# 大作业——题目 4：光线跟踪 报告文档

——无手工院异闻录 任泓旭 潘浩然 吴秉宪

一、完成内容

本作业使用 c++编程，构建了一个有几种不同材质物体的场景，采用光线跟踪算法来模拟了光线在这个场景中的传播、反射。此外，用户可以通过输入来选择不同的模型、材质和调整采样次数和光线反射次数，决定场景的效果是怎样的。

本程序的结果展现主要采用的是 openCV 的库，而光线追踪用到的类和函数是自己定义的，没有使用小作业时常用的 glfw。(由于所需要用到的类相对于 glfw 来说并不多)

代码由一个 cpp 文件和三个 h 文件组成，具体说明如下。

二、代码说明

main.cpp：这是程序的主要部分，在这里，程序对不同材质进行了设置，包括漫反射材质、金属材质和玻璃材质等。同时，整个场景的搭建也是在这里完成的，从左到右依次设置了一个球、一个手或茶壶、另一个球模型。他们具体是何种材质，由后面通过输入设置。此外，为了便于光照效果的体现，在场景的四周和底部设置了不同颜色的墙体，它们也会对光线漫反射，这样使得光线的效果更加便于辨认。在手动输入了三个模型的材质之后，还需要手动输入光的反射次数和采样时间，这样可以灵活设置渲染时间的长短，当然也会影响渲染结果的精细程度。这里也定义了 camera 来设置观察结果的一个固定视角，随后就光线追踪来计算每一个点的颜色

具体地说明光线追踪部分的实现方式：通过循环遍历每个像素，使用光线追踪算法计算像素的颜色值，并将结果保存到 result 图像中。

```
for (int j = image_height - 1; j >= 0; j--)
{
    for (int i = 0; i < image_width; i++)
    {
        color pixelvale(0, 0, 0);
        for (int s = 0; s < samplesnumber; ++s)
        {
            ray r = cam.get_ray(1.0 * i / image_width, 1.0 * j / image_height);
            pixelvale += ray_color(r, scene, background, max_depth);
        }

        // 将像素的颜色值保存到result图像中
        result.at<cv::Vec3b>(image_height - j - 1, i)[0] = static_cast<uchar>(255 * clamp(pow(sqrt(1.0 * pixelvale.z() / samplesnum
        result.at<cv::Vec3b>(image_height - j - 1, i)[1] = static_cast<uchar>(255 * clamp(pow(sqrt(1.0 * pixelvale.y() / samplesnum
        result.at<cv::Vec3b>(image_height - j - 1, i)[2] = static_cast<uchar>(255 * clamp(pow(sqrt(1.0 * pixelvale.x() / samplesnum
    }
}
```

这里用到了一个 ray_color 函数，它根据光线与场景中物体的交点，计算光线的反射、折射和散射等效果，并返回最终的颜色值。

```
color ray_color(const ray& r, const hittable& scene, color background, int depth)
{
    // 记录交点的信息
    hit_record rec;

    // 超过了反弹极限次数，返回0
    if (depth <= 0)
    {
        return color(1, 1, 1);
    }

    // 光线击中物体的话,递归操作, material 是有color的
    if (scene.hit(r, 0.001, infinity, rec))
    {
        // 每次反射更新ray
        ray scattered;
        // 反射后的色彩
        color attenuation;
        if (rec.mat_ptr->scatter(r, rec, attenuation, scattered))
        {
            if (scattered.direction().length_squared() > 1e36)
            {
                return attenuation;
            }
            auto temp = ray_color(scattered, scene, background, depth - 1);
            return attenuation * temp;
        }
        return color(0, 0, 0);
    }
```

通过两个嵌套的循环遍历每个像素。对于每个像素，使用 cam.get_ray 函数获取光线的起点和方向，并调用 ray_color 函数计算光线的颜色值。在每个像素上进行多次采样，将采样结果累加到 pixelvale 变量中。最后，将 pixelvale 中的颜色值进行归一化处理，并将结果保存到 result 图像中的相应位置。

```
ray get_ray(double s, double t) const
{
    vec3 rd = lens_radius * random_in_unit_disk();
    vec3 offset = u * rd.x() + v * rd.y();
    return ray(
        origin + offset,
        lower_left_corner + s * horizontal + t * vertical - origin - offset,
        random_double(time0, time1)
    );
}
```

在光线追踪的这个过程中，我们调用了 openMP 来对其进行并行加速，从而缩短了渲染所需要的时间。

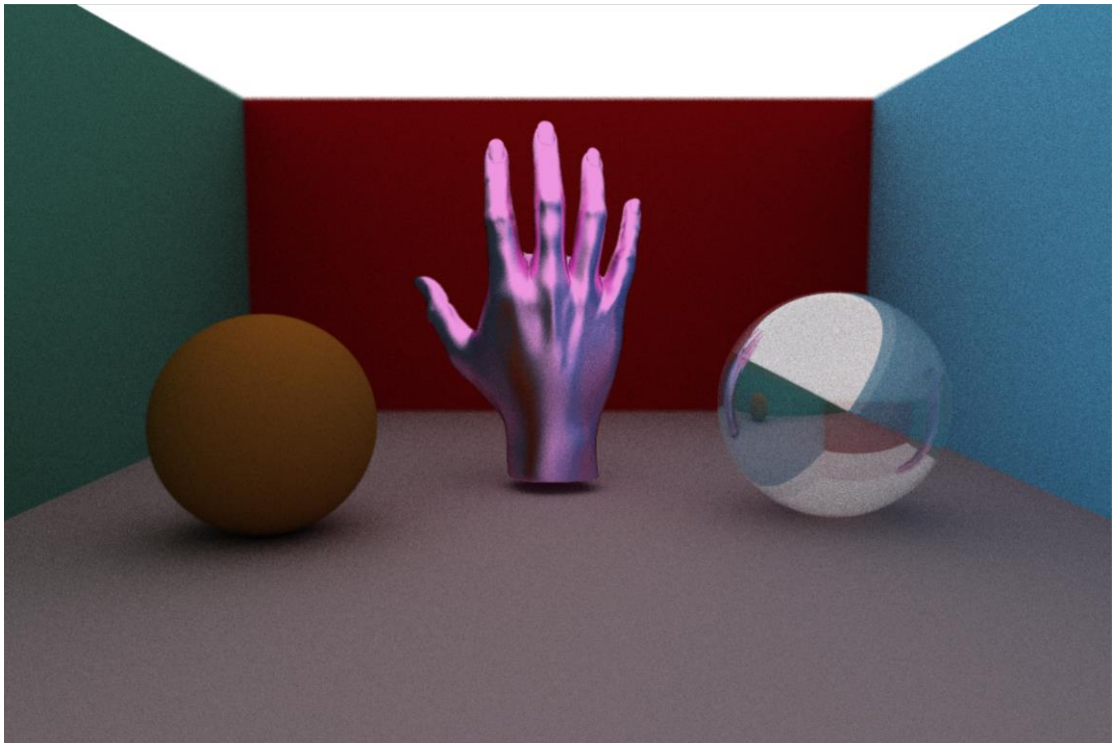
Utils.h：由于没有使用更庞大的 glfw，这个头文件里定义了一些用来使用的类和函数，比如说包括 vec3 这样的一个三维向量，并包括了常用的加减乘除、模长、点乘叉乘等功能；mat3x3 一个自定义矩阵类，用于表示 3x3 的矩阵，并且定义了一个 rotate 操作，即通过旋转矩阵来实现向量的旋转。Ray 则是一个自定义的射线类，用于表示光线在三维空间中的起点和方向。Camera 则显而易见的是一个摄像机，除了设置位置、lookat 和 up 以外，还定义了屏幕的宽高比、光圈大小、焦距。上一段所提到的 get_ray 函数也是在这里定义的，它获取了相机位置的光线。

Hitsphere.h：主要用于球模型，定义了多个与之相关的类和函数。其中，class material 是一个抽象基类，定义了材质的散射函数。它有几个派生类，包括 class lambertian、class metal、class dielectric 和 class lightsource，分别实现了不同类型的材质属性。class sphere 和 class plane 分别表示球体和平面对象，继承自 hittable 类。它们都有一个碰撞检测函数 hit，用于判断光线是否与对象相交，并计算交点的信息。ray_color 函数用于计算光线的颜色。通过以上类和函数，得以实现对光线是否通过球体进行判断和根据材质计算光照颜色。

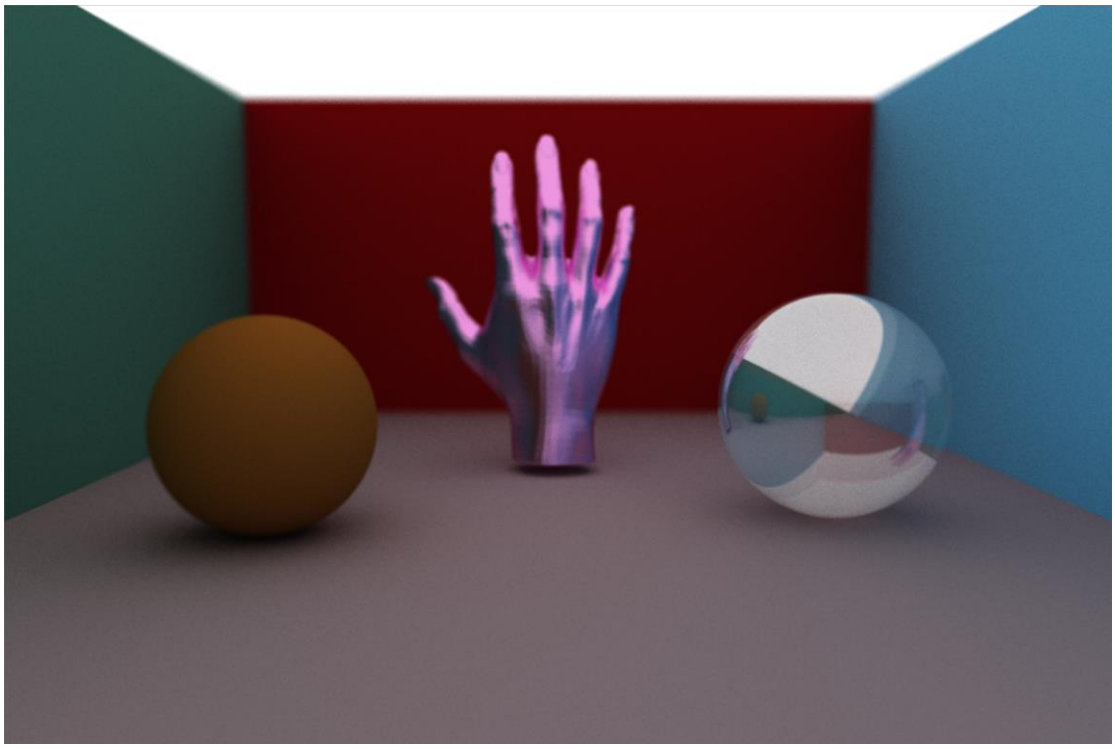
Triangle_mesh.h：这个是一个更复杂的三角网格模型，在本程序中，组成了一个手型的模型（和组名相对应了）或者是茶壶的模型，并且类似于 hitsphere 对球体的处理，实现了光线与模型碰撞的检测及其颜色的计算。具体来说，先定义了类 class triangle_mesh，包括了顶点、索引、包围盒和材质属性等信息，在构造函数 triangle_mesh::triangle_mesh 中通过 assimp 库加载 Hand.fbx 或者 teapot.obj 来导入这个模型。此外，使用了 BVH 树的数据结构存储网格，优化了速度

## 三、实现效果

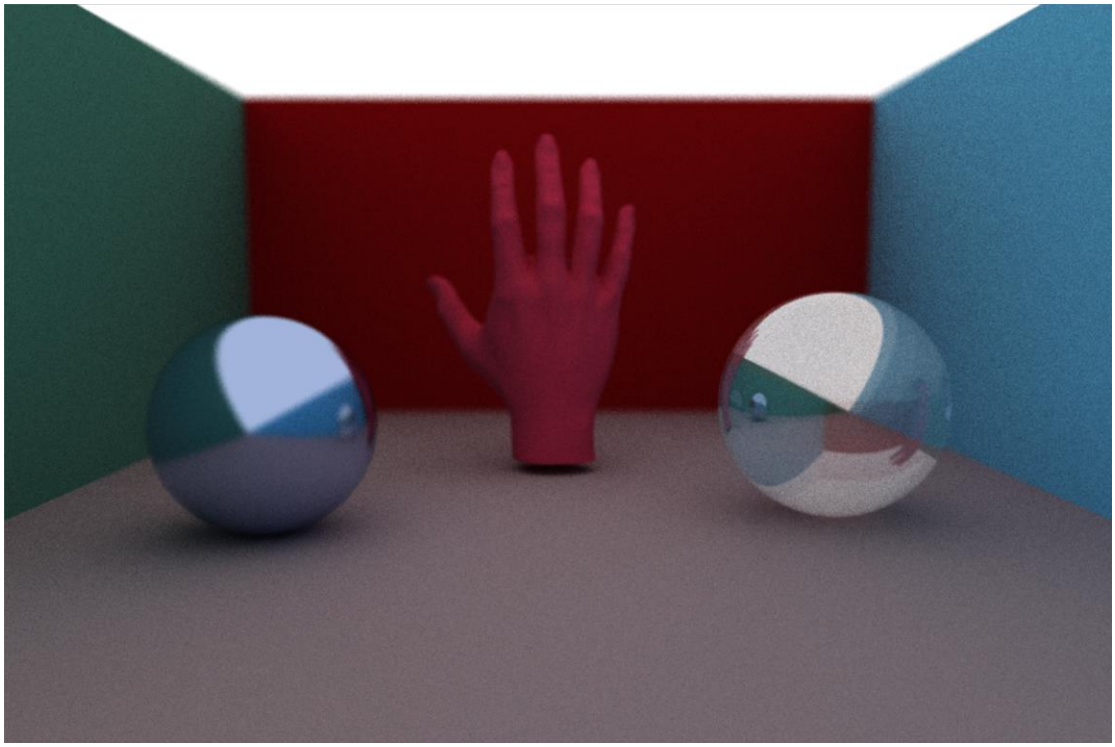
设置采样次数为 100，光线反射次数为 20，球的材质设为粗糙材质（漫反射）和玻璃，手的材质设为金属，程序通过 omp 调用 12 个线程用时约 1 分钟得到结果如下：



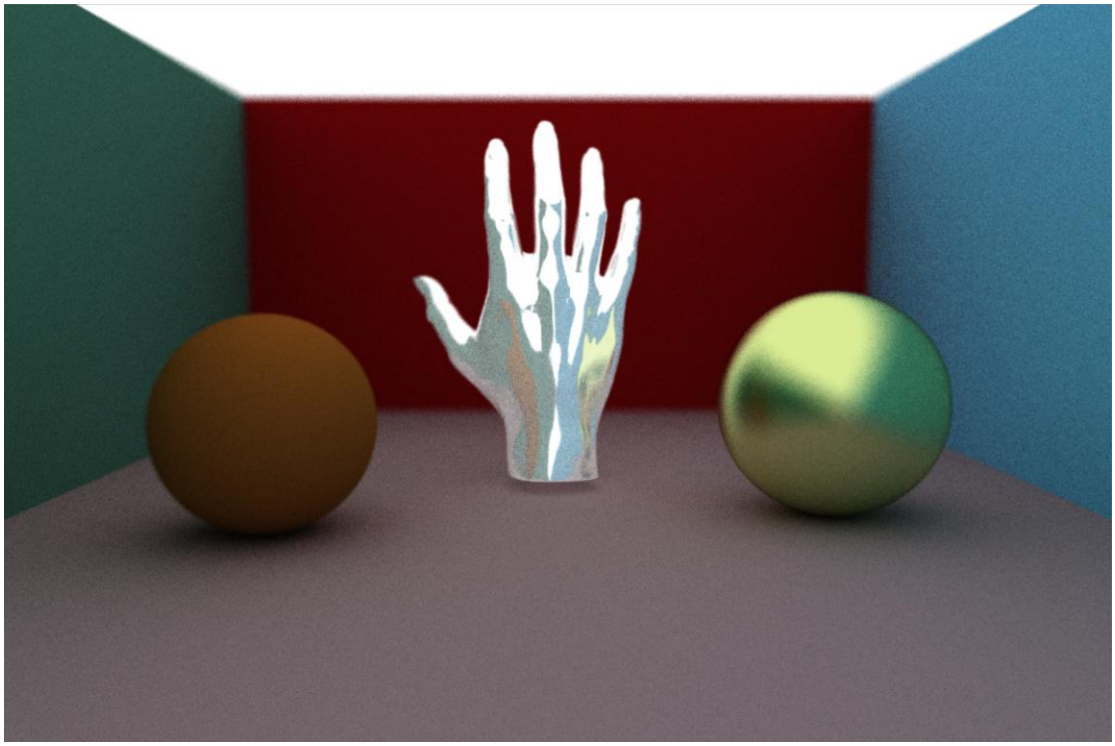
若设置采样次数 500，反射次数 50，用时约 500s，结果有较小幅度的更清晰：

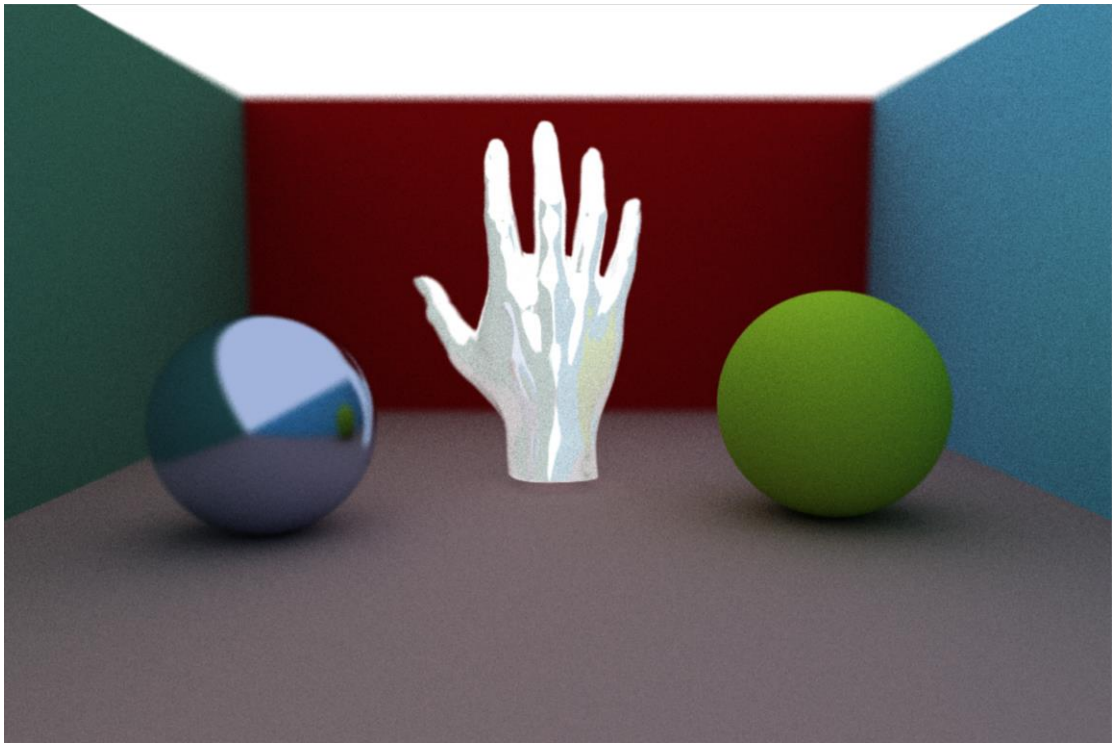


若将手的材质设置为粗糙（漫反射），得到结果如下:

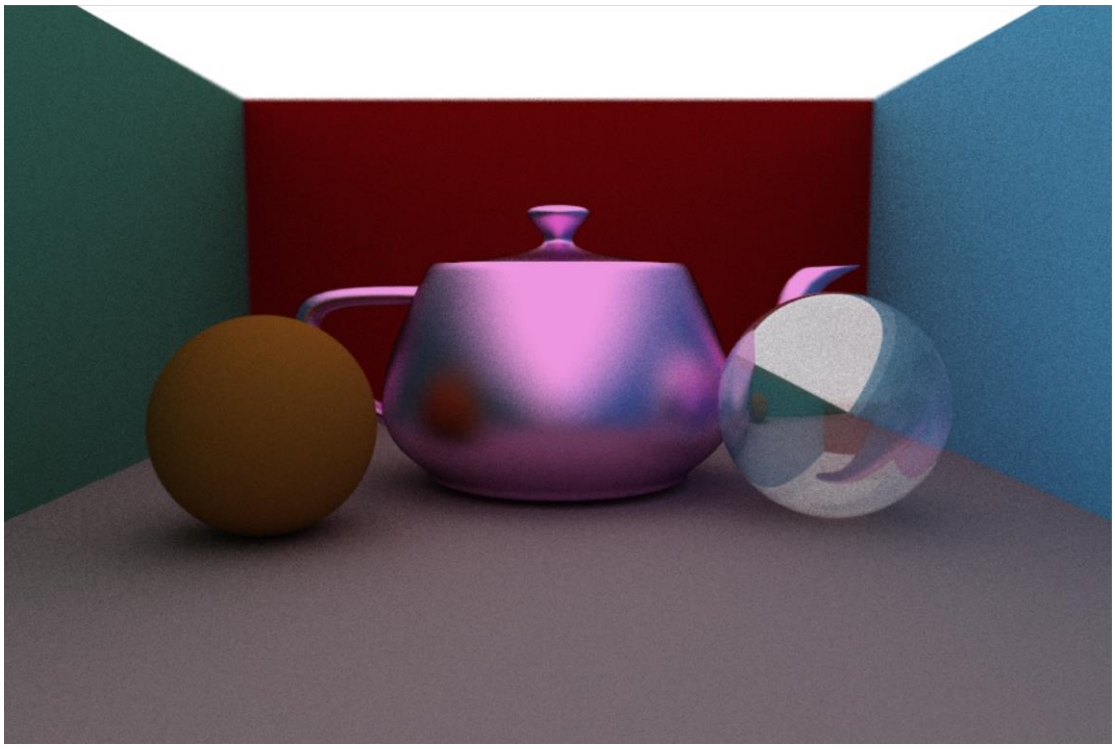


若将手的材质设置为玻璃，得到结果如下:

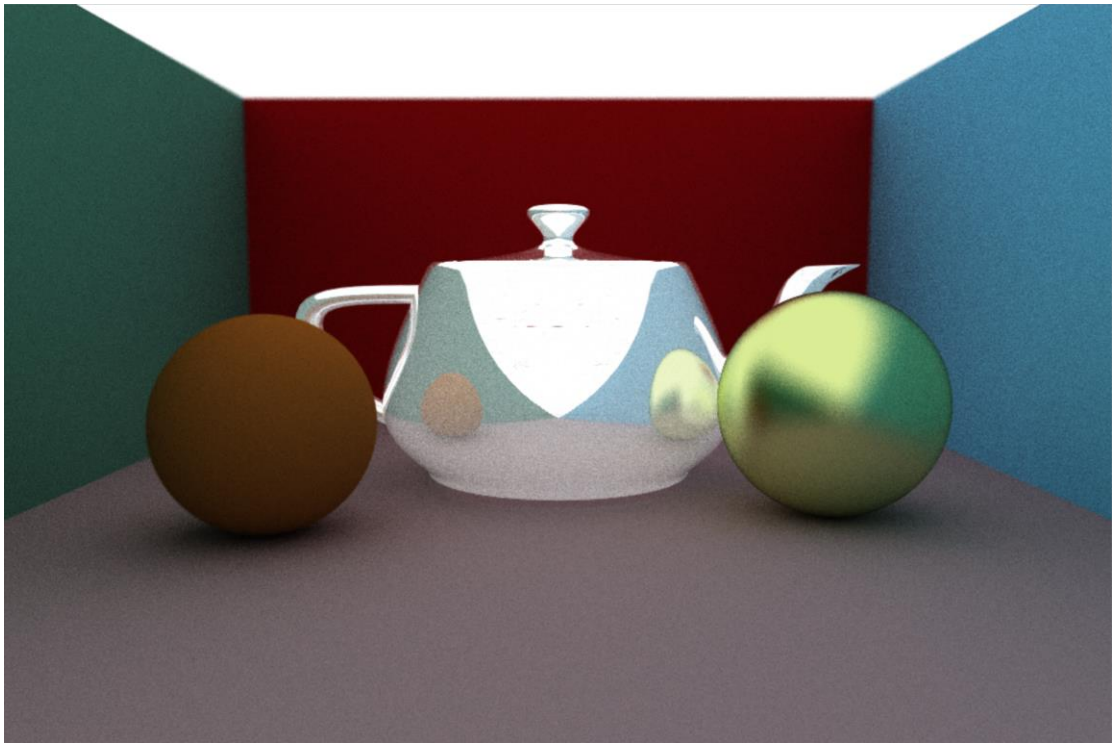




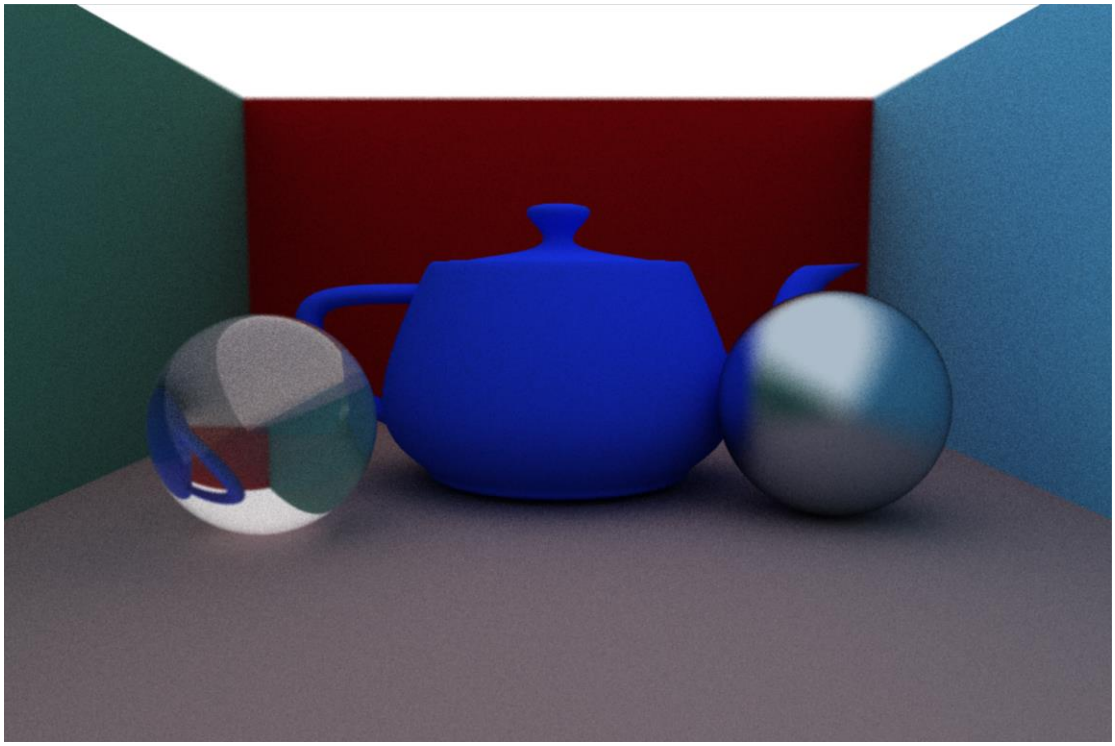
金属茶壶:



玻璃茶壶



粗糙茶壶



四、小组分工

任泓旭：添加了由三角网格模型构成的手、茶壶模型，对各模型完成了加载，搭建了场景四周的墙体及其材质，改进了光线追踪算法的实现（debug 和加速），实现了 openmp 并行加速

潘浩然：搭建代码框架，对金属材质实现了基本的光线追踪框架，撰写文档

吴秉宪：对粗糙材质（漫反射）和玻璃材质实现了基本的光线追踪框架，录制视频