真实感渲染

实验环境

Windows11

VS2022

OpenGL

glut

运行方法

`./bin`目录下运行`.msi`文件安装程序，后运行`.exe`文件运行。

可执行文件同目录下必须有`skybox`文件夹，`ply`文件夹，`config`文件。其中`config`文件分为4行，前三行为三个模型的坐标，前两行分别有三个浮点数表示xyz坐标，第三行除了xyz坐标还有一个alpha(0~1)值，表示其透明度。

第四行为光源的xyz坐标。

第五行为光线追踪模式中相机的xyz坐标。

第六行为光线追踪模式中相机的视点xyz坐标。

因此，文件中共有19个浮点数。

运行可执行文件中，根据命令行提示，输入1为Phong模型，输入2为光线追踪。

实现内容

- 构建一个静态场景，场景中包含一个透明材质的物体，模型均取自于清华云盘。
- 为场景添加点光源，采用Phong光照明模型。
- 基本的交互以及场景漫游功能。
- 自行定义场景文件，能配置模型的位置、材质、点光源的位置。
- 实现基本的光线跟踪算法，输出图片。

本次作业基于作业2完成，因此原有的天空盒和键盘及鼠标的交互方式保留

- 鼠标：按下左键拖动
- 键盘：WASD

实验原理

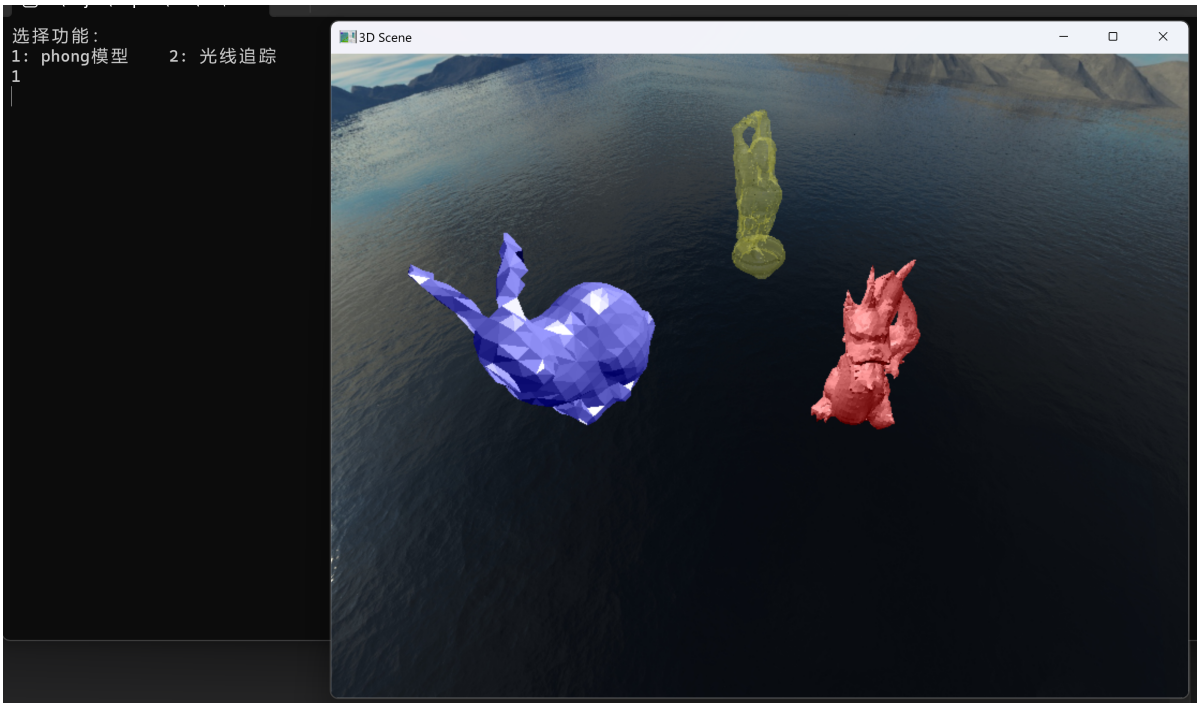
Phong模型

Phong光照模型通常包括环境光、漫反射光和镜面反射光。有公式如下：

$$I = I_a K_a + I_p K_d (L \cdot N) + I_p K_s (R \cdot V)^n$$

场景光源和物体表面的ambient、diffuse、specular均硬编码在程序内，因此只需简单的计算便可得出物体显现出来的颜色。

对于复杂的模型，我们通过对其的每一个三角面片计算Phong模型下的表面颜色得出模型整体的颜色。



光线追踪

光线追踪是一种用于模拟光线在场景中传播和相互作用的算法。它是一种基于物理光学原理的渲染技术，通过追踪光线的路径来模拟光的行为，从而生成逼真的图像。

> 基本的光线追踪算法步骤如下：
>
> 1. **光线生成**：从相机或观察者的位置出发，生成射向屏幕的光线。每条光线都对应屏幕上的像素。
> 2. **光线与物体相交检测**：对于每条射线，判断它是否与场景中的物体相交。
> 3. **光线与物体相交计算**：一旦光线与物体相交，算法需要确定相交点在场景中的位置以及与该点相关的属性（如法线、材质、颜色等）。
> 4. **光照计算**：基于相交点处的材质属性（如漫反射系数、镜面反射系数、折射率等），以及光线与表面法线的夹角等信息，计算该点的光照强度。
> 5. **递归处理**：在光线追踪中，还会涉及镜面反射、折射等光线路径的处理。当光线与反射表面相交时，可以生成新的反射光线，以追踪其路径。这个过程可以递归进行。
> 6. **合成图像**：对于每个像素，将其颜色值根据光线追踪的结果进行累积，最终合成出整个场景的图像。

下面给出核心部分伪码：

```
function traceRay(origin, direction, depth):
    if depth <= 0: // 递归深度达到上限时返回背景颜色或其他设定的颜色
        return backgroundColor

    intersectionPoint = findNearestIntersection(origin, direction) // 寻找光线与物体的相交点
    if intersectionPoint does not exist: // 如果没有相交点，返回背景颜色
        return backgroundColor

    surfaceNormal = calculateSurfaceNormal(intersectionPoint) // 计算相交点的表面法线
    material = getMaterialAtIntersection(intersectionPoint) // 获取相交点的材质信息

    ambientColor = calculateAmbient(material) // 计算环境光颜色
    diffuseColor = calculateDiffuse(material, surfaceNormal, lightDirection) // 计算漫反射颜色
    specularColor = calculateSpecular(material, surfaceNormal, lightDirection, viewDirection) // 计算镜面反射颜色

    totalColor = ambientColor + diffuseColor + specularColor // 合并颜色

    // 递归处理反射和折射
    reflectionRayDirection = calculateReflectionDirection(direction, surfaceNormal)
    reflectionColor = traceRay(intersectionPoint + epsilon * reflectionRayDirection, reflectionRayDirection, depth - 1)
    refractionRayDirection = calculateRefractionDirection(direction, surfaceNormal)
    refractionColor = traceRay(intersectionPoint + epsilon * refractionRayDirection, refractionRayDirection, depth - 1)

    finalColor = totalColor + material.reflectionCoefficient * reflectionColor + material.refractionCoefficient * refractionColor

    return finalColor
```

由于设备受限，最高的分辨率只尝试了150*150。100*100的结果如下图：