LearnOptix-v7系列3-Optix基本语义

Dezeming Family

2023年8月21日

DezemingFamily系列文章因为都是免费的电子文档，所以可以很方便地进行修改和重新发布。如果您获得了DezemingFamily的系列文章，可以从我们的网站[https://dezeming.top/]找到最新版。对文章的内容建议和出现的错误欢迎在网站留言。

全局光照(GI)有很多种解决方案，比如VXGI、Lumen、DDGI、SSGI、IBL、PRT、SurfelsGI等，其中，越来越火的Nvidia的RTX技术也是一些软硬件结合的实时光追解决方案。

目录

一　前要概述

Optix的缺点是就像D3D或者OpenGL，代码比较难以理解。更难的问题在于Optix的资料太少，几乎没有多少可以用来参考或者学习的资料，尤其是比较通俗的中文资料。笔者在广泛搜集和阅读源码后，公开了两部系列教程Optix v6和Optix v7两个系列。其中，Optix v6来自于对官方源码结合用户手册的解读；Optix v7来自于对一个已有教程的解读。

二　Optix v6结构

Optix v6结构相对比较简单和容易理解。

2 1　基本构成组件

主要的对象模型有：

- Context：用于运行 Optix 引擎的实例。
- Program：CUDA 函数，编译为 NVIDIA PTX 虚拟汇编语言 (virtual assembly language)。
- Variables：一种变量，用于将数据传入 Optix 程序。
- Buffer：绑定到一个变量的多维数组。
- TextureSampler：Buffer 的插值机制。
- Geometry：ray 可以相交的基元，比如三角形或者用户自定义类型。
- Material：材质程序，当 ray 相交与基元相交时执行。
- GeometryInstance：绑定 Geometry 和 Material。
- GroupNode：一系列安排在层次结构 (hierarchy) 中的对象。
- TransformNode：一个层次节点 (hierarchy node) 用于变换几何和 ray。
- SelectorNode：可编程层次节点，用来选择去遍历的子对象。
- AccelerationStructure：绑定到层次节点 (hierarchy node) 的加速结构对象。

组件程序有：

- Ray generation programs：光追管线的入口，由每个像素样本或者用户自定义的任务（比如一些自定义的特殊算法）来调用。
- Exception programs：异常控制。
- Closest hit programs：当 ray 找到最近交点时调用，在该程序中执行比如材质着色等程序。
- Any hit programs：当 ray 找到一个新的潜在的最近交点时调用，比如对于计算阴影时有用。
- Intersection programs：实现一个 ray 和基元交点测试，在 traversal 时调用。
- Bounding box programs：计算基元的世界空间包围盒，当构建一个新的加速结构时调用。
- Miss programs：当追踪的光线错过了所有几何结构时调用。
- Attribute programs：当与内置三角形相交时调用，用于为任何命中和最近命中的程序提供三角形特定属性。

主机可以对Variables（rtDeclareVariable）进行赋值，并在GPU设备上访问这些值。

主机上申请Buffer（rtBuffer），然后用于在GPU设备上访问Buffer里的值或者计算并往Buffer里面填充值。

GPU启动渲染需要一个入口程序，一般就是光线生成程序，我们可以在一个Context里设置多个入口程序，并在发射时根据情况选择使用哪个入口程序。

2 2 射线与求交

发射启动后，如果光线检测到与最近的基元求交得到了交点，就会调用该物体绑定的closest-hit程序，如果与任意一个物体相交（比如在阴影计算时），就会调用any-hit程序。如果与任何基元都没有交点，就会调用miss程序。

进行光线追踪时，一般都是这样调用：

```
1 rtTrace(top_object, ray, payload);
```

其中，第一个参数表示加速结构最顶层，ray就是光线，payload就是光线携带的信息，比如根据BRDF当前的光衰减量，或者光此时反弹的次数等。计算阴影着色时的射线和计算辐射度的射线都用rtTrace(...)来追踪。

每个基元类型都需要绑定一种求交程序（内置的三角形可以不用绑定），以及设置该基元的包围盒。

然后为每种基元设计closest-hit程序和any-hit程序，并且绑定到材质对象中。之后再把材质对象和基元绑定在一起。绑定好的基元和材质叫做几何实例(GeometryInstance)。多个几何实例可以构成一个几何组(GeometryGroup)。

对于射线遍历过程中发现的每个潜在的最近交点，都会执行 any-hit program。执行程序的交点可能不会沿着射线相交顺序排序，但如果需要，最终可以枚举射线与场景的所有交点。不过在计算阴影光线时，一般会给采样射线的负载Payload设置一个衰减值，当衰减值到0，则终止射线的遍历，否则就忽略交点：

```
1 if(optix::luminance(prd_shadow.attenuation) < importance_cutoff)
2     rtTerminateRay();
3 else
4     rtIgnoreIntersection();
```

2 3 纹理

纹理的定义看着跟CUDA有一些区别，但其实基本过程都是一致的（只不过Optix 7的函数名跟CUDA都是基本一样的）。

纹理需要在.cu文件里用rtTextureSampler声明，然后在主机上通过Context创建，然后初始化。

纹理并不会和材质绑定在一起，因此，哪怕是相同的材质类型，如果纹理不同，也需要分别实现不同的closest-hit程序。

2 4 三角Mesh

三角mesh可以自己写求交程序，也可以用optix自己内置的程序，但是如果我们用内置的程序的话，如果额外提供了法向量贴图，那么就需要实现attribute程序来计算着色法向量。attribute程序在closest-hit和any-hit程序执行之前执行，更新法向量等信息。

注意每个不同的mesh都有自己的indexBuffer以及vertexBuffer，自定义求交程序时，必须给一个参数primIdx，使得程序可以知道是与Buffer里哪个三角形获得了交点。

如果实例化的对象们的材质不同，假设总共涉及三种材质，那么就需要所有的实例化对象都包含这三种材质，并通过函数预先指定我们需要哪种材质。

三　Optix v7结构

Optix v7的结构就复杂了很多。尤其是引入了着色器绑定表/发射参数等概念。

Optix v6中，相机等参数都是使用rtDeclareVariable变量定义的，纹理/三角顶点数组等信息也都是通过全局变量来设置，比如rtBuffer/rtTextureSampler/rtDeclareVariable。在Optix v7中，这些内容都被列入了着色器绑定表里。

3 1　基本构成组件

Moudle会包含各种类型的程序，Moudle的创建需要两个很重要的参数结构：

```
1 OptixPipelineCompileOptions
2 OptixModuleCompileOptions
```

moudles定义了一些特性，这些特性来自于上面两个参数结构设置的属性，这些特性相当于给绑定到管线里的progrmas进行一些设置（比如第二个参数结构可以设置最大追踪深度/是否用离焦模糊等）。用于创建链接到单个管线中的程序组的所有moudles都必须使用相同的第一个参数结构（即，单个管线里的所有程序，最大追踪深度等属性都必须是一样的）。同一管线内的模块可能会有不同的第二个参数结构。

因此，如果我们想进行二次光线追踪来获得一幅图像，且两次光追用不同的管线，就需要设置两组管线，并且为其配置不同的Moudles。

3 2　管线

一个管线里可以有多个采样射线生成程序Raygen，可以有多个Miss程序，以及多个Hitgroup程序。每个Hitgroup程序都包含了closest-hit和any-hit程序。

3 3　着色器绑定表(SBT)

着色器绑定表可以理解为当响应了closest-hit或者any-hit程序以后，执行计算时需要访问的数据，比如使用哪个纹理/访问顶点数组。其实实际上，当物体与射线有交点时，去执行哪个closest-hit或者哪个any-hit程序，时根据物体绑定的SBT来决定的。

在构建加速结构时，每个OptixBuildInput对象（比如代表一个球体，或者代表一个mesh组）都对应一个物体ID。

当我们只有一种发射的射线类型时，每个物体绑定的一个SBT，该SBT对应一个Hitgroup程序就可以了。如果有两种发射的射线类型，而且每个物体都要实现对两种射线类型不同的着色计算，那么就需要两种类型的SBT，分别绑定两个不同的Hitgroup程序。当此时求交时是第一个SBT，那么第一个Hitgroup程序响应；当此时求交时是第二个SBT，那么第二个Hitgroup程序响应。

sbt.hitgroupRecordBase属性设置了Hitgroup的SBT的地址。用以GPU程序响应求交程序后访问SBT。

当设置有两种采样射线类型且需要访问两种不同的SBT时（采样辐射度射线/采样阴影的射线）时，所有OptixBuildInput对象对应的SBT都应该是两个，而不能有的是2个，有的是1个，否则就会在optixTrace(...)时因为SBT stride（optixTrace(...)是对整个场景的所有加载进去的物体进行光追）不一致而导致访问错误。我们再给两个例子加深一下印象。

现在假设有三个closest-hit程序与三个any-hit程序，设名字分别为：

```
1     __closesthit__1
2     __closesthit__2
3     __closesthit__3
4     __anyhit__1
5     __anyhit__2
6     __anyhit__3
```

hitgroup程序也有3个，分别绑定了上面的三组closest-hit和any-hit程序：

```
hitgroup_1
hitgroup_2
hitgroup_3
```

以上的假设并不会直接用到，仅仅是作为一些场景罢了。

3 3.1 第一个例子

假设我们只有一种类型的相机采样射线。

假如我们加载了5个meshes，每个mesh都包括不同数量的三角形。

假设初始化了5个hitgroup SBT。因为我们要设置optixTrace(...)的SBT offset为0，且SBT stride为1（每个mesh绑定的SBT数量都是1个），也就是说：第一个mesh相当于绑定了第一个hitgroup SBT；第二个mesh相当于绑定了第二个hitgroup SBT；以此类推。

不同的SBT可以绑定不同的hitgroup程序。

3 3.2 第二个例子

假如我们加载了5个meshes，每个mesh都包括不同数量的三角形。

假设初始化了10个hitgroup SBT，设为HitgroupRecord[10]。我们希望追踪相机射线或者阴影射线时，响应不同的求交程序（closest-hit程序与any-hit程序）。

我们要设置optixTrace(...)的SBT stride为2（每个mesh绑定的SBT数量都是2个），也就是说：

当optixTrace(...)的SBT offset为0时；当相机射线与mesh[0]有交点，那么此时响应的Hitgroup SBT就是HitgroupRecord[0]；当相机射线与mesh[1]有交点，那么此时响应的Hitgroup SBT就是HitgroupRecord[2]；当相机射线与mesh[2]有交点，那么此时响应的Hitgroup SBT就是HitgroupRecord[3]；以此类推。

当optixTrace(...)的SBT offset为1时；当相机射线与mesh[0]有交点，那么此时响应的Hitgroup SBT就是HitgroupRecord[1]；当相机射线与mesh[1]有交点，那么此时响应的Hitgroup SBT就是HitgroupRecord[3]；当相机射线与mesh[2]有交点，那么此时响应的Hitgroup SBT就是HitgroupRecord[5]；以此类推。

第一个mesh相当于绑定了第一个和第二个hitgroup SBT；第二个mesh相当于绑定了第二个和第四个hitgroup SBT；以此类推。根据optixTrace的SBT offset参数来设置meshes应该去响应哪个SBT。

四 小结

费了那么多笔墨，我觉得应该算是讲清楚了它们之间设置的主要异同的。说实话，当我看到Optix 7的代码时，我觉得自己就好像从没有接触过Optix一样，以前optix 6的很多流程都不再有效了。因此我打算花点时间把这些内容总结和整理一下，就构成了这一篇文章。

参考文献

[1] https://github.com/ingowald/optix7course

[2] https://owl-project.github.io/

[3] https://casual-effects.com/data/

[4] https://raytracing-docs.nvidia.com/optix7/guide/index.html#preface#

[5] https://raytracing-docs.nvidia.com/optix7/api/modules.html

[6] https://raytracing-docs.nvidia.com/optix6/api_6_5/index.html

[7] https://raytracing-docs.nvidia.com/