Student Presentation
Shading

- Shading approximates the physical properties of light
- What is the object’s color/material?
- How does light interact with the object?
- What is the camera’s position?

Toy Story (1995)
Big Hero 6 (2014)
Materials

- Dictates the way light interacts with the surface geometry
- Phong shading is a non-physically-based material model that roughly captures material properties

Ambient + Diffuse + Specular = Phong Reflection
Materials in Processing

- Ambient reflects flat light based on color parameters
  - `ambient(r, g, b)`
- Diffuse reflects based on angle to the light
  - Built into the lighting models
- Specular reflects based on the “shininess” of the object relative to the viewer direction
  - `specular(r, g, b) //color of highlights`
  - `shininess(s) //amount of highlight`
  - `lightSpecular(r, g, b) //specular light color`
Consider...

❖ What material properties will work for the following?
  ❖ A hotel wall
  ❖ The hood of a car
  ❖ An unglazed clay pot
  ❖ A glazed clay pot
❖ What other objects/material properties match well?
The Phong shading model can’t capture everything!

Many of the more “interesting” materials involve sub-surface scattering, or light bouncing off of multiple layers within the material…

Requires a more involved mathematical formula to replicate
Adding Detail

- Materials convey the underlying composition of the object, but how can we efficiently convey the surface color and patterns?
Textures

- Provides more detail across geometry
- Deforms with the geometry
- Mapping between geometry vertices \((x, y)\) and texture coordinates \((u, v)\)
PImage tex = loadImage("texture_file");
...
beginShape();
texture(tex);
vertex(x1, y1, z1, u1, v1);
vertex(x2, y2, z2, u2, v2);
vertex(x3, y3, z3, u3, v3);
vertex(x4, y4, z4, u4, v4);
endShape();
Exercise

Consider the previous in-class example. Outline the part of the image that will be displayed if the code is modified in the following way:

```cpp
texture(tex);
vertex(0, 0, 0, 0, 0);
vertex(350, 0, 0, .5, 0);
vertex(350, 200, 0, .5, 1);
vertex(0, 200, 0, 0, 1);
endShape();
```
(textureMode and textureWrap

- `textureMode(IMAGE)` sets mapping to number of pixels in texture image coordinates
- `textureMode(NORMAL)` sets mapping to normalized (0.0 - 1.0) texture image coordinates
- `textureWrap(CLAMP)` locks the texture into place
- `textureWrap(REPEAT)` repeats the texture along the surface)
Exercise

Consider the previous in-class example. How many times will the texture image be drawn if textureWrap is set to REPEAT and the vertices are modified as follows:

```cpp
vertex(0, 0, 0, 0, 0);
vertex(350, 0, 0, 3, 0);
vertex(350, 200, 0, 3, 4);
vertex(0, 200, 0, 0, 4);
```
Applying Textures to Meshes

- Possible to apply textures to meshes within Processing
  - Map all texture coordinates to vertices
  - Store in a GLModel (Java class for storing 3D model information in vertex buffers)
- But much easier to use 3D modeling programs like Blender or Maya!
OBJs and MTLs

- Create objects in .obj format and material properties in .mtl format then import into Processing
- How-to:
  - Processing -> File -> Examples -> Basics -> Shape -> LoadDisplayObj
Hands-on: Using Textures

❖ Today’s activities:

1. Recreate the scene you built for the last hands-on activities
2. Change the material properties of the 3D objects (modifying their shininess, ambience, and specularity)
3. Create a simple square or rectangle using Shape and apply a texture to it
4. Experiment with texture mode and texture wrapping options
5. Apply this texture to a more complicated shape