CUDA 和 OpenGL 互操作的实现及分析*

刘进锋,郭 雷
(西北工业大学 自动化学院,陕西 西安 710129)

摘 要:CUDA 和 OpenGL 互操作的基本方式是使用 CUDA 生成数据,再利用 OpenGL 在屏幕上绘制出数据所表示的图形。两者的结合可以通过使用 OpenGL 的 PBO(像素缓冲区对象)或 VBO(顶点缓冲区对象)两种方式来实现。描述了 CUDA 和 OpenGL 互操作的步骤并展示了一个使用 PBO 的实例。该实例运行结果表明,互操作的方式比单纯使用 OpenGL 方式快了 7~8 倍。

关键词:CUDA;OpenGL;像素缓冲区对象;顶点缓冲区对象
中图分类号:TP311 文献标识码:A 文章编号:1674-7720(2011)23-0040-03

Realization and analysis of CUDA and OpenGL interoperation

Liu Jinfeng, Guo Lei
(School of Automation, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract: The basic mode of CUDA and OpenGL interoperation is using CUDA to generate data and using OpenGL to display data. The mixing of them can be achieved by utilizing PBO (Pixel Buffer Object) or VBO (Vertex Buffer Object). The processes of CUDA and OpenGL interoperation are described and an instance is presented in this paper. Experiment of this instance shows that using interoperation gain 7~8 times fast than only using OpenGL.

Key words: CUDA; OpenGL; PBO; VBO

1 CUDA 与 OpenGL 概述

OpenGL 是图形硬件的软件接口,它是在 SGI 等多家世界著名的计算机公司的倡导下,以 SGI 的 GL 三维图形库为基础制定的一个通用、共享的、开放式的、性能卓越的三维图形标准。OpenGL 在医学成像、地理信息、石油勘探、气候模拟以及娱乐动画上有着广泛应用,它已经成为高性能图形和交互式视景处理的工业标准。

OpenGL 不是一种编程语言,而是一种 API(应用程序编程接口)。程序员可以使用某种编程语言(如 C 或 C++)编写绘图软件,其中调用了一个或多个 OpenGL 库函数。作为一种 API,OpenGL 遵循 C 语言的调用约定。OpenGL 开发资料可参考文献[1]和参考文献[2]。

图形处理器(GPU)原本是处理计算机图形的专用设备,近十年来,由于高清晰度复杂图形实时处理的需求,GPU 发展成为高并行度、多线程、多核的处理器。目前,主流 GPU 的运算能力已超过主流通用 CPU,从发展趋势上来看将来差距会越拉越大。为了合理地利用 GPU 资源,CUDA(统一计算设备架构)应运而生。CUDA 是一种由 NVIDIA 推出的通用并行计算架构[3],该架构使 GPU 能够解决复杂的计算问题,并且由于 CUDA 编程语言基于标准的 C 语言,从而大大提高了可编程性。

CUDA 和 OpenGL 互操作的基本方式是使用 CUDA 生成数据,然后使用 OpenGL 在屏幕上绘制出数据所表示的图形。两者的结合可以通过两种方式来实现:

(1)使用 OpenGL 的 PBO(像素缓冲区对象)。在该方式下,CUDA 直接生成像素数据,OpenGL 显示这些像素;

(2)使用 OpenGL 的 VBO(顶点缓冲区对象)。在该方式下,CUDA 生成顶点网格数据,OpenGL 可以根据需要绘制出平滑的表面图或线框图或一系列顶点。

这两种方式的核心都是利用 cudaGLMapBufferObject 函数将 OpenGL 的缓冲区映射到 CUDA 的内存空间上,这样,程序员就可以充分利用 CUDA 的优点写出性能高的程序在该内存空间上生成数据,这些数据不需要传送,OpenGL 可以直接使用。如果不使用 CUDA,这些数据需要由 CPU 来计算产生。一方面,CPU 的计算速度通常

*基金项目:国家自然科学基金资助项目(61063020)

比GPU慢；另一方面，这些数据需要传送到GPU上以供OpenGL显示使用。鉴于此，当数据量很大时，CUDA和OpenGL的混合使用效果明显。

2 CUDA和OpenGL互操作的过程[4]

CUDA和OpenGL互操作具体步骤如下：

(1)创建窗口及OpenGL运行环境。

(2)设置OpenGL视口和坐标系。要根据绘制的图形是2D还是3D等具体情况设置。(1)和(2)是所有OpenGL程序必需的，这里也没什么特殊之处，需要注意的是，后面的一些功能需要OpenGL 2.0及以上版本支持，所以在这里需要进行版本检查。

(3)创建CUDA环境。可以使用cuGLCtxCreate或cudaGLSetGLDevice来设置CUDA环境。该设置一定要放在其他CUDA的API调用之前。

(4)产生一个或多个OpenGL缓冲区用以和CUDA共享。使用PBO和使用VBO差不多，只是有些函数调用参数不同。以下是具体过程。

```
GLuint bufferID;
glGenBuffers(1,&bufferID);          //产生一个buffer ID
glBindBuffer(parameter1,bufferID);
    //将其设置为当前非压缩缓冲区，如果是PBO方式，
    parameter1设置为GL_PIXEL_UNPACK_BUFFER，如果
    是VBO方式，parameter1设置为GL_ARRAY_BUFFER
glBufferData (parameter1,parameter2,NULL,GL_DYNAMIC
_COPY);
//给该缓冲区分配数据，PBO方式下，parameter1设置为
GL_PIXEL_UNPACK_BUFFER，parameter1设置为图像的长度*
宽度*4。VBO方式下，parameter1设置为GL_ARRAY_BUFFER，
parameter2设置为顶点数*16，因为每个顶点包含3个浮点
坐标(x,y,z)和4个颜色字节(RGBA)，这样一个顶点包含16 B
```

(5)用CUDA登记缓冲区。登记可以使用cuGLRegisterBufferObject或cudaGLRegisterBufferObject，该命令告诉OpenGL和CUDA驱动程序该缓冲区为二者共同使用。

(6)将OpenGL缓冲区映射到CUDA内存。可以使用cuGLMapBufferObject或cudaGLMapBufferObject，它实际是将CUDA内存的指针指向OpenGL的缓冲区，这样如果只有一个GPU，就不需要数据传递。当映射完成后，OpenGL不能再使用该缓冲区。

(7)使用CUDA往该映射的内存写图像数据。前面的准备工作在这里真正发挥作用了，此时可以调用CUDA的kernel，像使用全局内存一样使用映射了的缓冲区，向其中写数据。

(8)取消OpenGL缓冲区映射。要等前面CUDA的活动完成以后，使用cuGLUnmapBufferObject或cudaGLUnmapBufferObject函数取消映射。

(9)前面的步骤完成以后就可以真正开始绘图了，OpenGL的PBO和VBO的绘图方式不同，分别为以下两个过程。

①如果只是绘制平面图形，需要使用OpenGL的PBO及纹理。

```
glEnable(GL_TEXTURE_2D);                    //使纹理可用
glGenTextures(1,&textureID);          //生成一个textureID
glBindTexture(GL_TEXTURE_2D,textureID);
                                  //使该纹理成为当前可用纹理
glTexImage2D (GL_TEXTURE_2D,0,GL_RGBA8,Width,
Height,0,GL_BGRA,GL_UNSIGNED_BYTE,NULL);
//分配纹理内存。最后的参数设置数据来源，这里设置
为NULL，表示数据来自PBO，不是来自主机内存
glTexParameteri (GL_TEXTURE_2D,GL_TEXTURE_MIN
_FILTER,GL_LINEAR);
glTexParameteri (GL_TEXTURE_2D,GL_TEXTURE_MAG_
FILTER,GL_LINEAR);//必须设置滤波模式，GL_LINEAR允
许图形伸缩时线性差值。如果不需要线性差值，可以用
GL_TEXTURE_RECTANGLE_ARB代替GL_TEXTURE_2D以提
高性能，同时在glTexParameteri()调用里使用GL_NEAREST
替换GL_LINEAR
```

然后就可以指定4个角的纹理坐标，绘制长方形了。

②绘制3D场景，需要使用VBO。

```
glEnableClientState(GL_VERTEX_ARRAY);
                                  //使顶点和颜色数组可用
glEnableClientState(GL_COLOR_ARRAY);
glVertexPointer(3,GL_FLOAT,16,0);
                                  //设置顶点和颜色指针
glColorPointer(4,GL_UNSIGNED_BYTE,16,12);
glDrawArrays(GL_POINTS,0,numVerticies);
//根据顶点数据绘图，参数可以使用GL_LINES,
GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES,
GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS,
GL_QUAD_STRIP,GL_POLYGON
```

(10)前后缓存区来回切换，实现动画显示效果。调用SwapBuffers()，缓冲区切换通常会在垂直刷新间隙来处理，因此，可以在控制面板上关掉垂直同步，使得缓冲区切换立刻进行。

3 CUDA和OpenGL互操作性能实例分析

3.1 测试实例

这是一个相对简单的实例，其主要功能是不断地动态改变一个纹理图案中每个像素的颜色并显示。该实例使用了OpenGL的PBO并利用了OpenGL与CUDA互操作方式，纹理图案数据的生成主要由CUDA的kernel函数完成，完整程序及CUDA的kernel函数请参看参考文献[5]。

如果不使用CUDA，整个程序结构变化不大，主要差别是生成该纹理图案的函数在CPU上运行，因而该函数及其调用方式要重写，具体函数如下：

```
void kernel (uchar4*pos,unsigned int width,unsigned int
```

```
height, float time)
{ unsigned int index, x, y;
  for(x=0; x<width; x++)
    for(y=0; y<height; y++)
{   unsigned char r=(x+(int)time)&0xff;
    unsigned char g=(y+(int)time)&0xff;
    unsigned char b=((x+y)+(int)time)&0xff;
index=x*width+y;
    pos[index].w=0;
    pos[index].x=r;
    pos[index].y=g;
    pos[index].z=b;
 }
}
```

其中，参数 pos 表示像素数组，width 为图像宽度，height 为图像高度，time 是每次调用该函数时固定递增的一个值。

3.2 测试结果

上述实例在两种环境中做了实验，CUDA 版本都是 3.2。测试环境 1 的主要配置如下：CPU 为 Intel Core i3-M380，主频为 2.53 GHz，GPU 为 NVIDIA NVS 3100M，内存为 2 GB。测试环境 2 的主要配置如下：CPU 是 Intel Core2 duo E7400，主频为 2.8 GHz，GPU 使用 GeForce 9800 GTX+，内存为 2 GB。测试时，显示设置的垂直同步要关闭。

测试时设置纹理图像的长和宽都是 512，CUDA 的线程块为 1 024，每个线程块内的线程数为 256，在 OpenGL 的显示回调函数里统计 f/s(刷新率)，结果如表 1 所示。

从实验结果可以看出，CUDA 与 OpenGL 结合的方式效果显著，显示速度比不使用 CUDA 提高了 7~8 倍。

CUDA 是一种较新的方便使用 GPU 进行通用计算的架构，OpenGL 是图形处理的工业标准。两者的互操作充分利用了 GPU 的特点，因而显得非常自然和合理，实验验证了两者配合使用的效果。该方式为高性能图形图像显示及科学计算可视化提供了良好的模式架构。

表 1 两种测试环境下使用和不使用 CUDA 的性能比较

测试环境	测试环境 1		测试环境 2	
方式	CUDA 与 OpenGL 结合	不使用 CUDA	CUDA 与 OpenGL 结合	不使用 CUDA
刷新率	510	60	1 510	208

参考文献

[1] WRIGHT R S, LIPCHAK B, HAEMEL N. OpenGL superbible (Fourth Edition)[M]. Addison-Wesley, 2007.
[2] AHN S H. The OpenGL tutorials [OL]. [2011-09-01].http://songho.ca/opengl/.
[3] NVIDIA Corporation. NVIDIA CUDA programming Guide Version 3.2[OL]. Mar. 2011,http://developer.nvidia.com/cuda.
[4] STAM J. What every CUDA programmer needs to know about OpenGL[OL]. [2011-09-01].http://nvidia.fullviewmedia.com/GPU2009/1001-valley-1055.html.
[5] FARBER R. CUDA, supercomputing for the masses: Part 15 [OL]. [2011-09-01].http://www.drdobbs.com/architecture-and-design/222600097.

(收稿日期：2011-09-13)

作者简介：
刘进锋，男，1971 年生，博士研究生，副教授，主要研究方向：图像处理，GPU 通用计算。
郭雷，男，1956 年生，博士生导师，教授，主要研究方向：图像处理，模式识别。