

浙江大学实验报告

课程名称：____计算机图形学____ 指导老师：____成绩：____

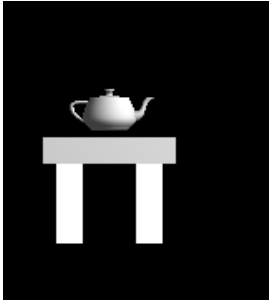
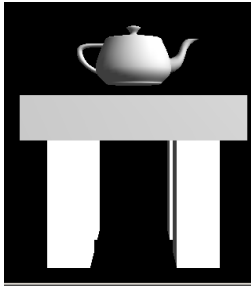
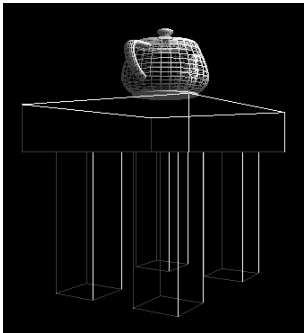
实验名称：____OpenGL 三维观察____ 实验类型：____基础实验____ 同组学生姓名：____

一、实验目的和要求

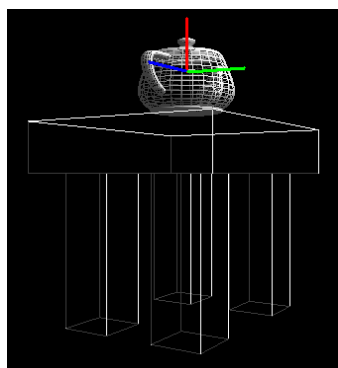
在模型变换实验的基础上，通过实现下述实验内容，掌握 OpenGL 中三维观察、透视投影、正交投影的参数设置，并能使用键盘移动观察相机，在透视投影和正交投影间切换，验证课程中三维观察的内容；进一步加深对 OpenGL 三维坐标和矩阵变换的理解和应用。

二、实验内容和原理

使用 Visual Studio C++编译已有项目工程，并修改代码生成以下图形：

 <p>正投影</p>	 <p>透视投影</p>
	使用键盘改变 camera 位置与观察方向 (按键为 W、S、A、D、Z、C，也可以自行设定)

添加键盘对茶壶的控制，主要是茶壶沿着桌面的平移操作（如下图中绿色和蓝色标示）和茶壶绕自身轴（如下图中红色标示）的旋转操作；按键为：l, j, I, k, e。具体对应关系可查看参考答案中的操作指南。



三、主要仪器设备

Visual Studio C++

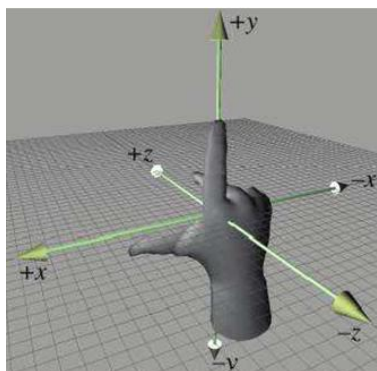
glut.zip

Ex3-vs2010 工程

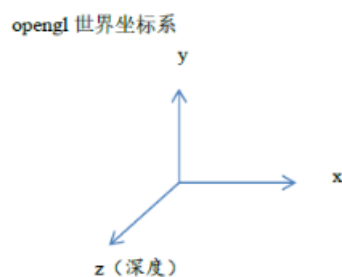
四、实验原理

1. OpenGL 世界坐标系

世界坐标系始终是固定不变的。OpenGL 使用右手坐标，这里有一个方法为：使用右手定则，比如如果把拇指指向右边，食指指向天空，那么中指将指向我们的背后，我们观察的方向是 Z 轴负半轴的方向，右手定则及 opengl 世界坐标系如图所示：



右手定则

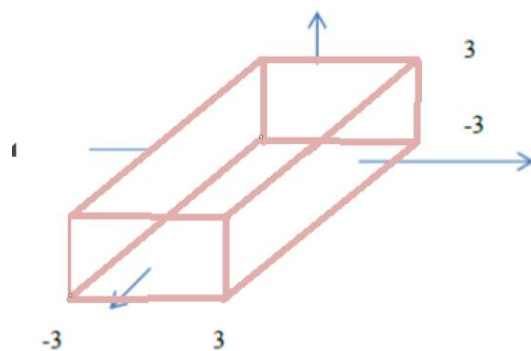


OpenGL 世界坐标系

2. 投影模式

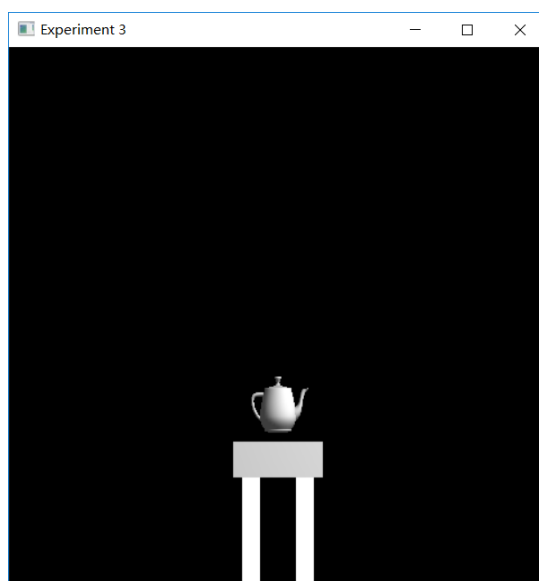
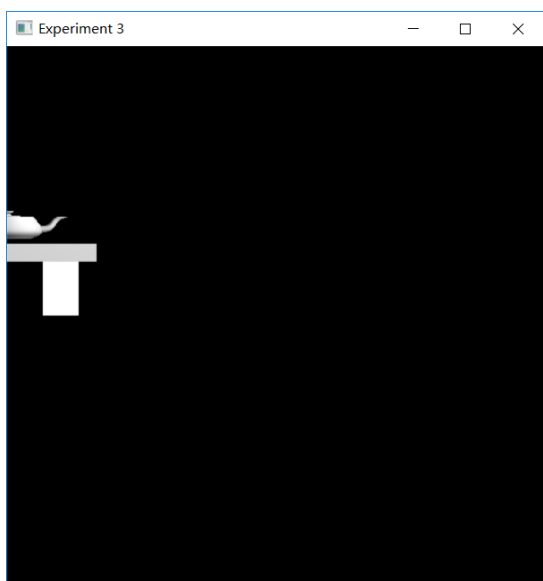
(1) 正投影

所用函数为正射投影函数 `void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)`。六个参数分别为视景体左面，右面，下面，上面，近处，远处的坐标。在本次实验中设置的参数为 `glOrtho(-3, 3, -3, 3, -100, 100)`；这六个参数划分出了一个立方体空间（如下图所示），将桌子和茶壶框在其中，不同于透视投影模式中正大远小的特征，在正投影模式下，物体大小在空间前后移动的过程中不会显现出变化。



六个参数所划分出的立方体空间

如果将参数设置为 `glOrtho(0, 3, -3, 3, -100, 100)` 或者 `glOrtho(-3, 3, 0, 3, -100, 100)` 则桌子与茶壶显示不正常（如图所示）



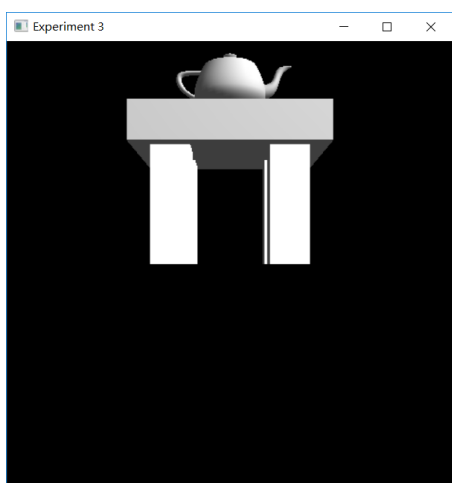
(2) 透视投影

所用函数为 `gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)`。四个参数分别为视角大小，实际窗口的纵横比，近处的截面，远处的截面。在本次实验中我们设置的参数为 `gluPerspective(45, whRatio, 1, 100)`。在透视投影中，物体的显示更加符合人眼的观察，物体在前后移动时，其大小会发生变化，即我们所说的“近大远小”。

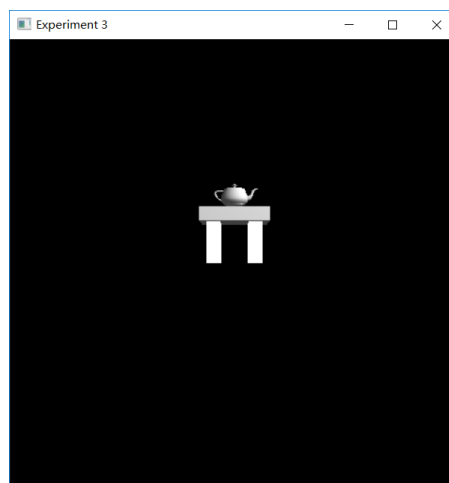
3. 初始观察方位

这里用到的函数为 `void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz)`。其中第一组 `eyex, eyey, eyez` 表示相机在世界坐标的位置，第二组 `centerx, centery, centerz` 表示相机镜头对准的物体在世界坐标的位置，第三组 `upx, upy, upz` 表示相机向上的方向在世界坐标中的方向。在这里我们用到的参数为 `gluLookAt(eye[0], eye[1], eye[2], center[0], center[1], center[2], 0, 1, 0)`；

值得注意的是，其中 `eye[2]` 的数值设定是根据透视投影的大小来调整的，比如当 `eye[2]` 的值为 3 的时候，初始的透视投影会变得很大(如下图所示)，观察起来并不方便，当 `eye[2]` 的值变为 8 时，初始的透视投影与初始的正投影大小相似(如下图所示)，故这里我们设置为 8。



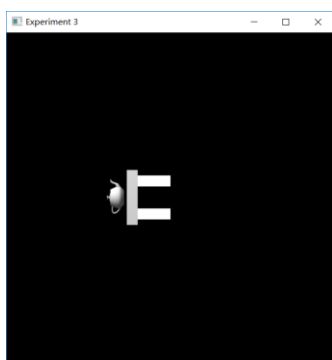
`eye[2]`设置为 3 时



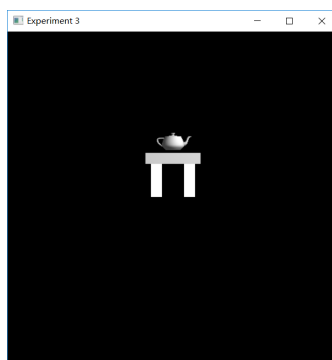
`eye[2]`设置为 8 时

在第三组数据中，我们设置相机向上的方向为 y 轴所在方向，这样可以看到整个桌身和壶身，是最合理的观察方向。如图所示，如果设置相机向上的方向为 x 轴和 z 轴所在方向，看到的分别是斜过

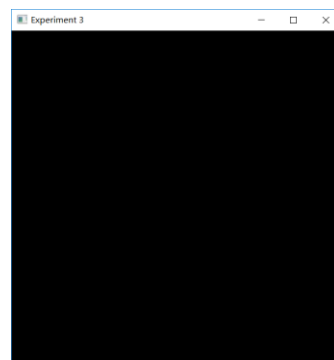
来的茶壶桌子和无茶壶桌子的场景。



相机向上的方向为 x 轴所在方向



相机向上的方向为 y 轴所在方向



相机向上的方向为 z 轴所在方向

4. 茶壶与桌子的移动

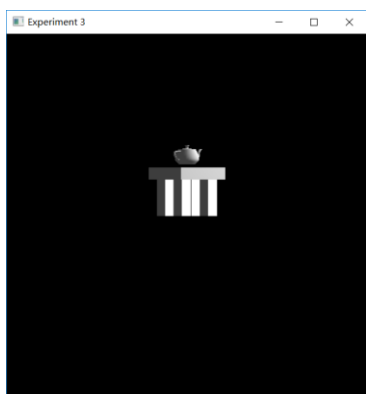
当我们未对物体进行平移旋转等操作时，物体处于原点，局部坐标系与世界坐标系重合，而之后进行的一系列操作是在物体的局部坐标系上进行的，进行矩阵操作后局部坐标系位置将发生变化。

在控制茶壶的移动时，有两种方法，这里以向右移动为例

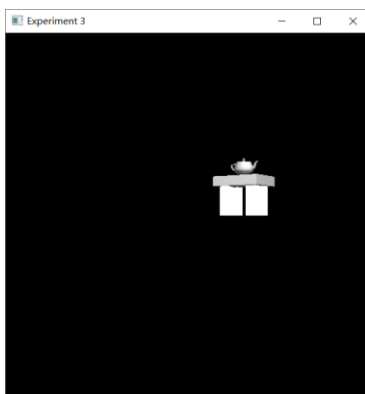
方法一：将相机在世界坐标系中向左移动，即 $\text{eye}[0]=\text{eye}[0]-0.2f$

方法二：将物体在世界坐标系中向右移动，即 $\text{center}[0]=\text{center}[0]+0.2f$

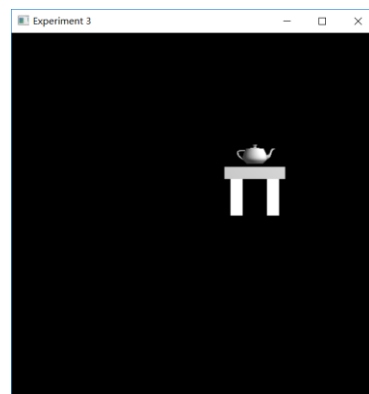
通过实验结果我们可以发现(如下图所示)，单独变化相机位置只会使得茶壶和桌子旋转一定的角度，但仍然处于画面正中央，而单独变化物体的位置时茶壶和桌子在整个视线中右移，但是茶壶和桌子出现了一定角度的旋转，而当我们两个操作结合在一起时，可以发现，桌子和茶壶实现了正常的移动。



单独变化相机的位置



单独变化物体的位置



变化物体和相机的位置

5. 深度测试

在这里我们所调用的函数是 `glEnable(GL_DEPTH_TEST)`

深度即像素点在 3D 世界中距离摄像机的距离，深度缓存中存储着每个像素点的深度值，深度值越大，则离摄像机越远。在不使用深度测试的时候，后绘制的物体会把先绘制的物体覆盖掉，但是有了深度缓冲以后，则可以按照远近顺序进行显示，根据坐标远近自动隐藏被遮住的物体，更加符合现实世界的表现。

6. 设置光源

添加光照效果，更容易表现三维的物体，在这里我们运用的创造光源的函数是 `void glLightfv(GLenum light, GLenum pname, const GLfloat *params)`。第一个参数 `light` 指定所创建的光源号，如 `GL_LIGHT0`、`GL_LIGHT1`、...、`GL_LIGHT7`；第二个参数 `pname` 制定光源特性；第三个参数设置相应的光源特性值。我们所用到的 `glLightfv(GL_LIGHT0, GL_POSITION, light_pos)` 用于设置 0 号光源的位置属性，`glLightfv(GL_LIGHT0, GL_AMBIENT, white)` 用于设置 0 号光源的环境光属性。

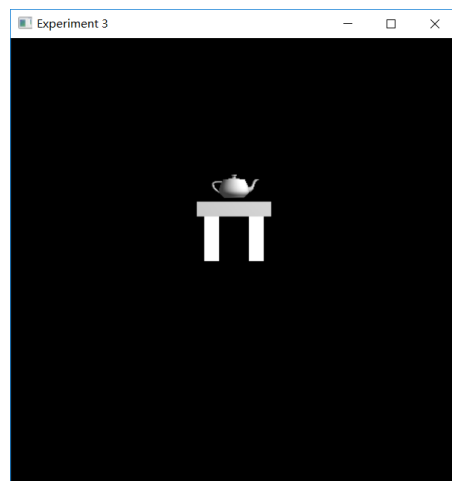
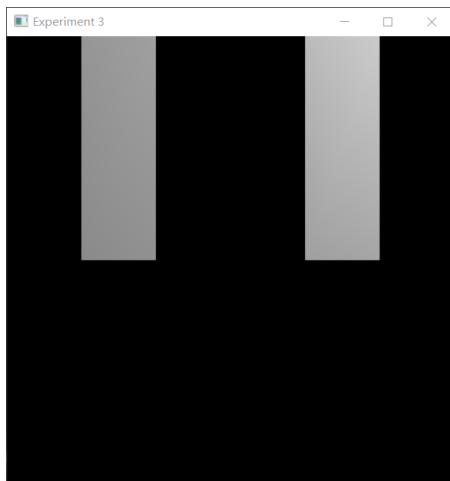
`void glEnable(GLenum cap)` 函数用来启动各种功能，其中 `cap` 是一个参数值，每个参数值有不同的功能，这里我们用 `glEnable(GL_LIGHT0)` 来启动 0 号光源。

五、实验步骤及过程分析

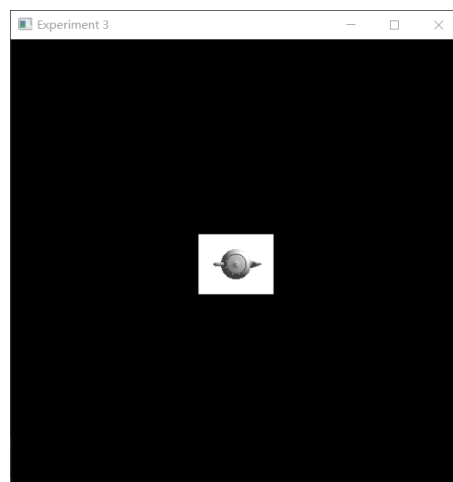
1. 对茶壶和桌子整体的操作

```
1. glRotatef(fRotate, 0, 1.0f, 0); // Rotate around Y axis
2. glRotatef(-90, 1, 0, 0); // Make the table facing the camera
3. glScalef(0.2, 0.2, 0.2); // Scale to make the object appear in the window at the appropriate size
4. Draw_Scene(place); // Draw Scene
5.
6. if (bAnim) fRotate += 0.5f; // Teapot and table rotate together
7. if (tpRotate) tRotate += 0.5f; // Teapot rotate
```

值得注意的是，这里运用了一个 `glScalef` 函数，如果没有这个函数，则桌子和茶壶的整体会非常大，如下图所示，当缩小为原来的 0.2 倍后就成为了合适的大小（如下图所示）



另外，这里用到的 `glRotatef` 函数目的是使视角面对壶身和桌身，如果没有执行此操作，则视角为桌面和壶盖（如图所示），不符合要求。



2. 场景绘制

```
1. void Draw_Scene(float place[])
```

```

2. {
3.     glPushMatrix();
4.     glTranslatef(place[0], place[1], place[2]); // Move the teapot to its location
5.     glRotatef(90, 1, 0, 0); //Put the kettle body facing the camera
6.     glRotatef(tRotate, 0, 1, 0); // Control the rotation of the teapot

7.     glutSolidTeapot(1); // Draw a teapot
8.     glPopMatrix();
9.
10.    // Draw the cube
11.
12.    // Draw four legs
13. }

```

这里的 `glRotatef(tRotate, 0, 1, 0)`，`glRotatef(90, 1, 0, 0)`和 `glTranslatef(place[0], place[1], place[2])`是控制单个茶壶旋转和移动的操作，在绘制茶壶时直接对物体进行平移和旋转。值得注意的是，如果不执行 `glRotatef(90, 1, 0, 0)`操作，在场景中茶壶是“躺”在桌面上的，场景中出现的是壶底，如图所示，这也是执行此项操作的意义。此外，如果想要单独移动茶壶的话，可以通过“i, j, k, l”四键来改变 `place[0]`和 `place[1]`的大小，从而调整茶壶的位置。

3. 键盘回调函数

当按下键盘按键时的一系列操作：WASDZC：控制相机上下左右前后移动；P：切换投影方式（正投影与透视投影）；O：切换渲染方式（填充模式与线框模式）；空格键：启动与暂停旋转（桌子与茶壶一起绕桌子中心轴旋转）；IKJL：控制茶壶前后左右移动；E：茶壶的启动与暂停旋转。相关代码在此不再赘述。

4. 整体操作顺序

首先按顺序依次调用，`glutDisplayFunc(redraw)`用来注册绘制回调函数；`glutReshapeFunc(reshape)`用来注册重绘回调函数，该函数在窗口大小改变以及初始窗口时被调用，在此函数中完成关于坐标系显示的一系列初始化；`glutKeyboardFunc(key)`用来注册按键回调函数，为了能够通过按键对物体进行操作，需要注册按键回调函数，此函数并非在程序启动时执行，而是等待用户完成某些操作后再通过函数指针调用函数；`glutIdleFunc(idle)`用来注册全局回调函数，空闲时调用，而函数`void idle()`中包含了`glutPostRedisplay()`函数，此函数用来标记当前窗口需要重新绘制，通过`glutMainLoop`下一次循环时，窗口显示将被回调以重新显示窗口的正常面板，调用`redraw()`函数。多次调用`glutPostRedisplay`，在下一个显示回调只产生单一的重新显示回调；`glutMainLoop()`则表示`glut`事件处理循环。

```

1. glutDisplayFunc(redraw); // Register a draw callback function that specifies the function to call when the window content needs to be redrawn
2. glutReshapeFunc(reshape); // The callback function when the registration window size changes.
3. glutKeyboardFunc(key); // Register key callback function
4. glutIdleFunc(idle); // Register global callback function : call when idle
5. glutMainLoop(); // Glut event processing loop

```

六、实验结果与心得

1. 实验结果：如所包含的 exe 所示。

2. 实验心得：

(1) 本次实验用到了多次平移、旋转及缩放操作，需要注意的是，完成基本的平移旋转缩放操作需要调 `glTranslatef`, `glRotatef` 和 `glScalef` 三个函数，原理为在当前操作矩阵上乘以一个平移，旋转或缩放矩阵。在绘制时，需要先设置当前矩阵的模式为模型矩阵，并且将矩阵初始化为单位矩阵。需要注意到，相乘顺序与实际情况是相反的：最后乘的操作矩阵事实上最先和图形矩阵相乘。

(2) 在 `redraw()` 函数中和 `Draw_Scene()` 函数中分别用到了 `glRotatef` 函数，用来调整茶壶桌子整体的方向和茶壶自身的方向，这提醒我需要在程序运行后及时发现场景中的问题，对场景进行一定的旋转等操作，使其显示正确的画面。

(3) `eye[2]` 的参数和进行 `redraw()` 函数中的 `glScalef()` 函数对于视角有非常大的影响，如果 `eye[2]` 的参数设置不合适或者未进行缩放操作，会造成场景中的桌子过于大，甚至只能看到一个桌子腿这样的情况发生。

(4) 在进行对桌子和茶壶整体的移动操作时，需要综合考虑相机位置和物体位置两个因素，才能呈现出正确的画面。

七、源代码

```
1. #include <stdlib.h>
2. #include "GL/glut.h"
3.
4.
5. float fTranslate;
6. float fRotate = 0.0f; // Set initial rotation value to 0.0f
7. float fScale = 1.0f; // Set initial scale value to 1.0f
8.
9. bool bPersp = false; // Judging whether it is a perspective projection or a orthographic
    projection
10. bool bAnim = false; // Determine if the teapot and table rotate
11. bool bWire = false; // Determine if the drawing mode is linear or filled
12.
13. int wHeight = 0;
14. int wWidth = 0;
15.
16. float tRotate = 0.0f; // the rotation value of the teapot
17. bool tpRotate = false; // Determine if the teapot rotate
18.
19. void Draw_Leg() // Draw the leg
20. {
21.     glScalef(1, 1, 3); // Stretch the model three times in the z direction
22.     glutSolidCube(1.0); // Draw a cube with a side length of one
23. }
24.
25. void Draw_Scene(float place[])
26. {
27.     glPushMatrix();
28.     glTranslatef(place[0], place[1], place[2]); // Move the teapot to its location
```

```

29.     glRotatef(90, 1, 0, 0); //Put the kettle body facing the camera
30.     glRotatef(tRotate, 0, 1, 0); // Control the rotation of the teapot

31.     glutSolidTeapot(1); // Draw a teapot
32.     glPopMatrix();
33.
34.     // Draw the cube
35.     glPushMatrix();
36.     glTranslatef(0, 0, 3.5);
37.     glScalef(5, 4, 1); // Stretch the model five times in the x direction and four times
    in the y direction
38.     glutSolidCube(1.0); // Draw a cube with a side length of one
39.     glPopMatrix();
40.
41.     // Draw four legs
42.     glPushMatrix();
43.     glTranslatef(1.5, 1, 1.5); // Move the leg to its position
44.     Draw_Leg(); // Draw a leg
45.     glPopMatrix();
46.
47.     glPushMatrix();
48.     glTranslatef(-1.5, 1, 1.5); // Move the leg to its position
49.     Draw_Leg(); // Draw a leg
50.     glPopMatrix();
51.
52.     glPushMatrix();
53.     glTranslatef(1.5, -1, 1.5); // Move the leg to its position
54.     Draw_Leg(); // Draw a leg
55.     glPopMatrix();
56.
57.     glPushMatrix();
58.     glTranslatef(-1.5, -1, 1.5); // Move the leg to its position
59.     Draw_Leg(); // Draw a leg
60.     glPopMatrix();
61.
62. }
63.
64. void updateView(int width, int height)
65. {
66.     glViewport(0, 0, width, height); // Reset The Current Viewport
67.
68.     glMatrixMode(GL_PROJECTION); // Select The Projection Matrix
69.     glLoadIdentity(); // Reset The Projection Matrix
70.

```



```

71.     float whRatio = (GLfloat)width / (GLfloat)height; // Set the display scale
72.
73.     if (bPersp) {
74.         gluPerspective(45, whRatio, 1, 100); // Perspective mode, the parameters of the
           function are angle of view, aspect ratio, near, far
75.     }
76.     else
77.         glOrtho(-3, 3, -3, 3, -100, 100);
78.
79.     glMatrixMode(GL_MODELVIEW); // Select The Modelview Matrix
80. }
81.
82. void reshape(int width, int height)
83. {
84.     if (height == 0) // Prevent A Divide By Zero By
85.     {
86.         height = 1; // Making Height Equal One
87.     }
88.
89.     wHeight = height;
90.     wWidth = width;
91.
92.     updateView(wHeight, wWidth); // Update perspective
93. }
94.
95. void idle()
96. {
97.     glutPostRedisplay(); // Call the current drawing function
98. }
99.
100. float eye[] = { 0, 0, 8 }; // The place of the camera
101. float center[] = { 0, 0, 0 }; // The place of the object
102. float place[] = { 0, 0, 5 }; //The place of the teapot
103.
104. void key(unsigned char k, int x, int y)
105. {
106.     switch (k)
107.     {
108.     case 27:
109.         case 'q': {exit(0); break; } // exit
110.         case 'p': {bPersp = !bPersp; updateView(wHeight, wWidth); break; } //Switch orthogr
           aphic projection and perspective projection
111.
112.         case 'r': {bAnim = !bAnim; break; } // Switch the rotation mode

```

```

113.     case 'o': {bWire = !bWire; break; } // Switch the rendering mode
114.
115.     case 'a': {// Move left
116.         center[0] = center[0] + 0.2f; // The object moves to the left
117.         eye[0] = eye[0] + 0.2f; // The viewpoint moves to the right
118.         break;
119.     }
120.     case 'd': {// Move right
121.         center[0] = center[0] - 0.2f; //The object moves to the right
122.         eye[0] = eye[0] - 0.2f; // The viewpoint moves to the left
123.         break;
124.     }
125.     case 'w': {// Move up
126.         center[1] = center[1] - 0.2f; //The object moves up
127.         eye[1] = eye[1] - 0.2f; // The viewpoint moves down
128.         break;
129.     }
130.     case 's': {// Move down
131.         center[1] = center[1] + 0.2f; // The object moves down
132.         eye[1] = eye[1] + 0.2f; // The viewpoint moves up
133.         break;
134.     }
135.     case 'z': {// Move forward
136.         center[2] = center[2] - 0.2f; // The object moves forward
137.         eye[2] = eye[2] - 0.2f; // The viewpoint moves backwards
138.         break;
139.     }
140.     case 'c': {// Move backwards
141.         center[2] = center[2] + 0.2f; // The object moves backwards
142.         eye[2] = eye[2] + 0.2f; // The viewpoint moves forward
143.         break;
144.     }
145.
146.     // Teapot related operations
147.     case 'l': {// Move the teapot to the right
148.         place[0] = place[0] + 0.2f; // The teapot moves right
149.         if (place[0] > 1.5f) place[0] = 1.5f; // Prevent the teapot from going beyond t
he desktop
150.         break;
151.     }
152.     case 'j': {// Move the teapot to the left
153.         place[0] = place[0] - 0.2f; // The teapot moves left
154.         if (place[0] < -1.5f) place[0] = -1.5f; // Prevent the teapot from going beyond
the desktop

```

```

155.         break;
156.     }
157.     case 'i': { // Move the teapot backwards
158.         place[1] = place[1] + 0.2f; // The teapot moves backwards
159.         if (place[1] > 1.5f) place[1] = 1.5f; // Prevent the teapot from going beyond t
            he desktop
160.         break;
161.     }
162.     case 'k': { // Move the teapot forward
163.         place[1] = place[1] - 0.2f; // The teapot moves forward
164.         if (place[1] < -1.5f) place[1] = -1.5f; // Prevent the teapot from going beyond
            the desktop
165.         break;
166.     }
167.     case 'e': { // Rotate the teapot
168.         tpRotate = !tpRotate;
169.         break;
170.     }
171.     }
172. }
173.
174.
175. void redraw()
176. {
177.
178.     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
179.     glLoadIdentity(); // Reset The Current Modelview M
        atrix
180.
181.     // The place of the camera,
182.     // the place of the object
183.     // and the observation direction
184.     gluLookAt(eye[0], eye[1], eye[2],
185.         center[0], center[1], center[2],
186.         0, 1, 0);
187.
188.     if (bWire) {
189.         glPolygonMode(GL_FRONT_AND_BACK, GL_LINE); // Set the polygon drawing mode: fro
            nt and back, line type
190.     }
191.     else {
192.         glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // Set the polygon drawing mode: fro
            nt and back, fill type
193.     }

```

```

194.
195.     glEnable(GL_DEPTH_TEST); // Open depth test
196.     glEnable(GL_LIGHTING); // Turn on lighting mode
197.     GLfloat white[] = { 1.0, 1.0, 1.0, 1.0 }; // Define color
198.     GLfloat light_pos[] = { 5,5,5,1 }; // Define the source position
199.
200.     glLightfv(GL_LIGHT0, GL_POSITION, light_pos); // Set the illumination position of t
        he zeroth source
201.     glLightfv(GL_LIGHT0, GL_AMBIENT, white); // Set the illumination color after the mu
        ltiple reflection of the zeroth source
202.     glEnable(GL_LIGHT0); // Turn on the zeroth light source
203.
204.     glRotatef(fRotate, 0, 1.0f, 0); // Rotate around Y axis
205.     glRotatef(-90, 1, 0, 0); // Make the table facing the camera
206.     glScalef(0.2, 0.2, 0.2); // Scale to make the object appear in the window at the ap
        propriate size
207.     Draw_Scene(place); // Draw Scene
208.
209.     if (bAnim) fRotate += 0.5f; // Teapot and table rotate together
210.     if (tpRotate) tRotate += 0.5f; // Teapot rotate
211.
212.     glutSwapBuffers();
213. }
214.
215. int main(int argc, char *argv[])
216. {
217.     glutInit(&argc, argv); // Initialize the glut library
218.     glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE); // Specify the window di
        splay mode that the function glutCreateWindow will create. RGB mode Double buffering
219.     glutInitWindowSize(480, 480); // Set the window position, which is the position of
        the top left corner of the window relative to the entire screen
220.     int windowHandle = glutCreateWindow("Experiment 3"); // Set the window title
221.
222.     glutDisplayFunc(redraw); // Register a draw callback function that specifies the fu
        nction to call when the window content needs to be redrawn
223.     glutReshapeFunc(reshape); // The callback function when the registration window size
        changes.
224.     glutKeyboardFunc(key); // Register key callback function
225.     glutIdleFunc(idle); // Register global callback function : call when idle
226.
227.     glutMainLoop(); // Glut event processing loop
228.     return 0;
229. }

```

