# [OpenGL与CUDA互操作方式总结](https://www.cnblogs.com/csuftzzk/p/cuda_opengl_interoperability.html)

## 一、介绍

CUDA是Nvidia推出的一个通用GPU计算平台，对于提升并行任务的效率非常有帮助。本人主管的项目中采用了OpenGL做图像渲染，但是在数据处理方面比较慢，导致帧率一直上不来。于是就尝试把计算工作分解成小的任务，使用核函数在CUDA中加速计算。对于CUDA和OpenGL如何交互以前从来没有接触过，这次在实施时趟了不少的坑。在这里记录下OpenGL与CUDA的互操作的两种方式。

## 二、基本操作流程

OpenGL与CUDA互操作可以分成2种：

一种是OpenGL将Buffer对象注册到CUDA中去，供CUDA读写操作，然后再在OpenGL中使用。一般这种情况下注册的是VBO和PBO，VBO一般用于存储顶点坐标、索引等数据；PBO则一般用于存储图像数据，因此称作**Pixel Buffer Object。**

另一种是OpenGL将Texture对象注册到CUDA中去，经CUDA处理后得到纹理内容，然后在OpenGL中渲染出来。

不过不管是哪一种互操作类型，其操作流程是一致的：

* 在OpenGL里面初始化Buffer Object
* 在CUDA中注册OpenGL中的Buffer Object
* CUDA锁定资源，获取操作资源的指针，在CUDA核函数中进行处理
* CUDA释放资源，在OpenGL中使用Buffer Object

下面就以代码为例，讲讲2种方式的异同：

（1）OpenGL PBO/VBO在CUDA中的使用

// 初始化Buffer Object

//vertex array object

glGenVertexArrays(1, &this->VAO);

//Create vertex buffer object

glGenBuffers(2, this->VBO);

//Create Element Buffer Objects

glGenBuffers(1, &this->EBO);

//Bind the Vertex Array Object first, then bind and set vertex buffer(s) and attribute pointer(s).

glBindVertexArray(this->VAO);

// 绑定VBO后即在CUDA中注册Buffer Object

glBindBuffer(GL\_ARRAY\_BUFFER, this->VBO[0]);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(\*this->malla)\*this->numPoints, this->malla, GL\_DYNAMIC\_COPY);

cudaGraphicsGLRegisterBuffer(&this->cudaResourceBuf[0], this->VBO[0], cudaGraphicsRegisterFlagsNone);

glBindBuffer(GL\_ARRAY\_BUFFER, this->VBO[1]);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(\*this->malla)\*this->numPoints, this->malla, GL\_DYNAMIC\_COPY);

cudaGraphicsGLRegisterBuffer(&this->cudaResourceBuf[1], this->VBO[1], cudaGraphicsRegisterFlagsNone);

// 在CUDA中映射资源，锁定资源

cudaGraphicsMapResources(1, &this->cudaResourceBuf[0], 0);

cudaGraphicsMapResources(1, &this->cudaResourceBuf[1], 0);

point \*devicePoints1;

point \*devicePoints2;

size\_t size =sizeof(\*this->malla)\*this->numPoints;

// 获取操作资源的指针，以便在CUDA核函数中使用

cudaGraphicsResourceGetMappedPointer((void \*\*)&devicePoints1, &size, this->cudaResourceBuf[0]);

cudaGraphicsResourceGetMappedPointer((void \*\*)&devicePoints2, &size, this->cudaResourceBuf[1]);

// execute kernel

dim3 dimGrid(20, 20, 1);

dim3 dimBlock(this->X/dimGrid.x, this->Y/dimGrid.y, 1);

modifyVertices<<<dimGrid, dimBlock>>>(devicePoints1, devicePoints2,this->X, this->Y);

modifyVertices<<<dimGrid, dimBlock>>>(devicePoints2, devicePoints1,this->X, this->Y);

// 处理完了即可解除资源锁定，OpenGL可以开始利用处理结果了。

// 注意在CUDA处理过程中，OpenGL如果访问这些锁定的资源会出错。

cudaGraphicsUnmapResources(1, &this->cudaResourceBuf[0], 0);

cudaGraphicsUnmapResources(1, &this->cudaResourceBuf[1], 0);

值得注意的是，由于这里绑定的是VBO，属于Buffer对象，因此调用的CUDA API是这两个：

cudaGraphicsGLRegisterBuffer();

cudaGraphicsResourceGetMappedPointer();

（2）OpenGL Texture在CUDA中的使用

// 初始化两个Texture并绑定

cudaGraphicsResource\_t cudaResources[2];

GLuint textureID[2];

glEnable(GL\_TEXTURE\_2D);

glGenTextures(2, textureID);

glBindTexture(GL\_TEXTURE\_2D, textureID[0]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, 1000, 1000, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, NULL);

glBindTexture(GL\_TEXTURE\_2D, textureID[1]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, 1000, 1000, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, NULL);

// 在CUDA中注册这两个Texture

cudaError\_t err = cudaGraphicsGLRegisterImage(&cudaResources[0], textureID[0], GL\_TEXTURE\_2D, cudaGraphicsRegisterFlagsWriteDiscard);

if(err != cudaSuccess)

{

std::cout <<"cudaGraphicsGLRegisterImage: "<< err <<"Line: "<< \_\_LINE\_\_;

return -1;

}

err = cudaGraphicsGLRegisterImage(&cudaResources[1], textureID[1], GL\_TEXTURE\_2D, cudaGraphicsRegisterFlagsWriteDiscard);

if (err != cudaSuccess)

{

std::cout <<"cudaGraphicsGLRegisterImage: "<< err <<"Line: "<< \_\_LINE\_\_;

return -1;

}

//在CUDA中锁定资源，获得操作Texture的指针，这里是CudaArray\*类型

cudaError\_t err = cudaGraphicsMapResources(2, cudaResource, 0);

err = cudaGraphicsSubResourceGetMappedArray(&this->cuArrayL, cudaResource[0], 0, 0);

err = cudaGraphicsSubResourceGetMappedArray(&this->cuArrayR, cudaResource[1], 0, 0);

// 数据拷贝至CudaArray。这里因为得到的是CudaArray，处理时不方便操作，于是先在设备内存中

// 分配缓冲区处理，处理完后再把结果存到CudaArray中，仅仅是GPU内存中的操作。

cudaMemcpyToArray(cuArrayL, 0, 0, pHostDataL, imgWidth\*imgHeight \*sizeof(uchar4), cudaMemcpyDeviceToDevice);

cudaMemcpyToArray(cuArrayR, 0, 0, pHostDataR, imgWidth\*imgHeight \*sizeof(uchar4), cudaMemcpyDeviceToDevice);

//处理完后即解除资源锁定，OpenGL可以利用得到的Texture对象进行纹理贴图操作了。

cudaGraphicsUnmapResources(1, &cudaResource[0], 0);

cudaGraphicsUnmapResources(1, &cudaResource[1], 0);

注意这里因为使用的是Texture对象，因此使用了不同的API：

cudaGraphicsGLRegisterImage();

cudaGraphicsSubResourceGetMappedArray();

VBO/PBO是属于OpenGL Buffer对象，而OpenGL Texture则是另一种对象。因此，两种类型的处理需要区别对待。在这个地方耽搁了很久，就是因为没有看文档说明。下面一段话正是对这种情况的说明：

From the CUDA Reference Guide entry for `cudaGraphicsResourceGetMappedPointer()`:

> If resource is not a buffer then it cannot be accessed via a pointer and cudaErrorUnknown is returned.

From the CUDA Reference Guide entry for `cudaGraphicsSubResourceGetMappedArray()`:

> If resource is not a texture then it cannot be accessed via an array and cudaErrorUnknown is returned.

In other words, use \*\*GetMappedPointer\*\* for mapped buffer objects. Use \*\*GetMappedArray\*\* for mapped texture objects.

## 三、参考链接

* <http://stackmirror.cn/page/4ejhmgxan1w>
* <https://stackoverflow.com/questions/21765604/draw-image-from-vertex-buffer-object-generated-with-cuda-using-opengl>
* <https://stackoverflow.com/questions/19244191/cuda-opengl-interop-draw-to-opengl-texture-with-cuda?rq=1>
* <https://www.3dgep.com/opengl-interoperability-with-cuda/>