计算机图形学作业报告

廖蕾 16340135

- 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。

  在这次的作业中需要用到glm库，需要从github上下载他的源码，链接地址请点击这里。下载之后将其中的glm文件夹添加到项目include中即可。
  在以下的代码中，用到的着色器GLSL有： 定义`vertexShaderSource`如下：

```
const char *vertexShaderSource = "#version 330 core\n"
"layout(location = 0) in vec3 aPos;\n"
"layout(location = 1) in vec3 color;\n"
"out vec3 ourColor;\n"
"uniform mat4 model;\n"
"uniform mat4 view;\n"
"uniform mat4 projection;\n"
"void main()\n"
"{\n"
"       gl_Position = projection * view * model * vec4(aPos, 1.0);\n"
"       ourColor = color;\n"
"}\n";
```

  定义`fragmentShaderSource`如下：

```
const char *fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor; \n"
"in vec3 ourColor; \n"
"void main()\n"
"{\n"
"       FragColor = vec4(ourColor, 1.0); \n"
"}\n";
```

  在这里坐标系的使用用到了三个`glm::mat4`变量：

  - model：将局部坐标系转换成世界坐标系
  - view：将世界坐标系转换成摄像机坐标系
  - projection：将摄像机坐标系进行裁剪，将摄像机坐标系转换成屏幕坐标系

  在画立方体的时候，用到立方体的六个面顶点信息，每个面两个三角形，每个三角形3个顶点，一共需要定义36个顶点信息：

```
float vertices[216] = {
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
```

```
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.0f,

        -0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,

        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, 1.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,

         0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,

        -0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 1.0f,
        -0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 1.0f,
        -0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 1.0f,

        -0.5f,  0.5f, -0.5f,  1.0f, 1.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 1.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  1.0f, 1.0f, 1.0f,
        -0.5f,  0.5f, -0.5f,  1.0f, 1.0f, 1.0f
};
```

在渲染的过程中，先得到这36个顶点和颜色的信息，进行渲染：

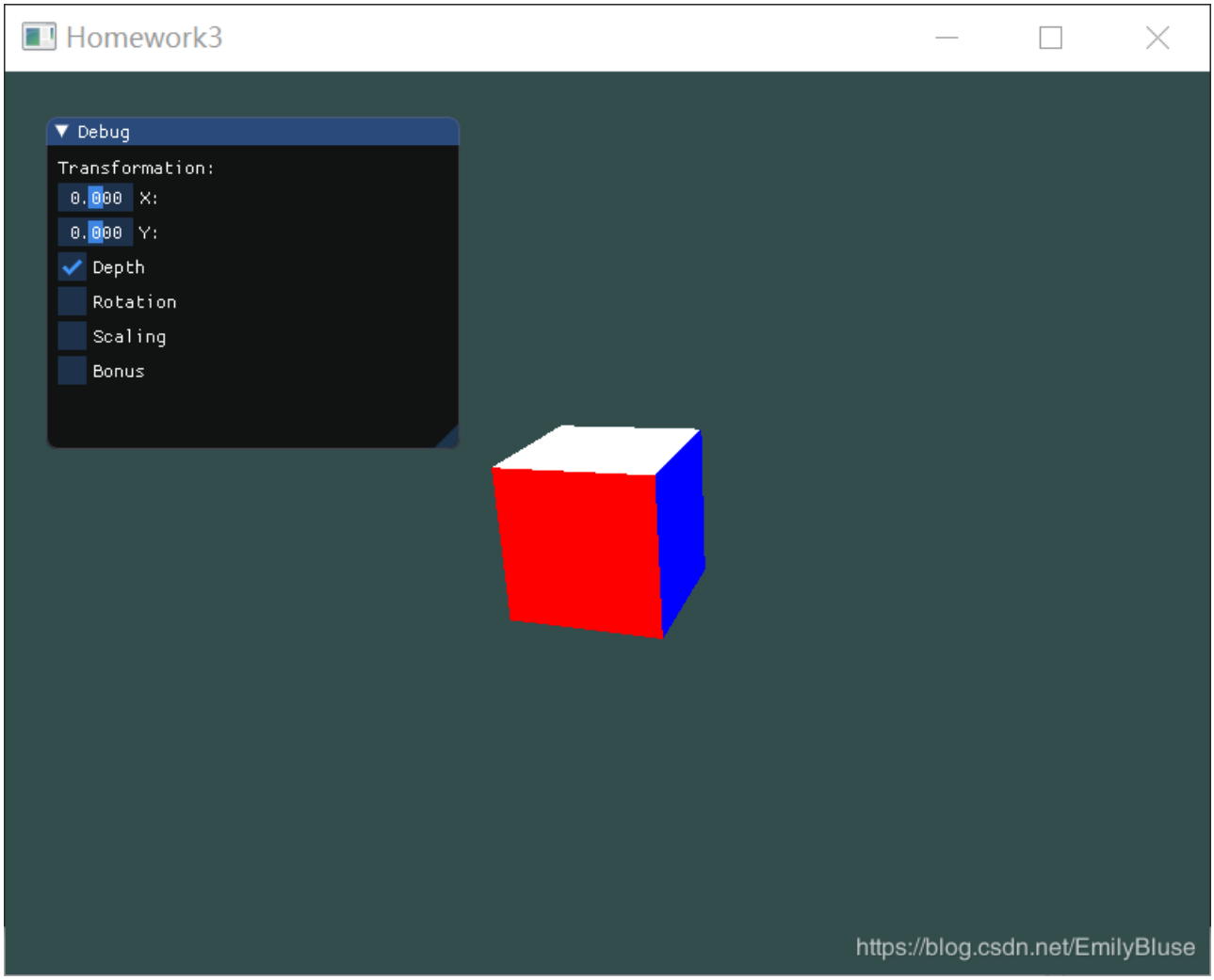
```
glBindBuffer(GL_ARRAY_BUFFER, VAO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

// position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
(void*)0);
glEnableVertexAttribArray(0);
// texture coord attribute
```

```
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3
* sizeof(float)));
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// activate shader
glUseProgram(shaderProgram);
```

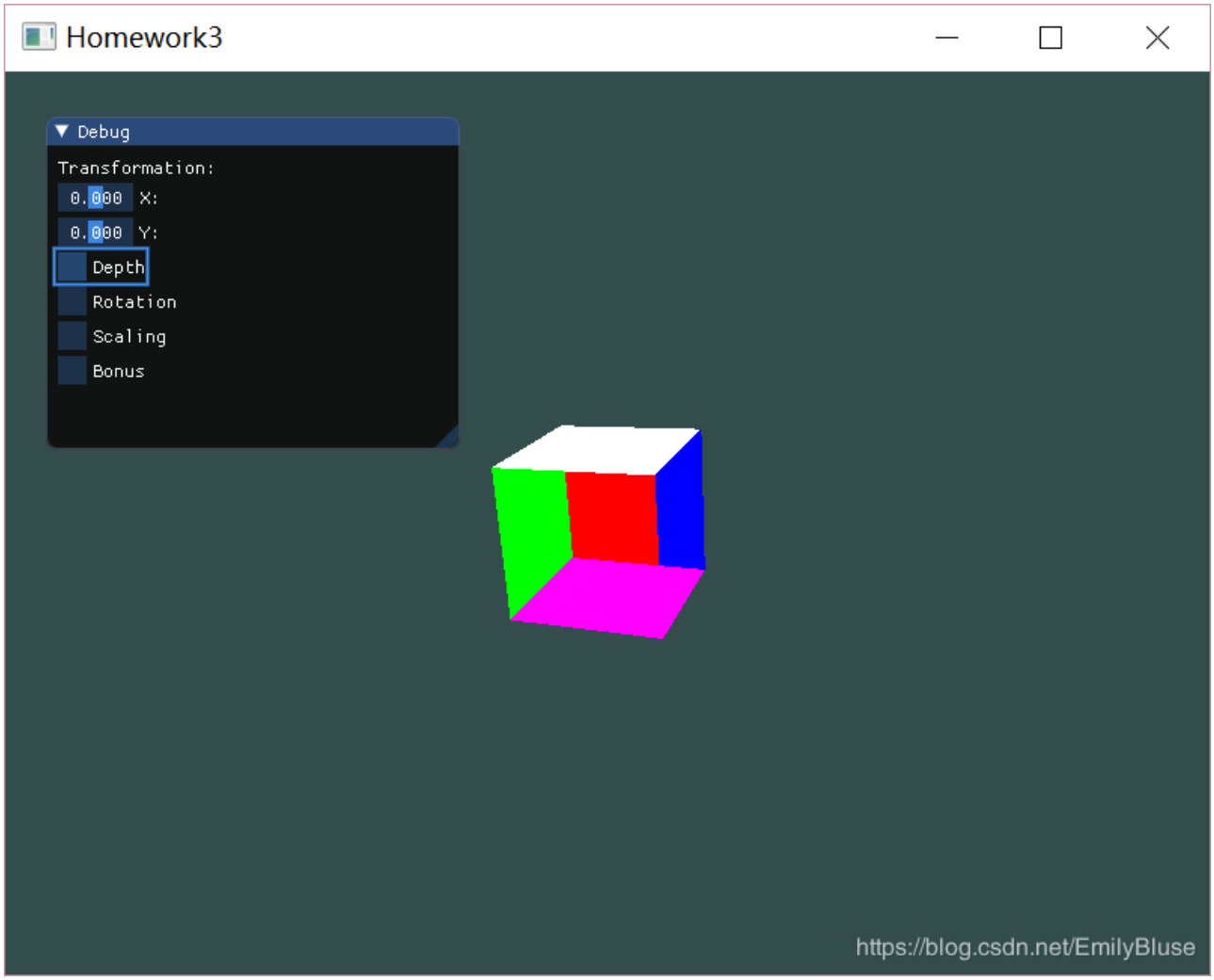
之后再根据定义的model、view和projection三个变量，得到3d的立方体，这里是利用了透视投影：

```
glm::mat4 model = glm::mat4(1.0f); // make sure to initialize matrix to
identity matrix first
glm::mat4 view = glm::mat4(1.0f);
glm::mat4 projection = glm::mat4(1.0f);

view = glm::translate(view, glm::vec3(0.0f, 0.0f, -5.0f));
view = glm::rotate(view, glm::radians(30.0f), glm::vec3(1.0f, -1.0f, 0.0f));
projection = glm::perspective(glm::radians(60.0f), (float)WINDOW_WIDTH /
(float)WINDOW_HEIGHT, 0.1f, 100.0f);

string name = "view";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(view));
name = "projection";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(projection));

model = glm::translate(model, glm::vec3(translatio_x, translatio_y, 0.0f));
model = glm::scale(model, glm::vec3(scale_x, scale_y, scale_z));
name = "model";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

这样通过渲染可以得到3d的立方体（默认开启深度测试）：



当关闭深度测试之后得到结果如下：

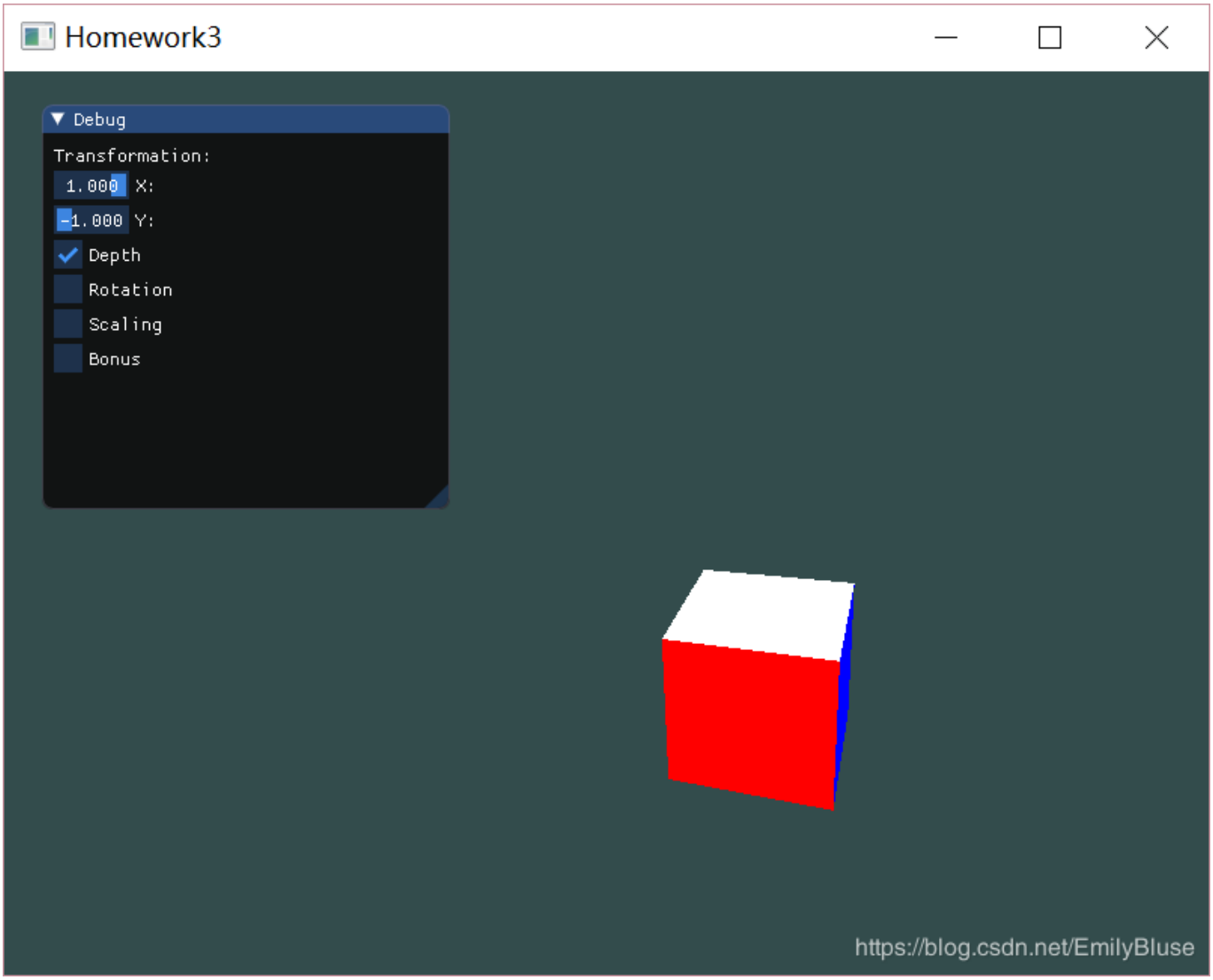


- 平移(Translation)：使画好的cube沿着水平或垂直方向来回移动。 在构造正方体的时候，引入了两个参数`translation_x`和`translation_y`，将model用glm中`translate`函数进行变换：

```
float translation_x = 0.0;
float translation_y = 0.0;
model = glm::translate(model, glm::vec3(translation_x, translation_y,
0.0f));
```

model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(1.0f, 0.0f, 1.0f)); name = "model";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36); ```
再用相同的方法去渲染立方体就好了，得到结果如下（动图麻烦TA打开同文件夹下的transformation.gif查

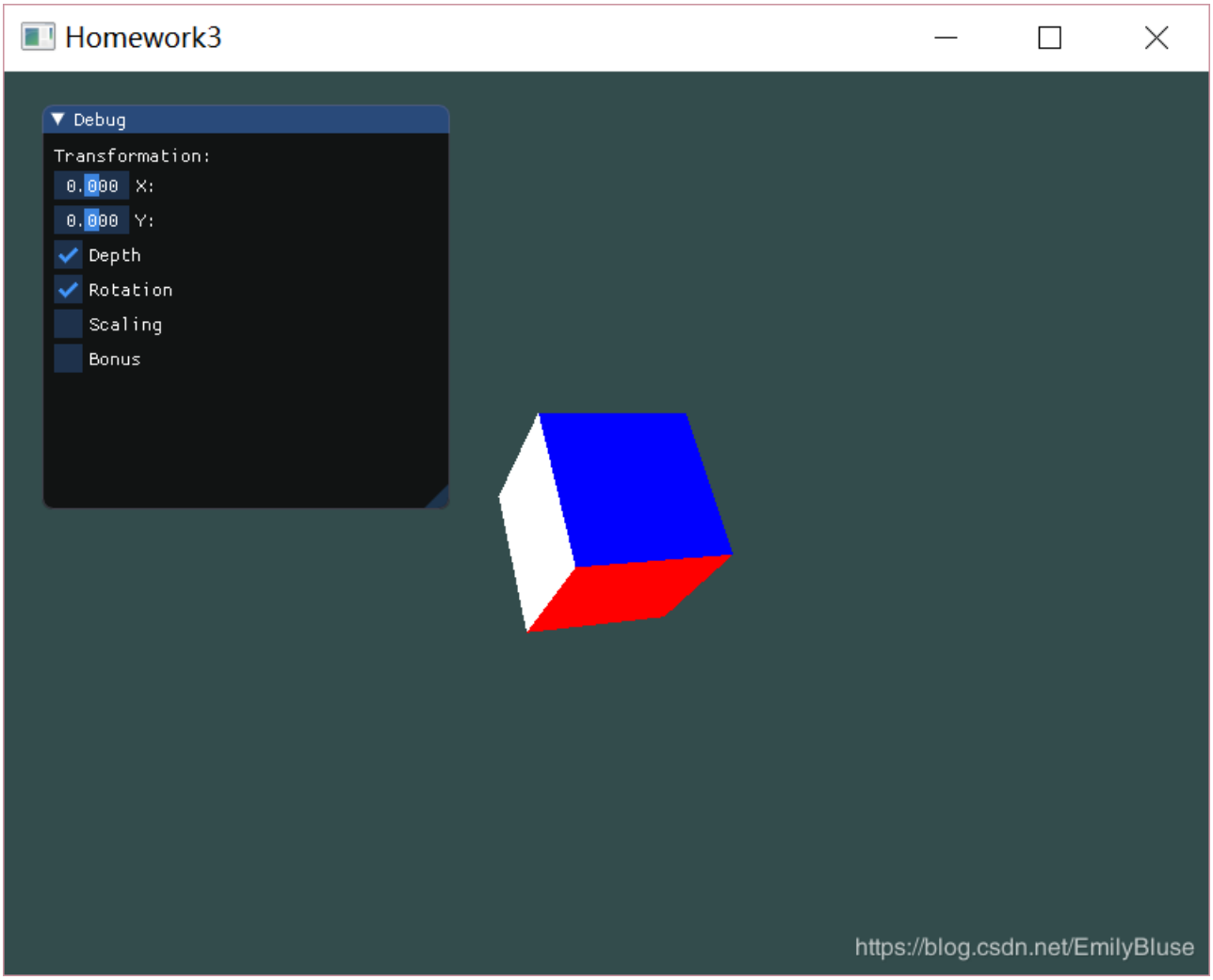
看）：



- 旋转(Rotation)：使画好的cube沿着XoZ平面的x=z轴持续旋转。
  这里用到的变换方法和平移差不多，利用`glm::rotation`函数，对model进行变化，再用`glfwGetTime`这个函数获得时间，从而可以形成自己转动的效果，代码如下：

```
model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(1.0f, 0.0f,
1.0f));
name = "model";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

得到结果如下（动图麻烦TA打开同文件夹下的transformation.gif查看）：



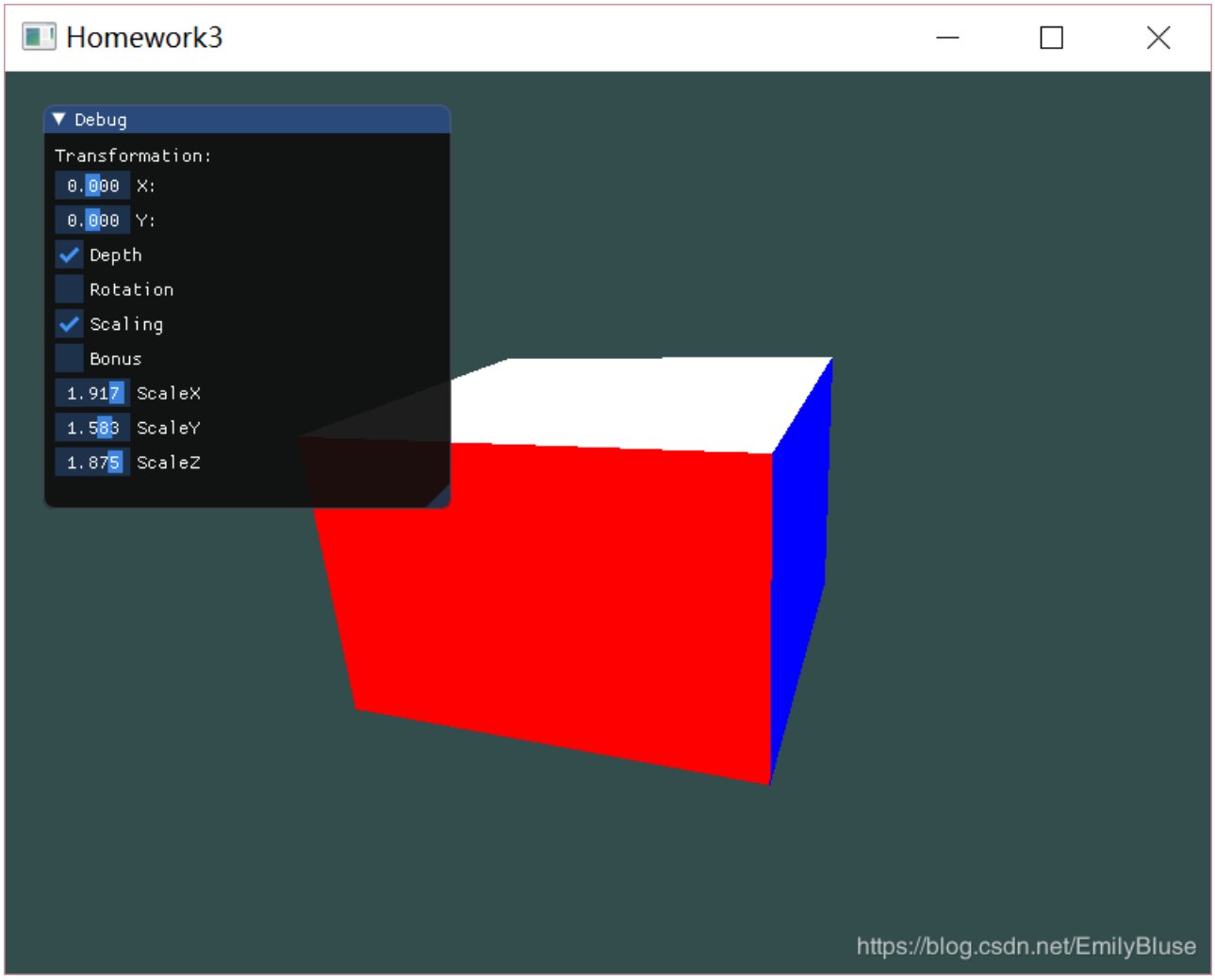
- 放缩(Scaling)：使画好的cube持续放大缩小。
  放缩的方法和上面的相似，利用glm::scale函数对model进行操作，需要引入三个变量去调节x、y、z三轴的放缩：

```
float scale_x = 1.0;
float scale_y = 1.0;
float scale_z = 1.0;

model = glm::scale(model, (abs(sin((float)glfwGetTime())) + 0.1f) *
glm::vec3(scale_x, scale_y, scale_z));
```

然后再进行渲染即可：

```
model = glm::translate(model, glm::vec3(translation_x, translation_y,
0.0f));
model = glm::scale(model, glm::vec3(scale_x, scale_y, scale_z));
name = "model";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

得到结果如下（动图麻烦TA打开同文件夹下的transformation.gif查看）：



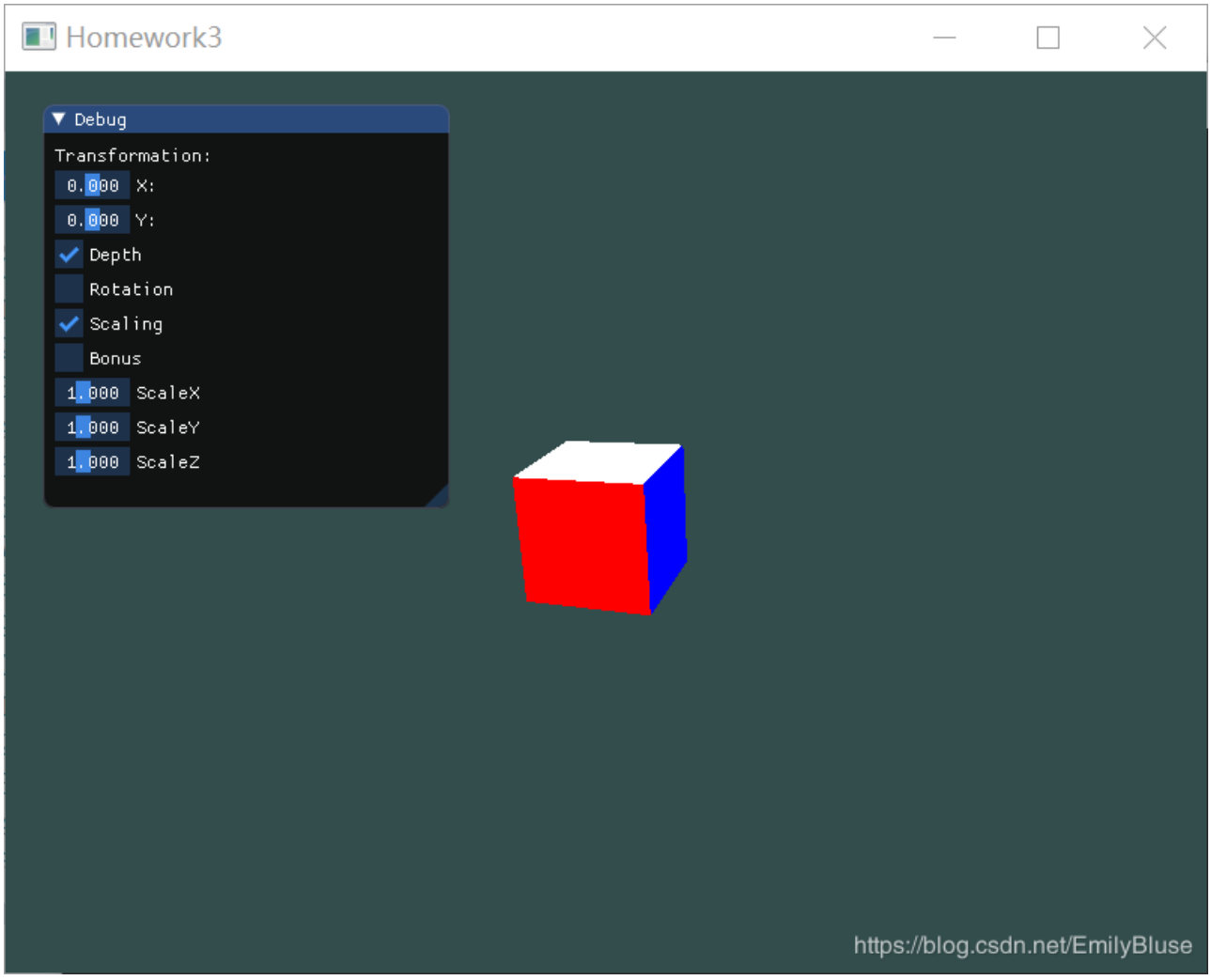
- 在GUI里添加菜单栏，可以选择各种变换。
  添加GUI在上面几个步骤中有涉及，主要是添加一些能手动选择值的变量，然后加入到变换当中：

```
bool depth = true;
bool rotation = false;
bool scaling = false;
bool bonus = false;

float translation_x = 0.0;
float translation_y = 0.0;

float scale_x = 1.0;
float scale_y = 1.0;
float scale_z = 1.0;

//开始循环
while (!glfwWindowShouldClose(window))
{
        // input
        // -----
```

```
        processInput(window);
        glfwPollEvents();
        // render
        // ------
        ImGui_ImplGlfwGL3_NewFrame();
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // also clear
the depth buffer now

        {
                ImGui::Text("Transformation: ");
                ImGui::PushItemWidth(50);
                ImGui::SliderFloat("X: ", &translation_x, -1.0f, 1.0f);
                ImGui::SliderFloat("Y: ", &translation_y, -1.0f, 1.0f);
                ImGui::Checkbox("Depth", &depth);
                ImGui::Checkbox("Rotation", &rotation);
                ImGui::Checkbox("Scaling", &scaling);
                ImGui::Checkbox("Bonus", &bonus);

        //渲染图形
        ...
                // create transformations
                //计算变换
                ...
                //选择展示内容
                if (!bonus) {
                    //不是bonus内容

                        if (rotation) {
                                model = glm::rotate(model,
(float)glfwGetTime(), glm::vec3(1.0f, 0.0f, 1.0f));
                        }
                        if (scaling) {
                                ImGui::SliderFloat("ScaleX", &scale_x, 0.5f,
2.0f);
                                ImGui::SliderFloat("ScaleY", &scale_y, 0.5f,
2.0f);
                                ImGui::SliderFloat("ScaleZ", &scale_z, 0.5f,
2.0f);
                                model = glm::scale(model,
(abs(sin((float)glfwGetTime())) + 0.1f) * glm::vec3(scale_x, scale_y,
scale_z));
                        }
                        ...

                }
                else
                {
                        //Bonus内容
                        ...
                }
                if (depth) {
                        glEnable(GL_DEPTH_TEST);
                }
```

```
                else {
                        glDisable(GL_DEPTH_TEST);
                }
        }
        glUseProgram(shaderProgram);
        ImGui::Render();
        ImGui_ImplGlfwGL3_RenderDrawData(ImGui::GetDrawData());
        glfwSwapBuffers(window);
        glfwPollEvents();
}
```

最后得到的效果可以看展示视频，以及上面几个图片，这里用一张图片展示GUI效果：



- 结合Shader谈谈对渲染管线的理解 渲染管线包括以下几个步骤：
    - 顶点处理：输入顶点（坐标，颜色，状态，纹理坐标，发现），计算输出在本地坐标系中的结果，然后再将本地坐标系转换为世界坐标，再将世界坐标转换到投影坐标系中。
    - 几何处理器：将每个点连接成面，面和面组装成模型，将摄影机看不到的面从模型中剔除，不参与计算。摄像机、包括我们人所看到的范围实际上都是一个椎体，人的眼睛为出发点，是一个顶点，眼睛会有一个角度，视角能看到一个最远的地方。因为人能看到的区域是一个椎体，所以3D引擎里面拿了一个摄像机的椎体视角范围来模拟人看到的范围。有可能某些模型的面比人看到的范围还大，那么就会在边缘处就会做一些裁剪，这就是面的截取 。

- 光栅化：将以向量为基本的面转换成一个个点阵形式的像素集合 模型相当于是一个矢量（既有大小又有方向的量）的内容，模型上由某一个点到某一个点之间，它是一个距离，但是实际上，用户在设备上最终输出的都是以像素点来输出的，会把一个平面拆分成一个一个的像素点，每一个像素点在后面会做单独的处理，如果一个像素点区域被两个模型所占，如果一个模型为黄色，一个为红色，那么最终这个像素只能以一种颜色进行输出，光栅化就会做这样的处理，让一个像素点上只会出现一种颜色。
- 片段处理器：通过输入插值得到的片段信息，加上纹理、雾化等操作，输出片段的颜色值和深度值。

在这样的过程中，shader里的顶点着色器主要作用于第一阶段，片段着色器用于第四阶段。

Bonus

- 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。
我这里用平移+缩放+旋转，得到新的一个正方体，围着之前的正方体旋转，同时之前的正方体也在自转，模仿地球绕太阳转，同时两个还能自转，代码实现如下：

```
float radius = 2.0f;
float camX = sin(glfwGetTime()) * radius;
float camZ = cos(glfwGetTime()) * radius;

model = glm::translate(model, glm::vec3(camX, 0.0f, camZ));
model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(0.0f, 1.0f,
0.0f));
model = glm::scale(model, 0.5f * glm::vec3(1.0f, 1.0f, 1.0f));

name = "model";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);

model = glm::mat4(1.0f);
model = glm::rotate(model, (float)glfwGetTime()*0.5f, glm::vec3(0.0f, 1.0f,
0.0f));

name = "model";
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1,
GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

得到结果如下（动图麻烦TA打开同文件夹下的transformation.gif查看）：