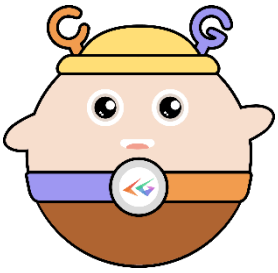
華中科技大學



计算机图形学课程

实验：球的绘制

目录

封装 Shader

在正式开始介绍球模型的绘制之前，我们需要将和着色器有关的操作进行封装，使其成为一个工具类，这里不介绍具体实现，仅介绍封装的必要性和使用方法。

- 为什么要封装 Shader
- 如何使用

1 为什么要封装 Shader

封装后，在应对存在多个着色器程序的渲染流程时，可以更方便使用不同的着色器程序，同时也可以降低代码冗余

2 如何使用

如下，传入参数分别为顶点着色器和片元着色器的路径，在封装了 Shader 类后，我们就可以通过一行代码去创建一个新的着色器对象：

```
Shader shader("res/shader/task3.vs", "res/shader/task3.fs");
```

假如我们在绘制时需要切换到某个着色器并且使用它，我们仅需要一行代码：

```
shader.Use();
```

假如我们需要向着色器中传入一种类型的值，我们也仅需要一行代码去解决它（name 为着色器中的名称，value 为你希望设置的值）：

```
SetFloat(string &name, float value)
```

绘制球模型

在正式绘制球模型之前，我们先来介绍一下读完下面的部分你会了解些什么。

- 球面顶点遍历
- 构造三角形图元
- 开启线框模式
- 开启面剔除

1 球面顶点遍历

我们要构造绘制球面的顶点数组，首先需要知道如何遍历球面的所有顶点。

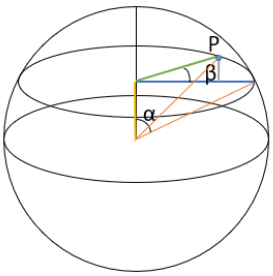


图 1-1 球体

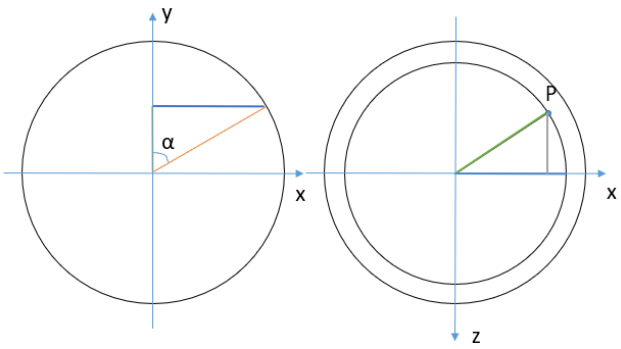


图 1-2 球体正视图和俯视图

在这幅图中，我们假设 P 是球体上的任意一点，如何计算这一点的坐标呢？我们可以通过如图所示的α 角和β 角和半径 R 来表示这个点的坐标。

如图∠α 和球体半径（橙色细线）可以计算出球体上 P 点的 Y 轴坐标即 R*cos∠α ，通过∠α 也可获得如图所示蓝线长度，即 R* sin∠α ，蓝线长度等于绿线长度，然后可以根据绿线长度继续计算出 P 点 z 坐标即 R* sin∠α *sin∠β 和 P 点 x 坐标即 R* sin∠α *cos∠β 。

那么当我们知道了球体上任意一个点都可以由α 和β 以及半径表示出来。

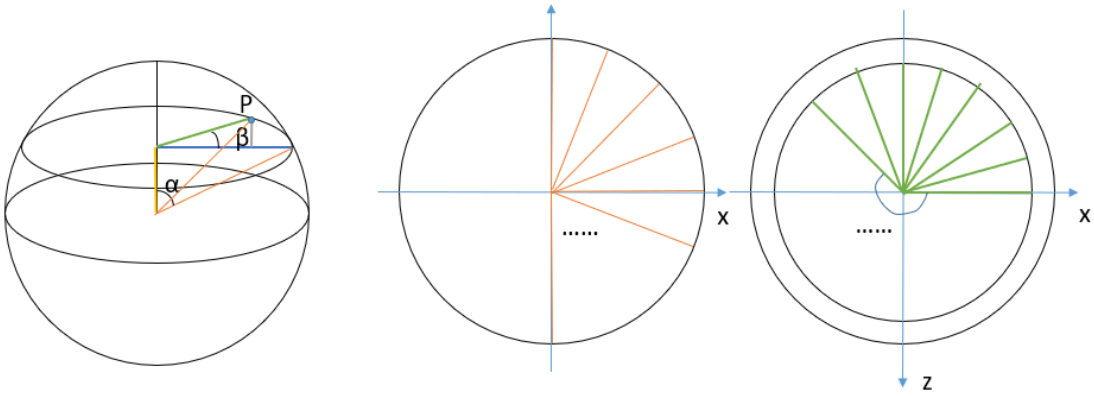


图 1-3 球体微分

我们将 0~π 的α 角分为 Y_SEGMENTS 份，将 0~2π 的β 角分成 X_SEGMENTS 份，则球上一个点的α 和β 角可以表示为π / Y_SEGMENTS *n 和 2π / X_SEGMENTS *m（0<n< Y_SEGMENTS,0<m<X _SEGMENTS ）

这样，通过遍历每一种α 和β 的组合，可以得到球体表面上离散点的集合

这一部分就是生成顶点的代码实现，Y_SEGMENTS 和 X_SEGMENTS 表示将α 和β 分割了多少份，y 和 x 表示分别是第几份，以此进行遍历，xSegment*2.0f*PI 即β 角，ySegment*PI 即α 角。

```
// 生成球的顶点
for (int y = 0; y <= Y_SEGMENTS; y++)
{
        for (int x = 0; x <= X_SEGMENTS; x++)
        {
```

```
                float xSegment = (float)x / (float)X_SEGMENTS;
                float ySegment = (float)y / (float)Y_SEGMENTS;
                float xPos = std::cos(xSegment * 2.0f * PI) *
std::sin(ySegment * PI);
                float yPos = std::cos(ySegment * PI);
                float zPos = std::sin(xSegment * 2.0f * PI) *
std::sin(ySegment * PI);



                sphereVertices.push_back(xPos);
                sphereVertices.push_back(yPos);
                sphereVertices.push_back(zPos);
        }
}
```

需要注意的是，在横向分割时，n=0 和 n= X_SEGMENTS 是位置相同的点，如果后期需要加载纹理则不能随意将之舍去，所以顶点总数实际上是 (X_SEGMENTS + 1)* (Y_SEGMENTS+1)

2 构造三角形图元

由于 opengl 中是以三角形为基本图元进行绘制的，所以在我们得到了球面遍历的顶点数组后，我们需要扩充顶点，构造成基本的三角形图元。

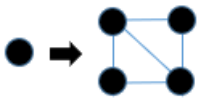


图 2-1 顶点扩充

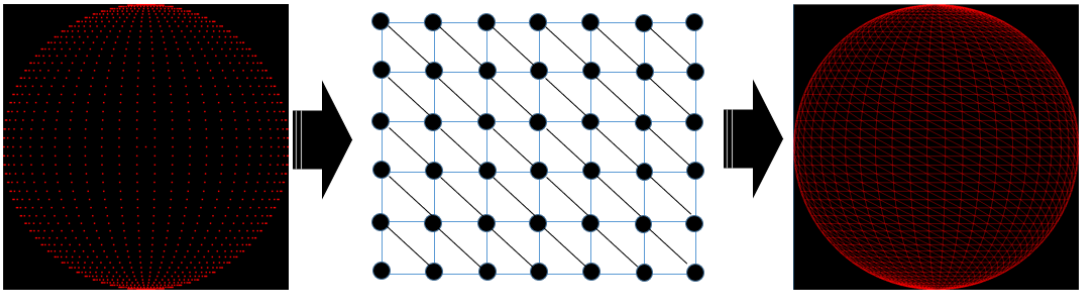


图 2-2 球体扩充效果

我们会将对遍历球面顶点数组的每个顶点，将其扩充为两个三角形，除去最后一行顶点和每一行最后一个点外，每一个顶点都会扩充成六个顶点，原数组一共有(X_SEGMENTS+1)* (Y_SEGMENTS+1)个顶点，扩充后有 X_SEGMENTS*Y_SEGMENTS * 6 个顶点，这里我们使用索引数组去构造三角形图元。

实现代码如下：

```
//根据球面上每一点的坐标，去构造一个一个三角形顶点数组
for (int i = 0; i < Y_SEGMENTS; i++)
{
        for (int j = 0; j < X_SEGMENTS; j++)
        {

                sphereIndices.push_back(i * (X_SEGMENTS+1) + j);
                sphereIndices.push_back((i + 1) * (X_SEGMENTS + 1) + j);
                sphereIndices.push_back((i + 1) * (X_SEGMENTS + 1) + j +
1);

                sphereIndices.push_back(i * (X_SEGMENTS + 1) + j);
                sphereIndices.push_back((i + 1) * (X_SEGMENTS + 1) + j +
1);
                sphereIndices.push_back(i * (X_SEGMENTS + 1) + j + 1);
        }
}
```

3 开启线框模式

为了观察我们绘制的球模型的顶点是否存在错误，由于 opengl 中默认的绘制方式为填充模式，所以我们需要改填充模式为线框模式，这样可以更好的检查绘制结果

如图为填充模式和线框模式的区别（左图为填充模式，右图为线框模式）。

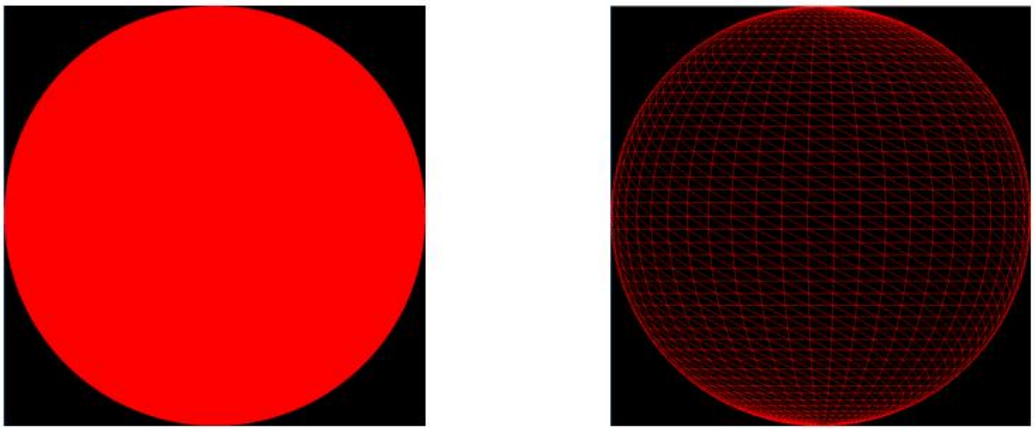


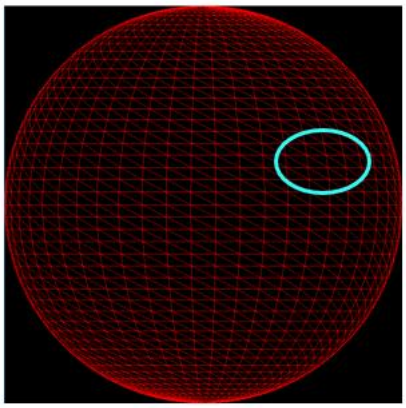
图 3-1 填充模式和线框模式效果对比

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE); //使用线框模式绘制
```

4 开启面剔除

由于 opengl 中绘制时会同时绘制正面和背面，并且我们开启了线框模式后，并不能通过深度测试将背面遮挡住，所以我们需要开启面剔除。来将背面剔除，面剔除主要是剔除顶点绕法为顺时针的面

如左下图，为开启面剔除后，将背面剔除的结果。



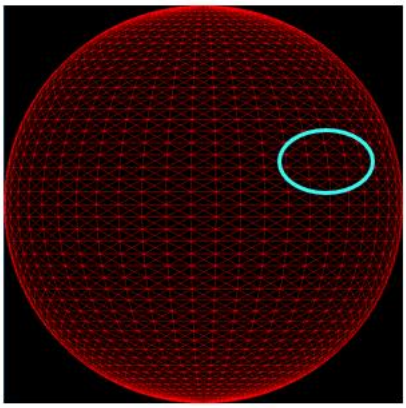


图 4-1 填充模式和线框模式效果对比

同样的，代码中我们只需要开启面剔除，并指定剔除面即可，GL_FRONT 为剔除正面，GL_BACK 为剔除背面。以下为代码实现：

```
//开启面剔除(只需要展示一个面，否则会有重合)
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

5 最后

当做好了以上的步骤之后，我们可以看到下面的结果，如果你绘制不出来造型规则的球体，希望你可以看一下附带的源码，比对一下差异。

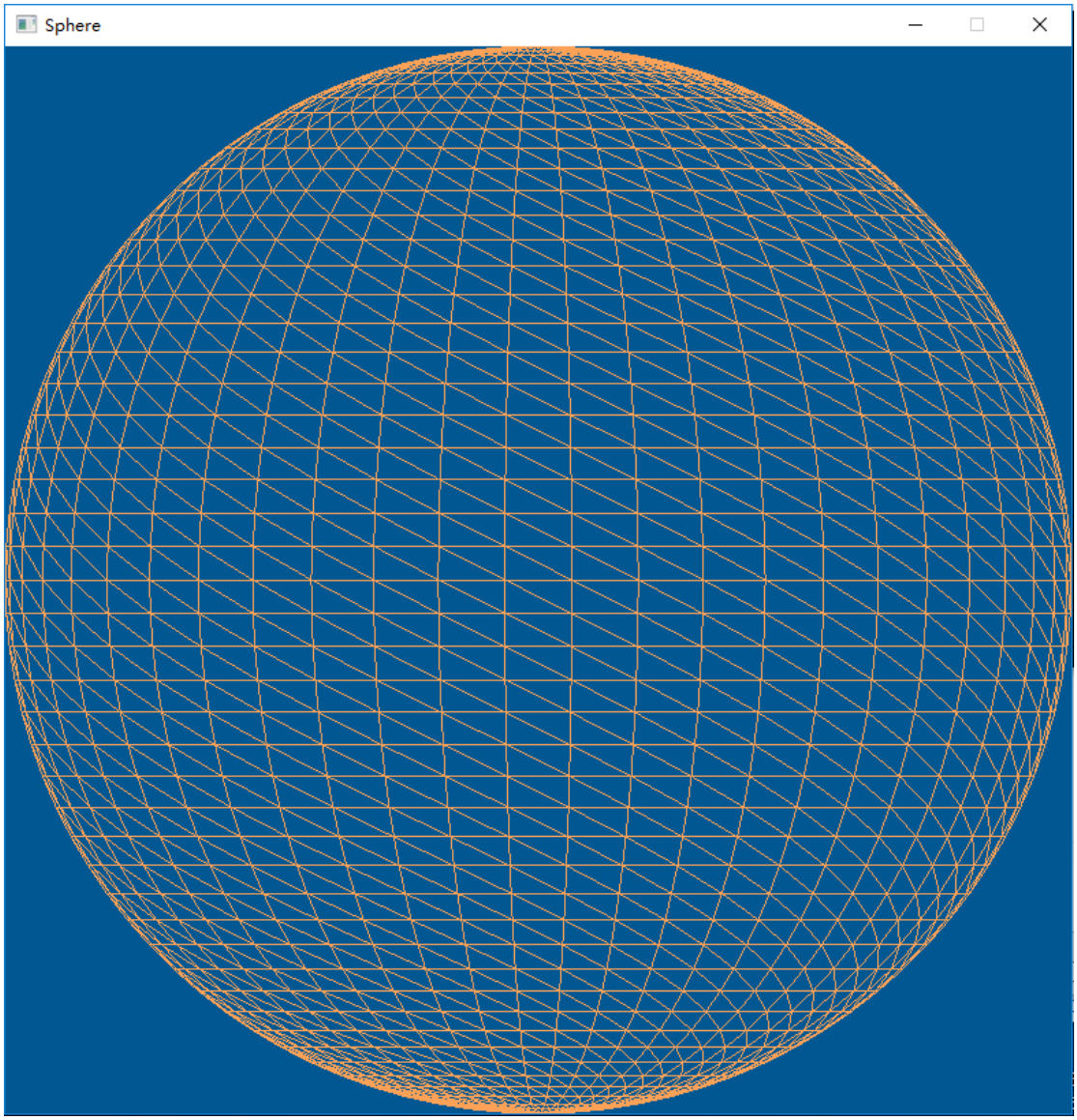


图 5-1 效果图