

**本科生毕业设计[论文]**

**基于PBR的渲染器的设计与实现**

院 系 软件学院

专业班级 软件工程1605

姓 名 杨文翔

学 号 U201617117

指导教师 万琳

2020年 5月 4日

**学位论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 年 月 日

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密囗，在 年解密后适用本授权书

2、不保密囗 。

（请在以上相应方框内打“√”）

作者签名： 年 月 日

导师签名： 年 月 日

# 摘 要

目前进行图形学学习总会绕不开了解PBR，但仅从理论上，比较难以理解PBR与基于传统光照模型渲染的区别。工业上，尤其是游戏行业，PBR是图形学程序员们避不开的一环。游戏引擎的尖端研究重点都集中于PBR上。故本次毕业设计计划设计制作一个PBR渲染器，在提供基本的基于物理渲染的基础上，设计一个能将PBR与传统光照模型渲染进行对比的功能。本文设计了基于物理渲染的渲染器软件，目标程序可以进行基于物理的渲染和传统光照模型的渲染。支持多种光源和基础物体的生成，同时能够通过相机进行场景的观察。为了让渲染结果更加真实，在渲染器中将会实现PBR算法，实现全局光照(Global Illumination)的效果。同时为了保证渲染的实时性，将使用基于图像的光照(IBL)来加速光照计算过程，在快速的同时又让渲染结果真实。

本文针对PBR渲染器的需求、设计与实现进行了分析与阐述。首先，阐述需求确定了渲染器的功能与性能。然后，按照需求对渲染器进行了设计与实现。在阐述PBR实现的过程中，对渲染方程的求解、Cook-Torrance BRDF模型和IBL技术进行了分析，并在分析的基础上对算法实现进行了说明。

本次课题基本上完成了需求中的所有功能，并通过了最终的功能测试与性能测试。在课题过程中，熟练了图形API(OpenGL)的使用和掌握了全局光照技术的相关算法。

**关键词**：实时渲染 全局光照 基于图像的光照 基于物理的渲染

# Abstract

At present, learning graphics can't avoid understanding PBR, but theoretically, it is difficult to understand the difference between PBR and rendering based on traditional lighting models. In industry, especially in the gaming industry, PBR is an inevitable part of graphics programmers. The focus of cutting-edge research on game engines is focused on PBR. Therefore, this graduation project plans to design and produce a PBR renderer. On the basis of providing basic physical rendering, design a function that can compare PBR with traditional lighting model rendering. This paper designs the renderer software based on physical rendering. The target program can perform physical-based rendering and traditional lighting model rendering. Support the generation of multiple light sources and basic objects, and at the same time be able to observe the scene through the camera. In order to make the rendering results more realistic, the PBR algorithm will be implemented in the renderer to achieve the effect of global illumination (Global Illumination). At the same time, in order to ensure the real-time rendering, image-based lighting (IBL) will be used to speed up the lighting calculation process and make the rendering results true at the same time.

This article analyzes and expounds the requirements, design and implementation of PBR renderer. First, the requirements are defined to determine the function and performance of the renderer. Then, the renderer is designed and implemented according to the requirements. In the process of explaining the implementation of PBR, the rendering equation solution, Cook-Torrance BRDF model and IBL technology are analyzed, and the algorithm implementation is explained on the basis of the analysis.

This topic basically completed all the functions in the requirements, and passed the final functional test and performance test. In the course of the subject, he was proficient in the use of graphics API (OpenGL) and mastered the relevant algorithms of global illumination technology.

**Key words：**Real-Time Rendering Global Illumination

Image-Based Lighting Physically Based Rendering:

目录

[摘 要 3](#_Toc40739535)

[Abstract 4](#_Toc40739536)

[**1** **绪论** 6](#_Toc40739537)

[**1.1** **课题背景** 6](#_Toc40739538)

[**1.2** **国内外研究现状** 6](#_Toc40739539)

[**1.3** **课题内容和目的** 7](#_Toc40739540)

[**1.4** **相关技术概念** 8](#_Toc40739541)

[**2** **PBR渲染器需求分析与设计** 10](#_Toc40739542)

[**2.1** **系统功能要求** 10](#_Toc40739543)

[**2.2** **系统性能需求** 12](#_Toc40739544)

[**2.3** **扩展要求** 12](#_Toc40739545)

[**2.4** **总体设计** 13](#_Toc40739546)

[**2.5** **详细设计** 14](#_Toc40739547)

[**2.6** **本章小结** 18](#_Toc40739548)

[**3** **PBR渲染器实现** 19](#_Toc40739549)

[**3.1** **开发环境** 19](#_Toc40739550)

[**3.2** **用户界面的实现** 19](#_Toc40739551)

[**3.3** **核心渲染部分实现** 24](#_Toc40739552)

[**3.4** **系统测试** 37](#_Toc40739553)

[**3.5** **本章小结** 40](#_Toc40739554)

[**4** **总结与展望** 41](#_Toc40739555)

[**4.1** **总结** 41](#_Toc40739556)

[**4.2** **展望** 41](#_Toc40739557)

[致 谢 43](#_Toc40739558)

[**参考文献** 44](#_Toc40739559)

1. **绪论**

本章对论文的课题背景、国内外研究现状和课题内容与意义做了阐述。并简单介绍了以下本次课题中将会使用的相关技术。

* 1. **课题背景**

在基于物理的渲染PBR（Physically Based Rendering）的概念出现前，若想渲染出一张高质量的图，需要机械化的死记各种参数，反复调节光照来让材质表现效果看上去比较真实。PBR之前的光照模型需要许多实践来得出看起来真实的结果。PBR的目的是利用更接近实际物理理论的光照计算，生成比以前的Phong、Blinn-Phong算法更真实的画面。基于物理的渲染其本质是在有限资源下，光学公式的理论与应用。PBR所能做的不只是让渲染画面更加真实，它还能让用户（尤其是美术工作者）通过调节几个物理属性就可以调节出想要的结果，而不必担心光照错误。

目前进行图形学学习总会绕不开了解PBR，但仅从理论上，比较难以理解PBR与基于传统光照模型渲染的区别。工业上，尤其是游戏行业，PBR是图形学程序员们避不开的一环。游戏引擎的尖端研究重点都集中于PBR上。故本次毕业设计计划设计制作一个PBR渲染器，在提供基本的基于物理渲染的基础上，设计一个能将PBR与传统光照模型渲染进行对比的功能。

* 1. **国内外研究现状**

2010年在SIGGRAPH上有公开讨论的Course，《SIGGRAPH 2010 Course: Physically-Based Shading Models in Film and Game Production》。

在离线渲染领域，迪士尼在创作电影《无敌破坏王》期间，他们对基于物理的渲染进行了系统的研究，最终开发出了迪士尼原则的BRDF（Disney Principled BRDF），这种BRDF模型几乎可以用于电影的每个表面。迪士尼动画工作室的Brent Burley于SIGGRAPH 2012上发表了《Physically-based shading at Disney》，正式提出了迪士尼原则的BRDF（Disney Principled BRDF）。该原则将材质复杂的物理属性用金属度、粗糙度等参数抽象简单化表达出来。在2012年，迪士尼又提出了IBL（Image Based， Lighting）技术，将其引入PBR渲染管线中，极大地提升了光照的真实性。

在实时渲染领域，Epic Game在2013的SIGGRAH上发布了《Real Shading in Unreal Engine 4》，介绍了他们在虚幻4中引入基于物理的渲染。他们提出对PBR在自家引擎中实现的几点需求：实时性能，需要在每次许多可见灯光的条件下高效地使用；简化复杂度，参数尽可能的少；直观的界面，更倾向易于理解的值，而不是像折射率这样的物理参数；感知线性，希望通过蒙版支持分层；简单掌握，要避免需要对电介质和导体的技术理解，同时最小化创建基本的貌似物理的材质所需要的努力；健壮，·参数的所有合并应该尽可能的健壮和可信；表达力强，基本着色模型需要足够描述覆盖显示世界中99%的材质；灵活，·需要足够灵活来允许非真实感渲染。由此，Epic提出了分解求和近似（split-sum approximation）的方法，这种方法大幅度地简化了环境光的高光计算，提供了PBR实时渲染的可能性。Unity在GDC 2014上做了《Physically Based Shading in Unity》的分享，介绍了Unity游戏引擎中使用的PBR技术。

* 1. **课题内容和目的**

对于图形学，其中重要的研究方向之一就是渲染。渲染简单来说，就是输入待处理的数据如顶点位置、纹理与材质等，计算机通过这些数据进行计算，最终输出一副数字图像。在本次的课题中，首先将实现使用传统光照模型进行渲染，能够观察到传统光照模型的渲染效果。对于渲染的输入物体方面，除了实现能够生成一些基础的物体形状之外应当实现对模型文件的解析与读取，同时还有物体纹理、材质等贴图的读取。为了让渲染结果更加真实，在渲染器中将会加入PBR算法，实现全局光照(Global Illumination)的效果。所谓的全局光照，是指被渲染场景中光源跟物体表面交互产生的直接光照和物体之间相互反射得到的间接光照的集合。它能够模拟光线从光源经过到达场景中的每个物体的过程，从而让光照更加接近自然光照现象。这样的渲染技术可以在虚拟场景中创造极强的真实感与沉浸感。课题中的渲染器将实现这样的基于物理的渲染。同时，本次课题的一个目标是设计一个能让用户在使用过程中学习PBR技术的渲染器，所以课题中将PBR算法的实现将拆分成三步，每一步完成PBR的部分计算，由此来展示传统光照模型到PBR的转换过程，最终转换为一个功能齐全的PBR渲染器。该渲染器提供场景编辑和保存，可用于对模型的PBR材质预览。为了让用户的操作能够得到及时的反馈并且能够快速浏览到渲染结果，渲染器实现的是实时渲染。在实时渲染中，画面出现卡顿和操作反馈的延迟会给用户带来糟糕的体验。因此大多数的实时渲染器都只计算了直接光照而没有间接光照的计算，这样导致了实时渲染的画质略低。但本次课题中的渲染器将会在保证实时性的前提下，引入对间接光照的计算从而实现全局光照，使画面更加真实。

本次毕业设计开发的渲染器面向于图形学PBR部分入门学习的用户，也能用于快速展示PBR的渲染效果，是一款轻量渲染器。在本次课题中，可以掌握目前主流的图形渲染技术，并且在实现过程中熟练图形API(OpenGL)的使用。更加关键的是，在课题过程中将能够掌握常用的全局光照算法并将其与实时渲染结合，实现出一个基于物理的实时渲染器。在这个过程中，可以进一步了解工业上对于全局光照的发展趋势和解决方案。本次毕业设计能在学习研究PBR算法的同时，实现一个基于预计算的PBR实时渲染器。并在实现过程中探索PBR在实时渲染中的优化和游戏引擎中对场景数据的组织与管理的方式。

* 1. **相关技术概念**
     1. **OpenGL**

OpenGL（Open Graphics Library）是用于渲染2D、3D图形的跨语言、跨平台的应用程序编程接口（API）。OpenGL规范描述了绘制2D和3D图形的抽象API。尽管这些API可以完全通过软件实现，但它是为大部分或者全部使用硬件加速而设计的。OpenGL有许多语言绑定，值得一提的包括：JavaScript绑定的WebGL（基于OpenGL ES 2.0在Web浏览器中的进行3D渲染的API）；C绑定的WGL、GLX和CGL；iOS提供的C绑定；Android提供的Java和C绑定。图1-1为OpenGL的渲染管线示意图。

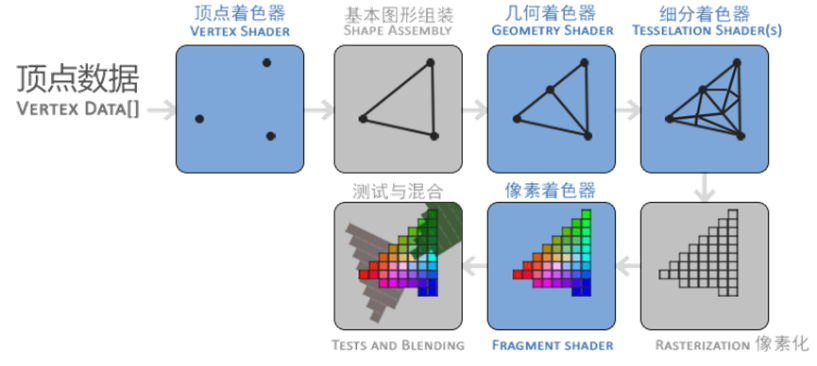


图1-1 OpenGL渲染管线

* + 1. **Qt**

Qt是一个跨平台的C++应用程序开发框架, 广泛用于开发GUI程序。使用Qt开发的软件，相同的代码可以在任何支持的平台上编译与运行，而不需要修改源代码。会自动依平台的不同，表现平台特有的图形界面风格。Qt利用信号与槽（signals/slots）机制取代传统的callback来进行对象之间的沟通。Qt5中加入了QtOpenGL模块，提供在应用程序中使用OpenGL和OpenGL ES加入3D图形。

* + 1. **基于物理的渲染(Physically based rendering,PBR)**

基于物理的渲染是计算机图形学中的一种方法，旨在以更准确地模拟现实世界中光的传播方式来渲染图形。 许多PBR渲染管线都以精确模拟真实感为目标。 双向反射率分布函数和渲染方程的可行和快速近似在该领域中具有重要的数学意义。 摄影测量法可用于帮助发现和编码材质的准确光学特性。 着色器可用于实现PBR原理。图1-2为基于PBR原理的渲染效果。



图1-2 iggraph13上EpicGames 的UE4引擎展示的PBR渲染结果

1. **PBR渲染器需求分析与设计**

本章对PBR渲染器整个系统进行全局分析，根据目标平台的硬件条件、基于物理渲染的普遍要求和实时渲染的普遍要求确定该系统的具体需求。主要从系统功能要求、系统性能要求、运行要求和扩展要求四个方面对目标应用程序进行需求分析。

* 1. **系统功能要求**
     1. **系统功能概述­**

本PBR渲染器的目标用户为图形学初学者。渲染器的开发目的是为了让初学者能够通过使用本PBR渲染器，能够对PBR中的各个阶段的计算所能提供的渲染效果有个清晰的认识，在使用的过程中学习到PBR的相关知识并且能够直观地看到PBR渲染方式与传统光照模型的区别。同时，在渲染对象方面，应该为用户提供几种基础的形体并且同时可以允许用户加载用户提供的模型与材质贴图。综上，为该系统提出以下功能需求，如表2-1所示：

|  |  |
| --- | --- |
| 支持传统光照模型 | 渲染器应支持基础的传统光照模型的渲染方式，用于与基于物理渲染的对比。 |
| 光源 | 渲染器应提供不同种类的光源类型，并且对每个光源可以进行位置的移动、朝向的变动和颜色改变等对光源的详细设置。 |
| 相机 | 相机可以在场景中进行移动与旋转，使用户能够自由观察创建的场景 |
| 天空盒的设置 | 为了让渲染结果更加真实，应让用户对天空盒进行设置，并将天空盒反应到物体的渲染上。 |
| 基础形状与模型导入 | 为了方便用户观察渲染效果，应提供几种基础形状供用户进行渲染。同时，用户也可以用渲染器加载已有的模型来进行渲染。 |
| 基于物理的渲染 | 本渲染器最重要的功能，用户可以通过功能按键来开启基于物理的渲染模式。其中基于物理的渲染应分成几个部分的功能，以渐进式开放给用户，使用户能感受到每一步计算的进步。 |
| PBR教学 | 应有对PBR拆解出的每一个功能有一份教学文档，可以直接在渲染器中查看，每完成一份文档的学习才可解锁对应的PBR功能。 |
| 渲染物体的材质设置 | 对于传统光照模型应有漫反射材质贴图和法线贴图的设置，当无贴图时可以向用户提供对物体颜色进行修改的功能。对于PBR，应提供反射度贴图、粗糙度贴图、金属度贴图和AO贴图的设置，同样当没有这几种贴图时应提供直接数值的修改功能。 |

表2-1 功能需求表

* + 1. **系统用例图**

用户将能对渲染器的相机、场景和光源进行控制；对于被渲染物体，用户可以进行创建并且对创建的物体进行各种设置；用户还可以开启文档阅读并且进行功能解锁。管理员需要与系统交互为对文档的更新。系统用例图如图2-1所示。

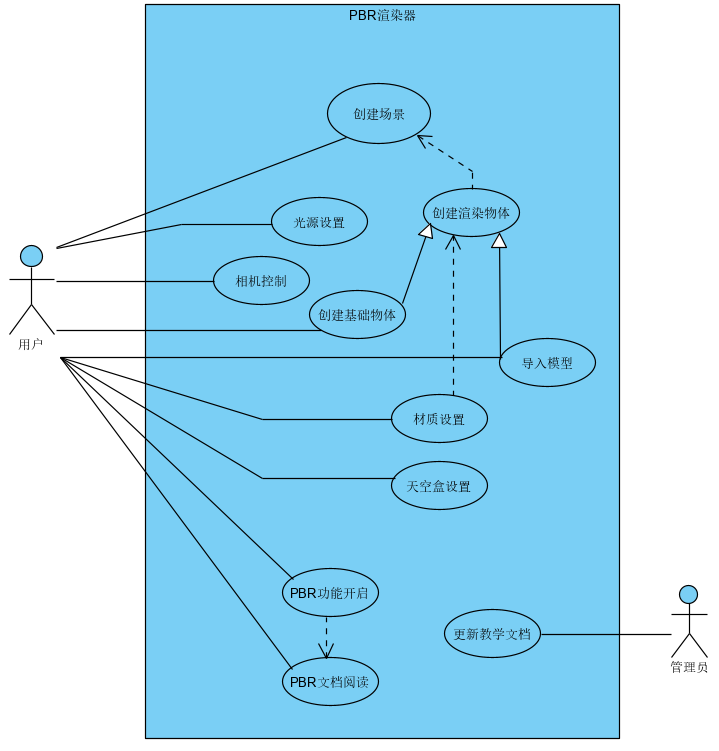
****

图2-1 系统用例图

* 1. **系统性能需求**

考虑到目标用户所使用硬件设备的多样性，并且本渲染器提供的是实时渲染对帧数要求比较高，对性能的需求相对来说比较严格。由于用户使用的显卡是多种多样的，渲染器应当最大程度上为用户的硬件做出妥协，在性能较差的显卡上也能保证在渲染出足够好的画面的同时满足实时性的要求。综合考虑之下，对系统性能做了以下要求，如表2-2所示：

|  |  |
| --- | --- |
| 基础物体生成与模型读取 | 基础物体的生成反应时间应当小于等于0.5s。由于不同的模型其顶点数不同，其加载的时间也有所不同。对于三角面数小于等于10000的模型要求加载时间应不长于0.5s；对于大于10000小于50000的模型，加载时间不长于1s；50000以上的模型不做具体的要求。 |
| 帧数 | 对于硬件平台中独立显卡的用户，要求渲染器的帧数应该达到60以上；对于硬件平台为核芯显卡的用户，渲染器的帧数应达到30以上。 |
| 响应 | 对于相机的操作，要求响应应在0.1s以内；对于场景物体的移动旋转等顶点位置的改变操作，要求响应在0.1s以内；对于对场景物体的贴图等材质属性的修改操作，响应应在0.5s以内；对于天空盒的设置，响应应在0.5s以内。 |

表2-2 性能需求表

* 1. **扩展要求**

对于渲染器的后续开发，可以提出以下几点扩展性要求：

支持多种模型文件的读取与渲染，并在读取后自动为其加载对应的正确贴图。目前广泛使用的模型格式有OBJ、3ds、fbx等，对于不同的格式需要有不同的加载方式。

让用户可以自定义光源数量，并在光源过多的情况下能够进行延迟渲染。当光源过多时，正向渲染会比较消耗性能，使用延迟渲染的方式可以减少许多不必要的光线计算以提高性能。

增加阴影计算的功能，并可以展示一些预计算的结果或是中间计算结果，比如：深度图、IBL的预计算等。

* 1. **总体设计**

渲染器的应用程序架构采用MVC架构模式，分为Model、View和Controller三部分。这样构建的好处是让用户界面的显示与渲染器中的数据和渲染过程分离开来，在测试时可以单独对处于model层的渲染模块进行测试。同时，也将用户的操作与用户界面显示分离开，在测试时同样可以对controller层中的交互逻辑进行单独测试。MVC架构使得渲染器的各个部分的耦合性降低，便于测试、维护和管理。下面对渲染器中每一个部分的作用进行描述：

**Model**

Model负责渲染器的渲染逻辑、渲染场景数据和功能开放管理的部分。Model将对Controller发送来的用户操作进行处理并反应到场景物体、相机和着色器上。

**View**

View负责将实时渲染好的画面显示给用户和用户界面的管理，对用户操作的响应进行显示。同时，View将用户的操作事件传递给Controller，让Controller对用户操作输入进行处理。

**Controller**

Controller负责用户交互的逻辑部分，对用户的输入进行响应并将输入传入Model中进行处理。

图3-1为MVC架构参考图。

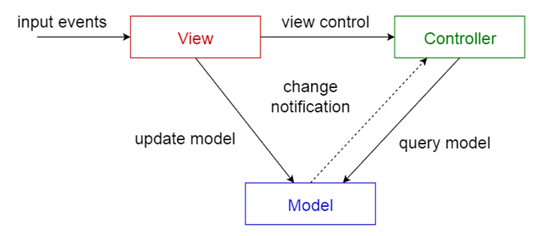


图3-1 MVC架构图

渲染器的渲染流程为：用户创建渲染场景；渲染器根据用户的功能开启情况选择相应的着色器和进行一些预处理计算；然后读取场景物体的数据进行光照计算；最终渲染出当前一帧的图像。流程图如图3-2所示。

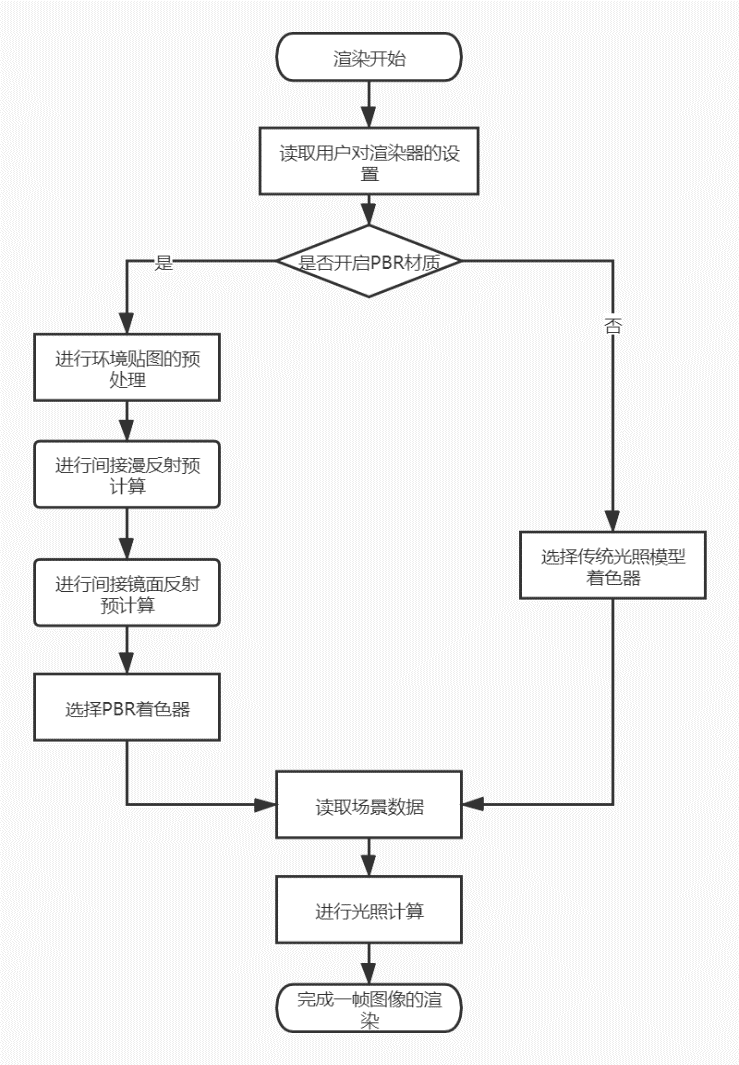


图3-2 渲染流程图

* 1. **详细设计**

根据总体设计中的PBR渲染器的架构和功能需求，对PBR渲染器进行模块划分，并在每个模块中根据功能需求进行详细的类设计。

* + 1. **模块设计**

PBR渲染器的业务逻辑根据功能需求主要划分为以下几个模块：

**渲染模块**

渲染模块为PBR核心模块，它调用OpenGL进行渲染工作，负责对整个场景进行渲染计算，同时还需要将渲染结果递交给界面模块进行显示。渲染模块的渲染方式有PBR模式和传统模式，这两种模式中还分为有贴图版本和无贴图版本，渲染方式根据用户使用时的设置改变。

**场景数据模块**

场景数据模块负责管理场景中需要渲染的物体数据，并将这些数据传递给渲染模块进行渲染。同时，场景数据模块还需要管理模型的加载过程，让模型正确地加载到场景中。

**显示模块**

显示模块负责显示渲染模块传递来的渲染结果，并且负责用户界面的显示。

**控制模块**

控制模块负责接收用户的输入并将输入传递给场景数据模块和渲染模块。用户对相机的输入和对场景物体的输入传递给场景数据模块；用户对渲染功能选择的输入应当传递给渲染模块，让渲染模块进行变更。

**学习文档模块**

学习文档模块为PBR渲染器中提供用户进行PBR的学习，是一个文档阅读器。学习文档模块需要将用户的学习进度传递给显示模块，让显示模块能根据用户的学习进度进行功能的解锁。

系统的模块图如图3-3所示。

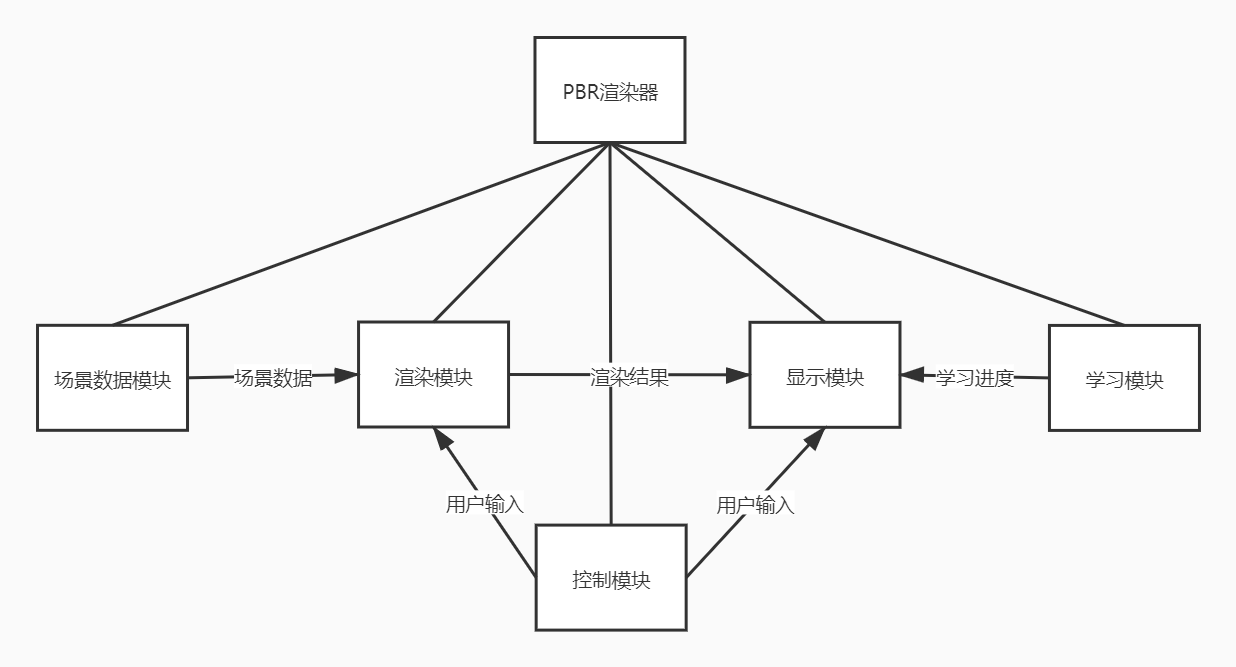


图3-3 系统模块图

* + 1. **类设计**

由于本PBR渲染架构采用的MVC架构，下面详细介绍围绕MVC架构的核心类：

**Render类**

Render类将继承Qt提供的QOpenGLWidget类和QOpenGLFunction类。QOpenGLWidget类是Qt中用于OpenGL渲染的类，它封装了一个OpenGL窗口，Render通过继承这个类来将渲染结果显示出来。QOpenGLFunction类是Qt对OpenGL原生代码的封装，通过继承它可以直接使用Render类去调用OpenGL的各种函数来进行渲染。Qt提供了不同版本的QOpenGLFunction来针对不同版本的OpenGL，Render类将继承QOpenGLFunctions\_4\_4\_Core来使用4.4版本的OpenGL。Render类属于MVC中的Model部分。Render类图如图3-4所示。

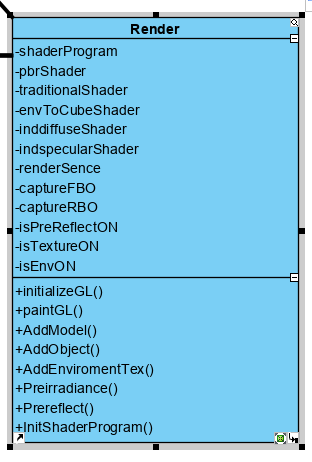


图3-4 Render类类图

**View类**

View类将继承QMainWindow类来进行用户界面的显示和渲染结果的显示。View类将接收Render类传递来的渲染结果并将结果显示在用户界面上。View类属于MVC的View部分。View类图如图3-5所示。

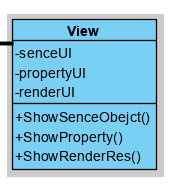


图3-5 View类类图

**Operation类**

Operation类负责接收所有的用户输入，并将输入交给Render类进行处理。Operation中有对相机的位置、方向的控制；用户对PBR渲染器功能选择的输入；用户对场景物体的增添和属性修改的输入。Operation类属于MVC中的Controller部分。Operation类图如图3-6所示。

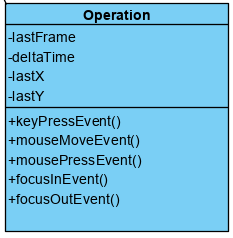


图3-6 Operation类类图

PBR渲染器的功能核心类为执行渲染的Render类，为了让Render类能快速渲染出结果，设计了几个辅助类来辅助Render类对渲染数据的管理。下面详细介绍这几个辅助类：

**Sence类**

Sence类管理所有的场景数据，包括场景物体、光源和相机。Sence类还能处理用户的物体增添和属性修改输入。

**RenderObject类**

RenderObject类为抽象类，所以需要渲染的物体都应该继承自它，并且交给Scene类进行管理。

**Camera类**

Camera类主要提供光照计算中所需要的相机数据和提供顶点变换时需要的一些矩阵。

**Shader类**

Shader类负责所有Shader程序的创建、编译和链接，还负责把应用程序中对Shader的数据传递到Shader程序中。

PBR渲染器的整体类图如图3-4所示。

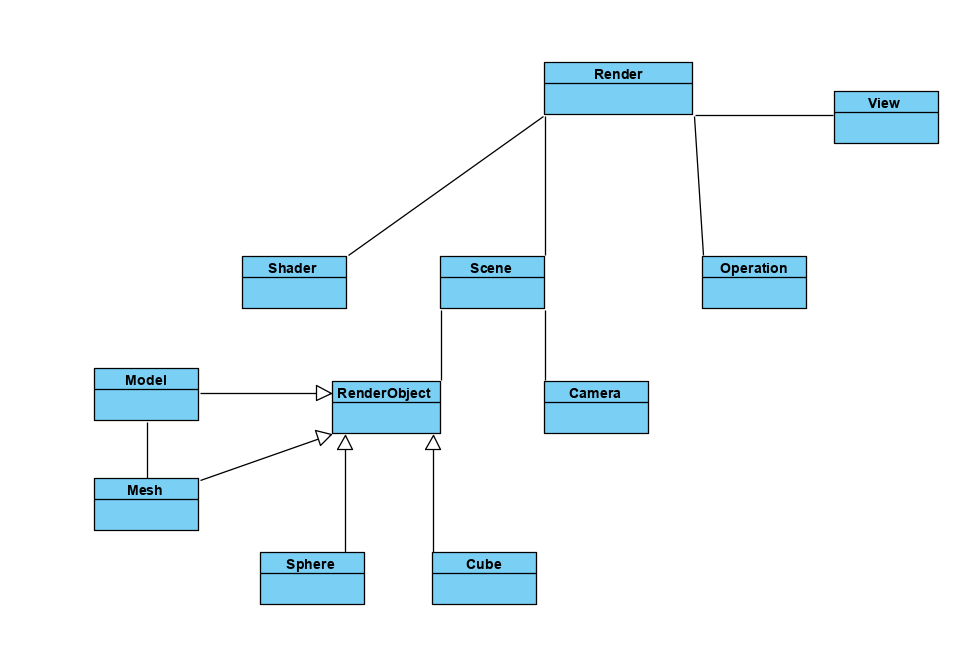


图3-7 PBR渲染器类图

* 1. **本章小结**

本章对整个渲染器做了需求上的约束。分别从系统功能需求、性能需求和扩展需求三部分对PBR渲染器系统进行了要求，并展示了系统用例图来解释用户与系统的交互与系统功能。然后对PBR渲染器进行了设计上的阐述。在总体设计中，对渲染器的架构进行了分析与设计。重点进行了详细设计。详细设计分为了模块设计与类设计进行分析与阐述，分别介绍了系统的模块划分与按功能对类进行阐述。

1. **PBR渲染器实现**

本章对PBR渲染器系统进行具体的实现，并对渲染器按照需求分析进行系统的功能测试和性能测试。

* 1. **开发环境**

PBR渲染器使用C++进行界面与渲染部分的开发。界面部分使用的是Qt 5.12.6作为图形界面框架，该框架提供了许多方便实用的界面开发组件，提高开发效率。渲染部分则是使用OpenGL4.4，现在主流GPU对该版本也同样有所支持。代码编辑使用了Visual Studio2019，在其中安装了Qt项目支持的插件，让开发中能够兼容Qt的同时可以使用该IDE丰富的功能。

在开发中，使用了glm数学库来辅助OpenGL中一些3D数学的数据结构的使用。在贴图读取方面，使用了stb\_image库来进行图片的读取，这个库支持了jpg、png等常见贴图文件读取的同时，还支持对环境贴图hdr文件的读取。在模型读取方面，使用了Assimp库来进行读取。

* 1. **用户界面的实现**
     1. **信号与槽机制**

信号与槽机制是 QT 的核心机制，是一种高级接口，应用于对象之间的通信，它是 QT 的核心特性，也是 QT 区别于其它工具包的重要地方。信号会在其所有者的某种状态发生改变时被发射，也可以自定发射条件，前提是在定义过这个信号的类及其派生类中进行发射。当一个信号被发射时，会触发与其相关联的槽，相关联的槽将立刻执行。信号与槽机制完全独立于任何 GUI 事件循环。只有当所有的槽返回以后发射函数（emit）才返回。 当某个信号与多个槽相关联，那么当这个信号被发射时，这些被关联的槽将会依次执行。槽是普通的 C++ 成员函数，可以被正常调用，它们唯一的特殊性就是很多信号可以与其相关联。槽可以有参数，与信号的参数相对应。

通过connect 函数可以将某个对象的信号与另外一个对象的槽函数相关联，这样当发射者发射信号时，信号关联的槽函数将被调用。Qt的信号与槽机制具有高度的灵活性：多个信号可以关联同一个槽；同时一个信号也可以关联多个槽；甚至一个信号可以关联另外一个信号。槽不用了解它是关联了哪些信号，只要使用connect函数关联信号与槽，Qt就保证了信号对应的槽会得到正确的调用。与之对应的，激发信号的Qt对象无须知道是哪个槽需要接收它发出的信号，它只需要做的是在状态发生改变时发送它自己的信号就可行，而不需要了解信号是否被接收到，更不需要了解槽函数对它的响应情况。这样的机制大幅度降低了程序中Qt对象的耦合度。

* + 1. **界面的具体实现**

本PBR渲染器的图形用户界面框架使用的是Qt，用户界面主要分为4个部分：顶部功能菜单栏，场景物体栏，属性栏和渲染显示界面。由渲染器的设计，用户界面在View类中进行实现，View类继承Qt提供的QMainWindow类。QMainWindow有Qt预先定义好的布局，可以在其中添加QToolBars、QDockWidgets、QMenuBar和QStatusBar等界面容器。布局有一个中心区域，可以被任何类型的小部件占用。中心小部件通常是标准的Qt小部件，比如QTextEdit或QGraphicsView。定制小部件还可以用于高级应用程序。本渲染器中的中心区域为自定义Qt组件，是继承了QOpenGLWidget的渲染显示组件。QMenuBar处放置顶部菜单栏，QDockWidgets处放置继承了QDockWidgets的自定义组件属性栏和场景栏。QMainWindow的布局如图4-1所示。

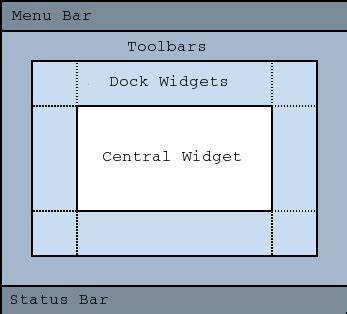


图4-1 QMainWindow布局

下面详细介绍View类中各个功能的实现。

**顶部菜单栏**

顶部菜单中，分为菜单项、功能解锁项和学习项。菜单项中有新建、打开和保存场景的功能按键；同时还有创建基础物体和导入模型的功能按键。在功能解锁项中，根据需求和设计将PBR功能划分为渐进式地3部分，PBR材质、间接漫反射和间接镜面反射；学习项将对应功能解锁项，提供开启每个部分文档的按钮。菜单栏中所有的按钮均用QAction实现。

QAction类可以作为抽象的用户界面对象被放置在窗口部件中。QAction可以被添加到QMainWindow中的菜单和工具栏中，并且可以自动保持在菜单和工具栏中的同步。QAction可以在其中存储许多有用的显示数据，本次渲染器中只用到它来存储它对应的文本数据，文本数据可以在构造函数中进行初始化。一旦QAction被创建了，那么就必须将它添加到相关的菜单和工具栏上。然后，可以使用connect函数将它们链接到实现相应点击功能的槽函数上。

在PBR渲染器中将会用到QAction信号中的triggered信号。当点击界面菜单栏上的按钮时，对应的QAction会发出triggered()信号。在View类中，编写对应的槽函数，然后使用connect函数连接信号和槽。

**场景物体栏**

场景物体栏中将显示尝尽里的所有物体，物体将呈树形显示。同时，在场景物体栏还有设置天空盒和光源属性的菜单项。

场景栏的实现使用的是Qt提供的QTreeView，它使用Model/View结构来管理数据与表示层的关系。QStandardItemModel是QTreeView的对应模型。它负责保存数据，每个数据项被表示为类QStandardItem的对象。根据QTreeView的结构，类QStandardItemModel将类QStandardItem的数据组织成树状数据结构。

QStandardItem中的数据类型为QVariant。QVariant是一个最为普遍的Qt数据类型的联合，其用法与C++中union的用法差不太多。为了将View类中场景物体的显示能与Render类中场景物体连接起来，QStandardItem中的QVariant中将存储场景物体对应的指针。当用户点击场景栏中的物体时，可以获取对应的物体指针，通过这个指针就可以访问和操作该物体的属性。

在场景物体栏种还能够进行天空盒和光源属性的设置，这个页面与场景物体页面以标签页的方式进行组织。这个标签页为Qt提供的QTabWidget组件。天空盒的设置为用户在用户界面中进行HDR文件的选择，然后View类将文件地址传递给Render类进行处理，具体实现函数为：AddEnviromentTexture。光源属性则是View类接收到数值改变的信号后，直接发送更改信号，将界面上设置的数值传递给Render类中的槽函数进行实时的修改。

**属性栏**

属性栏将显示用户点击的物体对应的物体属性，属性的读取将通过点击的QStandardItem中存储的指针数据进行。属性栏的显示由Qt提供的QLineEdit、QSlider和QPushButton组件进行实现。

对于每种贴图的添加，在添加后还需要在属性栏种显示贴图的预览。贴图的预览将用QLabel实现。在通过用户提供的地址读取了贴图图片后，用QPixmap对贴图数据进行放缩，使其缩小到预定好的QLabel大小，然后将QLabel的内容设定为处理好的QPixmap。

**渲染显示界面**

该界面将直接显示Render类中的渲染结果，Render类的实现将在后续讲解。

最终，PBR渲染器的用户界面的呈现效果如图4-2所示。

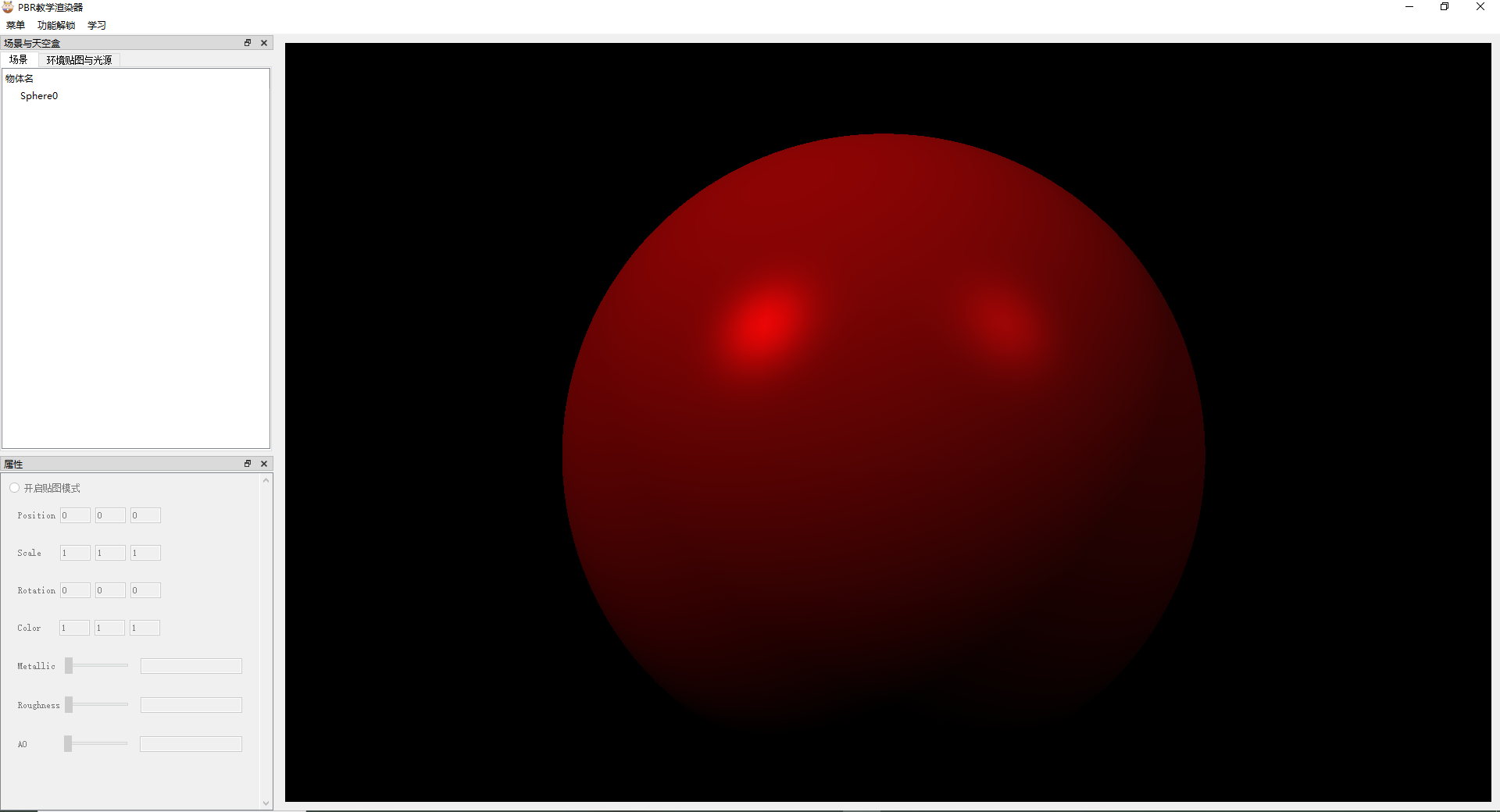


图4-2 用户界面

* + 1. **用户界面中的操作实现**

用户界面与用户操作通过信号与槽机制进行连接。

**菜单栏**

菜单项全部是用QAction进行实现。所以需要响应用户操作触发的triggered()信号，以下为Operation类中对该信号响应的槽函数。

AddSphere：用于响应在场景添加球体

AddCube：用于响应在场景添加立方体

AddModel：用于响应在场景加载用户指定的模型

UnlockMaterial：用于响应PBR材质功能的解锁

UnlockDiffuse：用于响应间接漫反射光照功能的解锁

UnlockSpecular：用于响应间接镜面反射光照功能的解锁

ShowPartOne：用于响应学习文档1的显示

ShowPartTwo：用于响应学习文档2的显示

ShowPartThree：用于响应学习文档3的显示

**场景物体栏**

场景物体栏中的项是QTreeView进行组织。当用户点击其中任意一项时，会触发QTreeView的clicked()信号，这个信号同时会传递一个点击项在树中的索引。槽函数需要接收这个索引并且根据索引在树中找到对应项来进行下一步处理。这个信号对应的槽函数为：

ShowProperties：在这个槽函数中，将获取点击到的项中存储的物体指针，这个指针将会传递给Render类进行选中物体的设置。

**属性栏**

实现用户通过QLineEdit和QSlider进行属性的设置。当用户对QLineEdit进行属性修改时，将会触发QLineEdit的textChange()信号，以下为对该类信号响应的槽函数：

ChangePosition：对物体位置属性的修改

ChangeScale：对物体大小的修改

ChangeRotation：对物体朝向的修改

ChangeColor：无贴图时对物体颜色的修改

ChangeMetallic：无贴图时对物体金属度的修改

ChangeRoughness：无贴图时对物体粗糙度的修改

ChangeAO：无贴图时对物体AO的修改

属性栏中，金属度、粗糙度和AO项同时存在着QSlider和QLineText两种属性值的表示方式，所以需要将两者的数值改变同步。以下为实现同步的槽函数：

MetallicEditFromSlider与MetallicSliderFromEdit

RoughnessEditFromSlider与RoughnessSliderFromEdit

AOEditFromSlider与AOSliderFromEdit

属性栏中，对于贴图添加的操作是通过QPushButton进行实现。当用户点击某种贴图的添加按钮时，将会触发QPushButton的clicked()信号。以下为该信号响应的贴图添加槽函数：

SetDiffuseButtonON

SetNormalButtonON

SetMetallicButtonON

SetRoughnessButtonON

SetAOButtonON

属性栏中还有一个用QRadioButton实现的贴图开关，其点击信号为clicked()，需要响应这个信号来对Render类中着色器的选择布尔变量进行设置。同时还得开放属性栏的贴图按钮。该信号的响应槽函数为：

SettextureON

SetProperties

* 1. **核心渲染部分实现**
     1. **相机与被渲染物体**

相机为渲染场景中的观察者，它能够在场景中进行自由的移动与旋转来对场景进行观察。对于被渲染物体，它能够被用户进行位置、大小和朝向的设置。这些对于位移、旋转和缩放的操作，在计算机图形学中通常使用四阶矩阵进行变换。图4-3为一个基本的矩阵坐标变化：

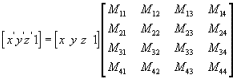


图4-3 矩阵坐标变化

对于缩放操作，其变换矩阵的样式应如4-4图所示：



图4-4 缩放矩阵

对于平移操作，其变换矩阵的样式应如图4-5所示：



图4-5 平移矩阵

对于旋转操作，将分为x轴、y轴和z轴3种不同的变化矩阵，如图4-6所示，其中θ代表旋转的角度：

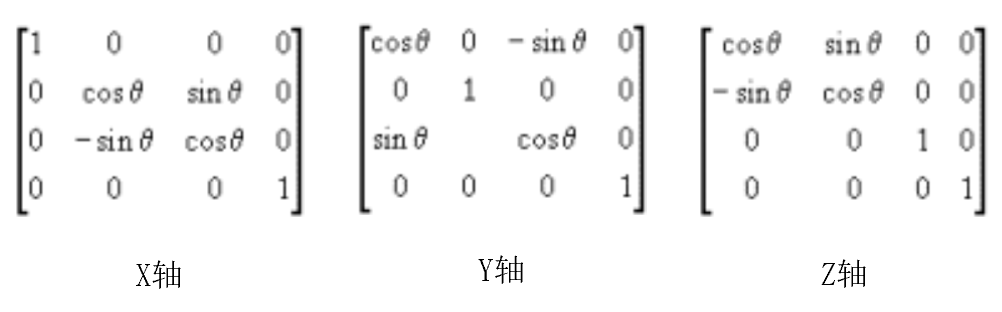


图4-6旋转矩阵

对相机和物体进行位移、旋转和缩放操作时，实质上是将以上3种矩阵进行矩阵相乘来得到最终的变换效果。当在进行矩阵相乘时还需要注意顺序，在实际运行的时候变换顺序是与相乘顺序相反，也就是最终结果的矩阵效果是等同于每个矩阵的效果从左至右依次出现的。

对于相机的旋转，还得将x轴、y轴和z轴的旋转角该称为3种欧拉角：俯仰角(Pitch)、偏航角(Yaw)和滚转角(Roll)，如图4-7所示：

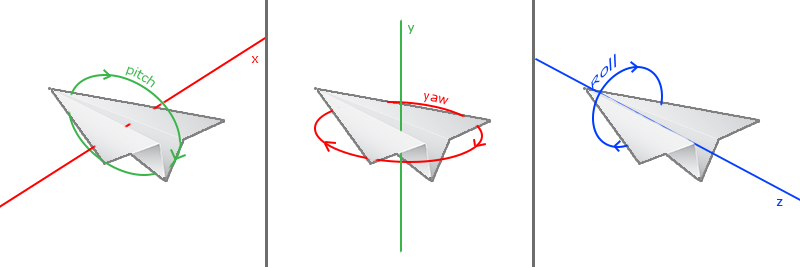


图4-7 三种欧拉角

偏航角表示左右旋转的角，俯仰角是描述上下旋转的角，滚转角则代表摄像机翻滚旋转的角。每个欧拉角都有一个值来表示，把三个角结合起来就能够计算在3D空间中任何的旋转向量从而得到旋转矩阵。对于本渲染器中的相机系统来说，只需要关心俯仰角和偏航角。根据鼠标的输入，将在渲染循环中计算与上一帧鼠标位置的偏移量。根据OpenGL的屏幕空间坐标系的排布，x轴的偏移量对应着偏航角的变化，y轴的偏移量对应着俯仰角的变化。将偏移量乘以一个灵敏度，就可以来进行相机的旋转调节。对于相机的位移与被渲染物体的位移和缩放，则直接进行矩阵乘法即可完成。

* + 1. **模型的读取**
       1. **OBJ文件格式**

OBJ文件是一种在 3D软件模型之间的互导过程中是最不容易出错的一类标准3D模型文件格式，这得力于OBJ文件是一种文本文件。OBJ文件由一行行文本组成，每行由关键字开头来标记当前行的数据类型。OBJ中的重要关键字如下：

v: 表示这行数据为一个顶点的x、y、z坐标值，每个坐标值为单精度浮点数

vt: 表示这行数据为纹理坐标的uv值，每个坐标值为单精度浮点数

vn: 表示这行数据为法向量的x、y、z坐标值，每个坐标值单精度浮点数

g: 表示组，后跟参数为组名称，指定从此行之后到下一个以g开头的行之间的所有元素将整合为一组

f: 表示面，即一个三角形图元

mtllib: 此关键字后参数为文件名称，指定了obj文件所使用的材质库文件(mtl文件)的文件名称

图4-8将展示一个基本的obj文件样式。

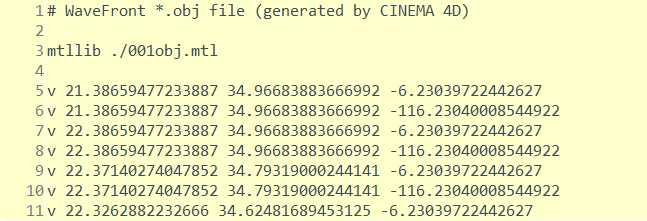


图4-8 obj文件示例

OBJ文件不包含顶点的颜色信息，这些颜色信息将储存在一个后缀是".mtl"的独立文件中。mtl文件是obj文件附属的材质库文件，其中包含材质的漫射(diffuse)，环境(ambient)，光泽(specular)的RGB的定义值，以及反射(specularity)，折射(refraction)，透明度(transparency)等其它特征。由此可见，OBJ材质库的是不支持诸如金属度(Metallic)、粗糙度(Roughness)和AO等PBR材质的。mtl文件使用的关键字如下：

Ka：材质的环境光（ambient color）

Kd：散射光（diffuse color）

Ks：镜面光（specular color）

Ke：放射光（emissive color）

Ns：材质发光亮度

sharpness：材质的锐度（sharpness）

illum：照明度（illumination）

图4-9将展示一个基本的mtl文件格式。

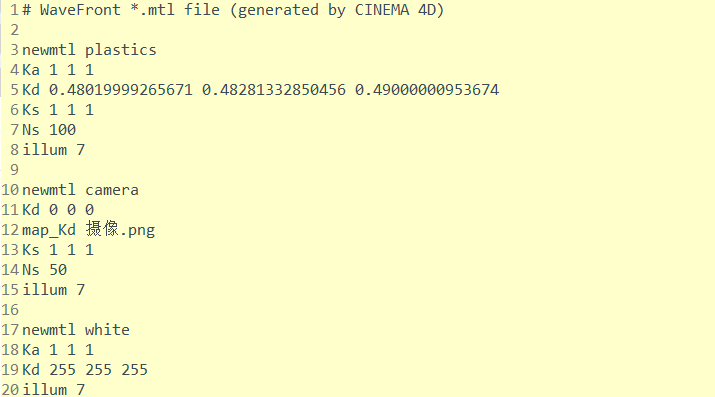


图4-9 mtl文件示例

虽然可以修改mtl文件，让一些不用的贴图位置来对应PBR材质，以实现自动读取贴图的功能，但还是很容易造成贴图错误。所以在本PBR渲染器中，自动读取普通材质贴图而PBR材质贴图需要用户自行手动添加。

* + - 1. **Assimp库的数据结构**

Assimp会将所有的模型数据加载至Assimp的一种通用数据结构中。不论导入的是什么种类的模型文件格式这个数据结构保持不变。这个通用数据结构中可以抽象多种不同的模型文件格式，用同一种方式访问需要的数据。Assimp数据结构的（简化）模型如图4-10所示：

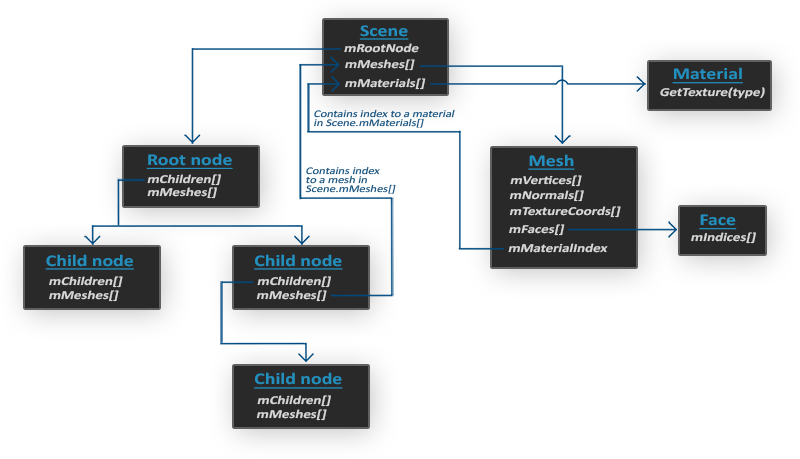


图4-10 Assimp数据结构

* + - 1. **Render中的实现**

当用户通过操作用户界面发送了添加模型的信号后，Render类中的AddModel槽函数进行响应。同时，该信号中应该传送模型的地址。由于渲染是实时进行的，如果实时将模型数据添加到场景物体数组中，将不能确定这个过程究竟发生在渲染的哪一步。这很有可能使得在渲染过程中读取到脏数据。所以，AddModel将会设置Render的一个添加模型的标志布尔变量，然后在下一帧渲染开始时将模型数据添加到场景物体数据数组中进行渲染。

对于通过Assimp读取的模型文件，将通过Model类和Mesh类进行处理。Model类中将存储模型的所有Mesh和贴图数组。Mesh中将会存储读取的顶点数据和索引数据来用于渲染。同时，在处理模型文件的过程中将会为每个Mesh创建VAO、VBO和EBO。然后，在渲染时，Render类将会统一调用Model的父类定义的Draw函数。由于是虚函数，实际上调用Model实现的Draw函数，在Model的Draw函数中将会负责将每个Mesh进行绘制。

* + 1. **基于物理渲染的实现**

**微平面模型**

能量守恒 微平面模型使用的不是物理学意义中的能量守恒(Energy Conservation)。微平面模型中的能量守恒只是要求对于非发光面射光线的能量不能超过入射光线的能量。为了遵守这样的一种能量守恒定律，需要对漫反射光和镜面反射光之间做出明确的区分。当一束光线与表面交互后，它将会分离成一束折射光线和一束反射光线。反射光线是不会进入表面的光线，即镜面光照。而折射光线就是余下的会进入表面并被吸收的光线，即漫反射光照。根据材料的不同，光线还有不同的表现。对于金属(Metallic)表面，当考虑反射与折射的时候需要注意金属表面对光的反应与介电质(Dielectrics)材料(即非金属材料)表面是不同的。虽然反射与折射原理是相同的，但是所有的折射光都会被直接吸收而不会散开，只留下反射光，这将会导致金属表面不会有漫反射颜色。

微平面法线分布函数D 法线分布函数表示在给定的粗糙度下，微平面法线向量与半程向量方向相同的概率。此处的半程向量根据观察方向与入射方向计算：

根据概率的特征，D通常在其作用域归一化为1，即：

法线分布函数是由表面粗糙度决定，用表示这个粗糙度参数。一般的关于光泽BRDF球状分布，在函数图像上会具有很长的拖尾。这个拖尾在渲染出的画面中表示为高光点边缘的模糊。在迪士尼给出的BRDF模型中，使用的是分布，它具有较长的拖尾，得到比较好的画面效果：

其中，粗糙度是介于0到1之间的值，当其为0时表示一个绝对光滑的表面，等于1时表示绝对粗糙的表面；为半程向量与物体法线的夹角。

几何遮挡函数G 几何遮挡函数的一个特征是其值介于0和1之间，描述的是在微观面元中，有多少是同时能够被入射方向和反射方向看见的。几何遮挡函数通常依赖于法线分布函数，所以将会从法线分布函数按史密斯阴影函数推导出真实的几何遮挡函数，然后通过roughness参数来直接计算一个近似函数。为了满足实时渲染计算的快速需求，在大多数情况下使用的是Schlick近似模型：

其中，为法线方向，为观察方向，表示粗糙度。一种是针对直接光照，另一种是IBL光照的重映射（Remapping）用于间接光照的计算。

为了有效的估算几何部分，可以使用史密斯法(Smith’s method)将观察方向对应的几何遮蔽(Geometry Obstruction)和光线方向向量对应的几何阴影(Geometry Shadowing)都考虑进去：

菲涅尔方程 在自然现象中，一个物体的反射的光线与折射的光线比率会随着观察的角度变化而发生改变。菲涅尔方程就是描述被反射的光线对比光线被折射的部分所占的比率。当光线与表面进行交互的时候，菲涅尔方程会根据观察角度给出被反射的光线所占的比率。利用这个比率和能量守恒原则，可以直接得出光线被折射的部分以及剩余的能量。

当对任何物体或者材质表面进行垂直观察时，它们都有一个基础反射率(Base Reflectivity)，用表示。当以一定的角度对平面进行观察时，所有反射光线会变得明显起来。菲涅尔公式取决于入射角和两种介质的折射系数，用表示通过菲涅耳公式得到的反射率。在渲染领域通常不会使用原始的菲涅耳公式，使用的是通过Fresnel-Schlick方法得到的近似方程：

**渲染方程**

为了让计算机模拟自然的光照现象，描述虚拟场景中光照的发生过程，Kajiya提出了渲染方程的理论。该方程可以描述表面某点在某个方向上的出射光照辐照强度的计算过程：

其中，为在点处出射光的辐照强度；为点处的自发光的辐照强度；表示光线对一个给定了材质属性的点反射出来的光线所作出的贡献程度，即双向反射函数（bidirectional reflectance distribution function，BRDF）；为点处入射光的辐照亮度； 为点处法线与入射光方向的夹角；积分为对点处法线所在的半空间中所有的光线进行的积分。图 4-11将展示渲染方程的积分空间。

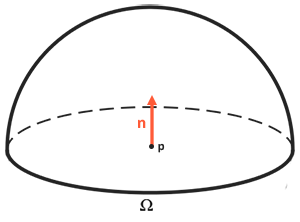
****

图4-11 积分空间

**Cook-Torrance BRDF模型**

该模型包括两个部分，漫反射和镜面反射：

其中，为入射光中被折射部分的能量占比，是被反射部分的占比。被称为Lambertian漫反射，表示如下：

为镜面反射部分，表示如下：

字母D，F与G分别代表着一种类型的函数，即上面所讨论的三种函数：正态分布函数(Normal Distribution Function)，菲涅尔方程(Fresnel Rquation)和几何函数(Geometry Function)。

**蒙特卡洛积分**

在渲染方程中的被积函数往往比较复杂，不能简单求出数值解。蒙特卡洛积分的目的是对积分进行近似计算。由大数定律可知:

令，上述期望近似变为积分的估值:

该估值的期望为:

由此可得该估计可以用作的近似，该估计的方差为:

可见方差随的增加线性降低，又由于误差与标准差成正比，因此误差随着线性降低。理论上蒙特卡洛方法没有对进行要求，但由上式可知分布越接近误差会更小。

* + - 1. **PBR材质**

本PBR渲染器中，将PBR材质分为以下4个部分：

反照率贴图 反照率(Albedo)贴图为每一个材质指定表面颜色或者基础反射率，即表面的颜色。

金属度贴图 金属度(Metallic)贴图逐个像素的指定是不是金属质地的。

粗糙度贴图 粗糙度(Roughness)贴图可以以像素为单位指定表面的粗糙度。采样得到的粗糙度数值会用于几何遮挡函数和菲涅尔方程中。

环境光遮蔽贴图 环境光遮蔽(Ambient Occlusion, AO)贴图为表面和周围潜在的几何图形指定了额外的阴影因子。

* + - 1. **间接漫反射光照的预计算**

为了以更有效的方式解决渲染方程中的积分，我们需要对其大部分结果进行预处理或称预计算。渲染方程中漫反射的部分为：

可以看到漫反射Lambert项是一个常数项（颜色、折射率和π在整个积分是常数），不依赖于任何积分变量，它被移出漫反射积分。由此，可以计算或预计算一个新的立方体贴图，它在每个采样方向中存储漫反射积分的结果，这些结果是通过卷积计算出来的。为了对环境贴图进行卷积，通过对半球上的大量方向进行离散采样并对其辐射度取平均值，来计算每个输出采样方向的积分。用来采样方向的半球，要面向卷积的输出采样方向 。预计算的立方体贴图，在每个采样方向 上存储其积分结果，可以理解为场景中所有能够击中面向 的表面的间接漫反射光的预计算总和。图4-12为一个预计算结果：

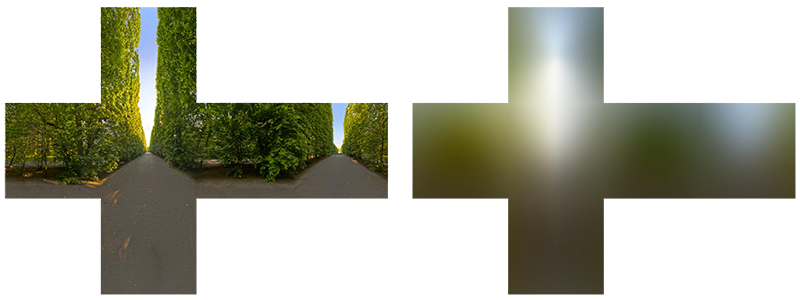


图4-12 间接漫反射预计算结果示例

* + - 1. **间接镜面反射光照的预计算**

Cook-Torrance 镜面部分（乘以）在整个积分上并不是常数，同时受到入射光方向和视角方向的影响。如果计算所有入射光方向加所有可能的视角方向的积分，根据排列组合的原理，它们组合结果的数量极其庞大，对于实时计算代价太昂贵。因此，Epic Games 提出了一个解决方案，分割求和近似法（split sum approximation）。分割求和近似将方程的镜面部分分割成两个独立的部分，可以单独求卷积，最后在 PBR 着色器中求和，以用于间接镜面反射部分 IBL。渲染方程的镜面反射部分：

可以看到积分不仅仅取决于还依赖，无法用两个方向向量采样去预计算的立方体图。Epic Games 的分割求和近似法将预计算分成两个单独的部分求解，再将两部分组合起来得到后文给出的预计算结果。分割求和近似法将镜面反射积分拆成两个独立的积分：

乘积左侧的部分被称为预滤波环境贴图。这个类似于间接漫反射中的辐照度图，也是预先计算的环境卷积贴图，但在计算过程中考虑了粗糙度。随着粗糙度的增加，参与环境贴图卷积的采样向量会更分散，导致反射结果更模糊。所以对于卷积的每个粗糙度级别，渲染器将按顺序把模糊后的结果存储在预滤波贴图的 mipmap 中。图4-13为一张预滤波贴图和其mipmap的展示。



图4-13 预滤波环境贴图示例

乘积右侧的部分为镜面反射积分的 BRDF 部分。假设每个方向的入射辐射度都是白色的，即 ，就可以在给定粗糙度、光线 法线 夹角 的情况下，对BRDF的结果进行预计算。将预计算好的 BRDF的结果将存储在一张 2D 查找纹理(LUT)上，称为BRDF积分贴图。2D 查找纹理在R通道存储是菲涅耳响应的系数，在G通道存储是偏差值，它提供了镜面反射积分的乘积左侧的部分。图4-14为一张已计算好的BRDF的LUT。

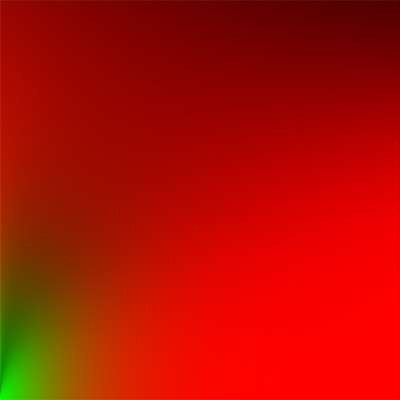


图4-14 BRDF的2D LUT

* + - 1. **着色器中的实现**

PBR渲染器中，顶点着色器将统一只进行坐标变换，故可以使用同一个。按照设计，渲染模式有传统光照和被分为3部分功能进行开放的PBR模式，与此对应，PBR渲染器中的有4类片元着色器进行场景物体的渲染。其分别为：

traditionalFragment：进行传统光照模型的光照计算

PBRFragment：PBR材质直接光照的光照计算

irradiancePBR：PBR材质添加间接漫反射光照的光照计算

final：PBR材质添加间接镜面反射光照的光照计算

在片元着色器中通用的有几项结构体：

PointLight：其中会接收应用程序传来的点光源的位置、颜色和衰减系数，着色器中有该结构体数组来接收场景所有点光源。

SpotLight：其中会接收应用程序传来的手电筒光源的位置、颜色、衰减系数和内外圈大小，着色器中有该结构体数组来接收场景所有手电筒光源。

Material：传统光照的着色器中材质有diffuse、normal；PBR着色器中有albedo、normal、metallic、roughness和ao。

在光照计算方面，都有两种光源的计算函数：CalcPointLight和CalcSpotLight，只是在内部传统光照与PBR的计算不相同。两种光源将通过循环被遍历计算。

在PBR的光照计算中还会用到以下几个函数来进行一些数据项的计算：

GeometrySchlickGGX：几何遮挡函数

DistributionGGX：法线分布函数

GeometrySmith：几何遮蔽与几何阴影整合

fresnelSchlick：菲涅尔方程，直接PBR光照计算使用

fresnelSchlickRoughness：根据粗糙度的菲涅尔方程，间接 PBR光照计算使用

在进行预计算前首先需要对读取的HDR环境贴图进行处理，将其处理成立方贴图方便下两步的预计算。这步处理将在equirectangle\_to\_cubemap着色器中进行处理，将会以渲染一个立方体的方式进行。处理后将会生成一个environmentMap纹理供irradiance\_convolution着色器和prefilter着色器使用。

irradiance\_convolution着色器中将进行间接漫反射的预计算，对environmentMap进行卷积处理得到辐照图。

prefilter着色器中会根据粗糙度进行重要性采样。其实现方式为：开始一个大循环，生成一个随机（低差异）序列值，用该序列值在切线空间中生成样本向量，将样本向量变换到世界空间并对场景的辐射度采样。这里使用的是低差异 Hammersley 序列。基于特定的粗糙度输入和低差异序列值 Xi，获得了一个采样向量，该向量大体围绕着预估的微表面的半向量。输入的粗糙度随着预过滤的立方体贴图的 mipmap 级别变化（从0.0到1.0），我们根据据粗糙度预过滤环境贴图，把结果存在 prefilteredColor 里。再用 prefilteredColor 除以采样权重总和，其中对最终结果影响较小的采样最终权重也较小。此为之前所述渲染方程拆分的左半部分。

brdf着色器中将会预计算BRDF。此为渲染方程拆分的右半部分，等同于在纯白的环境光或者辐射度恒定为1.0的设置下，对镜面 BRDF 求积分。着色器将角度和粗糙度作为输入，以重要性采样产生采样向量，在整个几何体上结合 BRDF 的菲涅耳项对向量进行处理，然后输出每个样本上F0的系数和偏差，最后取平均值。

* 1. **系统测试**

在PBR渲染器完成后，对系统进行测试，包括功能测试和性能测试两个部分。测试平台为Windows系统；GTX1660Ti显卡；i7-9750H CPU；16GB内存。

* + 1. **功能测试**

功能测试主要对PBR渲染器按照需求分析中提出的功能需求进行测试与验证。

**测试任务**

|  |  |
| --- | --- |
| 支持传统光照模型 | 测试是否支持传统光照模型和对多种光源的支持。 |
| 基础物体的生成和模型的导入 | 测试基础物体的生成和模型的导入，并对物体的贴图加载和属性修改进行测试。 |
| 相机 | 测试相机的移动与旋转功能。 |
| 学习文档 | 测试PBR学习文档的显示功能和PBR功能的解锁，并测试解锁的可使用PBR功能。 |

**测试结果与分析**

测试任务1：对传统光照模型的支持、多种光源的支持

测试结果如图4-11所示，物体初始的渲染方式为传统光照模式，同时还有不同种类的光源，还可以对光源属性进行设置。

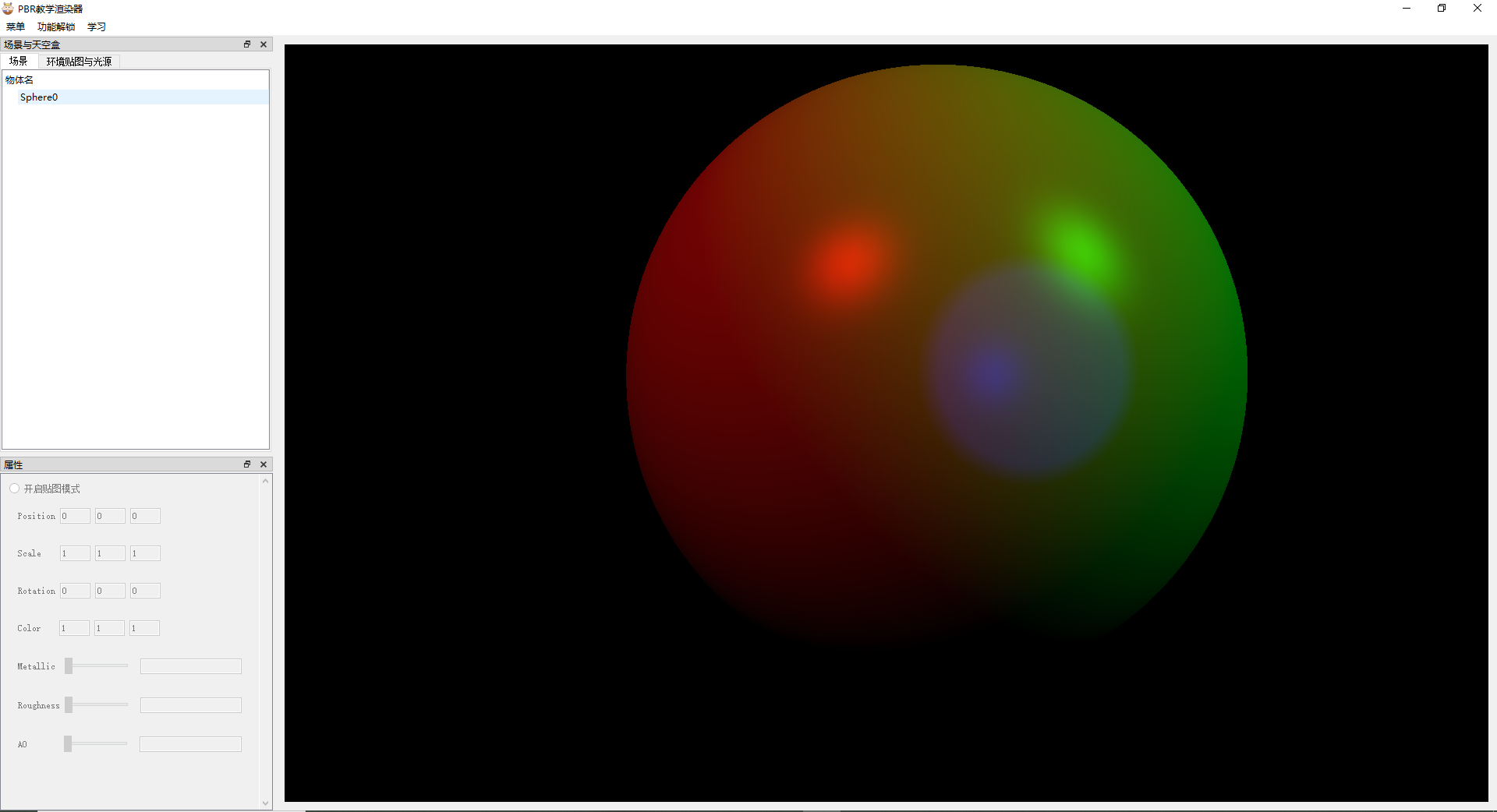


图4-11 基础物体的传统光照渲染

测试任务2：基础物体的生成和模型的导入与物体属性修改

在菜单栏中可以选择基础物体与模型导入；在场景栏中，可以对点击选择物体并对其属性进行设置；当点击开启贴图模式时，可以进入带贴图的渲染，提供贴图后能正确加载到物体上，测试结果如图4-12所示。

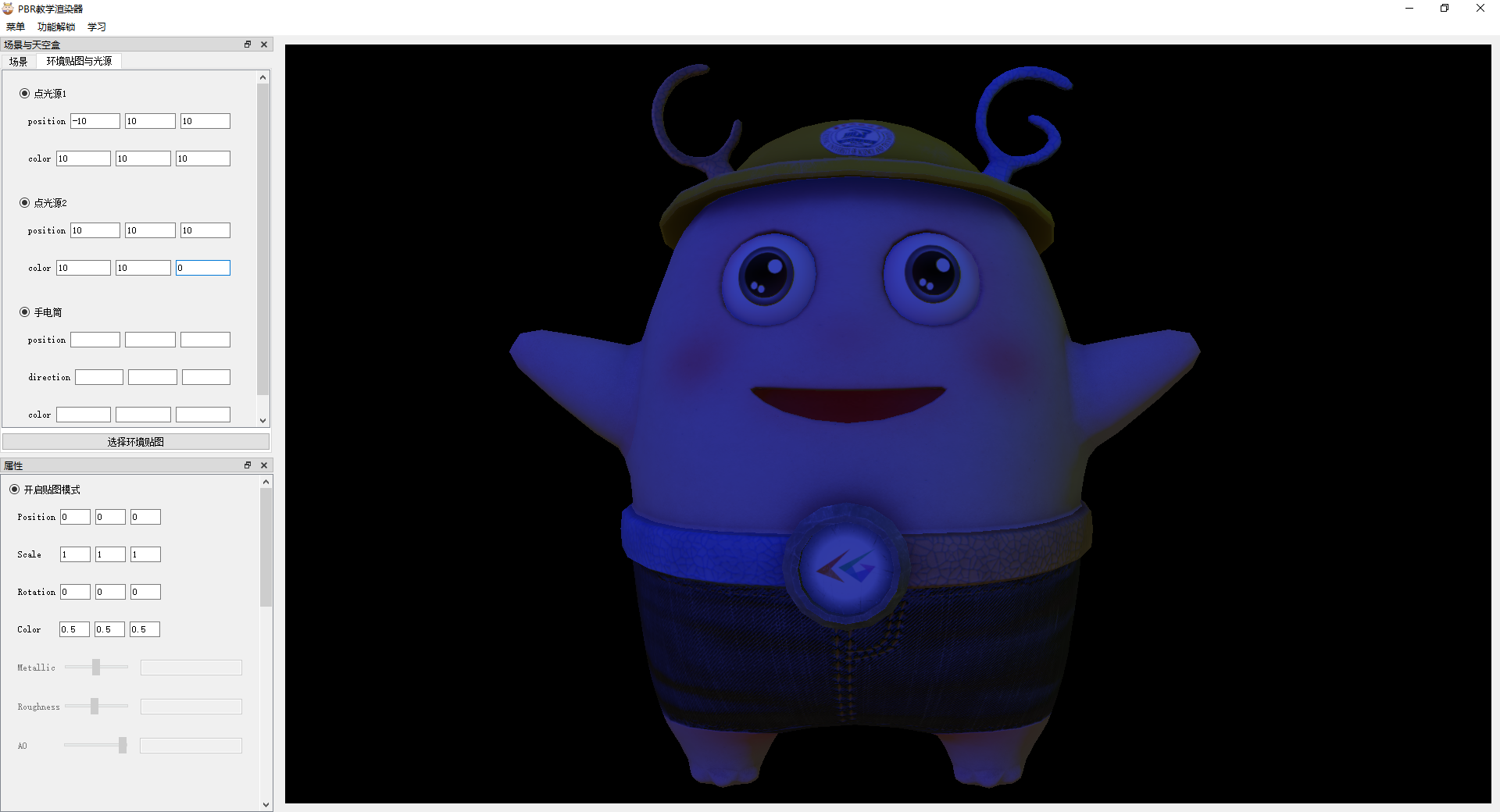


图4-12带贴图模型的传统光照渲染

测试任务3：相机的移动与旋转功能

使用键盘上的WASDQE可以进行相机的前后左右上下的操作，并且使用鼠标可以进行视角的旋转。

测试任务4：学习文档的显示功能和PBR功能

点击3份文档的打开按钮均能打开文档并且进行阅览。阅览完毕后点击PBR材质的解锁按键可以在属性栏中看到PBR材质对应属性的解锁，并且物体的渲染为基于物理的渲染；点击间接漫反射的解锁按键，然后在场景栏中进行环境贴图的添加，可以看到场景有一个天空盒并且物体的渲染稍稍真实；点击间接镜面反射的解锁按键，可以明显看到物体的渲染更加真实，反射的细节增多；在属性栏对无贴图的物体进行PBR材质值的的修改可以反映到物体上；在属性栏中可以对无法自动加载的PBR贴图进行手动加载；测试结果如图4-13所示。

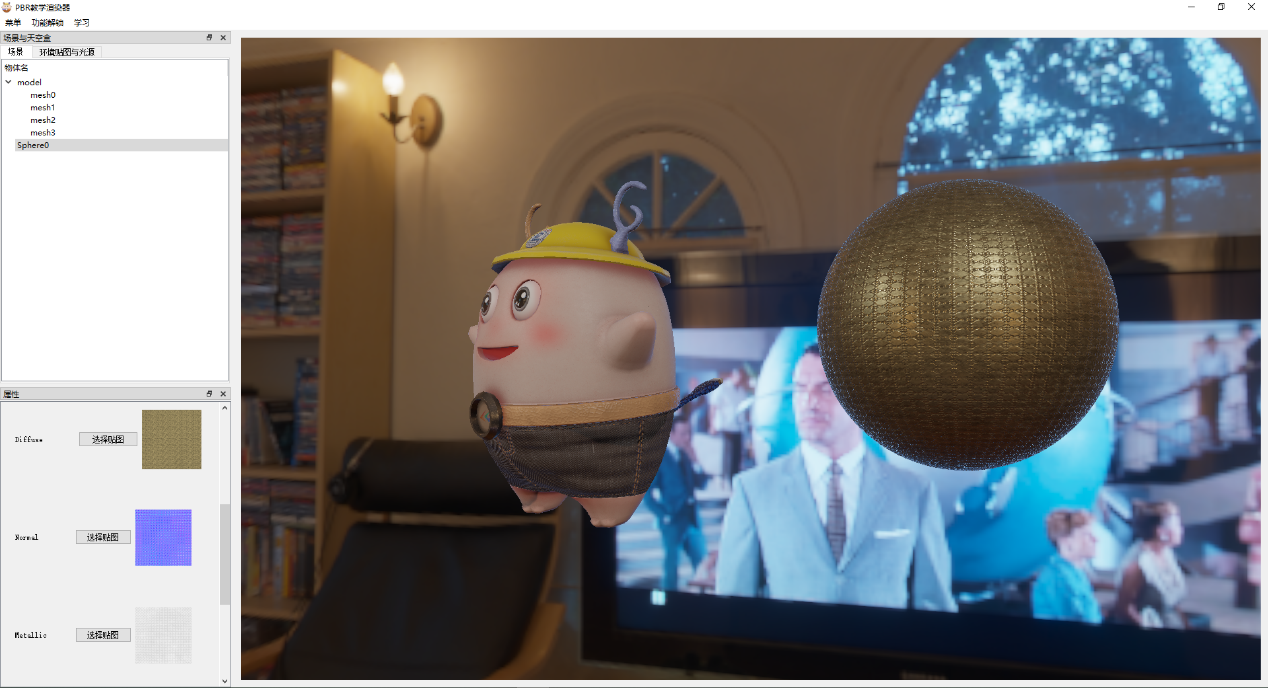


图4-13 PBR效果

在测试中发现，物体位置旋转与位移的设置操作由于是直接输入数值来进行会导致有些不够直观，用户不能够清晰了解变化后的结果。建议后续能够直接在渲染画面中进行拖动设置。相机的旋转操作，速度是一定的，当使用不同DPI的鼠标时导致速度不能自由调整以适应用户习惯。

* + 1. **性能测试**

性能测试主要对PBR渲染器按照需求分析中提出的功能需求进行测试与验证。

**测试工具**

对于帧数测试，将使用Fraps进行实时帧数测试。Fraps是Beepa开发的Windows的基准测试，屏幕捕获和屏幕录制实用程序。它可以从使用DirectX和OpenGL的软件中捕获。响应测试为自行编写测试函数进行测试。

**测试设计**

|  |  |
| --- | --- |
| 帧数测试 | 首先进行传统光照模型方式渲染的测试，物体数为5，其中基础物体3个，模型2个，每个模型将有不同的顶点数；然后进行完整的PBR功能开启后的帧数测试，测试物体与传统光照想相同。 |
| 响应测试 | 分别进行基础物体生成与不同面数模型读取、相机操作、属性修改的响应测试。 |

**测试结果与分析**

在帧数方面，测试中的帧数基本稳定在120帧；在生成物体、读取模型和添加环境贴图时会出现掉帧现象，此时帧数为83到90帧左右，加载完毕后会恢复到120帧并稳定。帧数方面符合性能要求。

经测试，在生成基础物体时，PBR渲染器的响应时间未超过需求所规定的0.5s响应时间。在对模型加载的测试中，测试了三角形面片数为6954的模型，未出现卡顿现象，加载时间在0.5s以内；测试三角形面片数为19058的模型时，出现的卡顿现象，但响应时间满足需求中的性能要求；当测试更多三角形面片数的模型时，卡顿现象变得明显，响应时间也更长。

对于相机操作的响应，普通状态下响应时间正常；在模型加载的瞬间，测试时会有点卡顿的现象；对场景物体的属性改变，在测试中基本没有卡顿现象，响应时间完全符合需求中的性能要求。

* 1. **本章小结**

本章对渲染器的实现细节进行了阐述并对渲染器的测试进行了阐述。实现部分重要叙述了有关于PBR的部分，详细介绍了渲染方程和Cook-Torrance BRDF模型并阐述了相关实现；介绍了IBL的相关算法与实现。在测试部分，阐述了按照需求进行的功能测试与性能测试。

1. **总结与展望**

本章对PBR渲染器的设计和实现工作进行总结，并针对目前的不足之处进行展望。

* 1. **总结**

基于物理渲染技术与全局光照技术是近几年来游戏工业中的热门研究领域，各种PBR技术也有着广泛而深入的应用。本文围绕着PBR技术，进行了如下工作：

本文首先探讨了基于物理渲染的技术背景和PBR渲染器的实现意义进行了探讨，并对当前PBR技术的研究状况进行了调查。然后对PBR渲染器进行了需求分型并对渲染器系统进行了架构设计和模块划分。同时，还确定了开发与运行的环境。在实现部分，先对相关技术进行解释。讨论了微平面模型与影响这种表面光照表示的几个方程，与这种方程在实际运用种的近似。探讨了渲染方程的快速求解方法，对渲染方程的分割来进行预计算。然后讨论系统每个部分的实现工作。阐述了所用到的一些Qt组件与一些重要的数据结构。分析了整个渲染系统的实现思路与最终的实现效果。最后，跟据最开始的需求分析进行了系统的测试工作。对系统的功能测试介绍了测试任务与测试结果，并进行了分析。对系统的性能测试介绍了测试的方法与结果。最终，课题所要实现的PBR渲染器系统完成了实现并通过测试。

本次课题过程中，了解到了自生对C++这门编程语言的不够熟练，导致了许多不必要的bug出现。但同时也更加掌握OpenGL这个图形接口。虽然在实现过程中造成了一些匪夷所思的程序崩溃，但在自主调试与探索下将错误进行了修正，并且进一步熟悉个课题中用到的各种技术与算法。在了解到自己的不足的同时，收获到了更多的知识。

* 1. **展望**

虽然PBR渲染器可以进行基础的PBR效果展示，但还有一些地方可以进行改进和扩展：

1. 对更多种类了模型文件进行支持，Assimp库本身有着对多种模型文件进行读取的功能，如3ds、stl等。理论上对更多种类了模型文件进行支持是能够实现的。
2. 支持更多种类的贴图，如视差贴图。视差贴图会让渲染出的物体更加真实，让PBR的效果更好。也可以支持更多的PBR算法，比如与动态天空盒技术进行结合，实现出更加真实的渲染画面。
3. 支持用户自定义着色器。目前系统只能使用已编写好的着色器来进行渲染，无法自定义一些渲染细节的计算。
4. 对于最后的测试，应当可以使用更加完整的工具链，来对帧率、内存等信息进行及时的查看。这样还有利于系统开发中的调试。

# 致 谢

感谢我的母校华中科技大学，这是一所优秀的高等学府，不仅有着严谨的治学态度和良好的学术氛围，也给予学生很大的空间自由全面发展，“明德、厚学、求是、创新”的校训我也始终铭记于心，我有幸能在此度过四年的学习生活。

感谢我的毕业设计指导老师万琳老师，在毕业设计过程中为我提供了必要的参考资料，在写论文的过程中也给予了我很多帮助和建议，令我获益匪浅。

感谢我所在的技术团队Memo游戏工作室，让我能够在工作室中锻炼自己，积累了许多技术经验，并结交了志同道合的朋友一起外出比赛，做自己爱好的事情。

最后感谢我的家人和我大学期间遇到的各位同学、朋友、老师，感谢他们在我大学期间给予我的各种帮助，我对此心怀感激，在他们的支持下我得以顺利完成大学四年的学业

**参考文献**

1. Scott Meyers.Effective C++[M].电子工业出版社:,2006.
2. Martin Mittring. The technology behind the “unreal engine 4 elemental demo”. Advanced Real-Time Rendering in 3D Graphics and Games, 2012.
3. Sam Martin and Per Einarsson. A real-time radiosity architec- ture for video games. SIGGRAPH 2010, 2010.
4. Brent Burley. Physically based shading at disney. 2012.
5. KenPerlinandEricM.Hoffert.Hypertexture.ComputerGraphics (SIGGRAPH ’89 Proceedings), 23(3), July 1989.
6. 安晓辉.Qt Quick核心编程[M].电子工业出版社:,2015.
7. Qt 5.14.2 Released, 2020.
8. Jasmin Blanchette, Mark Summerfield.C++ GUI Qt 4编程[M].电子工业出版社:,2013.
9. 萨默菲尔德.Qt高级编程[M].电子工业出版社:,2011.
10. Tomas Akenine-Möller,Eric Haines,Naty Hoffman.Real-Time Rendering 4th Edition[M].A K Peters/CRC Press:,2018:392-424.
11. 秦春林.全局光照技术，从离线到实时渲染[M].电子科技大学出版社:,2018:243-277.
12. Chris Green. Improved alpha-tested magnification for vector textures and special effects. Advanced Real-Time Rendering in 3D Graphics and Games(SIGGRAPH Course 07), 2007.
13. Jan Novak, Thomas Engelhardt, and Carsten Dachsbacher. Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. 2011.
14. Matt Pharr and Greg Humphreys. Physicall Based Rendering: From the Theory to Implementation. Elsevier, second edition, 2010.
15. James T. Kajiya. The rendering equation. 1986.
16. Marries Van De Hoef. Real-time dynamic radiosity for high quality global illumination. December 2013.
17. Tomomichi KANEKO, Toshiyuki TAKAHEI, Masahiko INAMI, Naoki KAWAKAMI, Yasuyuki YANAGIDA, Taro MAEDA, and Susumu TACHI. Detailed shape representation with parallax mapping. Proceedings of ICAT, 2001.
18. Green,R.Spheriacal harmoric lighting: The gritty details, 2003.
19. Art Owen and Yi Zhou. Safe and effective importance sampling. 1998.
20. Michael Bunnell. Dynamic ambient occlusion and indirect light-ing. GPU Gems 2, 2005.
21. RobertL.Cook.Stochasticsamplingincomputergraphics.ACM Transactions on Graphics, 5(1), January 1986.
22. Victor Elvira, Luca Martino, David Luengo, and Monica F. Bugallo. Efficient multiple importance sampling estimators. 2015.
23. Ned Greene. Environment mapping and other applications of world projections. November 1986.
24. Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Pro- gressive photon mapping. ACM Transactions on Graphics, 27(5), December 2008.
25. Henrik Wann Jensen. Global illumination using photon maps. 1996.
26. Sam Martin and Per Einarsson. A real-time radiosity architec- ture for video games. SIGGRAPH 2010, 2010.