ILLUMINATION

Class 5
Shading models

- Simulate physical interaction of light and matter.
- Physically correct shading models are complex and expensive
  - Light properties
  - Material properties
  - Inter-object relations
- Good approximations are possible
  - Approximations of physical laws
  - Heuristics that look good
Light Sources

- Point light source (A)
- Parallel light source (B)
- Area light source (C)
Point Light Source

- Light originates from a point
- Represented by position and color
- The point may be at infinity
- Approximate light sources whose dimensions are small relative to the objects
Parallel Light Source

- Light rays are parallel
- Represented by direction and color
- Can be modeled as a point light source at infinity
- Approximate far and large light sources
Area Light Source

- Light originates at a finite area
  - Window
  - Fluorescent
- Sometimes called distributed source
Parallel Vs. Point Source

- Parallel light source
- Point light source
Intensity Attenuation

- Radial intensity attenuation:
  \[ f_r = \frac{1}{a_0 + a_1 d + a_2 d^2} \]

- Angular intensity attenuation:
  \[ f_\alpha = (V_{obj} \cdot V_{light})^n \]

- How can we represent a spotlight?
Object illumination is affected by the following:

- Ambient light
- Diffuse reflection
- Specular reflection
Ambient Light

- Assumes non directional light in the background
- Object illuminated with same light everywhere
- The illumination equation:
  \[ I_{amb} = I_a K_a, \]

  \( I_a \) – ambient light intensity

  \( K_a \) – fraction of ambient light reflected from surface
Diffuse Reflection

- Reflects incoming light uniformly in all directions
- Known as Lambert reflection
- Represents matte (non shining) surfaces
- Reflected light is proportional to $\langle N \cdot L \rangle = \cos(\theta)$
Diffuse Reflection

- Illumination due to point light source:
  \[ I_{diff} = I_p K_d \langle N \cdot L \rangle = I_p K_d \cos(\theta), \]
  \( I_p \) – point light intensity
  \( K_d \) – diffuse-reflection coefficient

- Total illumination:
  \[ I = I_{amb} + I_{diff} = I_a K_a + I_p K_d \langle N \cdot L \rangle \]
Specular Reflection

- Shiny (metallic) surfaces reflect light in a preferred direction R and its vicinity
- Ideal shiny surface (mirror) reflects only in the direction R
- **Phong model**: attenuation is relative to \( (R \cdot V)^n = \cos^n(\alpha) \) (no physical basis):
Specular Reflection

- Illumination due to point light source:
  \[ I_{spec} = I_p K_s (R \cdot V)^n = I_p K_s \cos^n(\alpha), \]
  - \( I_p \) – point light intensity
  - \( K_s \) – specular-reflection coefficient
  - \( n \) – specularity exponent

- Calculation of \( R \):
  \[ R = 2 \langle N \cdot L \rangle N - L \]
Illumination Equation

\[ I = \sum_i [I_{a_i}K_a + I_{p_i}K_d\cos(\theta) + I_{p_i}K_s\cos^n(\alpha)] \]

- \( N \) – point normal
- \( V \) – viewing direction
- \( L \) – lighting direction
- \( R \) – reflection direction
- \( d \) – diffuse, \( a \) – ambient, \( s \) – specular
- \( I \) – intensity
- \( K \) – surface coefficient,
- \( \theta \) – angle between \( N \) and \( L \)
- \( \alpha \) – angle between \( V \) and \( R \)
Illumination comparison

- Ambient
- Specular, $n = 100$
- Diffuse
- Specular, $n = 8$
Surface Rendering Methods

- Constant (Flat)
- Gouraud
- Phong

- Also named ‘Shading’
Constant Surface Rendering

- Color each polygon according to its normal
  - Piecewise linear polygonal models
  - Light source is far from the object
  - Viewer is far from the object
Gouraud Surface Rendering

- Gouraud’s method:
  - Compute average normal on each vertex
  - Compute light intensity on each vertex
  - Use linear interpolation to compute light intensity on a point on the polygon
    - Interpolation is done on image plane using scan-conversion

- Can it support specular highlights?
Phong Surface Rendering

- Phong’s method:
  - Compute average normal on each vertex
  - Use linear interpolation to compute normal on a point on the polygon
    - Interpolation is done on image plane using scan-conversion
  - Compute light intensity on each point
- Expensive but more accurate
Rendering Quiz
LIGHTING IN OSG
Lights in the graph

The graph
TODO

- Specify normals with your geometry
- Enable lighting and set other lighting state
- Specify the light source properties
  - attach it to your scene graph
- Specify surface material properties
Normals

- Enable normal rescaling
  - uniform scaling

```cpp
osg::StateSet* state = geode->getOrCreateStateSet();
state->setMode( GL_RESCALE_NORMAL, osg::StateAttribute::ON );
```

- Enable normalization
  - non-uniform scaling

```cpp
osg::StateSet* state = geode->getOrCreateStateSet();
state->setMode( GL_NORMALIZE, osg::StateAttribute::ON );
```
Lighting State

- Enable lighting
- Enable at least one light source

```cpp
osg::StateSet* state = root->getOrCreateStateSet();
state->setMode( GL_LIGHTING, osg::StateAttribute::ON );
state->setMode( GL_LIGHT0, osg::StateAttribute::ON );
state->setMode( GL_LIGHT1, osg::StateAttribute::ON );
```
Light Sources

- Create an `osg::Light`
  - `GL_LIGHT0, ..., GL_LIGHT7`
- Add the `Light` to an `osg::LightSource` node
- Add the `LightSource` node to your scene graph

`LightSource` is effectively a leaf node containing a single `Light` definition.
Adding light to graph

// Light Source node...
osg::ref_ptr<osg::LightSource> ls = new osg::LightSource;

//... is added to the root
root->addChild( ls.get() );

// Light Source holds Light
osg::ref_ptr<osg::Light> light = new osg::Light;
light->setLightNum(2); // Indicate which light (Default – 0)
ls->setLight( light.get() );
Light Properties

// Create a white spot light source
osg::ref_ptr<osg::Light> light = new osg::Light;
light->setAmbient( osg::Vec4( .1f, .1f, .1f, 1.f )); // (r,g,b,1)
light->setDiffuse( osg::Vec4( .8f, .8f, .8f, 1.f )); // (r,g,b,1)
light->setSpecular( osg::Vec4( .8f, .8f, .8f, 1.f )); // (r,g,b,1)
light->setPosition( osg::Vec4( 0.f, 0.f, 0.f, 1.f )); //(x,y,z,w)
light->setDirection( osg::Vec3( 1.f, 0.f, 0.f )); //(x,y,z)
light->setSpotCutoff(25.f);
light->setConstantAttenuation(0.5f);
Light Position

- By current transformation of LightSource
  - Default

- Absolute value

```cpp
go!::ref_ptr<go!::LightSource> ls = new go!::LightSource;
ls->setReferenceFrame( go!::LightSource::ABSOLUTE_RF );
```
Material Properties

- Create a Material object
- Set colors and other parameters
- attach it to a StateSet in your scene graph
Material

```cpp
osg::StateSet* state = node->getOrCreateStateSet();
osg::ref_ptr<osg::Material> mat = new osg::Material;
...
state->setAttribute(mat.get());
```

- **Faces:**
  - `osg::Material::FRONT`
  - `osg::Material::BACK`
  - `osg::Material::FRONT_AND_BACK`
  - When do we need back faces?
mat->setDiffuse( osg::Material::FRONT, osg::Vec4( .2f, .9f, .9f, 1.f ) );
// (r,g,b,1)
mat->setSpecular( osg::Material::FRONT, osg::Vec4( 1.f, 1.f, 1.f, 1.f ) );
// (r,g,b,1)
mat->setShininess( osg::Material::FRONT, 96.f ); // 1-128
mat->setAlpha( osg::Material::FRONT, 0.5f ); // 0-1
mat->setEmission(Face,osg::Vec4(1.f, 1.f, 1.f, 1.f )); // 0-1
Shade model

osg::ShadeModel* sm = new osg::ShadeModel();
sm->setMode( osg::ShadeModel::FLAT);
state->setAttribute( sm );

- Shade Models:
  - FLAT
  - SMOOTH