

计算机图形学原理

-----教程 3
李瑞辉

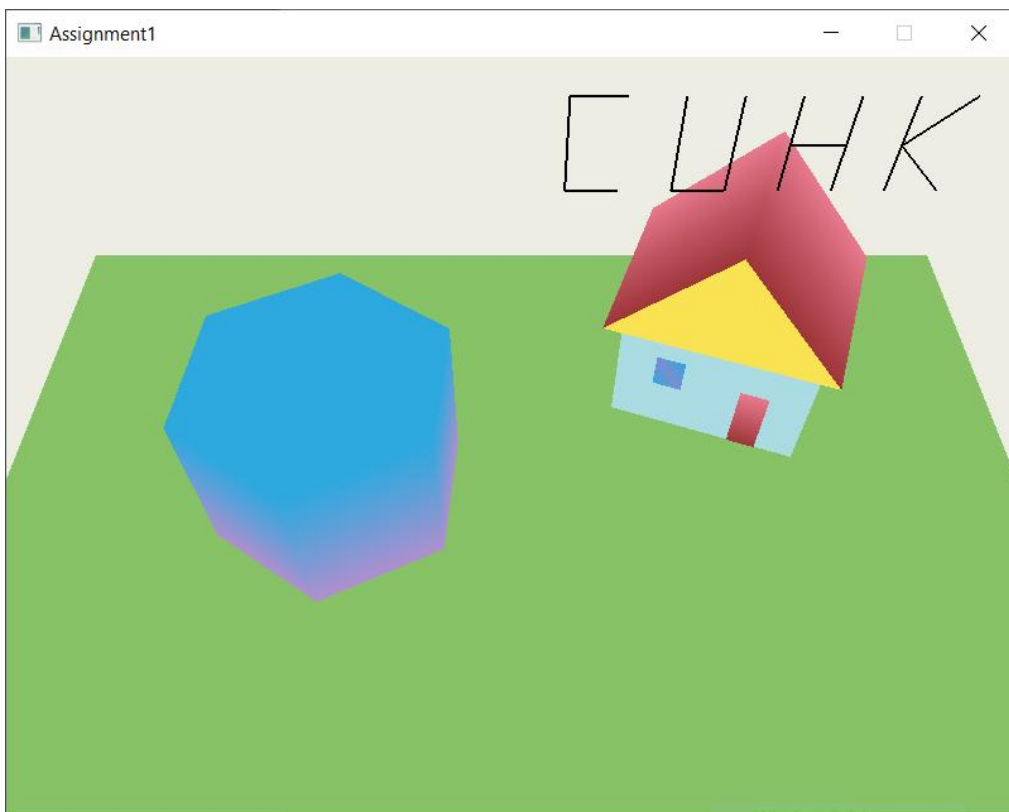
大纲

作业 1 中的基本要求

- 如何渲染 3D 对象
- 绘制多个对象
- 关于作业 1 的提示

作业 1

3D 场景（见一些很好的例子）。



+

用户交互

作业 1

基本要求：

- OpenGL 代码应该使用OpenGL 3.0+ 的可编程管线而不是固定管线。
- 至少绘制一个2D 对象和两个3D 对象。
- 确保至少有一个对象是用索引绘制的。
- 创建至少三种键盘和/或鼠标事件，例如旋转、平移和缩放。
- 设计对象变换，包括旋转、平移和缩放。
- 使用透视投影（45.0度，任意方位，0.1，20.0）绘制场景。
- 启用深度测试 实现遮挡。

作业 1

基本要求:

- OpenGL 代码应该使用OpenGL 3.0+ 的可编程管线而不是固定管线。

```
glBegin ( type ) ;  
  
    glVertex3f ( ... ) ;  
  
    glVertex3f ( ... ) ;  
  
    glVertex3f ( ... ) ;  
  
    .....  
  
glEnd() ;
```

```
glMatrixMode ( GL_MODELVIEW ) ;  
glLoadIdentity () ;  
glPushMatrix() ;  
    glTranslatef ( ball_X , ball_Y , ball_Z ) ;  
    glRotatef ( ball_ang , ball_dirX , ball_dirY , ball_dirZ ) ;  
    glScalef ( ball_Sx , ball_Sy , ball_Sz ) ;  
    Draw_ball() ;  
glPopMatrix() ;  
glMatrixMode() ;  
    glTranslatef ( cube_X , cube_Y , cube_Z ) ;  
    glRotatef ( cube_ang , cube_dirX , cube_dirY , cube_dirZ ) ;  
    glScalef ( cube_Sx , cube_Sy , cube_Sz ) ;  
    Draw_cube() ;  
glPopMatrix() ;
```

作业 1

基本要求:

- OpenGL 代码应该使用OpenGL 3.0+ 的可编程管线而不是固定管线。

```
const GLfloat triangle[] =  
{  
x, y, z    -0.5f, -0.5f, +0.0f,  // left  
r, g, b    +1.0f, +0.0f, +0.0f,  // color  
  
           +0.5f, -0.5f, +0.0f,  // right  
           +1.0f, +0.0f, +0.0f,  
  
           +0.0f, +0.5f, +0.0f,  // top  
           +1.0f, +0.0f, +0.0f,  
};
```

定义对象数据

```
GLuint vaoID;  
glGenVertexArrays(1, &vaoID);  
glBindVertexArray(vaoID);  //first VAO  
  
GLuint vboID;  
glGenBuffers(1, &vboID);  
glBindBuffer(GL_ARRAY_BUFFER, vboID);  
glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);  
  
// 1st attribute: vertex position  
glEnableVertexAttribArray(0);  
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), 0);  
  
// 2nd attribute: vertex color  
glEnableVertexAttribArray(1);  
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),  
    (char*)(3 * sizeof(float)));
```

使用 VAO 和 VBO!

作业 1

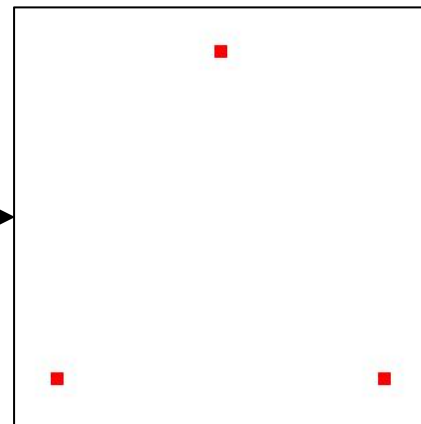
基本要求:

- 至少绘制一个2D 对象和两个3D 对象。

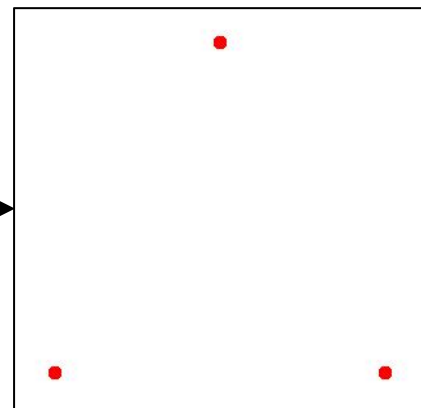
示例: GL_POINTS (可选)

```
const GLfloat triangle[] =  
{  
    -0.5f, -0.5f, +0.0f, //left  
    +1.0f, +0.0f, +0.0f, //color  
  
    +0.5f, -0.5f, +0.0f, //right  
    +1.0f, +0.0f, +0.0f,  
  
    +0.0f, +0.5f, +0.0f, //top  
    +1.0f, +0.0f, +0.0f,  
};
```

```
glPointSize(10.0f);  
glDrawArrays(GL_POINTS, 0, 3);
```



```
glEnable(GL_POINT_SMOOTH);  
glPointSize(10.0f);  
glDrawArrays(GL_POINTS, 0, 3);
```



作业 1

基本要求:

- 至少绘制一个2D 对象和两个3D 对象。

示例: GL `const GLfloat triangle[] =`

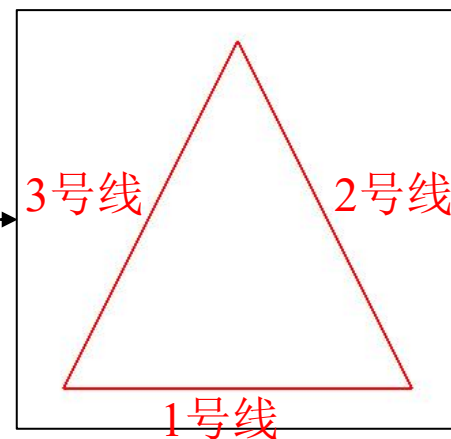
```
{
    -0.5f, -0.5f, +0.0f, //left
    +1.0f, +0.0f, +0.0f, //color
    +0.5f, -0.5f, +0.0f, //right
    +1.0f, +0.0f, +0.0f,
    +0.5f, -0.5f, +0.0f, //right
    +1.0f, +0.0f, +0.0f,
    +0.0f, +0.5f, +0.0f, //top
    +1.0f, +0.0f, +0.0f,
    +0.0f, +0.5f, +0.0f, //top
    +1.0f, +0.0f, +0.0f,
    -0.5f, -0.5f, +0.0f, //left
    +1.0f, +0.0f, +0.0f, //color
};
```

1号线

2号线

3号线

→ `glLineWidth(1.5f);`
`glDrawArrays(GL_LINES, 0, 6);`



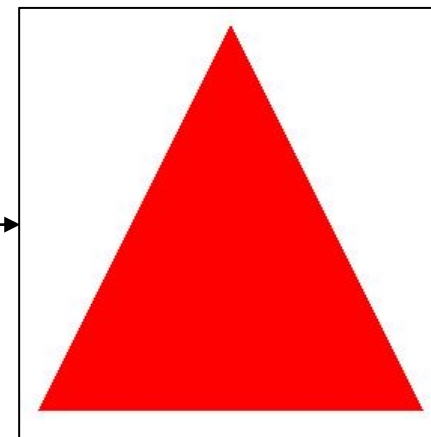
作业 1

基本要求:

- 至少绘制一个2D 对象和两个3D 对象。
示例: GL_TRIANGLES

```
const GLfloat triangle[] =  
{  
    -0.5f, -0.5f, +0.0f, //left  
    +1.0f, +0.0f, +0.0f, //color  
  
    +0.5f, -0.5f, +0.0f, //right  
    +1.0f, +0.0f, +0.0f,  
  
    +0.0f, +0.5f, +0.0f, //top  
    +1.0f, +0.0f, +0.0f,  
};
```

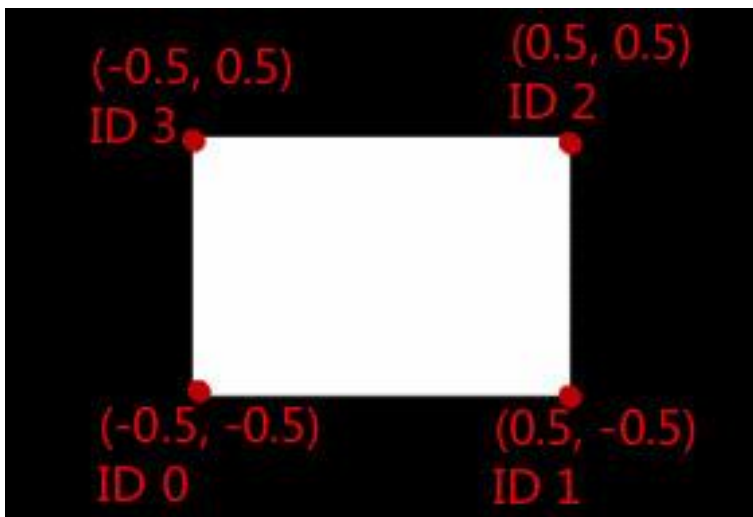
→ glDrawArrays(GL_TRIANGLES, 0, 6); →



Assignment 1

基本要求:

- 确保至少有一个对象是用索引绘制的。



```
void glDrawArrays(GLenum mode,  
                  GLint first,  
                  GLsizei count);
```

```
const GLfloat square[] =  
{  
    -0.5f, -0.5f, +0.0f, // position 0  
    +0.5f, -0.5f, +0.0f, // position 1  
    -0.5f, +0.5f, +0.0f, // position 3  
  
    +0.5f, -0.5f, +0.0f, // position 1  
    +0.5f, +0.5f, +0.0f, // position 2  
    -0.5f, +0.5f, +0.0f, // position 3  
};
```

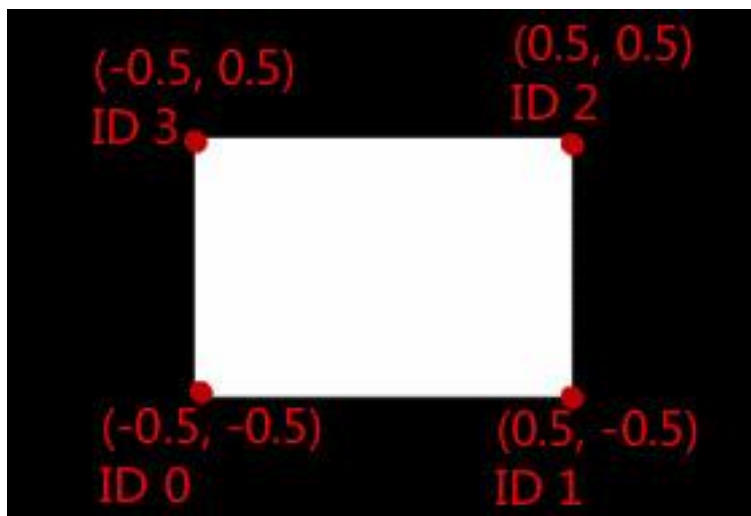
```
glDrawArrays(GL_TRIANGLES, 0, 6);
```

without indexing: redundant <https://docs.gl/gl4/glDrawArrays>

作业 1

基本要求:

- 确保至少有一个对象是用索引绘制的。（参考三角代码注释行！）



```
const GLfloat square[] = {  
    -0.5f, -0.5f, +0.0f, // position 0  
    +0.5f, -0.5f, +0.0f, // position 1  
    +0.5f, +0.5f, +0.0f, // position 2  
    -0.5f, +0.5f, +0.0f, // position 3  
};  
  
GLuint indices[] = {  
    0, 1, 3,  
    1, 2, 3  
};
```

```
void glDrawElements(GLenum mode,  
    GLsizei count,  
    GLenum type,  
    const GLvoid * indices);
```

```
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

带索引

<https://docs.gl/gl4/glDrawElements>

作业 1

基本要求:

- 确保至少有一个对象是用索引绘制的。（请参阅三角形代码注释行！）

在 `sendDataToOpenGL`

```
glGenBuffers(1, &vboID);
glBindBuffer(GL_ARRAY_BUFFER, vboID);
glBufferData(GL_ARRAY_BUFFER,
             sizeof(plane),
             plane, GL_STATIC_DRAW);

// vertex position
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), 0);
// vertex color
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (char*)(3 * sizeof(float)));

// draw indices
GLuint indexBufferID;
glGenBuffers(1, &indexBufferID);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexBufferID);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices),
             indices, GL_STATIC_DRAW);
```

照常

指数

作业 1


基本要求:

- 创建至少三种键盘和/或鼠标事件。

```
void key_callback(GLFWwindow* window,
    int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if (key == GLFW_KEY_W && action == GLFW_PRESS) { ... }
    if (key == GLFW_KEY_A && action == GLFW_PRESS) { ... }
    if (key == GLFW_KEY_S && action == GLFW_PRESS) { ... }
    if (key == GLFW_KEY_D && action == GLFW_PRESS) { ... }
}

void mouse_button_callback(GLFWwindow* window,
    int button, int action, int mods)
{
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS) {
        ...
    }
}
```

由你决定

https://www.glfw.org/docs/latest/input_guide.html

作业 1

基本要求：

- 设计对象变换，包括旋转、平移和缩放。

转换命令：

1. Translate : `glm::translate (mat4 (1.0f), vec3 (dx, dy, dz)) ;`

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Scale : `glm::Scale (mat4 (1.0f), vec3 (x, y, z)) ;`

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

作业 1

基本要求:

- 设计对象变换，包括旋转、平移和缩放。（检查！度数或弧度）

3. 旋转一个圆形X 轴: `glm::rotate(mat4(1.0f), θ , vec3(1, 0, 0));`

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. 旋转Y轴: `glm::rotate(mat4(1.0f), θ , vec3(0, 1, 0));`

$$\begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5. 旋转一个圆形Z 轴: `glm::rotate(mat4(1.0f), θ , vec3(0, 0, 1));`

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

作业 1

基本要求:

- 设计对象变换，包括旋转、平移和缩放。（检查！度数或弧度）
- 复合变换（例如，旋转后平移）

• 方式 1 (glm):

```
glm::mat4 trans = glm::mat4(1.0f);  
trans = glm::translate(trans, glm::vec3(0.5f, -0.5f, 0.0f));  
trans = glm::rotate(trans, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));
```

```
glm::mat4 rotation = glm::rotate(glm::mat4(1.0f), glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));  
glm::mat4 translation = glm::translate(glm::mat4(1.0f), glm::vec3(0.5f, -0.5f, 0.0f));  
glm::mat4 trans = translation * rotation;
```


作业 1

代码示例：翻译您的对象

主.cpp

```
mat4 modelTransformMatrix = glm::translate(mat4(), vec3(-0.45f, 0.45f, 0.0f));
GLint modelTransformMatrixUniformLocation =
    glGetUniformLocation(programID, "modelTransformMatrix");
glUniformMatrix4fv(modelTransformMatrixUniformLocation, 1,
    GL_FALSE, &modelTransformMatrix[0][0]);
```

顶点着色器代码.glsl

```
in layout(location=0) vec3 position;
in layout(location=1) vec3 vertexColor;

uniform mat4 modelTransformMatrix;

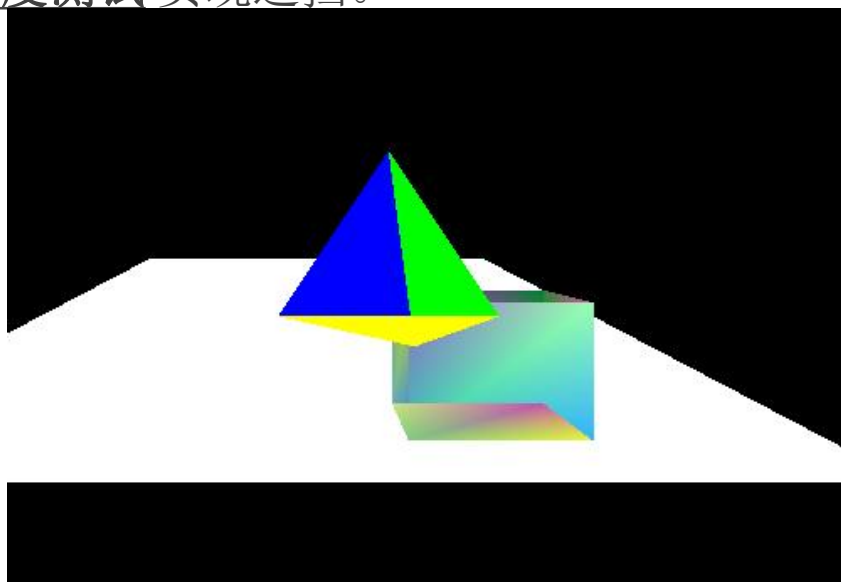
out vec3 theColor;

void main()
{
    vec4 v = vec4(position, 1.0);
    vec4 newPosition = modelTransformMatrix * v;
    gl_Position = newPosition;
    theColor = vertexColor;
}
```

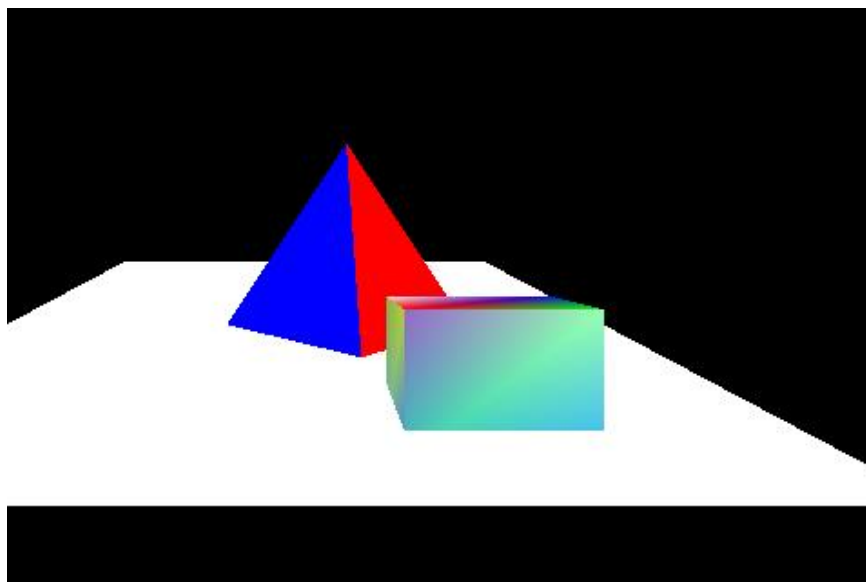
作业 1

基本要求：

- 启用深度测试 实现遮挡。



深度测试禁用



深度测试已启用

作业 1

基本要求:

- 启用深度测试 实现遮挡。

```
glEnable(GL_DEPTH_TEST);
```

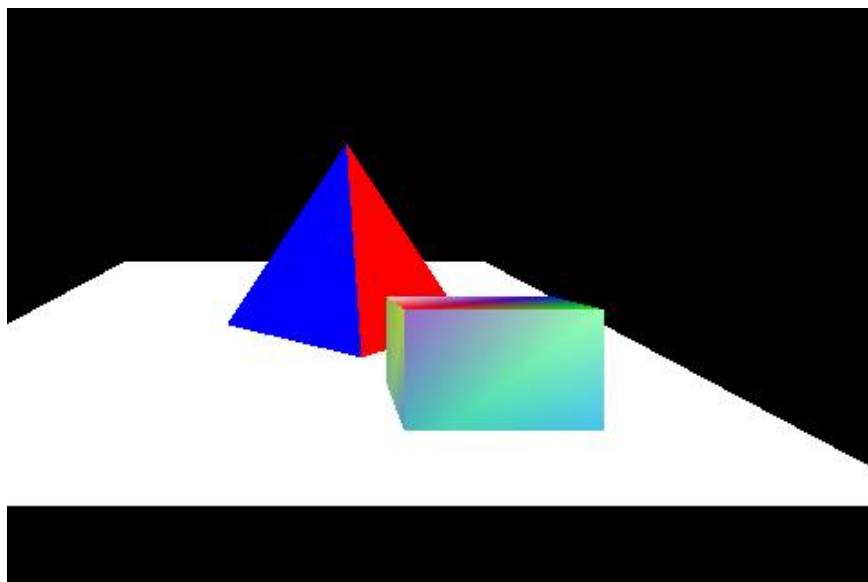
只运行一次

(例如, 在`initializedGL`中添加这一行)

```
glClear(GL_COLOR_BUFFER_BIT  
        | GL_DEPTH_BUFFER_BIT);
```

每次运行

(在`paintGL`中添加这一行)



深度测试已启用

大纲

- 作业 1 中的基本要求

如何渲染 **3D** 对象
(模型、视图、投影矩阵)

- 绘制多个对象
- 关于作业 1 的提示

作业 1

基本要求:

- 使用透视投影（45.0度，任意方位，0.1，20.0）绘制场景。

- 投影：3D场景⇒2D画面

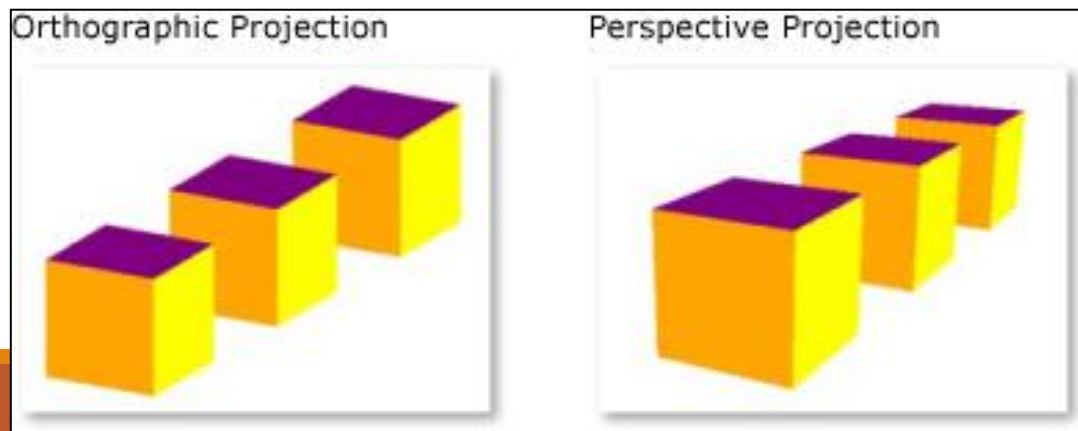
投影方法

透视投影

正投影

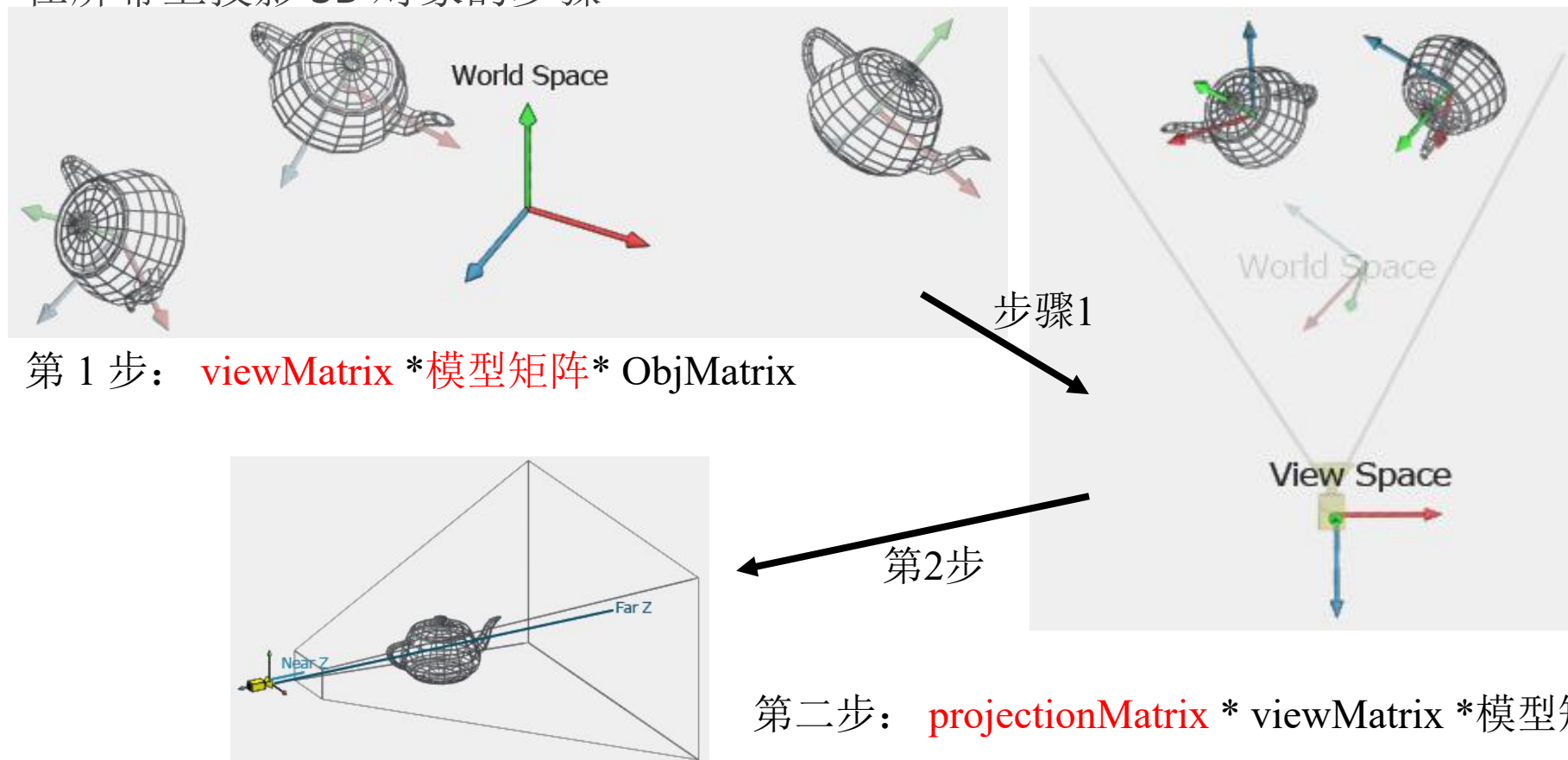
- 最近的东西看起来更大
- 有消失点
- 平行线无限远相接

- 一切似乎都是平等的
- 无消失点
- 平行线永不相交



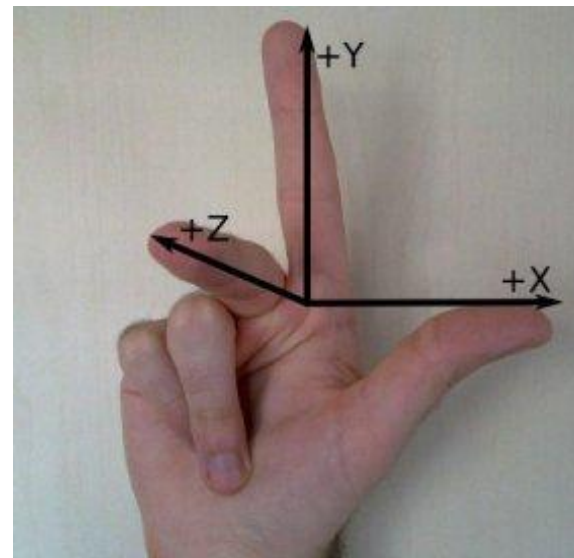
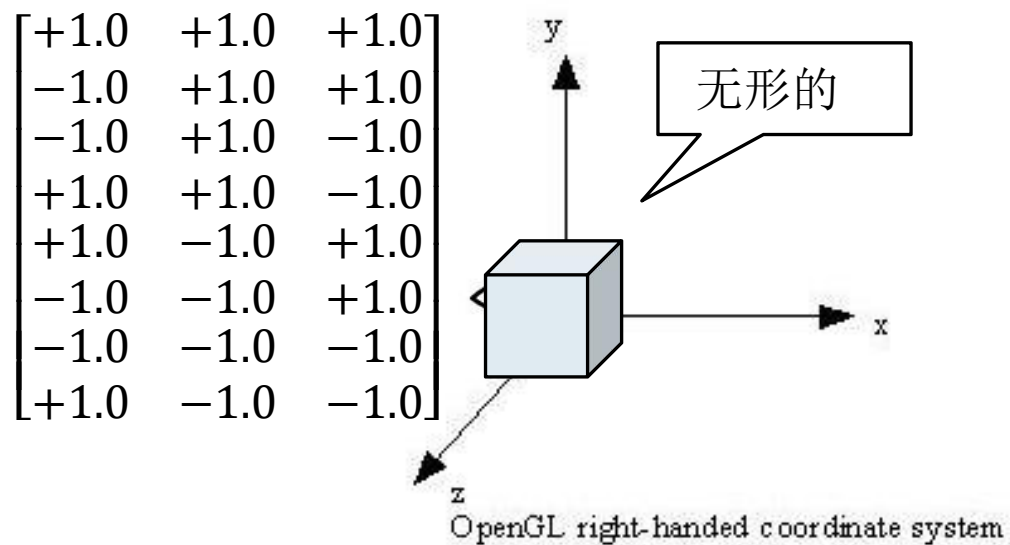
作业 1

在屏幕上投影 3D 对象的步骤



作业 1

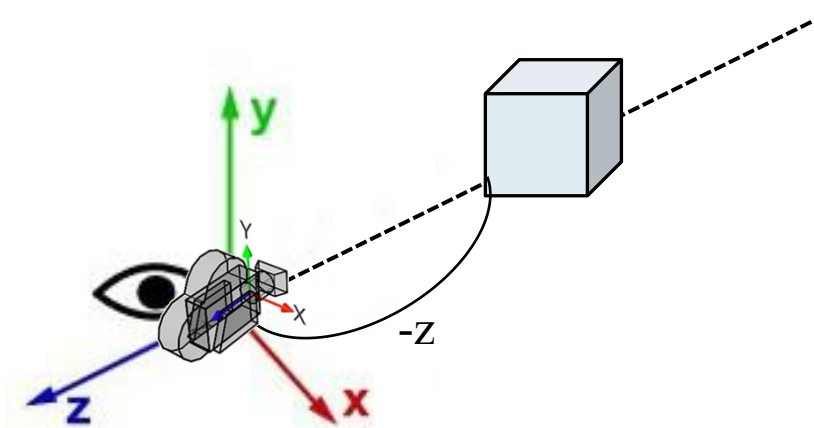
3D 坐标系



如果未指定，相机（眼睛）将放置在原点。
我们最好设置视图矩阵！

作业 1

将物体移动到相机前面。

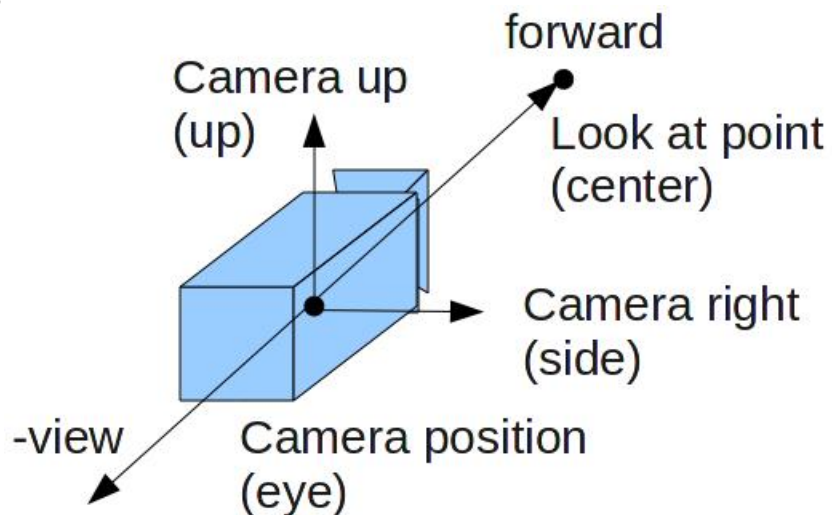


`glm::translate(mat4(1.0f), vec3(0.0f, 0.0f, -z))`, z为正
右手坐标系。

作业 1

查看矩阵示例

- 眼睛位于 (0.0, 0.0, 5.0) 并注视 (0.0, 0.0, 0.0)。
- 尝试改变眼睛位置和且标并查看结果。



glm::lookat (位置, 目标,

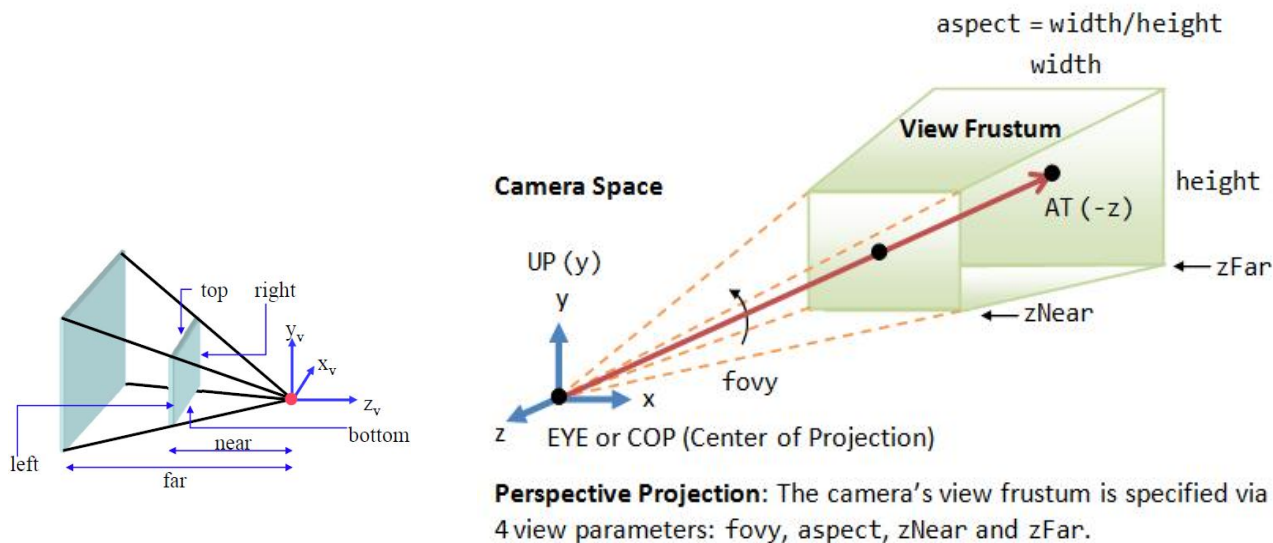
```
glm::mat4 viewMatrix = glm::lookAt(glm::vec3(0.0f, 0.0f, 5.0f),  
    glm::vec3(0.0f, 0.0f, 0.0f),  
    glm::vec3(0.0f, 1.0f, 0.0f));
```

• <https://learnopengl.com/Getting-started/相机>

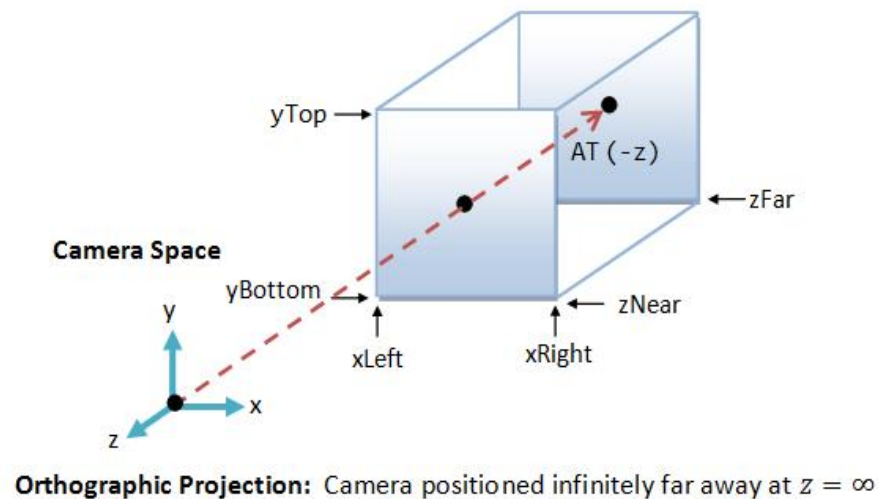
作业 1

投影矩阵示例

- 将 3D 位置转换为屏幕上的 2D 位置。



`glm::perspective(fovy , aspect , zNear , zFar)`
或者 `glm::frustum (左, 右, 下, 上, zNear , zFar)`



`glm::ortho (左, 右, 下, 上, zNear , zFar)`
默认情况下: 正交 (-1、1、-1、1、-1、1)

作业 1

投影矩阵代码

模型矩阵：平移、旋转或缩放物体（可能随物体变化）。

视图矩阵：你的相机在哪里（可以在每张图中设置一次）。

投影矩阵：3D场景⇒2D图片（一般不会变）。

主.cpp

```
glm::mat4 projectionMatrix = glm::perspective(glm::radians(45.0f), 1.0f, 1.0f, 100.0f);
GLint projectionMatrixUniformLocation =
    glGetUniformLocation(programID, "projectionMatrix");
glUniformMatrix4fv(projectionMatrixUniformLocation, 1,
    GL_FALSE, &projectionMatrix[0][0]);
```



注意参数
(参见规范)

顶点着色器代码.glsl

```
in layout(location = 0) vec3 position;
in layout(location = 1) vec3 vertexColor;
```

```
uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;
```

```
out vec3 theColor;
```

```
void main()
```

```
{
```

```
    vec4 v = vec4(position, 1.0);
```

```
    vec4 out_position = projectionMatrix * viewMatrix * modelMatrix * v;
```

```
    gl_Position = out_position;
```

```
    theColor = vertexColor;
```

```
}
```

Bonus：例如，您可以更改fov
并在 README.txt 中描述视觉效果

Try with different parameters in the perspective projection (e.g., fov, etc.) and discuss the effect.

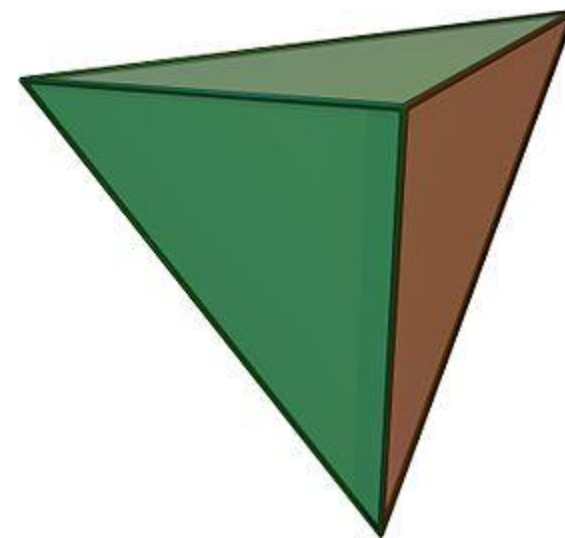
5%

作业 1

渲染 3D 对象

1. (可选) 定义模型矩阵 (平移、旋转或缩放 3D 对象)。
2. 定义适当的视图和投影矩阵 (在 `main.cpp paintGL` 中)。
重要提示: 因为最好使用良好的视点和投影来查看 3D 对象。
第 23 页和第 25 页。
3. 相应地修改着色器代码。P25.
4. 渲染三角形并检查矩阵设置是否正确。
5. 使用数据创建 3D 对象。
(例如, 向三角形添加一个顶点 \Rightarrow 四面体, 或创建一个立方体)

提示: 尝试一些简单的东西并确保你的模型、视图、投影矩阵是正确的!



大纲

- 作业 1 中的基本要求
- 如何渲染 3D 对象

绘制多个对象

- 关于作业 1 的提示

作业 1

绘制多个对象

1. 例如，您有两个对象。
将全局 VAO 变量声明为长度为 2 的数组。

```
#include "Dependencies/glew/glew.h"
#include "Dependencies/GLFW/glfw3.h"

#include "Dependencies/glm/glm.hpp"
#include "Dependencies/glm/gtc/matrix_transform.hpp"

#include <iostream>
#include <fstream>

GLint programID;
GLuint vao[2];
```

作业 1

绘制多个对象

2. 在 `sendDataToOpenGL` 中，对于第一个对象，将 `vaoID` 替换为 `vao [0]`。
(是否将 VBO 声明为数组并不那么重要，您可以简单地遵循这一点)。

```
GLuint vboID;  
glGenVertexArrays(1, &vao[0]);  
glBindVertexArray(vao[0]);  
  
glGenBuffers(1, &vboID);  
glBindBuffer(GL_ARRAY_BUFFER, vboID);  
.....
```

3. 像往常一样用 VBO 发送第一个对象的数据 (参考 T02.pdf)。

4. 生成并绑定第二个 VAO。

```
glGenVertexArrays(1, &vao[1]);  
glBindVertexArray(vao[1]);  
  
glGenBuffers(1, &vboID);  
glBindBuffer(GL_ARRAY_BUFFER, vboID);
```

5. 发送第二个对象的数据...

作业 1

绘制多个对象

6. 在`paintGL`中, 设置适当的模型、视图、投影矩阵。

```
glBindVertexArray(vao[0]);  
glDrawArrays(GL_TRIANGLES, 0, 6);
```

7. 绑定第一个 VAO 并绘制:

8. 可能需要更改模型矩阵。
(可选地改变视图矩阵, 很少改变投影矩阵)

```
glBindVertexArray(vao[1]);  
glDrawElements(GL_TRIANGLES, 72, GL_UNSIGNED_INT, 0);
```

9. 绑定第二个 VAO 并绘制:

回想一下我们在 T02 中学到的内容。

大纲

- 作业 1 中的基本要求
- 如何渲染 3D 对象
- 绘制多个对象

关于作业 1 的提示

作业 1

提示

1. 基于三角形演示代码。
2. 添加视图和投影矩阵。 P28.
3. 添加一个简单的对象来绘制多个对象。 P29.
4. 继续其他操作（例如，索引、深度测试、转换、回调等）

基本原则：

- 从简单的事情开始。
- 除非你能得到最初正确的结果，否则不要尝试复杂的事情（即，一件一件地调试；要有耐心）。