.

Red cube?

Two red trapezoids?

Flat red polygon?
What is color?

- Spectrum of the light reflected off a surface.
- In 3D we can't get away with only saying that something is red.
- We need to say that somewhere we have a some kind of light source.
Directional light

- Ok, we have a light direction
- A surface
- Viewer

Viewer doesn't see surface point 4?
Directional light

- Reality – our surfaces are diffusely reflective!
Diffuse surface

- So all we need now is the angle between the surface normal and the light's direction.

By the way, the scattered light intensities may not be equal in all directions...

- Why this angle?
Diffuse surface

Hint?
Diffuse surface

- The actual light energy per surface unit depends on the angle.

\[
\frac{1}{\cos(45^\circ)} \approx 1.42 \quad \frac{1}{\cos(80.81^\circ)} \approx 6.26
\]
Diffuse surface & directional light

- Given a surface point and a light source, we can calculate the color of that surface point.
- We use cosine between the surface normal and a vector going towards the light source.

![Diagram showing cosine between surface normal and direction towards light]
Diffuse surface and directional light

- To find the cosine of the angle, we can use a scalar / dot product operation.

\[ \mathbf{v} \cdot \mathbf{u} = |\mathbf{v}| |\mathbf{u}| \cos(\text{angle}(\mathbf{u}, \mathbf{v})) \]

- Geometric definition

\[ \mathbf{v} \cdot \mathbf{u} = v_1 u_1 + v_2 u_2 + v_3 u_3 \]

- Algebraic definition

- Because we now have normals, geometric definition simplifies to:

\[ \mathbf{v} \cdot \mathbf{u} = |\mathbf{v}| |\mathbf{u}| \cos(\alpha) = 1 \cdot 1 \cdot \cos(\alpha) = \cos(\alpha) \]
Diffuse surface and directional light

• So if we put those two definitions together:

\[ v \cdot u = v_1 \cdot u_1 + v_2 \cdot u_2 + v_3 \cdot u_3 = \cos(\alpha) \]

This should be quite easy for the computer to calculate...

• Do we now need to define a new operation (dot product) for the computer?

Is this similar to something we already know?

More code?
Diffuse surface and directional light

• Dot product of two vectors $u$ and $v$ is the same as vector multiplication.

$$v \cdot u = v_1 \cdot u_1 + v_2 \cdot u_2 + v_3 \cdot u_3 = \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = v^T u$$

• So for our surface point we get:

$$Intensity = \text{lightDirection}^T \cdot \text{surfaceNormal}$$

$$I = l^T \cdot n$$

$$I \in [0,1]$$

What is the visual result of that?
Diffuse surface and directional light

- Two things were missing:
  - Intensity of the light source
  - Color of our material
- Also, this will apply to each of 3 RGB channels.

\[
I_R = n^T \cdot l \cdot L_R \cdot M_R \\
I_G = n^T \cdot l \cdot L_G \cdot M_G \\
I_B = n^T \cdot l \cdot L_B \cdot M_B
\]

Light that light source emits

Light that material reflects
Diffuse surface and directional light

What color are the apples if red light shines upon them?

What is wrong with this example? (2+ things)
Point light

- Point lights work the same way, but the light source is a point.
Point light

- Sometimes distance attenuation parameters are added.
- In OpenGL: \[ \text{attenuation} = \frac{1}{k_c + k_l \cdot d + k_q \cdot d^2} \]

- In THREE.js:
  
  PointLight(hex, intensity, distance)

  *Distance* - *If non-zero, light will attenuate linearly from maximum intensity at light position down to zero at distance.*

http://threejs.org/docs/#Reference/Lights/PointLight
Ambient light

- So, now we have 2 lights and a diffuse surface.
- Are we OK?
Ambient light

- World contains much more than 1 cube and a light source.
- Do you remember what that was?
- Calculating every reflection from every other object is time-consuming.
- What can we do?
Ambient light

- Ambient light source – estimates the light reflected off of other objects in the scene
- Ambient material property – how much object reflects that light (usually same as diffuse)
Lambert material

- So together with diffuse lighting we get:

\[
I_R = L_{AR} \cdot M_{AR} + n^T \cdot l \cdot L_{DR} \cdot M_{DR}
\]

\[
I_G = L_{AG} \cdot M_{AG} + n^T \cdot l \cdot L_{DG} \cdot M_{DG}
\]

\[
I_B = L_{AB} \cdot M_{AB} + n^T \cdot l \cdot L_{DB} \cdot M_{DB}
\]

Red channel

Green channel

Blue channel

Ambient term

Diffuse term

What could go wrong?
Is this it?

- Well, we've already made a very rough approximation of reality with the ambient term.
- But is there anything else that we've forgotten?
Specular highlight

- Depends on the viewer's position.
- We have to get back to the angles.
- At point 4, which viewer direction should produce more specular highlight?
- How to calculate that based on $\alpha$?
Specular highlights

- Ok, let's add a specular term based on the actual reflection direction ($r$) and viewer direction ($v$).

\[
I_R = L_{A_R} \cdot M_{A_R} + n^T \cdot l \cdot L_{D_R} \cdot M_{D_R} + r^T \cdot v \cdot L_{S_R} \cdot M_{S_R}
\]

\[
I_G = L_{A_G} \cdot M_{A_G} + n^T \cdot l \cdot L_{D_G} \cdot M_{D_G} + r^T \cdot v \cdot L_{S_G} \cdot M_{S_G}
\]

\[
I_B = L_{A_B} \cdot M_{A_B} + n^T \cdot l \cdot L_{D_B} \cdot M_{D_B} + v^T \cdot r \cdot L_{S_B} \cdot M_{S_B}
\]

Light properties are usually same in same channel.

Is there something missing?
Specular highlights

- Let's calculate specular highlight for different angles:

<table>
<thead>
<tr>
<th>$M_S$</th>
<th>$L_S$</th>
<th>$\alpha$</th>
<th>$\sim\cos(\alpha)$</th>
<th>$\sim l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>1</td>
<td>10°</td>
<td>0.98</td>
<td>0.25</td>
</tr>
<tr>
<td>0.25</td>
<td>1</td>
<td>20°</td>
<td>0.94</td>
<td>0.24</td>
</tr>
<tr>
<td>0.25</td>
<td>1</td>
<td>30°</td>
<td>0.87</td>
<td>0.22</td>
</tr>
<tr>
<td>0.25</td>
<td>1</td>
<td>40°</td>
<td>0.77</td>
<td>0.19</td>
</tr>
<tr>
<td>0.25</td>
<td>1</td>
<td>50°</td>
<td>0.64</td>
<td>0.16</td>
</tr>
<tr>
<td>0.25</td>
<td>1</td>
<td>60°</td>
<td>0.5</td>
<td>0.12</td>
</tr>
<tr>
<td>0.25</td>
<td>1</td>
<td>70°</td>
<td>0.34</td>
<td>0.09</td>
</tr>
<tr>
<td>0.25</td>
<td>1</td>
<td>80°</td>
<td>0.17</td>
<td>0.04</td>
</tr>
<tr>
<td>0.25</td>
<td>1</td>
<td>90°</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Assume we're dealing with one channel (e.g. red)
Assume the channel values are between [0, 1] (mapped later to [0, 255])
Specular highlights

- How to increase the contrast? Use a power.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\sim \cos^2(\alpha)$</th>
<th>$\sim I$</th>
<th>$\sim \cos^3(\alpha)$</th>
<th>$\sim I$</th>
<th>$\sim \cos^4(\alpha)$</th>
<th>$\sim I$</th>
<th>$\sim \cos^5(\alpha)$</th>
<th>$\sim I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10°</td>
<td>0.97</td>
<td>0.24</td>
<td>0.96</td>
<td>0.24</td>
<td>0.94</td>
<td>0.23</td>
<td>0.92</td>
<td>0.23</td>
</tr>
<tr>
<td>20°</td>
<td>0.88</td>
<td>0.22</td>
<td>0.83</td>
<td>0.21</td>
<td>0.78</td>
<td>0.20</td>
<td>0.73</td>
<td>0.18</td>
</tr>
<tr>
<td>30°</td>
<td>0.75</td>
<td>0.19</td>
<td>0.65</td>
<td>0.16</td>
<td>0.56</td>
<td>0.14</td>
<td>0.49</td>
<td>0.12</td>
</tr>
<tr>
<td>40°</td>
<td>0.59</td>
<td>0.15</td>
<td>0.45</td>
<td>0.11</td>
<td>0.34</td>
<td>0.09</td>
<td>0.26</td>
<td>0.07</td>
</tr>
<tr>
<td>50°</td>
<td>0.41</td>
<td>0.10</td>
<td>0.27</td>
<td>0.07</td>
<td>0.17</td>
<td>0.04</td>
<td>0.11</td>
<td>0.03</td>
</tr>
<tr>
<td>60°</td>
<td>0.25</td>
<td>0.06</td>
<td>0.13</td>
<td>0.03</td>
<td>0.06</td>
<td>0.02</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>70°</td>
<td>0.12</td>
<td>0.04</td>
<td>0.04</td>
<td>0.01</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>80°</td>
<td>0.03</td>
<td>0.01</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>90°</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Specular highlights

- Specify some **shininess** value \( c \) for the material

\[
I_R = L_{A_R} \cdot M_{A_R} + n^T \cdot l \cdot L_{D_R} \cdot M_{D_R} + (r^T \cdot v)^c \cdot L_{S_R} \cdot M_{S_R}
\]

\[
I_G = L_{A_G} \cdot M_{A_G} + n^T \cdot l \cdot L_{D_G} \cdot M_{D_G} + (r^T \cdot v)^c \cdot L_{S_G} \cdot M_{S_G}
\]

\[
I_B = L_{A_B} \cdot M_{A_B} + n^T \cdot l \cdot L_{D_B} \cdot M_{D_B} + (r^T \cdot v)^c \cdot L_{S_B} \cdot M_{S_B}
\]

Ambient approx.  Lambertian reflectance  Phong's shininess term

Something still missing?  Phong lighting model
Specular highlights

c=0

c=90

c=30

c=300
Blinn-Phong model

- Sometimes Phong's term is replaced with Blinn-Phong's term.
- Instead of viewer direction and reflected light's direction, we use the surface normal and a half angle between the light source and the viewer.
Blinn-Phong model

- There are some differences
- These are not the only two possibilities

DEMO 2: http://cgdemos.tume-maailm.pri.ee/
THREE.JS videos: https://www.udacity.com/course/viewer#!/c-cs291/l-124106593/m-157996647
Standard Graphics Pipeline

- **Vertex transformations**
  - Vertex shader, PVM * v
  - Perspective division, Viewport transformation

- **Rasterization**
  - Culling – remember the face directions?
  - Clipping – some parts are out of view

- **Fragment shading**

- **Visibility tests & Blending**
Vertex shader

- Code you send to GPU to operate on each vertex.

```glsl
#version 120

varying vec3 normal;
varying vec4 vertex_pos;

void main(void) {
    gl_Position = ftransform();
    vertex_color_if_lighting_disabled = gl_Color;
    // Send the normal and vertex positions down to the fragment shader
    normal = gl_NormalMatrix * gl_Normal;
    vertex_pos = gl_ModelViewMatrix * gl_Vertex;
}
```

- Corresponds to OpenGL 2.1
- GLSL 1.20.8
- ` varying type is interpolated`
- This shader needs to set `gl_Position`
- `ftransform()` is `gl_ProjectionMatrix * gl_ModelViewMatrix * gl_Vertex`;
Fragment shader

- Code you send to GPU to operate on each fragment (pixel)

```cpp
#version 120

varying vec3 normal;
varying vec4 vertex_pos;
uniform vec3 viewer_pos;

vec4 frag_color = vec4(0, 0, 0, 0);
void addLight(int lightIndex);

void main(void) {
    frag_color = gl_LightModel.ambient * gl_FrontMaterial.ambient;
    addLight(0);
    gl_FragColor = frag_color;
}
```

Uniform variables you send to the shader

OpenGL will give lights info (like the ambient term).

Must set the gl_FragColor
Fragment shader

```glsl
void addLight(int lightIndex) {
    float attenuation = 1.0; // no attenuation
    vec3 light_dir = normalize(gl_LightSource[lightIndex].position.xyz);

    vec3 light_reflection = 2 * normal * dot(light_dir, normal) - light_dir;
    frag_color += attenuation *
        gl_FrontMaterial.diffuse *
        gl_LightSource[lightIndex].diffuse *
        clamp(dot(normal, light_dir), 0, 1);
    frag_color += attenuation *
        gl_FrontMaterial.specular *
        gl_LightSource[lightIndex].specular *
        pow(clamp(dot(viewer, light_reflection), 0, 1), gl_FrontMaterial.shininess);
}
```

Lights in OpenGL are indexed

This code only supports directional light

What lighting model is this?

Why clamp? What's clamp??
New GLSL shaders

```glsl
#version 330

layout(location = 0)in vec4 vert;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

void main() {
    gl_Position = projection * view * model * vertex;
}
```

This is what you send to the shader via Buffer Objects (VBO)

```glsl
#version 330

out vec4 outputColor;

void main() {
    outputColor = vec4(1.0f, 1.0f, 1.0f, 1.0f);
}
```

This will be the output of the shader

Which is which?

Corresponds to OpenGL 3.3

Extensive tutorial: http://arcsynthesis.org/gltut/
Raytracing

- Cast a ray and find the closest object it hits.
- Recurse from the hitpoint.
- Where to cast a ray from and where to?
  - Camera, pass through some pixel in screen
  - From hit point to reflection direction
  - From hit point to refraction direction
  - From a front of a spaceship to forwards direction
Raytracing

- Realistic reflections
- Realistic lighting
- Realistic shadows
Raytracing

• How to find a hit point?
• I assume at this point I don't have much time left, so... I recommend to watch this: https://www.youtube.com/watch?v=zQlvR_6IayM
• Basic idea is to find an intersection of the ray and a plane defined by a triangle.

Our geometry consists of small triangles.

Remember, vertices of a triangle were always on some plane.
Raytracing

- Usually quite resource consuming
- Can be optimized with:
  - Bounding Volume Hierarchies
  - Space Partitioning

Example of one space partitioning structure... there are many.
Conclusion

- Computer graphics can be used to create a illusion of reality

- First approximation is of the shape – geometry
- GPU wants those triangles
- Vertices and transformation matrices
Conclusion

- Many ways to approximate lighting (Lambert, Phong, Blinn), reflections, refractions, shadows...
- Ambient, diffuse, specular terms

\[ I = L_A \cdot M_A + n^T \cdot l \cdot L_D \cdot M_D + (r^T \cdot v)^c \cdot L_S \cdot M_S \]

Direction towards light, surface normal, reflection direction, direction towards viewer
Thanks for listening!