Computer Graphics Seminar

MTAT.03.305

Spring 2018
Previously...

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

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

When is this true?
Now we add color?

This isn't quite true...

What exactly is here?

Adding color...?
Material properties

- We want GPU to take into account a color property when rendering some geometry.
Material properties

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

http://cgdemos.tume-maailm.pri.ee/
What is color?
What is color?

- Spectrum of the **light reflected** off a surface.
What is color?

- Spectrum of the light reflected off a surface.
- In 3D it is not enough to just say that a thing is red.
What is color?

- Spectrum of the **light reflected** off a surface.
- In 3D it is not enough to just say that a **thing is red**.
- We need to say that somewhere we have a some kind of **light source**.
Directional light

- Ok, we define a light direction
Directional light

- Ok, we define a light direction
- A surface
Directional light

- Ok, we define a light direction
- A surface
- Viewer
Directional light

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

Viewer does not see surface point at 4?
Directional light

- Reality – our surfaces are diffusely reflective!
Diffuse Reflection

- Light entering at a specific angle
Diffuse Reflection

- Photon excites an atom
Diffuse Reflection

- The energy is transferred to the next atom
Diffuse Reflection

- The energy is transferred to the next atom
- Some energy is absorbed
Diffuse Reflection

- Excited atoms vibrate, giving off heat
Diffuse Reflection

- Finally photon exits the surface
Diffuse Reflection

- In a quite random direction
Diffuse Reflection

- This is *generally* how *pigments* work

Nice post: https://physics.stackexchange.com/a/240848
Diffuse Reflection

- Can be caused by other reasons too!
Diffuse Reflection

- Can be caused by other reasons too!
- For example structural coloration in nature.

https://en.wikipedia.org/wiki/Pollia_condensata

All of these feathers are actually brown.
Diffuse Reflection

- Can be caused by other reasons too!
- For example **structural coloration** in nature.
Diffuse Reflection

- Let's assume diffuse light scatters uniformly
Diffuse Reflection

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

More correct is direction towards the light.

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

• Why this angle?
Diffuse Reflection

Hint?
Diffuse Reflection

- 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 Reflection & Directional Light

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

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

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

Geometric definition

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

Algebraic definition
Diffuse Reflection & Directional Light

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

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

Geometric definition

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

Algebraic definition

- Because we have normalized (unit) vectors, geometric definition simplifies to:

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

- The dot product and the cosine between two vectors are used quite often in CG.
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:

\[
\text{Intensity} = \text{directionTowardsLight}^T \cdot \text{surfaceNormal} \\
I = l^T \cdot n \\
I \in [0,1]
\]

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

- Two things were missing:
  - Intensity of the light source $L$
  - Reflectivity of our material $M$
Diffuse surface and directional light

- Also the color!
- We 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 material reflects

Light that light source emits
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.
Point light

- Sometimes distance **attenuation** parameters are added.
- In OpenGL:

  \[
  \text{attenuation} = \frac{1}{k_c + k_l \cdot d + k_q \cdot d^2}
  \]

  Usually 1 (why?)

- In Three.js:

  ```javascript
  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 know what scene this is?
- 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 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)
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:

\[ I_R = L_{A_R} \cdot M \cdot L_{D_R} \cdot M \cdot n^T \cdot l \cdot L_{D_R} \cdot M \]

\[ I_G = L_{A_G} \cdot M \cdot L_{D_G} \cdot M \]

\[ I_B = L_{A_B} \cdot M \cdot L_{D_B} \cdot M \]

What could go wrong?
Is this it?

- Well, we have already made a very rough approximation of reality with the ambient term.
- Is there anything else that we have forgotten?
Specular Reflection

- Materials also reflect light specularly.
Specular Reflection

- Materials also reflect light specularly.
- Especially varnished materials and metals!
Specular Reflection

- Materials also reflect light specularly.
- Especially varnished materials and metals!
- Specular reflection is the direct reflection of the light from the environment.
Specular Reflection

- Materials also reflect light specularly.
- Especially varnished materials and metals!
- Specular reflection is the direct reflection of the light from the environment.
- Often we want just a **specular highlight** – that is the **reflection of the light source**!
Specular highlight

- Depends on the viewer's position.
Specular highlight

- At point 4, which viewer direction should produce more specular highlight?
Specular highlight

- How to calculate that based on $\beta$?
Specular highlights

- Ok, so add a specular term based on the actual reflection direction \((r)\) and viewer direction \((v)\).

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

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

\[
I_B = L_{AB} \cdot M_{AB} + n^T \cdot l \cdot L_{DB} \cdot M_{DB} + v^T \cdot r \cdot L_{SB} \cdot M_{SB}
\]
Specular highlights

- Calculating 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>$\sim I$</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 are 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>$\sim l$</th>
<th>$\sim \cos^3(\alpha)$</th>
<th>$\sim l$</th>
<th>$\sim \cos^4(\alpha)$</th>
<th>$\sim l$</th>
<th>$\sim \cos^5(\alpha)$</th>
<th>$\sim l$</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>

Values above 0.25
Specular highlights

- Specify some *shininess* value $c$ for the material

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

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

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

c=0

c=30

c=300

c=90
Phong's Lighting Model

\[ I_R = L_{AR} \cdot M_{AR} + n^T \cdot l \cdot L_{DR} \cdot M_{DR} + (r^T \cdot v)^c \cdot L_{SR} \cdot M_{SR} \]

\[ I_G = L_{AG} \cdot M_{AG} + n^T \cdot l \cdot L_{DG} \cdot M_{DG} + (r^T \cdot v)^c \cdot L_{SG} \cdot M_{SG} \]

\[ I_B = L_{AB} \cdot M_{AB} + n^T \cdot l \cdot L_{DB} \cdot M_{DB} + (r^T \cdot v)^c \cdot L_{SB} \cdot M_{SB} \]

Ambient light approximation.
Phong's Lighting Model

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

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

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

Lambertian / diffuse reflectance
Phong's Lighting Model

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

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

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

Phong's specular reflectance term
Phong's Lighting Model

\[ I_R = L_{AR} \cdot M_{AR} + n^T \cdot l \cdot L_{DR} \cdot M_{DR} + (r^T \cdot v)^c \cdot L_{SR} \cdot M_{SR} \]

\[ I_G = L_{AG} \cdot M_{AG} + n^T \cdot l \cdot L_{DG} \cdot M_{DG} + (r^T \cdot v)^c \cdot L_{SG} \cdot M_{SG} \]

\[ I_B = L_{AB} \cdot M_{AB} + n^T \cdot l \cdot L_{DB} \cdot M_{DB} + (r^T \cdot v)^c \cdot L_{SB} \cdot M_{SB} \]

Something still missing?
Blinn-Phong model

- Sometimes Phong's specular term is replaced with Blinn-Phong's specular term.
Blinn-Phong model

- Sometimes Phong's specular term is replaced with Blinn-Phong's specular term.

- Instead of viewer direction and reflected light's direction, we use the **surface normal** and a **half angle vector** 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/](http://cgdemos.tume-maailm.pri.ee/)
The Standard Graphics Pipeline

- **Data**
- **Vertex transformations**
  - **Vertex shader**
  - **Culling & Clipping**
    - Perspective division, Viewport transformation
    - Culling – remember the face directions?
    - Clipping – some parts are out of view
- **Rasterization**
- **Fragment shading**
- **Visibility tests & Blending**
  - Fragment shader
Vertex Shader (1)

- Executed in parallel for each vertex
- Purpose is to transform the coordinates

At least OpenGL 4.0

```cpp
#version 400

uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;

layout(location=0) in vec3 position;

void main(void) {
    gl_Position = projectionMatrix * viewMatrix * modelMatrix * vec4(position, 1.0);
}
```
Vertex Shader (1)

- Executed in parallel for each vertex
- Purpose is to transform the coordinates

Uniforms are variables, which have the same values for all vertices

```glsl
#version 400

uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;

layout(location=0) in vec3 position;

void main(void) {
    gl_Position = projectionMatrix * viewMatrix * modelMatrix * vec4(position, 1.0);
}
```
Vertex Shader (1)

- Executed in parallel for each vertex
- Purpose is to transform the coordinates

```glsl
#version 400

uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;

layout(location=0) in vec3 position;

void main(void) {
    gl_Position = projectionMatrix * viewMatrix * modelMatrix * vec4(position, 1.0);
}
```

Primary input value is the vector with positional coordinates (different for each vertex)
Vertex Shader (1)

- Executed in parallel for each vertex
- Purpose is to transform the coordinates

Matrix-vector multiplication transforms the position from model's local space to clip space (and automatically later on to screen space)
Vertex Shader (2)

- Output variables will be interpolated to fragments

```glsl
#version 400
uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;

layout(location=0) in vec3 position;
layout(location=1) in vec3 color;
layout(location=2) in vec3 normal;

out vec3 interpolatedColor;
out vec3 interpolatedNormal;
out vec3 interpolatedPosition;

...  
```

Each vertex can have more different data assigned to it.
Vertex Shader (2)

- Output variables will be interpolated to fragments

```cpp
#version 400

uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;

layout(location=0) in vec3 position;
layout(location=1) in vec3 color;
layout(location=2) in vec3 normal;

out vec3 interpolatedColor;
out vec3 interpolatedNormal;
out vec3 interpolatedPosition;

... We can specify output variables, which we will need to assign and will be interpolated
We want to work in one specific space (usually it is the camera's space)

Normals need to be transformed a bit differently...

This code is pretty non-optimal... Makes a lot of unnecessary calculations...
We want to work in one specific space (usually it is the camera's space)

```cpp
void main(void) {
    mat3 normalMatrix = transpose(inverse(mat3(modelMatrix)));
    mat4 modelViewMatrix = viewMatrix * modelMatrix;

    gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    interpolatedNormal = normalMatrix * normal;
    interpolatedPosition = (modelViewMatrix * vec4(position, 1.0)).xyz;
    interpolatedColor = color;
}
```

We calculate and assign the values for our output variables.

This code is pretty non-optimal... Makes a lot of unnecessary calculations...
Fragment Shader (1)

- Executed in parallel for each fragment
- Purpose is to calculate the color value

```glsl
#version 400

out vec4 fragColor;

void main(void) {
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```
Fragment Shader (1)

- Executed in parallel for each fragment
- Purpose is to calculate the color value

Fragment shader's output variable will be the color

```glsl
#version 400

out vec4 fragColor;

void main(void) {
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```
Fragment Shader (1)

- Executed in parallel for each fragment
- Purpose is to calculate the color value

```
#version 400

out vec4 fragColor;

void main(void) {
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Everything rendered with this shader will be uniformly red
Fragment Shader (2)

- Uniforms can also be accessed here

```glsl
#version 400

uniform vec3 color;

out vec4 fragColor;

void main(void) {
    fragColor = vec4(color, 1.0);
}
```

Marginally better then the previous example
#version 400

uniform vec3 lightPosition;
uniform vec3 viewerPosition;
in vec3 interpolatedColor;
in vec3 interpolatedNormal;
in vec3 interpolatedPosition;

out vec4 fragColor;

void main(void) {

vec3 viewerPosition = vec3(0.0); // Camera space

vec3 n = normalize(interpolatedNormal);
vec3 l = normalize(lightPosition - interpolatedPosition);
vec3 v = normalize(viewerPosition - interpolatedPosition);
vec3 r = normalize(reflect(-1, n));

vec3 color = vec3(0.1, 0.1, 0.1) + max(0.0, dot(l, n)) * interpolatedColor + pow(max(0.0, dot(r, v)), 200.0);

fragColor = vec4(color, 1.0);

All positions and vectors need to be in the same space for the math to work.
What lighting model is this?
GLSL in WebGL

- WebGL is based on OpenGL 2.0
- Everything is pretty much the same, but instead of `in` and `out` you write `varying` variables.

Common values are prepended to this by Three.js

```html
<script id="phongVertexShader" type="x-shader/x-vertex">
    varying vec3 interpolatedPosition;
    varying vec3 interpolatedNormal;

    void main() {
        interpolatedPosition = (modelViewMatrix * vec4(position, 1.0)).xyz;
        interpolatedNormal = normalMatrix * normal;
        gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }
</script>
```
GLSL in WebGL

```glsl
<script id="phongFragmentShader" type="x-shader/x-fragment">
  uniform vec3 lightPosition;
  varying vec3 interpolatedPosition;
  varying vec3 interpolatedNormal;
  
  void main() {
    vec3 color = vec3(1.0, 0.0, 0.0);
    gl_FragColor = vec4(color, 1.0);
  }
</script>
```

In reality you'll do similar calculations here as before
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!