Ch 6: Colors and Lighting

OpenGL Light

Three steps to add a basic lighting sources:

1. Enable lighting
2. Define the position of the light source
3. Set materials (colors)

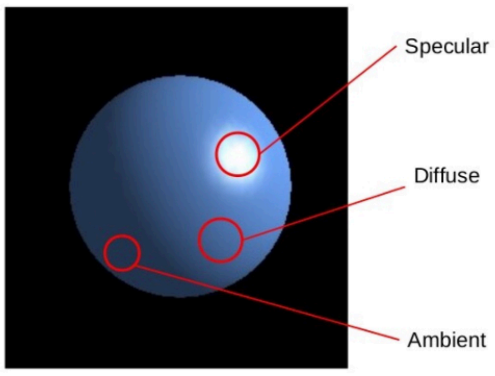
1. Enable lighting
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

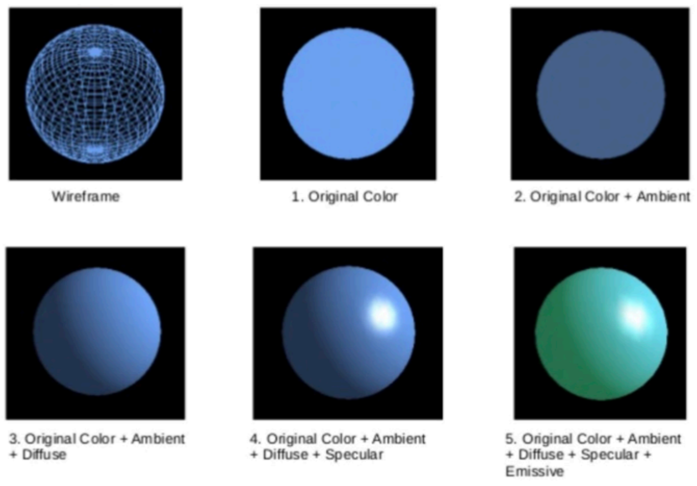
2. Set lighting position
glLight — set light source parameters (glLightfv)
glLightfv(**GL_LIGHT0**, **GL_POSITION**, light_postion);

3. Set materials
glEable(**GL_COLOR_MATERIAL**);

Different Types of Light

- Specular: sets of colors for **highlight**
- Diffuse: the color of the mesh when it is **illuminated**
- Ambient: the color of the mesh when it is **not illuminated**
- Emissive: the type of light which is being emitted by the object




Wireframe
1. Original Color
2. Original Color + Ambient
3. Original Color + Ambient + Diffuse
4. Original Color + Ambient + Diffuse + Specular
5. Original Color + Ambient + Diffuse + Specular + Emissive

Material Colors

When the lighting is enabled, the color is not determined by glColor, but by the **material colors (by default is a diffuse gray material)** combined with the **light colors** using the lighting computations.

• Without glEnable(GL_COLOR_MATERIAL), glColor is no longer being used.

Shade Model
glShadeModel: specify the shading technique, GL_FLAT and GL_SMOOTH
• **By default: GL_SMOOTH**

Ch7: OpenGL Programing Tools

Pixel Art
Two kinds of pixels: cube pixels and sphere pixels. Use many of these pixels with different positions, colors and sizes to simulate the appearance of items.
Translation:

```
void oneSphere(GLfloat x, GLfloat y, GLfloat z, GLfloat R){
    glPushMatrix();
    glTranslated(x, y, z);
    glutSolidSphere(R, 50, 50);
    glPopMatrix();
}
```

Modular Programming
Modular Programming: breaking a program up into smaller, manageable functions or modules
– improve **maintainability** of programs
– Make **writing** programs easier
– Enable the **reusability** of function modules

Global Data
Use Variables: the values can be easily passed on between functions

Ch8: OpenGL Interactive Events

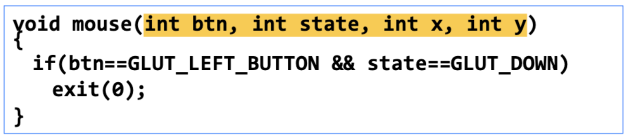
GLUT Callbacks
- glutMouseFunc
- glutMotionFunc
- glutKeyboardFunc
- glutSpecialFunc

Mouse Callback
glutMouseFunc(mymouse);
void mymouse(GLint **button**, GLint **state**, GLint **x**, GLint **y**) { .. }
Button:
- GLUT_LEFT_BUTTON
- GLUT_MIDDLE_BUTTON
- GLUT_RIGHT_BUTTON

State:
- GLUT_UP (release the mouse)
- GLUT_DOWN (press the mouse)

Position in window (x, y)

Terminating a Program
exit(0); #include <cstdlib>



```
void mouse(int btn, int state, int x, int y)
{
  if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
    exit(0);
}
```

Positioning in Screen
The position in the screen is measured in pixels with origin at the **top-left corner**.
However, OpenGL uses a world coordinate system with origin at the **bottom left**
- Must invert y coordinate returned by callback by height of window **(y = h – y)**

Motion Callback
glutMotionFunc(myMouseMotion);
void myMouseMotion(int x, int y);

Keyboard Callback
glutKeyboardFunc(mykey);
void mykey(**unsigned** char key, int x, int y)
delete, escape, backspace — ASCII
e.g., useful for the player to know the position of mouse while using the keyboard to control the game

Unsigned Char v.s. Signed Char
Signed char range: –128 ~ 127
Unsigned char range: 0 ~ 255

Special keyboard Callback
glutSpecialFunc(mySpecialKey): special keys (F1 — F10, 上下左右 …)
void mySpecialKey(int key, int x, int y) {
}

Modifier Keys
– GLUT_ACTIVE_SHIFT
– GLUT_ACTIVE_CTRL
– GLUT_ACTIVE_ALT
Pressed by glutGetModifiers()

```
void mySpecialKey(int key, int x, int y) {
    if ((key == GLUT_KEY_LEFT) && (glutGetModifiers() == GLUT_ACTIVE_SHIFT))
            exit(0);}
```

Menus
1. Define entries for menu (glutCreateMenu( ))
2. Define action for each menu item (glutAddMenuEntry( ), glutAddSubMenu( ))
3. Attach menu to a mouse button (glutAttachMenu( ))

```
GLint menu_id = glutCreateMenu(mymenu);
glutAddMenuEntry("clear screen", 1);
glutAddMenuEntry("exit", 2);
glutAttachMenu(GLUT_LEFT_BUTTON);
```

```
void mymenu(int id)
{
 if(id == 1) ...//Do something something;
 if(id == 2) ...//Do something something;
}
```

Question:

d) Consider the code shown in **Code Box 1** on the next page.

[9 marks]

i) Which two statements can be removed from the code without modifying its output?

(2 marks)

ii) Explain the purpose of the following statement:

```
glGetIntegerv (GL_VIEWPORT, viewport);
```

(4 marks)

iii) Modify the code so that a click on the right button of the mouse terminates the program.

(3 marks)

```
#include <GL/glut.h>

GLdouble W = 640.0;
GLdouble H = 480.0;
GLint size = 30;

void myInit(void) {
   glClearColor (0.0, 0.0, 0.0, 1.0);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D (0, W, 0, H);
}

void mydisplay(){
}

void drawSquare(int x, int y)
{
    GLint viewport[4];
    glGetIntegerv (GL_VIEWPORT, viewport);
    y = viewport[3] - y;
    glShadeModel(GL_SMOOTH);
    glBegin(GL_POLYGON);
        glVertex2i(x+size, y+size);
        glVertex2i(x-size, y+size);
        glVertex2i(x-size, y-size);
        glVertex2i(x+size, y-size);
     glEnd();
     glFlush();
}

void mymouse(int btn, int state, int x, int y)
{
   if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
      drawSquare(x, y);
}

int main(int argc, char** argv){
      glutInit(&argc, argv);
      glutInitWindowSize(W, H);
      glutCreateWindow("simple");
      glutDisplayFunc(mydisplay);
      glutMouseFunc(mymouse);
      myInit();
      glutMainLoop();
}
```

(a)

```
glClearColor(0.0, 0.0, 0.0, 1.0);
glShadeModel(GL_SMOOTH); by default, it is GL_SMOOTH
```

(b)

glGetIntegerv is a function that can get the viewport. It returns for values (x, y, w, h). As for viewport[3], it refers to the height of the whole viewport. Since the openGL has origin at the bottom-left, however the screen regards the origin at the top-left corner, we need to reverse the y value by h – y which is viewport[3] – y to replace y. Then we can draw the square at the place we want.

(c)

```
/* this line will be added at the top*/
#include <cstdlib>

/* the following lines are added in the mymouse() function*/
if (btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN) {
        exit(0);
}
```

Ch9: OpenGL Camera and Transformation

Geometric Transformations: Translation, Scaling, Rotation

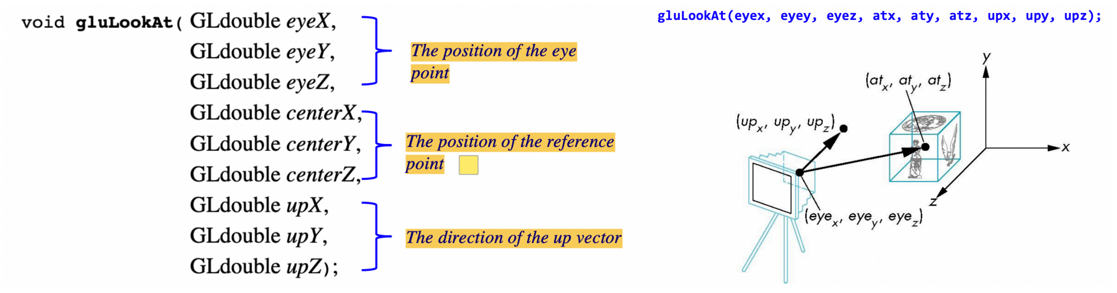
Pushing & Popping Matrix Stacks

- OpenGL maintains a matrix stack for both the model–view and projection matrices
- Save the current transformation state and then restore it after transformation changes

1. Push the current matrix onto the stack with **glPushMatrix** to save it
2. Make matrix to the current matrix
3. Popping the matrix off the stack with **glPopMatrix** then restores it

Camera

gluLookAt: define a viewing transformation



```
void gluLookAt( GLdouble eyeX,
                GLdouble eyeY,
                GLdouble eyeZ,
                GLdouble centerX,
                GLdouble centerY,
                GLdouble centerZ,
                GLdouble upX,
                GLdouble upY,
                GLdouble upZ);
```

The position of the eye point

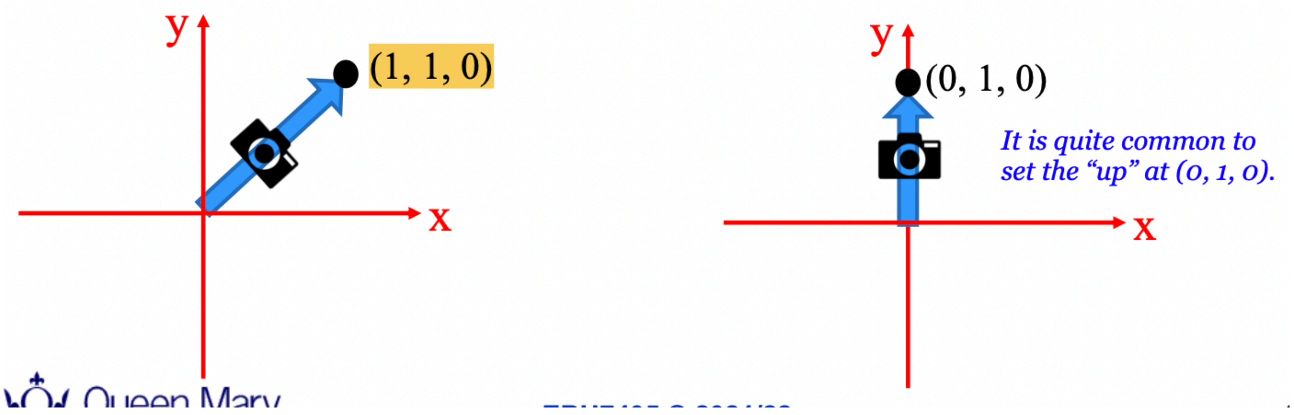
The position of the reference point

The direction of the up vector

```
gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);
```

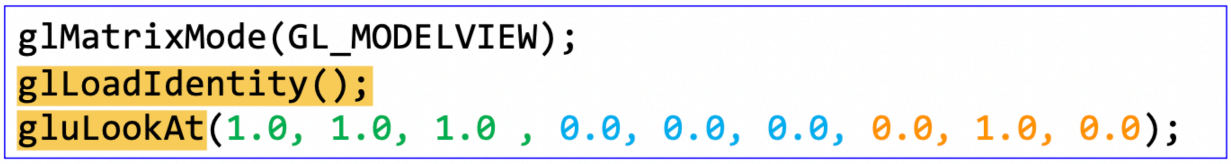
Eye Coordinate:
By default the camera is pointing towards **negative z direction**

Up Coordinate



It is quite common to set the “up” at (0, 1, 0).

Isometric View with gluLookAt



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(1.0, 1.0, 1.0 , 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```

Set matrix in model view, load identity matrix

Translation
glTranslate — multiply the current matrix with the translation matrix

Rotation
glRotate — multiply the current matrix with the rotation matrix

Scaling
glScale — multiply the current matrix with the scaling matrix

Tips for translation and rotation:
- **Multiple translations** can be **put in the same line** at the same time
- **Multiple rotations** must be done **one at a time**

Question:

ii) Which OpenGL function allows you to set the location and orientation of the camera? How does it work?

(3 marks)

(2) gluLookAt() function set the location and orientation of the camera. There are 9 arguments for this function gluLookAt(x_eye, y_eye, z_eye, x_at, y_at, z_at, x_up, y_up, z_up). The first three is the position of the camera, the middle three is the position of the reference point which set the orientation, the last three is the direction of the up vector of the camera.

Question:

a) For each OpenGL function below, state if it is a "primitive generating" function or a "state changing" function. Justify your answers.

[8 marks]

i) glOrtho

(3 marks)

ii) glScale

(2 marks)

iii) glVertex

(3 marks)

(1) glOrtho is a state changing function. The function is used for projection transformation to define the viewing volume and affects the projection matrix, which belongs to the states.
(2) glScale is a state changing function. This function is used for model-view transformation for scaling the objects, which belongs to the states.

(3) glVertex is a primitive generating function. This function is sued to create 2D or 3D points which are the elementary geometric primitives of 3D models

Question:

c) Starting from the object model shown on **Figure 2 a)**, find the single modelling transformation that can create the object instance shown in **Figure 2 b)**. Give the corresponding OpenGL command with appropriate arguments. Justify your choice of transformation and arguments.

[6 marks]

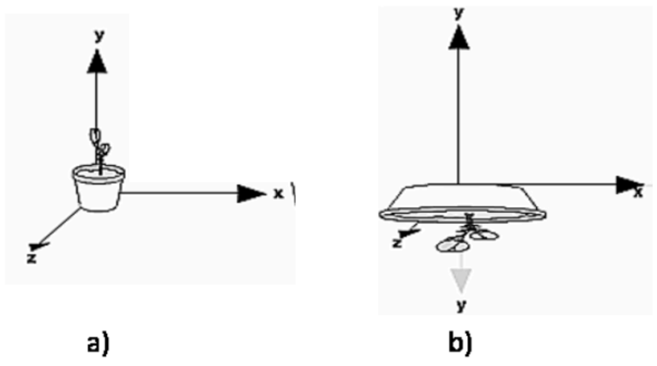


Figure 2.

(c) The transformation is a **non-uniform** scaling transformation.
OpenGL function: glScalef(5.0, –1.0, 1.0);

Question:

c) Write the OpenGL functions that correspond to the three transformations illustrated in **Figure 2**. Estimate their parameters.

[6 marks]

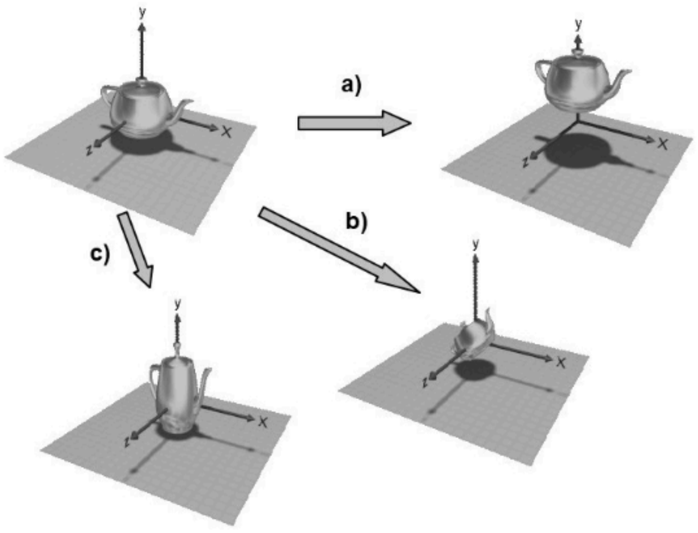


Figure 2.

(c)

For a) it goes through a translation transformation: glTranslatef(0.0, 5.0, 0.0);
For b) it goes through a rotation transformation: glRotatef(45.0, 0.0, 0.0, 1.0);
For c) it goes through a scaling transformation: glScalef(0.5, 2.0, 0.5);