Attribution/License

- Original Materials developed by Mike Shah, Ph.D. (www.mshah.io)
- This work was initially developed for the 2020 PetaScale Blue Waters Institute
- Funding for the development of this work came from http://shodor.org/
- This slideset and associated source code may be used freely
  - Attribution is appreciated but not necessary

Graphics Interop with OpenGL

Interoperability on the GPU

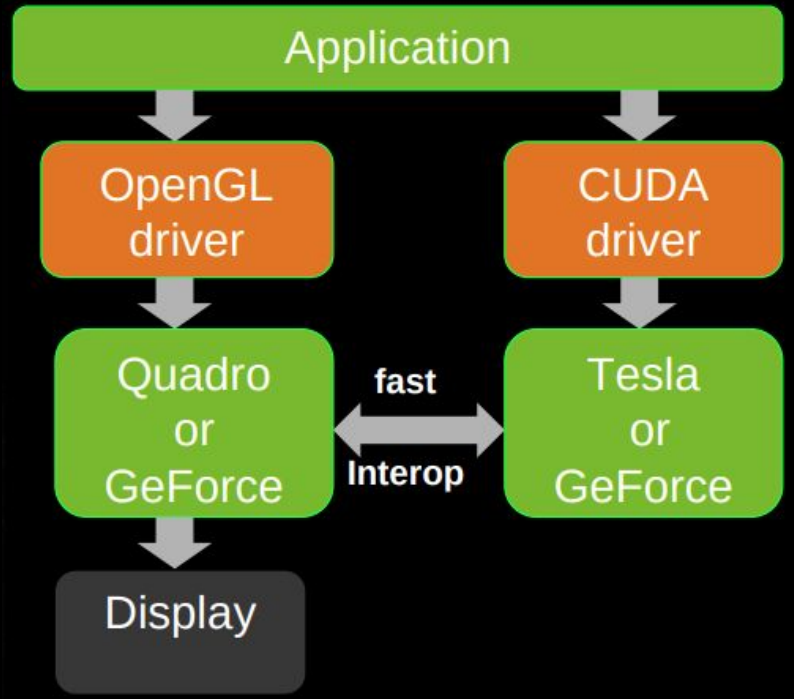
- Interoperability is the ability of a computer system to exchange and make use of information.
- In this lesson we are going to look at interoperability on the GPU within a graphics application

GPU APIs

- There are many ways to utilize our GPU
  - This series is about using CUDA for general purpose computation on the GPU
- However, another popular API is in the domain of Graphics: OpenGL
  - OpenGL is a graphics specific API implemented by companies like NVidia, AMD, and Intel to create graphics applications (e.g. games or visualizations)
- So using the OpenGL API, this is another way to ‘ship’ data to the GPU.
  - Often times however, it can be nice to manipulate that data using CUDA as well.

CUDA and OpenGL interop (1/2)

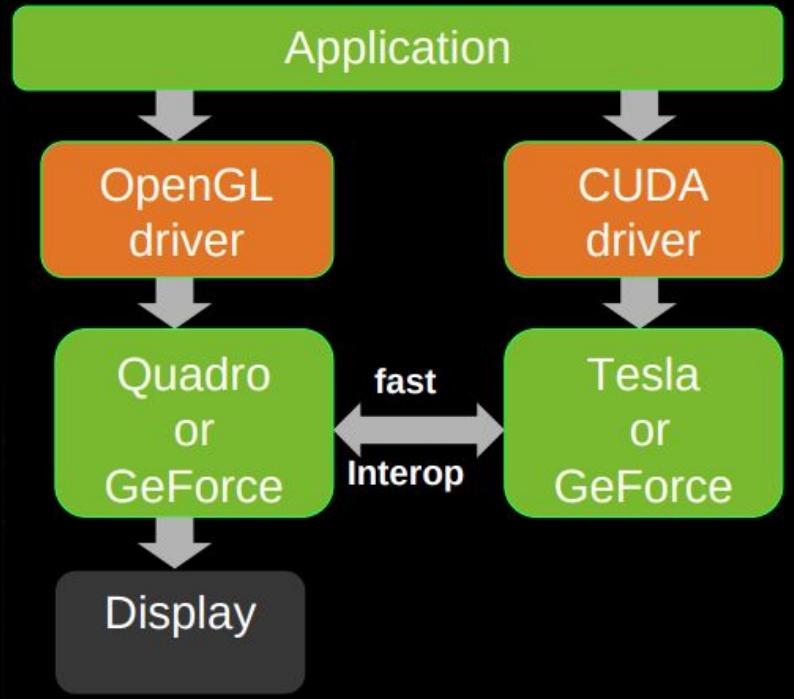
- To the right is a diagram showing the basic idea
  - Our application can be using OpenGL and CUDA
  - And at some point the information that is transferred to our graphic card (or multiple graphic cards) can talk to each other and access information sent from each of the APIS
  - Then finally the display is sent to the user.



https://on-demand.gputechconf.com/gtc/2010/presentations/S12310-OpenGL-Pre-Conference-Tutorial.pdf

CUDA and OpenGL interop (2/2)

- Our goal is going to be to build a graphics application that:
  - Uses OpenGL for rendering
  - Uses CUDA for general purpose calculations on the GPU



https://on-demand.gputechconf.com/gtc/2010/presentations/S12310-OpenGL-Pre-Conference-Tutorial.pdf

CUDA + OpenGL

- Sometimes it’s a bit unclear why we cannot use either API to do both.
- Remember however that each was designed for a specific purpose.

- CUDA excels at performing calculations, generating data, and doing other image manipulation tasks (e.g. our gray_scale filter where we can actually change values and preserver those changes in memory)
- OpenGL excels at drawing pixels and vertices, and rendering geometry (i.e. rasterizing) very fast

CUDA + OpenGL

- CUDA uses the functions we have been learning to perform memory management (cudaMalloc)
- OpenGL uses generic buffers (index buffer objects, vertex buffer objects, pixel buffer objects) to store data
- CUDA + OpenGL talk using a few interop functions
    - Map and UnMap in order to move OpenGL buffers into CUDA’s memory space on the GPU

Our Application

Our Goal: To render a shape to the screen using OpenGL, and manipulate (i.e. do the computation) on CUDA.

- Now this sounds like a BIG problem, so let’s break it into smaller pieces. There are a lot of moving parts

Breakdown

- Let’s break our BIG problem down into some smaller steps:

1. We need to create a window and perform some general setup
   a. For this we will use SDL2
   b. You could use other tools like freeglut for instance.
2. We need to enable OpenGL and create an object
3. We need to create a CUDA kernel that does *something* interesting to that geometry

Windowing

SDL2

OpenGL Setup

CUDA Setup

OpenGL Interoperability with CUDA

- CUDA provides the following documentation for OpenGL Interoperability which we can refer to as a guide
  - https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__OPENGL.html

5.13. OpenGL Interoperability

This section describes the OpenGL interoperability functions of the CUDA runtime application programming interface. Note that mapping of OpenGL resources is performed with the graphics API agnostic, resource mapping interface described in Graphics Interopability.

Enumerations

```
enum cudaGLDeviceList
```

Functions

```
__host__cudaError_t cudaGLGetDevices ( unsigned int* pCudaDeviceCount, int* pCudaDevices, unsigned int  cudaDeviceCount, cudaGLDeviceList deviceList )
```
Gets the CUDA devices associated with the current OpenGL context.

```
__host__cudaError_t cudaGraphicsGLRegisterBuffer ( cudaGraphicsResource** resource, GLuint buffer, unsigned int  flags )
```
Registers an OpenGL buffer object.

```
__host__cudaError_t cudaGraphicsGLRegisterImage ( cudaGraphicsResource** resource, GLuint image, GLenum target, unsigned int  flags )
```
Register an OpenGL texture or renderbuffer object.

```
__host__cudaError_t cudaWGLGetDevice ( int* device, HGPUNV hGpu )
```
Gets the CUDA device associated with hGpu.