#### 实验环境

* **硬件环境**：
  + 操作系统：Windows 10
  + GPU：NVIDIA GTX 1650
  + 内存：16GB
* **软件环境**：
  + Python 版本：3.8
  + OpenGL 版本：3.3
  + 主要库：pyOpenGL、Pyrr、PIL、glfw、numpy
  + 纹理图片：earth.jpg

#### 实验方法

**创建 OpenGL 环境**：

* + 使用 glfw 初始化窗口和 OpenGL 上下文。
  + 设置窗口大小为 800x600，开启深度测试。

**生成球体数据**：

* + 球体是通过经纬度方式生成的，包括顶点位置、法向量、纹理坐标和索引数据。
  + 使用 generate\_sphere() 函数生成球体的顶点数据。

**编译着色器程序**：

* + 顶点着色器计算每个顶点的位置、法向量，并将纹理坐标传递到片段着色器。
  + 片段着色器计算环境光、漫反射和高光，并根据光照计算最终颜色。
  + 光源位置、视点位置和纹理均通过 uniform 变量传递。

**加载纹理**：

* + 使用 PIL 库加载并翻转纹理图片（因为 OpenGL 中的纹理坐标系与图片坐标系是相反的）。
  + 使用 glTexImage2D 加载纹理并生成 Mipmap 纹理。

**设置光照与相机**：

* + 使用 Pyrr 库创建视图矩阵和投影矩阵。
  + 设置点光源位置、光源颜色和观察者位置。

**渲染过程**：

* + 清除颜色和深度缓存。
  + 激活着色器程序并为每个顶点设置相应的变换矩阵。
  + 绘制球体并交换缓冲区。

#### 实验步骤

**初始化 OpenGL 环境**：

* + 使用 glfw.init() 初始化 GLFW 库。
  + 创建一个窗口并设置上下文。

**编译着色器**：

* + 创建顶点着色器和片段着色器，使用 compileShader 和 compileProgram 编译和链接着色器程序。

**生成球体数据**：

* + 使用经纬度法生成球体顶点数据，包括法向量和纹理坐标。

**加载纹理**：

* + 使用 PIL 库加载纹理图片，并使用 OpenGL API 绑定和生成纹理。

**设置视图和投影**：

* + 使用 pyrr.matrix44.create\_look\_at 创建视图矩阵。
  + 使用 pyrr.matrix44.create\_perspective\_projection 创建投影矩阵。

**主循环**：

* + 在每帧中清除缓存，绑定着色器程序，设置 uniform 变量，绘制球体。

**结束程序**：

* + 使用 glfw.terminate() 结束 OpenGL 环境。

#### 实验结果

在运行程序后，窗口中会显示一个带有地球纹理的球体。通过调整光源的位置、强度和视角，球体的光照效果可以实时变化。通过对不同的纹理图片进行加载，球体表面的纹理可以灵活更换。

实验中的光照效果包括：

* **环境光**：模拟球体周围的柔和光照。
* **漫反射**：模拟光源照射到球体表面的效果。
* **高光**：模拟反射光源光线到视角的高光效果。

