Lighting. Shading. Action!

and texturing too..
Review Rasterization

- Transform primitives into screen coordinates
- Use scanline conversion to rasterize
- Interpolate attributes across polygon
- Make sure to only include fragments that are INSIDE the polygon
Rasterization Demo
Perspective Correct interpolation

Example

\[ f = 0 \]
\[ Z_e = -2 \]

\[ f = 1 \]
\[ Z_e = -20 \]
Perspective Correct interpolation Example

\[ f = 0 \]
\[ w_c = 2 \]
\[ \alpha_1 = \alpha_2 \]
\[ f' \]
\[ f = 1 \]
\[ w_c = 20 \]
Perspective Correct interpolation Example

$f_0 = 0$
$w_0 = 2$

$f_1 = 1$
$w_1 = 20$

$\alpha_1 = \alpha_2$

$f' = \frac{\alpha_1}{\alpha_1 + \alpha_2} \frac{f_0}{w_0} + \frac{\alpha_2}{\alpha_1 + \alpha_2} \frac{f_1}{w_1}$
Perspective Correct interpolation Example

\[ f_0 = 0 \]
\[ w_0 = 2 \]
\[ f_1 = 1 \]
\[ w_1 = 20 \]

\[
\alpha_1 = \alpha_2
\]

\[
f' = \frac{0.5 \times \frac{0}{2} + 0.5 \times \frac{1}{20}}{0.5 \times \frac{1}{2} + 0.5 \times \frac{1}{20}}
\]

\[
= \frac{1}{40} \times \frac{10}{10} = \frac{1}{40}
\]
Interpolating normals, remember to normalize!
Pixels vs. Fragments

Pixels have:
- Color

Fragments have:
- Color
- Interpolated Depth
- Interpolated normals
- Interpolated view, light, and reflection vectors
- Interpolated texture coordinates
- ...

Bottom Line: fragments are combined to give final pixel values.
Lighting and Shading

- 3 main components: ambient, diffuse and specular
- Both materials AND lights control these properties
- We break up the intensity, $I$, into two components
  - The amount of light, $L$
  - The reflectance, $R$
- $I_* = L_* R_*$ for each component (ambient, diffuse, specular)
- $I_{final} = I_a + I_d + I_s$
- Although we write 1 equation, for each component, there is actually 1 equation *per color channel* (Red, Green, Blue)
Light Properties

- Three components $L_a, L_d, L_s$
  - Actually 9 ($L_{a,red}, L_{a,blue}, L_{a,green}, \ldots L_{s,blue}, L_{s,green}$)
- Set by `mglLight(MGLight light, MGLight_param pname, MGLfloat *values)`
  - Ex. Float `rgb[]` {1.0, 0.1, 0.2};
    `mglLight(MGL_LIGHT, MGL_LIGHT_SPECULAR, rgb);`
Light Properties

- Three components $L_a, L_d, L_s$
  - Actually 9 ($L_{a,red}, L_{a,blue}, L_{a,green}, \ldots, L_{s,blue}, L_{s,green}$)
- Set by `mglLight(MGLlight light, MGLlight_param pname, MGLfloat *values)`
  - Ex. Float rgb[] {1.0, 0.1, 0.2};
    `mglLight(MGL_LIGHT0, MGL_LIGHT_SPECULAR, rgb);`
Light Properties

- Three components $L_a, L_d, L_s$
- Actually 9 ($L_{a,red}, L_{a,blue}, L_{a,green}, \ldots L_{s,blue}, L_{s,green}$)
- Set by `mglLight(MGLlight light, MGLight_param pname, MGLfloat *values)`
  - Ex. Float `rgb[] {1.0, 0.6, 0.5};
mglLight(MGL_LIGHT0, MGL_LIGHT_SPECULAR, rgb);`
Material Properties

- Three components $k_a, k_d, k_s$
  - Actually 9 ($k_{a,red}, k_{a,blue}, k_{a,green}, \ldots k_{s,blue}, k_{s,green}$)
- Set by `mglMaterial(MGLmat_param pname, MGLfloat *values)`
  - Ex. `Float rg[] {1.0, 0.6, 0.5};
mglMaterial( MGL_MAT_DIFFUSE, rg);`
Material Properties

- Three components $k_a, k_d, k_s$
  - Actually 9 ($k_{a, red}, k_{a, blue}, k_{a, green}, \ldots k_{s, blue}, k_{s, green}$)
- Set by `mglMaterial(MGLmat_param pname, MGLfloat *values)`
  - Ex. Float `rgb[] {1.0, 0.6, 0.5};`  
    `mglMaterial(MGL_MAT_DIFFUSE, rgb);`

$k_{d, red} \quad k_{d, green} \quad k_{d, blue}$
Ambient

- $I_a = L_a \times k_a$
Diffuse

- $I_d = R_d L_d$
- $R_d = k_d (N \cdot L) = k_d \cos(\theta_d)$
Diffuse

- $I_d = R_d L_d$
- $R_d = k_d (N \cdot L) = k_d \cos(\theta_d)$

Material Color

Amount of light reflected
Specular

- Only view-dependent component
- \( I_s = L_S R_s \)
- \( R_s = k_s (R \cdot V)^\alpha \)

Material
Color

Amount of light reflected
Final Color

- Add them all up!
- $I = I_a + I_d + I_s$
Toon Shading

- Typically, $N \cdot L, R \cdot V \in [0,1]$
- Toon shading discretizes this range to get a “banded” effect
- Trivial to implement
  - Convert value of dot products to a step function
  - Ex. $N \cdot L, R \cdot V \in \{0, 0.1, 0.7, 1\}$
  - Play around with different functions
Toon Shading
Textures

- Textures are more than just images painted onto objects!
- Think of them as arbitrary *functions* that can be applied to surfaces
- Modern graphics techniques makes heavy use of textures for advanced rendering effects
  - Shadows Maps
  - Light maps
  - Normal Maps
  - What else?
Textures

- Each vertex has \((u,v) \in [0,1] \times [0,1]\), that index into a location in the texture.

- In OpenGL, texture mapping requires the specification of:
  - The texture image
  - The mapping between object and texture coordinates
  - Filtering mechanism (Assignment 2: Linear for minification, Linear or Nearest for Magnification)
  - Application (Assignment 2: Modulation)
Textures and Lighting

- In Assignment 2, we are just modulating the material color by the texture color

  \[ R_d = t_d(u, v)k_d \cos(\theta_d) \]
  \[ R_s = t_s(u, v)k_s(R \cdot V) \]
  \[ \text{If either texture is disabled, } t_*(u, v) = 1 \]
Texture Slots

- Notice that the texture applied to the diffuse component can be different from the one applied to the specular component.
- These are the two different texture “slots”: diffuse and specular.
- In MiniGL, textures for each slot are enabled and specified separately.
Texture Slots

Example

//Add a specular texture, turn off diffuse textures
mglTextureSlot(MGL_TEX_SPECULAR);
mglTexturesEnabled(true);
mglUseTexture(2);

mglTextureSlot(MGL_TEX_DIFFUSE)
mglTexturesEnabled(false);
Study the “earth” example
Texture Filtering

texels applied to 3D polygon

grid of pixels on screen

Minification

Magnification
Texture Filtering Methods

- **Nearest**: Round the interpolated \((u,v)\) coordinate to the nearest texel
- **Linear**: Perform weighted average 2x2 patch of texels
- For magnification, both Nearest and Linear give reasonable results
- For minification, filtering is absolutely necessary
Assignment 2

- Make sure you have a robust, perspective correct interpolation algorithm
- Need to interpolate a lot of things
  - Tex coords
  - Normal,
  - What else?
- All of the above should be stored along with your vertices.
  - Don’t forget to transform your normals by the Normal Matrix (see previous review session)