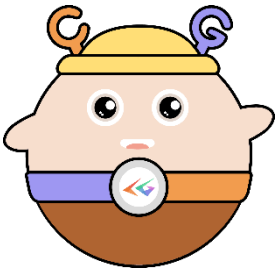
华中科技大学



计算机图形学课程

实验：立方体旋转

目录

绘制旋转立方体

在正式搭建环境之前，我们先来介绍一下读完下面的部分你会了解些什么。

- 绘制出旋转立方体需要的新知识
- 认识一些 OpenGL 的新功能

接下来，我们来介绍一下绘制旋转立方体。绘制效果如下：

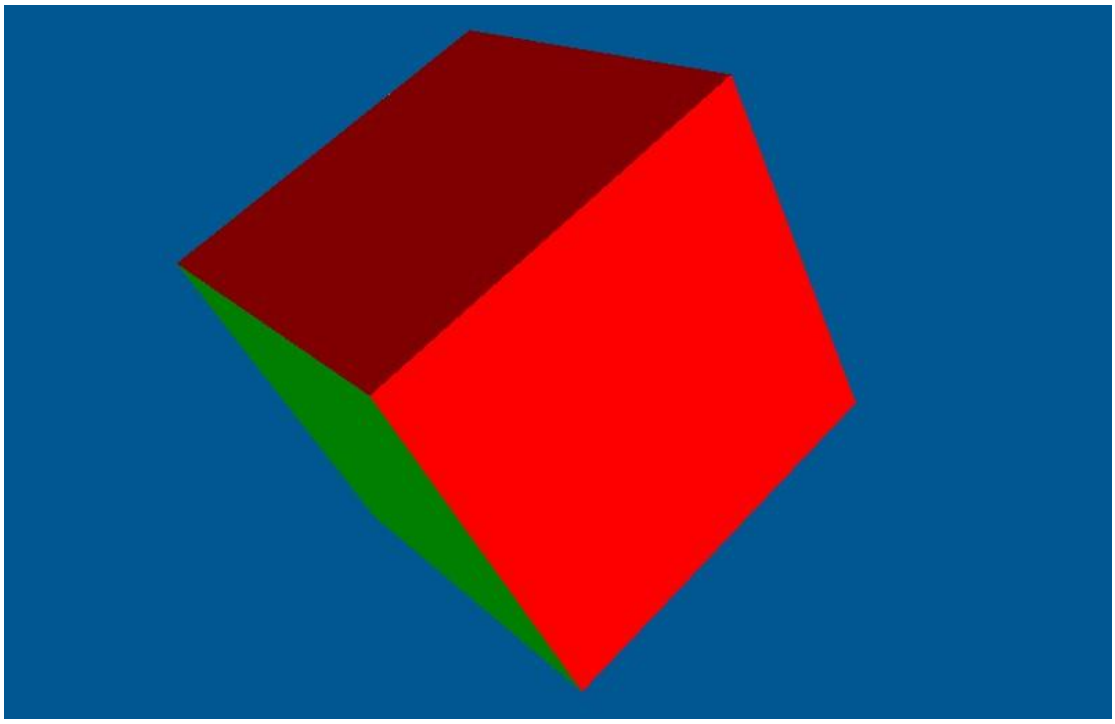


图 1 绘制旋转立方体的效果

1 Z-缓冲

Z-缓存(Z-Buffer):这是一项处理 3D 物体深度信息的技术，它对不同物体和同一物体不同部分的当前 Z 坐标进行纪录，也就是在 3D 环境中，每个像素会利用一组数据资料来定义像素在显示时的纵深度（即 Z 轴坐标值）。在进行着色时，对那些在其他物体背后的结构进行消隐，使它们不被显示出来。

在 opengl 中，坐标映射到屏幕屏幕空间后，其 z 值即最终 z-缓冲的值，只

需记录每个屏幕像素点的 z 值，并与当前绘制的片元的 z 值进行比对，即可判断物体是否遮挡或被遮挡。实际应用中，我们只需要开启深度测试即可。

```
glEnable(GL_DEPTH_TEST);
```

2 GLM 函数库

GLM 是 OpenGL Mathematics 的缩写，它是一个只有头文件的库，也就是说我们只需包含对应的头文件就行了，不用链接和编译。GLM 可以在它们的网站上下载。把头文件的根目录复制到你的 includes 文件夹，就可以使用这个库了。使用这个库，好处就在于我们只需要输入特定参数，就可以生成我们需要的矩阵。

3 PVM 矩阵

PVM 矩阵即 P:projection;V:view;M:model。其中，model 矩阵对应从局部坐标系到世界坐标系的变换，模型矩阵是一种转换矩阵，它能通过对对象进行平移、缩放、旋转来将它置于它本应该在的位置或方向。

view 矩阵对应从世界坐标系到观察坐标系的变换，观察坐标系就是从摄像机的角度观察到的坐标系。而这通常是由一系列的平移和旋转的组合来平移和旋转场景从而使得特定的对象被转换到摄像机前面。

projection 矩阵对应从观察坐标系到剪裁空间的变换，它指定了坐标的范围，例如，每个维度都是从-1000 到 1000。投影矩阵接着会将在它指定的范围内的坐标转换到标准化设备坐标系中(-1.0，1.0)。所有在在范围(-1.0,1.0)外的坐标都不会被绘制出来并且会被裁剪。由投影矩阵创建的观察区域被称为平截头体，且每个出现在平截头体范围内的坐标都会最终出现在用户的屏幕上。将一定范围内的坐标转化到标准化设备坐标系的过程(而且它很容易被映射到 2D 观察空间坐标)被称之为投影，因为使用投影矩阵能将 3 维坐标投影到很容易映射的 2D 标准化设备坐标系中。

一旦所有顶点被转换到裁剪空间，最终的操作——透视划分将会执行，在这个过程中我们将位置向量的 x，y，z 分量分别除以向量的齐次 w 分量；透视划分是将 4 维裁剪空间坐标转换为 3 维标准化设备坐标。这一步会在每一个顶点着色器运行的最后被自动执行。 在这一阶段之后，坐标经过转换的结果将会被映射到屏幕空间(由 glViewport 设置)且被转换成片段。

4 PVM 矩阵的使用

那么如何在实践中使用 pvm 矩阵呢，首先我们需要引入 GLM 函数库的头文件，我们所需要用到的功能都在这三个头文件中。

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

然后我们需要设置 view 矩阵的相关参数。

```
glm::vec3 camera_position = glm::vec3(0.0f, 0.0f, 3.0f);    // 摄像机位置
glm::vec3 camera_front = glm::vec3(0.0f, 0.0f, -1.0f);      // 摄像机方向
glm::vec3 camera_up = glm::vec3(0.0f, 1.0f, 0.0f);          // 摄像机上向量
```

然后我们需要设置 projection 矩阵的视野 fov：

```
float fov = 45.0f;
```

进入主循环之后我们先计算 model 矩阵，首先我们需要创建一个 model 矩阵，然后 glm：translate 函数是进行平移变换的矩阵，将物体平移（0.0,0.0,0.0）位置，然后进行旋转，第二个参数为旋转的角度，第三个参数为旋转轴。最后进行缩放 glm：：vec3 变量的三个值分别代表 x，y，z 方向的缩放比例。

```
glm::mat4 model(1);//model 矩阵，局部坐标变换至世界坐标
model = glm::translate(model, glm::vec3(0.0,0.0,0.0));
model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(0.5f, 1.0f,
0.0f));
model = glm::scale(model, glm::vec3(1.0f,1.0f,1.0f));
```

为了得到我们需要的 view 矩阵，我们需要先创建一个 mat4 view 矩阵，通过 glm 中的 rotate 函数去计算，第一个是相机的位置，第二个参数是相机所正对的目标的坐标，这里使用 camera_position+camera_front(相机的方向)，进行向量的加法之后可以获得相机正对的坐标，第三个参数是相机的上向量。

```
glm::mat4 view(1);//view 矩阵，世界坐标变换至观察坐标系
view = glm::lookAt(camera_position, camera_position + camera_front,
camera_up);
```

在 GLM 中可以这样创建一个透视投影矩阵，它的第一个参数定义了 fov 的值，它表示的是视野，并且设置了观察空间的大小。对于一个真实的观察效果，它的值经常设置为 45.0，但想要看到更多结果你可以设置一个更大的值。第二个参数设置了宽高比，由视口的高除以宽。第三和第四个参数设置了平截头体的近和远平面。我们经常设置近距离为 0.1 而远距离设为 100.0。所有在近平面和远平面的顶点且处于平截头体内的顶点都会被渲染。

```
glm::mat4 projection(1);//projection 矩阵，投影矩阵
projection = glm::perspective(glm::radians(fov), (float)screen_width /
screen_height, 0.1f, 100.0f);
```

我们之前的操作都只是得到 model，view 和 projection 矩阵，但是不能忘记将这三个矩阵传入到着色器，否则着色器是没有办法使用的。

glGetUniformLocation 可以获得某个着色器中参数的位置，第一个参数为着色器 id，第二个参数为该参数的名称。

```
int model_location = glGetUniformLocation(shader.ID, "model");
// 获取着色器内某个参数的位置
```

glUniformMatrix4fv 是向指定位置传入一个 4X4 的矩阵值。通过这两个函数我们将 pvm 矩阵传入着色器中。

```
glUniformMatrix4fv(model_location, 1, GL_FALSE,
glm::value_ptr(model));// 写入参数值
```

最后，在着色器中，对于矩阵的乘法，由于不符合乘法交换律，所以我们应当注意相乘顺序。

```
gl_Position = projection * view * model * vec4(aPos, 1.0);
```

整个绘制旋转立方体的程序的新知识点介绍就到此为止，效果如下：

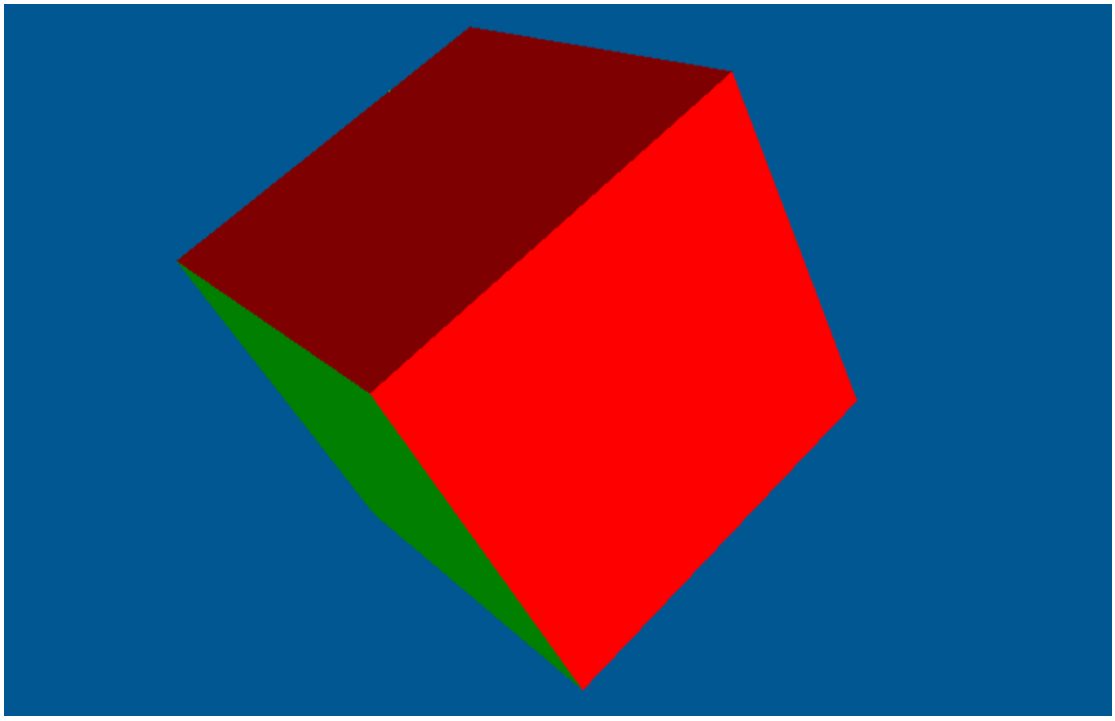


图 2 效果图