《计算机图形学》实验 4 实验报告

学生	刘沅昊	学号	15331220
学院	数据科学与计算机学院	年级专业	16 级软件工程 (数字媒体技术)

实验内容：

Basic:

1. 画一个立方体(cube)：边长为4， 中心位置为(0, 0, 0)。分别启动和关闭深度测试 `glEnable(GL_DEPTH_TEST)`、`glDisable(GL_DEPTH_TEST)`，查看区别，并分析原因。
2. 平移(Translation)：使画好的cube沿着水平或垂直方向来回移动。
3. 旋转(Rotation)：使画好的cube沿着XoZ平面的x=z轴持续旋转。
4. 放缩(Scaling)：使画好的cube持续放大缩小。
5. 在GUI里添加菜单栏，可以选择各种变换。
6. 结合Shader谈谈对渲染管线的理解

Hint: 可以使用GLFW时间函数`glfwGetTime()`，或者`<math.h>`、`<time.h>`等获取不同的数值

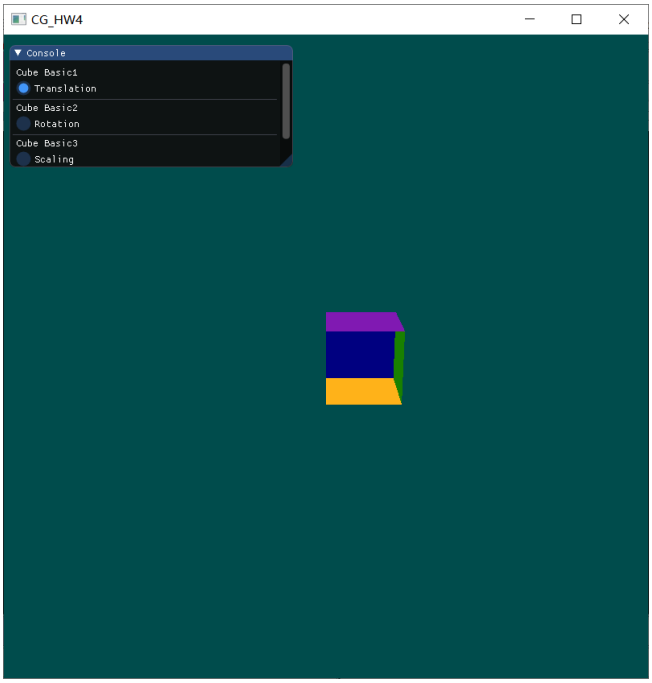
Bonus:

1. 将以上三种变换相结合，打开你们的脑洞，实现有创意的动画。比如：地球绕太阳转等。

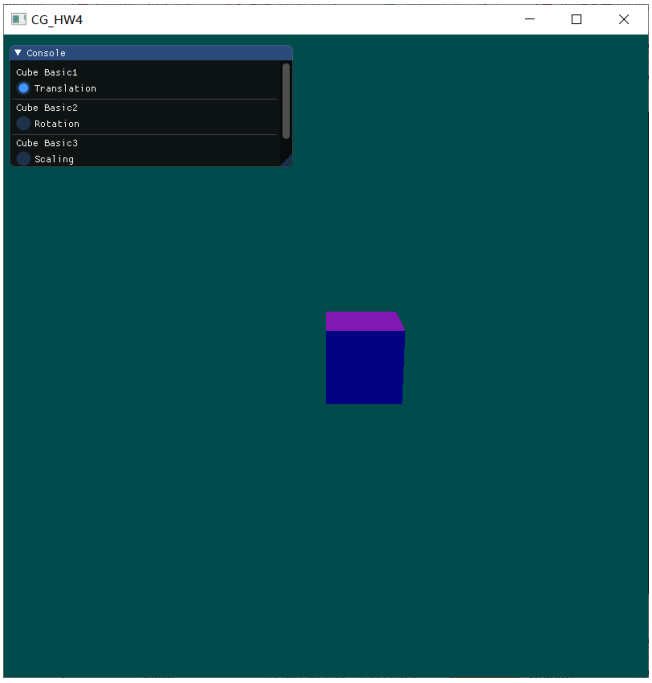
实验结果：

Basic：

1. 画一个立方体：边长为 4，中心位置为(0,0,0)。分别启动和关闭深度测试，查看区别，并分析原因。



图表 1 启动深度测试



图表 2 关闭深度检测

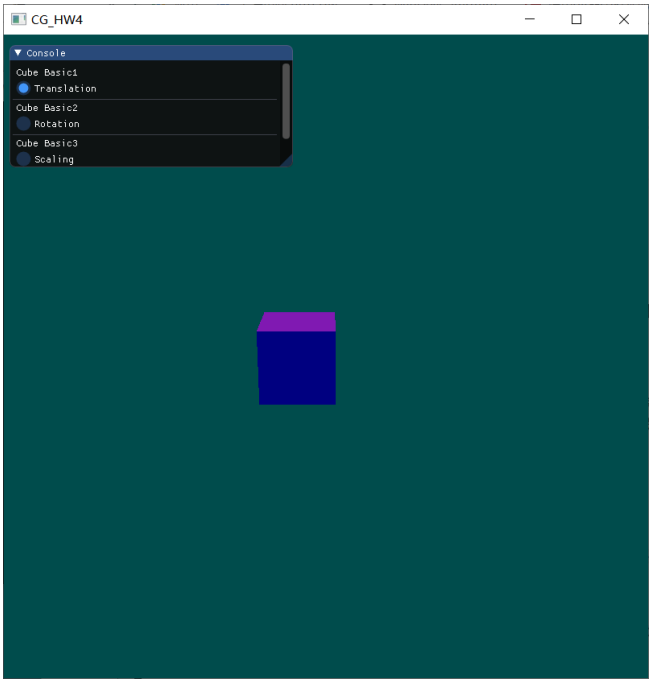
深度缓冲区原理就是把一个距离观察平面(近裁剪面)的深度值(或距离)与窗口中的每个像素相关联。首先，使用 glClear(GL_DEPTH_BUFFER_BIT),把所有像素的深度值设置为最大值(一般是远裁剪面)。然后，在场景中以任意次序绘制所有物体。硬件或者软件所执行的图形计算把每一个绘制表面转换为窗口上一些像素的集合，此时并不考虑是否被其他物体遮挡。其次，OpenGL 会计算这些表面和观察平面的距离。如果启用了深度缓冲区，在绘制每个像素之前，OpenGL 会把它的深度值和已经存储在这个像素的深度值进行比较。新像素深度值<原先像素深度值，则新像素值会取代原先的；反之，新像素值被遮挡，他颜色值和深度将被丢弃。为了启动深度缓冲区，必须先启动它，即 glEnable(GL_DEPTH_TEST)。每次绘制场

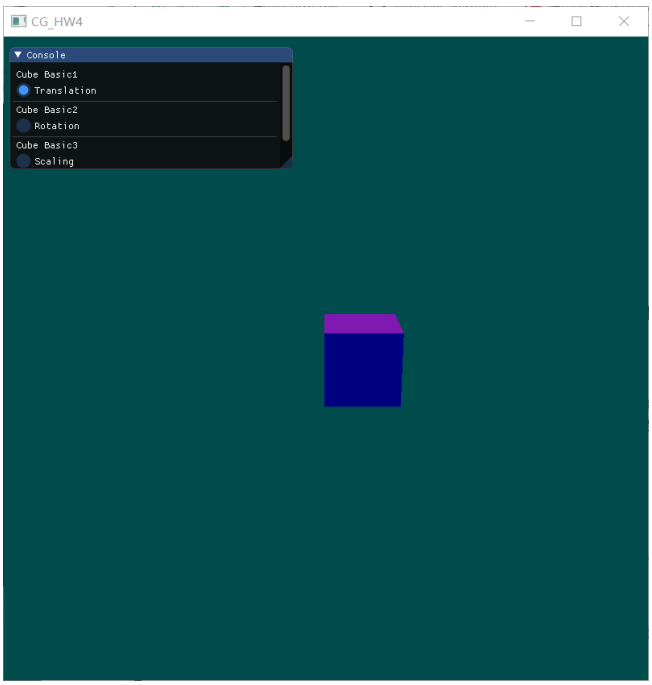
景之前，需要先清除深度缓冲区，即 glClear(GL_DEPTH_BUFFER_BIT)，然后以任意次序绘制场景中的物体。

2. 平移（Translation）：使画好的 cube 沿着水平或垂直方向来回移动

```
195             // 平移
196             model = glm::translate(model, glm::vec3(translate, 0.0f, 0.0f));
197             my_shader.setMat4("model", glm::value_ptr(model));
198             translate = translate + 0.005f * direction;
199             if (translate > 2.0f) direction = -1;
200             if (translate < -2.0f) direction = 1;
201             glBindVertexArray(VAO);
202             glDrawArrays(GL_TRIANGLES, 0, 36);
```

图表 3 通过设置平移因子控制立方体沿水平方向平移

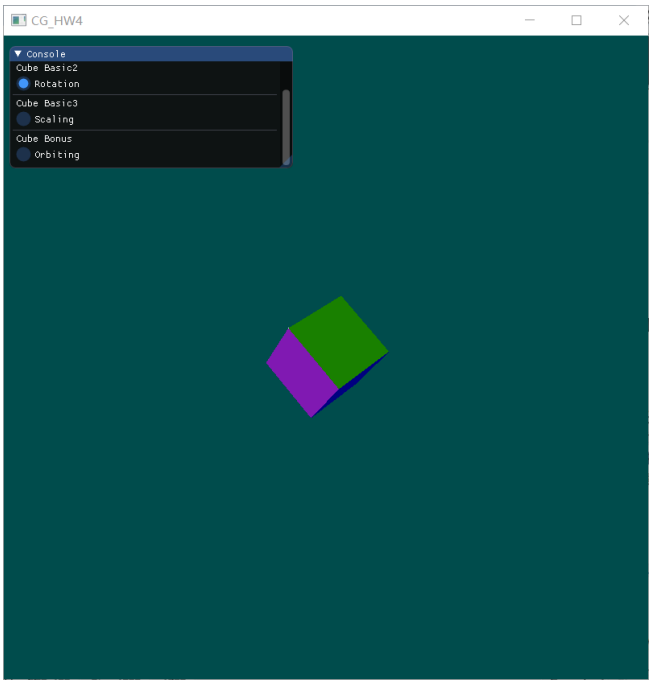


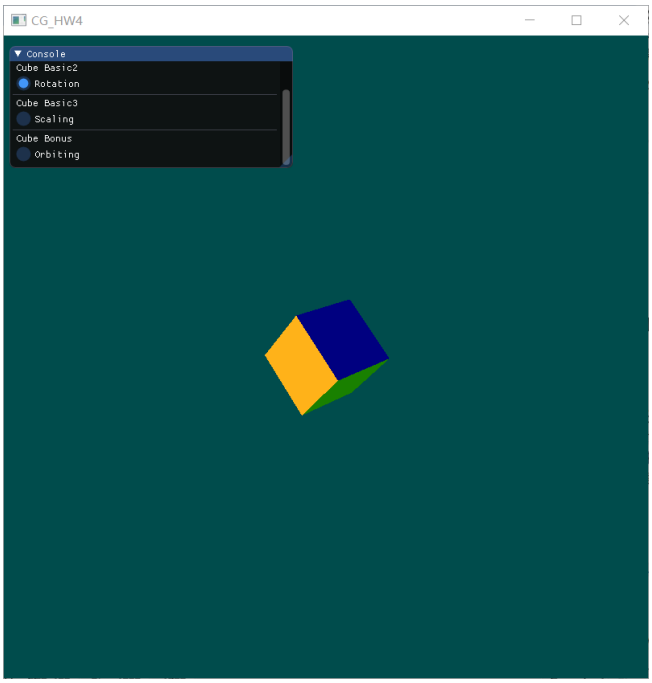


3. 旋转（Rotation）：使画好的 cube 沿着 XoZ 平面的 x=z 轴持续旋转

```
204         case 2:
205             // 旋转
206             model = glm::rotate(model, (float)glfwGetTime() * 10.0f, glm::vec3(1.0f, 0.0f, 1.0f));
207             my_shader.setMat4("model", glm::value_ptr(model));
208             glBindVertexArray(VAO);
209             glDrawArrays(GL_TRIANGLES, 0, 36);
```

图表 4 使立方体沿着 XoZ 平面的 x=z 轴持续旋转

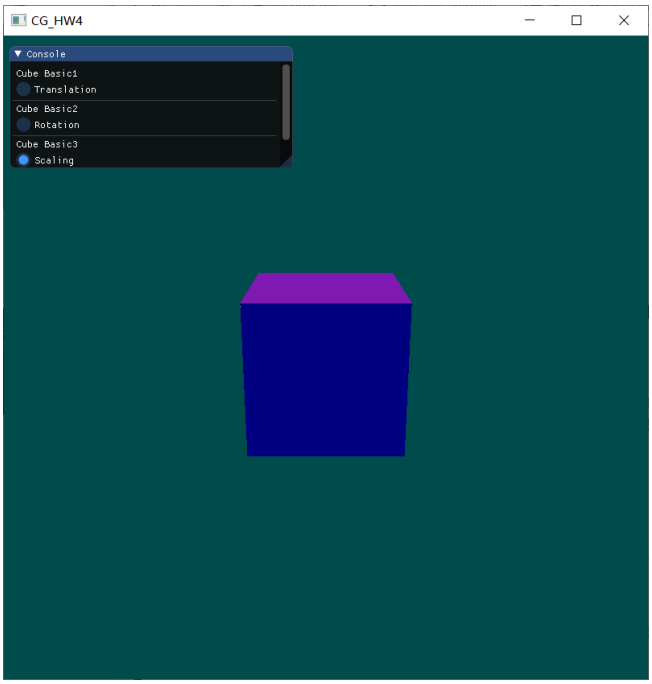


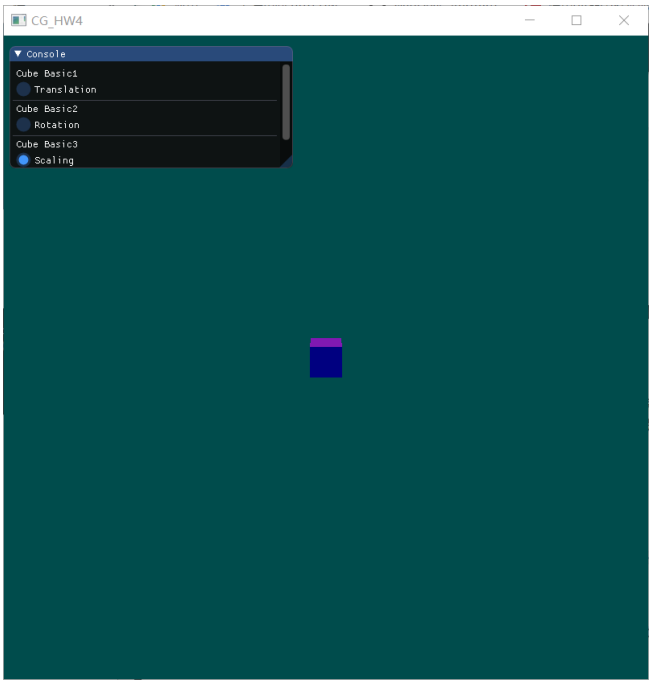


4. 放缩（Scaling）：使画好的 cube 持续放大缩小

```
212             // 缩放
213             model = glm::scale(model, glm::vec3(scale_factor, scale_factor, scale_factor));
214             my_shader.setMat4("model", glm::value_ptr(model));
215             scale_factor = scale_factor + 0.005f * direction;
216             if (scale_factor > 2.0f) direction = -1;
217             if (scale_factor < 0.3f) direction = 1;
218             glBindVertexArray(VAO);
219             glDrawArrays(GL_TRIANGLES, 0, 36);
220             break;
```

图表 5 设置缩放因子控制立方体的缩放





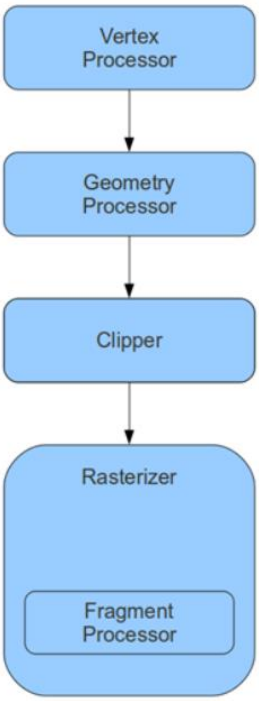
5. 在 GUI 里面添加菜单栏，可以选择各种变换

```
155        {
156            ImGui::Begin("Console");
157
158            ImGui::Text("Cube Basic1");
159            ImGui::RadioButton("Translation", &choose, 1);
160            ImGui::Separator();
161            ImGui::Text("Cube Basic2");
162            ImGui::RadioButton("Rotation", &choose, 2);
163            ImGui::Separator();
164            ImGui::Text("Cube Basic3");
165            ImGui::RadioButton("Scaling", &choose, 3);
166            ImGui::Separator();
167            ImGui::Text("Cube Bonus");
168            ImGui::RadioButton("Orbiting", &choose, 4);
169
170            ImGui::End();
171        }
```

6. 结合 Shader 谈谈对渲染管线的理解

Shader：中文翻译即着色器，是一种较为短小的程序片段，用于告诉图形硬件如何计算和输出图像，过去由汇编语言来编写，现在也可以使用高级语言来编写。一句话概括：Shader 是可编程图形管线的算法片段。它主要分为两类：Vertex Shader 和 Fragment Shader。

渲染管线：渲染管线也称为渲染流水线，是显示芯片内部处理图形信号相互独立的并行处理单元。一个流水线是一序列可以并行和按照固定顺序进行的阶段。就像一个在同一时间内，不同阶段不同的汽车一起制造的装配线，传统的图形硬件流水线以流水的方式处理大量的顶点、几何图元和片段。其流程图如下：



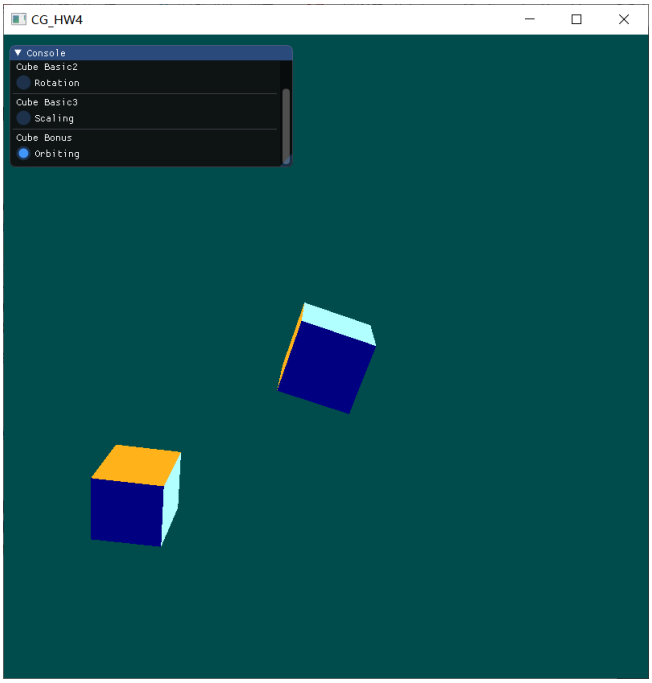
第一步是 vertex processor(vs)，第二步是 geometry processor(gs)，第三步是 clip 操作（还应该包括 PA, primitive assembly），最后是光栅化以及 fragment processor(ps)。

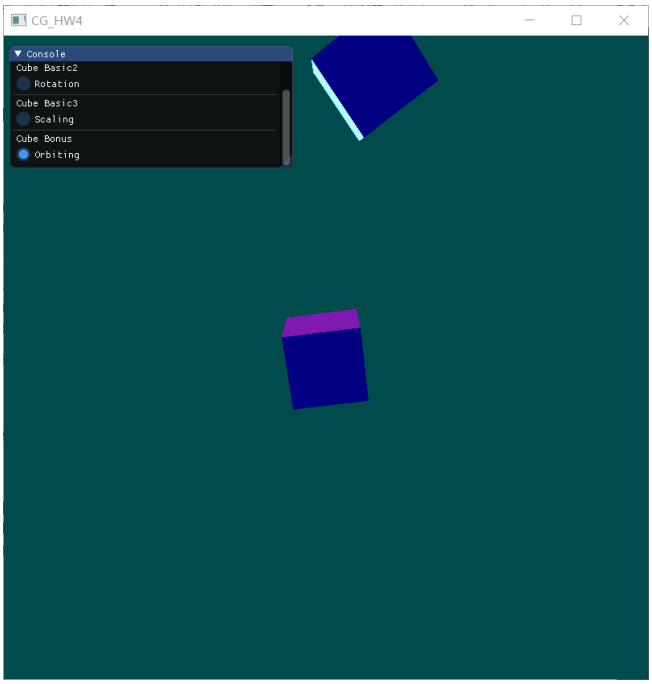
1. 顶点处理(vertex processor)，该阶段主要是对每个顶点执行 shader 操作，顶点数量在 draw 函数中指定。顶点 shader 中没有任何体元语义的内容，仅是针对顶点的操作，比如坐标空间变化，纹理坐标变化等等。每个顶点都必须执行顶点 shader，不能跳过该阶段，执行完顶

点 shader 后，顶点进入下一个阶段。
2. 几何处理(geometry processor)，在该阶段，顶点的邻接关系以及体元语义都被传入 shader，在几何 shader 中，不仅仅处理顶点本身，还要考虑很多附加的信息。几何 shader 甚至能改变输出体元语义类型，比如输入体元是一系列单独的点(point list 体元语义)，而输出体元则是三角形或者两个三角形组成的四边形等等， 甚至我们还能在几何 shader 中输入多个顶点，对于每个顶点输出不同语义的体元。
3. clipper 阶段，或者称作裁剪阶段，这是一个固定管线模块，它将用裁剪空间的 6 个面对体元进行裁剪操作，裁剪空间外的部分将会被移去，这时可能会产生新的顶点和新的三角形，用户也可以定义自己的裁剪平面进行裁剪操作，裁剪后的三角形将会被传到光栅化阶段。[注：在 clipper 之前，会有 PA 阶段，就是顶点 shader 或几何 shader 处理过的顶点被重新装配成三角形]
4. 光栅化阶段和片元操作阶段，clipper 后的三角形会先被光栅化，产生很多的 fragment(片元)，接着执行片元 shader，在片元 shader 中，可能会装入纹理，从而产生最终的像素颜色。（fragment 可以理解为带 sample、深度信息的像素）

Bonus：

1. 将以上三种变换相结合，打开你们的脑洞，实现有创意的动画，比如：地球绕太阳转等。





地球绕太阳转，即太阳的自转（旋转），地球的自转（旋转）和围绕太阳的公转（平移）。将上面的基本操作结合就做好了。具体动画请查看 gif。