LearnOptix 小项目-AI 去噪器

Dezeming Family

2023 年 5 月 9 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 年 5 月 21 日：完成第一版。

目录

一　项目简介

为了保证不同的小项目之间彼此独立，本项目直接解读 Optix SDK 源码，而不是以前自己构建工程。源码放在了 1-1 目录下。

optixDenoiser.cpp 的 main 函数中主要执行的函数是：

```
glutInitialize( &argc, argv );
createContext();
loadTrainingFile(training_file);
setupCamera();
loadGeometry();
context->validate();
glutRun();
```

showBuffer 是一个全局变量，用于定义去显示哪个 Buffer，可以用不同的按键来设置：

```
// 0-denoised（按键'd'）
// 1-original（按键'o'）
// 2-tonemapped（按键't'）
// 3-albedo（按键's'）
// 4-normal（按键'n'）
int showBuffer = 0;
```

denoiseBlend 表示原始渲染结果乘以一个系数，然后加上去噪结果乘以 1 减该系数。默认设置为 0.0，最高设置为 1.0。使用按键'0' 到'9' 来进行设置。

denoiseMode 用来表示去噪中使用的 GBuffers 有哪些。使用按键'm' 来递增 denoiseMode 值，当超过 2 的时候归 0。

```
// The denoiser mode.
// 0 - RGB only
// 1 - RGB + albedo
// 2 - RGB + albedo + normals
int denoiseMode = 0;
```

我们会用到下面的一些 Buffer，有些 Buffer 是内置函数使用的，因此在.cu 函数里没有定义：

```
// 自定义Buffer
rtBuffer<float4, 2> output_buffer;
rtBuffer<float4, 2> input_albedo_buffer;
rtBuffer<float4, 2> input_normal_buffer;
// 内置Buffer
tonemapped_buffer
// 色调映射和去噪阶段使用的内置buffer
input_buffer
output_buffer
// 训练数据的buffer
training_data_buffer
```

下面这个变量用于确定显示几帧未去噪的结果以后再显示去噪的结果：

```
int numNonDenoisedFrames = 4;
```

设置为 0 以后，就可以只显示去噪后的结果。我们可以看出，Optix 的 AI 去噪效果并不是很好，时序稳定性较差，伪影非常明显。

与先前的光线追踪和路径追踪不同，去噪中涉及后处理阶段，在 Optix5.0 中引入了内置的后处理阶段相关功能，用下面的函数来设定：

```
rtPostProcessingStageCreateBuiltin
```

二　基本的渲染代码

sqrt_num_samples 是一个 int 类型变量，其平方值就是每个像素采样的个数。比如如果是 2，那么一个像素就要采样 4 个点，因此一个像素格子首先被均分为上下左右四块，每一个小块单独做 jittering 采样。

```
unsigned int x = samples_per_pixel%sqrt_num_samples;
unsigned int y = samples_per_pixel/sqrt_num_samples;
float2 jitter = make_float2(x-rnd(seed), y-rnd(seed));
float2 d = pixel + jitter*jitter_scale;
```

之后，在渲染中需要存储 albedo 和 normal 到两个 Buffer 中：

```
rtBuffer<float4, 2> input_albedo_buffer;
rtBuffer<float4, 2> input_normal_buffer;
```

这两个 Buffer 的赋值就是每个像素的光线与物体最近的交点处的 albedo 和 normal。如果是多帧渲染的结果，那么该值就是多帧的 albedo 和 normal 的平均值。

渲染过程就是使用路径追踪，包括俄罗斯轮盘赌、面光源采样。注意.cu 文件里有个变量 countEmitted，这是记录当前 radiance 是否包含光照的布尔类型值。如果当前点是漫反射点，需要对光进行直接采样；当当前点采样反射出去光线直接与面光源相交时，就不再在 radiance 上加上面光源的发射辐射度了（黑体辐射，射到该光源上的光全部被吸收，因此采样时，相机光线与该光源相交时就会停止）。

三　后处理

下面两个变量是与测试模式相关，没有 OpenGL 交互，所以不用在意。

```
bool denoiser_perf_mode = false;
int  denoiser_perf_iter = 1;
```

与后处理有关的变量是：

```
// Post-processing
CommandList commandListWithDenoiser;
CommandList commandListWithoutDenoiser;
PostprocessingStage tonemapStage;
PostprocessingStage denoiserStage;
Buffer denoisedBuffer;
Buffer emptyBuffer;
```

其中，emptyBuffer 是一个全 0 的 Buffer，当选择只用 RGB 输入到去噪器中时，就可以把 Albedo 和 Normal 都使用该 Buffer。

与训练数据有关的变量：

```
Buffer trainingDataBuffer;
// The path to the training data file set with -t or empty
std::string training_file;
// The path to the second training data file set with -t2 or empty
std::string training_file_2;
// Toggles between using custom training data (if set) or the built in training data.
bool useCustomTrainingData = true;
// Toggles the custom data between the one specified with -t1 and -t2, if available.
bool useFirstTrainingDataPath = true;
```

后处理阶段可以使用一组具有特定名称的固定输入和输出变量进行配置。注意，即使这些变量是内置的，也必须像任何其他 OptiX 变量一样在应用程序中声明：

```
// float4 类型的 RTbuffer
input_buffer
// float4 类型的 RTbuffer
output_buffer
// 下面两个缓冲区的第四个通道将被忽略，但必须具有与输入缓冲区相同的类型（float4类型）和维度
input_albedo_buffer
input_normal_buffer
```

3 1 色调映射阶段

色调映射是后处理的第一个阶段，把 HDR 值映射到 LDR 值。

色调映射的输入图像设置为渲染的输出：

```
context["output_buffer"]
```

色调映射的输出图像设置为：

```
context["tonemapped_buffer"]
```

色调映射阶段的创立：

```
tonemapStage = context->createBuiltinPostProcessingStage("TonemapperSimple");
tonemapStage->declareVariable("input_buffer")->set(getOutputBuffer());
tonemapStage->declareVariable("output_buffer")->set(getTonemappedBuffer());
tonemapStage->declareVariable("exposure")->setFloat(0.25f);
tonemapStage->declareVariable("gamma")->setFloat(2.2f);
```

然后创建一个执行语句，就是我们指定的运行管线，我们有两种管线，一种是只包含色调映射，没有去噪阶段的管线 commandListWithoutDenoiser；另一种色调映射和去噪阶段都有的管线 commandListWithDenoiser。我们显示一下后者的创建：

```
commandListWithDenoiser = context->createCommandList();
// 加入渲染阶段
commandListWithDenoiser->appendLaunch(0, width, height);
```

```
4 // 加入色调映射阶段
5 commandListWithDenoiser->appendPostprocessingStage(tonemapStage, width,
      height);
6 // 加入去噪阶段
7 commandListWithDenoiser->appendPostprocessingStage(denoiserStage, width,
      height);
8 commandListWithDenoiser->finalize();
```

3 2 去噪阶段

OptiX 降噪器带有内置的预训练模型。该模型由称为训练数据的二进制 blob 表示，是训练的底层深度学习系统，具有大量处于不同收敛阶段的渲染图像训练得到。由于训练需要大量的计算，资源和获得足够数量的图像对可能很困难，因此通用的模型包含在 OptiX 中。该模型适用于实践中的许多渲染器，但当应用于用户自己的渲染器生成的图像时，可能并不总是导致最佳结果，因为与原始训练数据中存在的噪声特征相比，用户自己设计的渲染器和场景的渲染结果的噪声特征可能不同。

四 小结

本文介绍了 Optix 源码实现的 AI 去噪器的流程。Optix 系列写到现在大概前后用了三个月的业余时间，也算是不小的工作量了。因为项目关系，将暂停 Optix 系列的写作，等以后有时间再做项目补充。

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