LearnOptix 系列 7-三角形

Dezeming Family

2023 年 5 月 1 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。

2023-05-06：完成第一版。

目录

一 三角形 Mesh 的基本知识点 **1**

1 1 RTgeometrytriangles 1

1 2 另外的 RTgeometrytriangles 函数 2

1 3 三角形属性 2

1 4 多材质 4

1 5 其他内容 4

二 三角形的构建和使用 **5**

2 1 obj 读取 5

2 2 obj 转 Optix 5

三 小结 **6**

参考文献 **7**

一　三角形 Mesh 的基本知识点

RTgeometrytriangles 类型为 OptiX 提供了对三角形的内置支持，是 RTgeometry 类型的补充。它不需要自定义交点和边界框程序；应用程序只需要向 OptiX 提供三角形数据。OptiX 提供了一个内置的求交程序和一个用于三角形基元的 BVH 构建机制。加速度结构遍历和求交所需的三角形数据直接存储在加速结构内（因此速度会很快）。

应用程序应尽可能使用 RTgeometrytriangles 类型，在图灵架构下提供了硬件加速求交。

1 1　RTgeometrytriangles

GeometryTriangles 是一堆三角形构成的一个 mesh，因此其实它相当于一个包含一堆三角形的节点，就叫做 GeometryTriangles 节点 (GeometryTriangles node)。

NVIDIA OptiX 支持两种类型的三角形数据：索引的 (indexed) 三角形集合和无组织的 (unorganized) 三角形集合。（非索引的 (Non-indexed) 三角形集有时被称为 “triangle soup”，以强调三角形之间缺乏结构关系。）在任何一种情况下，实际顶点（无论是否索引）都是通过如下函数设置的：

```
RTresult rtGeometryTrianglesSetVertices(
    // GeometryTriangles node
    RTgeometrytriangles geometrytriangles,
    // 顶点缓冲区顶点数。对于无组织三角形集，必须是三角形数的三倍
    unsigned int          vertexCount,
    // GeometryTriangles的三角形集
    RTbuffer              vertexBuffer,
    // （可选参数）到vertex_buffer中第一个顶点的偏移量（以字节为单位）
    RTsize                vertexBufferByteOffset,
    // vertex_buffer中顶点之间的步长（以字节为单位）
    RTsize                vertexByteStride,
    //
    RTformat              positionFormat );
```

参数解释如下：

```
RTformat::RT_FORMAT_FLOAT3
RTformat::RT_FORMAT_HALF3
// 假设第三个分量为零，这对平面几何图形很有用
RTformat::RT_FORMAT_FLOAT2
// 假设第三个分量为零，这对平面几何图形很有用
RTformat::RT_FORMAT_HALF2
```

在有索引三角形的情况下，索引的三元组引用形成三角形的顶点。如果设置了索引缓冲区，则假设几何体是以索引三角形的形式给出的。如果未设置索引缓冲区，则假定几何体是作为三角形的无组织集合提供的。索引缓冲区是通过如下函数设置的：

```
RTresult RTAPI rtGeometryTrianglesSetTriangleIndices(
    // GeometryTriangles节点
    RTgeometrytriangles geometrytriangles,
    // 存储index的buffer
    RTbuffer              indexBuffer,
    // indexBuffer中第一个索引的偏移量（以字节为单位）
    RTsize                indexBufferByteOffset,
```

```
8     // indexBuffer中索引的三元组之间的步长（以字节为单位）。三元组中的索引之
        间不能有任何间距，只有三元组之间才支持间距（注意一般都是设置为0，表示
        三角形索引都是紧挨着的，但是有特殊情况，比如原始数据中除了三角形索
        引，后面还跟着三角形序号等其他信息，那么就应该跨越相应的字节数）
9     RTsize                  triIndicesByteStride ,
10    RTformat                triIndicesFormat  );
```

triIndicesFormat 有两种形式：

```
1 RTformat::RT_FORMAT_UNSIGNED_INT3
2 RTformat::RT_FORMAT_UNSIGNED_SHORT3
```

可以通过传递 NULL 作为 indexBuffer 参数来取消先前设置的索引缓冲区的设置，例如：

```
1 rtGeometryTrianglesSetTriangleIndices(
2     ggeometrytriangles , NULL, 0, 0, RT_FORMAT_UNSIGNED_INT3);
```

1 2 另外的 RTgeometrytriangles 函数

RTgeometrytriangles 的创建、销毁、检验、设置三角形数量和索引偏移，以及变量的指定与 RTgeometry 的相应函数类似。

```
1  rtGeometryTrianglesCreate
2  rtGeometryTrianglesDestroy
3  rtGeometryTrianglesValidate
4  rtGeometryTrianglesGetContext
5  rtGeometryTrianglesSetPrimitiveCount
6  rtGeometryTrianglesGetPrimitiveCount
7  rtGeometryTrianglesGetPrimitiveIndexOffset
8  rtGeometryTrianglesSetPrimitiveIndexOffset
9  rtGeometryTrianglesDeclareVariable
10 rtGeometryTrianglesQueryVariable
11 rtGeometryTrianglesRemoveVariable
12 rtGeometryTrianglesGetVariable
13 rtGeometryTrianglesGetVariableCount
```

我们等要用到的时候再讲解。

1 3 三角形属性

在讲解属性之前，有必要明确一句：(1) 情况一：Optix 中三角形 Mesh 可以完全不用自己实现各种功能（不需要再自己实现 any-hit program 和 closest-hit program 以及包围盒程序），但是有些人会问，如果我们的数据提供了着色法向量或者法线贴图，应该怎么办呢？这个时候 Optix 给我们了一种选择，即实现 attribute program，这个函数在调用 any-hit program 和 closest-hit program 之前执行，可以在这个函数里计算着色法向量或者纹理坐标等。(2) 情况二：就是我们自己写一个求交函数，调用 Optixu 的三角 Mesh 求交程序（下面会介绍）。

所以这可能牵扯到一个问题，比如求交中找最近交点时，RTgeometrytriangles 的求交是要与该节点内部的所有三角形都要进行的，找到最近的交点；而自定义的话只能跟一个三角形求交（见 triangle_mesh.cu 文件），除非遍历三角形节点下的全部三角形。这里需要注意的是，自定义求交函数一定有一个参数 primIdx，其实这个参数就相当于告诉代码要去与节点下的哪个三角形进行求交了（所以相当于自定义的求交也会借助 RTgeometrytriangles 内的加速结构）。

```
void meshIntersect( int primIdx )
```

与三角形求交的 Optixu 函数有以下几种（我们一般使用第一个就可以了）：

```
optix::intersect_triangle
optix::intersect_triangle_branchless
optix::intersect_triangle_earlyexit
```

这些函数的参数列表都一样（n 是得到的非单位化法向量，如果每个顶点都提供了法向量，那么这个值就没用了；t 是光线前进的距离；beta 和 gamma 是重心坐标）：

```
(const Ray &     ray,
const float3 &  p0, const float3 &     p1, const float3 &     p2,
float3 &        n,
float &         t,
float &         beta, float &    gamma )
```

自定义的求交（使用 Optixu 函数）和内置求交相比，我测试了一下速度，好像并没有什么区别，也可能是场景三角形数量比较少导致的（测试的 Mesh 里只有十几万个三角形）。

关于 attribute program，注意与 RTgeometry 不同，RTgeometrytriangles 需要提供一个属性程序来设置可以在 any-hit program 或 closest-hit program 中使用的属性。

```
RTresult rtGeometryTrianglesSetAttributeProgram(
    RTgeometrytriangles geometrytriangles, RTprogram program)
RTresult rtGeometryTrianglesGetAttributeProgram(
    RTgeometrytriangles geometrytriangles, RTprogram* program)
```

注意，属性 program 是可选的。如果未设置，则使用默认的程序，该程序提供 float2 类型的属性 rtTrianglearycentrics，两个 float 值描述了光线与三角形相交时的重心坐标。

前面提到过，属性变量提供了一种在相交程序和着色程序之间通信数据的机制（例如，曲面法线和纹理坐标）。三角形 Mesh 通过内置的求交机制进行求交。三角形上交点的重心坐标是已知的，然而，一些其他信息需要由用户计算。要计算额外的三角形相关属性（例如几何法线）时，用户可以指定属性程序。然后，这些属性被提供给 any-hit program 和 closest-hit program 使用。

通过添加内置三角形，可以添加属性程序作为计算三角形相关属性的函数。属性程序在光线与三角形成功求交后执行，在执行 any-hit program 和 closest-hit program 之前执行。定义在属性程序中的属性与在求交程序中的属性相同。

```
rtDeclareVariable( type, variableName, attribute attributeName,);
rtDeclareVariable( float2, barycentrics, attribute rtTriangleBarycentrics,);
```

如果没有定义其他属性程序，则 any-hit program 和 closest-hit program 中的属性 rtTrianglearycentrics 可用。

属性程序中有几种语义可用于识别三角形并计算所需的属性，比如”Triangle index semantic”、”Hit information semantics” 和”Triangle intersection barycentrics”。

RTgeometrytriangles 类型的三角形索引可以使用 rtGetPrimitiveIndex 语义进行查询。rtGetPrimitiveIndex 提供了基元索引，该索引类似于通常作为参数传递给自定义求交程序的索引：

```
RT_PROGRAM void intersect(int primIdx) {
  intersect_sphere<false>();
}
```

如果在几何体（geometry 或 GeometryTriangles 节点）上指定了基元索引偏移，则 rtGetPrimitiveIndex 会报告几何体的基元索引（在 N 个基元的范围内 $[0,N)$）加上偏移。

注意 rtGetPrimitiveIndex 语义在 attribute，“Any-hit programs”，“Closest-hit programs”和“Intersection and bounding box programs”中可用，也适用于非三角形。

用来访问是否光线与三角形求交的语义：

```
bool rtIsTriangleHit()
bool rtIsTriangleHitFrontFace()
bool rtIsTriangleHitBackFace()
```

1 4 多材质

与 RTgeometry 类型不同，RTgeometrytriangles 具有内置的求交机制，因此无法通过设备内部的 rtReportIntersection 指定材质索引（其实可以使用我们写的函数调用完内置求交函数以后，再调用 rtReportIntersection 指定材质索引）。三角形对多材质的支持不同。三角形网格允许对网格进行静态（预启动）分隔 (partition)，为 RTgeometrytriangles 的每个三角形分配一个材质槽。如果未另行指定，则所有三角形都只有一个材质槽。

函数设置与 GeometryTringles 一起使用的材质数量:

```
rtGeometryTrianglesSetMaterialCount
```

此数字必须等于在 GeometryInstance（GeometryTriangles 附加到该实例）处设置的材质数量。通过指定将每个三角形映射到一个材质槽的每个三角形索引来实现三角形网格的分隔（在范围 [0,num_materials-1] 内）。映射是通过函数下面函数设置的：

```
rtGeometryTrianglesSetMaterialIndices
```

实际材质在 GeometryInstance 里设置。

当附加到多个 GeometryInstance 时，可以实例化 GeometryTriangles。在这种情况下，附着到每个 GeometryInstance 的材质可能不同（实际上会导致每个几何体实例的材质不同；允许使用材质选项板 (material palettes)），但对于所有 instances，材质的数量必须相同（也就是说，如果实例化的几何三角形中，材质数最多的一个几何三角形节点有 3 种材质，那么所有的这些几何三角形都需要设置三种材质，但是如果实际实现时只用到了一种，那么就用到了每个该几何三角形中的每个三角形单独的材质索引）。

注意，前面讲过的两种情况，如果我们不自定义求交，那么就得需要下面的代码一来支持报告交点材质；如果自定义了求交，那么可以在求交函数中根据材质索引来直接访问 Buffer，即在 gpu 中访问代码二定义的 buffer：

```
// 代码一
rtGeometryTrianglesSetMaterialIndices( m_geometryTriangles,
    material_index_buffer->get(), material_index_buffer_byte_offset,
material_index_byte_stride, material_index_format )
// 代码二
optix::GeometryInstance geom_instance;
optix_mesh.geom_instance[ "material_buffer" ]->setBuffer( buffers.
    mat_indices );
```

1 5 其他内容

关于运动模糊等有关的内容我们先暂不介绍，因为目前都还没涉及运动模糊，一次性介绍完不太直观。

二 三角形的构建和使用

源码见目录 2-1。

为了更容易演示三角形的加载过程，我们自定义一个 obj 读取函数。然后实现从读取到生成 buffer 再到加载到 Optix 的全流程。

2 1 obj 读取

obj 读取的代码见 ObjReader.h 的 loadObj() 函数。注意我们实现的函数仅支持打开 nanosuit.obj 这个模型，要想打开其他的 obj 模型还需要丰富和完善其功能。

用 txt 文本格式打开 obj 文件，可以看到除了一些额外的信息（比如使用哪个 mtl 作为补充，或者纹理名称等），只有四种数据：

```
// 顶点
v -1.845563 1.146484 -0.746034
// 纹理坐标（如果大于1就相当于将纹理重新扩充然后取值，比如镜像反转warp）
vt 1.436523 0.857666
// 法向量
vn 0.088626 -0.993774 0.067354
// 表面索引，每组表示为表面的第i个顶点的v坐标/vt坐标/vn坐标（i=1,2,3）
f 53/29/53 54/30/53 55/31/54
```

使用 C++ 的文件流 fstream 就可以读取。注意 face 中顶点、纹理和法向量的索引都是从 1 开始的，所以需要都减 1。

创建 Mesh 的函数在 main.cpp 的 createGeometry() 函数中调用。

2 2 obj 转 Optix

大致分为两步：生成 Buffer 以及构建几何实例。

生成 Buffer 的过程和之前完全一样，不再赘述。构建几何实例的过程我们也都讲过，因此也不再赘述。

然后就是绑定材质了。

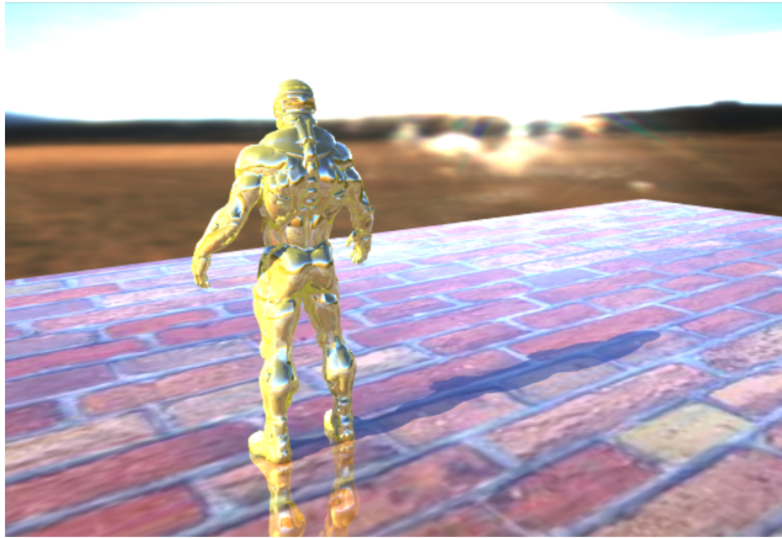
对于每个表面来说，其顶点索引和对应顶点的法向量索引可能是不一样的：

```
f 53/29/53 54/30/53 55/31/54
```

所以需要建立多个索引 Buffer，因此 optix SDK 中的 triangle_mesh.cu 是不能直接用的，我们需要修改一下。此外，直接把 mesh 的文件定义为 triangle_mesh.cu 也是不行的，因为 ustil 会先从 SDK 路径寻找该文件，因此会一直使用 optix 官方的 triangle_mesh.cu，因此我们的 mesh 文件命名为 triangle_mesh_1.cu。

其他过程与先前基本一致，大家可以自己实现一个 obj 读取和转换为 optix 的 mesh 的程序，亲自实现一遍可能比自己写代码能掌握地更牢固。

代码渲染结果如下：



三 小结

本文描述了一个独特的、不同寻常的几何基元：三角面片。Optix 的内置功能使得处理这种基元会有很高的效率。

下一本小书我们将实现一个路径追踪器，这会需要不小的工作量。

参考文献

[1] https://developer.nvidia.com/rtx/ray-tracing

[2] https://developer.nvidia.com/rtx/ray-tracing/optix

[3] https://developer.nvidia.com/blog/how-to-get-started-with-optix-7/

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

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

[6] https://developer.nvidia.com/designworks/optix/downloads/legacy

[7] https://raytracing-docs.nvidia.com/optix6/guide_6_5/index.html#guide#

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

[9] https://raytracing.github.io/books/RayTracingInOneWeekend.html

[10] https://learnopengl.com/Getting-started/Camera