可视化实验一报告

208704 杭天恺

January 2021

1 实验内容

- 熟悉 OpenGL 使用
- 显示一个球面和正方形表面
  - 数据结构、颜色（RGBA）、视见变换（改变观察点观察正方形显示）
  - 光照模型（改变镜面反射参数观察球面显示效果）
- 实验报告

2 环境设置

2.1 编程软件平台

Visual Studio 2019: https://visualstudio.microsoft.com/zh-hans/vs/

2.2 所需要的包

- GLFW: https://www.glfw.org/download.html
- GLAD: https://glad.dav1d.de/

其中 GLFW 需要通过 CMAKE 编译源代码生成。

3 实验原理

3.1 视见变换

3.1.1 模型变换

模型变换（model transform）是对物体本身的变换，常见的模型变换包括旋转，平移，缩放等。可以通过一个简单的 4×4 大小的仿射矩阵实现。

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{1}$$

其特例包括缩放变换

$$\mathbf{S}\left(s_x, s_y, s_z\right) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

平移变换

$$\mathbf{T}\left(t_x, t_y, t_z\right) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

以及利用 Rodrigues’ Rotation Formula 计算的旋转矩阵

$$\mathbf{R}(\mathbf{n}, \alpha) = \cos(\alpha)\mathbf{I} + (1 - \cos(\alpha))\mathbf{n}\mathbf{n}^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}} \tag{4}$$

3.1.2 视角变换

视角变换（view transform）用于与相机一起变换对象，其示意图如 1 所示。具体变换，首先将平移到远点，再进行变换。其变换过程对应矩阵操作如下

$$M_{\text{view }} = R_{\text{view }} T_{\text{view }} \tag{5}$$

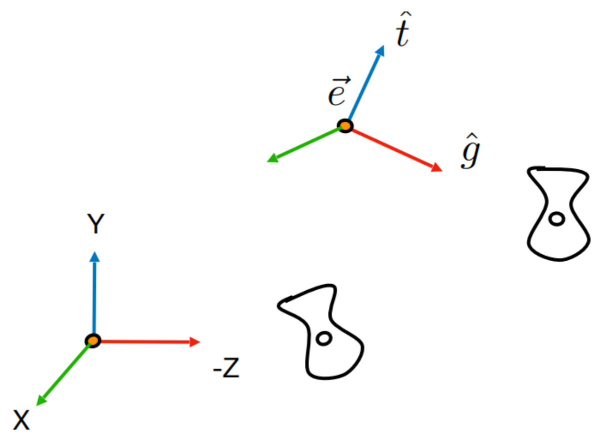

图 1: 视角变换示意图

其中

$$T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$R_{view} = \begin{bmatrix} x_{\hat{g}} \times \hat{t} & y_{\hat{g}} \times \hat{t} & z_{\hat{g} \times \hat{t}} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

3.1.3 投影变换

投影变换主要有两种，一种是正交投影（Orthographic projection），一种是透视投影变换（Perspective projection）。其示意图如图 2 所示。

我们在实际使用时，变换顺序为：model transform $\longrightarrow$ view transform $\longrightarrow$ projection transform，因此被简称为“MVP”变换。对应的 GLSL 着色器语言可以描述为

```
void main()
{
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = mat3(transpose(inverse(model))) * aNormal;
    gl_Position = projection * view * vec4(FragPos, 1.0);
}
```

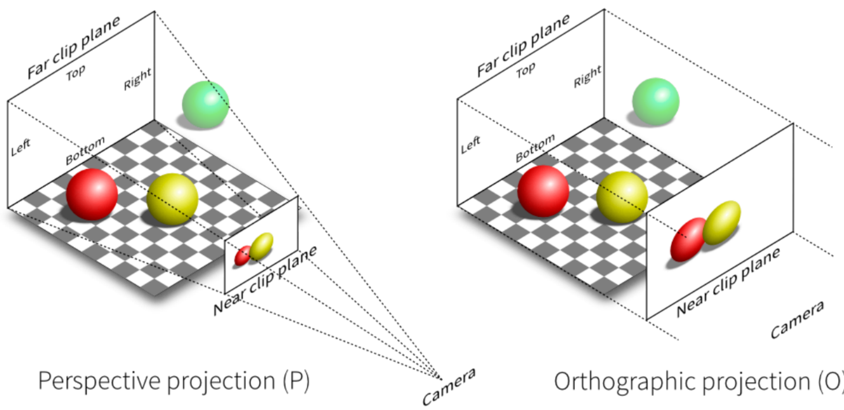

图 2: 投影变换

3.2 光照模型

冯氏光照模型的主要结构由 3 个元素组成：环境 (Ambient)、漫反射 (Diffuse) 和镜面 (Specular) 光照。

- 环境光照 (Ambient Lighting)：即使在黑暗的情况下，世界上也仍然有一些光亮 (月亮、一个来自远处的光)，所以物体永远不会是完全黑暗的。我们使用环境光照来模拟这种情况，也就是无论如何永远都给物体一些颜色。
- 漫反射光照 (Diffuse Lighting)：模拟一个发光物对物体的方向性影响 (Directional Impact)。它是冯氏光照模型最显著的组成部分。面向光源的一面比其他面会更亮。
- 镜面光照 (Specular Lighting)：模拟有光泽物体上面出现的亮点。镜面光照的颜色，相比于物体的颜色更倾向于光的颜色。

其相关表达式如下所示，

$$
\begin{aligned}
L &= L_a + L_d + L_s \\
&= k_a I_a + k_d \left(I/r^2\right) \max(0, \mathbf{n}\cdot\mathbf{l}) + k_s \left(I/r^2\right) \max(0, \mathbf{n}\cdot\mathbf{h})^p
\end{aligned}
\tag{8}
$$

```
void main()
{
```

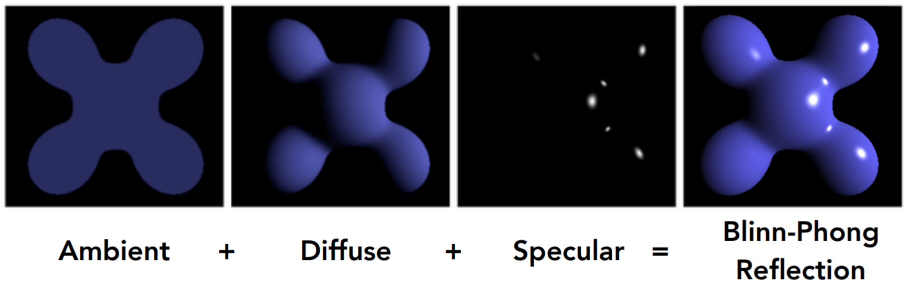

图 3: 光照模型示意图

```
    // ambient
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    // specular
    float specularStrength = 0.5;
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
    vec3 specular = specularStrength * spec * lightColor;

    vec3 result = (ambient + diffuse + specular) * objectColor;
    FragColor = vec4(result, 1.0);
}
```

4 实验结果

4.1 正方体表面-视见变换

不同视角观察得到的立方体表面如图 4 所示。

4.2 球面-光照模型

改变不同的观察视角，效果如 5 所示。

改变高光项系数，其结果如图 6 所示。第一排从左到右对应的高光项系数分别为 0.0，0.2，0.4，第二排从左到右对应的高光项系数分别为 0.5，0.6.0.8。

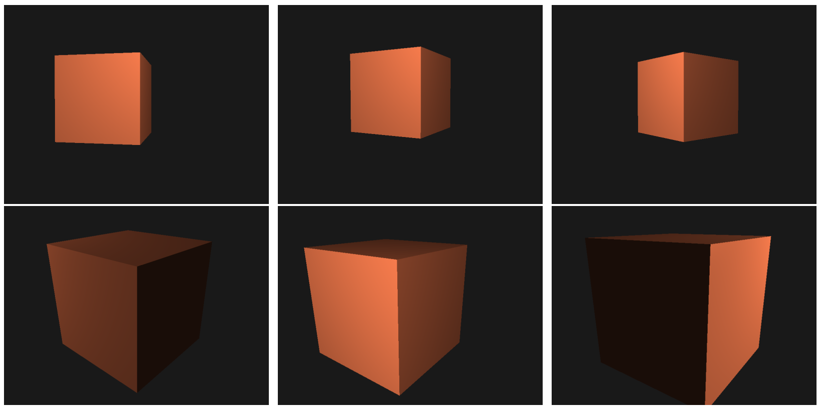

图 4: 不同视角观察得到的立方体表面

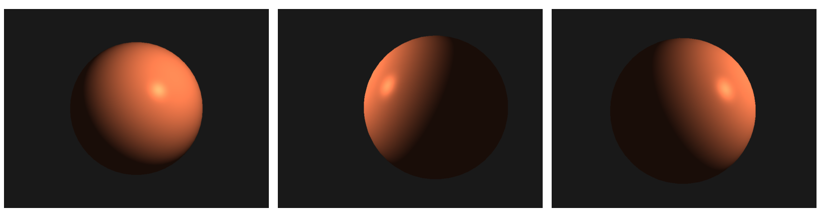

图 5: 观察不同视角球面观察结果

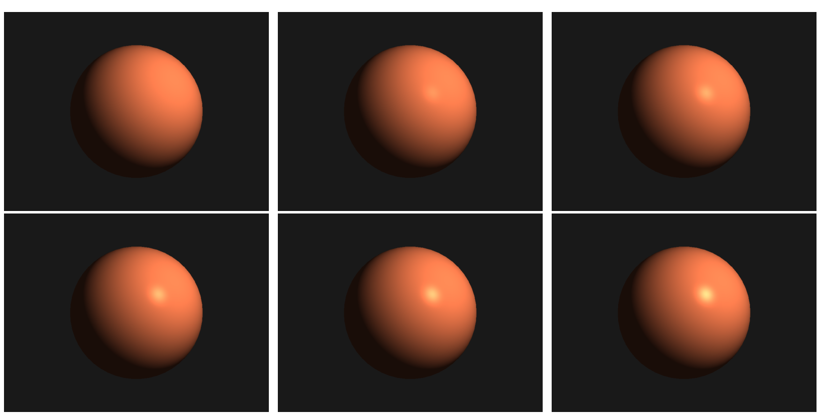

图 6: 改变高光项系数显示结果

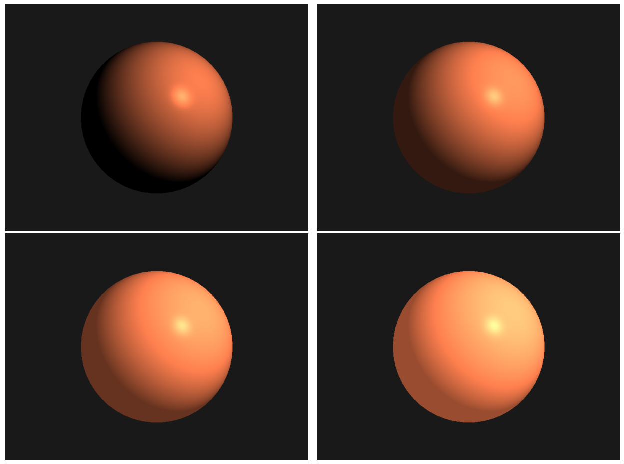

图 7: 改变漫反射项系数

改变漫反射项系数，其结果如图 7 所示。左上角对应的系数为 0，右上角为 0.2，左下角为 0.4，右下角为 0.6.

5 总结

本次实验我们通过学习 openGL 相关技术完成了一些绘制工作。具体代码见附录链接或者报告附件。主要通过正方体表面的绘制，我们学习了图形学中基础的 MVP 变换，即 model transform，view transform 以及 projection transform。通过球面绘制，我们学习了有关 Phong 模型进行光照着色的相关内容。总的来说，获益匪浅。

6 附录

6.1 代码链接

https://github.com/TiankaiHang/GlVisualization.git

6.2 主体代码

```
int draw_shape(int shape_flag) {
    // glfw: initialize and configure
    // ------------------------------
```

```
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    // --------------------
    GLFWwindow* window = glfwCreateWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "drawn_by_tiankai", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
    glfwSetCursorPosCallback(window, mouse_callback);
    glfwSetScrollCallback(window, scroll_callback);

    // tell GLFW to capture our mouse
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

    // glad: load all OpenGL function pointers
    // ---------------------------------------
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    // configure global opengl state
    // -----------------------------
    glEnable(GL_DEPTH_TEST);

    // build and compile our shader zprogram
    // ------------------------------------
    Shader lightingShader("myShader/basic_lightning.vs", "myShader/basic_lightning.fs");
    Shader lightCubeShader("myShader/cubic_lightning.vs", "myShader/cubic_lightning.fs");

    // set up vertex data (and buffer(s)) and configure vertex attributes
    // ------------------------------------------------------------------
    vector<float> Vertices;
    vector<int> Indices;

    if (shape_flag == DRAW_CUBIC)
        generateCubicData(Vertices, Indices);
    else if (shape_flag == DRAW_SPHERE)
        generateSphereData(Vertices, Indices);
    else
        throw "No implementations yet!";
    // first, configure the cube's VAO (and VBO)
    unsigned int VBO, cubeVAO, EBO;
    glGenVertexArrays(1, &cubeVAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);

    glBindVertexArray(cubeVAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, Vertices.size() * sizeof(float), &Vertices[0], GL_STATIC_DRAW);
```

```
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, Indices.size() * sizeof(int), &Indices[0], GL_STATIC_DRAW);

    // position attribute
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
    // normal attribute
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);

    //glBindBuffer(GL_ARRAY_BUFFER, 0);
    //glBindVertexArray(0);


    // second, configure the light's VAO (VBO stays the same; the vertices are the same for the light object which is also a 3D cube)
    unsigned int lightCubeVAO;
    glGenVertexArrays(1, &lightCubeVAO);
    glBindVertexArray(lightCubeVAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // note that we update the lamp's position attribute's stride to reflect the updated buffer data
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    // cout << Vertices.size() << " " << Indices.size() << endl;


    // render loop
    // -----------
    while (!glfwWindowShouldClose(window))
    {
        // per-frame time logic
        // --------------------
        float currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        // input
        // -----
        processInput(window);

        // render
        // ------
        glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // be sure to activate shader when setting uniforms/drawing objects
        lightingShader.use();
        lightingShader.setVec3("objectColor", 1.0f, 0.5f, 0.31f);
        lightingShader.setVec3("lightColor", 1.0f, 1.0f, 1.0f);
        lightingShader.setVec3("lightPos", lightPos);
        lightingShader.setVec3("viewPos", camera.Position);

        // view/projection transformations
        glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCREEN_WIDTH / (float)SCREEN_HEIGHT, 0.1f, 100.0f);
        glm::mat4 view = camera.GetViewMatrix();
        lightingShader.setMat4("projection", projection);
        lightingShader.setMat4("view", view);

        // world transformation
        glm::mat4 model = glm::mat4(1.0f);
        lightingShader.setMat4("model", model);
```

```

        // render the cube
        glBindVertexArray(cubeVAO);
        //glDrawArrays(GL_TRIANGLES, 0, 36);
        glDrawElements(GL_TRIANGLES, Indices.size(), GL_UNSIGNED_INT, 0);


        // also draw the lamp object
        //lightCubeShader.use();
        //lightCubeShader.setMat4("projection", projection);
        //lightCubeShader.setMat4("view", view);
        //model = glm::mat4(1.0f);
        //model = glm::translate(model, lightPos);
        //model = glm::scale(model, glm::vec3(0.2f)); // a smaller cube
        //lightCubeShader.setMat4("model", model);

        //glBindVertexArray(lightCubeVAO);
        //glDrawArrays(GL_TRIANGLES, 0, 36);


        // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
        // -------------------------------------------------------------------------------
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    // optional: de-allocate all resources once they've outlived their purpose:
    // ------------------------------------------------------------------------
    glDeleteVertexArrays(1, &cubeVAO);
    glDeleteVertexArrays(1, &lightCubeVAO);
    glDeleteBuffers(1, &VBO);
    glDeleteBuffers(1, &EBO);

    // glfw: terminate, clearing all previously allocated GLFW resources.
    // ------------------------------------------------------------------
    glfwTerminate();
}
```