Mitsuba 系列 5-一个完成的光线追踪器的移植

Dezeming Family

2023 年 4 月 5 日

DezemingFamily 系列书和小册子因为是电子书，所以可以很方便地进行修改和重新发布。如果您获得了 DezemingFamily 的系列书，可以从我们的网站 [https://dezeming.top/] 找到最新版。对书的内容建议和出现的错误欢迎在网站留言。

本文我们一鼓作气，实现 Mitsuba 完整的渲染器流程。我们只移植渲染器中满足最简单的渲染物体组件，比如漫反射材质、简单的面光源等。

目录

一　对象和插件

Mitsuba 的渲染相关类是用插件的方式管理的，但我们的工程并不要求我们使用插件，但还是有必要将插件管理的流程搞明白，以方便移植。

不过像 Mitsuba 这种模块众多的渲染器，使用插件管理确实是一种比较方便的方法，因此以后我们还会构建一个基于插件的工程。

PluginManager::createObject() 函数中会搜索插件并创建：

```
1 LockGuard lock(m_mutex);
2 // 保证该插件已经被加载了进来
3 ensurePluginLoaded(props.getPluginName());
4 // 使用插件创建对象
5 object = m_plugins[props.getPluginName()]->createInstance(props);
```

ensurePluginLoaded() 函数中，如果插件不存在，就搜索并加载插件。

由于 Mitsuba 0.5.0 编译得到的插件跟 0.6.0 版本有所不同，所以我们无法测试 Mitsuba 0.5.0 版本的插件。我们下一节开始介绍如何实现一个接口来创建 Mitsuba 对象。

二　创建我们自己的 CreateMtsObject 类

在 MainGUI 里有 createObject.h 和 createObject.cpp 两个文件，这两个文件有我们自定义的 CreateMtsObject 类，该类的很多功能都是基于 PluginManager 类来写的，只不过我们修改为了直接创建 Mitsuba 对象，而不是使用插件。

我们想要加载的文件 cbox.xml 里需要用到下面几个插件：

```
1 path
2 ldsampler
3 gaussian
4 hdrfilm
5 perspective
6 diffuse
7 area
8 obj
```

我们在本文的代码中为了简洁起见，也只加载这些类相应的源文件到项目中。关于这些对象的加载我们在之前的文章中都介绍过了，因此这里大家可以直接看源码就好，主要内容在 CreateMtsObject::createObject() 函数：

```
1 if (props.getPluginName() == "diffuse") {
2     object = (ConfigurableObject *)getSmoothDiffuse_Instance(props).get();
3 }
4 else if (props.getPluginName() == "area") {
5     object = (ConfigurableObject *)getAreaLight_Instance(props).get();
6 }
7 ......
```

三　开始渲染

启动渲染的程序写在了 IMAGraphicsView::setRendering() 函数里，该函数会调用 RenderJob::run()，执行渲染线程。

要确保渲染线程能够开启内核来渲染，需要下面的重要组件：

```
mitsuba::ref<mitsuba::QRenderListener> m_renderListener;
mitsuba::ref<mitsuba::RenderQueue> m_renderQueue;
mitsuba::Thread::EThreadPriority m_workerPriority;
```

并且需要在渲染前执行下面的函数：

```
    initWorkersProcessArgv()
```

正确初始化才能渲染。RenderJob::run() 中各个环节的执行大家都应该认真看一看，把这个环节掌握清楚。

注意下面的场景文件路径需要改成你的场景文件路径：

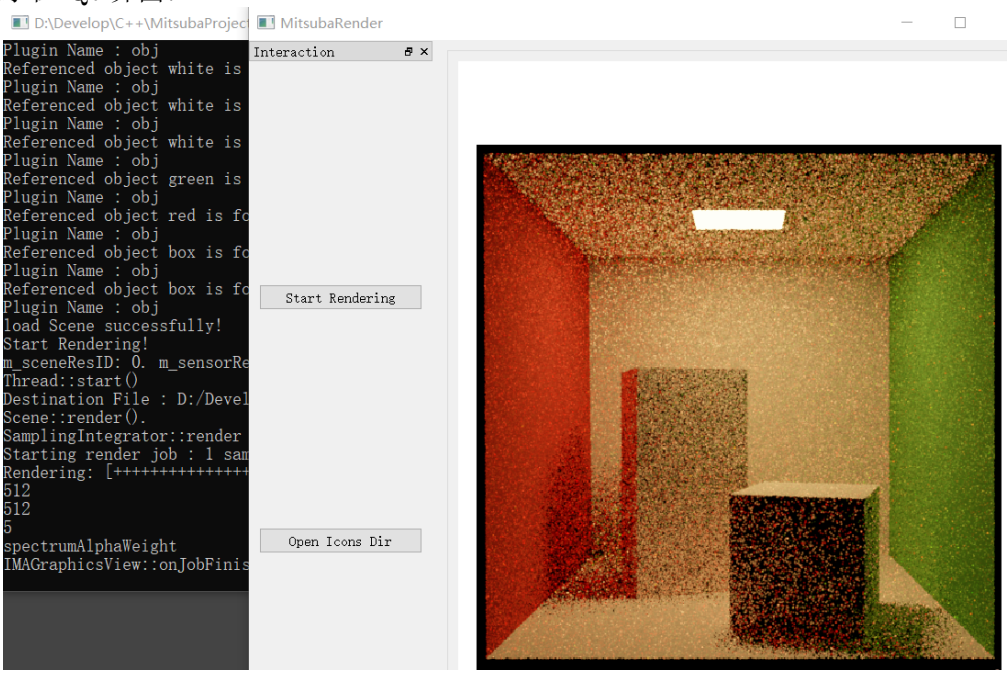
```
QString filename = "D:/Develop/C++/MitsubaProject/Mitsuba scenes/cbox/cbox.xml";
```

渲染得到的结果会被保存为相应的 PNG 图像，是使用 bitmap 类来保存的。film 有两种最常用，一种是 HDRFilm，另一种是 LDRFilm，我们下一节会介绍如何把这两种 bitmap 转换为 Qt 的图像，然后在 Qt 上显示。

四　转换为 Qt 图像并显示

这部分代码我们放在了 FrameBuffer.h 和 FrameBuffer.cpp 两个文件里。

这两个文件实现了我们自定义的 FrameBuffer 类,并且实现了 FrameBuffer 转 Qt 图像和 mitsuba::Bitmap 转 FrameBuffer 的功能（Bitmap2FrameBuffer() 函数），使得我们可以将渲染结果转换为 QPixelMap 对象，然后显示在 Qt 界面：



五　小结

自此，可以说整个 Mitsuba 的软件架构我们已经完全掌握了。Mitsuba v1 的主线任务已经完成。至于 Mitsuba 中实现的各种组件和功能的原理，我们将会在以后作为一个一个的小专题来介绍。但是我们的 Mitsuba v1 部分的源码就是这些了——其他部分相信读者可以很容易对照着我们已有的部分移植进来，然后编译并成功运行。

本文也是花了我较长时间来调试和分析，解决了不少 Bug，节省了用户配置各种第三方库的时间，可以很方便地进行二次开发。

参考文献

[1] https://www.mitsuba-renderer.org/index_old.html

[2] https://www.mitsuba-renderer.org/docs.html

[3] https://www.mitsuba-renderer.org/releases/

[4] https://github.com/mitsuba-renderer/mitsuba

[5] http://mitsuba-renderer.org/api/index.html

[6] https://www.mitsuba-renderer.org/download.html