CS312

OpenGL
Lights and Materials
Light and Matter

From a physical perspective, a surface can either
- emit light by self-emission (as a light bulb)
- reflect light from other sources that illuminate it.
Interaction Between Light and Surfaces

- (a) specular
- (b) diffuse
- (c) translucent
Light Sources

- Light can leave a surface through
  - self-emission and reflection.

- What specifies a light source
  - position
  - direction
  - intensity
Color Sources

- Not only do light sources emit different amounts of light at different frequencies, but also their directional properties vary with frequency.

- Our visual system is based upon three primaries
  - For most applications, it is sufficient to reduce each light to a 3-component frequency:
    \[ I = aI_r + \beta I_g + gI_b \]
Ambient Lights

Lights that are designed and positioned to provide uniform illumination throughout the room (kitchens, classrooms).

Achieved with light sources that have diffusers whose purpose is to scatter light in all directions.
- Florescent lights have covers designed to do this.

To the lit surface, ambient light has no apparent direction.
Point Sources (Diffuse)

- An ideal point source emits light equally in all directions.
- To the lit surface, diffuse light is directional.
- The intensity of illumination proportional to the distance, and also depends on the angle of impact.
Spotlights

- Spotlights are characterized by a narrow range of angles through which light is emitted.
  - A spotlight can be constructed from a point source by limiting the angles
If the light source is far from the surface, the direction of light is uniform across the entire surface (the sun).
Parallel Light Rays

- Equivalent to a source that illuminates objects with parallel rays of light.
- Graphics systems can carry out rendering calculations more efficiently for distant light sources than for near ones.
  - OpenGL allows both
Material Properties

- Three different reflections
  - ambient
  - diffuse
  - specular
Ambient Reflection

- The intensity of ambient light is the same at every point on the surface.
  - Some light is absorbed and some is reflected.
  - A surface has of course, three ambient coefficients and they can be different.
  - Hence, a sphere appears yellow under white ambient light if its blue ambient coefficient is small and its red and green coefficients are large.
Diffuse Reflection

- A perfectly diffuse reflector scatters the light that it reflects equally in all directions.
- Perfectly diffuse surfaces are so rough that there is no preferred angle of reflection.
Specular Reflection

- Only ambient and diffuse reflections result in shaded but dull, somewhat chalk-like surfaces.
- The highlights
Normal Vectors

- The surface normal gives the orientation.
- Given 3 noncollinear points, normal is
  - \( n = (p2-p0) \times (p1-p0) \)
  - Be careful about the order of the vectors. Reversing the order changes the surface from outward pointing to inward pointing.
GL Normals

- Associate a normal with a vertex through functions such as
  - `glNormal3f(nx, ny, nz);`
  - `glNormal3fv(ptr_to_array);`
- Normals are modal: if we define a normal before a sequence of vertices, this normal is associated with all the vertices.

- Set the normal to have unit length so cosine calculations are correct
  - Length can be affected by transformations
  - `glEnable(GL_NORMALIZE)` allows for autonormalization at a performance penalty
Polygonal Shading

Consider the polygon mesh shown here. We will consider three ways to shade the polygons: flat, interpolative or Gourand, and Phong shading.
Flat Shading

- For a flat polygon, the normal is constant.
- The shading calculations only need to be carried out once for each polygon.
  - `glShadeModel(GL_FLAT);`
Interpolative and Gourand Shading

- The normals are computed at each vertex. Colors and intensities of interior points are interpolated between vertices.
  - `glShadeModel(GL_SMOOTH);`
Phong Shading

- Instead of interpolating the intensities, interpolate the normals.
- Then do calculation of intensities using the interpolated normal (typically at scan conversion).

- Interpolating normals is much more expensive than interpolating colors in Gourand Shading.
- Phong shading (e.g., per pixel shading) can be implemented using shaders in OpenGL.
- Usually done off-line (not supported in OpenGL).
Light Sources in OpenGL

- OpenGL supports the four types of light sources that we just described, and allows at least 8 light sources per program.
- Each light source must be individually specified and enabled.
  - `glLightfv(source, parameter, pointer_to_array);`
  - `glLightf(source, parameter, value);`
Light Parameters

- The position (or direction) of the light, the amount of ambient, diffuse, and specular light associated with a source.

```c
GL float diffuse0[]= {1.0, 0.0, 0.0, 1.0};
...
```

```c
glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

- Note that we must enable both lighting and all the particular source lights.
Direction and Position

- When specifying a light position, a light may either be directional (rays parallel), or positional.

```c
float light0_pos[] = {1.0,1.0,1.0,0.0};
gllightfv(GL_LIGHT0, GL_POSITION, light0_pos);
```

- If the 4\textsuperscript{th} value is 0 then the light is directional. Otherwise it is positional.
Other Lighting Functions

- Change lighting model
  - `glLightModel*(Param, value);`
  - `GL_LIGHT_MODEL_AMIENT, (0.2, 0.2, 0.2)`
  - `GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE`
  - `GL_LIGHT_MODEL_TWO_SIDED, GL_FALSE`
Spotlights

- Use `glLightf` to set
  - Direction: `GL_SPOT_DIRECTION`
  - Cutoff: `GL_SPOT_CUTOFF`
  - Exponent: `GL_SPOT_EXPONENT`
- Shininess controlled by \( \cos^{\alpha} \phi \)
Moving Light Sources

- Light sources are geometric objects whose positions or directions are affected by the model-view matrix.

- Depending on where we place the position (direction) setting function, we can:
  - Move the light source(s) with the object(s)
  - Fix the object(s) and move the light source(s)
  - Fix the light source(s) and move the object(s)
  - Move the light source(s) and object(s) independently.
Materials Specifications

- Material reflective parameters are specified through the functions:
  - `glMaterialfv(face, type, pointer_to_array);`
  - `glMaterialf(face, value);`

- For Example:
  - `glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);`
Material Properties

- To specify different front- and back-face properties
  - Use `GL_FRONT` or `GL_BACK`

- The shininess of a surface (specular-reflection term) is specified as follows:
  - `glMaterialf(GL_FRONT, GL_SHININESS, 100.0);`
Material Properties

GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[] = {1.0, 0.8, 0.0, 1.0};
GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat shine = 100.0

glMaterialf(GL_FRONT, GL_AMBIENT, ambient);
glMaterialf(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialf(GL_FRONT, GL_SPECULAR, specular);
glMaterialf(GL_FRONT, GL_SHININESS, shine);
Emissive Term

- We can simulate a light source in OpenGL by giving a material an emissive component.
- This color is unaffected by other light sources.

```glsl
GLfloat emission[] = 0.0, 0.8, 0.1, 1.0);
glMaterialf(GL_FRONT, GL_EMISSION, emission);
```

Red light + Green Emissive = Yellow light
Steps in OpenGL shading

1. Enable shading and select model
2. Specify normals
3. Specify material properties
4. Specify lights
Efficiency

- Because material properties are part of the state, if we change materials for many surfaces, we can affect performance.
- We can make the code cleaner by defining a material structure and setting all materials during initialization:

  ```c
  typedef struct materialStruct {
     GLfloat ambient[4];
     GLfloat diffuse[4];
     GLfloat specular[4];
     GLfloat shineness;
  } MaterialStruct;
  ```
- We can then select a material by a pointer.
Smooth Shading

- We can set a new normal at each vertex
- Easy for sphere model
  - If centered at origin $n = p$
- Now smooth shading works
- Note *silhouette edge*
Gouraud and Phong Shading

- **Gouraud Shading**
  - Find average normal at each vertex (vertex normals)
  - Apply Phong model at each vertex
  - Interpolate vertex shades across each polygon

- **Phong shading**
  - Find vertex normals
  - Interpolate vertex normals across edges
  - Find shades along edges
  - Interpolate edge edge shades across polygons
Comparison

- If the polygon mesh approximates surfaces with a high curvatures, Phong shading may look smooth while Gouraud shading may show edges.
- Phong shading requires much more work than Gouraud shading.
  - Usually not available in real time systems.
- Both need data structures to represent meshes so we can obtain vertex normals.