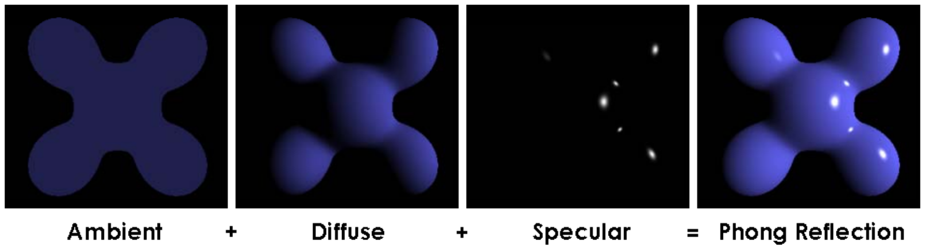
Computer Graphics: Illumination II

Part 2 – Lecture 5

Today's Outline

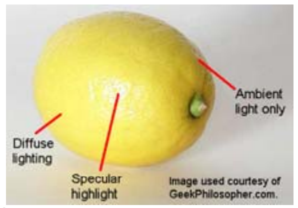
- Recap: Phong Illumination Model
- Lights in OpenGL
- Materials in OpenGL



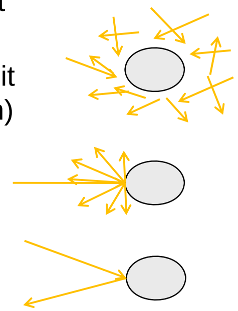
PHONG ILLUMINATION MODEL

Image thanks to Brad Smith

Types of Light Reflection



- In the real world:
  - Light reflected unlimited number of times
  - Reflections change the appearance of the light
- In CG we need to keep computation time short:
  - Can often calculate only one reflection per vertex
  - Consider different light appearances as different types of reflection
- **Ambient reflection**: light reflected so many times, it is everywhere (like uniform background illumination)
- **Diffuse reflection**: light scattered from one point equally (more or less) into all directions
- **Specular reflection**: light rays bounce off in pretty much only one direction (like from a mirror)
- Type of reflection can depend on light source characteristics and the material of the object



Ambient Reflection

We construct an equation for $\mathbf{R_a}$:

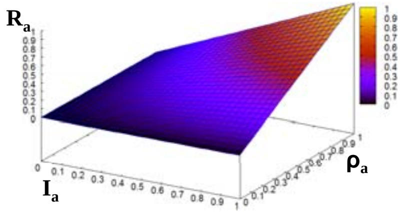
$$\mathbf{R_a} = \mathbf{I_a}\ \rho_a$$

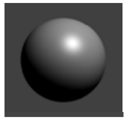
Variable	Influence on $\mathbf{R_a}$
$\mathbf{I_a}$	Proportional
$\boldsymbol{\rho_a}$	Proportional
d	No influence
v	No influence

How to deal with colors (RGB)?

- Instead of just $\mathbf{I_a}$, use $\mathbf{I_{ar}}$, $\mathbf{I_{ag}}$, $\mathbf{I_{ab}}$ $\rightarrow$ colored light
- Instead of just $\boldsymbol{\rho_a}$, use $\boldsymbol{\rho_{ar}}, \boldsymbol{\rho_{ag}}, \boldsymbol{\rho_{ab}}$ $\rightarrow$ colored materials
- Compute reflected light for each color:

$$\mathbf{R_{ar}} = \mathbf{I_{ar}}\ \rho_{ar}$$
$$\mathbf{R_{ag}} = \mathbf{I_{ag}}\ \rho_{ag}$$
$$\mathbf{R_{ab}} = \mathbf{I_{ab}}\ \rho_{ab}$$




No ambient light


A lot of ambient light

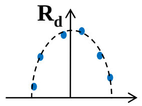
Diffuse Reflection

We construct an equation for $\mathbf{R_d}$:

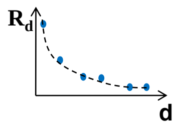
$$\mathbf{R_d} = \mathbf{I_d}\ \rho_d \frac{s \cdot m}{|s||m|} \,/\, (k_c + k_l d + k_q d^2)$$

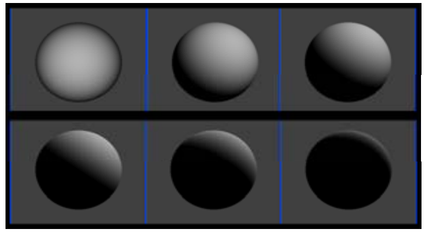
Variable	Influence on $\mathbf{R_d}$
$\mathbf{I_a}$	Proportional
$\boldsymbol{\rho_d}$	Proportional
s	Lambert's law
d	Divide by ($k_c + k_l d + k_q d^2$)
v	No influence

- Add color by calculating $\mathbf{R_{dr}}$, $\mathbf{R_{dg}}$, $\mathbf{R_{db}}$ using $\mathbf{I_{dr}}$, $\mathbf{I_{dg}}$, $\mathbf{I_{db}}$ and $\boldsymbol{\rho_{dr}}, \boldsymbol{\rho_{dg}}, \boldsymbol{\rho_{db}}$ instead of just $\mathbf{R_d}$, $\mathbf{I_d}$ and $\boldsymbol{\rho_d}$



Angle between **s** and **m**




Lambertian spheres (diffuse reflectors)

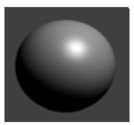
Specular Reflection

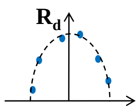
We construct an equation for $\mathbf{R_d}$:
(assuming we have calculated **r** from **s** and **m**)

$$\mathbf{R_s} = \mathbf{I_s}\ \rho_s \left(\frac{v \cdot r}{|v||r|}\right)^{\alpha} \,/\, (k_c + k_l d + k_q d^2)$$

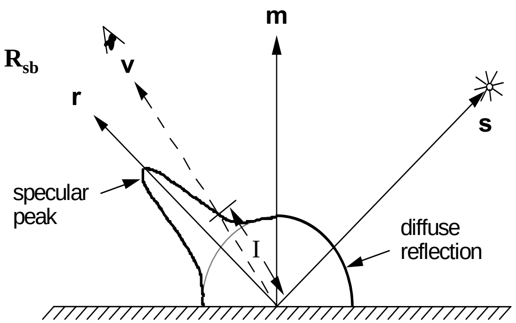
Variable	Influence on $\mathbf{R_s}$
$\mathbf{I_s}$	Proportional
$\boldsymbol{\rho_s}$	Proportional
r and v	Highlight intensity
α	Highlight size
d	Divide by ($k_c + k_l d + k_q d^2$)

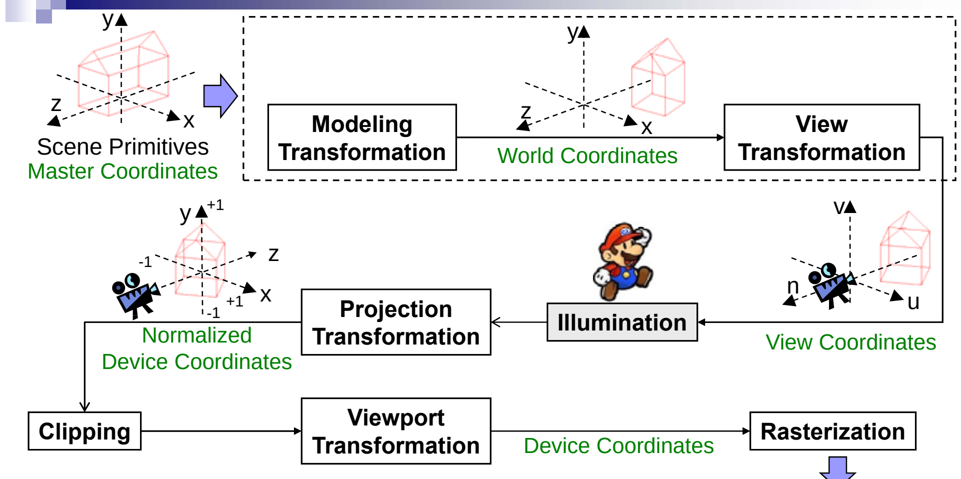
- Add color by calculating $\mathbf{R_{sr}}$, $\mathbf{R_{sg}}$, $\mathbf{R_{sb}}$ using $\mathbf{I_{sr}}$, $\mathbf{I_{sg}}$, $\mathbf{I_{sb}}$ and $\boldsymbol{\rho_{sr}}, \boldsymbol{\rho_{sg}}, \boldsymbol{\rho_{sb}}$ instead of just $\mathbf{R_s}$, $\mathbf{I_s}$ and $\boldsymbol{\rho_s}$



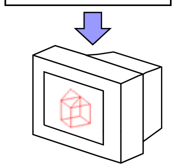


Angle **φ** between **v** and **r**





LIGHTS IN OPENGL



Setting Up Lights

```
float lightPos0[] = {-1.0, 2.0, 3.0, 1.0}; // point source
glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);

float lightPos1[] = {0.0, 1.0, 2.0, 0.0};  // directional
glLightfv(GL_LIGHT1, GL_POSITION, lightPos1);

glEnable(GL_LIGHTING);    // enable lighting in general
glEnable(GL_LIGHT0);      // enable light number 0
glEnable(GL_LIGHT1);      // enable light number 1
```

For setting the properties of lights, use one of

`glLightfv(GLenum light, GLenum pname, float* params)`

`glLightf(GLenum light, GLenum pname, float param)`

- `light` selects a light `GL_LIGHT`i with 0 < i < `GL_MAX_LIGHTS` (8)
- pname selects a property to set (e.g. `GL_POSITION`)

- For point sources: set position to (x, y, z, 1)
- For directional light sources: set position to (x, y, z, 0)
  (x,y,z) points towards the light source

Intensities and Attenuation

```
float l0_ambient[] = {0.2, 0.2, 0.2, 1.0};
float l0_diffuse[] = {0.8, 0.7, 0.7, 1.0};
float l0_specular[] = {1.0, 1.0, 1.0, 1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, l0_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, l0_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, l0_specular);

glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

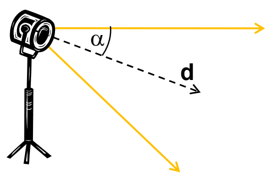
- Set light intensities as RGBA: A (alpha) for color blending, is usually 1
- Attenuation: how intensity decreases with distance from light source
  - Default: k_c=1, k_l=0, k_q=0 (does not decrease with distance)
  - Change for more realism (but slower rendering)

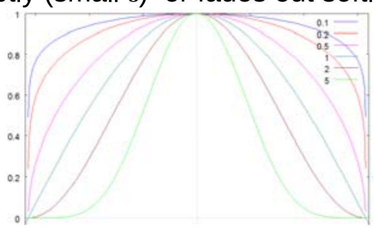
$$\mathbf{R} = \mathbf{I_a}\,\rho_a + (\mathbf{I_d}\,\rho_d \frac{s\cdot m}{|s||m|} + \mathbf{I_s}\,\rho_s \left(\frac{h\cdot m}{|h||m|}\right)^{\alpha}) / (k_c + k_l d + k_q d^2)$$

Spotlights

```
float d[] = {0.0, 0.0, -1.0}; // spotlight direction
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, d);

glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);  // α=45°
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 4.0); // ε=4.0
```

- Position of spotlight set up just as for point sources
- Spotlight points in direction **d**
- Cutoff angle α determines size of spotlight
- Exponent ε determines attenuation towards the borders,
  i.e. if light is cut off abruptly (small ε) or fades out softly (large ε)





Attenuation of

$(\cos(\beta))^{\varepsilon}$

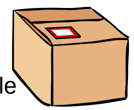
β = angle between light ray and d

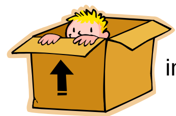
Lighting Model Parameters

- Global ambient light:
  `float global_amb [] = {0.1, 0.1, 0.1, 1.0};`
  `glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_amb);`
- Force view direction vector **v** to be (0, 0, 1) (in view coords.) for all vertices:
  `glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);`

  Why? → less calculations for directional light
  With **s** and **v** constant, **h**=normalized(**s**+**v**) is constant as well
- Switch on lighting of back-facing polygons using reversed surface normal (make inside of objects visible):
  `glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);`

Closed object: only outside needs to be visible



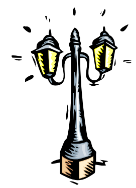


Open object: inside needs to be visible as well

More About Setting Up Lights

- In `init()`: set up all light properties that do not change, e.g. intensities, spotlight α and ε
- In `display()`: set up properties that change during rendering, e.g. light position, direction
- Treat lights like objects: use MODELVIEW matrix, push and pop to set up light position
- A light illuminates only the primitives drawn after the light is enabled
  - Example:
    enable light 0, draw object A, enable light 1, draw object B
    Result:
    A illuminated only with light 0, but B illuminated with lights 0 and 1
  - Can also disable lights with `glDisable()`
- Don't forget: need to enable each light as well as lighting in general

Positioning Lights

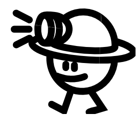


Light position independent of viewer

1. Set up view matrix (e.g. `gluLookAt(…)`)
2. Position light
3. Draw scene

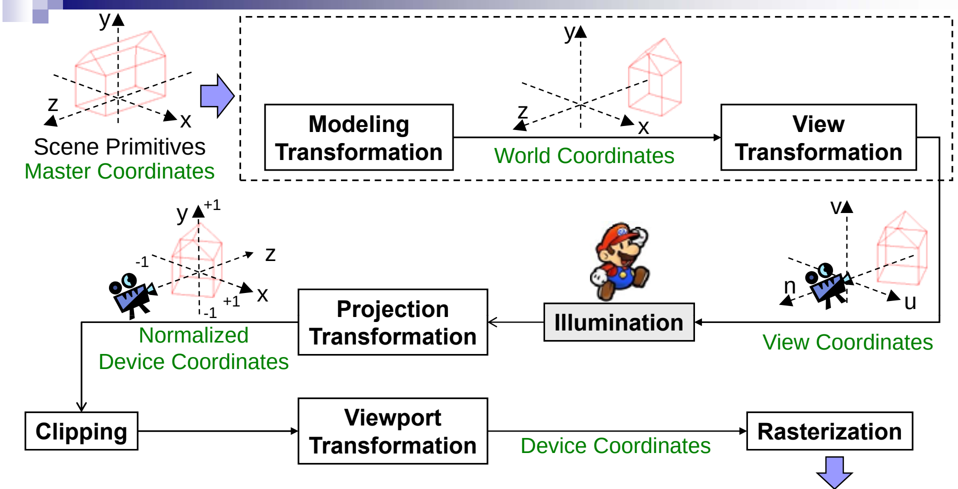
→ stationary lights (e.g. lamppost) or lights moving with scene objects (e.g. car driving by)

Light position relative to viewer

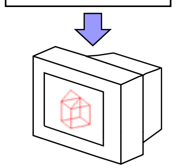


1. Set MODELVIEW matrix to identity
2. Position light relative to viewer (at origin)
3. Set view matrix (e.g. `gluLookAt(…)`)
4. Draw scene

→ lights attached to viewer (e.g. miner's headlamp)



MATERIALS IN OPENGL



Using Materials

```
float ambient[] = {0.1, 0.1, 0.1, 1.0};        // ρar, ρag, ρab, 1
glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
float diffuse[] = {0.4, 0.4, 0.6, 1.0};        // ρdr, ρdg, ρdb, 1
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
float specular[] = {0.8, 0.8, 1.0, 1.0};       // ρsr, ρsg, ρsb, 1
glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
glMaterialf(GL_FRONT, GL_SHININESS, 40.0);     // α=40
```

Set the current material, then draw primitives (they will use the material)

`glMaterialfv(GLenum face, GLenum pname, float* params)`

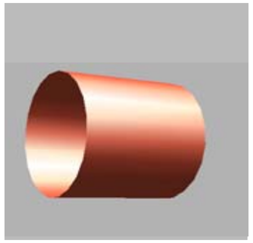
`glMaterialf(GLenum face, GLenum pname, float param)`

- `face` selects side to use material on (`GL_FRONT`, `GL_BACK` or `GL_FRONT_AND_BACK`)
- `pname` selects a property to set (e.g. `GL_AMBIENT`, `GL_EMISSION`, `GL_AMBIENT_AND_DIFFUSE`, `GL_SHININESS`, …)

- Set coefficients as RGBA: A (alpha) for color blending, is usually 1

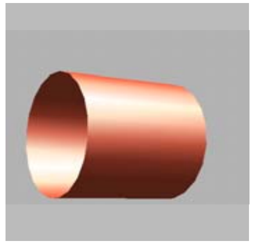
Example: Shaded Cylinder 1

```
const float LIGHT_POS[] = {50, 100, 30,1};
const float LIGHT_AMB[] = {0,0,0,1};
const float LIGHT_COL[] = {1,1,1,1};
const float GLOBAL_AMB[] = {0.4, 0.4, 0.4, 1};

CTrackball trackball;

void init(void) {
  glClearColor (0.7f, 0.7f, 0.7f, 1);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(30, 1.0, 1.0, 20.0);
  trackball.tbInit(GLUT_LEFT_BUTTON);
  glEnable(GL_DEPTH_TEST);
  glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, 1);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT,
    GLOBAL_AMB);
  glLightfv(GL_LIGHT0, GL_POSITION, LIGHT_POS);
  glLightfv(GL_LIGHT0, GL_AMBIENT, LIGHT_AMB);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, LIGHT_COL);
  glLightfv(GL_LIGHT0, GL_SPECULAR, LIGHT_COL);
  glEnable(GL_LIGHT0);
  glEnable(GL_LIGHTING);
}
```



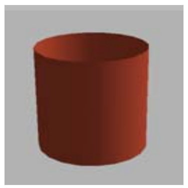
Example: Shaded Cylinder 2

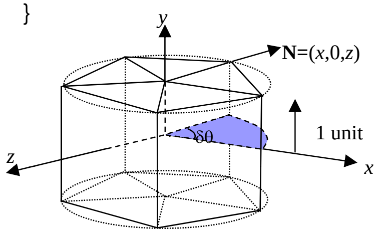
```
const float CYLINDER_COL[] = {0.8, 0.3, 0.2, 1};
const float CYLINDER_SPEC[] = {1,1,1,1};
const float CYLINDER_SHININESS = 10;

void display(void) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(0,0,10, 0,0,0, 0,1,0);
  trackball.tbMatrix();
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, CYLINDER_COL);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, CYLINDER_SPEC);
  glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, CYLINDER_SHININESS);

  drawCylinder();
}
```



Example: Shaded Cylinder 3

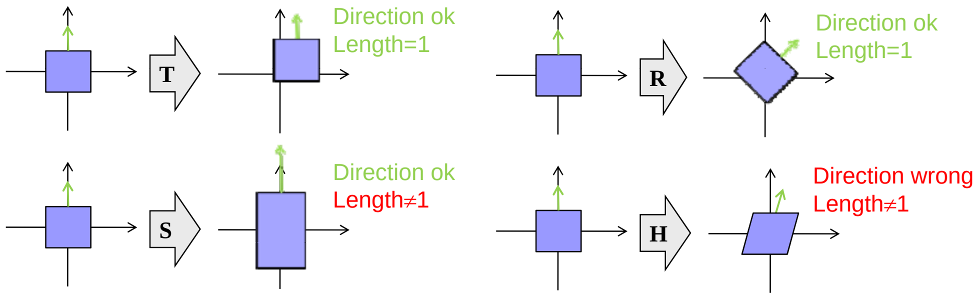
```
const int NUM_STRIPS = 10;
const float PI = 3.1415926;
const float DELTA_THETA =
    (float) (2 * PI / NUM_STRIPS);

void drawCylinder() {
  for (int strip = 0; strip < NUM_STRIPS; strip++) {
    float theta = strip * DELTA_THETA;
    float thetaNext = theta + DELTA_THETA;
    float x = (float) cos(theta);
    float z = -(float) sin(theta);
    float xNext = (float) cos(thetaNext);
    float zNext = -(float) sin(thetaNext);
    glBegin(GL_POLYGON);
    glNormal3f(x, 0, z);
    glVertex3f(x, 1, z);
    glVertex3f(x, -1, z);
    glNormal3f(xNext, 0, zNext);
    glVertex3f(xNext, -1, zNext);
    glVertex3f(xNext, 1, zNext);
    glEnd();
  }
  glFlush();
  glutSwapBuffers();
}
```





Surface Normals

- Before each call to `glVertex3f(x,y,z)` the normalized surface normal has to be set with `glNormal3f(x,y,z)`
- Problem: when transforming vertices with the MODELVIEW matrix, the surface normal needs to be adjusted, e.g.



- OpenGL adjusts normal direction for you ☺
- But OpenGL does not renormalise the transformed normal unless you call `glEnable(GL_NORMALIZE)` (→ slower)

Transformation of Surface Normals

How to adjust surface normal **n** after arbitrary transformation **M** ?

Answer: adjust by transforming with $\mathbf{Q} = (\mathbf{M}^{-1})^T = (\mathbf{M}^T)^{-1} = \mathbf{M}^{-T}$

Proof: let $\mathbf{p}_1$ and $\mathbf{p}_2$ be two points on a polygon with normal **n**

1. $\mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1) = 0 \Leftrightarrow \mathbf{n}^T(\mathbf{p}_2 - \mathbf{p}_1) = 0$ (**n** perpendicular to polygon)
2. This has also to be true after transforming $\mathbf{p}_1$ and $\mathbf{p}_2$ by **M** and **n** by **Q,** i.e. $(\mathbf{Qn})^T(\mathbf{M}(\mathbf{p}_2 - \mathbf{p}_1)) = 0$
3. Apply rule from matrix algebra: $(\mathbf{Qn})^T = \mathbf{n}^T\mathbf{Q}^T$: $\mathbf{n}^T\mathbf{Q}^T\mathbf{M}(\mathbf{p}_2 - \mathbf{p}_1) = 0$ whenever $\mathbf{n}^T(\mathbf{p}_2 - \mathbf{p}_1) = 0$
4. Solution is $\mathbf{Q}^T\mathbf{M} = \mathbf{I} \Rightarrow \mathbf{Q}^T = \mathbf{M}^{-1} \Rightarrow \mathbf{Q} = (\mathbf{M}^{-1})^T = \mathbf{M}^{-T}$

Note: the adjusted normal is not always normalized

SUMMARY

Summary

1. Set up properties of OpenGL lights with `glLightf[v](…)`
   - Directional lights, point lights and spotlights (cutoff angle α and ε)
   - Ambient, diffuse and specular light intensity (RGBA)
2. Set the current material with `glMaterialf[v](…)`
   - All subsequently drawn primitives will use the material
   - Ambient, diffuse and specular reflection coefficients (RGBA)
3. Make sure to set up and maintain surface normals correctly

References:

- OpenGL Lights: Hill, Chapter 8.2.8
- OpenGL Materials: Hill, Chapter 8.2.9
- OpenGL API Reference: http://www.cs.auckland.ac.nz/compsci372s1c/resources/manpagesOpenGL

Quiz

1. How would you change the ShadedCylinder program to…
   1. Make the highlight band broader?
   2. Make the highlight band brighter?
   3. Change the colour of the highlight band?
   4. Remove the highlight altogether?

2. In ShadedCylinder, does the mouse rotate the scene and the light together, or just the scene? Whichever it does, make it do the other.

3. Describe two transformations after which the surface normals need adjustment.