LearnOptix 系列 5-纹理映射

Dezeming Family

2023 年 4 月 25 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。

目录

一　纹理知识点

本节介绍纹理的相关知识点。

1 1　纹理基本功能

OptiX 纹理支持常见的纹理映射功能，包括纹理过滤、各种 warp 模式和纹理采样。用于创建纹理对象的函数：

```
rtTextureSamplerCreate
```

每个纹理对象都与包含纹理数据的一个或多个缓冲区相关联。缓冲区可以是 $1D$、$2D$ 或 $3D$，并且可以使用函数进行设置：

```
rtTextureSamplerSetBuffer
```

设置 minification、magnification 和 mipmapping 的过滤方法函数：

```
rtTextureSamplerSetFilteringModes
```

使用下面的函数按维度指定范围 $[0.0, 1.0]$ 之外的纹理坐标的换行：

```
rtTextureSamplerSetWrapMode
```

给定纹理的最大各向异性是使用下面函数各向异性设置的：

```
rtTextureSamplerSetMaxAnisotropy
```

该值将被 clamp 在 $[1.0, 16.0]$ 的范围内。

设置 texturesampler 的索引模式的函数：

```
rtTextureSamplerSetIndexingMode
```

索引模式有两种：

```
// 归一化坐标访问
RT_TEXTURE_INDEX_NORMALIZED_COORDINATES
// 按数组索引值
RT_TEXTURE_INDEX_ARRAY_INDEX
```

比如一个数组 data 有 100 个元素，要找到第 30 个元素，按数组索引值索引，就是索引 29，按照归一化访问索引就是 0.29（坐标归一化到 [0,1] 区间）。

指定纹理值转换为归一化 float 类型值：

```
rtTextureSamplerSetReadMode
```

指定参数是：

```
RT_TEXTURE_READ_NORMALIZED_FLOAT
```

给出较完整的示例代码：

```
RTcontext context = ...;
RTbuffer tex_buffer = ...; //2D buffer
RTtexturesampler tex_sampler;
rtTextureSamplerCreate(context, &tex_sampler);
rtTextureSamplerSetWrapMode(tex_sampler, 0, RT_WRAP_CLAMP_TO_EDGE);
rtTextureSamplerSetWrapMode(tex_sampler, 1, RT_WRAP_CLAMP_TO_EDGE);
```

```
rtTextureSamplerSetFilteringModes(tex_sampler, RT_FILTER_LINEAR,
    RT_FILTER_LINEAR, RT_FILTER_NONE);
rtTextureSamplerSetIndexingMode(tex_sampler,
    RT_TEXTURE_INDEX_NORMALIZED_COORDINATES);
rtTextureSamplerSetReadMode(tex_sampler, RT_TEXTURE_READ_NORMALIZED_FLOAT);
rtTextureSamplerSetMaxAnisotropy(tex_sampler, 1.0f);
rtTextureSamplerSetBuffer(tex_sampler, 0, 0, tex_buffer);
```

OptiX 程序可以使用 CUDA C 内置的 tex1D、tex2D 和 tex3D 函数访问纹理数据。

```
rtTextureSampler<uchar4, 2, cudaReadModeNormalizedFloat> t;
...
float2 tex_coord = ...;
float4 value = tex2D( t, tex_coord.x, tex_coord.y );
```

1 2 纹理扩展功能

Optix3.9 以后，开始支持立方体 (cube)、分层 (layered)mipmapped 纹理，API 调用为：

```
rtBufferMapEx
rtBufferUnmapEx
rtBufferSetMipLevelCount
```

也移除了一些不用的功能，比如

```
rtTextureSamplerGetArraySize
rtTextureSamplerGetMipLevelCount
```

分层纹理等效于 CUDA 分层纹理和 OpenGL 纹理数组 (texture arrays)。它们是通过调用 rtBufferCreate（参数为 RT_BUFFER_LAYERED）来创建的，创建 cube maps 还需要通过参数 RT_BUFFER_CUBEMAP。在这两种情况下，缓冲区的深度维度 (depth dimension) 用于指定层数或立方体面，而不是 3D 缓冲区的厚度。

采样 CubeMap 的函数：

```
rtTexCubemap
rtTexCubemapLod
rtTexCubemapLayered
rtTexCubemapLayeredLod
```

可以通过提供用于 mipmapping 的细节级别 (level of detail) 或用于各向异性滤波的梯度 (gradients for anisotropic filtering)（比如光线微分会使用的功能）来对纹理进行采样。分层纹理（纹理阵列）需要整数层数：

```
float4 v;
if (mip_mode == MIP_DISABLE)
    v = rtTex2DLayeredLod<float4>(tex, uv.x, uv.y, tex_layer);
else if (mip_mode == MIP_LEVEL)
    v = rtTex2DLayeredLod<float4>(tex, uv.x, uv.y, tex_layer, lod);
else if (mip_mode == MIP_GRAD)
    v = rtTex2DLayeredGrad<float4>(tex, uv.x, uv.y, tex_layer, dpdx, dpdy);
```

1 3　无绑定纹理

从 OptiX 3.0 版本开始，OptiX 支持无绑定纹理 (bindless textures)。无绑定纹理允许 OptiX 程序引用纹理，而无需将其绑定到特定变量。这是通过使用纹理 ID 来实现的。使用无绑定纹理，可以在多个纹理之间动态切换，而无需在程序中明确声明所有可能的纹理，也无需手动实现切换代码。

选择的纹理集可以具有不同的属性，如 wrap mode 和不同的尺寸，从而提高了纹理阵列的灵活性。要从现有纹理采样器获取设备句柄 (device handle)，可以使用：

```
RTtexturesampler tex_sampler = ...;
int tex_id;
rtTextureSamplerGetId( tex_sampler, &tex_id );
```

纹理 ID 值是不可变的，并且在销毁其关联的纹理采样器之前一直有效。通过使用输入缓冲区或 OptiX 变量，使纹理 ID 可用于 OptiX 程序：

```
// 1D buffer
RTbuffer tex_id_buffer = ...;
unsigned int index = ...;
void* tex_id_data;
rtBufferMap( tex_id_buffer, &tex_id_data );
((int*)tex_id_data)[index] = tex_id;
rtBufferUnmap( tex_id_buffer );
```

与 CUDA C 的纹理函数类似，OptiX 程序可以使用 rtTex1D、rtTex2D 和 rtTex3D 函数以无绑定的方式访问纹理：

```
rtBuffer<int, 1> tex_id_buffer;
unsigned int index = ...;
int tex_id = tex_id_buffer[index];
float2 tex_coord = ...;
float4 value = rtTex2D<float4>( tex_id, tex_coord.x, tex_coord.y );
```

二　代码实现一：基本纹理的加载

本节代码见 2-1 目录。

为了导入图像，我们需要一个图像读入功能，这里我们用常用的 stb_lib 来读取图像：

```
// 一定要在引用头文件之前定义宏
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
// 纹理读取
int imgW, imgH, nrChannels;
unsigned char *data = stbi_load("resources/girl.png", &imgW, &imgH, &
    nrChannels, 0);
```

注意读取的文件类型一定是 png 图像，保证是四通道的，否则后面内存拷贝时就会访问出错。

纹理的实现可以参考 optixTextureSampler 官方样例，我们改为了使用 optixu 的函数来完成，并且将加载的图像作为纹理。

在 Optix 程序里，地板的纹理放在了我们定义的 girl.cu 文件：

```
// 纹理
rtTextureSampler<uchar4, 2, cudaReadModeNormalizedFloat> input_texture;
```

采样的纹理坐标来自于 parallelogram.cu 求交以后得到的纹理值。我们的背景是埃罗芒阿老师（图像文件放在了 resources 目录下）：

三　代码实现二：环境光

虽然我们已经有了不错的地板纹理，但是周围环境还不是很好，我们希望使用一个环境光让周围环境更好看。

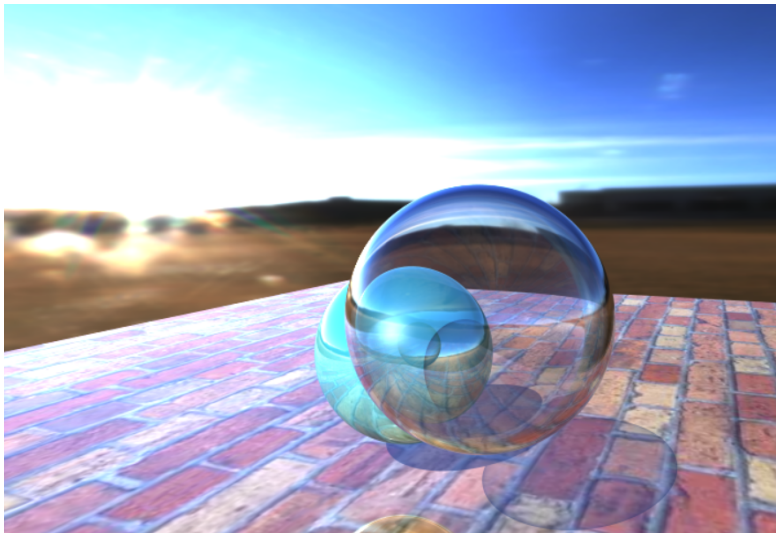
加载纹理的代码可以直接使用 loadTexture() 函数，执行默认设置（比如线性插值、归一化纹理坐标、归一化 float 读取值等）：

```
const float3 default_color = make_float3(1.0f, 1.0f, 1.0f);
const std::string texpath = "./resources/CedarCity.hdr";
context["envmap"]->setTextureSampler(sutil::loadTexture(context, texpath,
    default_color));
```

在 background.cu 中（我们把 constantbg.cu 改为了 background.cu），根据光线的方向来生成对应的全景图坐标，然后访问全景图：

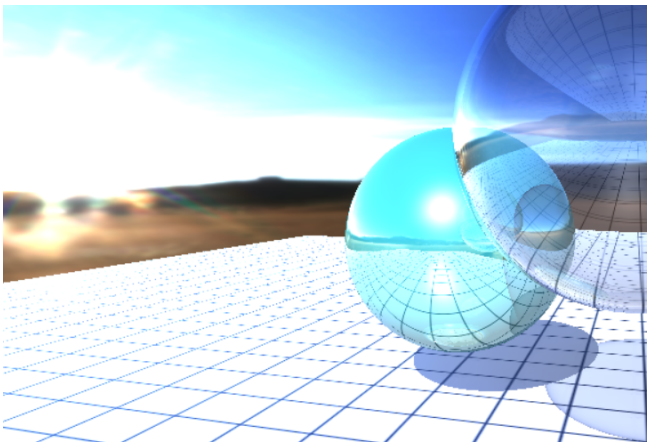
```
rtDeclareVariable(optix::Ray, ray, rtCurrentRay, );
rtTextureSampler<float4, 2> envmap;
RT_PROGRAM void envmap_miss()
{
  float theta = atan2f( ray.direction.x, ray.direction.z );
  float phi   = M_PIf * 0.5f -  acosf( ray.direction.y );
  float u     = (theta + M_PIf) * (0.5f * M_1_PIf);
  float v     = 0.5f * ( 1.0f + sin(phi) );
  prd_radiance.result = make_float3( tex2D(envmap, u, v) );
}
```

渲染结果为：



四　小结

如果我们不进行多帧累加，每个像素都在像素中心发射采样光线，那么得到的渲染结果就是：



可以看到远处的纹理断断续续的，通过 LOD 和 Mipmap 可以得到解决，但本文暂不涉及这些内容。本文介绍了一些基本的纹理操作，而更复杂的纹理操作，比如基于光线微分的抗锯齿方法的实现，我们可能在后期会以小型付费项目的形式提供给大家（坚持创作实属不易，因为暂时没有在国内备案，国外租网页服务器费用确实也比较高，如果教程有帮助，还希望大家多多支持）。

下一篇文章中，我们将会介绍渲染中最常用的三角 mesh 是如何集成到 Optix 中的，这样我们就可以加载和使用一些有趣的模型了。

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