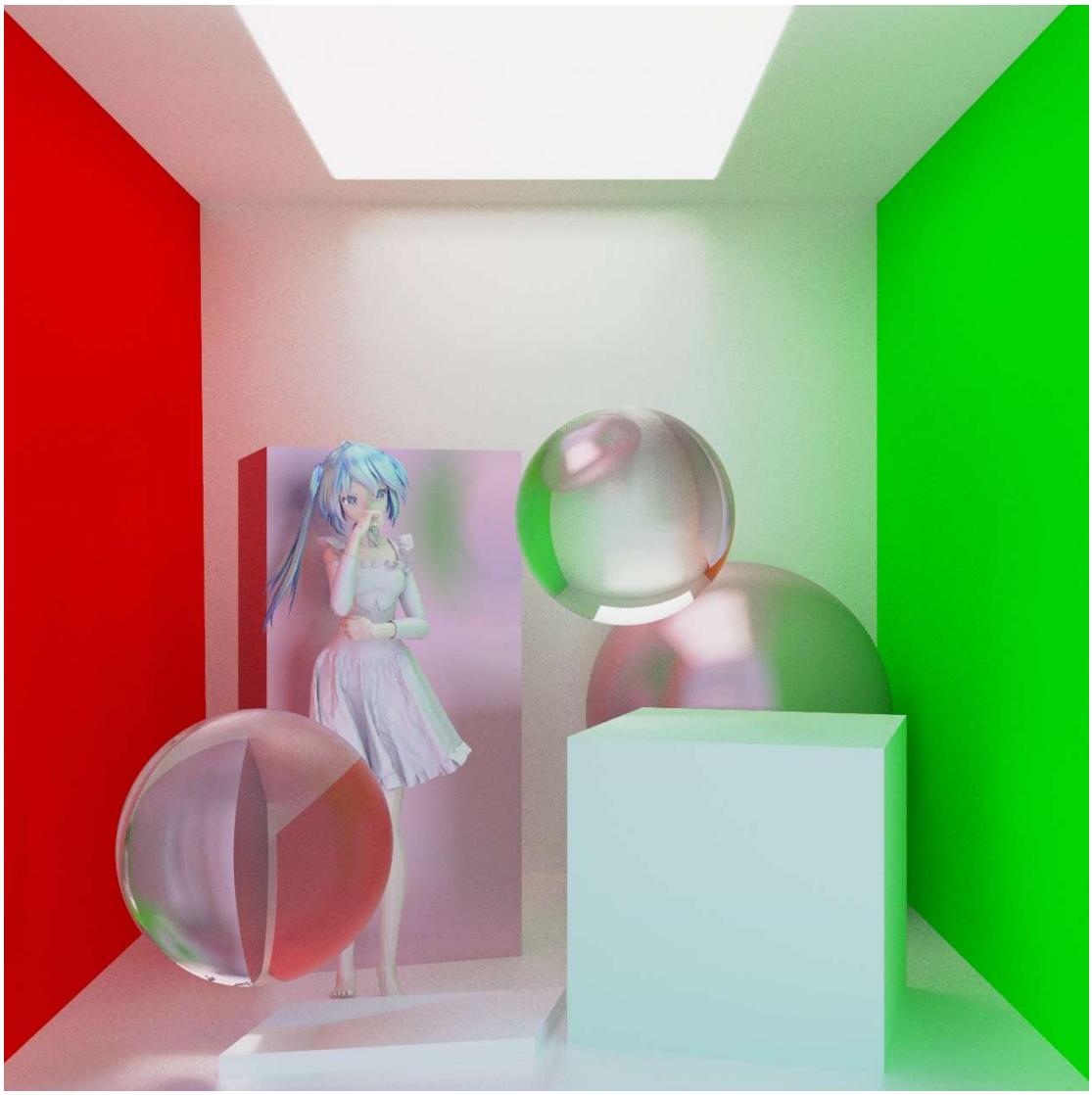
基于蒙特卡洛方法的高性能路径追踪渲染器

——SimplePathTracer

（根据 Ray Tracing in One Weekend Series[1]，使用 Java 语言进行实现）



开发者：王哲 （20373209）

指导老师：李波

项目规模：源代码 2131 行

所属学院：计算机学院

目录

项目简介

本软件项目名为 SimplePathTracer，是一款纯命令行控制的路径追踪渲染器。

本软件旨在创建一款简单的 ，高性能的，能渲染绚丽三维场景画面的路径追踪渲染器。在面向对象的编程模型下，使用 JAVA 编写此程序。封面为此渲染器最终渲染成品（场景模型及贴图全部为公开资源，已附带在程序包中）。

使用说明

程序包文件目录说明

本程序包内，根目录下有两个文件夹，分别命名为 renderRes 与 src，其中 src 目录中存放着本程序使用的所有 JAVA 源代码，而在 renderRes 中存放着渲染器 Demo 所使用的三维模型文件以及贴图文件，材质描述文件等。

在根目录下，还存有多个用于运行 Demo 的 bat 脚本文件，预先编译好的程序 JAR 包（JAVA8 版本以及 JAVA17 版本），预先渲染出的 Demo 渲染出图。

本程序中，所有贴图，以及输出图像均使用 PNG 格式，而三维模型描述文件使用非标准的 WaveFront OBJ 格式，三位贴图文件使用修改过的非标准 WaveFront MTL 格式，故直接从标准三位软件中导出的格式，需要进行一定修改之后，才能用于此渲染器。具体对于格式的描述请见后续 IO 格式部分。

Demo 使用说明

本程序文件包内存有用于演示渲染器功能的场景启动脚本，以及预先编译好的 Jar 包。本程序分别针对 Java8 以及 Java17 环境，编译了两个 Jar 包，分别以_Java8，_Java17 作为名称后缀。

在本程序包文件夹内，存有多个可供选择使用的渲染 BAT 文件，文件名均已 runDemo 开头，格式形如 runDemo_[xxxx]p[yyyy]s。这些渲染 BAT 文件均用于渲染标题封面的场景，不过使用了不同的分辨率与采样率。其中[xxxx]表示渲染的分辨率（长宽一致），[yyyy]用于表示渲染的采样率。这其中 512p16s 为渲染较为快的样例，可以较为快的得到渲染结果（一分钟内），而 2560p256s 为封面使用的配置，需要非常长的渲染时间。

在渲染结束之后，不同后缀批处理得到的渲染结果会保存成各自文件名的文件，如 _512p16s 会保存文件为 512p16s.png。

本 Demo 场景为一个常见的 CornellBox[2] 配合一个较高面数的公开发布的初音未来三维模型。整体场景面数超过了 5 万面，存在一个面光源，多种类型的，如反射、折射、漫反射、自发光的材质，并启用了环境天光。这个场景较为充分的展示了此渲染器的可行性。用此较为复杂的 Demo，可以合适的展示本渲染器的“通用性”，与“高性能”。

命令行参数说明

本程序由于并非实时应用，渲染图像的时间非常久，故而没有设计可供直观操作的 GUI 界面，仅使用命令行参数对程序进行整体渲染器的控制。下面是本渲染器使用的三类控制命令。

-obj [String.objfilepath]

本命令用于读入指定需要使用的场景的模型 obj 文件。其中 objfilepath 参数为指向需要打开的文件的相对路径地址。本命令行可在参数中多次使用，多次使用会读入多个 obj 模型文件，同时进行渲染。

-mtl [String.mtlfilepath]

本命令用于读入指定需要使用的材质描述 mtl 文件。其中 mtlfilepath 指向需要打开的文件的相对路劲地址。本命令行同样可在参数中多次使用，多次使用会读入多个 mtl 文件中所使用的材质。

-cam [float.camX] [float.camY] [float.camZ] [float.dstX] [float.dstY] [float.dstZ] [float.Fov] [float.Focusdistance] [float.apertune]

本命令用于设置渲染时使用的镜头参数，输入参数为九个浮点值，其中，前三项描述摄像机的空间坐标 X,Y,Z，中间三项描述摄像机的朝向信息。后三项分别是设想机的视角大小（FOV），对焦距离（Focus Distance），光圈大小（Apertune）参数。其中视角使用角度制表示，对焦距离与光圈的值均使用场景单位长度作为单位。

当不使用此命令行进行配置时，将使用针对 Cornell Box 场景配置的默认摄像机配置。

-res [int.Weight] [int.Hight]

本命令用于配置渲染时候使用的分辨率，输入参数为两个整型，分别表示渲染时的水平像素数，垂直像素数。当不使用此命令时，将会采用默认的渲染分别率，即 500x500。当多次使用此命令时，仅有最后一次输入的命令有效。

-depth [int,maxDepth]

此命令用于配置渲染时的最大光线反射迭代深度，即一条光线在场景中遇到光源前，最多会反射的次数。输入参数为一个整型，表示最大迭代深度。当不使用此命令时，最大反射深度默认为 6。当多次使用此命令时，仅最后一次的命令有效。

-sr [int.samplingRate]

此命令用于配置渲染器运行时的采样率，即每个像素上发射的采样光线数量。输入参数为一个整型，表示配置的光线采样率。当没有使用此命令时候，采样率将被默认设置为 16。当多次使用此命令时，则仅最后一次的配置有效。

-sky [int.type]

此命令用于配置渲染器运行时，使用的 MissShader，即光线在没有与场景中任何一个物体相交时，返回颜色的着色器。类比于真实世界的情形，MissShader 即为天空颜色的着色器。本渲染器中，实现了三类 MissShader，分别是 GradientSky，BlackSky，PinkSky。其中 GradientSky 为从地平线方向到顶方向的渐变色蓝色天空着色器，BlackSky 与 PinkSky 均为纯色着色器。其中 PinkSky 返回粉色的天空，而 BlackSky 则返回暗淡浅蓝色的天空。

此命令的参数为一个整型，表示将要使用的天空。当设置为 0 时，将使用 BlackSky 做为 MissShader。当设置为 1 时，将使用 GradientSky 作为 MissShader。当设置为 2 时，将使用 PinkSky 作为 MissShader。当使用其他值，或不使用此命令时，默认使用 BlackSky 作为 MissShader。

-out [String.outputFilePath]

此命令用于配置输出文件的相对路径，输入的参数为一个字符串，表述输出的相对路劲。需要注意的是，路劲中不能有空格。当多次使用此命令时，仅最后一次输入有效。当不使用此命令时，输出文件名默认为 output.png。

文件 IO 格式

OBJ 文件

本程序中使用的 OBJ 格式基本遵从标准 OBJ 文件的规格，使用 Ascil 格式记录三位模

型。文件中每个有效行均有一个标识符表示类型，分别为 v，vt，f，b，usemtl 五种。其余字符开头的行均将被视为无效行而忽略。

v 标识符表示该行记录一个点，点的坐标以浮点数的形式记录在 V 表示符之后，以 X,Y,Z 的顺序储存。

vt 标识符表示该行记录一个材质 uvw 信息点，uvw 详细以浮点的形式紧跟其后，以 u，v，w 的顺序储存。其中 w 信息虽然也会进行读入，但在本程序中，并没有实际发挥作用。

f 标识符表示改行记录一个三角面（标准 OBJ 支持多边形，但在本渲染器中，仅能使用三角面）。F 标识符之后会接着三组数对，每组数对是由两个整形由斜杠分隔符分割开组成。每个数对中的第一个表示三角面中一个空间坐标点的序号（即在此 f 之前，按照 v 出现顺序为依据的序号，从一开始），数对中的第二个表示 vt 点的序号，概念同上。

b 标识符为非标准的 obj 格式实现，表示一个球体。B 之后跟着四个浮点值，分别是球体中心的 x y z 坐标以及球体半径 r。

usemtl 标识符用于标识之后使用的材质。Usemtl 之后跟着一不含空格的字符串，表示之后的所有三角面与球体使用的材质名。对应材质名的材质将在 mtl 文件中进行定义。

MTL 文件

本程序中使用的 mtl 文件同样不是标准的 MTL 文件，mtk 文件类似 obj 文件，每一个有效行均有标识符，表示符包括 newmtl，map_K[x]系列标识符，K[x]系列标识符，Roughness 标识符，Fuzz 表示符号，Kn 标识符。

newmtl 标识符后接一字符串，用于定义新材质，材质名即为字符串内容。newmtl 行之后，直到下一行 newmtl 之间，均为此新材质的描述内容。

描述内容中，map_K[x]系列标识符，与 K[x]均用于设置某一类的贴图，其中前者将图片作为贴图，后者将常量作为贴图。前置后需要接着贴图的相对路径（相对可执行 JAR 文件），后者需要接着三个浮点值（0-1）分别表示 R,G,B 分量的值。

Roughness 标识符号，此标识符号后面接一个浮点数，用于描述材质的粗糙度（0-1）。具体的在此程序中，粗糙度的含义为在材质表面光线发生漫反射的概率。关于粗糙度的定义，在不同的标准渲染器实现中，各有不同。

Fuzz 标识符号，此标识符号后面接一个浮点数，标识物体表面发生镜面反射时候的模糊度（出射光方向的随机程度）。

Kn 标识符号，此标识符后面接一个浮点数，用于标识材质内部的折射率数值。（仅对于可折射物体有效）。

Refractable 标识符号，当出现此标识符时，说明此材质描述的物体表面是可以使用折射模型进行描述的。

对于 K[x]以及 map_K[x]系列中的[x]，包括下列四种类型 d,l,r,t。

其中，d 类型，标识该材质描述漫反射颜色，l 类型标识该材质描述发光体颜色，r 标识反射类型颜色，t 标识投射类型（折射）颜色。

渲染器内置材质

现代渲染器，一般都有支持程序化的，而不仅是纹理化的着色节点。限于工程量要求，

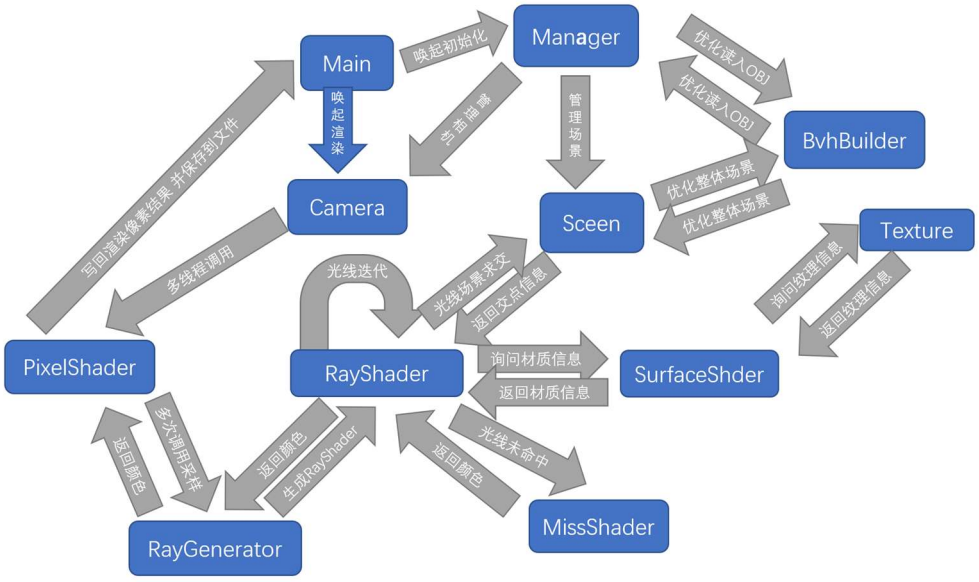
以及 Java 并非完全的解释性语言（没有 Eval 语句），故本渲染器不支持直接读取外部文件实现程序化纹理节点（不过可以通过向渲染器内部添加符合 Texture Implement 要求的 Class 文件，实现自定义的程序化纹理节点）。

本程序内置了一种棋盘格形式的程序化纹理，棋盘格纹理会在物体表面上，依据 uv 值进行展开，展开成水平，垂直方向各有 1000 格灰白交错格子的纹理。对此程序化纹理的调用，可以通过使用材质名，Inner_CheckerTexture 对其进行调用。

程序结构及代码说明

整体框架结构

整体程序框架如下图所示（对基础类如向量类 Vec3 及光线载体类 Ray 进行了省略）



整体程序的入口在 Main 中，main 将输入参数读入，并传入 Manager 中，对输入参数进行解析。在 Manager 包中，包含了对三维储存结构，如 OBJ，MTL 文件的解析函数。根据输入的参数，会自动的对选定的 OBJ 以及 MTL 文件进行解析，读入，并对数据结构进行一定的优化储存。

在初始化完成，并且完成加速结构的构建之后，会立刻通过 Camera 类中的渲染方法启动渲染。渲染中，Camera 会构建一个线程池（依据用户电脑的逻辑核心数-1 决定线程的数量）用于渲染每一行。每一行的渲染结果，会通过 Camera 多线程调用的 PixelShader 方法写回 Main 中的结果数组中。由于多线程中，每一个线程负责的像素不同，故不会出现多个线程同时写回同一个数组位置，造成多线程锁的情况，避免了可能的性能下降。

渲染完成之后，结果会有 Main 方法写回到期望保存的文件中，并销毁线程池，结束程序。

代码结构

本程序的代码，整体被分为五大类，放入五个包中，分别是

com 包，存有一些会公用的方法/类，以及基础类。

manager 包，存有一些类，用于实现对渲染器整体的管理，参数的设定，模型，材质，贴图的读入，管理等。

camera 包，存有一些和相机，以及相机的渲染主方法相关的类。

sceen 包，存放场景相关的类。

shader 包，存放渲染器使用的所有着色器。

基础类介绍

本程序 com 包中的一些类为本程序的基础类，被广泛的依赖。

Main 类位于 com.main 包中，Main 类中包含了本渲染器的入口方法 Main.main()，本渲染器的执行从这里开始。Main 类中，同时存了一些渲染器需要使用的“全局可配置变量”，作为一个“Parameter Server”。在 Main 方法中，完成了命令行参数的传递，对几类配置方法的调用，作业对象的初始化，并唤启渲染，写回结果。

com.vector 是基础向量包，是本程序基本所有三维向量运算的基础类包，包内含有 Vec3 类与 Ray 类。其中 Ray 类承载光线，并完成几类光线的基本运算（如计算光线距离某点处的点坐标，计算反射光线，计算折射光线等）。其中 Ray 的反射，折射运算的公式，可以依据几何光学以及解析几何推到而来，在部分 Ray 部分的代码中，有参考 Ray Tracing in One Weekend Series[1] 的 CPP 代码实现。

Vec3 类是本程序基础中的基础，承载所有三维向量的运算，如向量加减，点乘，叉乘，以及复杂些的产生随机向量，向量标准化，获取向量程度等等。

com.function 包中，包含了菲涅尔公式的原型以及图形学领域常用的一种简化计算版本（近似），用于辅助折射方面的计算，并提供更为接近现实的计算结果。

interface 介绍

本程序定义了四类 interface，分别是

Hitable，用于表示场景中，可求交，可影响光线的实体。

Sky，用于表示所有的 MissShader

GrayTexture，用于表示所有的灰度材质

Texture，用于表示所有的彩色材质

Hitable 提供了四个需要实现的方法，分别是 hit 方法，用于与光线进行求交，并返回碰撞的基础信息。getHitRecord 方法用于获取具体的碰撞点表面信息，generateAABB 方法用于获取实体的 AABB 包围盒信息，getCenter 方法用于获取实体中心。这四个方法中，前两个直接用于实体的光线求交，而后面两个方法则是用来帮助 AABB-BVH 加速结构的构建的。在本渲染器中，实现了 Hitable 接口的类型有 Bvh，Triangle，Ball 类型三种，分别为本渲染器支持的加速结构求交，三角面求交，以及球类求交。由于计算机图像领域，大多数的三维

模型以三角面形式表示，故该渲染器可以较为通用的渲染大部分的三维模型。

Sky 提供了唯一一个需要实现的方法，即 getColor，此方法输入一条光线，返回该光线所对应的未命中颜色。

GrayTexture 与 Texture 均仅有一个需要实现的方法，即 getTexture，且均输入需要的材质 uv 坐标。不过两者返回值一个为浮点型灰度值，一个为 Vec3 类型的颜色向量。

加速结构介绍

本程序在光线求交方面（场景遍历），使用了基于 AABB 构建的层次包围盒加速结构。此加速结构的核心思想在于，光线能碰到某一个区域的必要而不充分条件是能碰到某个能包含前述区域的区域。使用 AABB 包围盒，将物体使用按照层次排列的一系列包围盒包住，可以使光线求交计算时，排除大量的不可能存在交点的情况。

AABB 值得是轴对齐包围盒，这种包围盒，每一个面均与某一轴平面平行，故只需要确定其左下角，右上角两个坐标，就确定了包围盒的形状。这种实现，便于储存，也便于计算。

其中，BVHBuilder，是本程序中实现的加速结构 BVH 构建器。此模块用于从三角面/球体（实体）线性队列构建用来加速的 BVH（Bounding Volume Hierarchy，层次包围盒）树结构。对线性队列进行光线追踪，会有 O(n*k)的时间复杂度（k 为场景面数，n 为光线数量），而在 BVH 加速之后（本质上是一颗平衡二叉搜索树），复杂度会下降为 O（nLogk）。Manager 每读入一个 OBJ 文件，就会对该 OBJ 文件构建一次 BVH 加速结构。由于每个 OBJ 文件一般会有比较好的集中性，针对每个 OBJ 文件分别构建局部 BVH，再对于整个场景中的每一颗 BVH 树构建顶级的 BVH，可以提供更好的加速效果。在 BVHBuild 的过程中，此渲染器构建的是 AABB 包围盒的层次包围盒，在构建树时，使用每个面/球体中心的 MortonCode 作为建树依据。在此渲染器中，并没有启用 SAH 这种启发式的构建优化算法，但依然得到了非常可观的加速效果。

多线程加速渲染介绍

路径追踪是非常消耗 CPU 性能的一类计算机程序。为了充分利用计算机的资源，并加速这类程序的运行，本渲染器中引入了多线程渲染的概念。

本渲染器，使用了 Java 中 executorService 提供的多线程池服务，实现了多线程的渲染。在配置线程池大小时，选择了固定大小，且以 CPU 的逻辑核心数-1 作为线程池的大小（空余一个线程进行其余操作，不至于卡死电脑）。

在多线程任务的分配中，将每一渲染行全部以队列的形式提交给线程池，让线程池的空闲线程执行队列中行的渲染，以更为充分的利用每一个 cpu 核心的局部 L2 独享缓存作为回写，且保证每个线程之间执行的光线具有一定的局部相似性，访问相同的面，减少主存时间，充分利用高速 Cache，提高访存性能，提高性能，且不产生多线程写入锁。

Rayshader 介绍

Rayshader 模块是本程序光线迭代的主类，在此渲染器中，位于 shader 包中，类名为

RayShader。每一条光线产生之后，都会配合产生一个 Rayshader 进行计算。Rayshader 将光线投递至 Sceen 中进行求交，并获得交点信息（如果没有交点，则进行 missShader 的计算），依据交点信息更新循环记录的 muler 以及 color，并产生新的出射光线信息储存起来，等待下一次循环使用。

在 Ray Tracing in One Weekend Series 的书籍中，对光线颜色的计算采用的是迭代式，但这种迭代式，可能会在光线深度较高时，出现栈溢出，且频繁的函数入栈，出栈无益于性能，故在 Rayshader 中，依靠记录光线 muler 以及光源的值，成功使用循环替代了迭代。每一次循环，会用材质表面记录的特定需要信息，计算 muler，并依据上一次循环的 muler 以及材质表面发光信息，计算新的 color 信息。最终返回 color 值，就可以得到等价于迭代实现的效果。

三角形 Hitable 实现简介

三角形是渲染中，最为常见的一类渲染对象。在本渲染器中，对三角形的实现就显得尤为重要和基本。

上述 Interface 简介中，已经介绍了三角形 Triangle 类实现的 interface，Hitable，此类包含 hit，getHitRecord，generateAABB，getCenter 四个方法。

其中，hit 方法用于三角形的求交，在此方法中，首先将三角形扩展为无限平面，使用解析几何的向量算法算出代求光线于无限平面的交点，之后，计算该交点于三角形一点（记作点一）的连线如何使用点 12，点 13 的向量表示（s，t）坐标，并判断 s+t 是否小于等于 1，若小于，则说明交点位于三角形内，判断交上，否之不然即可。

而 generateAABB，仅需要返回能包住三角形的最小矩形盒子的左下，右上两个坐标即可，因此，从三角形三个点的坐标的每一个分量中，分别选择最大，最小值作为右上，坐下坐标对应分量的值即完成了计算。

在 getCenter 中，使用了三角形三点坐标合除以三作为中心。

在 getHitRecord 中，需要获取三角形的法向量，通过对三角形两边求叉积，并标准化完成。

球体 Hitable 实现简介

球体是渲染中，比较好看，且容易实现的物体。

对于球体的 hit 函数，实际上是求解了光线参数方程与球体表面方程联立结果。两方程联立之后，得到一个一元二次方程，通过求根公式，并判断根是否合理，完成对球体的求交。

对于球体 AABB 包围盒的生成，非常简单，使用球体中心的坐标，加上(1,1,1)向量乘以球体半径的向量，就得到了球体包围盒右上点的坐标，减去刚才所述的向量，就得到了左下点的坐标。

对于 getCenter 方法，返回球心坐标即可。

加速结构 BVH 的 Hitable 实现简介

为了加速结构中的一致性，将 BVH 包围盒同样视作一个 Hitable。

Bvh 的结构记录中，分别记录了左孩子与右孩子（均为 Hitable）的引用，以及本级 BVH 的 aabb 包围盒信息。

对于加速结构的 Hit，首先判断是否命中本加速结构的包围盒，若每命中，则一定没有命中此加速结构。若命中，则分别去与左孩子与右孩子进行求交，若命中，则返回两者中距离更近，且不为负的一个。对于包围盒的命中，实际上是通过分别求取了三个坐标在包围盒内是，直线参数方程 t 的求值范围，通过判断求值区域合法与否来判读是否与包围盒相交的，具体实现可以参考 sceen.aabb 中的 AABB 类的 hit 方法。

对于加速结构的 getHitRecord 调用，是不合法的调用，按照正常思路，是不可能调用到这个方法的。

对于加速结构的 generateAABB 方法，会返回此加速结构的 AABB 包围盒。

对于加速结构的 getCenter 方法，会返回此加速结构的 AABB 包围盒中心，但是此方法同样不应被调用，属于不合法调用。

总结及 TodoList

本渲染器已经实现了对一般渲染功能较为基本的支持，如纹理化的表面材质系统，反射，折射，漫反射，球体曲面，三角面的支持，且有了一定的性能基础。对比商业级渲染器，得到的结果可以接受，产生的画面，观感上，也比较的令人舒适。不过仍有非常非常多的改进方向，包括但不限于：

1. 增加动画支持，如骨骼动画，或者序列 OBJ 支持
2. 优化现有 BVH 构建策略，引入 SAH 等启发式的构建方法
3. 优化文件 IO 支持，使用更为标准，更为通用的，兼容的格式
4. 优化性能，使用 CUDA/OPTIX/Vulkan 等 GPU 编程接口大幅提高性能
5. 提供 GUI 的前端客户程序，以方便使用
6. 支持自定义程序化纹理节点
7. 支持体素（体积纹理），以实现烟雾等非实体的渲染支持
8. 支持高级渲染效果，如次表面散射，各向异性等
9. 使用 BDRF 等概率论方法优化采样效率，优化渲染性能。

参考

1. Ray Tracing in One Weekend.
   raytracing.github.io/books/RayTracingInOneWeekend.html