CUDA SAMPLES

TRM-06704-001_v5.5 | May 2013

Reference Manual

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1. NEW FEATURES

1.1. New Features in CUDA Toolkit 5.5

NVIDIA® CUDA™ Toolkit version 5.5 introduces some exciting new features and capabilities.

1.1.1. CUDA Version 5.5 Highlights

- Performance improvements in CUDA toolkit for Kepler GPUs (SM 3.0 and SM 3.5)
- Makefiles projects have been updated to properly find search default paths for OpenGL, CUDA, MPI, and OpenMP libraries for all OS Platforms (Mac, Linux x86, Linux ARM).
- Linux and Mac project Makefiles now invoke NVCC for building and linking projects.
- Added **`0_Simple/cppOverload`** - new SDK sample that demonstrates how to use C ++ overloading with CUDA.
- Added **`6_Advanced/cdpBezierTesselation`** - new SDK sample that demonstrates how to use NPP for JPEG compression on the GPU
- Added **`7_CUDALibrariess/jpegNPP`** - new SDK sample that demonstrates how to use NPP for JPEG compression on the GPU.
- CUDA Samples now have better integration with Nsight Eclipse IDE.
- **`6_Advanced/ptxjit`** sample now includes a new API to demonstrate PTX linking at the driver level.

1.1.2. New CUDA 5.5 Code Samples

`cdpBezierTesselation`

This sample demonstrates an advanced method of implenting Bezier Line Tessellation using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

cppOverload

This sample demonstrates how to use C++ function overloading on the GPU.

jpegNPP

This sample demonstrates a simple image processing pipline. First, a JPEG file is huffman decoded and inverse DCT transformed and dequantized. Then the different plances are resized. Finally, the resized image is quantized, forward DCT transformed and huffman encoded.

ptxjit

This sample uses the Driver API to just-in-time compile (JIT) a Kernel from PTX code. Additionally, this sample demonstrates the seamless interoperability capability of the CUDA Runtime and CUDA Driver API calls. For CUDA 5.5, this sample shows how to use cuLink* functions to link PTX assembly using the CUDA driver at runtime.

1.2. New Features in CUDA Toolkit 5.0

NVIDIA® CUDA™ Toolkit version 5.0 introduces some exciting new features and capabilities. To illustrate the capabilities and advantages of the new features, the CUDA Toolkit includes many new and improved code samples. In addition, existing code samples have been upgraded to take advantage of the new features. This document serves as a guide to the new code samples as they relate to the new CUDA Toolkit Version 5.0 and Version 5.0 feature list.

1.2.1. CUDA Version 5.0 Highlights

- Native support for Kepler GPUs (SM 3.5), with CUDA Dynamic Parallelism as a new CUDA 5.0 feature.
- Overall improvements in driver and toolkit for Kepler GPUs (SM 3.0) performance.
- All projects and Makefiles have been updated accordingly.
- New directory structure for CUDA samples. Samples are classified accordingly to categories: **0_Simple**, **1_Utilities**, **2_Graphics**, **3_Imaging**, **4_Finance**, **5_Simulations**, **6_Advanced**, and **7_CUDALibraries**

1.2.2. CUDA Dynamic Parallelism Samples in CUDA 5.0 and CUDA 5.5

cdpSimplePrint

This sample demonstrates simple **printf** implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

cdpSimpleQuickSort

This sample demonstrates a simple quicksort implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

cdpAdvancedQuickSort

This sample demonstrates an advanced quicksort implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

cdpBezierTesselation

This sample demonstrates an advanced method of implenting Bezier Line Tessellation using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

cdpLUDecomposition

This sample demonstrates LU Decomposition implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

cdpQuadTree

This sample demonstrates Quad Trees implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

simpleDevLibCUBLAS

This sample implements a simple CUBLAS function calls that call GPU device API library running CUBLAS functions. CUBLAS device code functions take advantage of CUDA Dynamic Parallelism and requires compute capability of 3.5 or higher.

1.2.3. New Revised CUDA Code Samples

simpleIPC

This CUDA Runtime API sample is a very basic sample that demonstrates Inter Process Communication with one process per GPU for computation. Requires Compute Capability 2.0 or higher and a Linux Operating System.

simpleSeparateCompilation

This sample demonstrates a CUDA 5.0 feature, the ability to create a GPU device static library and use it within another CUDA kernel. This example demonstrates how to pass in a GPU device function (from the GPU device static library) as a function pointer to be called. Requires Compute Capability 2.0 or higher.

bindlessTexture

This example demonstrates use of **cudaSurfaceObject**, **cudaTextureObject**, and MipMap support in CUDA. Requires Compute Capability 3.0 or higher.

stereoDisparity

A CUDA program that demonstrates how to compute a stereo disparity map using SIMD SAD (Sum of Absolute Difference) intrinsics. Requires Compute Capability 2.0 or higher.

1.3. New Features in CUDA Toolkit 4.2

segmentationTreeThrust

This example demonstrates a method to build image segmentation trees using Thrust. This algorithm is based on Boruvka's MST algorithm.

1.4. New Features in CUDA Toolkit 4.1

MersenneTwisterGP11213

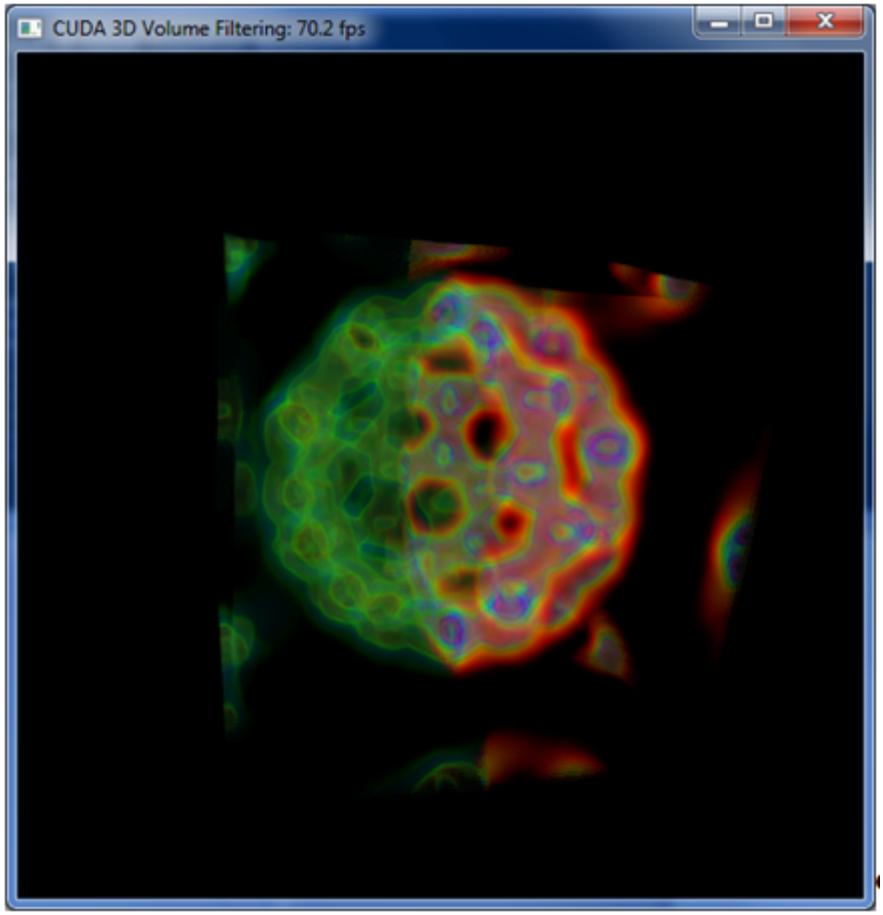
This sample implements Mersenne Twister GP11213, a pseudorandom number generator using the **CURAND** library.

HSOpticalFlow

When working with image sequences or video it's often useful to have information about objects movement. Optical flow describes apparent motion of objects in image sequence. This sample is a Horn-Schunck method for optical flow written using CUDA.

volumeFiltering

This sample demonstrates basic volume rendering and filtering using 3D textures.



simpleCubeMapTexture

This sample demonstrating how to use **texcubemap** fetch instruction in a CUDA C program.

simpleAssert

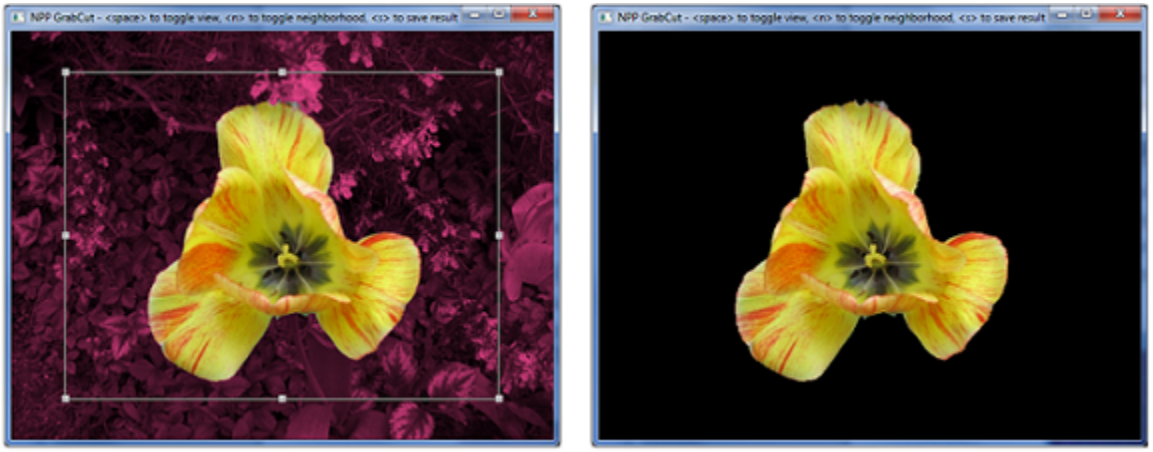
This sample demonstres how to use GPU assert in a CUDA C program.

NPP

For additional information about **NPP**, please refer to the document *NPP_Library.pdf* included with the CUDA toolkit.

grabcutNPP

CUDA implementation of Rother et al. GrabCut approach using the 8 neighborhood **NPP** Graphcut primitive introduced in CUDA 4.1. (C. Rother, V. Kolmogorov, A. Blake. *GrabCut: Interactive Foreground Extraction Using Iterated Graph Cuts. ACM Transactions on Graphics (SIGGRAPH'04)*, 2004).



Chapter 2. GETTING STARTED

This chapter documents minimum requirements and installation instructions followed by details on how to use the samples with your own CUDA projects.

2.1. Supported OS Platforms and Compilers

2.1.1. Supported Windows Platforms

OS Platform Support with CUDA 5.5

- Added support for Visual Studio 2012

OS Platform Support with CUDA 5.0

- Added support for Windows 8
- Removed support for Visual Studio 2005

OS Platform Support with CUDA 5.5

- Added support for Windows 8
- Removed support for Visual Studio 2005

OS Platform Support with CUDA 4.2 and 4.1

- No changes

OS Platform Support with CUDA 4.0

- New compilers supported

  Visual Studio 10 (2010)
- Continued supported compilers

  Visual Studio 9 (2008)

- Continued supported OS

  Windows XP, Windows Vista, Windows 7
  Windows Server 2008 and 2008 R2

OS Platform Support added to CUDA 3.0 Release

- Windows 7 32 and 64
- Windows Server 2008 and 2008 R2

OS Platform Support to CUDA 2.2

- Vista 32 and 64bit, WinXP 32 and 64-bit

  Visual Studio 9 (2008)

2.1.2. Supported Linux Platforms

OS Platform Support with CUDA 5.5

- New OS Platforms added

  Ubuntu 12.04 (gcc 4.6)
  Ubuntu 12.10 (gcc 4.7)
  Fedora18 (64-bit only) (gcc 4.7)
  OpenSUSE-12.2 (gcc 4.6.2, glibc 2.13)
  ICC Compiler 12.1 64-bit
- Platforms continued support

  RHEL 5.5+ 64-bit (gcc 4.1.2, glibc 2.5)
  RHEL 6.X (gcc 4.4.5, glibc 2.12)
  Mac OSX 10.8.x
  Mac OSX 10.7.x
  SLES-11 SP1 (gcc 4.3.4, glibc 2.11.1)
  SLES-11 SP2 (gcc 4.3.4, glibc 2.11.3)
  ICC Compiler 12.1
  Windows Server 2008 R2
  Windows XP
  Windows Vista/Win7/Win8
- Platforms no longer supported

  Fedora16 (gcc 4.6.2, glibc 2.14.90)
  Ubuntu-11.04 (gcc 4.4.5, glibc 2.12.1)
  Ubuntu-11.10 (gcc 4.6.1, glibc 2.13)

OS Platform Support with CUDA 5.0

- New OS Platforms added

  Ubuntu 11.10 (gcc 4.6.2, glibc 2.13)
  Fedora16 (gcc 4.6.1, glibc 2.12.90)
  RHEL 5.5+ 64-bit (gcc 4.1.2, glibc 2.5)
  RHEL 6.X (gcc 4.4.5, glibc 2.12)
  OpenSUSE-11.2 (gcc 4.5.1, glibc 2.11.3)
  OpenSUSE-12.1 (gcc 4.6.2, glibc 2.13)
  ICC Compiler 12.1 64-bit

- Platforms no longer supported

  ICC Compiler 11.1 64-bit
  RHEL 5.5+ 32-bit (gcc 4.1.2, glibc 2.5)
  OpenSUSE-11.2 (gcc 4.4.1, glibc 2.10.1)
  SLES-11.1 (gcc 4.3.4, glibc 2.11.1)
  Fedora14 (gcc 4.5.1, glibc 2.12.90)
  Ubuntu-11.04 (gcc 4.5.2, glibc 2.13)

OS Platform Support with CUDA 4.2

- New OS Platforms added

  OpenSUSE-11.2 (gcc 4.5.1, glibc 2.11.3)

- Platforms no longer supported

  OpenSUSE-11.2 (gcc 4.4.1, glibc 2.10.1)

OS Platform Support with CUDA 4.1

- New OS Platforms added

  Ubuntu 11.04,
  Fedora 14,
  RHEL-5.5, 5.6, 5.7 (32-bit and 64-bit)
  RHEL-6.X (6.0, 6.1) (64-bit only),
  ICC Compiler 11.1 (32-bit and 64-bit) Linux

- Continued OS Platforms

  SLES 11.1,
  Ubuntu 10.04,
  OpenSUSE-11.2 (gcc 4.4.1, glibc 2.10.1)

- Platforms no longer supported

  Ubuntu 10.10,

Fedora 13,
RHEL-4.8

OS Platform Support with CUDA 4.0

- New OS Platforms added

  SLES11-SP1,
  RHEL-6.0 (64-bit only),
  Ubuntu 10.10
- Continued OS Platforms

  OpenSUSE-11.2
  Fedora 13,
  RHEL-4.8 (64-bit only),
  RHEL-5.5
- Platforms no longer supported

  RHEL-4.8 (32-bit only)
  Ubuntu 10.04,
  SLED11-SP1

OS Platform Support added to CUDA 3.2

- Additional Platform Support Linux 32 and 64:

  Fedora 13,
  Ubuntu 10.04,
  RHEL-5.5,
  SLED-11SP1,
  ICC (64-bit Linux only?)
- Platforms no longer supported

  Fedora 12,
  Ubuntu 9.10
  RHEL-5.4,
  SLED11

OS Platform Support added to CUDA 3.1

- Additional Platform Support Linux 32 and 64:

  Fedora 12,
  OpenSUSE-11.2,
  Ubuntu 9.10
  RHEL-5.4

- Platforms no longer supported

  Fedora 10,
  OpenSUSE-11.1,
  Ubuntu 9.04

OS Platform Support added to CUDA 3.0

- Linux Distributions 32 and 64:

  RHEL-4.x (4.8),
  RHEL-5.x (5.3),
  SLED-11
  Fedora10,
  Ubuntu 9.04,
  OpenSUSE 11.1 (gcc 3.4, gcc 4)

2.1.3. Supported Mac Platforms

OS Platform Support with CUDA 5.5

OS Platform Support with CUDA 5.0

- Added support for Mac OS X 10.8.x
- Added support for Mac OS X 10.7.4
- Removed support for Mac OS X 10.6.8

OS Platform Support with CUDA 4.2

- Official support for Mac OS X 10.7.3

OS Platform Support with CUDA 4.1

- No changes

OS Platform Support with CUDA 4.0

- New OS Platforms added

  Mac OS X Lion 10.7.x
- Continued OS Platforms

  Mac OS X Snow Leopard 10.6.x
- Platforms no longer supported ?

OS Platform Support added to CUDA 3.2

- Mac OS X Snow Leopard 10.6.4
- Mac OS X Snow Leopard 10.6.5

OS Platform Support added to CUDA 3.1 Beta

- Mac OS X Snow Leopard 10.6.3

  32/64-bit for CUDA Driver API
  32/64-bit for CUDA Runtime API

OS Platform Support added to CUDA 3.0 Release

- Mac OS X Snow Leopard 10.6.x

  32/64-bit for CUDA Driver API
  32-bit for CUDA Runtime API

OS Platform Support added to CUDA 3.0 Beta 1

- Mac OS X Snow Leopard 10.6 (32-bit)

OS Platform Support added to CUDA 2.2

- Mac OS X Leopard 10.5.6+ (32-bit)

  (llvm-)gcc 5.0 Apple

2.2. Installation Instructions

2.2.1. Windows Installation Instructions

CUDA 5.5 Toolkit Installer includes CUDA Toolkit 5.5 and Version R319 Driver (Windows XP, Vista, Win7, or Windows Server 2008 R8), and CUDA Samples.

1. Uninstall any previous versions of the NVIDIA CUDA Toolkit and NVIDIA GPU Computing SDK:

   You can uninstall the NVIDIA CUDA Toolkit through the **Start** menu: **Start menu** > **All Programs** > **NVIDIA Corporation** > **CUDA Toolkit** > **Uninstall CUDA**

   You can uninstall the NVIDIA GPU Computing SDK through the **Start** menu: **Start menu** > **All Programs** > **NVIDIA Corporation** > **NVIDIA GPU Computing SDK** > **Uninstall NVIDIA GPU Computing SDK**
2. Install version Release 5.5 of the NVIDIA CUDA Toolkit by launching:

   ```
   cuda_5.5.xx_[winxp_general|winvista_win7_win8_general|
   winvista_win7_win8_notebook]_[32|64].exe
   ```

   The filename depends on the Windows operating system being used.

This installs the Toolkit, CUDA Samples, and Driver. Each of these components can be installed optionally in the installation GUI when launched for the first time. If you install the Driver via silent install, only the display driver and CUDA driver will be included. If you need the full NVIDIA driver to be installed, please uncheck **Silent Driver Install**. The full NVIDIA driver installation will happen after the Toolkit and CUDA Samples are installed.

3. Build the 32-bit and/or 64-bit **`release`** or **`debug`** configurations of the project examples using the provided:

   `*_vs2008.sln`
   solution files for Microsoft Visual Studio 2008

   `*_vs2010.sln`
   solution files for Microsoft Visual Studio 2010

   You can:

   - Use the solution files located in each of the example directories in:

     ```
     CUDA Samples\v5.5\<category>
     ```

   - Use the global solution files located under:

     ```
     CUDA Samples\v5.5\
                      samples_vs2008.sln
                      samples_vs2010.sln
     ```

   - The **`simpleD3D9`** example and many others including CUDA DirectX samples require that Microsoft DirectX SDK (June 2010 or newer) is installed and that the VC++ directory paths are properly set up (located in **Tools** > **Options...**).
   - Prior to CUDA 5.5, CUDA Sample projects referenced a utility library with header and source files called **`cutil`**. This has been removed with the CUDA Samples in CUDA 5.5, and replaced with header files found in **`CUDA Samples\v5.5\common\inc`**: **`helper_cuda.h`**, **`helper_cuda_gl.h`**, **`helper_cuda_drvapi.h`**, **`helper_functions.h`**, **`helper_image.h`**, **`helper_math.h`**, **`helper_string.h`**, and **`helper_timer.h`**

     These files provide utility functions for CUDA device initialization, CUDA error checking, string parsing, image file loading and saving, and timing functions. The CUDA Samples projects no longer have references and dependencies to **`cutil`**, and will now use these helper functions going forward.

4. Run the examples from the **`release`** or **`debug`** directories located in:

   ```
   CUDA Samples\v5.5\bin\win[32|64]\[release|debug]
   ```

   Notes:

   - The **`release`** and **`debug`** configurations require a CUDA-capable GPU to run properly (see *CUDA-Enabled GPUs* in the *CUDA Programming Guide* for a complete list of CUDA-capable GPUs).

2.2.2. Linux Installation Instructions

The default installation folder **<SAMPLES_INSTALL_PATH>** is **~/NVIDIA_CUDA_Samples**. Also, a read-only copy of the samples can be found in **/usr/local/cuda-5.5/samples**.

- Before installing the combined installer, you must be in a console mode. Exit the GUI of your Linux environment by pressing **Ctrl+Alt+Backspace**.
- For some Linux distributions, you may need to stop GDM via:

```
> sudo /etc/init.d/gdm stop
```

or

```
> /sbin/init 3
```

It is also possible to extract the individual packages for separate installation. Please refer to the *Getting Started Guide for Linux* for more details.

1. Install the CUDA 5.5 Toolkit with one of the following commands:
   - For 32-bit Linux distributions:

```
> sudo sh cuda_5.5.xx_linux_32_[distro].run
```

   - For 64-bit Linux distributions:

```
> sudo sh cuda_5.5.xx_linux_64_[distro].run
```

For optimus configurations, you may need to add **--optimus** to the CUDA Toolkit Installer. If you are instead installing a stand-alone driver on an Optimus system, you must pass **--no-opengl-files** to the installer and decline the **xorg.conf** update at the end of the installation.

You are prompted for the path where you want to put the CUDA Toolkit (**/usr/local/cuda-5.5** is the default) and CUDA Samples (**~/NVIDIA_CUDA-5.5** is the default). CUDA Samples are treated like user development code (it is a collection of CUDA examples). During installation, the prompt is to accept the default or override it with a specified path to which the user has write permissions.

After installation, you can find the location of the files here:

CUDA Toolkit: **/usr/local/cuda-5.5** with a symbolic link **/usr/local/cuda** point to this folder.
CUDA Samples: **$(HOME)/NVIDIA_CUDA-5.5**

In addition, a pristine read-only version of the samples can also be found in **/usr/local/cuda-5.5**

2. Set up environment variables for CUDA Development.
   You may want to add this to your **~/.bash_profile**:
   - Add the following to your system **PATH**:
     ```
     export PATH=/usr/local/cuda-5.5/bin:$PATH
     ```
   - Add the following to your **LD_LIBRARY_PATH** (if running on a 32-bit OS)
     ```
     export LD_LIBRARY_PATH=/usr/local/cuda-5.5/lib:$LD_LIBRARY_PATH
     ```
   - Add the following to your **LD_LIBRARY_PATH** (if running on a 64-bit OS)
     ```
     export LD_LIBRARY_PATH=/usr/local/cuda-5.5/lib64:$LD_LIBRARY_PATH
     ```
3. Build the CUDA Samples projects:
   ```
   cd <SAMPLES_INSTALL_PATH>
   make
   ```

   Adding the following in **make** builds for specific targets:

   make x86_64=1
   for 64-bit targets
   make i386=1
   for 32-bit targets
   make
   for the **release** configuration
   make dbg=1
   for the **debug** configuration

   Prior to CUDA 5.5, CUDA Sample projects referenced a utility library with header and source files called CUTIL. Also many of the **Makefile** projects have been rewritten to be self contained and no longer depend on **common.mk**. CUTIL has been removed with the CUDA Samples in CUDA 5.5, and replaced with helper functions found in **NVIDIA_CUDA-5.5/common/inc**: **helper_cuda.h**, **helper_cuda_gl.h**, **helper_cuda_drvapi.h**, **helper_functions.h**, **helper_image.h**, **helper_math.h**, **helper_string.h**, **helper_timer.h**

   These helper functions handle CUDA device initialization, CUDA error checking, string parsing, image file loading and saving, and timing functions. The CUDA Samples projects no longer have references and dependencies to CUTIL, and now use these helper functions going forward.
4. Run the CUDA examples (32-bit or 64-bit Linux):
   ```
   cd <SAMPLES_INSTALL_PATH>/bin/linux/release
   matrixmul
   ```
   (or any of the other executables in that directory)

2.2.3. Mac OS X Installation Instructions

The default installation folder **<SAMPLES_INSTALL_PATH>** is:
```
/Developer/NVIDIA/CUDA-5.5/samples
```

For Snow Leopard (10.6), Lion (10.7), and Mountain Lion (10.8):

To boot up in 32-bit kernel mode, after Power-On (and hearing the boot up sound), hit keys **3** and **2** at the same time immediately after the startup sound. The OS will startup in a 32-bit kernel mode.

To boot up with a 64-bit kernel, during Power-On, hit keys **6** and **4** at the same time.

Please install the packages in this order.

1. Install the NVIDIA CUDA Toolkit Installer Package (Mac OSX Leopard)
   - Do you have a Quadro 4000 for Mac and/or recently updated to the Mac OSX 10.7.x? If so, please first install the release 256 driver for Mac. You can download the package from here:

     http://www.nvidia.com/object/quadro-macosx-256.01.00f03-driver.html
   - For NVIDIA GeForce GPU or Quadro GPUs, install this package:

     `cuda_5.5.xx_macos.pkg`
2. Install version 5.5 Release of the CUDA 5.5 Toolkit installer by executing the file:

   `cuda_5.5.xx_macos.pkg`

   This package will work Mac OS X running 32/64-bit. CUDA applications built in 32/64-bit (CUDA Driver API) are supported. CUDA applications built as 32/64 bit (CUDA Runtime API) are supported. (10.6 SnowLeopard, 10.7 Lion, and 10.8 Mountain Lion)

   You are now able to pick which packages you wish to install
   - CUDA Driver is installed to **`/Library/Frameworks/CUDA.framework`**
   - CUDA Toolkit is installed to **`/Developer/NVIDIA/CUDA-5.5`** (previous toolkit installations will automatically be moved to **`/Developer/NVIDIA/CUDA-#.#`**)
   - CUDA Samples will be installed to **`/Developer/NVIDIA/CUDA-5.5/samples`**

   After installation, you may want to add the following paths to your environment:

   ```
   > export PATH=/Developer/NVIDIA/CUDA-5.5/bin:$PATH
   > export DYLD_LIBRARY_PATH=/Developer/NVIDIA/CUDA-5.5/lib:$DYLD_LIBRARY_PATH
   ```

   To make these settings permanent, place them in **`~/.bash_profile`**
3. Build the SDK project examples:
   - Go to **`<SAMPLES_INSTALL_PATH>`** (**`cd <SAMPLES_INSTALL_PATH>`**)
   - Build:

     `make x86_64=1`
     for 64-bit targets

     `make i386=1`
     for 32-bit targets

     `make`
     for the **`release`** configuration

     `make dbg=1`
     for the **`debug`** configuration

Prior to CUDA 5.5, CUDA Sample projects referenced a utility library with header and source files called CUTIL. Also many of the **Makefile** projects have been rewritten to be self contained and no longer depend on **common.mk**. CUTIL has been removed with the CUDA Samples in CUDA 5.5, and replaced with helper functions found in **NVIDIA_CUDA-5.5/common/inc**: **helper_cuda.h**, **helper_cuda_gl.h**, **helper_cuda_drvapi.h**, **helper_functions.h**, **helper_image.h**, **helper_math.h**, **helper_string.h**, **helper_timer.h**

These helper functions handle CUDA device initialization, CUDA error checking, string parsing, image file loading and saving, and timing functions. The CUDA Samples projects no longer have references and dependencies to CUTIL, and now use these helper functions going forward.

4. Run the CUDA examples:

```
cd <SAMPLES_INSTALL_PATH>/bin/darwin/[release|debug]
./matrixmul
```

(or any of the other executables in that directory)

2.3. Using CUDA Samples to Create Your Own CUDA Projects

2.3.1. Creating CUDA Projects for Windows

Creating a new CUDA Program using the CUDA Samples infrastructure is easy. We have provided a **template** and **template_runtime** project that you can copy and modify to suit your needs. Just follow these steps:

(**<category>** refers to one of the following folders: **0_Simple**, **1_Utilities**, **2_Graphics**, **3_Imaging**, **4_Finance**, **5_Simulations**, **6_Advanced**, **7_CUDALibraries**.)

1. Copy the content of:

```
CUDA Samples\v5.5\<category>\template
```

or

```
CUDA Samples\v5.5\<category>\template_runtime
```

to a directory of your own:

```
CUDA Samples\v5.5\<category>\myproject
```

2. Edit the filenames of the project to suit your needs.
3. Edit the ***.sln**, ***.vcproj** and source files.

Just search and replace all occurrences of **template** or **template_runtime** with **myproject**.

4. Build the 32-bit and/or 64-bit, release or debug configurations using:

myproject_vs2008.sln

myproject_vs2010.sln

5. Run **myproject.exe** from the **release** or **debug** directories located in:

```
CUDA Samples\v5.5\bin\win[32|64]\[release|debug]
```

6. Now modify the code to perform the computation you require.
   See the *CUDA Programming Guide* for details of programming in CUDA.

2.3.2. Creating CUDA Projects for Linux

The default installation folder **<SAMPLES_INSTALL_PATH>** is **NVIDIA_CUDA_5.5_Samples** and <category> is one of the following: **0_Simple**, **1_Utilities**, **2_Graphics**, **3_Imaging**, **4_Finance**, **5_Simulations**, **6_Advanced**, **7_CUDALibraries**.

Creating a new CUDA Program using the NVIDIA CUDA Samples infrastructure is easy. We have provided a **template** or **template_runtime** project that you can copy and modify to suit your needs. Just follow these steps:

1. Copy the **template** or **template_runtime** project:

```
cd <SAMPLES_INSTALL_PATH>/<category>
cp -r template <myproject>
```

   or (using **template_runtime**):

```
cd <SAMPLES_INSTALL_PATH>/<category>
cp -r template_runtime <myproject>
```

2. Edit the filenames of the project to suit your needs:

```
mv template.cu myproject.cu
mv template_kernel.cu myproject_kernel.cu
mv template_gold.cpp myproject_gold.cpp
```

   or (using **template_runtime**):

```
mv main.cu  myproject.cu
```

3. Edit the **Makefile** and source files.
   Just search and replace all occurrences of **template** or **template_runtime** with **myproject**.
4. Build the project as (release):

```
make
```

   To build the project as (debug), use "make dbg=1":

5. Run the program:

```
../../bin/linux/x86_64/release/myproject
```

6. Now modify the code to perform the computation you require.
   See the *CUDA Programming Guide* for details of programming in CUDA.

2.3.3. Creating CUDA Projects for Mac OS X

The default installation folder **<SAMPLES_INSTALL_PATH>** is: **/Developer/NVIDIA/CUDA-5.5/samples**

Creating a new CUDA Program using the NVIDIA CUDA Samples infrastructure is easy. We have provided a **template** project that you can copy and modify to suit your needs. Just follow these steps:

(<category> is one of the following: **0_Simple**, **1_Utilities**, **2_Graphics**, **3_Imaging**, **4_Finance**, **5_Simulations**, **6_Advanced**, **7_CUDALibraries**.)

1. Copy the template project:
```
cd <SAMPLES_INSTALL_PATH>/<category>
cp -r template <myproject>
```
2. Edit the filenames of the project to suit your needs:
```
mv template.cu myproject.cu
mv template_kernel.cu myproject_kernel.cu
mv template_gold.cpp myproject_gold.cpp
```
3. Edit the **Makefile** and source files.

   Just search and replace all occurrences of **template** with **myproject**.
4. Build the project as (release):
```
make
```
   Note: To build the project as (debug), use "make dbg=1"
5. Run the program:
```
../../bin/darwin/x86_64/release/myproject
```
   (It should print **PASSED**.)
6. Now modify the code to perform the computation you require.

   See the *CUDA Programming Guide* for details of programming in CUDA.

Chapter 3. SAMPLES REFERENCE

This document contains a complete listing of the code samples that are included with the NVIDIA CUDA Toolkit. It describes each code sample, lists the minimum GPU specification, and provides links to the source code and white papers if available.

The code samples are divided into the following categories:

Simple Reference
Basic CUDA samples for beginners that illustrate key concepts with using CUDA and CUDA runtime APIs.

Utilities Reference
Utility samples that demonstrate how to query device capabilities and measure GPU/CPU bandwidth.

Graphics Reference
Graphical samples that demonstrate interoperability between CUDA and OpenGL or DirectX.

Imaging Reference
Samples that demonstrate image processing, compression, and data analysis.

Finance Reference
Samples that demonstrate parallel algorithms for financial computing.

Simulations Reference
Samples that illustrate a number of simulation algorithms implemented with CUDA.

Advanced Reference
Samples that illustrate advanced algorithms implemented with CUDA.

Cudalibraries Reference
Samples that illustrate how to use CUDA platform libraries (NPP, CUBLAS, CUFFT, CUSPARSE, and CURAND).

3.1. Simple Reference

cppOverload

This sample demonstrates how to use C++ function overloading on the GPU.

Minimum Required GPU SM 2.0

CUDA API	cudaFuncSetCacheConfig, cudaFuncGetAttributes
Key Concepts	C++ Function Overloading, CUDA Streams and Events
Source	zip | tar.gz

Simple Quicksort (CUDA Dynamic Parallelism)

This sample demonstrates simple quicksort implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

Minimum Required GPU	KEPLER SM 3.5
Key Concepts	CUDA Dynamic Parallelism
Source	zip | tar.gz

Simple Print (CUDA Dynamic Parallelism)

This sample demonstrates simple printf implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

Minimum Required GPU	KEPLER SM 3.5
Key Concepts	CUDA Dynamic Parallelism
Source	zip | tar.gz

Simple Static GPU Device Library

This sample demonstrates a CUDA 5.0 feature, the ability to create a GPU device static library and use it within another CUDA kernel. This example demonstrates how to pass in a GPU device function (from the GPU device static library) as a function pointer to be called. This sample requires devices with compute capability 2.0 or higher.

Minimum Required GPU	SM 2.0
Key Concepts	Separate Compilation
Source	zip | tar.gz

Simple CUDA Callbacks

This sample implements multi-threaded heterogeneous computing workloads with the new CPU callbacks for CUDA streams and events introduced with CUDA 5.0.

Minimum Required GPU	SM 1.0

CUDA API	cudaStreamCreate, cudaMemcpyAsync, cudaStreamAddCallback, cudaStreamDestroy
Key Concepts	CUDA Streams, Callback Functions, Multithreading
Source	zip | tar.gz

simpleIPC

This CUDA Runtime API sample is a very basic sample that demonstrates Inter Process Communication with one process per GPU for computation. Requires Compute Capability 2.0 or higher and a Linux Operating System

Minimum Required GPU	SM 2.0
CUDA API	cudaIpcGetEventHandlet, cudaIpcOpenMemHandle, cudaIpcCloseMemHandle, cudaFreeHost, cudaMemcpy
Key Concepts	CUDA Systems Integration, Peer to Peer, InterProcess Communication
Source	zip | tar.gz

simpleAssert

This CUDA Runtime API sample is a very basic sample that implements how to use the assert function in the device code. Requires Compute Capability 2.0 .

Minimum Required GPU	SM 2.0
CUDA API	cudaMalloc, cudaMallocHost, cudaFree, cudaFreeHost, cudaMemcpy
Key Concepts	Assert
Source	zip | tar.gz

Simple Cubemap Texture

Simple example that demonstrates how to use a new CUDA 4.1 feature to support cubemap Textures in CUDA C.

Minimum Required GPU	SM 2.0
CUDA API	cudaMalloc, cudaMalloc3DArray, cudaMemcpy3D, cudaCreateChannelDesc, cudaBindTextureToArray, cudaMalloc, cudaFree, cudaFreeArray, cudaMemcpy
Key Concepts	Texture, Volume Processing
Source	zip | tar.gz

Simple Peer-to-Peer Transfers with Multi-GPU

This application demonstrates the new CUDA 4.0 APIs that support Peer-To-Peer (P2P) copies, Peer-To-Peer (P2P) addressing, and UVA (Unified Virtual Memory Addressing) between multiple Tesla GPUs.

Minimum Required GPU	SM 2.0
CUDA API	cudaDeviceCanAccessPeer, cudaDeviceEnablePeerAccess, cudaDeviceDisablePeerAccess, cudaEventCreateWithFlags, cudaEventElapsedTime, cudaMemcpy
Key Concepts	Performance Strategies, Asynchronous Data Transfers, Unified Virtual Address Space, Peer to Peer Data Transfers, Multi-GPU
Source	zip | tar.gz

Using Inline PTX

A simple test application that demonstrates a new CUDA 4.0 ability to embed PTX in a CUDA kernel.

Minimum Required GPU	SM 1.0
CUDA API	cudaMalloc, cudaMallocHost, cudaFree, cudaFreeHost, cudaMemcpy
Key Concepts	Performance Strategies, PTX Assembly, CUDA Driver API
Source	zip | tar.gz

Simple Layered Texture

Simple example that demonstrates how to use a new CUDA 4.0 feature to support layered Textures in CUDA C.

Minimum Required GPU	SM 2.0
CUDA API	cudaMalloc, cudaMalloc3DArray, cudaMemcpy3D, cudaCreateChannelDesc, cudaBindTextureToArray, cudaMalloc, cudaFree, cudaFreeArray, cudaMemcpy
Key Concepts	Texture, Volume Processing
Source	zip | tar.gz

simplePrintf

This CUDA Runtime API sample is a very basic sample that implements how to use the printf function in the device code. Specifically, for devices with compute capability less than 2.0, the function cuPrintf is called; otherwise, printf can be used directly.

Minimum Required GPU	SM 1.0
CUDA API	cudaPrintfDisplay, cudaPrintfEnd
Key Concepts	Debugging
Source	zip | tar.gz

Simple Surface Write

Simple example that demonstrates the use of 2D surface references (Write-to-Texture)

Minimum Required GPU	SM 2.0
CUDA API	cudaMalloc, cudaMallocArray, cudaBindSurfaceToArray, cudaBindTextureToArray, cudaCreateChannelDesc, cudaMalloc, cudaFree, cudaFreeArray, cudaMemcpy
Key Concepts	Texture, Surface Writes, Image Processing
Source	zip | tar.gz

Simple Multi Copy and Compute

Supported in GPUs with Compute Capability 1.1, overlaping compute with one memcopy is possible from the host system. For Quadro and Tesla GPUs with Compute Capability 2.0, a second overlapped copy operation in either direction at full speed is possible (PCI-e is symmetric). This sample illustrates the usage of CUDA streams to achieve overlapping of kernel execution with data copies to and from the device.

Minimum Required GPU	SM 1.1
CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMemcpyAsync
Key Concepts	CUDA Streams and Events, Asynchronous Data Transfers, Overlap Compute and Copy, GPU Performance
Source	zip | tar.gz

Vector Addition

This CUDA Runtime API sample is a very basic sample that implements element by element vector addition. It is the same as the sample illustrating Chapter 3 of the programming guide with some additions like error checking.

Minimum Required GPU	SM 1.0
CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaEventSynchronize, cudaMalloc, cudaFree, cudaMemcpy
Key Concepts	CUDA Runtime API, Vector Addition
Source	zip | tar.gz

Vector Addition Driver API

This Vector Addition sample is a basic sample that is implemented element by element. It is the same as the sample illustrating Chapter 3 of the programming guide with some additions like error checking. This sample also uses the new CUDA 4.0 kernel launch Driver API.

Minimum Required GPU	SM 1.0
CUDA API	cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuMemAlloc, cuMemFree, cuMemcpyHtoD, cuMemcpyDtoH, cuLaunchKernel
Key Concepts	CUDA Driver API, Vector Addition
Source	zip | tar.gz

Template using CUDA Runtime

A trivial template project that can be used as a starting point to create new CUDA Runtime API projects.

Minimum Required GPU	SM 1.0
CUDA API	cudaMalloc, cudaMallocHost, cudaFree, cudaFreeHost, cudaDeviceSynchronize, cudaMemcpy
Key Concepts	CUDA Data Transfers, Device Memory Allocation
Source	zip | tar.gz

Template

A trivial template project that can be used as a starting point to create new CUDA projects.

Minimum Required GPU	SM 1.0
CUDA API	cudaMalloc, cudaFree, cudaDeviceSynchronize, cudaMemcpy
Key Concepts	Device Memory Allocation
Source	zip | tar.gz

C++ Integration

This example demonstrates how to integrate CUDA into an existing C++ application, i.e. the CUDA entry point on host side is only a function which is called from C++ code and only the file containing this function is compiled with nvcc. It also demonstrates that vector types can be used from cpp.

Minimum Required GPU	SM 1.0
CUDA API	cudaMalloc, cudaFree, cudaMemcpy
Source	zip | tar.gz

asyncAPI

This sample uses CUDA streams and events to overlap execution on CPU and GPU.

Minimum Required GPU	SM 1.1
CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMemcpyAsync
Key Concepts	Asynchronous Data Transfers, CUDA Streams and Events
Source	zip | tar.gz

Clock

This example shows how to use the clock function to measure the performance of kernel accurately.

Minimum Required GPU	SM 1.0
CUDA API	cudaMalloc, cudaFree, cudaMemcpy
Key Concepts	Performance Strategies

Source	zip | tar.gz

Simple Atomic Intrinsics

A simple demonstration of global memory atomic instructions. Requires Compute Capability 1.1 or higher.

Minimum Required GPU	SM 1.1
CUDA API	cudaMallco, cudaFree, cudaMemcpy, cudaFreeHost
Key Concepts	Atomic Intrinsics
Source	zip | tar.gz

Pitch Linear Texture

Use of Pitch Linear Textures

Minimum Required GPU	SM 1.0
CUDA API	cudaMallocPitch, cudaMallocArray, cudaMemcpy2D, cudaMemcpyToArray, cudaBindTexture2D, cudaBindTextureToArray, cudaCreateChannelDesc, cudaMalloc, cudaFree, cudaFreeArray, cudaUnbindTexture, cudaMemset2D, cudaMemcpy2D
Key Concepts	Texture, Image Processing
Source	zip | tar.gz

simpleStreams

This sample uses CUDA streams to overlap kernel executions with memory copies between the host and a GPU device. This sample uses a new CUDA 4.0 feature that supports pinning of generic host memory. Requires Compute Capability 1.1 or higher.

Minimum Required GPU	SM 1.1
CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMemcpyAsync
Key Concepts	Asynchronous Data Transfers, CUDA Streams and Events
Source	zip | tar.gz

Simple Templates

This sample is a templatized version of the template project. It also shows how to correctly templatize dynamically allocated shared memory arrays.

Minimum Required GPU	SM 1.0
Key Concepts	C++ Templates
Source	zip | tar.gz

Simple Texture

Simple example that demonstrates use of Textures in CUDA.

Minimum Required GPU	SM 1.0
CUDA API	cudaMalloc, cudaMallocArray, cudaMemcpyToArray, cudaCreateChannelDesc, cudaBindTextureToArray, cudaMalloc, cudaFree, cudaFreeArray, cudaMemcpy
Key Concepts	CUDA Runtime API, Texture, Image Processing
Source	zip | tar.gz

Simple Texture (Driver Version)

Simple example that demonstrates use of Textures in CUDA. This sample uses the new CUDA 4.0 kernel launch Driver API.

Minimum Required GPU	SM 1.0
CUDA API	cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuLaunchKernel, cuCtxSynchronize, cuMemcpyDtoH, cuMemAlloc, cuMemFree, cuArrayCreate, cuArrayDestroy, cuCtxDetach, cuMemcpy2D, cuModuleGetTexRef, cuTexRefSetArray, cuTexRefSetAddressMode, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuParamSetTexRef
Key Concepts	CUDA Driver API, Texture, Image Processing
Source	zip | tar.gz

Simple Vote Intrinsics

Simple program which demonstrates how to use the Vote (any, all) intrinsic instruction in a CUDA kernel. Requires Compute Capability 1.2 or higher.

Minimum Required GPU	SM 1.2

CUDA API	cudaMallco, cudaFree, cudaMemcpy, cudaFreeHost
Key Concepts	Vote Intrinsics
Source	zip | tar.gz

simpleZeroCopy

This sample illustrates how to use Zero MemCopy, kernels can read and write directly to pinned system memory. This sample requires GPUs that support this feature (MCP79 and GT200).

Minimum Required GPU	SM 1.2
CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaHostAlloc, cudaHostGetDevicePointer, cudaHostRegister, cudaHostUnregister, cudaFreeHost
Key Concepts	Performance Strategies, Pinned System Paged Memory, Vector Addition
Source	zip | tar.gz
Whitepaper	CUDA2.2PinnedMemoryAPIs.pdf

Simple Multi-GPU

This application demonstrates how to use the new CUDA 4.0 API for CUDA context management and multi-threaded access to run CUDA kernels on multiple-GPUs.

Minimum Required GPU	SM 1.0
CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMemcpyAsync
Key Concepts	Asynchronous Data Transfers, CUDA Streams and Events, Multithreading, Multi-GPU
Source	zip | tar.gz

Matrix Multiplication (CUBLAS)

This sample implements matrix multiplication from Chapter 3 of the programming guide. To illustrate GPU performance for matrix multiply, this sample also shows how to use the new CUDA 4.0 interface for CUBLAS to demonstrate high-performance performance for matrix multiplication.

Minimum Required GPU	SM 1.0

CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMalloc, cudaFree, cudaMemcpy, cublasCreate, cublasSgemm
Key Concepts	CUDA Runtime API, Performance Strategies, Linear Algebra, CUBLAS
Source	zip | tar.gz

Matrix Multiplication (CUDA Runtime API Version)

This sample implements matrix multiplication and is exactly the same as Chapter 6 of the programming guide. It has been written for clarity of exposition to illustrate various CUDA programming principles, not with the goal of providing the most performant generic kernel for matrix multiplication. To illustrate GPU performance for matrix multiply, this sample also shows how to use the new CUDA 4.0 interface for CUBLAS to demonstrate high-performance performance for matrix multiplication.

Minimum Required GPU	SM 1.0
CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaEventSynchronize, cudaMalloc, cudaFree, cudaMemcpy
Key Concepts	CUDA Runtime API, Linear Algebra
Source	zip | tar.gz

Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version)

This sample revisits matrix multiplication using the CUDA driver API. It demonstrates how to link to CUDA driver at runtime and how to use JIT (just-in-time) compilation from PTX code. It has been written for clarity of exposition to illustrate various CUDA programming principles, not with the goal of providing the most performant generic kernel for matrix multiplication. CUBLAS provides high-performance matrix multiplication.

Minimum Required GPU	SM 1.0
CUDA API	cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuMemAlloc, cuMemFree, cuMemcpyHtoD, cuMemcpyDtoH, cuLaunchKernel
Key Concepts	CUDA Driver API, CUDA Dynamically Linked Library
Source	zip | tar.gz

Matrix Multiplication (CUDA Driver API Version)

This sample implements matrix multiplication and uses the new CUDA 4.0 kernel launch Driver API. It has been written for clarity of exposition to illustrate various CUDA programming principles, not with the goal of providing the most performant generic kernel for matrix multiplication. CUBLAS provides high-performance matrix multiplication.

Minimum Required GPU	SM 1.0
CUDA API	cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuMemAlloc, cuMemFree, cuMemcpyHtoD, cuMemcpyDtoH, cuLaunchKernel
Key Concepts	CUDA Driver API, Matrix Multiply
Source	zip | tar.gz

simpleMPI

Simple example demonstrating how to use MPI in combination with CUDA. This executable is not pre-built with the SDK installer.

Minimum Required GPU	SM 1.0
CUDA API	cudaMallco, cudaFree, cudaMemcpy
Key Concepts	CUDA Systems Integration, MPI, Multithreading
Source	zip | tar.gz

cudaOpenMP

This sample demonstrates how to use OpenMP API to write an application for multiple GPUs. This executable is not pre-built with the SDK installer.

Minimum Required GPU	SM 1.0
CUDA API	cudaMalloc, cudaFree, cudaMemcpy
Key Concepts	CUDA Systems Integration, OpenMP, Multithreading
Source	zip | tar.gz

3.2. Utilities Reference

Device Query

This sample enumerates the properties of the CUDA devices present in the system.

Minimum Required GPU	SM 1.0
CUDA API	cudaSetDevice, cudaGetDeviceCount, cudaGetDeviceProperties, cudaDriverGetVersion, cudaRuntimeGetVersion
Key Concepts	CUDA Runtime API, Device Query
Source	zip | tar.gz

Device Query Driver API

This sample enumerates the properties of the CUDA devices present using CUDA Driver API calls

Minimum Required GPU	SM 1.0
CUDA API	cuInit, cuDeviceGetCount, cuDeviceComputeCapability, cuDriverGetVersion, cuDeviceTotalMem, cuDeviceGetAttribute
Key Concepts	CUDA Driver API, Device Query
Source	zip | tar.gz

Bandwidth Test

This is a simple test program to measure the memcopy bandwidth of the GPU and memcpy bandwidth across PCI-e. This test application is capable of measuring device to device copy bandwidth, host to device copy bandwidth for pageable and page-locked memory, and device to host copy bandwidth for pageable and page-locked memory.

Minimum Required GPU	SM 1.0
CUDA API	cudaSetDevice, cudaHostAlloc, cudaFree, cudaMallocHost, cudaFreeHost, cudaMemcpy, cudaMemcpyAsync, cudaEventCreate, cudaEventRecord, cudaEventDestroy, cudaDeviceSynchronize, cudaEventElapsedTime
Key Concepts	CUDA Streams and Events, Performance Strategies
Source	zip | tar.gz

3.3. Graphics Reference

Bindless Texture

This example demonstrates use of cudaSurfaceObject, cudaTextureObject, and MipMap support in CUDA. A GPU with Compute Capability SM 3.0 is required to run the sample.

Minimum Required GPU	KEPLER SM 3.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Texture
Source	zip | tar.gz

Volumetric Filtering with 3D Textures and Surface Writes

This sample demonstrates 3D Volumetric Filtering using 3D Textures and 3D Surface Writes.

Minimum Required GPU	SM 2.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing, 3D Textures, Surface Writes
Source	zip | tar.gz

SLI D3D10 Texture

Simple program which demonstrates SLI with Direct3D10 Texture interoperability with CUDA. The program creates a D3D10 Texture which is written to from a CUDA kernel. Direct3D then renders the results on the screen. A Direct3D Capable device is required.

Minimum Required GPU	SM 1.0
CUDA API	cudaD3D10GetDevice, cudaD3D10SetDirect3DDevice, cudaGraphicsD3D10RegisterResource, cudaGraphicsResourceSetMapFlags,

	cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray, cudaGraphicsUnregisterResource
Key Concepts	Performance Strategies, Graphics Interop, Image Processing, 2D Textures
Source	zip | tar.gz

Simple D3D11 Texture

Simple program which demonstrates Direct3D11 Texture interoperability with CUDA. The program creates a number of D3D11 Textures (2D, 3D, and CubeMap) which are written to from CUDA kernels. Direct3D then renders the results on the screen. A Direct3D Capable device is required.

Minimum Required GPU	SM 1.0
CUDA API	cudaD3D11GetDevice, cudaD3D11SetDirect3DDevice, cudaGraphicsD3D11RegisterResource, cudaGraphicsResourceSetMapFlags, cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing
Source	zip | tar.gz

Simple Direct3D9 (Vertex Arrays)

Simple program which demonstrates interoperability between CUDA and Direct3D9. The program generates a vertex array with CUDA and uses Direct3D9 to render the geometry. A Direct3D capable device is required.

Minimum Required GPU	SM 1.0
CUDA API	cudaD3D9GetDevice, cudaD3D9SetDirect3DDevice, cudaGraphicsD3D9RegisterResource, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop
Source	zip | tar.gz

Simple D3D9 Texture

Simple program which demonstrates Direct3D9 Texture interoperability with CUDA. The program creates a number of D3D9 Textures (2D, 3D, and CubeMap) which are written to from CUDA kernels. Direct3D then renders the results on the screen. A Direct3D capable device is required.

Minimum Required GPU	SM 1.0

CUDA API	cudaD3D9GetDevice, cudaD3D9SetDirect3DDevice, cudaGraphicsD3D9RegisterResource, cudaGraphicsResourceSetMapFlags, cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray, cudaMemcpy3D, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Texture
Source	zip | tar.gz

Simple Direct3D10 (Vertex Array)

Simple program which demonstrates interoperability between CUDA and Direct3D10. The program generates a vertex array with CUDA and uses Direct3D10 to render the geometry. A Direct3D Capable device is required.

Minimum Required GPU	SM 1.0
CUDA API	cudaD3D10GetDevice, cudaD3D10SetDirect3DDevice, cudaGraphicsD3D10RegisterResource, cudaGraphicsResourceSetMapFlags, cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, 3D Graphics
Source	zip | tar.gz

Simple Direct3D10 Render Target

Simple program which demonstrates interoperability between CUDA and Direct3D10. The program takes RenderTarget positions with CUDA and and generates a histogram with visualization. A Direct3D Capable device is required.

Minimum Required GPU	SM 1.0
CUDA API	cudaD3D10GetDevice, cudaD3D10SetDirect3DDevice, cudaGraphicsD3D10RegisterResource, cudaGraphicsResourceSetMapFlags, cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Texture
Source	zip | tar.gz

Simple D3D10 Texture

Simple program which demonstrates how to interoperate CUDA with Direct3D10 Texture. The program creates a number of D3D10 Textures (2D, 3D, and CubeMap)

which are generated from CUDA kernels. Direct3D then renders the results on the screen. A Direct3D10 Capable device is required.

Minimum Required GPU	SM 1.0
CUDA API	cudaD3D10GetDevice, cudaD3D10SetDirect3DDevice, cudaGraphicsD3D10RegisterResource, cudaGraphicsResourceSetMapFlags, cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Texture
Source	zip | tar.gz

Simple OpenGL

Simple program which demonstrates interoperability between CUDA and OpenGL. The program modifies vertex positions with CUDA and uses OpenGL to render the geometry.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Vertex Buffers, 3D Graphics
Source	zip | tar.gz

Simple Texture 3D

Simple example that demonstrates use of 3D Textures in CUDA.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing, 3D Textures, Surface Writes
Source	zip | tar.gz

Mandelbrot

This sample uses CUDA to compute and display the Mandelbrot or Julia sets interactively. It also illustrates the use of "double single" arithmetic to improve precision when zooming a long way into the pattern. This sample use double precision hardware if a GT200 class GPU is present. Thanks to Mark Granger of NewTek who submitted this sample to the SDK!

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Data Parallel Algorithms
Source	zip | tar.gz

Marching Cubes Isosurfaces

This sample extracts a geometric isosurface from a volume dataset using the marching cubes algorithm. It uses the scan (prefix sum) function from the Thrust library to perform stream compaction.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	OpenGL Graphics Interop, Vertex Buffers, 3D Graphics, Physically Based Simulation
Source	zip | tar.gz

Volume Rendering with 3D Textures

This sample demonstrates basic volume rendering using 3D Textures.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource

Key Concepts	Graphics Interop, Image Processing, 3D Textures
Source	zip | tar.gz

3.4. Imaging Reference

Stereo Disparity Computation (SAD SIMD Intrinsics)

A CUDA program that demonstrates how to compute a stereo disparity map using SIMD SAD (Sum of Absolute Difference) intrinsics. Requires Compute Capability 2.0 or higher.

Minimum Required GPU	SM 2.0
Key Concepts	Image Processing, Video Intrinsics
Source	zip | tar.gz

Optical Flow

Variational optical flow estimation example. Uses textures for image operations. Shows how simple PDE solver can be accelerated with CUDA.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, Data Parallel Algorithms
Source	zip | tar.gz
Whitepaper	OpticalFlow.pdf

CUDA Video Encode (C Library) API

This sample demonstrates how to effectively use the CUDA Video Encoder API encode H.264 video. Video input in YUV formats are taken as input (either CPU system or GPU memory) and video output frames are encoded to an H.264 file

Minimum Required GPU	SM 1.0
CUDA API	CreateHWEncInterfaceInstance, CreateHWEncoder, GetHWEncodeCaps, IsSupportedCodec, IsSupportedCodecProfile, IsSupportedParam, EncodeFrameUT, RegisterCB, GetSPSPPS, SetCodecType, GetCodecType, SetParamValue, GetParamValue, SetDefaultParam, DestroyEncoder, SetParamValue, GetParamValue, cuvidCtxLock, cuvidCtxUnlock
Key Concepts	Graphics Interop, Video Compression

Source	zip | tar.gz
Whitepaper	nvcuvenc.pdf

Bilateral Filter

Bilateral filter is an edge-preserving non-linear smoothing filter that is implemented with CUDA with OpenGL rendering. It can be used in image recovery and denoising. Each pixel is weight by considering both the spatial distance and color distance between its neibors. Reference:"C. Tomasi, R. Manduchi, Bilateral Filtering for Gray and Color Images, proceeding of the ICCV, 1998, http://users.soe.ucsc.edu/~manduchi/Papers/ICCV98.pdf"

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing
Source	zip | tar.gz

DCT8x8

This sample demonstrates how Discrete Cosine Transform (DCT) for blocks of 8 by 8 pixels can be performed using CUDA: a naive implementation by definition and a more traditional approach used in many libraries. As opposed to implementing DCT in a fragment shader, CUDA allows for an easier and more efficient implementation.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, Video Compression
Source	zip | tar.gz
Whitepaper	dct8x8.pdf

1D Discrete Haar Wavelet Decomposition

Discrete Haar wavelet decomposition for 1D signals with a length which is a power of 2.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, Video Compression
Source	zip | tar.gz

CUDA Histogram

This sample demonstrates efficient implementation of 64-bin and 256-bin histogram.

Minimum Required GPU	SM 1.1
Key Concepts	Image Processing, Data Parallel Algorithms
Source	zip | tar.gz
Whitepaper	histogram.pdf

Box Filter

Fast image box filter using CUDA with OpenGL rendering.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing
Source	zip | tar.gz

CUDA and OpenGL Interop of Images

This sample shows how to copy CUDA image back to OpenGL using the most efficient methods.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing, Performance Strategies
Source	zip | tar.gz

Post-Process in OpenGL

This sample shows how to post-process an image rendered in OpenGL using CUDA.

Minimum Required GPU	SM 1.0

CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing
Source	zip | tar.gz

DirectX Texture Compressor (DXTC)

High Quality DXT Compression using CUDA. This example shows how to implement an existing computationally-intensive CPU compression algorithm in parallel on the GPU, and obtain an order of magnitude performance improvement.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, Image Compression
Source	zip | tar.gz
Whitepaper	cuda_dxtc.pdf

Image denoising

This sample demonstrates two adaptive image denoising technqiues: KNN and NLM, based on computation of both geometric and color distance between texels. While both techniques are implemented in the DirectX SDK using shaders, massively speeded up variation of the latter techique, taking advantage of shared memory, is implemented in addition to DirectX counterparts.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing
Source	zip | tar.gz
Whitepaper	imageDenoising.pdf

Sobel Filter

This sample implements the Sobel edge detection filter for 8-bit monochrome images.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,

	cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing
Source	zip | tar.gz

Recursive Gaussian Filter

This sample implements a Gaussian blur using Deriche's recursive method. The advantage of this method is that the execution time is independent of the filter width.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Image Processing
Source	zip | tar.gz

CUDA Video Decoder D3D9 API

This sample demonstrates how to efficiently use the CUDA Video Decoder API to decode MPEG-2, VC-1, or H.264 sources. YUV to RGB conversion of video is accomplished with CUDA kernel. The output result is rendered to a D3D9 surface. The decoded video is not displayed on the screen, but with -displayvideo at the command line parameter, the video output can be seen. Requires a Direct3D capable device and Compute Capability 1.1 or higher.

Minimum Required GPU	SM 1.1
CUDA API	cuDeviceGet, cuDeviceGetAttribute, cuDeviceComputeCapability, cuDeviceGetCount, cuDeviceGetName, cuDeviceTotalMem, cuD3D9CtxCreate, cuD3D9GetDevice, cuModuleLoad, cuModuleUnload, cuModuleGetFunction, cuModuleGetGlobal, cuModuleLoadDataEx, cuModuleGetTexRef, cuD3D9MapResources, cuD3D9UnmapResources, cuD3D9RegisterResource, cuD3D9UnregisterResource, cuD3D9ResourceSetMapFlags, cuD3D9ResourceGetMappedPointer, cuD3D9ResourceGetMappedPitch, cuParamSetv, cuParamSeti, cuParamSetSize, cuLaunchGridAsync, cuCtxCreate, cuMemAlloc, cuMemFree, cuMemAllocHost, cuMemFreeHost, cuMemcpyDtoHAsync, cuMemsetD8, cuStreamCreate, cuCtxPushCurrent, cuCtxPopCurrent, cuvidCreateDecoder, cuvidDecodePicture, cuvidMapVideoFrame, cuvidUnmapVideoFrame, cuvidDestroyDecoder, cuvidCtxLockCreate, cuvidCtxLockDestroy, cuCtxDestroy

Key Concepts	Graphics Interop, Image Processing, Video Compression
Source	zip | tar.gz
Whitepaper	nvcuvid.pdf

CUDA Video Decoder GL API

This sample demonstrates how to efficiently use the CUDA Video Decoder API to decode video sources based on MPEG-2, VC-1, and H.264. YUV to RGB conversion of video is accomplished with CUDA kernel. The output result is rendered to a OpenGL surface. The decoded video is black, but can be enabled with -displayvideo added to the command line. Requires Compute Capability 1.1 or higher.

Minimum Required GPU	SM 1.1
CUDA API	cuDeviceGet, cuDeviceGetAttribute, cuDeviceComputeCapability, cuDeviceGetCount, cuDeviceGetName, cuDeviceTotalMem, cuGLCtxCreate, cuGLGetDevice, cuModuleLoad, cuModuleUnload, cuModuleGetFunction, cuModuleGetGlobal, cuModuleLoadDataEx, cuModuleGetTexRef, cuGLMapResources, cuGLUnmapResources, cuGLRegisterResource, cuGLUnregisterResource, cuGLResourceSetMapFlags, cuGLResourceGetMappedPointer, cuGLResourceGetMappedPitch, cuParamSetv, cuParamSeti, cuParamSetSize, cuLaunchGridAsync, cuCtxCreate, cuMemAlloc, cuMemFree, cuMemAllocHost, cuMemFreeHost, cuMemcpyDtoHAsync, cuMemsetD8, cuStreamCreate, cuCtxPushCurrent, cuCtxPopCurrent, cuvidCreateDecoder, cuvidDecodePicture, cuvidMapVideoFrame, cuvidUnmapVideoFrame, cuvidDestroyDecoder, cuvidCtxLockCreate, cuvidCtxLockDestroy, cuCtxDestroy
Key Concepts	Graphics Interop, Image Processing, Video Compression
Source	zip | tar.gz
Whitepaper	nvcuvid.pdf

Bicubic B-spline Interoplation

This sample demonstrates how to efficiently implement a Bicubic B-spline interpolation filter with CUDA texture.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource

Key Concepts	Graphics Interop, Image Processing
Source	zip | tar.gz

FFT-Based 2D Convolution

This sample demonstrates how 2D convolutions with very large kernel sizes can be efficiently implemented using FFT transformations.

Minimum Required GPU	SM 1.0
CUDA API	cufftPlan2d, cufftExecR2C, cufftExecC2R, cufftDestroy
Key Concepts	Image Processing, CUFFT Library
Source	zip | tar.gz

CUDA Separable Convolution

This sample implements a separable convolution filter of a 2D signal with a gaussian kernel.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, Data Parallel Algorithms
Source	zip | tar.gz
Whitepaper	convolutionSeparable.pdf

Texture-based Separable Convolution

Texture-based implementation of a separable 2D convolution with a gaussian kernel. Used for performance comparison against convolutionSeparable.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, Texture, Data Parallel Algorithms
Source	zip | tar.gz

3.5. Finance Reference

Binomial Option Pricing

This sample evaluates fair call price for a given set of European options under binomial model. This sample will also take advantage of double precision if a GTX 200 class GPU is present.

Minimum Required GPU	SM 1.0
Key Concepts	Computational Finance
Source	zip | tar.gz
Whitepaper	binomialOptions.pdf

Black-Scholes Option Pricing

This sample evaluates fair call and put prices for a given set of European options by Black-Scholes formula.

Minimum Required GPU	SM 1.0
Key Concepts	Computational Finance
Source	zip | tar.gz
Whitepaper	BlackScholes.pdf

Niederreiter Quasirandom Sequence Generator

This sample implements Niederreiter Quasirandom Sequence Generator and Inverse Cumulative Normal Distribution function for Standart Normal Distribution generation.

Minimum Required GPU	SM 1.0
Key Concepts	Computational Finance
Source	zip | tar.gz

Monte Carlo Option Pricing with Multi-GPU support

This sample evaluates fair call price for a given set of European options using the Monte Carlo approach, taking advantage of all CUDA-capable GPUs installed in the system. This sample use double precision hardware if a GTX 200 class GPU is present. The

sample also takes advantage of CUDA 4.0 capability to supporting using a single CPU thread to control multiple GPUs

Minimum Required GPU	SM 1.0
Source	zip | tar.gz
Whitepaper	MonteCarlo.pdf

Sobol Quasirandom Number Generator

This sample implements Sobol Quasirandom Sequence Generator.

Minimum Required GPU	SM 1.0
Key Concepts	Computational Finance
Source	zip | tar.gz

Excel 2010 CUDA Integration Example

This sample demonstrates how to integrate Excel 2010 with CUDA using array formulas. This plug-in depends on the Microsoft Excel 2010 Developer Kit, which can be downloaded from the Microsoft Developer website. This sample is not pre-built with the CUDA SDK.

Minimum Required GPU	SM 1.0
Source	zip | tar.gz

Excel 2007 CUDA Integration Example

This sample demonstrates how to integrate Excel 2007 with CUDA using array formulas. This plug-in depends on the Microsoft Excel Developer Kit. This sample is not pre-built with the CUDA SDK.

Minimum Required GPU	SM 1.0
Source	zip | tar.gz

3.6. Simulations Reference

VFlockingD3D10

This sample demonstrates a CUDA mathematical simulation of group of birds behavior when in flight.

Minimum Required GPU	SM 1.0
CUDA API	cudaD3D10SetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Data Parallel Algorithms, Physically-Based Simulation, Performance Strategies
Source	zip | tar.gz

Fluids (Direct3D Version)

An example of fluid simulation using CUDA and CUFFT, with Direct3D 9 rendering. A Direct3D Capable device is required.

Minimum Required GPU	SM 1.0
CUDA API	cudaD3D9SetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, CUFFT Librarys, Physically-Based Simulation
Source	zip | tar.gz

Fluids (OpenGL Version)

An example of fluid simulation using CUDA and CUFFT, with OpenGL rendering.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, CUFFT Librarys, Physically-Based Simulation
Source	zip | tar.gz
Whitepaper	fluidsGL.pdf

CUDA FFT Ocean Simulation

This sample simulates an Ocean heightfield using CUFFT Library and renders the result using OpenGL.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource, cufftPlan2d, cufftExecR2C, cufftExecC2R, cufftDestroy
Key Concepts	Graphics Interop, Image Processing, CUFFT Library
Source	zip | tar.gz

Particles

This sample uses CUDA to simulate and visualize a large set of particles and their physical interaction. Adding "-particles=<N>" to the command line will allow users to set # of particles for simulation. This example implements a uniform grid data structure using either atomic operations or a fast radix sort from the Thrust library

Minimum Required GPU	SM 1.1
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Data Parallel Algorithms, Physically-Based Simulation, Performance Strategies
Source	zip | tar.gz
Whitepaper	particles.pdf

CUDA N-Body Simulation

This sample demonstrates efficient all-pairs simulation of a gravitational n-body simulation in CUDA. This sample accompanies the GPU Gems 3 chapter "Fast N-Body Simulation with CUDA". With CUDA 5.5, performance on Tesla K20c has increased to over 1.8TFLOP/s single precision. Double Performance has also improved on all Kepler and Fermi GPU architectures as well. Starting in CUDA 4.0, the nBody sample has been updated to take advantage of new features to easily scale the n-body simulation across multiple GPUs in a single PC. Adding "-numbodies=<bodies>" to the command line will allow users to set # of bodies for simulation. Adding “-numdevices=<N>” to the command line option will cause the sample to use N devices (if available) for simulation. In this mode, the position and velocity data for all bodies are read from system memory using “zero copy” rather than from device memory. For a small number of devices (4 or

fewer) and a large enough number of bodies, bandwidth is not a bottleneck so we can achieve strong scaling across these devices.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Data Parallel Algorithms, Physically-Based Simulation
Source	zip | tar.gz
Whitepaper	nbody_gems3_ch31.pdf

Smoke Particles

Smoke simulation with volumetric shadows using half-angle slicing technique. Uses CUDA for procedural simulation, Thrust Library for sorting algorithms, and OpenGL for graphics rendering.

Minimum Required GPU	SM 1.0
CUDA API	cudaGLSetGLDevice, cudaGraphicsMapResources, cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource
Key Concepts	Graphics Interop, Data Parallel Algorithms, Physically-Based Simulation
Source	zip | tar.gz
Whitepaper	smokeParticles.pdf

3.7. Advanced Reference

Quad Tree (CUDA Dynamic Parallelism)

This sample demonstrates Quad Trees implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

Minimum Required GPU	KEPLER SM 3.5
Key Concepts	CUDA Dynamic Parallelism
Source	zip | tar.gz

LU Decomposition (CUDA Dynamic Parallelism)

This sample demonstrates LU Decomposition implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

Minimum Required GPU	KEPLER SM 3.5
Key Concepts	CUDA Dynamic Parallelism
Source	zip | tar.gz

Advanced Quicksort (CUDA Dynamic Parallelism)

This sample demonstrates an advanced quicksort implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

Minimum Required GPU	KEPLER SM 3.5
Key Concepts	CUDA Dynamic Parallelism
Source	zip | tar.gz

simpleHyperQ

This sample demonstrates the use of CUDA streams for concurrent execution of several kernels on devices which provide HyperQ (SM 3.5). Devices without HyperQ (SM 2.0 and SM 3.0) will run a maximum of two kernels concurrently.

Minimum Required GPU	SM 1.3
Key Concepts	CUDA Systems Integration, Performance Strategies
Source	zip | tar.gz
Whitepaper	HyperQ.pdf

CUDA Parallel Prefix Sum with Shuffle Intrinsics (SHFL_Scan)

This example demonstrates how to use the shuffle intrinsic __shfl_up to perform a scan operation across a thread block. A GPU with Compute Capability SM 3.0. is required to run the sample

Minimum Required GPU	KEPLER SM 3.0
Key Concepts	Data-Parallel Algorithms, Performance Strategies
Source	zip | tar.gz

CUDA Segmentation Tree Thrust Library

This sample demonstrates an approach to the image segmentation trees construction. This method is based on Boruvka's MST algorithm.

Minimum Required GPU	SM 1.3
Key Concepts	Data-Parallel Algorithms, Performance Strategies
Source	zip | tar.gz

NewDelete

This sample demonstrates dynamic global memory allocation through device C++ new and delete operators and virtual function declarations available with CUDA 4.0.

Minimum Required GPU	SM 2.0
Source	zip | tar.gz

Function Pointers

This sample illustrates how to use function pointers and implements the Sobel Edge Detection filter for 8-bit monochrome images.

Minimum Required GPU	SM 2.0
Key Concepts	Graphics Interop, Image Processing
Source	zip | tar.gz

Interval Computing

Interval arithmetic operators example. Uses various C++ features (templates and recursion). The recursive mode requires Compute SM 2.0 capabilities.

Minimum Required GPU	SM 1.3
Key Concepts	Recursion, Templates
Source	zip | tar.gz

CUDA C 3D FDTD

This sample applies a finite differences time domain progression stencil on a 3D surface.

Minimum Required GPU	SM 1.0

Key Concepts	Performance Strategies
Source	zip | tar.gz

CUDA Context Thread Management

Simple program illustrating how to the CUDA Context Management API and uses the new CUDA 4.0parameter passing and CUDA launch API. CUDA contexts can be created separately and attached independently to different threads.

Minimum Required GPU	SM 1.0
CUDA API	cuCtxCreate, cuCtxDestroy, cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuLaunchKernel, cuMemcpyDtoH, cuCtxPushCurrent, cuCtxPopCurrent
Key Concepts	CUDA Driver API
Source	zip | tar.gz

Scalar Product

This sample calculates scalar products of a given set of input vector pairs.

Minimum Required GPU	SM 1.0
Key Concepts	Linear Algebra
Source	zip | tar.gz

Concurrent Kernels

This sample demonstrates the use of CUDA streams for concurrent execution of several kernels on devices of compute capability 2.0 or higher. Devices of compute capability 1.x will run the kernels sequentially.It also illustrates how to introduce dependencies between CUDA streams with the new cudaStreamWaitEvent function introduced in CUDA 3.2

Minimum Required GPU	SM 1.0
Key Concepts	Performance Strategies
Source	zip | tar.gz

Aligned Types

A simple test, showing huge access speed gap between aligned and misaligned structures.

Minimum Required GPU	SM 1.0
Key Concepts	Performance Strategies
Source	zip | tar.gz

PTX Just-in-Time compilation

This sample uses the Driver API to just-in-time compile (JIT) a Kernel from PTX code. Additionally, this sample demonstrates the seamless interoperability capability of CUDA runtime Runtime and CUDA Driver API calls.

Minimum Required GPU	SM 1.0
Key Concepts	CUDA Driver API
Source	zip | tar.gz

Eigenvalues

The computation of all or a subset of all eigenvalues is an important problem in Linear Algebra, statistics, physics, and many other fields. This sample demonstrates a parallel implementation of a bisection algorithm for the computation of all eigenvalues of a tridiagonal symmetric matrix of arbitrary size with CUDA.

Minimum Required GPU	SM 1.0
Key Concepts	Linear Algebra
Source	zip | tar.gz
Whitepaper	eigenvalues.pdf

Fast Walsh Transform

Naturally(Hadamard)-ordered Fast Walsh Tranform for batched vectors of arbitrary eligible(power of two) lengths

Minimum Required GPU	SM 1.0
Key Concepts	Linear Algebra, Data-Parallel Algorithms, Video Compression
Source	zip | tar.gz

Line of Sight

This sample is an implementation of a simple line-of-sight algorithm: Given a height map and a ray originating at some observation point, it computes all the points along the ray that are visible from the observation point. The implementation is based on the Thrust library (http://code.google.com/p/thrust/).

Minimum Required GPU	SM 1.0
Source	zip | tar.gz

Matrix Transpose

This sample demonstrates Matrix Transpose. Different performance are shown to achieve high performance.

Minimum Required GPU	SM 1.0
Key Concepts	Performance Strategies, Linear Algebra
Source	zip | tar.gz
Whitepaper	MatrixTranspose.pdf

CUDA Parallel Reduction

A parallel sum reduction that computes the sum of a large arrays of values. This sample demonstrates several important optimization strategies for 1:Data-Parallel Algorithms like reduction.

Minimum Required GPU	SM 1.0
Key Concepts	Data-Parallel Algorithms, Performance Strategies
Source	zip | tar.gz
Whitepaper	reduction.pdf

CUDA Parallel Prefix Sum (Scan)

This example demonstrates an efficient CUDA implementation of parallel prefix sum, also known as "scan". Given an array of numbers, scan computes a new array in which each element is the sum of all the elements before it in the input array.

Minimum Required GPU	SM 1.0
Key Concepts	Data-Parallel Algorithms, Performance Strategies

Source	zip | tar.gz

threadFenceReduction

This sample shows how to perform a reduction operation on an array of values using the thread Fence intrinsic. to produce a single value in a single kernel (as opposed to two or more kernel calls as shown in the "reduction" SDK sample). Single-pass reduction requires global atomic instructions (Compute Capability 1.1 or later) and the _threadfence() intrinsic (CUDA 2.2 or later).

Minimum Required GPU	SM 1.1
Key Concepts	Data-Parallel Algorithms, Performance Strategies
Source	zip | tar.gz

CUDA Radix Sort (Thrust Library)

This sample demonstrates a very fast and efficient parallel radix sort uses Thrust library (http://code.google.com/p/thrust/). The included RadixSort class can sort either key-value pairs (with float or unsigned integer keys) or keys only. The optimized code in this sample (and also in reduction and scan) uses a technique known as warp-synchronous programming, which relies on the fact that within a warp of threads running on a CUDA GPU, all threads execute instructions synchronously. The code uses this to avoid __syncthreads() when threads within a warp are sharing data via __shared__ memory. It is important to note that for this to work correctly without race conditions on all GPUs, the shared memory used in these warp-synchronous expressions must be declared volatile. If it is not declared volatile, then in the absence of __syncthreads(), the compiler is free to delay stores to __shared__ memory and keep the data in registers (an optimization technique), which will result in incorrect execution. So please heed the use of volatile in these samples and use it in the same way in any code you derive from them.

Minimum Required GPU	SM 1.0
Key Concepts	Data-Parallel Algorithms, Performance Strategies
Source	zip | tar.gz
Whitepaper	readme.txt

CUDA Sorting Networks

This sample implements bitonic sort and odd-even merge sort (also known as Batcher's sort), algorithms belonging to the class of sorting networks. While generally subefficient on large sequences compared to algorithms with better asymptotic algorithmic

complexity (i.e. merge sort or radix sort), may be the algorithms of choice for sorting batches of short- to mid-sized (key, value) array pairs. Refer to the excellent tutorial by H. W. Lang http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/indexen.htm

Minimum Required GPU	SM 1.0
Key Concepts	Data-Parallel Algorithms
Source	zip | tar.gz

Merge Sort

This sample implements a merge sort (also known as Batcher's sort), algorithms belonging to the class of sorting networks. While generally subefficient on large sequences compared to algorithms with better asymptotic algorithmic complexity (i.e. merge sort or radix sort), may be the algorithms of choice for sorting batches of short- to mid-sized (key, value) array pairs. Refer to the excellent tutorial by H. W. Lang http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/indexen.htm

Minimum Required GPU	SM 1.0
Key Concepts	Data-Parallel Algorithms
Source	zip | tar.gz

3.8. Cudalibraries Reference

simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)

This sample implements a simple CUBLAS function calls that call GPU device API library running CUBLAS functions. This sample requires a SM 3.5 capable device.

Minimum Required GPU	KEPLER SM 3.5
CUDA API	cublasCreate, cublasSetVector, cublasSgemm, cudaMalloc, cudaFree, cudaMemcpy
Key Concepts	CUDA Dynamic Parallelism, Linear Algebra
Source	zip | tar.gz

MersenneTwisterGP11213

This sample demonstrates the Mersenne Twister random number generator GP11213 in cuRAND.

Minimum Required GPU	SM 1.0
Key Concepts	Computational Finance, CURAND Library
Source	zip | tar.gz

GrabCut with NPP

CUDA Implementation of Rother et al. GrabCut approach using the 8 neighborhood NPP Graphcut primitive introduced in CUDA 4.1. (C. Rother, V. Kolmogorov, A. Blake. GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. ACM Transactions on Graphics (SIGGRAPH'04), 2004)

Minimum Required GPU	SM 1.0
Key Concepts	Performance Strategies, Image Processing, NPP Library
Source	zip | tar.gz

Image Segmentation using Graphcuts with NPP

This sample that demonstrates how to perform image segmentation using the NPP GraphCut function.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, Performance Strategies, NPP Library
Source	zip | tar.gz

Histogram Equalization with NPP

This SDK sample demonstrates how to use NPP for histogram equalization for image data.

Minimum Required GPU	SM 1.1
Key Concepts	Image Processing, Performance Strategies, NPP Library
Source	zip | tar.gz

FreeImage and NPP Interopability

A simple SDK sample demonstrate how to use FreeImage library with NPP.

Minimum Required GPU	SM 1.0
Key Concepts	Performance Strategies, Image Processing, NPP Library
Source	zip | tar.gz

Box Filter with NPP

A NPP SDK sample that demonstrates how to use NPP FilterBox function to perform a Box Filter.

Minimum Required GPU	SM 1.0
Key Concepts	Performance Strategies, Image Processing, NPP Library
Source	zip | tar.gz

Preconditioned Conjugate Gradient

This sample implements a preconditioned conjugate gradient solver on GPU using CUBLAS and CUSPARSE library.

Minimum Required GPU	SM 1.0
Key Concepts	Linear Algebra, CUBLAS Library, CUSPARSE Library
Source	zip | tar.gz

Monte Carlo Single Asian Option

This sample uses Monte Carlo to simulate Single Asian Options using the NVIDIA CURAND library.

Minimum Required GPU	SM 1.0
Key Concepts	Random Number Generator, Computational Finance, CURAND Library
Source	zip | tar.gz

Monte Carlo Estimation of Pi (batch QRNG)

This sample uses Monte Carlo simulation for Estimation of Pi (using batch QRNG). This sample also uses the NVIDIA CURAND library.

Minimum Required GPU	SM 1.0
Key Concepts	Random Number Generator, Computational Finance, CURAND Library
Source	zip | tar.gz

Monte Carlo Estimation of Pi (batch PRNG)

This sample uses Monte Carlo simulation for Estimation of Pi (using batch PRNG). This sample also uses the NVIDIA CURAND library.

Minimum Required GPU	SM 1.0
Key Concepts	Random Number Generator, Computational Finance, CURAND Library
Source	zip | tar.gz

Monte Carlo Estimation of Pi (batch inline QRNG)

This sample uses Monte Carlo simulation for Estimation of Pi (using batch inline QRNG). This sample also uses the NVIDIA CURAND library.

Minimum Required GPU	SM 1.0
Key Concepts	Random Number Generator, Computational Finance, CURAND Library
Source	zip | tar.gz

Monte Carlo Estimation of Pi (inline PRNG)

This sample uses Monte Carlo simulation for Estimation of Pi (using inline PRNG). This sample also uses the NVIDIA CURAND library.

Minimum Required GPU	SM 1.0
Key Concepts	Random Number Generator, Computational Finance, CURAND Library
Source	zip | tar.gz

Random Fog

This sample illustrates pseudo- and quasi- random numbers produced by CURAND.

Minimum Required GPU	SM 1.0
Key Concepts	3D Graphics, CURAND Library
Source	zip | tar.gz

ConjugateGradient

This sample implements a conjugate gradient solver on GPU using CUBLAS and CUSPARSE library.

Minimum Required GPU	SM 1.0
Key Concepts	Linear Algebra, CUBLAS Library, CUSPARSE Library
Source	zip | tar.gz

batchCUBLAS

A SDK sample that demonstrates how using batched CUBLAS API calls to improve overall performance.

Minimum Required GPU	SM 1.0
Key Concepts	Linear Algebra, CUBLAS Library
Source	zip | tar.gz

Simple CUBLAS

Example of using CUBLAS using the new CUBLAS API interface available in CUDA 4.0.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, CUBLAS Library
Source	zip | tar.gz

Simple CUFFT

Example of using CUFFT. In this example, CUFFT is used to compute the 1D-convolution of some signal with some filter by transforming both into frequency domain, multiplying them together, and transforming the signal back to time domain.

Minimum Required GPU	SM 1.0
Key Concepts	Image Processing, CUFFT Library
Source	zip | tar.gz

Chapter 4. KNOWN ISSUES

4.1. Known Issues in CUDA Samples for Windows

Please see the CUDA Toolkit Release Notes for additional issues.

- In code sample **`alignedTypes`**, the following aligned type does not provide maximum throughput because of a compiler bug:

```
typedef struct __align__(16) {
    unsigned int r, g, b;
} RGB32;
```

  The workaround is to use the following type instead:

```
typedef struct __align__(16) {
    unsigned int r, g, b, a;
} RGBA32;
```

  as illustrated in the sample.
- By default the CUDA Samples 5.5 will be installed to:

  so it will not have conflicts with Vista with UAC.

  By default, UAC is enabled for Vista. If UAC is disabled, the user is free to install the samples in other folders.

  Before CUDA 2.1, the samples installation path would be under:

```
Program Files\NVIDIA Corporation\NVIDIA CUDA SDK
```

  Starting with CUDA 2.1, the new default installation folder was:

```
Application Data\NVIDIA Corporation\NVIDIA CUDA SDK
```

  residing under **`All Users`** or **`Current`**.

  For NVIDIA GPU Computing 4.2 Release, the installation path was under:

  For NVIDIA GPU Computing 5.0 Release, the installation path was under:

With NVIDIA CUDA Samples 5.5 Release, the new default installation folder is:

residing under **`All Users`** or **`Current`**.

- There are number of samples that are not pre-built with the CUDA Samples. Why are these samples not pre-built?

  `cudaOpenMP`, **`simpleMPI`**, **`ExcelCUDA2007`**, **`ExcelCUDA2010`**

  The samples may depend on other header and library packages to be installed on the development machine. These are not distributed with the CUDA Samples, hence these are not pre-built.
- The following Direct3D samples are not officially supported on Telsa GPU:

  `cudaDecodeD3D9`, **`fluidsD3D9`**, **`simpleD3D9`**, **`simpleD3D9Texture`**, **`simpleD3D10`**, **`simpleD3D10Texture`**, **`simpleD3D11Texture`**, **`vFlockingD3D10`**

  These samples will not run and report that a Direct3D device is not available.

4.2. Known Issues in CUDA Samples for Linux

Please see the CUDA Toolkit Release Notes for additional issues.

- The samples that make use of OpenGL fail to build or link. This is because many of the default installations for many Linux distributions do not include the necessary OpenGL, GLUT, GLU, GLEW, X11, Xi, Xlib, or Xmi headers or libraries. Here are some general and specific solutions:
  - Redhat 4 Linux Distributions

    ```
    ld: cannot find -lglut
    ```

    On some Linux installations, building the **`simpleGL`** example shows the following linking error:

    ```
    /usr/bin/ld: cannot find -lglut
    ```

    Typically this is because the makefiles look for **`libglut.so`** and not for variants of it (like **`libglut.so.3`**). To confirm this is the problem, simply run the following command:

    ```
    ls /usr/lib | grep glut
    ls /usr/lib64 | grep glut
    ```

    You should see the following (or similar) output:

    ```
    lrwxrwxrwx  1 root root     16 Jan  9 14:06 libglut.so.3 ->
     libglut.so.3.8.0
    -rwxr-xr-x  1 root root 164584 Aug 14  2004 libglut.so.3.8.0
    ```

    If you have **`libglut.so.3`** in **`/usr/lib`** and/or **`/usr/lib64`**, simply run the following command as root:

    ```
    ln -s /usr/lib/libglut.so.3 /usr/lib/libglut.so
    ln -s /usr/lib64/libglut.so.3 /usr/lib64/libglut.so
    ```

If you do NOT have **libglut.so.3** then you can check whether the **glut** package is installed on your RHEL system with the following command:

```
rpm -qa | grep glut
```

You should see **freeglut-2.2.2-14** or similar in the output. If not, you or your system administrator should install the package **freeglut-2.2.2-14**. Refer to the Red Hat and/or rpm documentation for instructions.

If you have **libglut.so.3** but you do not have write access to **/usr/lib**, you can also fix the problem by creating the soft link in a directory to which you have write permissions and then add that directory to the library search path (**-L**) in the **Makefile**.

- Some Linux distributions (i.e., Redhat or Fedora) do not include the GLU library. For the latest packages download this file from this website. Please make sure you match the correct Linux distribution.

  http://fr.rpmfind.net/linux/rpm2html/search.php?query=libGLU.so.1&submit=Search+...
- (SLED11) SUSE Linux 11 is missing:

  libGLU, **libX11**, **libXi**, **libXm**, **libXmu**

  This particular version of SUSE Linux Enterprise Edition 11 (SLED11) does not have the proper symbolic links for the following libraries:

  - **libGLU**

    ```
    ls /usr/lib | grep GLU
    ls /usr/lib64 | grep GLU

    libGLU.so.1
    libGLU.so.1.3.0370300
    ```

    To create the proper symbolic links (32-bit and 64-bit OS):

    ```
    ln -s /usr/lib/libGLU.so.1 /usr/lib/libGLU.so
    ln -s /usr/lib64/libGLU.so.1 /usr/lib64/libGLU.so
    ```
  - **libX11**

    ```
    ls /usr/lib | grep X11
    ls /usr/lib64 | grep X11

    libX11.so.6
    libX11.so.6.2.0
    ```

    To create the proper symbolic links (32-bit and 64-bit OS):

    ```
    ln -s /usr/lib/libX11.so.6 /usr/lib/libX11.so
    ln -s /usr/lib64/libX11.so.6 /usr/lib64/libX11.so
    ```
  - **libXi**

    ```
    ls /usr/lib | grep Xi
    ls /usr/lib64 | grep Xi

    libXi.so.6
    libXi.so.6.0.0
    ```

    To create the proper symbolic links (32-bit and 64-bit OS):

    ```
    ln -s /usr/lib/libXi.so.6  /usr/lib/libXi.so
    ln -s /usr/lib64/libXi.so.6  /usr/lib64/libXi.so
    ```

- **libXm**

  ```
  ls /usr/lib | grep Xm
  ls /usr/lib64 | grep Xm
  ```

  ```
  libXm.so.6
  libXm.so.6.0.0
  ```

  To create the proper symbolic links (32-bit and 64-bit OS):

  ```
  ln -s /usr/lib/libXm.so.6  /usr/lib/libXm.so
  ln -s /usr/lib64/libXm.so.6  /usr/lib64/libXm.so
  ```

- **libXmu**

  ```
  ls /usr/lib | grep Xmu
  ls /usr/lib64 | grep Xmu
  ```

  ```
  libXmu.so.6
  libXmu.so.6.0.0
  ```

  To create the proper symbolic links (32-bit and 64-bit OS):

  ```
  ln -s /usr/lib/libXmu.so.6  /usr/lib/libXmu.so
  ln -s /usr/lib64/libXmu.so.6  /usr/lib64/libXmu.so
  ```

- Ubuntu Linux unable to build these samples that use OpenGL

  The default Ubuntu distribution is missing many libraries.

  - What is missing are the GLUT, Xi, Xmu, GL, and X11 headers. To add these headers and libraries to your distribution, type the following in at the command line:

    ```
    sudo apt-get install freeglut3-dev build-essential libx11-dev
     libxmu-dev libxi-dev libgl1-mesa-glx libglu1-mesa libglu1-mesa-dev
    ```

  - Note, by installing Mesa, you may see linking errors against **libGL**. This can be solved below:

    ```
    cd /usr/lib/
    sudo rm libGL.so
    sudo ln -s libGL.so.1 libGL.so
    ```

- In code sample **alignedTypes**, the following aligned type does not provide maximum throughput because of a compiler bug:

  ```
  typedef struct __align__(16) {
      unsigned int r, g, b;
  } RGB32;
  ```

  The workaround is to use the following type instead:

  ```
  typedef struct __align__(16) {
      unsigned int r, g, b, a;
  } RGBA32;
  ```

  as illustrated in the sample.

- Unable to build **simpleMPI** sample on Linux Distros

  ```
  simpleMPI.cpp:35:17: error: mpi.h: No such file or directory
  ```

  The Linux system is missing the libraries and headers for MPI.

  - For OpenSuSE or RedHat distributions: Search http://www.rpmfind.net for **openmpi-devel** for your specific distribution

    For Ubuntu or Debian distributions, using **apt-get**:

    ```
    sudo apt-get
    ```

```
install build-essential openmpi-bin openmpi-dev
```

- For 32-bit Linux distributions:

```
ln -s /usr/lib/mpi/gcc/openmpi/lib/libmpi_cxx.so.0 /usr/lib/
libmpi_cxx.so
ln -s /usr/lib/mpi/gcc/openmpi/lib/libmpi.so.0 /usr/lib/libmpi.so
ln -s /usr/lib/mpi/gcc/openmpi/lib/libopen-rte.so.0 /usr/lib/
libopen-rte.so
ln -s /usr/lib/mpi/gcc/openmpi/lib/libopen-pal.so.0 /usr/lib/
libopen-pal.so
```

- For 64-bit Linux distributions:

```
ln -s /usr/lib64/mpi/gcc/openmpi/lib64/libmpi_cxx.so.0 /usr/lib64/
libmpi_cxx.so
ln -s /usr/lib64/mpi/gcc/openmpi/lib64/libmpi.so.0 /usr/lib64/
libmpi.so
ln -s /usr/lib64/mpi/gcc/openmpi/lib64/libopen-rte.so.0 /usr/lib64/
libopen-rte.so
ln -s /usr/lib64/mpi/gcc/openmpi/lib64/libopen-pal.so.0 /usr/lib64/
libopen-pal.so
```

- Fedora 13 or 14 has linking error when building the following samples:

  MonteCarloMultiGPU, **simpleMultiGPU**, **threadMigration**

  The following error is seen:

```
make -C 6_Advanced/threadMigration/
make[1]: Entering directory `/root/sdk/C/6_Advanced/threadMigration'
/usr/bin/ld: obj/i386/release/threadMigration.cpp.o: undefined reference
 to symbol 'pthread_create@@GLIBC_2.1'
/usr/bin/ld: note: 'pthread_create@@GLIBC_2.1' is defined in DSO /lib/
libpthread.so.0 so try adding it to the linker command line
/lib/libpthread.so.0: could not read symbols: Invalid operation
collect2: ld returned 1 exit status
make[1]: *** [../../bin/linux/release/threadMigration] Error 1
make[1]: Leaving directory `/root/sdk/C/6_Advanced/threadMigration'
make: *** [6_Advanced/threadMigration/Makefile.ph_build] Error 2
```

  For these Linux distributions: Fedora 13 or 14, symbolic links are missing from the following libraries:

  libpthread

  To create the proper symbolic links (32-bit OS and 64-bit OS) type this:

```
ln -s /usr/lib/libpthread.so.0 /usr/lib/libpthread.so
ln -s /usr/lib64/libpthread.so.0 /usr/lib64/libpthread.so
```

4.3. Known Issues in CUDA Samples for Mac OS X

In addition, please look at the CUDA Toolkit Release Notes for additional issues.

- With release CUDA 5.0, support for Mac OS X 10.8.x (Mountain Lion) is added
- With release CUDA 4.0, support for Mac OS X 10.7.x (Lion) is added
- With release CUDA 3.1, Mac OS X now supports CUDA Runtime API (with 64-bit applications)
- CUDA 3.1 Beta and newer now supports 10.6.3 (Snow Leopard) 64-bit Runtime API.

- For CUDA 3.0, Note on CUDA Mac 10.5.x (Leopard) or 10.6.x (Snow Leopard). CUDA applications built with the CUDA driver API can run as either 32-bit or 64-bit applications. CUDA applications using CUDA Runtime APIs can only be built on 32-bit applications.

Chapter 5.
KEY CONCEPTS AND ASSOCIATED SAMPLES

The tables below describe the key concepts of the CUDA Toolkit and lists the samples that illustrate how that concept is used.

Basic Key Concepts

Basic Concepts demonstrates how to make use of CUDA features.

Table 1 Basic Key Concepts and Associated Samples

Basic Key Concept	Description	Samples
3D Graphics	*3D Rendering*	Random Fog, Simple Direct3D10 (Vertex Array), Simple OpenGL
3D Textures	*Volume Textures*	Simple Texture 3D
Assert	*GPU Assert*	simpleAssert
Asynchronous Data Transfers	*Overlapping I/O and Compute*	Simple Multi Copy and Compute, Simple Multi-GPU, Simple Peer-to-Peer Transfers with Multi-GPU, asyncAPI, simpleStreams
Atomic Intrinsics	*Using atomics with GPU kernels*	Simple Atomic Intrinsics
C++ Function Overloading	*Use C++ overloading with GPU kernels*	cppOverload
C++ Templates	*Using Templates with GPU kernels*	Simple Templates
CUBLAS	*CUDA BLAS samples*	Matrix Multiplication (CUBLAS)

Basic Key Concept	Description	Samples
CUBLAS Library	*CUDA BLAS samples*	Simple CUBLAS, batchCUBLAS
CUDA Data Transfers	*CUDA Data I/O*	Template using CUDA Runtime
CUDA Driver API	*Samples that show the CUDA Driver API*	Device Query Driver API, Matrix Multiplication (CUDA Driver API Version), Simple Texture (Driver Version), Using Inline PTX, Vector Addition Driver API
CUDA Dynamic Parallelism	*Dynamic Parallelism with GPU Kernels (SM 3.5)*	Simple Print (CUDA Dynamic Parallelism), simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)
CUDA Runtime API	*Samples that use the Runtime API*	Device Query, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Simple Texture, Vector Addition
CUDA Streams	*Stream API definies a sequence of operations that can be overlapped with I/O*	Simple CUDA Callbacks
CUDA Streams and Events	*Synchronizing Kernels with Event Timers and Streams*	Bandwidth Test, Simple Multi Copy and Compute, Simple Multi-GPU, asyncAPI, cppOverload, simpleStreams
CUDA Systems Integration	*Samples that integrate with Multi Process (OpenMP, IPC, and MPI)*	cudaOpenMP, simpleIPC, simpleMPI
CUFFT Library	*Samples that use the CUDA FFT accelerated library*	Simple CUFFT
CURAND Library	*Samples that use the CUDA random number generator*	MersenneTwisterGP11213, Random Fog
Callback Functions	*Creating Callback functions with GPU kernels*	Simple CUDA Callbacks
Computational Finance	*Finance Algorithms*	Black-Scholes Option Pricing, MersenneTwisterGP11213

Basic Key Concept	Description	Samples
Data Parallel Algorithms	*Samples that show good usage of Data Parallel Algorithms*	CUDA Separable Convolution, Texture-based Separable Convolution
Debugging	*Samples useful for debugging*	simplePrintf
Device Memory Allocation	*Samples that show GPU Device side memory allocation*	Template, Template using CUDA Runtime
Device Query	*Sample showing simple device query of information*	Device Query, Device Query Driver API
GPU Performance	*Samples demonstrating high performance and data I/O*	Simple Multi Copy and Compute
Graphics Interop	*Samples that demonstrate interop between graphics APIs and CUDA*	Bicubic B-spline Interoplation, Bilateral Filter, Box Filter, CUDA and OpenGL Interop of Images, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target, Simple Direct3D9 (Vertex Arrays), Simple OpenGL, Simple Texture 3D
Image Processing	*Samples that demonstrate image processing algorithms in CUDA*	Bicubic B-spline Interoplation, Bilateral Filter, Box Filter, Box Filter with NPP, CUDA Separable Convolution, CUDA and OpenGL Interop of Images, FreeImage and NPP Interopability, GrabCut with NPP, Histogram Equalization with NPP, Image Segmentation using Graphcuts with NPP, Pitch Linear Texture, Simple CUBLAS, Simple CUFFT, Simple D3D11 Texture, Simple Surface Write, Simple Texture, Simple Texture (Driver Version), Simple Texture 3D, Texture-based Separable Convolution
InterProcess Communication	*Samples that demonstrate Inter Process Communication between processes*	simpleIPC
Linear Algebra	*Samples demonstrating linear algebra with CUDA*	Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), batchCUBLAS, simpleDevLibCUBLAS

Basic Key Concept	Description	Samples
		GPU Device API Library Functions (CUDA Dynamic Parallelism)
MPI	*Samples demonstrating how to use CUDA with MPI programs*	simpleMPI
Matrix Multiply	*Samples demonstrating matrix multiply CUDA*	Matrix Multiplication (CUDA Driver API Version)
Multi-GPU	*Samples demonstrating how to take advantage of multiple GPUs and CUDA*	Simple Multi-GPU, Simple Peer-to-Peer Transfers with Multi-GPU
Multithreading	*Samples demonstrating how to use multithreading with CUDA*	Simple CUDA Callbacks, Simple Multi-GPU, cudaOpenMP, simpleMPI
NPP Library	*Samples demonstrating how to use NPP (NVIDIA Performance Primitives) for image processing*	Box Filter with NPP, FreeImage and NPP Interopability, GrabCut with NPP, Histogram Equalization with NPP, Image Segmentation using Graphcuts with NPP
OpenMP	*Samples demonstrating how to use OpenMP*	cudaOpenMP
Overlap Compute and Copy	*Samples demonstrating how to overlap Compute and Data I/O*	Simple Multi Copy and Compute
PTX Assembly	*Samples demonstrating how to use PTX code with CUDA*	Using Inline PTX
Peer to Peer	*Samples demonstrating how to handle P2P data transfers between multiple GPUs*	simpleIPC
Peer to Peer Data Transfers	*Samples demonstrating how to handle P2P data transfers between multiple GPUs*	Simple Peer-to-Peer Transfers with Multi-GPU
Performance Strategies	*Samples demonstrating high performance with CUDA*	Bandwidth Test, Box Filter with NPP, CUDA and OpenGL Interop of Images, Clock, FreeImage and NPP Interopability, GrabCut with NPP, Histogram Equalization with NPP, Image Segmentation using Graphcuts with NPP, Matrix Multiplication (CUBLAS), Simple Peer-to-Peer Transfers with Multi-GPU, Using Inline PTX, simpleZeroCopy

Basic Key Concept	Description	Samples
Pinned System Paged Memory	*Samples demonstrating how to properly handle data I/O efficiently between the CPU host and GPU video memory*	simpleZeroCopy
Separate Compilation	*Samples demonstrating how to use CUDA library linking*	Simple Static GPU Device Library
Surface Writes	*Samples demonstrating how to use Surface Writes with GPU kernels*	Simple Surface Write, Simple Texture 3D
Texture	*Samples demonstrating how to use textures GPU kernels*	Pitch Linear Texture, Simple Cubemap Texture, Simple D3D10 Texture, Simple D3D9 Texture, Simple Direct3D10 Render Target, Simple Layered Texture, Simple Surface Write, Simple Texture, Simple Texture (Driver Version), Texture-based Separable Convolution
Unified Virtual Address Space	*Samples demonstrating how to use UVA with CUDA programs*	Simple Peer-to-Peer Transfers with Multi-GPU
Vector Addition	*Samples demonstrating how to use Vector Addition with CUDA programs*	Vector Addition, Vector Addition Driver API, simpleZeroCopy
Vertex Buffers	*Samples demonstrating how to use Vertex Buffers with CUDA kernels*	Simple OpenGL
Volume Processing	*Samples demonstrating how to use 3D Textures for volume rendering*	Simple Cubemap Texture, Simple Layered Texture
Vote Intrinsics	*Samples demonstrating how to use vote intrinsics with CUDA*	Simple Vote Intrinsics

Advanced Key Concepts

Advanced Concepts demonstrate advanced techniques and algorithms implemented with CUDA.

Table 2 Advanced Key Concepts and Associated Samples

Advanced Key Concept	Description	Samples
2D Textures	*Texture Mapping*	SLI D3D10 Texture
3D Graphics	*3D Rendering*	Marching Cubes Isosurfaces
3D Textures	*Volume Textures*	Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
CUBLAS Library	*CUDA BLAS samples*	ConjugateGradient, Preconditioned Conjugate Gradient
CUDA Driver API	*Samples that show the CUDA Driver API*	CUDA Context Thread Management, Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), PTX Just-in-Time compilation
CUDA Dynamic Parallelism	*Dynamic Parallelism with GPU Kernels (SM 3.5)*	Advanced Quicksort (CUDA Dynamic Parallelism), LU Decomposition (CUDA Dynamic Parallelism), Quad Tree (CUDA Dynamic Parallelism), Simple Quicksort (CUDA Dynamic Parallelism)
CUDA Dynamically Linked Library	*Dynamic loading of the CUDA DLL using CUDA Driver API*	Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version)
CUDA Systems Integration	*Samples that integrate with Multi Process (OpenMP, IPC, and MPI)*	simpleHyperQ
CUFFT Library	*Samples that use the CUDA FFT accelerated library*	CUDA FFT Ocean Simulation, FFT-Based 2D Convolution
CUFFT Librarys	*Samples that use the CUDA FFT accelerated library*	Fluids (Direct3D Version), Fluids (OpenGL Version)
CURAND Library	*Samples that use the CUDA random number generator*	Monte Carlo Estimation of Pi (batch PRNG), Monte Carlo Estimation of Pi (batch QRNG), Monte Carlo Estimation of Pi (batch inline QRNG) , Monte Carlo Estimation of Pi

Advanced Key Concept	Description	Samples
		(inline PRNG), Monte Carlo Single Asian Option
CUSPARSE Library	*Samples that use the CUSPARSE (Sparse Vector Matrix Multiply) functions*	ConjugateGradient, Preconditioned Conjugate Gradient
Computational Finance	*Finance Algorithms*	Binomial Option Pricing, Monte Carlo Estimation of Pi (batch PRNG), Monte Carlo Estimation of Pi (batch QRNG), Monte Carlo Estimation of Pi (batch inline QRNG) , Monte Carlo Estimation of Pi (inline PRNG), Monte Carlo Single Asian Option, Niederreiter Quasirandom Sequence Generator, Sobol Quasirandom Number Generator
Data Parallel Algorithms	*Samples that show good usage of Data Parallel Algorithms*	CUDA Histogram, CUDA N-Body Simulation, Mandelbrot, Optical Flow, Particles, Smoke Particles, VFlockingD3D10
Data-Parallel Algorithms	*Samples that show good usage of Data Parallel Algorithms*	CUDA Parallel Prefix Sum (Scan), CUDA Parallel Prefix Sum with Shuffle Intrinsics (SHFL_Scan), CUDA Parallel Reduction, CUDA Radix Sort (Thrust Library), CUDA Segmentation Tree Thrust Library, CUDA Sorting Networks, Fast Walsh Transform, Merge Sort, threadFenceReduction
Graphics Interop	*Samples that demonstrate interop between graphics APIs and CUDA*	Bindless Texture, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, CUDA Video Encode (C Library) API, Fluids (Direct3D Version), Fluids (OpenGL Version), Function Pointers, Mandelbrot, Particles, Post-Process in OpenGL, Recursive Gaussian Filter, SLI D3D10 Texture, Smoke Particles, Sobel Filter, VFlockingD3D10, Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes

Advanced Key Concept	Description	Samples
Image Compression	*Samples that demonstrate image and video compression*	DirectX Texture Compressor (DXTC)
Image Processing	*Samples that demonstrate image processing algorithms in CUDA*	1D Discrete Haar Wavelet Decomposition, CUDA FFT Ocean Simulation, CUDA Histogram, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, DCT8x8, DirectX Texture Compressor (DXTC), FFT-Based 2D Convolution, Function Pointers, Image denoising, Optical Flow, Post-Process in OpenGL, Recursive Gaussian Filter, SLI D3D10 Texture, Sobel Filter, Stereo Disparity Computation (SAD SIMD Intrinsics), Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
Linear Algebra	*Samples demonstrating linear algebra with CUDA*	ConjugateGradient, Eigenvalues, Fast Walsh Transform, Matrix Transpose, Preconditioned Conjugate Gradient, Scalar Product
OpenGL Graphics Interop	*Samples demonstrating how to use interoperability CUDA with OpenGL*	Marching Cubes Isosurfaces
Performance Strategies	*Samples demonstrating high performance with CUDA*	Aligned Types, CUDA C 3D FDTD, CUDA Parallel Prefix Sum (Scan), CUDA Parallel Prefix Sum with Shuffle Intrinsics (SHFL_Scan), CUDA Parallel Reduction, CUDA Radix Sort (Thrust Library), CUDA Segmentation Tree Thrust Library, Concurrent Kernels, Matrix Transpose, Particles, SLI D3D10 Texture, VFlockingD3D10, simpleHyperQ, threadFenceReduction
Physically Based Simulation	*Samples demonstrating high performance collisions and/or physocal interactions*	Marching Cubes Isosurfaces

Advanced Key Concept	Description	Samples
Physically-Based Simulation	*Samples demonstrating high performance collisions and/or physocal interactions*	CUDA N-Body Simulation, Fluids (Direct3D Version), Fluids (OpenGL Version), Particles, Smoke Particles, VFlockingD3D10
Random Number Generator	*Samples demonstrating how to use random number generation with CUDA*	Monte Carlo Estimation of Pi (batch PRNG), Monte Carlo Estimation of Pi (batch QRNG), Monte Carlo Estimation of Pi (batch inline QRNG) , Monte Carlo Estimation of Pi (inline PRNG), Monte Carlo Single Asian Option
Recursion	*Samples demonstrating recursion on CUDA*	Interval Computing
Surface Writes	*Samples demonstrating how to use Surface Writes with GPU kernels*	Volumetric Filtering with 3D Textures and Surface Writes
Templates	*Samples demonstrating how to use templates GPU kernels*	Interval Computing
Texture	*Samples demonstrating how to use textures GPU kernels*	Bindless Texture
Vertex Buffers	*Samples demonstrating how to use Vertex Buffers with CUDA kernels*	Marching Cubes Isosurfaces
Video Compression	*Samples demonstrating how to use video compression with CUDA*	1D Discrete Haar Wavelet Decomposition, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, CUDA Video Encode (C Library) API, DCT8x8, Fast Walsh Transform
Video Intrinsics	*Samples demonstrating how to use video intrinsics with CUDA*	Stereo Disparity Computation (SAD SIMD Intrinsics)

Chapter 6.
CUDA API AND ASSOCIATED SAMPLES

The tables below list the samples associated with each CUDA API.

CUDA Driver API Samples

The table below lists the samples associated with each CUDA Driver API.

Table 3 CUDA Driver API and Associated Samples

CUDA Driver API	Samples
cuLaunchKernel	Vector Addition Driver API
cuMemAlloc	Vector Addition Driver API
cuMemFree	Vector Addition Driver API
cuMemcpyDtoH	Vector Addition Driver API
cuMemcpyHtoD	Vector Addition Driver API
cuModuleGetFunction	Vector Addition Driver API
cuModuleLoad	Vector Addition Driver API
cuModuleLoadDataEx	CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Vector Addition Driver API
cudaMalloc	Template using CUDA Runtime

CUDA Runtime API Samples

The table below lists the samples associated with each CUDA Runtime API.

Table 4 CUDA Runtime API and Associated Samples

CUDA Runtime API	Samples
CreateHWEncInterfaceInstance	CUDA Video Encode (C Library) API
CreateHWEncoder	CUDA Video Encode (C Library) API
DestroyEncoder	CUDA Video Encode (C Library) API
EncodeFrameUT	CUDA Video Encode (C Library) API
GetCodecType	CUDA Video Encode (C Library) API
GetHWEncodeCaps	CUDA Video Encode (C Library) API
GetParamValue	CUDA Video Encode (C Library) API
GetSPSPPS	CUDA Video Encode (C Library) API
IsSupportedCodec	CUDA Video Encode (C Library) API
IsSupportedCodecProfile	CUDA Video Encode (C Library) API
IsSupportedParam	CUDA Video Encode (C Library) API
RegisterCB	CUDA Video Encode (C Library) API
SetCodecType	CUDA Video Encode (C Library) API
SetDefaultParam	CUDA Video Encode (C Library) API
SetParamValue	CUDA Video Encode (C Library) API
cuArrayCreate	Simple Texture (Driver Version)
cuArrayDestroy	Simple Texture (Driver Version)
cuCtxCreate	CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuCtxDestroy	CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuCtxDetach	Simple Texture (Driver Version)

CUDA Runtime API	Samples
cuCtxPopCurrent	CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuCtxPushCurrent	CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuCtxSynchronize	Simple Texture (Driver Version)
cuD3D9CtxCreate	CUDA Video Decoder D3D9 API
cuD3D9GetDevice	CUDA Video Decoder D3D9 API
cuD3D9MapResources	CUDA Video Decoder D3D9 API
cuD3D9RegisterResource	CUDA Video Decoder D3D9 API
cuD3D9ResourceGetMappedPitch	CUDA Video Decoder D3D9 API
cuD3D9ResourceGetMappedPointer	CUDA Video Decoder D3D9 API
cuD3D9ResourceSetMapFlags	CUDA Video Decoder D3D9 API
cuD3D9UnmapResources	CUDA Video Decoder D3D9 API
cuD3D9UnregisterResource	CUDA Video Decoder D3D9 API
cuDeviceComputeCapability	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API
cuDeviceGet	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuDeviceGetAttribute	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API
cuDeviceGetCount	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API
cuDeviceGetName	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuDeviceTotalMem	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API
cuDriverGetVersion	Device Query Driver API
cuGLCtxCreate	CUDA Video Decoder GL API
cuGLGetDevice	CUDA Video Decoder GL API
cuGLMapResources	CUDA Video Decoder GL API

CUDA Runtime API	Samples
cuGLRegisterResource	CUDA Video Decoder GL API
cuGLResourceGetMappedPitch	CUDA Video Decoder GL API
cuGLResourceGetMappedPointer	CUDA Video Decoder GL API
cuGLResourceSetMapFlags	CUDA Video Decoder GL API
cuGLUnmapResources	CUDA Video Decoder GL API
cuGLUnregisterResource	CUDA Video Decoder GL API
cuInit	Device Query Driver API
cuLaunchGridAsync	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuLaunchKernel	CUDA Context Thread Management, Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Vector Addition Driver API
cuMemAlloc	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Vector Addition Driver API
cuMemAllocHost	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuMemFree	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Vector Addition Driver API
cuMemFreeHost	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuMemcpy2D	Simple Texture (Driver Version)
cuMemcpyDtoH	CUDA Context Thread Management, Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Vector Addition Driver API
cuMemcpyDtoHAsync	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API

CUDA Runtime API	Samples
cuMemcpyHtoD	Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Vector Addition Driver API
cuMemsetD8	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuModuleGetFunction	CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Vector Addition Driver API
cuModuleGetGlobal	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuModuleGetTexRef	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Simple Texture (Driver Version)
cuModuleLoad	CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Vector Addition Driver API
cuModuleLoadDataEx	CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Vector Addition Driver API
cuModuleUnload	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuParamSetSize	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuParamSetTexRef	Simple Texture (Driver Version)
cuParamSeti	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuParamSetv	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuStreamCreate	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuTexRefSetAddressMode	Simple Texture (Driver Version)
cuTexRefSetArray	Simple Texture (Driver Version)
cuTexRefSetFilterMode	Simple Texture (Driver Version)
cuTexRefSetFlags	Simple Texture (Driver Version)

CUDA Runtime API	Samples
cuTexRefSetFormat	Simple Texture (Driver Version)
cublasCreate	Matrix Multiplication (CUBLAS), simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)
cublasSetVector	simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)
cublasSgemm	Matrix Multiplication (CUBLAS), simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)
cudaBindSurfaceToArray	Simple Surface Write
cudaBindTexture2D	Pitch Linear Texture
cudaBindTextureToArray	Pitch Linear Texture, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture
cudaCreateChannelDesc	Pitch Linear Texture, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture
cudaD3D10GetDevice	SLI D3D10 Texture, Simple D3D10 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target
cudaD3D10SetDirect3DDevice	SLI D3D10 Texture, Simple D3D10 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target
cudaD3D10SetGLDevice	VFlockingD3D10
cudaD3D11GetDevice	Simple D3D11 Texture
cudaD3D11SetDirect3DDevice	Simple D3D11 Texture
cudaD3D9GetDevice	Simple D3D9 Texture, Simple Direct3D9 (Vertex Arrays)
cudaD3D9SetDirect3DDevice	Simple D3D9 Texture, Simple Direct3D9 (Vertex Arrays)
cudaD3D9SetGLDevice	Fluids (Direct3D Version)
cudaDeviceCanAccessPeer	Simple Peer-to-Peer Transfers with Multi-GPU
cudaDeviceDisablePeerAccess	Simple Peer-to-Peer Transfers with Multi-GPU
cudaDeviceEnablePeerAccess	Simple Peer-to-Peer Transfers with Multi-GPU
cudaDeviceSynchronize	Bandwidth Test, Template, Template using CUDA Runtime
cudaDriverGetVersion	Device Query

CUDA Runtime API	Samples
cudaEventCreate	Bandwidth Test, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Simple Multi Copy and Compute, Simple Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy
cudaEventCreateWithFlags	Simple Peer-to-Peer Transfers with Multi-GPU
cudaEventDestroy	Bandwidth Test, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Simple Multi Copy and Compute, Simple Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy
cudaEventElapsedTime	Bandwidth Test, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Simple Multi Copy and Compute, Simple Multi-GPU, Simple Peer-to-Peer Transfers with Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy
cudaEventQuery	Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Simple Multi Copy and Compute, Simple Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy
cudaEventRecord	Bandwidth Test, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Simple Multi Copy and Compute, Simple Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy
cudaEventSynchronize	Matrix Multiplication (CUDA Runtime API Version), Vector Addition
cudaFree	Bandwidth Test, C++ Integration, Clock, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Pitch Linear Texture, Simple Atomic Intrinsics, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture, Simple Vote Intrinsics, Template, Template using CUDA Runtime, Using Inline PTX, Vector Addition, cudaOpenMP, simpleAssert, simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism), simpleMPI
cudaFreeArray	Pitch Linear Texture, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture

CUDA Runtime API	Samples
cudaFreeHost	Bandwidth Test, Simple Atomic Intrinsics, Simple Vote Intrinsics, Template using CUDA Runtime, Using Inline PTX, simpleAssert, simpleIPC, simpleZeroCopy
cudaFuncGetAttributes	cppOverload
cudaFuncSetCacheConfig	cppOverload
cudaGLSetGLDevice	Bicubic B-spline Interoplation, Bilateral Filter, Bindless Texture, Box Filter, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA and OpenGL Interop of Images, Fluids (OpenGL Version), Mandelbrot, Marching Cubes Isosurfaces, Particles, Post-Process in OpenGL, Recursive Gaussian Filter, Simple OpenGL, Simple Texture 3D, Smoke Particles, Sobel Filter, Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
cudaGetDeviceCount	Device Query
cudaGetDeviceProperties	Device Query
cudaGraphicsD3D10RegisterResource	SLI D3D10 Texture, Simple D3D10 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target
cudaGraphicsD3D11RegisterResource	Simple D3D11 Texture
cudaGraphicsD3D9RegisterResource	Simple D3D9 Texture, Simple Direct3D9 (Vertex Arrays)
cudaGraphicsGLRegisterBuffer	Bicubic B-spline Interoplation, Bilateral Filter, Bindless Texture, Box Filter, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA and OpenGL Interop of Images, Fluids (Direct3D Version), Fluids (OpenGL Version), Mandelbrot, Marching Cubes Isosurfaces, Particles, Post-Process in OpenGL, Recursive Gaussian Filter, Simple OpenGL, Simple Texture 3D, Smoke Particles, Sobel Filter, VFlockingD3D10, Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
cudaGraphicsMapResources	Bicubic B-spline Interoplation, Bilateral Filter, Bindless Texture, Box Filter, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA and OpenGL Interop of Images, Fluids (Direct3D Version), Fluids (OpenGL Version), Mandelbrot, Marching Cubes Isosurfaces, Particles, Post-Process in OpenGL, Recursive Gaussian Filter, Simple OpenGL, Simple Texture 3D, Smoke Particles, Sobel Filter, VFlockingD3D10,

CUDA Runtime API	Samples
	Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
cudaGraphicsRegisterResource	Bicubic B-spline Interoplation, Bilateral Filter, Bindless Texture, Box Filter, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA and OpenGL Interop of Images, Fluids (Direct3D Version), Fluids (OpenGL Version), Mandelbrot, Marching Cubes Isosurfaces, Particles, Post-Process in OpenGL, Recursive Gaussian Filter, Simple OpenGL, Simple Texture 3D, Smoke Particles, Sobel Filter, VFlockingD3D10, Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
cudaGraphicsResourceGetMappedPointer	Bicubic B-spline Interoplation, Bilateral Filter, Bindless Texture, Box Filter, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA and OpenGL Interop of Images, Fluids (Direct3D Version), Fluids (OpenGL Version), Mandelbrot, Marching Cubes Isosurfaces, Particles, Post-Process in OpenGL, Recursive Gaussian Filter, Simple OpenGL, Simple Texture 3D, Smoke Particles, Sobel Filter, VFlockingD3D10, Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
cudaGraphicsResourceSetMapFlags	SLI D3D10 Texture, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target
cudaGraphicsSubResourceGetMappedArray	SLI D3D10 Texture, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target
cudaGraphicsUnmapResources	Bicubic B-spline Interoplation, Bilateral Filter, Bindless Texture, Box Filter, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA and OpenGL Interop of Images, Fluids (Direct3D Version), Fluids (OpenGL Version), Mandelbrot, Marching Cubes Isosurfaces, Particles, Post-Process in OpenGL, Recursive Gaussian Filter, Simple OpenGL, Simple Texture 3D, Smoke Particles, Sobel Filter, VFlockingD3D10, Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
cudaGraphicsUnregisterResource	Bicubic B-spline Interoplation, Bilateral Filter, Bindless Texture, Box Filter, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA and OpenGL Interop of Images, Fluids (Direct3D Version), Fluids (OpenGL Version), Mandelbrot,

CUDA Runtime API	Samples
	Marching Cubes Isosurfaces, Particles, Post-Process in OpenGL, Recursive Gaussian Filter, SLI D3D10 Texture, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target, Simple Direct3D9 (Vertex Arrays), Simple OpenGL, Simple Texture 3D, Smoke Particles, Sobel Filter, VFlockingD3D10, Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes
cudaHostAlloc	Bandwidth Test, simpleZeroCopy
cudaHostGetDevicePointer	simpleZeroCopy
cudaHostRegister	simpleZeroCopy
cudaHostUnregister	simpleZeroCopy
cudaIpcCloseMemHandle	simpleIPC
cudaIpcGetEventHandlet	simpleIPC
cudaIpcOpenMemHandle	simpleIPC
cudaMallco	Simple Atomic Intrinsics, Simple Vote Intrinsics, simpleMPI
cudaMalloc	C++ Integration, Clock, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Pitch Linear Texture, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture, Template, Template using CUDA Runtime, Using Inline PTX, Vector Addition, cudaOpenMP, simpleAssert, simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)
cudaMalloc3DArray	Simple Cubemap Texture, Simple Layered Texture
cudaMallocArray	Pitch Linear Texture, Simple Surface Write, Simple Texture
cudaMallocHost	Bandwidth Test, Template using CUDA Runtime, Using Inline PTX, simpleAssert
cudaMallocPitch	Pitch Linear Texture
cudaMemcpy	Bandwidth Test, C++ Integration, Clock, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Simple Atomic Intrinsics, Simple Cubemap Texture, Simple Layered Texture, Simple Peer-to-Peer Transfers with Multi-GPU, Simple Surface Write, Simple Texture, Simple Vote

CUDA Runtime API	Samples
	Intrinsics, Template, Template using CUDA Runtime, Using Inline PTX, Vector Addition, cudaOpenMP, simpleAssert, simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism), simpleIPC, simpleMPI
cudaMemcpy2D	Pitch Linear Texture
cudaMemcpy2DToArray	SLI D3D10 Texture, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target
cudaMemcpy3D	Simple Cubemap Texture, Simple D3D9 Texture, Simple Layered Texture
cudaMemcpyAsync	Bandwidth Test, Simple CUDA Callbacks, Simple Multi Copy and Compute, Simple Multi-GPU, asyncAPI, simpleStreams
cudaMemcpyToArray	Pitch Linear Texture, Simple Texture
cudaMemset2D	Pitch Linear Texture
cudaPrintfDisplay	simplePrintf
cudaPrintfEnd	simplePrintf
cudaRuntimeGetVersion	Device Query
cudaSetDevice	Bandwidth Test, Device Query
cudaStreamAddCallback	Simple CUDA Callbacks
cudaStreamCreate	Simple CUDA Callbacks
cudaStreamDestroy	Simple CUDA Callbacks
cudaUnbindTexture	Pitch Linear Texture
cufftDestroy	CUDA FFT Ocean Simulation, FFT-Based 2D Convolution
cufftExecC2R	CUDA FFT Ocean Simulation, FFT-Based 2D Convolution
cufftExecR2C	CUDA FFT Ocean Simulation, FFT-Based 2D Convolution
cufftPlan2d	CUDA FFT Ocean Simulation, FFT-Based 2D Convolution
cuvidCreateDecoder	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuvidCtxLock	CUDA Video Encode (C Library) API
cuvidCtxLockCreate	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API

CUDA Runtime API	Samples
cuvidCtxLockDestroy	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuvidCtxUnlock	CUDA Video Encode (C Library) API
cuvidDecodePicture	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuvidDestroyDecoder	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuvidMapVideoFrame	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API
cuvidUnmapVideoFrame	CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API

Chapter 7. FREQUENTLY ASKED QUESTIONS

The Official CUDA FAQ is available online on the NVIDIA CUDA Forums:

http://forums.nvidia.com/index.php?showtopic=84440

Please also see the CUDA Toolkit Release Notes for additional Frequently Asked Questions.