实验 6：加速结构

要求：

- **独立完成**
- **在截止时间之前提交**

---

1 总览

在之前的编程练习中，我们实现了基础的光线追踪算法，具体而言是光线传输、光线与三角形求交。我们采用了这样的方法寻找光线与场景的交点：遍历场景中的所有物体，判断光线是否与它相交。在场景中的物体数量不大时，该做法可以取得良好的结果，但当物体数量增多、模型变得更加复杂，该做法将会变得非常低效。因此，我们需要加速结构来加速求交过程。在本次练习中，我们重点关注物体划分算法 Bounding Volume Hierarchy (BVH)。本练习要求你实现 Ray-Bounding Volume 求交与 BVH 查找。

首先，你需要从上一次编程练习中引用以下函数：

• **Render()** in Renderer.cpp: 将你的光线生成过程粘贴到此处，并且按照新框架更新相应调用的格式。

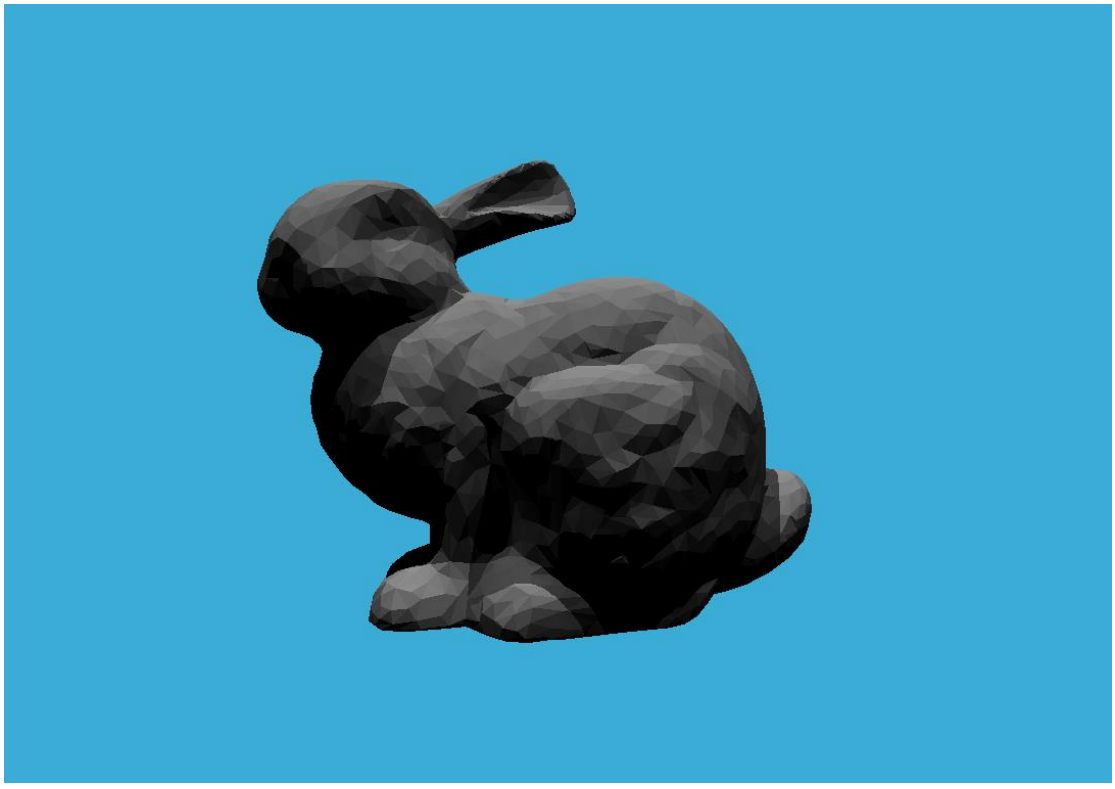
• **Triangle::getIntersection** in Triangle.hpp: 将你的光线-三角形相交函数粘贴到此处，并且按照新框架更新相应相交信息的格式。

在本次编程练习中，你需要实现以下函数：

• **IntersectP(const Ray& ray, const Vector3f& invDir, const std::array<int, 3>& dirIsNeg)** in the Bounds3.hpp: 这个函数的作用是判断包围盒 BoundingBox 与光线是否相交，你需要按照课程介绍的算法实现求交过程。

• **getIntersection(BVHBuildNode* node, const Ray ray)** in BVH.cpp: 建立 BVH 之后，我们可以用它加速求交过程。该过程递归进行，你将在其中调用你实现的 Bounds3::IntersectP。

如果实现是正确的，你将得到下图：



2 代码框架

在本次实验中，我们修改了代码框架中的如下内容：

• Material.hpp: 我们从将材质参数拆分到了一个单独的类中，现在每

个物体实例都可以拥有自己的材质。

• Intersection.hpp: 这个数据结构包含了相交相关的信息。

• Ray.hpp: 光线类，包含一条光的源头、方向、传递时间 t 和范围 range。

• Bounds3.hpp: 包围盒类，每个包围盒可由 pMin 和 pMax 两点描述（请思考为什么）。 Bounds3::Union 函数的作用是将两个包围盒并成更大的包围盒。与材质一样，场景中的每个物体实例都有自己的包围盒。

• BVH.hpp: BVH 加速类。场景 scene 拥有一个 BVHAccel 实例。从根节点开始，我们可以递归地从物体列表构造场景的 BVH。

3 提高项

实现 SAH 查找：自学 SAH(Surface Area Heuristic)，正确实现 SAH 加速(可参考 http://15462.courses.cs.cmu.edu/fall2015/lecture/acceleration/slide_024，也可以查找其他资料)。