# 计算机图形学

## 作业六：光照效果

数据科学与计算机学院　17大数据与人工智能

17341015　陈鸿峥

### 一、 实验原理

本次实验中我同样采用了固定管线和渲染管线两种方法来实现。

#### 1. 固定管线

在固定管线中只需将对应的光源模型(`glLightfv`)设定好，并把材料的材质模型(`glMaterialfv`)调整好（这两个函数都有下列四个参数输入），并调用`glutSolidTeapot`输出即可。

`GL_AMBIENT,GL_DIFFUSE,GL_SPECULAR,GL_POSITION`

#### 2. 渲染管线

渲染管线的过程则麻烦得多，为实现smooth shading的效果，这里采用了gouraud shading算法，即对于**每一顶点**进行光照的计算。

依照以下几个步骤进行处理：

1. 对顶点周围的面法向量求平均，得到顶点法向，即

$$\mathbf{n_v} = \frac{\sum_{k=1}^{N} \mathbf{n}_k}{\left|\sum_{k=1}^{N} \mathbf{n}_k\right|}$$

2. 分别计算ambient、diffuse、specular三个分量（计算过程见`shader.vert`）
3. 最后得到LightingColor（一个比例系数），将其传入片元着色器
4. 片元着色器将物体原有颜色与LightingColor相乘，得到反射出来的片元颜色

注意在进行着色器操作时，需要将model、view、projection这三个变换矩阵传入，方便计算最终显示出来的物体坐标和颜色。在旧版的glsl语言中是支持直接读出MVP矩阵的，但是后面的版本都需要从用户程序传入。因此借助glm运算库，将这三个矩阵计算出来后再传入到着色器中，减少渲染管线的计算量。如变换后的向量坐标位置通过下式给出

TransformedVector = TranslationMatrix * RotationMatrix * ScaleMatrix * OriginalVector;

变换后的颜色则通过gouraud算法给出。

具体着色器编程操作则类似第三次作业，需要将顶点着色器程序(`shader.vert`)和片元着色器程序(`shader.frag`)读入，进行编译链接，然后将对应的属性值绑定到着色器输入变量上。

相关参考资料也都附在程序的注释中。

## 二、 实验结果

双击`teapot-shader.exe`即可运行，实验结果如图1所示。

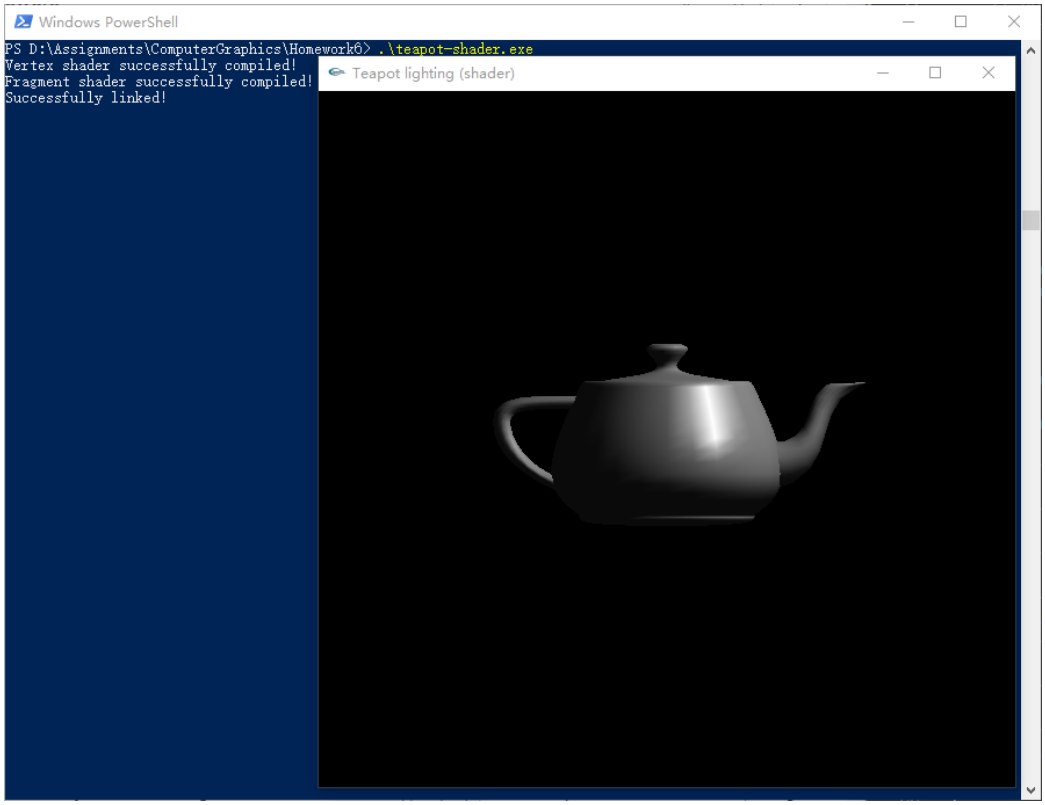


图 1: Teapot运行及光照结果

## 附录 A. 源代码

### 1. 固定管线(teapot.c)

```
#include <windows.h> // must be the first one to be included!
#include <stdlib.h>
#include <GL/glut.h>

void init(void)
{
    GLfloat ambient[] = {0.0, 0.0, 0.0, 1.0};
    GLfloat diffuse[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat position[] = {4.5, 4.5, 3, 1.0}; // fix position by model view matrix

    GLfloat lmodel_ambient[] = {0.2, 0.2, 0.2, 1.0};
    GLfloat local_view[] = {0.0};

```

```
15      // initialize lighting model
16      glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
17      glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
18      glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
19      glLightfv(GL_LIGHT0, GL_POSITION, position);
20      glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
21      glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);
22
23      glFrontFace(GL_CW);
24      glEnable(GL_LIGHTING); // global
25      glEnable(GL_LIGHT0); // each lighting
26      glEnable(GL_AUTO_NORMAL);
27      glEnable(GL_NORMALIZE);
28      glEnable(GL_DEPTH_TEST); // depth buffer
29      glEndList();
30  }
31
32  void display(void)
33  {
34      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
35      GLfloat mat[4];
36      glPushMatrix();
37      glTranslatef(2.0, 2.0, 0.0); // x, y, z
38
39      /*
40       * material properties
41       * constants reference from
42       * https://www.opengl.org/archives/resources/code/samples/redbook/teapots.c
43       */
44      mat[0] = 0.19225; mat[1] = 0.19225; mat[2] = 0.19225; mat[3] = 1.0; // rgb
45      glMaterialfv(GL_FRONT, GL_AMBIENT, mat);
46      mat[0] = 0.50754; mat[1] = 0.50754; mat[2] = 0.50754;
47      glMaterialfv(GL_FRONT, GL_DIFFUSE, mat);
48      // mat[0] = 0.508273; mat[1] = 0.508273; mat[2] = 0.508273;
49      mat[0] = 1; mat[1] = 1; mat[2] = 1; // reflect white lights
50      glMaterialfv(GL_FRONT, GL_SPECULAR, mat);
51      glMaterialf(GL_FRONT, GL_SHININESS, 0.2 * 128.0); // shine
52      glutSolidTeapot(1.0);
53
54      glPopMatrix();
55      glFlush();
56  }
57
58  /* Handler for window re-size event. Called back when the window first appears and
59     whenever the window is re-sized with its new width and height */
```

```
void reshape(GLsizei width, GLsizei height) { // GLsizei for non-negative integer
    // Compute aspect ratio of the new window
    if (height == 0) height = 1;            // To prevent divide by 0
    GLfloat aspect = (GLfloat)width / (GLfloat)height;

    // Set the viewport to cover the new window
    glViewport(0, 0, width, height);

    // Set the aspect ratio of the clipping volume to match the viewport
    glMatrixMode(GL_PROJECTION); // To operate on the Projection matrix
    glLoadIdentity();           // Reset
    // void glOrtho(GLdouble left, GLdouble right,
    //     GLdouble bottom, GLdouble top,
    //     GLdouble nearVal, GLdouble farVal);
    if (width <= height)
        glOrtho(0.0, 4.0, 0.0, 4.0*(GLfloat)height/(GLfloat)width, -10.0, 10.0);
    else
        glOrtho(0.0, 4.0*(GLfloat)width/(GLfloat)height, 0.0, 4.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(600,600);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Teapot Lighting");
    init();
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}

// gcc teapot.c -lglu32 -lglut32 -lopengl32 -o teapot.exe
```

编译指令如下：

```
gcc teapot.c -lglu32 -lglut32 -lopengl32 -o teapot.exe
```

**2. 渲染管线(teapot-shader.cpp)**

```
#include <windows.h> // must be the first one to be included!
#include <stdio.h>
#include <math.h>
```

```
4  #include <GL/glew.h>
5  #include <GL/glut.h>
6  #include <glm/glm.hpp>
7  #include <glm/gtc/matrix_transform.hpp>
8
9  GLuint teapotProgram;
10
11 const char* vertexShaderCode;
12 const char* fragShaderCode;
13 const char* loadShaderFile(const char *filename);
14
15 void init(void)
16 {
17     GLfloat light_ambient[] = {0.0, 0.0, 0.0, 1.0};
18     GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
19     GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
20     GLfloat light_position[] = {4.5, 4.5, 3 , 1.0}; // fix position by model view
           ↪ matrix
21     GLfloat obj_ambient[]  = {0.19225,0.19225,0.19225,1.0};
22     GLfloat obj_diffuse[]  = {0.50754,0.50754,0.50754};
23     GLfloat obj_specular[] = {1,1,1};
24     GLfloat obj_shininess[] = {64.0};
25
26     GLfloat lmodel_ambient[] = {0.2, 0.2, 0.2, 1.0};
27     GLfloat local_view[]   = {0.0};
28
29     // initialize lighting model
30     glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
31     glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
32     glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
33     glLightfv(GL_LIGHT0, GL_POSITION, light_position);
34     // material properties
35     glMaterialfv(GL_FRONT, GL_AMBIENT , obj_ambient); // rgb
36     glMaterialfv(GL_FRONT, GL_DIFFUSE , obj_diffuse);
37     glMaterialfv(GL_FRONT, GL_SPECULAR , obj_specular); // reflect white lights
38     glMaterialfv(GL_FRONT, GL_SHININESS, obj_shininess); // shine
39
40     glFrontFace(GL_CW);
41     glEnable(GL_LIGHTING); // global
42     glEnable(GL_LIGHT0); // each lighting
43     glEnable(GL_AUTO_NORMAL);
44     glEnable(GL_NORMALIZE);
45     glEnable(GL_DEPTH_TEST);
46
47     // position attribute
```

```
48      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
49      glEnableVertexAttribArray(0);
50      // normal attribute
51      glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 *
            ↪  sizeof(float)));
52      glEnableVertexAttribArray(1);
53  }
54
55  void display(void)
56  {
57      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
58      glUseProgram(teapotProgram);
59
60      // glm::mat4 proj = glm::perspective(45.0f, 800.0f / 600.0f, 0.1f, 100.0f);
61      GLfloat aspect = 1.0f;
62      glm::mat4 proj = glm::ortho(-3.0 * aspect, 3.0 * aspect, -3.0, 3.0, -10.0,
            ↪ 10.0);
63      glm::mat4 view = glm::lookAt(
64          glm::vec3(0.0f, -4.5f, 2.0f), // Camera is at (4.5,4.5,3), in World Space
65          glm::vec3(0.0f, 0.0f, 0.0f), // and looks at the origin
66          glm::vec3(0.0f, 0.0f, 1.0f)); // (0,1,0) Head is up
67      glm::mat4 model = glm::mat4(1.0f); // translate & rotate
68
69      glm::mat4 mvp = proj * view * model;
70      int lightPosLocation = glGetUniformLocation(teapotProgram,"lightPos");
71      glUniform3f(lightPosLocation,4.5,4.5,3);
72      int viewPosLocation = glGetUniformLocation(teapotProgram,"viewPos");
73      glUniform3f(viewPosLocation,4.5,4.5,4.5);
74      int lightColorLocation = glGetUniformLocation(teapotProgram,"lightColor");
75      glUniform3f(lightColorLocation,1,1,1);
76      int projectionLocation = glGetUniformLocation(teapotProgram,"projection");
77      glUniformMatrix4fv(projectionLocation,1,GL_FALSE,&proj[0][0]);
78      int modelLocation = glGetUniformLocation(teapotProgram,"model");
79      glUniformMatrix4fv(modelLocation,1,GL_FALSE,&model[0][0]);
80      int viewLocation = glGetUniformLocation(teapotProgram,"view");
81      glUniformMatrix4fv(viewLocation,1,GL_FALSE,&view[0][0]);
82      int objectColorLocation = glGetUniformLocation(teapotProgram,"objectColor");
83      glUniform3f(objectColorLocation,0.50754,0.50754,0.50754);
84      glutSolidTeapot(1.0);
85      glFlush();
86  }
87
88  /* Handler for window re-size event. Called back when the window first appears and
89     whenever the window is re-sized with its new width and height */
90  void reshape(GLsizei width, GLsizei height) { // GLsizei for non-negative integer
```

```
91      // Compute aspect ratio of the new window
92      if (height == 0) height = 1;            // To prevent divide by 0
93      GLfloat aspect = (GLfloat)width / (GLfloat)height;
94
95      // Set the viewport to cover the new window
96      // (x, y) is the left bottom corner
97      glViewport(0, 0, width, height); // i.e. the area that can be seen
98
99      // Set the aspect ratio of the clipping volume to match the viewport
100     glMatrixMode(GL_PROJECTION); // To operate on the Projection matrix
101     glLoadIdentity();          // Reset
102     /*
103      * The camera placed very far away, then become parallel projection
104      * an object appears to be the same size regardless of the depth
105      * void glOrtho(GLdouble left, GLdouble right,
106      *     GLdouble bottom, GLdouble top,
107      *     GLdouble nearVal, GLdouble farVal);
108      */
109     if (width >= height)
110         glOrtho(-3.0 * aspect, 3.0 * aspect, -3.0, 3.0, -10.0, 10.0);
111     else
112         glOrtho(-3.0, 3.0, -3.0 / aspect, 3.0 / aspect, -10.0, 10.0);
113     glMatrixMode(GL_MODELVIEW);
114 }
115
116 void keyPressed(unsigned char key, int x, int y)
117 {
118     printf("Pressed %c!\n", key);
119     display();
120 }
121
122 int main(int argc, char *argv[])
123 {
124     glutInit(&argc, argv);
125
126     glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
127
128     glutInitWindowPosition(100, 100);
129     glutInitWindowSize(600, 600);
130
131     glutCreateWindow("Teapot lighting (shader)");
132
133     glutReshapeFunc(reshape);
134     glutDisplayFunc(display);
135     glutKeyboardFunc(keyPressed);
```

```
136
137     glewInit();
138
139     int success;
140     char infoLog[1024];
141
142     // make shaders
143     vertexShaderCode = loadShaderFile("shader.vert");
144     fragShaderCode = loadShaderFile("shader.frag");
145     GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
146     glShaderSource(vertexShader, 1, &vertexShaderCode, NULL);
147     glCompileShader(vertexShader);
148     glGetShaderiv(vertexShader,GL_COMPILE_STATUS, &success);
149     if (!success){
150         glGetShaderInfoLog(vertexShader,1024,NULL,infoLog);
151         printf("Error: %s\n", infoLog);
152     } else
153         printf("Vertex shader successfully compiled!\n");
154
155     GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
156     glShaderSource(fragmentShader, 1, &fragShaderCode, NULL);
157     glCompileShader(fragmentShader);
158     glGetShaderiv(fragmentShader,GL_COMPILE_STATUS, &success);
159     if (!success){
160         glGetShaderInfoLog(fragmentShader,1024,NULL,infoLog);
161         printf("Error: %s\n", infoLog);
162     } else
163         printf("Fragment shader successfully compiled!\n");
164
165     teapotProgram = glCreateProgram();
166     glAttachShader(teapotProgram,vertexShader);
167     glAttachShader(teapotProgram,fragmentShader);
168
169     glLinkProgram(teapotProgram);
170     glGetShaderiv(teapotProgram,GL_LINK_STATUS, &success);
171     if (!success){
172         glGetShaderInfoLog(fragmentShader,1024,NULL,infoLog);
173         printf("Error: %s\n", infoLog);
174     } else
175         printf("Successfully linked!\n");
176
177     init();
178
179     // get into display
180     glutMainLoop();
```

```
181
182     return 0;
183 }
184
185 const char* loadShaderFile(const char *filename)
186 {
187     char* text = NULL;
188
189     if (filename != NULL) {
190         FILE *file = fopen(filename, "rt");
191
192         if (file != NULL) {
193             fseek(file, 0, SEEK_END);
194             int count = ftell(file);
195             rewind(file);
196
197             if (count > 0) {
198                 text = (char*)malloc(sizeof(char) * (count + 1));
199                 count = fread(text, sizeof(char), count, file);
200                 text[count] = '\0';
201             }
202             fclose(file);
203         }
204     }
205     return text;
206 }
207
208 // g++ -Iinclude teapot-shader.cpp -lopengl32 -lglew32 -lglut32 -o teapot-shader.
    ↪ exe
```

```
1  // Fragment shader
2
3  #version 330 core
4  out vec4 FragColor;
5
6  in vec3 LightingColor;
7
8  uniform vec3 objectColor;
9
10 void main()
11 {
12    FragColor = vec4(LightingColor * objectColor, 1.0);
13 }
```

```
1  // Vertex shader
```

```
2
3  #version 330 core
4  layout (location = 0) in vec3 aPos;
5  layout (location = 1) in vec3 aNormal;
6
7  out vec3 LightingColor; // resulting color from lighting calculations
8
9  uniform vec3 lightPos;
10 uniform vec3 viewPos;
11 uniform vec3 lightColor;
12
13 uniform mat4 model;
14 uniform mat4 view;
15 uniform mat4 projection;
16
17 /*
18  * Some references
19  * https://learnopengl.com/code_viewer.php?code=lighting/basic_lighting-exercise3
20  * http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/
21  */
22 void main()
23 {
24     gl_Position = projection * view * model * vec4(aPos, 1.0);
25
26     // gouraud shading
27     // ------------------------
28     vec3 Position = vec3(model * vec4(aPos, 1.0));
29     vec3 Normal = mat3(transpose(inverse(model))) * aNormal; // vec3(0.5,0.8,1);
30
31     // ambient
32     float ambientStrength = 0.1;
33     vec3 ambient = ambientStrength * lightColor;
34
35     // diffuse
36     vec3 norm = normalize(Normal);
37     vec3 lightDir = normalize(lightPos - Position);
38     float diff = max(dot(norm, lightDir), 0.0);
39     vec3 diffuse = diff * lightColor;
40
41     // specular
42     float specularStrength = 1.0; // this is set higher to better show the effect
           ↪ of Gouraud shading
43     vec3 viewDir = normalize(viewPos - Position);
44     vec3 reflectDir = reflect(-lightDir, norm);
45     float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
```

```
46      vec3 specular = specularStrength * spec * lightColor;
47
48      LightingColor = ambient + diffuse + specular;
49  }
```

编译指令如下：

```
g++ -Iinclude teapot-shader.c -lopengl32 -lglew32 -lglut32 -o teapot-shader.exe
```