RGB Mode and Index Mode

• If each pixel value in the frame buffer is an RGB vector, *RGB mode*.

• Each pixel value an index into a color look-up table called a *colormap, index mode*.

• The pixel color is specified in the colormap instead of the frame buffer.
Color-index mode a memory trade-off except:

- If you’re porting an existing application
- Tricks as color-map animation and drawing in layers.

<table>
<thead>
<tr>
<th>Frame buffer</th>
<th>Colormap</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Frame buffer diagram" /></td>
<td><img src="image2.png" alt="Colormap diagram" /></td>
</tr>
</tbody>
</table>

- **R** | **G** | **B** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

black: 00000000
red: 11100000
**Color Interpolation**

- \texttt{glShadeModel(GL\_FLAT)}, only one color in a primitive (the last vertex in a primitive)

- \texttt{glShadeModel(GL\_SMOOTH)}, color interpolation
  
  - For a line, the vertex colors are linearly interpolated along the pixels between the two end vertices.
  
  - a line has 5 pixels, and the end point colors are (0,0,0) and (0,0,1), then, after the interpolation, the 5 pixel colors will be (0,0,0), (0,0,1/4), (0,0,2/4), (0,0,3/4), and (0,0,1).
Discrete interpolation

• each RGB component is interpolated separately

• Given end point intensities ($I_{\lambda,1}$ and $I_{\lambda,2}$) and the number of pixels ($N$), the intensity increment of the linear interpolation is:

$$\Delta I_{\lambda} = \frac{(I_{\lambda,2} - I_{\lambda,1})}{(N-1)}$$

• For a polygon, OpenGL first interpolates along the edges, and then along the horizontal scan-lines during scan-conversion.
Continuous interpolation

- each RGB component is interpolated separately

- Given end point intensities \((I_{\lambda 1} \text{ and } I_{\lambda 2})\) and the distance to the end point is \(0 \leq \alpha \leq 1\), the intensity of the linear interpolation is:

\[
I_{\lambda} = (1-\alpha)*I_{\lambda 1} + \alpha* I_{\lambda 2}
\]
/* Example 3.1.shading.c: OpenGL flat or smooth shading */

void drawtriangle(float *v1, float *v2, float *v3)
{
    glBegin(GL_TRIANGLES);
        glColor3f(1,0,0); glVertex3fv(v1)
        glColor3f(0,1,0); glVertex3fv(v2);
        glColor3f(0,0,1); glVertex3fv(v3);
    glEnd();
}

drawColorCoord(float xlen, float ylen, float zlen)
{
    // coordinate lines
    glBegin(GL_LINES);
        glVertex3f(0,0,0);glVertex3f(0,0,zlen);
        glVertex3f(0,0,0);glVertex3f(0,ylen, 0);
        glVertex3f(0,0,0);glVertex3f(xlen,0,0);
    glEnd();

    // coordinate labels: X, Y, Z
    glPushMatrix();
        glTranslatef(xlen,0.,0.);
        glScalef(xlen/Width,xlen/Width,1);
        glutStrokeCharacter(GLUT_STROKE_ROMAN, 'X');
    glPopMatrix();
}
void display(void)
{

cnt++;
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // alternating between flat & smooth
    if (cnt % 50 == 0) glShadeModel(GL_SMOOTH);
    if (cnt%100 == 0) glShadeModel(GL_FLAT);

drawColorCoord(1.);
drawtriangle(vdata[0], vdata[1], vdata[2]);

glutSwapBuffers();
}
ILLUMINATION (LIGHTING)

LIGHTING MODELS FOR A POINT

• Have different illumination models (shading models, lighting models); the models have no grounding in theory, but work well.

SHADING MODELS FOR A POLYGON/MODEL

• Shade surfaces based on the position, orientation, and characteristics of the surfaces and the light sources illuminating them.

• Approximation methods: Interpolation
**NORMAL VECTORS**

1. A normal vector (or normal) points in a direction perpendicular to a surface (in general).
2. An object’s normal vectors define its orientation relative to light sources. These vectors are used by OpenGL to determine how much light the object receives at its vertices.
3. Lighting for vertices is calculated after MODELVIEW transformation before PROJECTION transformation (vertex shading)

```c
glBegin (GL_POLYGON);
  glNormal3fv(n0);
  glVertex3fv(v0);
  glNormal3fv(n1);
  glVertex3fv(v1);
  glNormal3fv(n2);
  glVertex3fv(v2);
  glNormal3fv(n3);
  glVertex3fv(v3);
glEnd();
```
Transforming Normals

• A point \( \mathbf{v} = (x, y, z, w)^T \) is on a plane \( \mathbf{p} = (a, b, c, d) \) if
  \[
  ax + by + cz + d = 0
  \]
  \( \mathbf{p} \mathbf{v} = 0 \)

• if \( \mathbf{v} \) is transformed by matrix \( \mathbf{M} \), then
  \[
  \mathbf{p} \mathbf{M}^{-1} \mathbf{M} \mathbf{v} = 0
  \]

• So \( \mathbf{M} \mathbf{v} \) lies in the plane \( \mathbf{p} \mathbf{M}^{-1} \). The normal \( \mathbf{n} = \mathbf{p} \). The transformed normal \( \mathbf{N} \) along with \( \mathbf{v} \) is:
  \[
  \mathbf{N} = \mathbf{n} \mathbf{M}^{-1}
  \]
  that is
  \[
  \mathbf{N}^T = (\mathbf{M}^{-1})^T \mathbf{n}^T
  \]

• **Conclusion**: normals are transformed by the inverse transpose of the matrix that transforms the points.

```cpp
glEnable(GL_REESCALE_NORMAL); // rescale back to unit.
  // Normals are affected by scale only
glEnable(GL_NORMALIZE); // automatically normalize
```

Copyright @ 2001 by Jim X. Chen:
  jchen@cs.gmu.edu
**Lighting Components**

**Emissive Component**

\[ I_{\lambda e} = M_{\lambda \text{emission}} \]

- where \( \lambda \) is an RGB component or A (alpha), and \( M_{\lambda \text{emission}} \) is the material’s emission property.
- Each color component is calculated independently.
- In OpenGL, *emission* is a material property that is neither dependent on any light source nor considered a light source.
- Emissive material does not emit light, it displays its own color.
- The vertex’s corresponding surface has two sides, the front and the back, which can be specified with different material properties.
**Ambient Component**

- Multiple reflections of light from the environment
- ambient light impinges equally on all surfaces from all directions.

\[ I_{\lambda a} = L_{\lambda a} M_{\lambda a} \]

where \( L_{\lambda a} \) represents the light source’s ambient intensity and \( M_{\lambda a} \) is the material’s ambient property.
**Diffuse Component**

Diffuse color is the reflection from a dull surface material that appears equally bright from all viewing directions.

$$I_{\lambda d} = L_{\lambda d} M_{\lambda d} (n \cdot L)$$

where $L_{\lambda d}$ is the light source’s diffuse intensity, $M_{\lambda d}$ is the material’s diffuse property, $L$ is the light source direction, and $n$ is the surface normal direction.

$\cos \theta$ is between $0$ and $1$ when $\theta$ is between $0^\circ$ and $90^\circ$. When $\theta$ is greater than $90^\circ$, the diffuse intensity is set to zero.

$$\cos \theta = \frac{n \cdot L}{|n|}$$
\[ I_{\lambda d} = L_{\lambda d} M_{\lambda d} (n \cdot L) \]

- we can call `glEnable(GL_NORMALIZE)`, which enables the OpenGL system to normalize each normal before calculating the lighting.

- If a point light source is sufficiently distant from the objects being shaded, it makes essentially the same angle with all surfaces sharing the same surface normal.
The radius and the height of the cone are the same (unit length)

\[ n_1 = v_1 + v_3, \quad n_2 = v_2 + v_3, \quad \text{and} \quad n_3 = n_1 + n_2. \]

```c
void drawConeSide(float *v1, float *v2, float *v3) {
    float v11[3], v22[3], v33[3]; int i;
    for (i=0; i<3; i++) {
        v11[i] = v1[i] + v3[i]; // normal for cone vertex 1
        v22[i] = v2[i] + v3[i]; // normal for cone vertex 2
        v33[i] = v11[i] + v22[i]; // normal for cone vertex 3
    }
    glBegin(GL_TRIANGLES);
    glNormal3fv(v11);
    glVertex3fv(v1);
    glNormal3fv(v22);
    glVertex3fv(v2);
    glNormal3fv(v33);
    glVertex3fv(v3);
    glEnd();
}
```

The radius and the height of the cone are the same (unit length)
**Specular Component**

Highlight reflection from a smooth-surface material that depends on the reflection direction $R$ (which is $L$ reflected along the normal) and the viewing point direction $V$.

$$I_{\lambda s} = L_{\lambda s}M_{\lambda s}\left(\frac{N \cdot (L + V)}{|L + V|}\right)^{\text{shininess}}$$

where $L_{\lambda s}$ is the light source’s specular intensity, $M_{\lambda s}$ is the material’s specular property, and *shininess* is the material’s shininess property.

$$\cos \alpha = \frac{N \cdot (L + V)}{|L + V|}$$
The viewpoint, as we discussed in the viewing transformation, is at the origin (facing the negative z axis).

We use `glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE)` to specify the viewpoint at $(0, 0, 0)$.

OpenGL allows us to specify the viewpoint at infinity. Since this assumption is only used to simplify lighting calculations, the viewpoint is not changed for other graphics calculations, such as projection.

\[
\cos \alpha = \frac{N \cdot (L + V)}{|L + V|}
\]
**Lighting Model**

\[
I_\lambda = I_{\lambda e} + I_{\lambda a} + I_{\lambda d} + I_{\lambda s} \\
= I_{\lambda e} + I_{\lambda L}
\]

While ambient, diffuse, and specular intensities depend on the light source, emissive intensity does not.
Movable Light Source

- Lighting is calculated according to the transformed position.
- To simulate a visible light source, we can specify the light source and draw an object at the same position.
- Lighting is calculated after ModelView transformation.

J3_7_MoveLight
Spot Light Effect

- A real light source may not generate equal intensity in all directions:

\[ I_\lambda = I_{\lambda e} + f_{\text{spot}} I_{\lambda L} \]

where \( f_{\text{spot}} \) is called the *spot light effect factor*. In OpenGL, it is calculated as follows:

\[ f_{\text{spot}} = (-L \cdot D_{\text{spot}})^{\text{spotExp}} \]

where \((-L)\) is a unit vector pointing from the light source to the vertex pixel, \(D_{\text{spot}}\) is the direction of the light source, and \(\text{spotExp}\) is a specified constant.
\[ \mathbf{I}_\lambda = \mathbf{I}_{\lambda e} + f_{\text{spot}} \mathbf{I}_\lambda \mathbf{L} \]

\[ f_{\text{spot}} = \left( -\mathbf{L} \cdot \mathbf{D}_{\text{spot}} \right)^{\text{spotExp}} \]

where \((-\mathbf{L})\) is a unit vector pointing from the light source to the vertex pixel, \(\mathbf{D}_{\text{spot}}\) is the direction of the light source, and \(\text{spotExp}\) is a specified constant.

\[ \cos \gamma = \frac{(-\mathbf{L}) \cdot \mathbf{D}_{\text{spot}}}{|\mathbf{D}_{\text{spot}}|} \]
Light Source Attenuation

The intensity from a point light source to a vertex pixel can be attenuated by the distance $d_L$ the light travels:

$$I_\lambda = I_{\lambda e} + f_{\text{att}} f_{\text{spot}} I_{\lambda L}$$

where $f_{\text{att}}$ is called the light source attenuation factor. In OpenGL, $f_{\text{att}}$ is calculated as follows:

$$f_{\text{att}} = \frac{1}{A_c + A_1 d_L + A_q d_L^2}$$
Multiple Light Sources

We can also specify $K$ multiple light sources:

$$I_\lambda = I_{\lambda e} + \sum_{i=0}^{K-1} f_{\text{att}i} f_{\text{spot}i} I_{\lambda L_i}$$

J3_10_Lights
OpenGL Lighting

Defining Light Source Properties

• `void glLight{if}[v](GLenum light, GLenum pname, TYPE param)`
  – light can be GL_LIGHT0, GL_LIGHT1, ..., or GL_LIGHT7.
  – pname specifies a named parameter.
  – param indicates the values to which the pname characteristic is set;
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_AMBIENT</td>
<td>(0.0, 0.0, 0.0, 1.0)</td>
<td>ambient RGBA</td>
</tr>
<tr>
<td>GL_DIFFUSE</td>
<td>(1.0, 1.0, 1.0, 1.0)</td>
<td>diffuse RGBA</td>
</tr>
<tr>
<td>GL_SPECULAR</td>
<td>(1.0, 1.0, 1.0, 1.0)</td>
<td>specular RGBA</td>
</tr>
<tr>
<td>GL_POSITION</td>
<td>(0.0, 0.0, 1.0, 0.0)</td>
<td>position of light</td>
</tr>
<tr>
<td>GL_SPOT_DIRECTION</td>
<td>(0.0, 0.0, -1.0)</td>
<td>direction of spotlight</td>
</tr>
<tr>
<td>GL_SPOT_EXPONENT</td>
<td>0.0</td>
<td>spotlight exponent</td>
</tr>
<tr>
<td>GL_SPOT_CUTOFF</td>
<td>180.0</td>
<td>spotlight cutoff angle</td>
</tr>
<tr>
<td>GL_CONSTANT_ATTENUATION</td>
<td>1.0</td>
<td>constant attenuation factor</td>
</tr>
<tr>
<td>GL_LINEAR_ATTENUATION</td>
<td>0.0</td>
<td>linear attenuation factor</td>
</tr>
<tr>
<td>GL_QUADRATIC_ATTENUATION</td>
<td>0.0</td>
<td>quadratic attenuation factor</td>
</tr>
</tbody>
</table>

\[(\text{Max}\{V \cdot d, 0\})^{\text{GL_SPOT_EXPONENT}}\]

V is the normalized direction of the vertex from GL_SPOT_POSITION.
Directional and Positional light source

```c
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- If \( w \) is zero, the corresponding light source is directional (at infinite), & the \((x, y, z)\) values describe its direction. This direction is transformed by the modelview matrix.

- If the \( w \) value is nonzero, the light is positional, and the \((x, y, z)\) values specify the location of the light in homogeneous object coordinates. This location is transformed by the modelview matrix and stored in eye coordinates.
Attenuation factor = \( \frac{1}{k_c + k_l \cdot d + k_q \cdot d^2} \)

where

\[ d = \text{distance between the light's position and the vertex} \]

\[ k_c = \text{GL_CONSTANT_ATTENUATION} \]
\[ k_l = \text{GL_LINEAR_ATTENUATION} \]
\[ k_q = \text{GL_QUADRATIC_ATTENUATION} \]

\[
\begin{align*}
glLightf(GL\_LIGHT0, GL\_CONSTANT\_ATTENUATION, 2.0); \\
glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, 1.0); \\
glLightf(GL\_LIGHT0, GL\_QUADRATIC\_ATTENUATION, 0.5); \\
\end{align*}
\]

- The ambient, diffuse, and specular values are all attenuated. Only the emission and global ambient values aren’t attenuated.
- Emitted light: it originates from an object and is unaffected by any light sources.
Selecting an OpenGL Lighting Model

• Global Ambient Light (not from any particular source)
  
  \[
  \text{GLfloat lmodel\_ambient[] = \{ 0.2, 0.2, 0.2, 1. \};}
  \text{glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, lmodel\_ambient);}
  \]

• Local or Infinite Viewpoint
  
  A local viewpoint tends to more realistic results (specular highlights), but overall performance is decreased:

  \[
  \text{glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE);}\]

  This call places the viewpoint at (0, 0, 0) in eye coordinates.

• Two-sided Lighting
  
  \[
  \text{glLightModeli(LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);}\]

  both front-facing polygons and back-facing polygons are lit. OpenGL reverses the normals to calculate the lighting for back-facing polygons.
Defining Material Properties

```c
void glMaterial{if}[v](GLenum face, GLenum pname, TYPEparam);
```

- `face`: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK.
- Material properties are similar to creating light sources
  GL_SHININESS; GL_EMISSION; GL_AMBIENT_AND_DIFFUSE;

Color Material Mode

```c
void glColorMaterial(Glenum face, Glenum mode);
```

- `face`: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK.
- `mode`: GL_AMBIENT, GL_DIFFUSE, GL_EMISSION; etc.

Need to call `glEnable(GL_COLOR_MATERIAL)`, then you can change the current color using glColor*() to change the material properties.
The Mathematics of Lighting

• Spotlight Effect

Evaluates to one of 3 possible values:

- 1 if it isn’t a spotlight (GL_SPOT_CUTOFF is 180.0).
- 0 if the vertex lies outside the cone of illumination.
- \((\max \{v \cdot d, 0\})^{\text{GL_SPOT_EXPONENT}}\) where: 
  - \(v\) is unit from the spotlight (GL_POSITION) to the vertex;
  - \(d\) is the spotlight’s direction (GL_SPOT_DIRECTION)
• **Specular Term**

\[(\max \{s \cdot n, 0\})^{\text{shininess}} \cdot \text{specular}_{\text{light}} \cdot \text{specular}_{\text{material}}\]

\[s = L + V\]

if GL_LIGHT_MODEL_LOCAL_VIEWER is not true, \(V = (0, 0, 1)\)

\[\cdot\text{However, if } L \cdot n = 0, \text{ the specular term is } 0.\]

• **Together a vertex color**

\[= \text{emission}_{\text{material}} + \text{ambient}_{\text{light_model}} \cdot \text{ambient}_{\text{material}} + \sum_{i=0}^{n-1} \left\{ \left(\frac{1}{k_c + k_l \cdot d + k_q \cdot d^2}\right)^i \cdot \text{(spotlight effect)}_i \right\} \cdot \left[\text{ambient}_{\text{light}} \cdot \text{ambient}_{\text{material}} + \right.\]

\[\left. (\max\{L \cdot n, 0\}) \cdot \text{diffuse}_{\text{light}} \cdot \text{diffuse}_{\text{material}} + \right.\]

\[\left. (\max\{s \cdot n, 0\})^{\text{shininess}} \cdot \text{specular}_{\text{light}} \cdot \text{specular}_{\text{material}}\right\}_i\]
Lighting Equation

\[ I_\lambda = I_{\lambda e} + \sum_{i=0}^{K-1} f_{\text{att}i} f_{\text{spot}i} I_{\lambda L_i} \]

\[
f_{\text{att}} = \frac{1}{A_c + A_1d_L + A_q d_L^2}
\]

\[
f_{\text{spot}} = (\max\{(-L \cdot D_{\text{spot}}), 0\})^{\text{spotExp}}
\]

\[ I_{\lambda L} = I_{\lambda a} + I_{\lambda d} + I_{\lambda s} \]

\[ I_{\lambda a} = L_{\lambda a} M_{\lambda a} \]

\[ I_{\lambda d} = L_{\lambda d} M_{\lambda d} \max\{(n \cdot L/|L|), 0\} \]

\[ I_{\lambda s} = L_{\lambda s} M_{\lambda s} \left( \frac{n \cdot (L + V)}{|L + V|} \right)^{\text{shininess}} \]
**HW**

Implement the following using your homework and with `myPerspective` and `myLookat`:

1. Three movable light sources that goes with the three moons;
2. Look at from the earth to the sun;
3. Design your own light sources;
4. Design your own material properties.