Computer Graphics Lighting

Why Do We Care About Lighting?

Lighting "dis-ambiguates" 3D scenes

The Surface Normal

A surface normal is a vector perpendicular to the surface.

Sometimes surface normals are defined or computed per-face:

\[ n = (P1 - P0) \times (P2 - P0) \]

Sometimes they are defined per-vertex to best approximate the underlying surface that the face is representing.

Setting a Surface Normal in OpenGL

```
glMatrixMode(GL_MODELVIEW);
glTranslatef(tx, ty, tz);
glRotatef(degrees, ax, ay, az);
glScalef(sx, sy, sz);
glNormal3f(nx, ny, nz);
glColor3f(r, g, b);
glBegin(GL_TRIANGLES);
glVertex3f(x0, y0, z0);
glVertex3f(x1, y1, z1);
glVertex3f(x2, y2, z2);
glEnd();
```

Flat Shading (Per-face)

```
glMatrixMode(GL_MODELVIEW);
glTranslatef(tx, ty, tz);
glRotatef(degrees, ax, ay, az);
glScalef(sx, sy, sz);
glShadeModel(GL_FLAT);
glNormal3f(nx, ny, nz);
glColor3f(r, g, b);
glBegin(GL_TRIANGLES);
glVertex3f(x0, y0, z0);
glVertex3f(x1, y1, z1);
glVertex3f(x2, y2, z2);
glEnd();
```
glMatrixMode( GL_MODELVIEW );
glTranslatef( tx, ty, tz );
glRotatef( degrees, ax, ay, az );
glScalef( sx, sy, sz );
glShadeModel( GL_SMOOTH );
glColor3f( r, g, b );
glBegin( GL_TRIANGLES );
glNormal3f( nx0, ny0, nz0 );
glVertex3f( x0, y0, z0 );
glNormal3f( nx1, ny1, nz1 );
glVertex3f( x1, y1, z1 );
glNormal3f( nx2, ny2, nz2 );
glVertex3f( x2, y2, z2 );
glEnd();

OpenGL Surface Normals Need to be Unitized by Someone

OpenGL expects the normal vector to be a unit vector, that is: nx^2 + ny^2 + nz^2 = 1
If it is not, or if you are using scaling transformations, you can force
OpenGL to do the unitizing for you with:
glEnable( GL_NORMALIZE );

The OpenGL “built-in” Lighting Model

1. Ambient = a constant  Accounts for light bouncing “everywhere”
2. Diffuse = I * cosΘ  Accounts for the angle between the incoming light and the surface normal
3. Specular = I * cosS
   Accounts for the angle between the “perfect reflector” and the eye. The exponent, S, accounts for surface shininess

Note that cosΘ is just the dot product between unit vectors L and n
Note that cosS is just the dot product between unit vectors R and E

You are all familiar with the Diffuse Lighting effects
Diffuse Lighting actually works because of spreading out the same amount of light energy across more surface area

\[ \text{Diffuse} = I \cdot \cos \theta \]

You are all familiar with the Specular Lighting effects

These are not actually metal. They are wood with special paint that mimics the metallic reflection highlights.

We can mimic the same effects digitally!

The Specular Lighting equation is a heuristic that approximates reflection from a rough surface

\[ \text{Specular} = I \cdot \cos S \]

These all have metallic-looking surfaces. What tells you that?

It's the shiny-reflection spots.

The Three Elements of OpenGL Lighting

\[ \text{Ambient} + \text{Diffuse} + \text{Specular} = \text{Result} \]

Types of Light Sources

Point

Directional (Parallel, Sun)

Spotlight
**Lighting Examples**

**Point Light at the Eye**

**Point Light at the Origin**

**Spot Lights**

**Colored Lights Shining on Colored Objects**

Colored Lights shining on colored objects.

**White Light**

**Green Light**

**Too Many Lighting Options**

If there is one light and one material, the following things can be set independently:

- Global scene ambient red, green, blue
- Light position: x, y, z
- Light ambient red, green, blue
- Light diffuse red, green, blue
- Light specular red, green, blue
- Material reaction to ambient red, green, blue
- Material reaction to diffuse red, green, blue
- Material reaction to specular red, green, blue
- Material specular shininess

This makes for 25 things that can be set for just one light and one material! While many combinations are possible, some make more sense than others.

**Ways to Simplify Too Many Lighting Options**

1. Set the ambient light globally using, for example, `glLightModeliv(GL_LIGHT_MODEL_AMBIENT, MulArray3(.3f, White))` i.e., set it to some low intensity of white.
2. Set the light’s ambient component to zero.
3. Set the light’s diffuse and specular components to the full color of the light.
4. Set each material’s ambient and diffuse to the full color of the object.
5. Set each material’s specular component to some fraction of white.

```c
float White[4] = {1.,1.,1.,1.};
// utility to create an array from 3 separate values:
float *Array3(float a, float b, float c)
{
    static float array[4];
    array[0] = a;
    array[1] = b;
    array[2] = c;
    array[3] = 1.;
    return array;
}
// utility to create an array from a multiplier and an array:
float *MulArray3(float factor, float array0[3])
{
    static float array[4];
    array[0] = factor * array0[0];
    array[1] = factor * array0[1];
    array[2] = factor * array0[2];
    array[3] = 1.;
    return array;
}
```

The 4th element of the array being set to 1.0 is there on purpose. The reason is coming up soon!
**Setting the Material Characteristics**

```c
// Back
glMaterialfv(GL_BACK, GL_AMBIENT, MulArray3(.4, White));
glMaterialfv(GL_BACK, GL_DIFFUSE, MulArray3(1., White));
glMaterialfv(GL_BACK, GL_SPECULAR, Array3(0., 0., 0.));
glMaterialf(GL_BACK, GL_SHININESS, 0.);
glMaterialfv(GL_BACK, GL_EMISSION, Array3(0., 0., 0.));

// Front
glMaterialfv(GL_FRONT, GL_AMBIENT, MulArray3(1., rgb));
glMaterialfv(GL_FRONT, GL_DIFFUSE, MulArray3(1., rgb));
glMaterialfv(GL_FRONT, GL_SPECULAR, MulArray3(.7, White));
glMaterialf(GL_FRONT, GL_SHININESS, 8.);
glMaterialfv(GL_FRONT, GL_EMISSION, Array3(0., 0., 0.));
```

**Setting the Light Characteristics**

```c
// Light Model
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, MulArray3(.2, White));
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

// Light 0
glLightfv(GL_LIGHT0, GL_AMBIENT, Array3(0., 0., 0.));
glLightfv(GL_LIGHT0, GL_DIFFUSE, LightColor);
glLightfv(GL_LIGHT0, GL_SPECULAR, LightColor);
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.);
```

**Setting the Light Position**

```c
// Translate the object into the viewing volume:
gluLookAt(XEYE, YEYE, ZEYE, 0., 0., 0., 0., 1., 0.);
```

**Sidebar: Why are Light Positions 4-element arrays where the 4th element is 1.0? Homogeneous Coordinates!**

The light position gets transformed by the ModelView matrix at the moment the `glLightfv(…, GL_POSITION, …)` function is called...

```c
float *Array3(float a, float b, float c)
{
    static float array[4];
    array[0] = a;
    array[1] = b;
    array[2] = c;
    array[3] = 1.;
    return array;
}
```

**Homogeneous Coordinates let us Represent Points at Infinity**

This is useful to be able specify a parallel light source by placing the light source position at infinity.

The point (1,2,3,1) represents the 3D point (1,2,3)
The point (1,2,3,5) represents the 3D point (2,4,6)
The point (1,2,3,01) represents the point (100,200,300)

So, (1,2,3,0) represents a point at infinity, along the ray from the origin through (1,2,3).

Points-at-infinity are used for parallel light sources and some shadow algorithms.
Additional Parameters for Spotlights

- \texttt{glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, Array3(xdir, ydir, zdir))}
  Specifies the spotlight-pointing direction. This gets transformed by the current value of the ModelView matrix.

- \texttt{glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, e)}
  Specifies the spotlight directional intensity. This acts very much like the exponent in the specular lighting equation.

- \texttt{glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, deg)}
  Specifies the spotlight maximum spread angle.

A Shortcut I Like

```cpp
void SetMaterial(float r, float g, float b, float shininess)
{
  glMaterialfv(GL_BACK, GL_EMISSION, Array3(0., 0., 0.));
  glMaterialfv(GL_BACK, GL_AMBIENT, MulArray3(.4f, White));
  glMaterialfv(GL_BACK, GL_DIFFUSE, MulArray3(1., White));
  glMaterialfv(GL_BACK, GL_SPECULAR, Array3(0., 0., 0.));
  glMaterialf(GL_BACK, GL_SHININESS, 2.f);
  glMaterialfv(GL_FRONT, GL_EMISSION, Array3(0., 0., 0.));
  glMaterialfv(GL_FRONT, GL_AMBIENT, Array3(r, g, b));
  glMaterialfv(GL_FRONT, GL_DIFFUSE, Array3(r, g, b));
  glMaterialfv(GL_FRONT, GL_SPECULAR, MulArray3(.8f, White));
  glMaterialf(GL_FRONT, GL_SHININESS, shininess);
}
```

Additional Parameters for Spotlights

- \texttt{glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, e)}
  Specifies the spotlight directional intensity. This acts very much like the exponent in the specular lighting equation.

- \texttt{glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, deg)}
  Specifies the spotlight maximum spread angle.

How Does OpenGL Define GL_FRONT and GL_BACK?

- GL_FRONT
  Vertices are CCW when viewed from the outside.

- GL_BACK
  Vertices are CW when viewed from the outside.

Shortcuts I Like

```cpp
void SetPointLight(int ilight, float x, float y, float z, float r, float g, float b)
{
  glLightfv(ilight, GL_POSITION, Array3(x, y, z));
  glLightfv(ilight, GL_AMBIENT, Array3(0., 0., 0.));
  glLightfv(ilight, GL_DIFFUSE, Array3(r, g, b));
  glLightfv(ilight, GL_SPECULAR, Array3(r, g, b));
  glLightf(ilight, GL_CONSTANT_ATTENUATION, 1.);
  glLightf(ilight, GL_LINEAR_ATTENUATION, 0.);
  glLightf(ilight, GL_QUADRATIC_ATTENUATION, 0.);
  glEnable(ilight);
}
```

```cpp
void SetSpotLight(int ilight, float x, float y, float z, float xdir, float ydir, float zdir, float r, float g, float b)
{
  glLightfv(ilight, GL_POSITION, Array3(x, y, z));
  glLightfv(ilight, GL_SPOT_DIRECTION, Array3(xdir, ydir, zdir));
  glLightf(ilight, GL_SPOT_EXPONENT, 1.);
  glLightf(ilight, GL_SPOT_CUTOFF, 45.);
  glLightfv(ilight, GL_AMBIENT, Array3(0., 0., 0.));
  glLightfv(ilight, GL_DIFFUSE, Array3(r, g, b));
  glLightfv(ilight, GL_SPECULAR, Array3(r, g, b));
  glLightf(ilight, GL_CONSTANT_ATTENUATION, 1.);
  glLightf(ilight, GL_LINEAR_ATTENUATION, 0.);
  glLightf(ilight, GL_QUADRATIC_ATTENUATION, 0.);
  glEnable(ilight);
}
```

Sidebar: Note that we are computing the light intensity at each vertex first, and then interpolating that intensity across the polygon second. That is, you are only using the lighting model at each vertex. You can do an even better job if you interpolate the normal across the polygon first, and then compute the light intensity with the lighting model at each fragment second.

But, for that, you will need the Shaders course (CS 457/557)
gMatrixMode( GL_MODELVIEW );
gTranslatef( tx, ty, tz );
gRotatef( degrees, ax, ay, az );
gScalef( sx, sy, sz );
gShadeModel( GL_SMOOTH );
gBegin(GL_TRIANGLES );
gColor3f( r0, g0, b0 );
gVertex3f( x0, y0, z0 );
gColor3f( r1, g1, b1 );
gVertex3f( x1, y1, z1 );
gColor3f( r2, g2, b2 );
gEnd( );

Sidebar: Smooth Shading can also interpolate vertex colors, not just the results of the lighting model

This is especially useful when using colors for scientific visualization:

Smooth Shading can also interpolate vertex colors, not just the results of the lighting model

Tricky Lighting Situations

Notice the lighting in the fur!

Tricky Lighting Situations

Beware of Mach Banding

Our vision systems can’t handle abrupt changes in intensity.

Beware of Mach Banding

Notice how these vertical stripes look “scalloped”, like a Greek column. But, they are solid-color stripes. What is going on?
In fact, our vision systems can't even handle abrupt changes in the slope of intensity.

Think of the Mach Banding problem as being similar to trying to round second base at a 90° angle.